Automating Industrial Event Stream Analytics: Methods, Models, and Tools

Zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften
(Dr.-Ing.)

von der KIT-Fakultät für Wirtschaftswissenschaften
des Karlsruher Instituts für Technologie (KIT)

genehmigte
DISSERTATION

von

M. Sc. Philipp Zehnder

Tag der mündlichen Prüfung:	23. September 2021
Referent:	Prof. Dr. York Sure-Vetter
Korreferent:	Jun.-Prof. Dr. Anne Meyer

Karlsruhe 2021

Abstract

Industrial event streams are an important cornerstone of Industrial Internet of Things (IIoT) applications. For instance, in the manufacturing domain, such streams are typically produced by distributed industrial assets at high frequency on the shop floor. To add business value and extract the full potential of the data (e.g. through predictive quality assessment or maintenance), industrial event stream analytics is an essential building block. One major challenge is the distribution of required technical and domain knowledge across several roles, which makes the realization of analytics projects time-consuming and error-prone. For instance, accessing industrial data sources requires a high level of technical skills due to a large heterogeneity of protocols and formats. To reduce the technical overhead of current approaches, several problems must be addressed. The goal is to enable so-called "citizen technologists" to evaluate event streams through a self-service approach. This requires new methods and models that cover the entire data analytics cycle.

In this thesis, the research question is answered, how citizen technologists can be facilitated to independently perform industrial event stream analytics. The first step is to investigate how the technical complexity of modeling and connecting industrial data sources can be reduced. Subsequently, it is analyzed how the event streams can be automatically adapted (directly at the edge), to meet the requirements of data consumers and the infrastructure. Finally, this thesis examines how machine learning models for industrial event streams can be trained in an automated way to evaluate previously integrated data. The main research contributions of this work are:

1. A semantics-based adapter model to describe industrial data sources and to automatically generate adapter instances on edge nodes.
2. An extension for publish-/subscribe systems that dynamically reduces event streams while considering requirements of downstream algorithms.
3. A novel AutoML approach to enable citizen data scientists to train and deploy supervised ML models for industrial event streams.

The developed approaches are fully implemented in various high-quality software artifacts. These have been integrated into a large open-source project, which enables rapid adoption of the novel concepts into real-world environments. For the evaluation, two user studies to investigate the usability, as well as performance and accuracy tests of the individual components were performed.

Contents

Figures

Tables

Listings

Abbreviations

AAS	Asset Administration Shell
AI	Artificial Intelligence
AutoML	Automated Machine Learning
BI	Business Intelligence
CEP	Complex Event Processing
CPS	Cyber Physical Systems
CRISP-DM	Cross-Industry Standard for Data Mining
DSL	Domain Specific Language
EC	Event Consumer
EP	Event Producer
EPA	Event Processing Agents
EPL	Event Processing Language
EPN	Event Processing Network
ERP	Enterprise Resource Planning
ETL	Extract, Transform, Load
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
ICS	Industrial Control Systems
IIoT	Industrial Internet of Things
IoT	Internet of Things
IT	Information Technology
JSON	JavaScript Object Notation
JSON-LD	JavaScript Object Notation for Linked Data
MES	Manufacturing Execution Systems
ML	Machine Learning

MLPA	Machine Learning Pipeline Agent
MQTT	Message Queuing Telemetry Transport
OPC UA	Open Platform Communication Unified Architecture
OT	Operational Technology
PLC	Programmable Logic Controller
RDF	Resource Description Framework
ROS	Robot Operating System
SCADA	Supervisory Control and Data Acquisition
SDK	Software Development Kit
SQL	Structured Query Language
STCS	Solar Thermal Climate System
SUS	System Usability Scale
TRI	Technology Readiness Index
UEQ	User Experience Questionnaire
umati	Universal machine technology interface
UML	Unified Modeling Language
URL	Uniform Resource Locator
XML	Extensible Markup Language

1 Introduction

The Industrial Internet of Things (IIoT) is an enabler for data-driven analytics as more and more machines within industrial production facilities are able to communicate with each other. Based on this large amount of data, generated by machines and their integrated sensors, there is the potential to create added value for companies. This added value can differ greatly among a variety of use cases, e.g. improvement of production processes and product quality [Rehman et al. 2019], increased autonomy of machines [Maier et al. 2017], or increase of productivity and efficiency [Xu et al. 2018]. The produced data can be very complex, of high volume, and high velocity, requiring methods from the field of Artificial Intelligence (AI) in order to evaluate them automatically. However, to be able to use these methods, a lot of domain knowledge as well as technical knowledge is needed e.g. to connect data sources, process data, or to train Machine Learning (ML) models. The domain knowledge is necessary to understand the production process as well as the meaning of data. People with domain knowledge typically lack the technical and data science expertise necessary to build an environment for analytics. Therefore, *technical specialists* are often needed even though they are not experts in the domain. This means either that domain experts have to be trained, or they must work closely with technical specialists. In the past, it has been shown that advances in Information Technology (IT) often make it possible that tools and techniques that previously were addressed only for technical specialists, are additionally available for less technical users. One such example are Online Application Generators (OAG), which facilitate the creation of applications with little technical knowledge [Oltrogge et al. 2018]. In this work we try to bridge the gap between technical specialists and *business domain specialists* in the field of industrial analytics.

Business domain specialists usually have a comprehensive knowledge about the domain and the manufacturing processes, but typically only have basic computer science or data science knowledge [Gröger 2018]. However, since they already understand the meaning of the data, they should be enabled to evaluate it. In this work, we refer to *citizen technologists*, which "are able to do things that previously required discipline experts"[1]. These are business domain specialists with special technical training. The goal of this thesis is to reduce the required technical knowledge and therefore to facilitate citizen

[1]https://chiefmartec.com/2018/05/democratizing-martech-marketing-technologists/ (accessed on 04/10/2021)

technologists to use advanced analytics by introducing methods, models and tools that enable the automation of *industrial event stream analytics*. To make this possible, three phases must be considered: data must first be *ingested* from a data source, then *integrated* from multiple data sources, before it can be *analyzed*.

The process of fetching raw data from data sources and providing it in a format that can be used for further processing is called data ingestion [Hueske and Kalavri 2019]. This is especially challenging in IIoT scenarios, due to the high heterogeneity of sensors and machine interfaces. Therefore, we try to enable *citizen developers*[2] to connect such sources. Citizen developers are citizen technologists, who have a basic understanding of the commonly used communication protocols and formats in industrial environments. They do not need to know how to implement software to connect machines, but they must be able to provide the necessary configurations of the various protocols. In industrial environments a variety of different systems exist that produce data which might be relevant. Such systems can range from Enterprise Resource Planning (ERP), Manufacturing Execution Systems (MES), Supervisory Control and Data Acquisition (SCADA) systems, sensors, to Programmable Logic Controllers (PLCs) controlling machines [Rehman et al. 2019]. In this work we focus on data primarily produced by sensors and machines. A major issue is to deal with the various available communication protocols and data representations that machines from different vendors or domains have. There is no standard protocol or principle of communication that covers them all [Gosewehr et al. 2017]. Therefore, a flexible solution is needed that can handle all these different sources and harmonize the data to make it processable. More importantly, it needs to describe its semantics in a machine processable way. Additionally to the already described challenges of heterogeneity in data sources, the machines and sensors often do not provide the necessary infrastructure (e.g. computation, storage) to process the data. The reason for this is that they were designed to automate production processes, consequently data should be stored and processed in separate systems [Trunzer et al. 2019]. Usually adapters are used to read data from the data sources, however, the creation of those is often hard to automate.

After a connection is established, data generated by the different data sources must be aggregated and integrated before it can be analyzed [Trunzer et al. 2017]. As data is produced constantly by sensors and machines, it is referred to as *event streams*, a set of associated events that are usually temporarily ordered [Etzion and Niblett 2010]. In this work, we refer to event streams in the context of IIoT as *industrial event streams* because of its special characteristics, which are explained in this work in more detail. To cope with those characteristics (e.g. potentially high volume, high frequency), the transmission must be optimized and the size of data to be transmitted should be minimized to reduce the network load [Xu et al. 2018]. This especially means that data must not only be sent to a central instance, furthermore it should be processed in close proximity to the source to harmonize it before it is transmitted. Techniques from the field of *edge computing* can

[2]https://www.gartner.com/en/information-technology/glossary/citizen-developer (accessed on 04/10/2021)

be used, where it is possible to collect and process data based on its geographic location without transmitting it to the cloud [Shi et al. 2016]. These local compute capabilities can be leveraged to intelligently route the data and thus reduce the overall load on the network.

Due to the ever-increasing amount of data and high dimensionality, it can become quite complex to apply classic methods such as rule-based approaches to analyze data. Learning-based techniques from the field of analytics and ML can be used to train models once enough "good quality" data is available. To train such models, usually *data scientists* are required. They are capable of getting answers to important business questions, by analyzing large amounts of data [Davenport and Patil 2012]. To understand the business problems as well as the problems of the domain, data scientists must work closely together with domain specialists, which are able to explain the meaning of the data (e.g. effects on the production process). In this work, we take a slightly different approach, since data science knowledge is still rare, especially in manufacturing companies. Our aim is to enable *citizen data scientists* to perform such tasks. These are business domain specialists that know how to generate models using techniques from the field of advanced analytics even if their primary job is outside of the field of statistics and analytics[3]. Their main expertise is not technically-focused, rather they have the necessary domain expertise as well as a basic understanding of data analytics to be able to assess how well the generated models perform. Citizen data scientists are needed because the meaning of industry data is highly dependent on the domain and the specific application. The training process should be automated as much as possible to reduce the technical complexity of selecting the parameters and ML models. Recent developments in the field of self-service data analytics and Automated Machine Learning (AutoML) reduce the complexity for the training process and make it available for less technical users. However, the goal of this thesis is not to fully automate the process, but to keep the human in the loop. Citizen technologists should be supported by introducing methods, models, and tools to perform the analytics task themselves. As an important requirement this should be done in a user-friendly way and data visualizations should be used to ease the interaction with the data [Han et al. 2011; Michalczyk et al. 2020]. To ensure this, we evaluated our approach in two different user studies, showing that even less technical users are able to perform those tasks.

1.1. Research Questions

The main goal of this thesis is to enable citizen technologists to connect and analyze industrial event streams. Therefore, data of machines and sensors must first be connected and ingested, then the data of the different data sources must be integrated into a

[3]https://blogs.gartner.com/carlie-idoine/2018/05/13/citizen-data-scientists-and-why-they-matter/ (accessed on 04/10/2021)

harmonized data foundation. Based on this integration, it is possible to combine data from multiple sources and analyze it to create new insights. All this has to be feasible for *business domain specialists*. Both the technical requirements to work with industry data and the analytics of this data are very complex and requires specialized knowledge. Consequently, we introduce *citizen technologists*, which are business domain specialists with additional technical training, who are not necessarily able to implement solutions in programming languages, but are able to solve technical tasks with the appropriate tooling. Furthermore, we distinguish between *citizen developers* focusing on connecting machines and *citizen data scientists* focusing on the analytics tasks. Both of them do not have a complete computer science or data science education, but have a basic understanding of the field and are able to accomplish technical tasks with the appropriate methods, models, and tools. Hence, many challenges remain to be addressed, like the provisioning of a model to describe the high heterogeneity of different data sources, or the integration of different data sources in a distributed environment. To analyze data, a lot of domain knowledge as well as data science knowledge is required to have both the skills to process the data, as well as to understand its meaning. The demand for all these different skills leads to the fact that IIoT data science projects involve many different roles from different areas (e.g. domain expert, data scientist, IT administrator, ...). This results in a high communication overhead, which makes these projects quite time consuming. Our goal is to enable citizen technologists to evaluate data and to carry out such projects themselves. This leads us to the following research question:

How to facilitate data-driven analytics of industrial event streams by citizen technologists?

In the main research question, the term *industrial event streams* is used. Industrial event streams can be described by the following characteristics. They have a high frequency, individual event streams are homogeneous, and multiple streams can be produced by geo-distributed assets using heterogeneous event types (e.g. sensor measurements, or images). Due to those characteristics, it is often required to leverage edge processing techniques to process data in close proximity to its source. This thesis tries to facilitate data-driven analytics, enabling citizen technologists to leverage machine data by connecting it themselves. Based on this data, learning techniques should be applied to train ML models that are deployed on real-time event streams to generate new business relevant insights (e.g. reduce scrap rate). To overcome the limitations of rule-based analytics, techniques from data driven analytics can be leveraged. Instead of defining a rule, a model is trained on historic data to detect certain situations directly on event streams. The main research question is further divided into three sub-research questions.

Research Question 1 (Ingest)**.** *How can we support citizen developers in modeling and connecting heterogeneous industrial event streams?*

The first research question targets the data ingestion by connecting new industrial data sources. The question is, how citizen developers with basic technical understanding are able to extract data from such sources. For example, they are not able to implement data adapters in a programming language, but have the knowledge about the concepts of industrial protocols. The main challenge is to deal with the technical complexity of different protocols, formats, and standards that are used in industry settings and provide a solution that can support old legacy machines and is extensible to support future standards. Additionally, the characteristics of industrial event streams have to be handled. Since the data quality of various sources can be quite different, a possibility for harmonizing it directly upon ingestion time is required. This research question is answered in Chapter 6 (and evaluated in Chapter 9).

Research Question 2 (Integrate)**.** *Based on semantic event models, how can we automatically adapt industrial event streams to meet the demands of infrastructure and subscribers?*

The second research question focuses on integrating data from different industrial event streams and adapt the transmission of the data based on the requirements of the algorithms processing the data. The previously ingested data must be integrated to form a foundation for the analytics tasks, with the goal of reducing the impact on the infrastructure. This should be done without manual effort, by leveraging the semantic data model (a result of research question one). The technical complexity of distributed data processing should be abstracted. A challenge here is to optimize the data flow in a geo-distributed environment while satisfying the different requirements of individual subscribers. Therefore, a distributed architecture is required that is capable to support those functionalities, especially in dynamic changing environments. Based on the structure of the event streams and the performed analytics pipelines, the stream might be adapted while ensuring the correct performance of the analytics task. All changes on the data should happen within the transportation layer, without affecting the analytics algorithms evaluating the event stream. This research question is answered in Chapter 7 (and evaluated in Chapter 9).

Research Question 3 (Analyze)**.** *How can citizen data scientists be enabled to use automated machine learning to analyze industrial event streams themselves?*

This research question addresses the automatic analytics of industrial event streams to gain new insights from the integrated data. How can supervised ML tasks be defined by citizen data scientists and how can this model be trained automatically on industrial event streams? The main challenge is to automate the data science process to create new ML models. Furthermore, trained models must be deployed in a distributed streaming environment and applied on industrial event streams, where the data distribution might change over time. This research question is answered in Chapter 8 (and evaluated in Chapter 9).

1.2. Research Methodology

In this work, we use the research methodology of Design Science. This paradigm "seeks to extend the boundaries of human and organizational capabilities by creating new and innovative artifacts" [Hevner et al. 2004]. The aim is to solve human or organizational problems with the help of IT artifacts. These artifacts can take different forms, ranging "from software, formal logic, and rigorous mathematics to informal natural language descriptions" [Hevner et al. 2004]. To perform effective design-science research, seven guidelines are introduced: ***Design as an Artifact (G1)***, ***Problem Relevance (G2)***, ***Design Evaluation (G3)***, ***Research Contributions (G4)***, ***Research Rigor (G5)***, ***Design as a Search Process (G6)***, and ***Communication of Research (G7)***. All these guidelines are considered in this thesis and explained in the following.

The main parts explain how citizen technologists are enabled to perform industrial event stream analytics and are presented in Chapter 6 (Ingest), Chapter 7 (Integrate), and Chapter 8 (Analyze). Software artifacts were developed based on the novel concepts presented in this thesis. Most of them were integrated into the open source software project Apache StreamPipes (incubating)[4] ***(G1)***. StreamPipes was transferred to the Apache Software Foundation and is part of the Apache Incubator Program at the time of this writing. The relevance of the problem ***(G2)*** is motivated in Chapter 3, by showing the need to reduce the technical complexity for industrial event stream analytics. ***(G3)*** Both performance and usability were considered in the evaluation. Different evaluation methods were chosen and performed for the different contributions of this thesis. All results of the evaluation are presented in Chapter 9. The main contribution of this work ***(G4)***, is to enable citizen technologists to apply advanced analytics themselves on industrial event streams. In order to follow the guidelines ***(G5)*** and ***(G6)*** , the foundations are presented in Chapter 2 and the related work is discussed in detail in Chapter 4. Early feedback from open source users, as well as from participants of the conducted user studies, were taken into account to improve and validate the developed approaches. To reach a wide audience from the research, technology, and business domain, the results of this research were published and discussed at research conferences, as well as on international fairs and technology conferences ***(G7)***.

1.3. Contributions

In this thesis, we enable citizen technologists to ingest, integrate, and analyze industrial event streams. To accomplish this, we provide several contributions, which are outlined in this section.

[4]https://streampipes.apache.org (accessed on 04/10/2021)

- **(C1) Adapter Model** With this model, it is possible to describe data sources and automatically generate adapter instances to connect to those sources based on a semantically-enriched description. This model is extensible and supports both standard protocols and formats as well as custom and legacy interfaces. As data can come from many heterogeneous sources, an adapter must be able to harmonize data. Therefore, we introduce transformation rules, that can be used to preprocess and clean data on ingestion. To meet the potentially high frequency requirements of industrial event streams the adapters can be deployed on edge nodes. On those they can access and preprocess the data in close proximity to the source, before the harmonized data is forwarded onto a distributed message broker to be analyzed by upstream services.

- **(C2) Intelligent data reduction** A distributed master- / worker-based approach is used for the architecture. This allows workers to handle most of the data load in close proximity to the source, and leaves the master in charge of communicating and managing the entire system. To be capable to process the potentially large event streams, event reduction strategies are proposed, reducing the event size and transmission frequency. A concept for a message broker, automatically adapting the industrial event streams according to the requirements of the subscribers, is developed as a wrapper around existing topic-based message brokers . The terms virtual and partial events are introduced, where virtual events are reconstructed at the subscriber without transmitting all information. Partial events contain only parts from the original event and are used as a basis for the reconstruction of virtual events.

- **(C3) Enable ML for citizen data scientists** We provide a holistic approach with focus on citizen data scientists, to explore and train ML models specially designed for industrial event streams. It is possible to train and deploy supervised ML models based on previously integrated data. Therefore, three types of Machine Learning Pipeline Agents (MLPAs) are provided that can be applied on event streams, directly after training. A standardized workflow of defining the ML problem is introduced and the model is then automatically trained on data from industrial event streams. Based on the results of this training, citizen data scientists can select the best model for their use case and directly deploy it on event streams.

- **(C4) Software Prototype** Contributions **C1 - C3** are realized in software prototypes. Most of them are integrated into the Apache StreamPipes project and are made available as open source software to the community. The software is designed to cover the whole lifecycle, starting with connecting different machines over a graphical user interface. The connected event streams leverage the newly developed message broker to intelligently reduce the transmitted data. Furthermore, a data explorer was developed to get insights into the data by visually analyzing it, as well as manually adding labels for supervised learning tasks. A guided workflow is introduced to define the ML problem based on the labeled data. The ML model is trained automatically in the background, then stored in a model repository, from where it can be deployed to process event streams.

1.4. Research Projects and Publications

The concepts, methods and implemented prototypes of this work were developed in two projects funded by the German BMVI (Federal Ministry of Transport and Digital Infrastructure) as well as further projects and collaborations. Conceptual contributions are peer-reviewed and were presented at various research conferences as outlined below. In addition, the results were showcased at international exhibitions and technical developer conferences. In the following, the projects and publications are listed.

WEKOVI (Tools for the simple creation of complex comparison indicators, 07/2017-07/2019, BMVI)

In WEKOVI, a user-friendly, extendable software platform was developed that enables users to use open data sources (e.g. environment sensors) and calculate complex comparison indicators, such as an air quality index. Those indicators can be used to calculate a rating and compare different locations with each other. In WEKOVI, the architecture for integrating heterogeneous data sources as well as the data model for connecting new data sources and automatically deploying adapters was developed. For processing those sources, the concepts for data reduction in a distributed broker-based architecture were developed.

Publications

- Philipp Zehnder, Dominik Riemer. **Modeling self-service machine-learning agents for distributed stream processing.** Proceedings of the IEEE International Conference on Big Data (BigData). 2017, Boston, MA, USA. (see [Zehnder and Riemer 2017]).
- Philipp Zehnder, Dominik Riemer. **Representing Industrial Data Streams in Digital Twins using Semantic Labeling.** Proceedings of the IEEE International Conference on Big Data (BigData). 2018, Seattle, WA, USA. (see [Zehnder and Riemer 2018]).

OCROSS (Open Data Crowd Sensing Service for the easy fusion of annotated and swarm-based mass data, 10/2018 - 05/2021, BMVI)

In OCROSS, the aim was to record the condition of roads using video, image, and sensor data. A smartphone app is used to collect mass data from the crowd. This data has to be anonymized, combined with other data sources and analyzed to detect the conditions of the road. In addition to monitoring the road condition, image and sensor data was used to check, whether road construction sites have been set up correctly or not. For this purpose, traffic signs are recognized in the images and their position is compared with a control plan. In OCROSS, the methods and concepts for automated data analytics was developed (AutoML). It was examined how it is possible to train ML models in a way that is feasible for citizen data scientists and deployed them on live event streams.

Publications

- Philipp Zehnder, Patrick Wiener, Dominik Riemer. **Using Virtual Events for Edge-based Data Stream Reduction in Distributed Publish/Subscribe Systems.** Proceedings of the 3rd IEEE International Conference on Fog and Edge Computing (ICFEC). 2019, Larnaca, Cyprus. (see [Zehnder et al. 2019]).
- Philipp Zehnder, Patrick Wiener, Tim Straub, Dominik Riemer. **StreamPipes Connect: Semantics-Based Edge Adapters for the IIoT.** Proceedings of the 17th International Conference - The Semantic Web (ESWC). 2020, Heraklion, Crete, Greece. (see [Zehnder et al. 2020]).

Projects & Cooperations

Additionally to the previously mentioned projects, the requirements for the developed concepts were collected and tested in further projects and cooperations. This gave us the chance to validate the approaches in different real-world scenarios as well as with data from different domains. The following list shows the various projects, research and industry collaborations.

- **Test Field Autonomous Driving Baden-Württemberg** is an area to test drive autonomous vehicles in normal road traffic. In this project, the developed concepts and prototypes are used as part of the backend for ingesting and integrating data from multiple sensors and services.
- **HoLL-Therm** The thermal storage capacity of buildings should be used to control energy consumption, both in a grid-serving and efficient manner with the aid of sensors and actuators. Therefore, different kinds of data sources were integrated, in order to evaluate the performance of the event stream adaption and reduction.
- **Industry & Research Cooperations** In addition to the previously mentioned research projects, the concepts and prototypes were tested in industry projects as well as further developed in research cooperations, where we collected valuable feedback. Also we had the chance to test the prototypes in different machines and production facilities, which helped to optimize the concepts.

1.5. Guide to the Reader

This section presents the structure of this thesis and gives an overview of the individual chapters, which is depicted in Figure 1.1. Chapter 1 provides an introduction into the topic of industrial event stream analytics and why it is important to enable people with little technical knowledge to perform this task. Chapter 2 highlights the foundations in the areas of industrial systems, event streams and how to process them, as well as automated analytics. Chapter 3 first provides a deep dive into the subject of industrial event stream analytics. Second a motivating example is illustrated, which is used throughout this work to introduce the models and concepts. Finally, the different problems that are addressed in this work are depicted and a requirements elicitation is performed. The related work is introduced in Chapter 4 and structured as follows: First, research approaches in the field of integrated IIoT systems are introduced, then related work in the context of connecting and modeling IIoT data sources is presented. Further, relevant approaches for data processing and self-service data analytics are introduced. Chapter 5 provides an overview of the three main parts of this work. Chapter 6 shows how data can be ingested from heterogeneous industrial data sources. It begins by introducing a model that can be used to describe the individual sources. Based on this model, adapters can automatically be instantiated on edge devices. A particular focus of this chapter is how to enable citizen developers to perform this task themselves. Chapter 7 deals with the integration of different data sources and event reduction strategies, to dynamically reduce the amount of transferred data. Chapter 8, the last of the three main chapters, shows how previously connected data can be analyzed by citizen data scientists, who are able to train supervised ML models with the introduced approach. Chapter 9 reports the evaluation of the main contributions of this thesis. Both usability and performance of the system are evaluated. In the last chapter, the work is concluded and potential future research directions are presented.

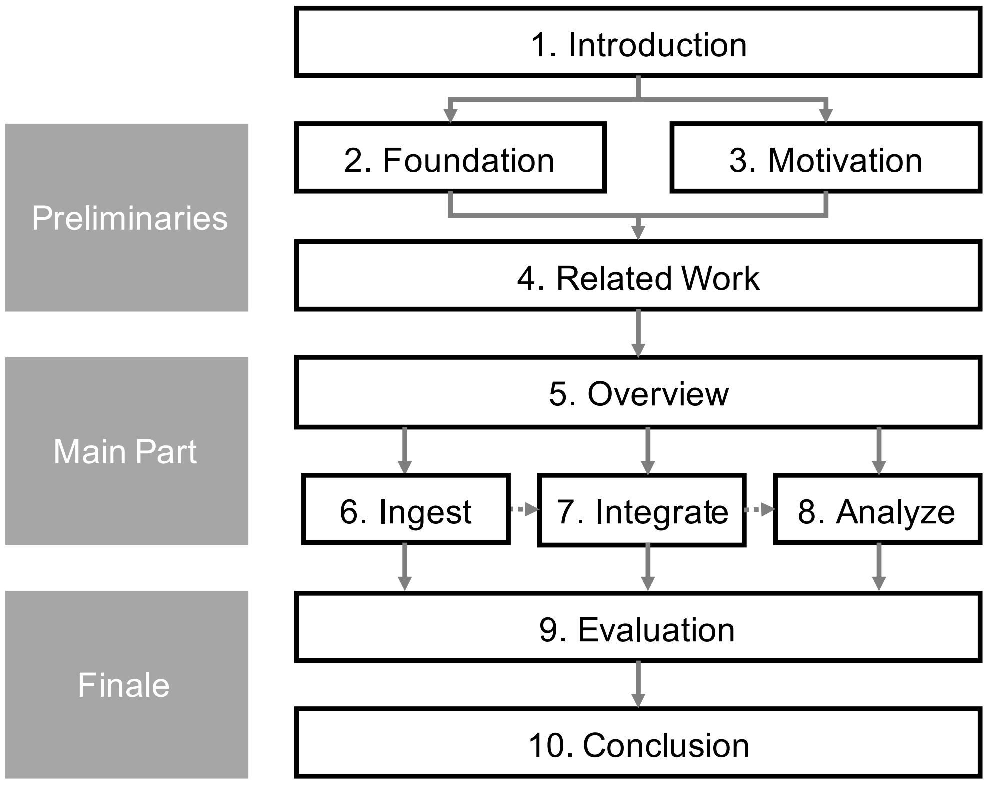


Figure 1.1.: Structure of this thesis

Part I.

Preliminaries

2 Foundations

In this chapter, the foundations for the rest of this thesis are presented by introducing definitions and concepts from the field of *automated industrial event stream analytics*. The chapter is structured according to the three areas of the terms: First, Section 2.1 deals with the term *industrial data management*, then Section 2.2 introduces *event streams*, and finally, Section 2.3 is about *(automated) analytics*. The following section describes different foundations in the field of industrial data management, focusing on the Industrial Internet of Things (IIoT). The concepts and technologies behind this term are introduced and placed in the context of industry 4.0 and cyber-physical systems. After that, different concepts and terms related to event streaming are introduced, which are relevant because the data in this thesis is processed according to these principles. The chapter concludes with different data processing methodologies (cloud and edge computing). For data analytics, mainly the differences between rule-based and learning-based approaches are explained. Furthermore, terms from the area of Machine Learning (ML) are introduced and how the training process of a ML model can be further automated.

2.1. Industrial Data Management

In industry settings there are two main technologies that are the driver for analytics on industrial data. Operational Technology (OT) is already used for a long time to automate production processes, whereas Information Technology (IT) is an enabler for exchanging and analyzing data. These two areas are merging more and more. In the first section, terms like Industrial Internet of Things and Industry 4.0 are introduced, which were originally driven by computer scientists [Jeschke et al. 2017]. Then, Industrial Control Systems are discussed, which are responsible for controlling processes of plants and machines. These systems are getting smarter and more connected. The individual areas are increasingly merging, which leads to a greater intertwining between OT and IT.

2.1.1. Industrial Internet of Things (IIoT)

The term Internet of Things (IoT) "generally refers to scenarios where network connectivity and computing capability extends to objects, sensors and everyday items not normally considered computers, allowing these devices to generate, exchange and consume data with minimal human intervention" [Rose et al. 2015]. This means that more and more things obtain computing and networking capabilities, which enables them to communicate with each other independently of humans. In industrial applications, these things are often sensors and machines on shop floors. The combination of industrial applications and IoT is a different world, both in meaning and concepts, so a deep understanding of the term industrial is necessary [Xu et al. 2018]. The potential of the industrial internet is seen particularly by the industry, as a report by GE and Accenture shows. It describes that the "Industrial Internet enables companies to use sensors, software, machine-to-machine learning and other technologies to gather and analyze data from physical objects or other large data streams and then use those analyses to manage operations and in some cases to offer new, value-added services" [GE 2015]. One of the key features of machine-to-machine interaction is the merging of physical und digital processes [Schilberg et al. 2017]. Those two worlds open the possibility for companies to break new ground. As a consequence the Industrial Internet of Things (IIoT) plays a key role. It is defined as: "A system comprising networked smart objects, cyber-physical assets, associated generic information technologies and optional cloud or edge computing platforms, which enable real-time, intelligent, and autonomous access, collection, analysis, communications, and exchange of process, product and/or service information, within the industrial environment, so as to optimize overall production value. This value may include; improving product or service delivery, boosting productivity, reducing labour costs, reducing energy consumption, and reducing the build to-order cycle" [Boyes et al. 2018]. A special focus in this work is on machine data, how it is exchanged, stored, and analyzed. In IIoT it is particular essential to store historical and real-time data and integrate it at different levels [Rehman et al. 2019]. Additionally, this data enables the incorporation of ML into all IIoT systems [Xu et al. 2018].

Another frequently used term is Industry 4.0. It represents the fourth industrial revolution, "in which information communication technologies are applied to industrial manufacturing and automation so that the productivity and efficiency can be improved" [Xu et al. 2018]. Both terms, IIoT and Industry 4.0, are often used synonymously [Rehman et al. 2019]. The term IIoT often describes the technology movement, whereas Industry 4.0 is often more associated with the economic impact [Jeschke et al. 2017]. In this work, we use those terms as synonymes, since industrial event streams play a key role in both of them. Both, IIoT and Industry 4.0, have a high complexity, which is why there is no single standard. This requires to combine different standards from multiple domains to cover all aspects [Schleipen et al. 2016]. As mentioned before, the view on physical systems in industrial settings is very important, so the notion of cyber physical systems will be explained in the next section.

2.1.2. Industrial Control Systems

The central role in IIoT play industrial machines. These machines are already equipped with a lot of different sensors and actuators. However, data of those sensors has been used mainly for direct feedback control in real-time, or for forensic use, when it was archived [Lade et al. 2017]. The main purpose was to ensure that the machines operate as expected. Such systems are often referred to as Industrial Control Systems (ICS). It "is a collective term typically used to describe different types of control systems and associated instrumentation, which include the devices, systems, networks, and controls used to operate and/or automate industrial processes" [Boyes et al. 2018]. Traditionally, ICS systems were independent of IT systems, often used in isolated settings running proprietary control protocols for specialized hardware [Boyes et al. 2018]. Programmable Logic Controllers (PLCs) are normally used to execute predefined, fixed logical programs implemented in a proprietary language. Those automation systems used to be closed loop control systems on the shop floor connected through a field bus protocol [Schilberg et al. 2017]. To connect several of them, Supervisory Control and Data Acquisition (SCADA) systems are often used to control entire plants and distributed systems. SCADA represents the largest subgroup of ICS [Boyes et al. 2018]. It is used for high-level management of processes and is connected to several other systems like PLCs, computers or networking devices. Standards like Open Platform Communication Unified Architecture (OPC UA) try to bride this gap between IT and OT, by making machine data available, providing a client-server architecture and a rich meta-data model. Communication within a factory used to be rather simple, *sensors*, *actors*, and *controllers* were only connected for a specific task [Neumann 2007]. *Sensors* measure target parameters and provide the unfiltered output to a controller. *Controllers* receive those measurements and store, process, or provide them to a function to calculate an output value, which is sent to actuators. The *actuators* act upon the received commands from the controller and are able to directly influence the physical environment [Rüth et al. 2017]. Nowadays, this data is used and combined with other sources into Cyber Physical Systems (CPS), which are defined as "a new generation of systems with integrated computational and physical capabilities that can interact with humans through many new modalities" [Baheti and Gill 2011]. The challenge addressed in this work, is to bridge the gap between the data sources within ICS systems by making this data easier accessible, and enable people with less technical knowledge to analyze the resulting event streams, which is explained in the following.

2.2. Event Streams

In this section, the basics of event streaming technologies are presented and the publish-/-subscribe paradigm is introduced. This is followed by the paradigm of edge computing, showing how data can be processed in close proximity to the data source.

2.2.1. Event Streaming

The most basic element for event streaming is an event that is a "record(s) of an activity in a System" [Luckham 2002]. In this work, we define events as: "An occurrence within a particular system or domain; it is something that has happened or is contemplated as having happened in that domain" [Etzion and Niblett 2010]. The structure of an event is described by an event schema, defining the event type. Events usually occur in an event stream, which is defined as a set of associated events, that are often totally temporally ordered [Etzion and Niblett 2010]. Such event streams can be evaluated with event processing that is defined as "computing that performs operations on events"[Etzion and Niblett 2010]. Common event processing operations include reading, creating, transforming, deleting events.

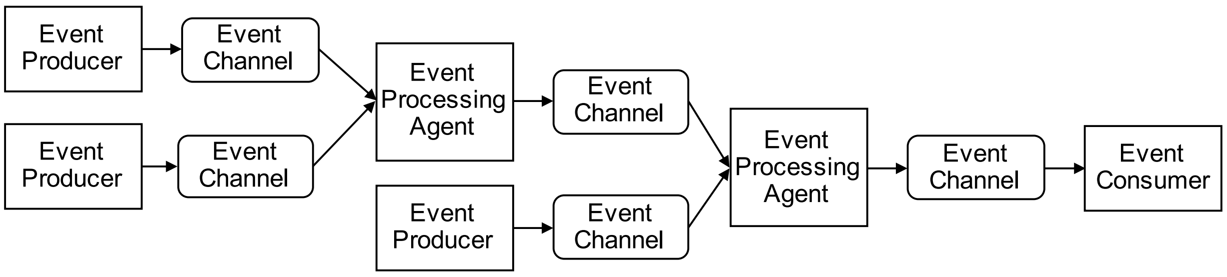


Figure 2.1.: Example Event Processing Network (EPN) consisting of Event Producer, Event Processing Agents, Event Channels, and Event Consumer

In event processing, events are processed in an event driven architecture, which is called an Event Processing Network (EPN). EPNs can be described as a graph where events are produced by Event Producers (EPs), processed by Event Processing Agents (EPAs) and consumed by Event Consumers (ECs) [Etzion and Niblett 2010]. To forward events between the individual components, event channels are used that establish a connection between these components. Figure 2.1 shows such a network with three different EPs, two EPAs performing operations on the events, and one EC. Several realizations for modeling EPNs exist, for example programmatically, via a Domain Specific Language (DSL), or graphically. A DSL is a computer programming language addressing a specific domain with limited expressiveness [Fowler 2010]. Graphical tools can also be used by by non-technical users. In [Riemer 2016] a graphical approach is presented that allows users to express their interest in certain situations or perform data harmonization by defining EPNs in the form of pipelines. Those pipelines are then deployed as distributed EPNs. In this work, the word pipeline is used as a synonyme to EPN.

2.2.2. Publish- / Subscribe

When dealing with large-scale distributed systems, the publish-/subscribe paradigm is often used to route data between services. In such systems, subscribers register their interest in events or event patterns and are then asynchronously notified, when a publisher emits a new event that match their registered interest [Eugster et al. 2003]. In this paradigm the individual components have a "full decoupling in time, space, and synchronization between publishers and subscribers" [Eugster et al. 2003]. This decoupling reduces the dependencies between the individual components. Furthermore, it reduces the complexity, because subscribers are usually not interested in every event from all publishers.

There are three main types of publish-/subscribe systems: topic-based, content-based, and type-based. In topic-based systems, events are published on a topic or subject, and subscribers can register and receive all events published to a certain topic or subject. In content-based systems, a subscriber can register to a specific content of events, and it is ensured that only events fulfilling those requirements are received. Events within each topic often have the same event schema. This led to the development of type-based publish-/subscribe systems, where subscribers do not subscribe to a topic, they rather can subscribe to events of a specific type [Eugster et al. 2003]. The decoupling of publishers and subscribers means in particular that they have no knowledge of each other. The publisher can act independently of potential dependent services that consume the data and reduce the complexity of the logic from the individual components. In this work, we focus on topic-based publish-/subscribe systems, because they are well established and are already used in the context of IIoT.

2.2.3. Edge Computing

So far in this chapter, we showed what events are and how they are processed in a streaming fashion, e.g. using a publish-/subscribe approach. We will now briefly introduce a processing paradigm that enables location-aware data processing in distributed IIoT scenarios. In recent years, there has been a strong trend to run applications and data processing in the cloud, where "cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services" [Armbrust et al. 2010]. Accordingly, there is a centralized compute infrastructure, which is easy to scale through virtualization, but does not leverage compute resources that are available near the data sources. This is important in IIoT scenarios, because those sources are usually highly distributed and require a low latency. Figure 2.2 shows the cloud computing paradigm on the left. Raw data is produced by data producers on the top left and sent to the cloud. On the bottom, some computing devices are illustrated. Their main purpose is to consume data from the cloud. This is mainly achieved via sending requests to the cloud and receiving the results. When the data sources are distributed, one possibility is to transmit all data to the cloud. However, this could lead to an overload of the network infrastructure. Additionally, the latency introduced in transmitting data might be too high for some analytics tasks [Xu et al. 2018]. Privacy policies cannot be adhered, because all the data must be shared and no local (pre-)processing takes place [Shi et al. 2016]. Transmitting all data might result in a high consumption of bandwidth, which could be reduced when the data is processed locally [Satyanarayanan 2017]. To achieve this, data can be processed close to the data source at the network edge [Shi and Dustdar 2016]. We define edge as "any computing and network resources along the path between data sources and cloud data centers" [Shi et al. 2016]. Edge computing is different from cloud computing, because it "tends to leverage the computing and storage capabilities from edge devices (e.g., computing edge nodes)" [Xu et al. 2018]. Both processing paradigms are not distinct, they often can be combined to leverage the individual advantages and find the best solution for the respective application.

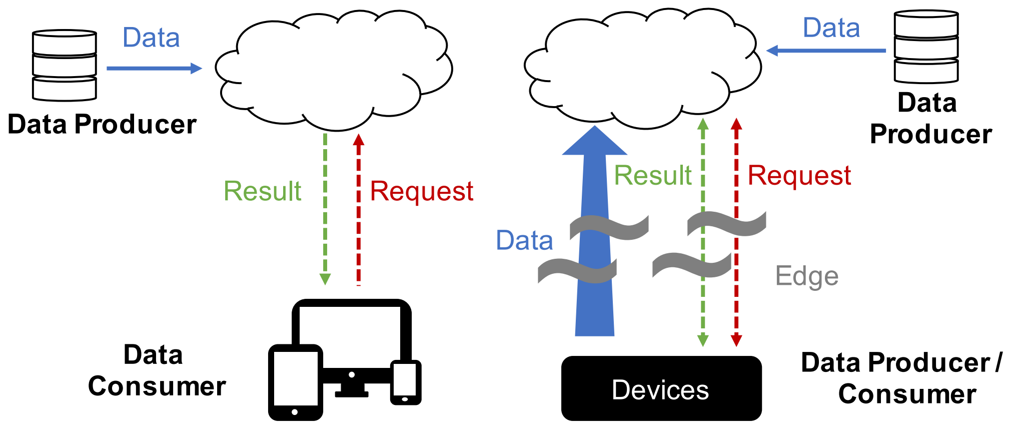


Figure 2.2.: Cloud computing is illustrated on the left, edge computing on the right [Shi et al. 2016]

Figure 2.2 on the right illustrates the edge computing paradigm. There a very heterogeneous set of devices exist, not restricted to compute devices such as computers and smart phones, which are depicted at the bottom. In the context of this work, these are mostly machines or compute resources that are located at the shop floor. These devices do not only consume data, they also provide data to the cloud. This potentially large volume of data is indicated with the blue arrow. In order to not send all data to the cloud, it is necessary to leverage local compute resources at the edge to process parts of this data. There is still bidirectional communication between devices and the cloud, but the goal is to offload compute from the cloud to the edge. Various allocation strategies can be used to allocate the workload, e.g., to distribute it across all available nodes. The entire spectrum can be exploited from performing all processing in the cloud, to a complete edge computing scenario where everything is processed locally and only results are shared. Edge computing devices consume and produce data and are capable of performing computing tasks, because they have data storage, caching, and processing capabilities [Shi et al. 2016]. In this work, we leverage edge computing techniques to run and instantiate adapters to connect directly to industrial machines.

2.3. (Automating) Analytics

This section focuses on how event streams can be analyzed. First, rule-based analytics are introduced by presenting how this is performed on event streams. Then, limitations of such approaches are shown and how they can be overcome by leveraging ML. Different types of learning-based analytics approaches are presented. Additionally, it is shown how the training process for learning-based methods can be further automated.

2.3.1. Rule-Based Analytics

Event streams can be exploited by leveraging rule-based analytics. Therefore, rules must be defined that are then applied on the event streams. One such technique is Complex Event Processing (CEP), where mainly declarative queries are formulated that are then executed in a CEP engine. Such queries are often formulated in an Event Processing Language (EPL), a high-level language to define the behavior of event processing agents[1]. In such languages, Structured Query Language (SQL) like queries can be defined, which are deployed into a query engine. Once a query is deployed it is constantly applied on incoming events, e.g. to detect patterns or certain situations.

For analytics algorithms it is important not only to look at individual events, but to consider instead the context of an event. Therefore, windows are an essential concept. Different types of windows exist and they can be based either on time or on counting the amount of events. In sliding windows, a "window will move or "slide" in time with a period that is usually smaller than the size of the window" [Ari et al. 2012]. This means, there is an overlap of windows and events are usually part of multiple windows. For tumbling windows there is no overlap, so the offset is equal to the size of the window, meaning there is no event that is part of two different windows [Ari et al. 2012]. Tumbling windows are also sometimes referred to as batch windows. Another window type are session windows. Those are "dynamically-sized, non-overlapping, data-driven windows"[2]. Session windows are grouped by a certain key and have no fixed window size. A session is comprised by "a series of events happening in adjacent times followed by a period of inactivity" [Hueske and Kalavri 2019]. Consequently, the window size is not fixed, it rather depends on how long the session is active. Further types of windows exist, but those are not relevant in the context of this work.

Multiple operations can be applied on events, which is usually done within EPAs that are part of an EPN, presented earlier in this chapter. In [Etzion and Niblett 2010], a hierarchy for different types of those agents is introduced. This hierarchy contains filters, different transformations, and pattern detections, which are defined declaratively. However, this

[1]https://complexevents.com/2011/08/23/event-processing-glossary-version-2/ (accessed on 04/10/2021)

[2]https://kafka.apache.org/ (accessed on 04/10/2021)

requires that such rules must be defined by an expert in the domain who is capable to create rules. Furthermore, such rules can become quite complex and sometimes it is impossible to formulate a rule. In those cases, learning-based analytics can be applied that are described next.

2.3.2. Learning-Based Analytics

In learning-based analytics data is not analyzed by predefined rules, rather historical data is used to train a corresponding ML model. Those models are then applied on new data to generate new insights. This is especially interesting for cases where the underlying relationships in the data should be learned, which are often complex and non-linear [Lade et al. 2017]. To do that, analytics software can be used to make predictions about unknown events [Rehman et al. 2019]. Data analytics is defined as: "the application of computer systems to the analysis of large data sets for the support of decisions" [Runkler 2016]. Furthermore, "advanced analytics is a general term which simply means applying various advanced analytic techniques to data to answer questions or solve problems" [Bose 2009].

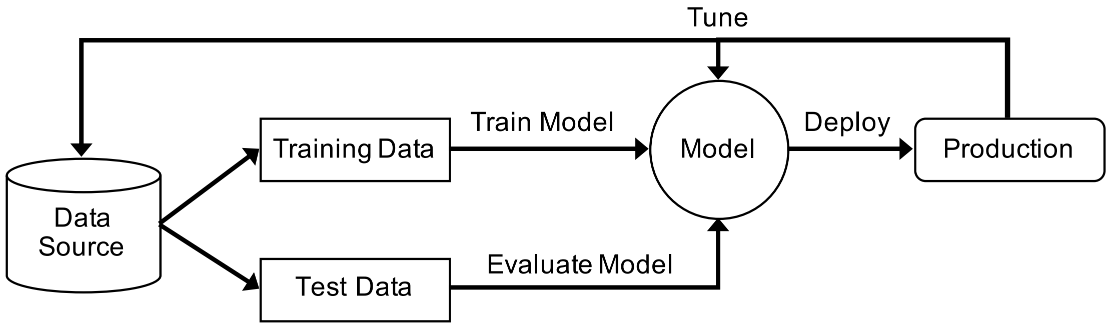


Figure 2.3.: Generic supervised machine learning workflow [Landset et al. 2015]

This especially includes techniques from the fields of ML and Artificial Intelligence (AI) [Gröger 2018]. In this work, we focus on ML for event streams where ML is defined as: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [Mitchell 1997]. One distinguishes between supervised ML, where a label exists for given instances, and unsupervised ML when the data is unlabeled [Kotsiantis 2007]. This thesis focuses on supervised ML models, because we want to leverage the knowledge of domain specialists to generate labels for the data. In general, the workflow for a supervised ML task consists of three steps as depicted in Figure 2.3. First, the data from the data source is divided into a training and test set. A model is trained on the training data and evaluated on the test data. Then the model is deployed into a production system on the right. Based on feedback from the production deployment, the model is further tuned [Landset et al. 2015]. In supervised ML it is distinguished between classification and regression. Unknown labels, representing a class are predicted by classification models, based on the known features for the prediction [Kotsiantis 2007]. For regression, the learning task is to predict a numerical value, instead of a categorical label. Both, classification and regression require a proper set of labeled example data to train a model in order to predict unseen examples [Bifet et al. 2018].

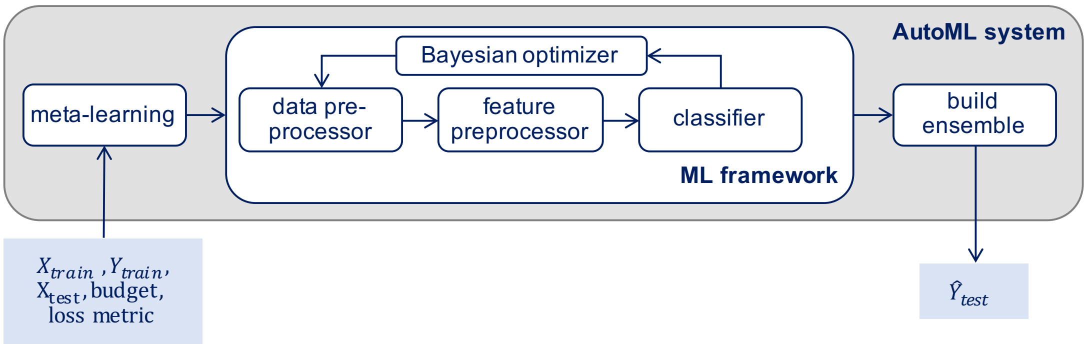


Figure 2.4.: Example of different components of an AutoML pipeline [Feurer et al. 2015]

Defining and tuning ML models is a time intensive task and requires a lot of data science expertise [Gijsbers et al. 2019]. Automated Machine Learning (AutoML) has the goal to automate this process of finding the best model and parameters for a specific problem. It is defined as: "the problem of automatically (without human input) producing test set predictions for a new dataset within a fixed computational budget" [Feurer et al. 2015]. Figure 2.4 shows such an AutoML pipeline. As input, a training data set X_{train} and the corresponding labels Y_{train} are given. Additionally, a test data set X_{test} is provided as well as a a resource budget, and a loss metric. The goal is to calculate the optimal labels for the test set $\hat{Y}_{test}$ regarding the loss metric and given resource budget.

In this chapter, foundations from the field of industrial data management, event processing and learning-based analytics were given. The introduced terms and concepts are an important basis for the rest of this work.

3 Motivation

After introducing the foundations in the previous chapter, this chapter provides the motivation for this thesis. It is organized in four sections as follows: First, Section 3.1 provides a definition and an introduction into industrial event stream analytics. Then, in Section 3.2 an example production line is presented that is used throughout this work to illustrate the introduced methods, models, and tools. Section 3.3 identifies problem statements that are relevant to enable industrial event stream analytics for citizen technologists. In the last section of this chapter requirements based on the previously discussed problem statements are elicited.

3.1. Industrial Event Stream Analytics

Within industrial scenarios, machines often communicate over proprietary protocols. However, this communication usually only takes place within machines and is not synchronized with other machines. Data produced by sensors and actors is used for process control and is usually not persisted for further analytics [Maier et al. 2017]. In this work, we define data produced by such data sources (e.g. machines, sensors, ...) as industrial event streams (Section 1.1), because they introduce new challenges that must be handled when processing and analyzing them. Events are potentially sampled in a high frequency, because sensors and actuators are designed to control processes in real-time. Results must also be available with a low latency [Rüth et al. 2017], which requires computing devices in close proximity to the machine, to avoid transmission of the data over a large network. The physical assets that produce the data are geographically distributed, with multiple assets usually grouped on one shop floor in one building. Such production facilities can be distributed over the whole world. There is a high heterogeneity in both the different event types (e.g. sensor measurements, images), as well as in protocols and interfaces to access the data. However, data produced within one event stream has a homogeneous event schema. This might evolve over time, but such changes are rare. One of the major challenges is to extract business insights and knowledge from this data [Gröger 2018]. To do this, data must not only be processed. Data analytics is required to draw new conclusions or get new insights (exploratory or predictive) [Cao 2017]. Even though production processes are becoming increasingly automated, it is still important to include a verification by humans [Gröger et al. 2016]. Those processes are always very specialized and specific domain knowledge is required to understand and interpret the data and results. Therefore, domain specialists should be included into the analytics task and results must be provided in a way that they can be used by humans. To perform analytics, it is particularly challenging to take into account the high heterogeneity of data sources and the high number of different sensors. This leads to a high dimensionality of the data to be analyzed [Rehman et al. 2019]. To handle this type of data, advanced analytic techniques are required. In order to create them by citizen technologists, they must be automated as much as possible. Additionally, it is important to note that event streams might change over time and specific techniques are required to train and execute Machine Learning (ML) models on them.

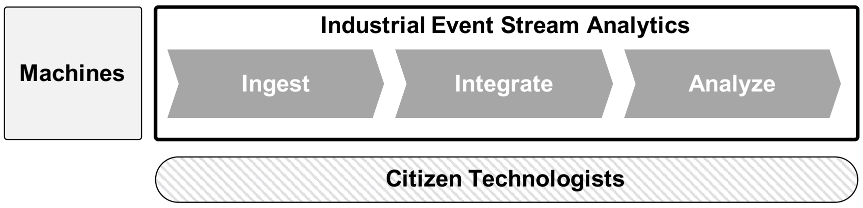


Figure 3.1.: Different phases of industrial event stream analytics

In this work, we describe three phases in industrial event stream analytics, as can be seen in Figure 3.1. On the left are industrial assets, like machines providing the interfaces to access data. First, data must be ingested, then data of multiple devices must be integrated in a harmonized message broker, before it can be analyzed by applying advanced analytics techniques. All of this is done by citizen technologists, which are domain specialists with the necessary technical knowledge.

3.2. Running Example

In this section, an example production line is presented that is used as a motivating scenario throughout this thesis. Although this production line is used as an illustrative example that presents a simplified view, it contains all the relevant characteristics to introduce the problems addressed in this work. The production line manufactures a metal cover, that is later installed in a machine. For this purpose, structures are milled into a raw piece of metal and three screws are pre-assembled, which are later used to fixate it during assembly. Such production lines are usually highly customized, according to the goods they produce. Therefore, it is important to develop a flexible solution that can be easily used in other scenarios and is not just implemented for one case. State-of-the art production lines are highly automated, but often function as a black box for the operator. In this case for example, a camera is used to inspect the milled parts, to confirm that everything is in order for further processing, but there are no possibilities to use other sensor data to track the quality or detect deviations in the process. If a part is bad it is separated and reworked manually or thrown away. This can significantly increase the overall production costs. The business goal is to constantly improve the quality of the produced goods and keep the scrap rate as low as possible, to ensure a high quality for customers. To further improve this processes, sensor data and analytics from the field of Artificial Intelligence (AI) should be used. For implementation, complex technical solutions must be developed by technical experts to connect, process and evaluate the data. Therefore, the aim of this thesis is to enable citizen developers and citizen data scientists to perform those tasks by providing them with methods, models, and tools to connect, process, and analyze the industrial event streams.

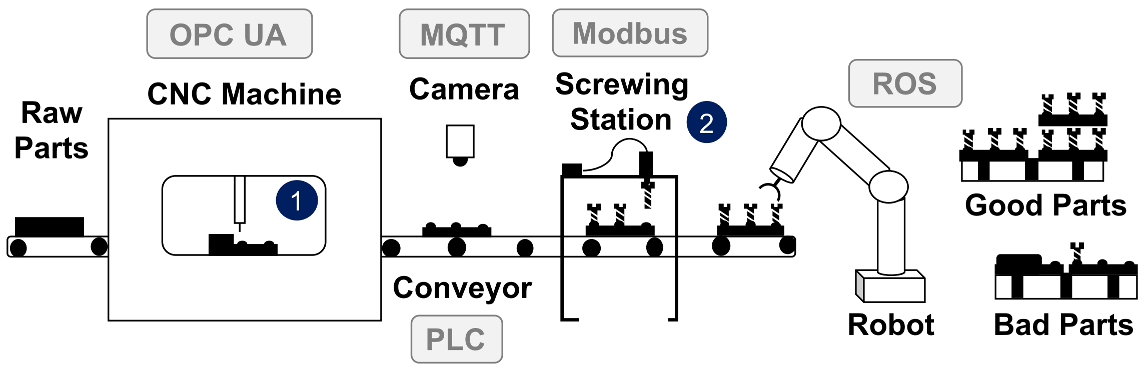


Figure 3.2.: Example production line with the goals to predict ❶ tool wear and ❷ monitor screwing station

Figure 3.2 shows our example production line, where raw metal parts are put on a conveyor on the left, processed at two different stations and then the finished metal covers are placed on a pallet by a robot arm on the right. The conveyor forwards the workpiece from machine to machine for further processing. The first station is a CNC machine that mills a structure into the workpiece and drills three holes for the screws. After the CNC machine, a camera is mounted that verifies if the holes are in the right place and no milling chips are in the holes, because otherwise the screwing station would not be able to tighten the screws. The camera sends the result to the screwing station. When the part is good the screwing station attaches three screws and forwards it to the robot arm. If the result was bad no screws are attached and the part is directly forwarded to the robot. The robot arm puts the finished covers either on a pallet for good parts or if there was a problem on one of the previous steps on the pallet with bad parts. All sensors and actuators in the system are currently used only to control the process, which was designed and built by the machine builder. Now the efficiency shall be further increased by leveraging the sensor data and analytics methods. Tool wear and tool breakage are often the source of failures [Berger et al. 2017]. Therefore, as shown in ❶, the wear of the milling head should be monitored. Another goal is to achieve zero-defects, which is a great challenge [Rehman et al. 2019]. The CNC machine already has a quality control with the camera system, additionally, the sensor data of the screwing station ❷ should be monitored to validate that the screws are assembled correctly. To achieve these business goals, all data produced by different machines must be *ingested* and *integrated* before it can be *analyzed*.

In this scenario, many different assets produce heterogeneous data, ranging from status information (e.g. conveyor on/off), over sensor data (e.g. temperature in machine, joint states of robot) to images (e.g. camera monitoring quality). All those assets typically use various industrial protocols and interfaces to provide data (e.g. Open Platform Communication Unified Architecture (OPC UA), Modbus, Message Queuing Telemetry Transport (MQTT), Programmable Logic Controllers (PLCs), ...). In a first step, the data must be ***ingested***, then it must be harmonized (e.g. measurement units might differ between machines or vendors). Once the data can be read from the different assets it must be ***integrated*** and combined. This is especially relevant for geo-distributed assets since their only connection might be over the internet. Industrial assets can produce a lot of data, but not all of this data is relevant for analytics. Often it is not known in advance, which data is relevant, requiring dynamic reduction strategies capable of intelligently reducing the event streams. Collecting and combining data is the foundation to train ML models to ***analyze*** the data. The citizen data scientist needs to visually explore the data to gain a better understanding of the measured values and combine it with the domain knowledge. Based on this initial exploration, labels (e.g. manually monitored quality of milling head or classify scrap parts) must be provided to train a supervised ML model on historic data. Once this model is trained, it must be deployed to automatically analyze the industrial data streams produced by the assets.

This example is used throughout this thesis, especially in the main parts Section 6, Section 7, and Section 8 to illustrate the main ideas.

3.3. Problem statements

Based on the example presented in the previous section, problem statements are derived. These problems exist when trying to enable business users to evaluate industrial event streams. Special attention is paid to the shortcomings of existing approaches and solutions in order to illustrate which problems still need to be solved. The first subsection deals with organizational problems mainly related to the people involved in performing industrial event stream analytics. Whereas the second subsection focuses on technical problems of the individual phases when ingesting, integrating and analyzing industrial data.

3.3.1. Organizational Problems

In this section, several organizational problems are discussed that occur when industrial assets should be connected and analyzed. First, it is introduced what kind of communication between all the different involved roles is required and how this communication overhead can be reduced by enabling citizen technologist to perform more tasks themselves. Then, the technical debt is elaborated, which must be mastered in order to analyze industrial event streams.

Communication Overhead

To realize projects that leverage industrial event stream analytics, typically teams involving several different roles are required. On the one hand domain experts are needed that have the necessary domain knowledge and on the other hand, technical experts are required to realize and implement the solutions. It takes a long time to build such skillsets and teams and they cause high running costs [Khalajzadeh et al. 2018]. Figure 3.3 shows typical roles that are part of such a project. A *business domain specialist*, e.g. manufacturing process engineer, is an expert in the domain and understands all the processes as well as the data sources [Gröger 2018]. However, they usually lack the technical ability to use advanced analytics or to implement technical solutions. *Device developers* are responsible to write drivers for sensors, actuators, storages or other end user applications of the domain and have a deep understanding of protocols and individual devices [Patel and Cassou 2015]. Data from the connected devices is handled by *data engineers* to build a clean foundation as a basis for analytics tasks. Therefore, a computing and storage infrastructure must be created that is capable of ingesting, cleaning, conforming, shaping, and transforming data [Rehman et al. 2019]. This data is then used as a foundation by data scientists to generate learning models. *Data scientists* can be defined as "people who understand how to fish out answers to important business questions from today's tsunami of unstructured information" [Davenport and Patil 2012]. Once a model is found and properly evaluated, it can be deployed and integrated into an application.

All of those roles play a key part in the realization of industrial event stream analytics and it is important that they work closely together in the development process. A good communication is key to the success of a project, especially because such projects are developed agil and conditions might change over time. This results in high communication overhead between the roles. Communication is made more difficult because everyone has their own area of expertise and a common language must be developed first. The structural differences between each role and their area of expertise cause inefficiencies and increase the complexity of collaboration [Gröger 2018].

To reduce the technical complexity, there has been an increasing effort to enable business domain specialists to handle technical tasks. We call them *Citizen Technologists* in this

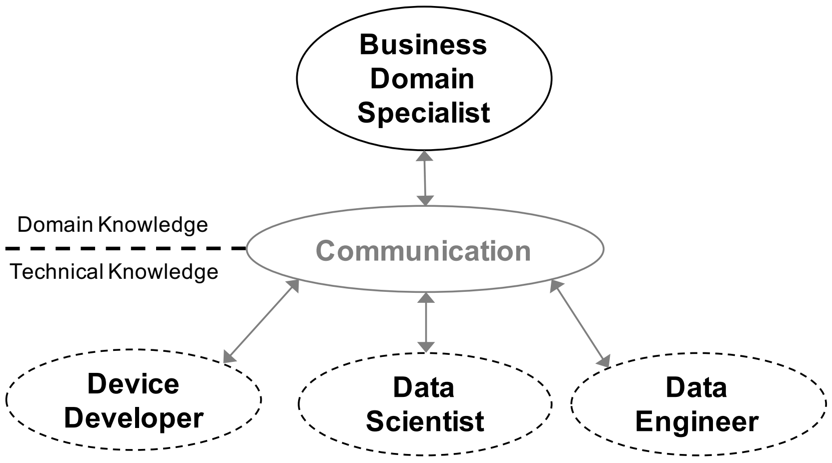


Figure 3.3.: Communication in an industrial event stream analytics project

thesis and they are able to perform tasks that usually requires technical specialists[1]. Figure 3.4 shows an overview of different types of citizen technologists. It should be noted that these are roles and a single person can take several of these roles. Furthermore, there are additional roles, but the ones shown in the figure are the relevant ones for this work.

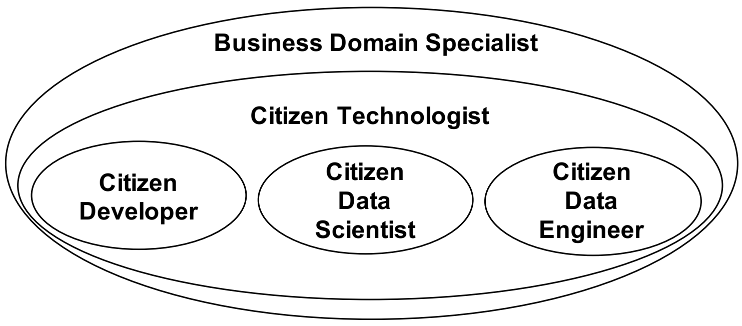


Figure 3.4.: Overview of different citizen technologists

Gartner defines a *Citizen Developer* as "a user who creates new business applications for consumption by others using development and runtime environments sanctioned by corporate IT"[2]. Such applications can also be mobile apps, so that users can interact via their smartphone with the system [Oltrogge et al. 2018]. In this work, one of the main tasks of a citizen developer is to connect industrial assets by modeling adapters that read sensor values. A *Citizens Data Engineer's* main task is to prepare the data in a way that it can be analyzed. This role is closely related to the so called *Pattern Engineer*, who

[1]https://chiefmartec.com/2018/05/democratizing-martech-marketing-technologists/ (accessed on 04/10/2021)

[2]https://www.gartner.com/en/information-technology/glossary/citizen-developer (accessed on 04/10/2021)

composes processing pipelines by choosing appropriate event streams and transforms them with processing elements [Riemer 2016]. The resulting events are then stored and can be used by *Citizen Data Scientists* as a foundation for the training of ML models. A citizen data scientist is defined by Gartner as "a person who creates or generates models that use advanced diagnostic analytics or predictive and prescriptive capabilities, but whose primary job function is outside the field of statistics and analytics".[3] Since these are people from the domain, they are also responsible for providing or verifying the labels for supervised ML. The gap between the technical and business role must further be reduced to enable citizen technologists to perform such tasks themselves. However, a very deep technical understanding is often still required, which is why solutions have to be developed to target less technical users [Khalajzadeh et al. 2018]. The technical debt that must be handled by the presented roles is explained in more detail in the next section.

Technical Debt

In order to perform analytics on data from all assets of a company, a distributed system is necessary that is capable of applying ML on this data. In such systems, many components must work together and the ML part is relatively small compared to the whole system, as depicted in Figure 3.5. It is “common to incur massive ongoing maintenance costs in real-world ML systems” [Sculley et al. 2015], which is mainly due to all the different components that are required. This shows that a holistic solution is needed and that the focus should not only be on the training of ML models.

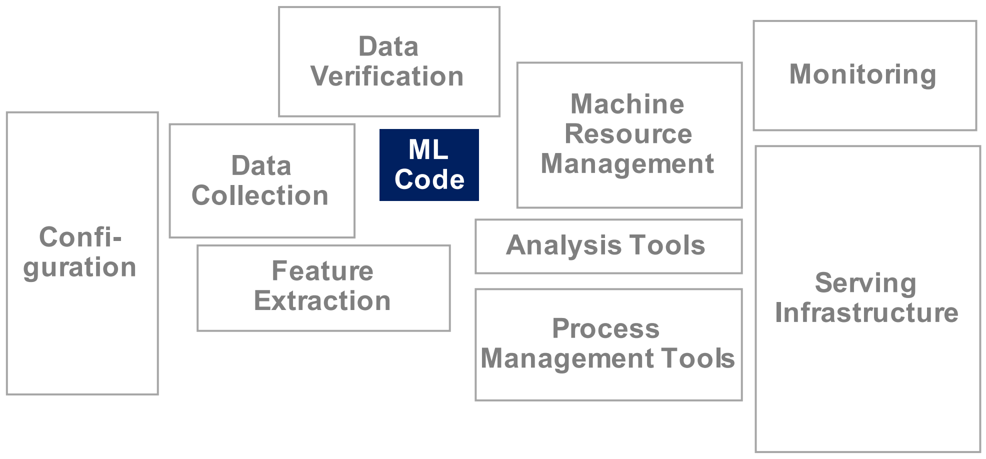


Figure 3.5.: Parts of a machine learning solution [Sculley et al. 2015]

Figure 3.5 shows the different parts that need to be addressed when applying ML models on production data. For example, on the right side the serving infrastructure is de-

[3]https://blogs.gartner.com/carlie-idoine/2018/05/13/citizen-data-scientists-and-why-they-matter/ (accessed on 04/10/2021)

picted. It must be capable of processing all data in a distributed environment and be flexible enough to deploy new models dynamically on event streams. Additionally, this infrastructure must be monitored to ensure that there are no problems during runtime. Besides the infrastructure, a large configuration effort exists, as shown on the left side of the figure. This concerns both the configuration of the ML parameters, as well as the configuration for the different services that are part of the serving infrastructure. Further, data collection, data verification, and feature extraction must be considered. Often many different technologies are used to solve the individual tasks of a data analytics workflow, as can be seen in the example environment in [Lade et al. 2017]. Each of those systems must be administered by a technical expert, therefore, an integrated overall solution is preferred which can be administered automatically to a large extent. A system that is used by citizen technologists must be able to automate most of those tasks, because they do not have the technological know-how to perform them manually. To enable this, we introduce concepts for a holistic solution targeted at citizen technologists. It must support the connection of data sources, transmit and process event streams in a distributed broker based architecture, as well as training and deploying supervised ML models.

3.3.2. Technical Problems

In the previous section the organizational problems that arise when applying analytics on industrial event streams were presented. Now, the focus is on the technical problems that occur with existing solutions. First, the machine connectivity is targeted and what the technical obstacles are to ingest and integrate data from industrial sources. Then, rule-based analytics is introduced and what the limitations are for those techniques. In the last subsection the complexity of data science is discussed and how automation can help to lower the technical barriers.

Machine Connectivity

Data is the foundation to perform any advanced analytics, so the first goal is to collect this data. It is not only important to have a high quantity of data, it must also be of high quality to achieve good analytics results. This means the data does not only need to be extracted from the physical assets, like machines and sensors. But it further needs to be preprocessed and prepared. In a first step data must be ingested, which is a challenging task due to the high heterogeneity of protocols and formats, as well as the high technical complexity of the industrial domain. Additionally, it is very likely that similar machines of different vendors have diverse data models, adding even more complexity. This means we need a way to connect to the external data sources and make the data processable. Several new technologies and standards were developed to ease and standardize the machine connection. OPC UA[4] for example is such a standard that is getting more and

[4]https://opcfoundation.org/ (accessed on 04/10/2021)

more popular in resent years. Further standards exist, unfortunately there is not one standard and it is likely to assume that there always will be many different standards. Also machines have a rather long live time, which makes it likely that there are many legacy machines on the shop floor as well.

PLC controllers are often used in automated production lines to control machines by implementing execution logic. Often, they only provide proprietary interfaces requiring custom solutions to read data. Open Source libraries such as Apache PLC4X[5] introduce a unified interface to such devices, but still a lot of programming effort and knowledge is required that usually cannot be performed by citizen technologists. For the different standards and protocols (e.g. Robot Operating System (ROS), MQTT, Hypertext Transfer Protocol (HTTP)) multiple libraries are available. In this section we use PLC4X as a representative example to illustrate the technical barrier for none technical users. Listing 3.1 shows a code snipped of the Apache PLC4X library, connecting to and reading data from the PLC controlling the conveyor in our example in Section 3.2. First, a connection to the PLC has to be established (line 1). Then a builder request, containing the PLC registers to read is constructed and executed (line 4 - line 10). The values can then be read from the response in order to work with them (line 12 - line 13). With PLC4X, an abstraction layer was created that allows developers to connect such data via multiple software programming languages (e.g. Java).

```
1  try (PlcConnection plcConnection = new PlcDriverManager()
2                                              .getConnection("s7:192.168.188.22")) {
3
4    PlcReadRequest.Builder builder = plcConnection.readRequestBuilder();
5    builder.addItem("I_entry", "%I0.3:BOOL");
6    builder.addItem("I_speed", "%I1.0:INT");
7
8    PlcReadRequest readRequest = builder.build();
9
10   PlcReadResponse response = readRequest.execute().get();
11
12   boolean entry = response.getBoolean("I_entry");
13   int speed = response.getInteger("I_speed");
14 }
```

Listing 3.1: Java code to read the speed and status of a light barrier from a PLC

To connect data sources with such libraries, it is still necessary to write, compile and deploy code using the respective programming language. However, there are solutions (e.g. Kafka Connect[6]) that try to reduce this implementation effort. Kafka Connect provides a toolbox of different connectors where users only have to define configuration parameters to start and execute those adapters. It is not necessary to write software, but it still requires a deep technical understanding. In addition to the connection, data has to be (pre-) processed and cleaned, which is often done manually. For this purpose, rules are implemented to run at the message broker used for data integration. Examples for such

[5]https://plc4x.apache.org/ (accessed on 04/10/2021)
[6]https://www.confluent.io/connectors/ (accessed on 04/10/2021)

message brokers are Apache Kafka[7] or Eclipse Mosquitto[8]. Often, tooling is provided to facilitate the work with those brokers and integrate data, like the previously introduced Kafka Connect. Such brokers can be used to process industrial event streams, however, they lack the ability to dynamically adapt the event stream at the producer. Consumers - like ML algorithms - do not always need data in the original quality and frequency, since features are often calculated before the learned model is applied. Depending on those consumer demands, the event stream could be reduced early in the producer, resulting in a lower consumption of the network bandwidth. To reduce the complexity of the algorithms consuming the data, this reduction logic should be performed by the broker itself not effecting the consumer.

So far, the technical problems of ingesting and integrating industrial event streams were illustrated, which are usually solved by software engineers. In this work, we try to enable citizen developers to perform those tasks.

Limitations of Rules

In this section the focus is on the analytics part. First, it is shown how rule-based approaches are used to evaluate event streams and what the limits for those approaches are. Then, techniques from the field of data science are shown and how they can solve those limitations.

Rule-based methods are a common way of evaluating event streams. Users can define rules that are then constantly applied on event streams. This can range from simple rules (e.g. filters and aggregations to clean up data), up to more complex rules (e.g. detect complex time dependent patterns). This is a well established field, with different existing solutions. Two main user interaction methods exist, either providing a Domain Specific Language (DSL) or a Graphical User Interface (GUI) for users to define rules manually.

As introduced in the foundations in Section 2.2, streaming engines often provide a Structured Query Language (SQL) based DSL to define queries for event streams. Examples for such engines are Esper[9], Apama by Software AG[10], or Siddhi[11]. All of those tools have different features, however, their core functionality is similar and will be explained using Siddhi. For this example, we use the previously connected data from the conveyor PLC containing the speed. Due to operating errors of a shop floor worker it is possible that the speed of the conveyor is set too high. To detect this situation, a rule should be established that constantly monitors the data. Listing 3.2 shows the resulting query. Line 1 contains the from statement, where the *conveyor stream* and a time window of five seconds is specified. From this stream, the property *converyorId* is selected and the

[7]https://kafka.apache.org/ (accessed on 04/10/2021)
[8]https://mosquitto.org/ (accessed on 04/10/2021)
[9]https://www.espertech.com/ (accessed on 04/10/2021)
[10]https://www.apamacommunity.com/ (accessed on 04/10/2021)
[11]https://siddhi.io/ (accessed on 04/10/2021)

average speed within the time window is calculated. This smoothens the curve and ensures that individual outliers do not have a big impact. The event stream is grouped by the *conveyorId* and events are only forwarded when the having statement is fulfilled and the *average conveyor speed* is higher than five. Such DSLs are quite flexible and the queries can be adapted for different use cases, however, they are still quite technical and users have to learn the syntax. That is why graphical solutions are often used.

```
from ConveyorStream#window.time(5 sec)
select conveyorId, avg(speed) as avgSpeed
group by conveyorId
having avgSpeed > 5
```

Listing 3.2: Siddhi Query to aggregate and filter events

Tools with a GUI often allow users to model data flows, which are commonly used to process and harmonize data from various sources. Examples for such tools are Talend[12], or StreamSets[13]. They are usually used for Extract, Transform, Load (ETL) tasks, where one of the main goals is to gather data from many heterogeneous sources and store it in a database. Another such tool is Node-RED[14], which provides a graphical user interface to create event-driven applications. These solutions are more tailored to citizen technologists, but the focus is on rule-based analytics rather than learning-based approaches, and they do not cope with the characteristics of industrial event streams.

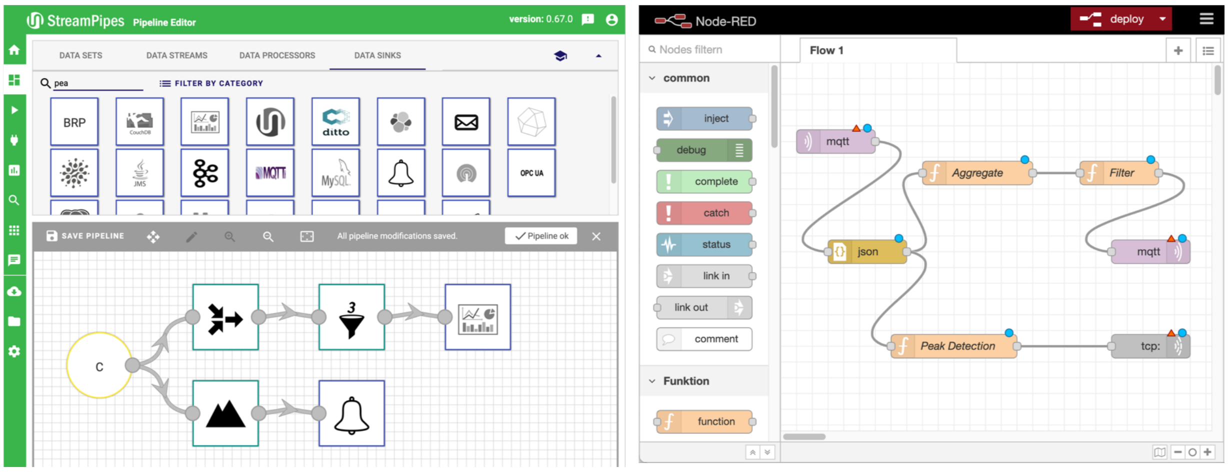


Figure 3.6.: Screenshot of Apache StreamPipes (left) and NodeRed (right)

Figure 3.6 shows the same example as before, that is now realized with two graphical tools. On the left Apache StreamPipes[15] is used and on the right NodeRed. In both

[12]https://www.talend.com/ (accessed on 04/10/2021)
[13]https://streamsets.com/ (accessed on 04/10/2021)
[14]https://nodered.org/ (accessed on 04/10/2021)
[15]https://streampipes.apache.org/ (accessed on 04/10/2021)

cases the previously described rule to detect the maximum speed is modeled and a peak detection is added. Peaks of the speed mean that the conveyor is accelerated for a short period of time and then breaks again. As this behavior might damage the motors and therefore it should be avoided. This shows the flexibility of graphical approaches, because the rule can be modeled in a drag-and-drop-like interface and results can be further reused by adding splits in the pipeline.

So far, we have shown different approaches on how to define rules that are then applied on event streams. Next, we want to describe limitations of rule based approaches. The chart in Figure 3.7 on the left shows the time-series for the speed value of the conveyor. The patterns can be clearly detected visually and rules can be derived from them and executed with one of the shown approaches (DSL or GUI).

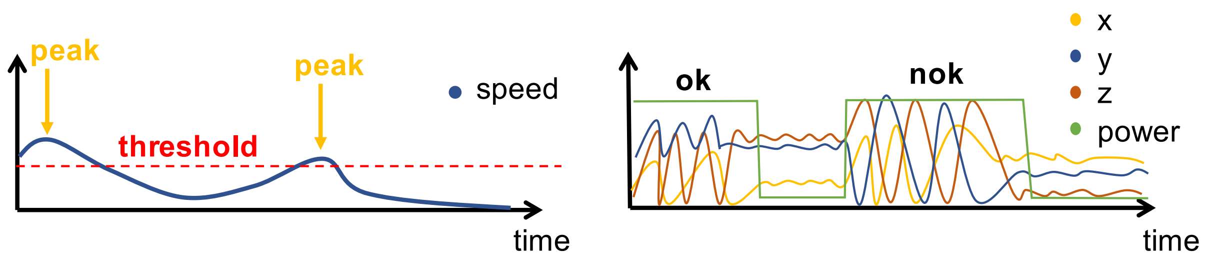


Figure 3.7.: On the left speed data. On the right data of an acceleration and power consumption sensor

However, patterns are not always easy to detect and the data can have high dimensionality. This is especially true for industrial event streams, where we often have a lot of different properties and also specific sensor types. One such an example is shown in Figure 3.7 on the right where we have an acceleration sensor and a power consumption sensor. The goal is to detect whether the produced part is ok or not ok. In this case, no simple threshold can be defined and additionally, such thresholds might be slightly different in another machine, depending on factors like temperature. In such cases rules often become too complex or it is not even possible to define a rule. Therefore, advanced analytics techniques could be helpful, where a ML model is trained on data and can then be applied on new data. Those models are often capable of detecting non-linear correlations. In order to train a model, training data must be available and model parameters must be defined. This will be discussed in more detail in the next section.

Complexity of Data Science

Specially qualified data scientists are used to analyze data and train ML models. However, in order to create good models, it is not sufficient to only have analytical skills, the related domain expertise is required as well [Huber et al. 2019]. A widely adopted approach for data science projects is the Cross-Industry Standard for Data Mining (CRISP-DM) [Shearer 2000] that is often applied by data scientists. It consists of six different phases shown in Figure 3.8.

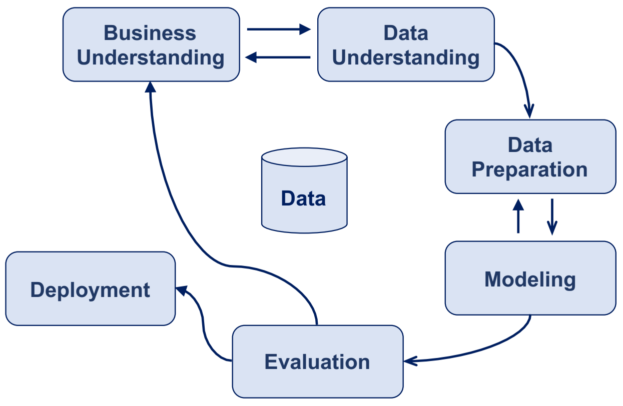


Figure 3.8.: CRISP-DM cycle [Shearer 2000]

The CRISP-DM cycle is an iterative approach to evaluate the data in an exploratory way. It starts with *business understanding* and the definition of the desired goal. This is followed by the *data understanding* phase where the availability of the necessary data is checked. Hereafter, the data is prepared, harmonized and cleaned in the *data preparation* phase. During the *modeling* phase, a model is created that is then *evaluated* in the following phase. If the model results are satisfactory it is deployed into production and the gained knowledge is used again in the business understanding phase. In the previous sections, the technical debts of the implementation of analytic projects have already been discussed. The CRISP-DM cycle is a methodology that describes how data can be analyzed, however, there is no focus on those technical realizations. Additionally, the data acquisition phase is not considered in CRISP-DM. Therefore extensions, like the one presented in [Huber et al. 2019], have been developed. As the data foundation is an important building block for good analytics results, this is a major focus in this work. Especially, because we target citizen technologists, it is important that this step is covered in a way that it can be performed with little technical know-how. After the data is available a model has to be trained. Therefore, it is necessary to perform the steps of data preparation, modeling, and evaluation. Those usually require a good understanding about data science techniques. There are efforts to automate these steps and provide Automated Machine Learning (AutoML) [Feurer et al. 2015]. These automations can be used by data scientists to find good models more efficiently, but they can also be used by citizen data scientists to train models. The goal of this work was not to develop a fully autonomous system, but rather to support humans in decision making, by automation of some technical steps and presentation of results. Current solutions do not focus on industrial event streams and often require the data to be in a form where the features and labels are present. This affects also the deployment of the trained models, because the time-series data must be transformed before the model can be applied. Current solutions also mainly focus on the data preparation and model training steps of the CRISP-DM cycle and pay less attention to the data acquisition and deployment of the models.

Next, we will present different tools that are used by data scientists within the CRISP-DM cycle to create models. Usually, the definition of the training task is done programmatically or with a graphical tool. In the end of this section, solutions are presented that use AutoML procedures and enable the user to create models in a guided process.

Several different programming libraries exist that allow users to train and apply ML models in the programming language of their choice. Examples for such libraries are scikit learn [Pedregosa et al. 2011], H2O [Cook 2016], or tensorflow[16]. Many other libraries exist and some are even available as open source software. Developers can choose one depending on the given requirements. Listing 3.3 shows an example how the training of models can be defined in scikit learn using python. First, the (tree-based) model that should be used is imported in line 1. Then, the training data is specified in line 3. This is an array of feature vectors for each training example. The label for each example is defined in line 4. After that, the model is initialized and the fit method is called to train the model. Once the model is trained, it can be applied on new feature vectors as can be seen in line 8. It is important that the feature vectors contain the same information in both the training data, as well as the new instances that should be predicted. The example only illustrates the base functionality, usually more code is required (e.g. create a train / test split of the data before training).

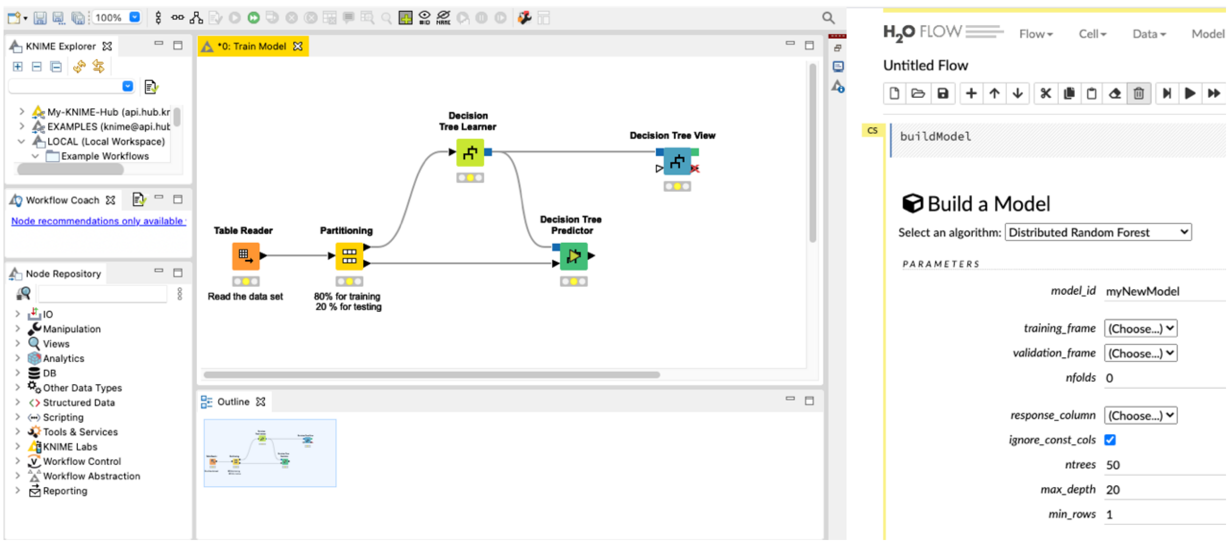
```
1 from sklearn import tree
2
3 train = [[2,2], [1,1]]
4 label = [1,0]
5 model = tree.DecisionTreeClassifier()
6 model.fit(train, label)
7
8 result = model.predict([2.,2.])
```

Listing 3.3: Python code training a decision tree with scikit-learn

Figure 3.9 shows two examples for graphical approaches to define and train ML models. Again, there are many different solutions available, both open source and also commercial products. The shown examples are only to illustrate the core functionality of tools within this category. On the left is KNIME [Berthold et al. 2009]. Analytics workflows can be defined in a drag-and-drop-like interface. On the right is the GUI for H2O Flow[17], where the configuration parameters can be defined and then the model is trained. H2O also has a programmable interface similar to the one shown in Listing 3.3. Such graphical tools focus on users with a deep data science knowledge, that do not prefer to define their analytics workflows in code.

The examples show that either a few lines of code or with a configuration in a GUI it is possible to train ML models. However, the complexity lies within the amount of

[16]https://www.tensorflow.org/ (accessed on 04/10/2021)
[17]https://www.h2o.ai/ (accessed on 04/10/2021)



```
buildModel
```

Build a Model

Select an algorithm: Distributed Random Forest

PARAMETERS

model_id	myNewModel
training_frame	(Choose...)
validation_frame	(Choose...)
nfolds	0
response_column	(Choose...)
ignore_const_cols	✓
ntrees	50
max_depth	20
min_rows	1

Figure 3.9.: Definition of a tree-based model using graphical tools. KNIME on the left and H2O flow on the right

configuration possibilities. There is not a single best model for all problems and each model type has many different configuration possibilities, which must be specified and might differ between data sets. To overcome this problem, solutions that use AutoML to find a good model for a specific ML problem were developed. Several open source libraries (e.g. auto-sklearn [Feurer et al. 2019], TPOT [Olson and Moore 2016]) exist. Their programming interfaces are similar to the one shown in Listing 3.3, but instead of specifying all model parameters, the frameworks search for the best configurations. A benchmark for the different approaches can be found in [Gijsbers et al. 2019]. Several commercial solutions exist, mostly from companies that already provide analytics solutions like RapidMiner[18]. Additionally, big technology companies like Microsoft provide solutions for AutoML within their cloud services. Those solutions mainly focus on citizen data scientists, but they target the data science workflow. Users are able to upload files and define their ML problem in a guided workflow. In this work, we focus on time-series data created by industrial machines, where the model should be directly deployed into a production system. Most of the existing tools focus on the low-level data analytics process of coding and providing basic visualizations of results [Khalajzadeh et al. 2018]. It is also challenging to develop standardized, generic reusable analytics services, therefore, a feasible approach is to create a domain-specific data analytics solution [Gröger 2018]. Current techniques tend to focus on automation of the training process, lacking a holistic approach to train models for industrial event streams and even applying them directly on real-time data.

[18]https://rapidminer.com/ (accessed on 04/10/2021)

3.4. Requirements Elicitation

In this section, we discuss requirements for methods, models, and tools that enable citizen technologists to analyze industrial event streams. Requirements elicitation is defined as "the process of seeking, capturing and consolidating requirements from available requirements sources" [Glinz 2011]. The requirements are elicited based on the previously discussed problem statements, as well as on various papers from the fiel of IIoT analytics. Figure 3.10 depicts an overview of the three steps of industrial event stream analytics and highlights the different requirements for each step. It begins with some overall requirements for the whole approach, continuing with requirements related to ingesting data. After that, requirements focusing on the integration of data from multiple sources are presented, closing with requirements that are related to analytics of industrial event streams are discussed.

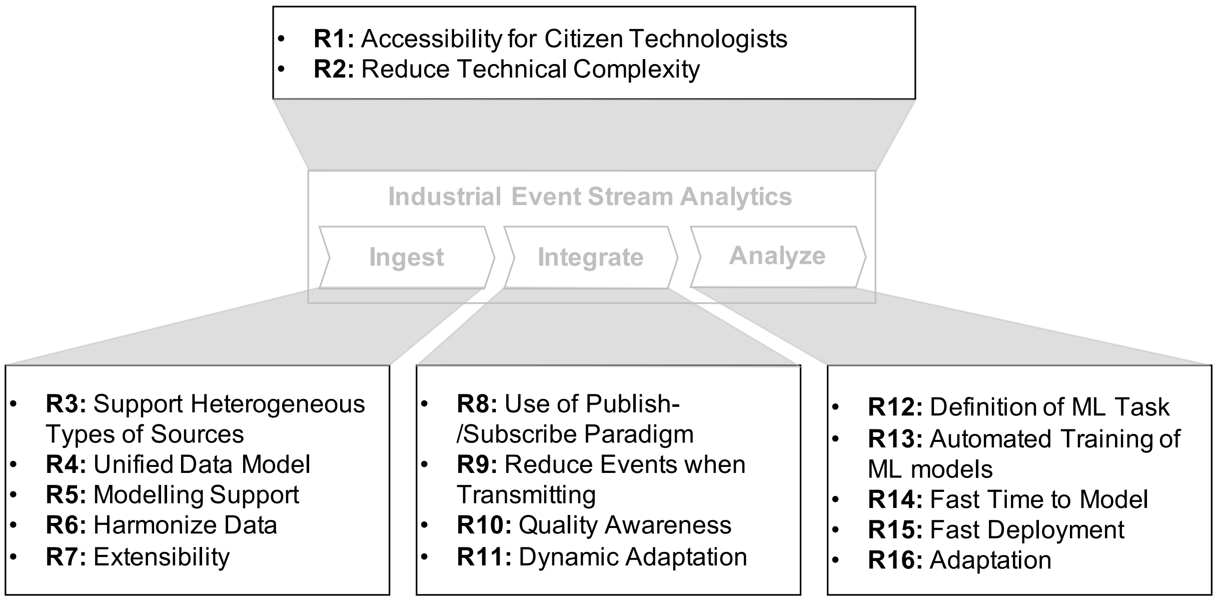

• R1: Accessibility for Citizen Technologists
• R2: Reduce Technical Complexity
Industrial Event Stream Analytics
Ingest
Integrate
Analyze
• R3: Support Heterogeneous Types of Sources
• R4: Unified Data Model
• R5: Modelling Support
• R6: Harmonize Data
• R7: Extensibility
• R8: Use of Publish-/Subscribe Paradigm
• R9: Reduce Events when Transmitting
• R10: Quality Awareness
• R11: Dynamic Adaptation
• R12: Definition of ML Task
• R13: Automated Training of ML models
• R14: Fast Time to Model
• R15: Fast Deployment
• R16: Adaptation

Figure 3.10.: Overview of the requirements for this work

3.4.1. Overall requirements

In this section, two overall requirements are presented, which permeate the entire work. We mainly target citizen technologists by providing an integrated approach to automate industrial event stream analytics.

Requirement R1: Accessibility for Citizen Technologists
Citizen technologists should be facilitated to perform industrial event stream analytics themselves.

The main expertise of citizen technologists lies usually in the subject domain and not in technical fields. Therefore, it is necessary to provide technical support for such roles and facilitate them to analyze the data on their own instead of relying on technical experts. This reduces the required communication overhead in data analytics projects.

Requirement R2: Reduce Technical Complexity
An integrated approach is necessary to realize data collection, training of ML models, and execution of those models.

There is a high technical debt when realizing systems for industrial event stream analytics, as introduced in the previous section. An integrated approach is required that covers all of the mentioned functionalities (e.g. data collection, model training, ...). Since the targeted user group are citizen technologists, it is important that the technical integrations can be guided and automated as much as possible.

3.4.2. Model & ingestion requirements

This requirements category focuses on ingesting data from industrial data sources. It covers both the model to describe different data sources as well as the description of the data itself and how the modeling can be achieved by citizen developers.

Requirement R3: Heterogeneous Types of Sources
Different kinds of industrial data sources as well as different types of data can be connected.

In industrial settings, many different standards and interfaces exist that can be used to collect data from machines. There is especially a wide heterogeneity of different machine types, all producing different kinds of data. Therefore, the model for accessing those data sources must be able to cope with such differences and support a wide variety of them. Here, the focus lies in particular on machines and sensor data, which was the most frequently mentioned content of data in a study of requirements for industry 4.0 data processing [Gölzer et al. 2015].

Requirement R4: Unified Data Model
A unified data model for consuming and processing connected event streams.

Once data sources are connected they must be described in a unified way. Both the semantic meaning and the technical interfaces should be harmonized to ease the subsequent analytics of this data [Gölzer et al. 2015]. There must be (semantic) meta information, such as a common vocabulary, that facilitates the integration and the analytics tasks.

Requirement R5: Modeling Support
Modeling support is provided to enable citizen developers to connect new industrial data sources.

A guided way to integrate new data sources is required that supports a citizen developer by reducing the technical complexity. The heterogeneity of different data sources makes it difficult to fully automate the whole process, especially as there is often a lack of (machine readable) meta-information. Hence, the goal is to provide a flexible way to quickly add new data sources in agile environments [Gröger et al. 2016].

Requirement R6: Harmonize Data
In order to ensure a harmonized data basis, preprocessing rules are required for the transformation of data upon ingestion.

Due to the different representations of data it must be possible to preprocess and harmonize the data on ingestion, in order to provide a clean data basis. This is one of the key requirements for analytics systems working on IIoT data [Rehman et al. 2019]. It should be done directly during ingestion to ensure that further calculations are performed on harmonized data.

Requirement R7: Extensibility
To be able to cope with changes in the future the approach should be extensible.

Due to the very high heterogeneity and still active development of new standards, the model describing the ingestion of industrial data sources must be extensible. It must be able to handle new types of data sources which might be implemented in the future, as well as domain specific standards that are not already integrated. Further, it is necessary to extend the model with new transformations when necessary, for example when some domain specific type of data requires certain preprocessing.

3.4.3. Integration requirements

The following requirements focus on the integration of industrial event streams. Data of different assets must be integrated and combined to create a basis of data for analytics. In particular, the distributed infrastructure must be taken into account and the transmission of data should be optimized to reduce the load in the network.

Requirement R8: Use of Publish-/Subscribe Paradigm
Already existing publish-/subscribe systems should be used and extended to process industrial event streams.

Existing solutions of message brokers realizing the publish-/subscribe paradigm are often already capable of routing data and making it available for different analytics algorithms. Those already existing technologies should be reused and extended to cope with the characteristics of industrial event streams. This can be achieved by designing a wrapper around existing technologies that provides functionalities while reusing already well established concepts.

Requirement R9: Reduce Events when Transmitting
Machines can produce large amounts of data, which should be (pre-)processed on the edge and intelligently reduced when transmitting, to reduce the load of the network.

Sensors and actuators in machines or production lines are designed to control processes in real-time and, therefore, often have a high sampling rate. If this information should now be used for analytics purposes, large amounts of data can be generated due to the high frequency and number of sensors. Analytics systems must be able to handle such enormous amounts of data and generate results with low latency [Rehman et al. 2019]. Therefore, edge computing devices are needed that can process data locally and possibly reduce the quantity before transmission over the network.

Requirement R10: Quality Aware
Subscribers must be able to define the quality of the required data.

When analyzing data, different subscribers might have different quality requirements for the same data, such as the required event stream frequency. To be able to realize this, it must be possible to define the quality requirements per subscriber instance and transmit the data accordingly. These quality requirements may change over time, as shown next.

Requirement R11: Dynamically Adapt Industrial Event Streams
Quality requirements might change over time, therefore, it must be possible to dynamically adapt streams itself according to the requirements of the consumers.

In order to apply ML models on event streams, it is not always necessary to transmit all the information. This should be exploited to reduce the events accordingly at the edge. Since the requirements of consumers can change constantly, it is necessary to dynamically adapt event streams automatically without the intervention of a citizen technologist.

3.4.4. Analytics-related requirements

The requirements in this subsection focus on analyzing industrial event streams by training and deploying supervised ML models. To do this, no specialized data scientist is required, rather citizen data scientists should be empowered because they have the necessary domain expertise to be able to understand the meaning of the data.

Requirement R12: Definition of ML Task
Citizen data scientists are able to define a machine learning task based on integrated data.

Business and domain decision parameters should be predicted on past data [Gölzer et al. 2015]. Therefore, citizen data scientists must be able to define ML tasks, which are then executed automatically and train a model. A guided workflow based on previously collected data can help the citizen data scientist to create a task definition. Once the training is completed, the resulting model should be presented to the citizen data scientist to validate its performance.

Requirement R13: Automated Training of ML Models on Industrial Event Streams
Supervised ML models for time-series data should be trained automatically on the basis of (labeled) industrial event streams.

Techniques and approaches from the field of AutoML should be used to automatically train supervised ML models on time-series data. Data should not be preprocessed or edited, before the AutoML approach can be applied. The previously created description of the data streams should be sufficient to define the ML problem and to train a model automatically.

Requirement R14: Fast Time to Model
For citizen data scientists it should be possible to define ML models with limited time effort.

The manual effort for training a ML model should be low due to the automatic search for the best configuration settings for a given data set. The training itself might take longer, but is completely automated, and does not need any manual input. Once the training is complete, it should be possible to evaluate the results in order to be able to directly deploy the best model.

Requirement R15: Fast Deployment of ML Models
Trained machine learning models can be deployed to process industrial event streams.

Previously trained and evaluated ML models can be deployed in a distributed environment as reusable components without any implementation effort. A graphical modeling interface should be provided for citizen technologists to integrate those models into pipelines and constantly process industrial event streams.

Requirement R16: Adaptation
As circumstances or data changes, it should be possible to (re-)train new models.

Data itself, or the distribution of the data might change over time. Therefore, event stream should be monitored if there are structural data drifts that might effect the quality of the predictions. All of this should be overseen by a citizen data scientist who can add additional domain knowledge as to why these changes occurred in the first place and retrain the model accordingly.

4 Related Work

In this section, related work in the context of industrial event stream analytics is presented and discussed. The main goal of this thesis is to facilitate data-driven analytics for citizen technologists. Therefore, an infrastructure and systems must be provided that are capable of performing Industrial Internet of Things (IIoT) analytics. Such systems are often complex and consist of multiple components. Figure 4.1 depicts the relationship between the different areas of related work and how they are linked to the different parts of this thesis. Section 4.1 presents research from *architectures for IIoT analytics* that are capable of supporting the whole data analytics workflow. After architectures have been assessed, the focus is placed on the three sub-aspects. First, the related work from the area of *4.2 connect & model IIoT data sources* is presented, then from *4.3 data processing & reduction*, and lastly discuss the area of *4.4 self-service data analytics*. These areas are important building blocks that must be considered in order to achieve the overall goal of providing a holistic solution for citizen technologists.

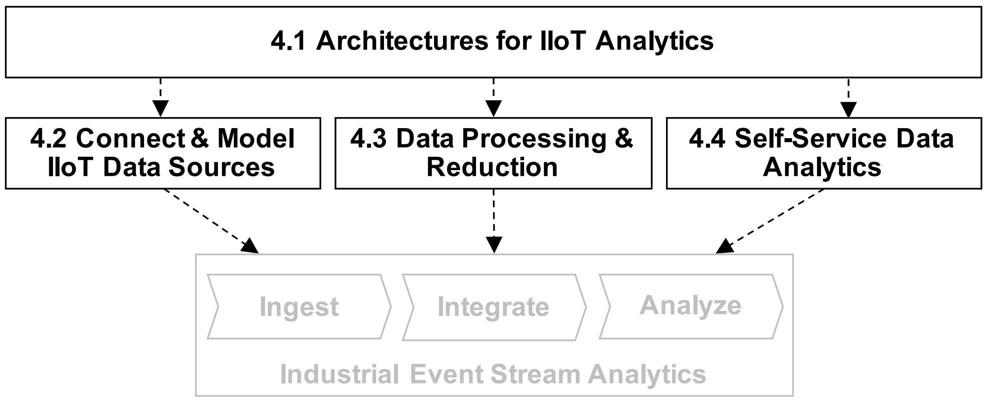

4.1 Architectures for IIoT Analytics
4.2 Connect & Model IIoT Data Sources
4.3 Data Processing & Reduction
4.4 Self-Service Data Analytics
Ingest
Integrate
Analyze
Industrial Event Stream Analytics

Figure 4.1.: The different areas of related work

4.1. Architectures for IIoT Analytics

IIoT analytics places special demands on the infrastructure and systems that process streaming data. Especially, the distributed setup and high heterogeneity of data sources introduces new challenges that must be covered by the system architecture and data models. In this section, we first introduce some reference architectures for IIoT analytics. Then, further approaches are presented, which are able to analyze data from heterogeneous data sources.

The PERFoRM [Gosewehr et al. 2017] project introduces a middleware-oriented architecture. The middleware acts as a mediator, allowing systems to communicate with multiple formats [Trunzer et al. 2019]. An information model for the semantic description is specified and "tools, which are developed within the project directly are expected to implement the standard interfaces" [Gosewehr et al. 2017] to communicate over the middleware. There are five main requirements on the middleware *1. aggregation*, *2. processing*, *3. presentation*, *4. publication*, and *5. protection*. These are important requirements that we also consider for the concepts of our message broker. The system provides a loosely coupled solution that is easy to extend since the middleware is the central exchange link. However, for legacy systems "tool-specific adapters need to be developed to expose its data" [Gosewehr et al. 2017]. This takes a lot of manual implementation effort, which is why we try to partially automate the adapter creation to speed up the process of data connection. Another flexible architecture for data mining in automated production systems is introduced in the IMPROVE project [Trunzer et al. 2017]. It is a "layered architecture [that] differentiates between data suppliers, data users, and dashboards" [Trunzer et al. 2019]. Likewise, a middleware is used for data management and integration. It is a four layer architecture consisting of a *data source layer*, *integration layer*, *analysis layer*, and a *dashboard layer*. The authors state that "programming adapters for legacy systems is a major obstacle"[Trunzer et al. 2017]. Inspired by the two previously introduced approaches the paper [Trunzer et al. 2019] proposes a reference architecture for IIoT applications. This reference architecture shows the importance to provide interfaces for humans to interact with the data, because they control the processes of the machines. Further, it is described that adapters are relevant to retrieve data out of all the different systems. At the "heart [of the architecture is] the data management and integration bus" [Trunzer et al. 2019]. Several technologies (e.g. Open Platform Communication Unified Architecture (OPC UA) or RabbitMQ) are proposed to implement this integration bus. In this thesis, we decided to rely on message brokers since they allow us to build a flexible distributed system processing streaming data. Here, also adapters are used to "translate between different information models and protocols" [Trunzer et al. 2019]. However, it is also stated that those adapters must be "programmed individually for each information model and protocol" [Trunzer et al. 2019], which leads to additional manual work. A more advanced architecture is introduced in the COGNITWIN project. In [Abburu et al. 2020] the authors state that a typical set-up requires a sensor network to continuously collect data and monitor different assets and processes. All this data should be stored in a database. At the center of the architecture for a cognitive digital twin is also a message broker to exchange data that is provided by adapters connected to machines supporting multiple formats. The so-called COGNITWIN Toolbox (CTT) consists of five layers. First, the *data ingestion and preparation layer*, where data and metadata is stored in repositories and several data ingestion and cleaning services are provided. With the adapter concept introduced in our work, it is possible to automatically create adapters without any implementation effort. Second, there is the *model management layer* ensuring access and storage of different types of models. Third, the *service management layer* processes data produced by the related physical assets. In our work, we try to enable citizen technologists by leveraging Automated Machine Learning (AutoML) approaches to train models with a service. The resulting models are then executed on event streams. The fourth layer focuses on visualizations & reporting, which is important for users to interact with the system. The last and fifth layer handles the management of the digital twins.

All the previously introduced reference architectures have in common that adapters connect machines to a central broker for data exchange, from which further analytics services can process the data and provide an additional interaction layer for users. Our approach is based on these concepts, but aims to reduce the technical complexity for the user. This was chosen, among other reasons, to ease the data exchange between different network zones, which are common in manufacturing setups.

There are more approaches to connect and process data from distributed systems, as presented in [Kirmse et al. 2018]. The authors describe a lightweight approach for data integration and big data analytics, that copes with the distributed nature of IIoT data sources. All raw events are stored in a distributed storage, where they are later available for analytics jobs. The complexity of data integration is reduced by only adding a minimal set of additional meta-information to the events, without further transforming them. The data is then analyzed at a central location with no network restrictions. Due to the high heterogeneity of data sources, we try to leverage the semantic knowledge to describe the data and (pre-)process it in close proximity to the source, which in the end eases the downstream analytics tasks. However, there are other approaches that deal with the semantic heterogeneity of IIoT systems. In [Jirkovský et al. 2017], several types of semantics heterogeneity are described and an ontology (called SHS ontology) is introduced that is used to transform and harmonize the data into a unified triple store optimized for big data applications. SHS is based on the SSN ontology[1]. Additionally, an architecture with four layers is proposed. First, the *data acquisition* layer is used to connect the different sources, then, in the *transformation layer* data is transformed according to the SHS ontology and stored in the *data storage* layer. Then, data can be queried via SPARQL queries and further analyzed with analytics frameworks in the *analytic* layer. In our approach, we use the flexibility of ontologies to harmonize data from different sources, but we use a more lightweight way for the events. Semantic descriptions are only used as meta-data and event streams are not transformed into triples. In this way, we can integrate existing data sources without having to change the data and still benefit from the advantages of machine readable semantic descriptions. We do not provide an ontology to describe domains, rather we leverage existing ontologies like the SSN to add meta-information to the data and create a domain model. As stated before, we rely on

[1]https://www.w3.org/2005/Incubator/ssn/ssnx/ssn (accessed on 04/10/2021)

a message broker for data integration and automatically generate adapters based on a semantic description of the data source.

Data integration can also be automated by model-based approaches, like in [Hufnagel and Vogel-Heuser 2015]. Adapters connect all IT systems to a data backbone, where the data is stored in a harmonized form. Users can integrate different data sources by modeling the harmonization steps in a Unified Modeling Language (UML)-like fashion and adapters are generated based on this model for the specific systems. Another model-based approach is presented in [Pauker et al. 2016], where the modeling of the virtual representation of machines is simplified. UML can be used to model the domain, which is then transformed into other models like OPC UA, MTConnect, or even programming languages (e.g. .NET). For each of those model transformations, the meta-model with corresponding mapping rules is used. This approach reduces the technical complexity for data integration, but users need to understand the UML syntax and explicitly create a model, sometimes even some code must be implemented manually. In our approach, we infer the model in the background by interaction of the user with a GUI. Additionally, we try to support the user during the modeling process by leveraging meta-information of the data source, when available. No explicit modeling is required. Furthermore, we provide meta-information about the data source that can be used for additional processing.

In this section, several approaches for systems that provide analytics on industrial event streams were introduced. It was shown that those systems consist of multiple components / layers (data acquisition, data processing, data analytics) that are required to analyze data from multiple heterogeneous industrial assets. All of those systems introduce a lot of technical complexity that must be processed by users. These areas of connecting, processing, and analyzing data will be examined in more detail in the following sections, as they need to be solved in order to provide an overall solution for citizen technologists.

4.2. Connect & Model IIoT Data Sources

To ingest data into an analytics systems, different physical assets must be connected. In this section first, several standards and protocols for IIoT systems are introduced. Then, approaches for semantic data modeling in the context of IIoT systems are shown. At the end of this section, several open source solutions are derived that are used for connection and modeling of IIoT data sources.

Nowadays, there is no universal standard that defines how all industrial assets communicate [Gosewehr et al. 2017; Berger et al. 2017]. However, several different standardization efforts exist, which are presented in the following:

OPC UA is a platform-independent protocol for IIoT communication, which is standardized in the IEC 62541 series [Commission et al. 2016]. It does not only describe how machines can exchange data, it is further possible to provide a semantic description for the data. Architecture-wise, it uses a Service-Oriented Architecture (SOA) consisting of multiple layers. "OPC UA does not only deal with simple data access, but also with a wide range of other aspects e.g. security, reliability, access control, alarming, or historical data" [Schleipen et al. 2016]. OPC UA is an integration standard enabling the cooperations of multiple stakeholders from different domains. Multiple joint working groups and companion specifications exist, which can be found on their website[2]. One such a companion specification is the universal machine technology interface (umati)[3]. It is developed by a community of members from the machine construction industry together with their customers and provides a standardized open interface based on OPC UA. Our aim is to leverage knowledge provided by the rich data-model of such information models. However, not all machines provide such an interface. In contrast to consumer hardware, machines in the production environment have a very long lifespan, which is partly due to the often immense investment costs. This means in particular that solutions must be applied in brownfield deployments, containing legacy systems, adding additional complexity [Boyes et al. 2018].

MTConnect [MTConnect 2021] is a data and information exchange standard that also defines a semantic data model for the manufacturing domain. Adapters are used to communicate with pieces of equipment and provide information to the MTConnect agent. Agents are responsible for handling the data in a structured way and provide it to clients (e.g. applications) that use the data. Data is exchanged as XML via HTTP as a transport protocol. Two main communication methods are provided request/response and publish-/subscribe. A companion specification for OPC UA also exists for MTConnect that defines mappings between the two standards [Foundation and Institute 2019]. The authors of [Liu et al. 2019] show, how a combination of OPC UA and MTConnect can look like as a standardized efficient data exchange between machines and tools. They state that "MTConnect provides a concrete information modeling method specifically designed for CNC machine tools with some predefined data structures and rules" [Liu et al. 2019]. However, "OPC UA offers a more generic information modeling method in order to cover a broader range of industrial equipment and systems" [Liu et al. 2019].

Another approach to standardize communication between machines on the shop-floor is the Asset Administration Shell (AAS) [Adolphs et al. 2016]. It provides a unified description for assets introduced by the initiative "Platform Industrie 4.0"[4]. It defines a unified wrapper around assets, its representation, and technical functionality. In [Tantik and Anderl 2017], a realization of the concept is introduced. Several options to serialize the models from the AAS exist (e.g. XML, JSON, RDF, OPC UA data model and AutomationML) [Abburu et al. 2020]. In our approach, the data source does not have to conform to a fixed standard. We do not necessarily need a wrapper like the AAS around the data source. However, if such a model exists, we try to leverage the provided

[2]https://opcfoundation.org/ (accessed on 04/10/2021)
[3]https://umati.org/ (accessed on 04/10/2021)
[4]https://www.plattform-i40.de/ (accessed on 04/10/2021)

meta-data to complete the configurations automatically and ease the adaption process. With Eclipse BaSyx3, an open source implementation based on the concepts of AAS is available [Abburu et al. 2020]. It provides common industry 4.0 components, as well as a SDK for a fast development. In our approach, we also provide a SDK, to allow the integration of new adapters in case further requirements arise.

The World Wide Web Consortium (W3C)[5] provides with the W3C Web of Things (WoT) [6] a specification that has the goal "to improve the interoperability and usability of the Internet of Things (IoT)". It consists of multiple building blocks, like the WoT Thing Description, as well as an abstract architecture. The WoT Things Description (TD) specification provides a meta-model with a "thing" at its core. A thing is described as "an abstraction of a physical or a virtual entity whose metadata and interfaces are described by a WoT TD"[7]. The TD does not define an API how to access "things", it rather requires one (or even more) description of existing API endpoints [Abburu et al. 2020]. The goal of this thesis is not to define a new standard that competes with the previously introduced standards, but focuses on industrial event stream analytics, where it is necessary to connect to data sources. We leverage existing and widely adopted standards to ease the ingestion process for citizen technologists.

Semantic descriptions play a key role in the previously introduced data models. Next, we introduce several approaches that use semantic technologies for data integration and harmonization. WInte.r [Lehmberg et al. 2017] for example supports standard data formats, like CSV, XML, or RDF. Data sets with different schemas can be merged, as well as units can be harmonized. We use similar techniques, but focus on streaming data of industrial assets instead of data sets. In Spitfire [Pfisterer et al. 2011], the goal was to develop a Semantic Web of Things. It focused on REST-like sensor interfaces, leveraging semantic technologies for interoperability. In our work, we concentrate on industrial protocols with high-frequency event streams that require local preprocessing. We use semantic descriptions for meta-information describing the streams. In the Big IoT API [Bröring et al. 2018], the focus lies on the interoperability between IoT platforms. In contrast, we focus on citizen developers to connect data, especially from IIoT data sources.

So far, standards and semantic models for IIoT systems and data modeling were introduced. In the following, several open source solutions are presented that can be used to integrate and model data. Initially, there are libraries that directly communicate with PLCs in machines. This used to be a very complex technical problem, because the APIs of manufacturers are not compatible and, therefore, individual implementations are always necessary. With Apache PLC4X[8], a solution was developed that solves this challenge. It provides a unified API for PLCs, similar to what JDBC does for databases. This makes it

[5]https://www.w3.org/ (accessed on 04/10/2021)
[6]https://www.w3.org/TR/wot-architecture/ (accessed on 04/10/2021)
[7]https://www.w3.org/TR/wot-thing-description/#thing (accessed on 04/10/2021)
[8]https://plc4x.apache.org/ (accessed on 04/10/2021)

possible to write reusable, vendor-independent code. We leverage such tools to enable an easy configurable and reusable approach. Even if such libraries greatly reduce the technical complexity, a software developer is still necessary to connect data. Therefore, we use such solutions for the technical connections, but in a way that end users only need to provide some initial configuration parameters.

With Eclipse Vorto[9], it is possible to describe the capabilities and functionalities of devices. The description of the devices can be defined using a Domain Specific Language (DSL). Based on this description, code can be automatically generated in a device agnostic way. An open online repository is provided that already contains several models for devices. In [Laverman et al. 2016] it is shown, how Eclipse Vorto can be used to integrate vehicle data into a smart home IoT system. In contrast to our work, Eclipse Vorto focuses on the model for the devices, which is described in the DSL. We try to infer the model based on runtime data and automatically create a data adapter. If a data source already provides a rich model like Vorto, we could leverage this meta-information to ease the configuration effort for users.

In this section, we presented several standards and approaches on how to model and connect IIoT data sources. Most of these approaches still require a certain level of technical expertise. In our method, we provide a guided workflow to support the user to connect new data sources by creating a model and automatically instantiate the adapters.

4.3. Data Processing & Reduction

Once data sources are connected, the data they produce must be processed and evaluated by algorithms and services. In the previous sections, it was shown that message brokers are generally used as a unified data backbone in distributed IIoT systems. A distributed based message broker ensures that the data is available for all services requiring it. In this part, approaches for different message broker systems are introduced first. Next, methods from Wireless Sensor Networks (WSN) are presented, which are capable of dynamically reducing the information before transmission. Finally, the influence of serialization on transmission performance is discussed.

There are different types of message brokers. In this work, we focus on content-based and topic-based publish-/subscribe systems. We see these as the most promising solutions for IIoT scenarios, because it is possible to subscribe to data of individual or multiple assets. In content-based publish-/subscribe systems, subscribers can define individual restrictions and the message broker ensures the routing of the messages that fulfill these restrictions. This could, for example, be used for an analytics algorithm, which is only interested in specific data produced by some assets (e.g. of critical situations). A popularity-based routing approach to optimize the data flow within the broker is

[9]https://www.eclipse.org/vorto/ (accessed on 04/10/2021)

proposed in [Salehi et al. 2017]. The authors of [Bhowmik et al. 2016] propose a different approach, using Software Defined Networking (SDN) techniques to reduce events on network devices. This makes it possible to filter events earlier, before they are transmitted to the subscriber, and reduces network load. In our approach, we also try to filter events early on edge devices close to the source. But we rely on topic-based brokers, as the analytics methods do mainly not rely on single contents of events. Usually all events of a certain asset are required.

In [Sommer et al. 2018], four broker technologies are compared, three of which are topic-based: Message Queuing Telemetry Transport (MQTT), AMQP, Apache Kafka. Their evaluation shows that all of the systems have a "throughput of more then 1000 mps [Messages per Second] for all tested payload sizes, which is relevant for industrial applications, as most control cycle times are between 1 and 10 ms" [Sommer et al. 2018]. Based on the technical and security capabilities of the examined message brokers, the authors argue that all of the systems can be used in industrial settings. However, the brokers have many differences and must be selected according to the specific use cases. In this work, we decided to introduce a wrapper around existing topic-based message brokers, to leverage the already available functionalities and extend them with dynamic reduction strategies. Latency is a big influencing factor in the evaluation of IIoT event streams. This plays a key role especially in distributed systems, because events have to be transmitted over a network. To reduce this influence, different proximity-based middleware technologies were developed to optimize the routing of events that use local compute power on edge resources. PubSubCoord [An et al. 2017] is such a solution, which supports scenarios where multiple Local Area Networks (LANs) are connected over a Wide Area Network (WAN). Each local network contains an edge broker that is linked to a routing broker. This routing broker acts as a mediator and routes data between the edge brokers. The goal is to reduce the latency for the clients by optimizing the routes. Another solution is EMMA [Rausch et al. 2018b], a QoS aware middleware based on MQTT. It consists of four components (gateway, broker, controller, monitoring protocol): First, gateways are used as buffers between the clients and the broker. With those *gateways*, it is possible to switch the broker during runtime. Second, there are *brokers* that implement the MQTT server protocol that also act as a topic bridge to forward events to other brokers. Third, there is the *controller* orchestrating the whole system and fourth, a *monitoring protocol* that monitors the network QoS in the distributed environment. With this solution, it is possible to dynamically adapt to changing conditions in the system. The concept of dynamic adaptation are inspired by principles of the field of osmotic computing [Rausch et al. 2018a]. In cloud-based systems, the configuration is usually set on deployment time. In [Rausch et al. 2018a] it is argued that edge-based brokers must be able to react to changes dynamically, since the network or clients in the system might change during runtime. They introduce the term osmotic pressure that is "calculated based on the amount of clients and their proximity to the resource" [Rausch et al. 2018a]. The goal is to share information between the clients as fast as possible, therefore, they introduce an indicator for proximity based on the network latency between clients. All of those concepts and message brokers focus on how to optimally route data between the clients. It is agnostic to the payload of events and it does not optimize what is transmitted, because no semantic understanding of the data is available. In this thesis, the content of the event schema is semantically described and this knowledge is used to dynamically reduce the payload. Since our solution is designed as a wrapper around topic-based approaches, it can be combined with the approaches from this field.

In Wireless Sensor Networks (WSNs), approaches exist that attempt to optimize data transmission according to the information represented in the data. WSN consist of distributed sensing devices connected via a wireless network. Often, these devices have limited resources and power, which means they must optimize the transmission of sensed information to reduce power consumption. Two approaches where individual sensors decide whether they transmit data or not are proposed in [Jiang et al. 2011] and [Harth and Anagnostopoulos 2017]. Receivers are capable of reconstructing the measurements based on previously transmitted data, although they are not actually being sent. Computing the data locally results in lower energy consumption versus sending the raw data to a central node. We use similar techniques and apply them on industrial event streams transmitted over message brokers. In addition, the subscribers are capable of dynamically changing the data quality. Most of the previously discussed approaches are independent of serialization, but this can have a significant impact on performance, as described in [Maeda 2012]. There are several techniques for performing specific compression algorithms, e.g., using shared dictionaries to compress events in publish/subscribe systems [Doblander et al. 2016]. Our approach focuses on reducing the content of event streams. Serialization and compression techniques are also combined to further reduce network load, but are not particularly part of this thesis.

This section discussed related work in the field of data processing and reduction of industrial event streams. Existing solutions do not leverage the semantics of the events or the characteristics of industrial event streams (e.g. high sampling rate and some values do not change too often) to reduce the amount of transmitted data. Often, no knowledge about the analytics algorithms is available that processes the data. With this information, dynamic reduction strategies can be applied on the event streams to automatically adapt them according to the requirements of publishers and subscribers.

4.4. Self-Service Data Analytics

In this section, related work in the field of data analytics and self-service data analytics is discussed. First, state-of-the art analytics approaches are presented, which often provide a programmable interface to define analytics tasks. Then, different approaches that try to reduce required technical knowledge are introduced, using DSLs to define the analytics task. In the end of this section, we describe different approaches from the field of AutoML that try to automatically search for the best configuration parameters for Machine Learning (ML) models.

Analyzing IIoT data with ML-based approaches can help to improve the productivity and reduce maintenance costs in manufacturing environments. For example, machines can automatically be monitored and early warnings can be created based on generated predictions. This is shown in [Kanawaday and Sane 2017], where a forecast with ARIMA is performed. Based on this forecast, a classifier is trained that predicts the quality of a part that is currently produced. Solutions are often tailored for individual use cases with a specific model at its core. Therefore, approaches and tools are required that enable the quick definition and training of ML models on historic data. Multiple open source analytics approaches exist to assist with the creation of ML models. Often, they were developed in academia, an excerpt of these is investigated in the following study [Landset et al. 2015], where the advantages and disadvantages for individual solutions are outlined. Additionally, commercial approaches from cloud vendors like Microsoft Azure[10] or Amazon AWS[11] exist, providing the ability to scale and integrate diverse third party solutions and sometimes even provide one click deployments. The main focus of such tools is on creation of ML models and manually implementing aspects of the software development lifecycle. They are technically too complex for citizen technologists with little to no data science experience, because all the configurations (e.g. model parameters) must be provided manually.

Once a new model is trained, the challenge is to deploy this model on new data (e.g. event streams). A common approach is to train the model offline with historic data and then apply it in a streaming environment. Therefore, the model parameters must be serialized and loaded into the streaming engine. To do this in a technology agnostic way, a standard like PMML [Guazzelli et al. 2009] can be used. After training, the model is written into a PMML file, which is then loaded into a scoring engine [Chaves et al. 2006] and applied on new data instances. Since the training phase and the usage of the model are separate, it is not possible to adequately react to changes in the data. To make this possible, one can use algorithms from the field of data stream mining, where the main algorithms are focused on classification, regression, clustering, and frequent pattern mining [Bifet et al. 2018]. Online training methods are used that constantly update the models. Different frameworks exist in this area, e.g. MOA [Bifet et al. 2010] or SAMOA [Morales and Bifet 2015] that provide an interface to define tasks for stream ML. New labels are constantly required in order to update the models online. However, it cannot be assumed that these labels will always exist, therefore, these often have to be created manually for industrial event streams. For all the solutions presented above, a high level of technical understanding is required in order to use them. Often, programming

[10]https://azure.microsoft.com/en-us/services/machine-learning/automatedml/ (accessed on 04/10/2021)

[11]https://aws.amazon.com/de/sagemaker/autopilot/ (accessed on 04/10/2021)

languages must be used directly to preprocess data and define the configuration for the ML task.

To lower this technical hurdle, DSLs were developed that can be used in a self-service fashion. Thereby citizen technologists are enabled to analyze data without implementing the solution in a programming language, rather defining the task in some DSL. This is already often used in Business Intelligence (BI) systems and the survey in [Michalczyk et al. 2020] provides an overview of state-of-the-art self-service BI and analytics tools. In the survey, the authors distinguish between the technical experience of the different kinds of users, all having the main focus on analyzing data. A concrete example is given in [Plazas et al. 2020], where a self-service BI approach with IoT data in the context of industry 4.0 is presented. BI experts and IoT experts work together and a UML model is provided for each of them. With those UML models, the authors try to improve the communication of the different roles involved in the complex task of data analytics. The process consists of six steps, starting with requirements of a business user and resulting in a final BI solution. Here, the focus is on creating database queries and not on the training of machine learning models.

DSLs can also be used to describe ML tasks to train models. As shown in the survey [Portugal et al. 2016], a distinction is made between textual and graphical approaches. BiDaML [Khalajzadeh et al. 2019] introduces a suite of visual languages to support end users in the data analytics process. They tackle the problem of communication between individual roles by providing five different diagrammatic notations. The main focus is on data scientists and software engineers, but the diagrams allow them to communicate with business owners and business analysts as well. It is described as a novel integrated suite of visual languages that supports end users in designing analytics solutions. The approach is similar to UML for the software development process. However, such solutions still require data scientists to build the models with a deep technical understanding about data analytics. Another approach is introduced in [Mahapatra 2019] where a graphical flow-based programming approach is presented. Source code is generated based on these data flow descriptions, and then executed in a (distributed) processing environment. The descriptions can also be used to define the training task of ML models, as well as the deployment of those models on new data. This reduces the required technical knowledge, especially because code for distributed systems is automatically generated based on the defined graphical flow. But ML models must still be configured manually for the training. Even if way less technological and software development know-how is required for the presented approaches, it is still necessary to provide the learning parameters to train a ML model. In order to reduce this effort, we will next discuss methods from the field of AutoML.

AutoML can be used to automatically find the best parameters for an ML task given a data set. Several approaches for batch data exist, however, there is no wholistic approach that enables citizen technologists to analyze industrial event streams. A challenge for event streams is how to deal with changes in data. The authors of [Celik and Vanschoren 2020] discuss several strategies how to handle such changes. They discuss different AutoML frameworks and state that those do not cover online learning algorithms. We decided to build a wrapper around existing frameworks to be able to reuse strategies for the automated search of a data science pipeline (e.g. model selection) and extend them with a parameter search required for industrial event streams. In [Madrid et al. 2019], the authors apply AutoML in the presence of drift. An extension of auto-sklearn, combined with a drift detector is proposed that automatically triggers a retrain of the model, once a drift is detected. In our work, we also use a drift detector when the model is applied on event streams, to notify a citizen data scientist that data has changed. We do not assume that there is always labeled data, so the citizen data scientist can decide if the model should be retrained.

AutoML on event streams is an active research topic. Automations for different learning algorithms are developed, but they are not yet integrated into a holistic framework, as realized for batch learning approaches. In [Bahri et al. 2020], a stream k-nearest neighbors (kNN) algorithm is introduced that performs an internal dimension reduction to reduce the resource usage. It monitors the system and dynamically tunes the parameters of the kNN algorithm in a streaming fashion. Our work leverages the variety of existing AutoML approaches and creates a wrapper to apply them on time-series data. There are already many different batch training algorithms integrated into AutoML frameworks and can therefore be used. This is especially helpful since different algorithms may perform differently on individual data. Mcfly [Kuppevelt et al. 2020], is a python package that provides a deep learning AutoML approach for time-series data. It delivers an automated parameter selection method for neural networks, but only supports classification-based approaches. Further, it requires manual programming to transform data into the right format and define a size for the sliding window. The presented approaches are designed to automate a specific task. Our goal is to provide a wholistic conceptual solution that supports citizen technologists during the whole data analytics cycle from connecting new sources, to deploying the trained ML model without writing code.

This section first introduced different analytics solutions, then related work that uses DSLs to reduce the technical complexity of the analytics tasks were presented. Followed by several approaches from the field of AutoML to show how the parameters for the ML task can be set automatically.

4.5. Conclusion

In this chapter, we presented related work in the field of industrial event stream analytics for citizen technologists. First, several approaches for architectures and systems were presented that enable the usage of data produced by machines. In many of these architectures it can be seen that data connection is mainly done manually. A central message broker is used to share the data between the different distributed services. Afterwards, state-of-the-art from the fields of data connection and model creation was discussed. Especially, the connection and modeling of data is often still a very technical task that requires knowledge of the used standards and protocols. Subsequently, processing and reduction of data using broker technologies was described. Message brokers are already widely used in IIoT systems, but current solutions lack the ability to dynamically adjust to changes of the subscribers by leveraging the semantics of the events to reduce the transmitted data. Finally, related work from the field of self-service analytics was presented. It is shown how ML can be used to perform those analytics tasks and different approaches to reduce the technical complexity are discussed. All this describes the current state-of-the-art on how to process machine data.

Part II.

Main Part

5 Overview

This chapter provides an overview of the main part of this thesis. In the beginning, a high-level structure of the three main phases is given, which is followed by a more detailed walkthrough of the individual components, to show how industrial assets are connected and industrial event streams are analyzed.

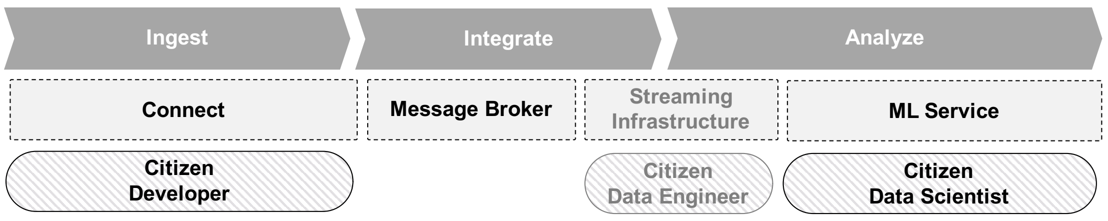


Figure 5.1.: Industrial event stream analytics

Figure 5.1 depicts an overview of the structure of this work and how industrial event stream analytics can be realized by citizen technologists. On the top, the three main phases, *ingest*, *integrate*, and *analyze* are illustrated. Each of these phases will be discussed individually in the next chapters. Chapter 6 (Ingest) focuses on citizen developers and how they are enabled to *connect* data from physical assets like machines and sensors. To achieve this, a user models the data sources, based on which the adapters that create industrial event streams are automatically generated. Chapter 7 (Integrate) focuses on how data of multiple sources can be harmonized in a *message broker*. Further, the event streams are automatically adapted based on the previously created model and requirements of the data subscribers. The process is fully automated, so no user role directly interacts with the message broker. Chapter 8 (Analyze) describes the last phase, where a *ML service* is introduced that is capable of training and deploying supervised machine learning models. This service focuses on citizen data scientists, by enabling them to define a desired training configuration and train the model automatically. Since we focus on event streams, a *streaming infrastructure* is required that is capable of executing pipelines. This is used for both, preparing data for the ML service and deploying the trained models. Citizen data engineers mainly interact with this service, however, in this work, we reuse existing streaming approaches, which is why we do not focus on this role.

Next, we will give a more detailed overview and show how the individual components interact with each other.

Figure 5.2 presents an overview of how the data is processed. The industrial assets ❶ are located at the shop floor and provide data via interfaces. The figure depicts machines from the production line of the example in Section 3.2. In the first step, data is ingested and prepared to be available for further analytics, using the connect framework.

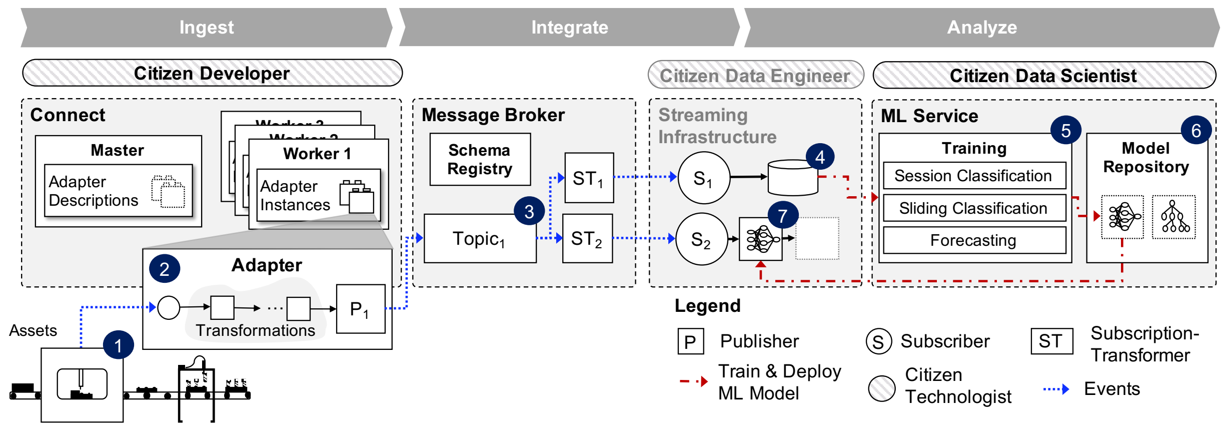


Figure 5.2.: End-to-end view of the developed approach

On the left are the different components of the connect framework. It consists of a *master*, containing all the *adapter descriptions* of the semantic adapter model. Furthermore, it is also responsible for managing all the *workers*. Within the workers, the *adapter instances* are running. These instances are directly connected to the assets, from where they read data. To instantiate an adapter instance, a user (citizen developer) has to provide a semantic description based on the adapter model. To simplify this process, user support is given that leverages already existing meta-information of the endpoints. Such an adapter instance representation can be seen at ❷. The first component is responsible for connecting the data source, reading the data, interpreting it, and translating it into a harmonized format. Then, *transformations* are applied to preprocess events (e.g. convert units) before they are transmitted with a *publisher* P_1 to a *topic* in the *message broker* ❸. This message broker is used to integrate data from multiple sources. The main goal is to dynamically adapt the event streams to reduce the load on the network. For this, reduction strategies on event streams are needed, which can be applied automatically. Those strategies run within the *publisher* P_1, as well as within the *subscription-transformer* (ST_1, ST_2) to prepare the data individually for each *subscriber* (S_1, S_2). Industrial event streams are usually processed in a streaming infrastructure and the data can for example be stored in a time-series database ❹. In the *analyze* phase, data stored in the time-series database is used as the foundation of the *training service* ❺. This service supports three different kinds of machine learning problems, *forecasting*, *sliding window classification*, and *session window classification*. Citizen data scientists are provided with a guided workflow to specify the training task, which is then performed automatically. Once the training is complete, the resulting models are stored in the *model repository* ❻, where they can be inspected by the citizen data scientist. The best model can then be used by a citizen data engineer in event processing pipelines ❼ to analyze event streams.

In the following chapters, the individual contributions of this thesis are explained in detail. At the beginning of each chapter, a walkthrough section refers to this graphic and explains the individual components in more detail. In order to be able to use the components in a modular way, they were designed in a microservice fashion. Individual services are provided via container technologies, as these are particularly well suited in edge processing scenarios [Ismail et al. 2015; Rehman et al. 2019].

6 Ingest

This chapter deals with the ingestion of data and is the first of the three main research parts of this thesis. It tackles the question how citizen developers can be supported in modeling and connecting heterogeneous industrial data sources. To cope with this technical complex challenge, an adapter model is introduced which can be used to instantiate adapters automatically. Furthermore, data quality is relevant for good analytics results, so it is necessary to preprocess and harmonize data. Overall, this task should be performed by users with little technical knowledge, which is why a good user support for the individual steps is necessary.

The chapter is structured as follows, first a walkthrough is provided in Section 6.1, to give a high-level overview. Then, some terms & definitions are introduced in Section 6.2. It discusses how the event model looks like, further it introduces virtual sensors and the concept for adapters. In Section 6.3, the model for adapters and transformation rules is introduced and how this model can be used to connect and transform data of industrial assets. After that, Section 6.4 shows the advantages of the model and how the system leverages the different characteristics to support users when connecting new data sources. At the end of this chapter a full example is presented in Section 6.5 that illustrates how the introduced concepts can be applied to ingest data of a production line before the chapter is concluded in Section 6.6.

6.1. Walkthrough

The main goal of this chapter is to connect different industrial assets and make its data processable. Therefore, a citizen developer can create a model of the data source (adapter description), which is then used to automatically instantiate adapters. Those adapters are deployed using a master/worker paradigm, where the master manages the workers. The adapter instances are then launched on workers. It is also taken into account that it is possible to harmonize the data directly on ingestion by applying transformations on the data before forwarding it onto a harmonized message broker.

Figure 6.1 depicts the ingest part from the overview of the previous section (see Figure 5.1). It presents the different components used to ingest data of the industrial assets

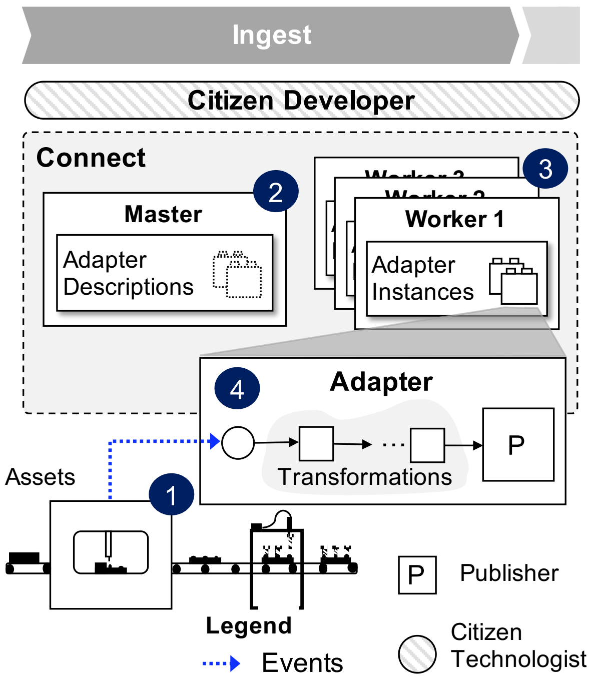


Figure 6.1.: Overview of components in the ingest chapter

(e.g. the production line ❶ from our motivation). Those assets provide sensor values via interfaces. The blue dotted line illustrates the event stream from the source over an adapter instance, running in a worker. At ❷, the connect master can be seen, that manages all workers, adapter instances, as well as the data models (e.g. adapter descriptions). It further provides an interface for citizen developers to interact with the system. Adapter instances run within ❸ worker nodes, which are usually deployed on edge nodes near the data sources. Multiple of those workers can be deployed, each of which is registered at the master. Every worker can contain multiple adapter instances, which process the data. The reason for this design decision was to connect and preprocess data in close proximity to the data source. A single adapter ❹ is responsible for connecting to the data source and perform preprocessing functions on the data before it is forwarded onto a message broker.

6.2. Terms & Definitions

In this section data models are presented, which serve as a basis for the concepts in the further course of this thesis. To describe events, an event model is introduced which describes the event schema consisting of multiple individual event properties. Based on this event schema, the similarity between two data sources can be described, with the concept of virtual sensors. In the end of this section adapters are introduced and the differences between generic and specific adapters are defined.

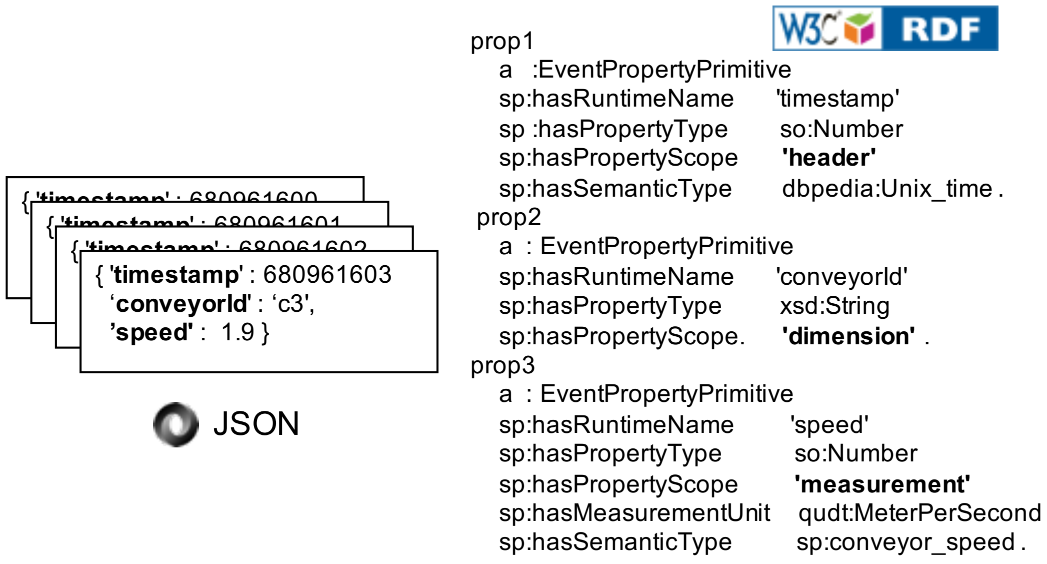
6.2.1. Event Model

Events are a basic concept used throughout this thesis. The foundation of events was introduced in Section 2.2. In this section, the characteristics of events that are especially relevant for industrial event streams are described. The event model distinguishes between event instances, produced by assets containing the actual data and the event description containing meta-information. The event schema is described in a machine readable format, the Resource Description Framework (RDF)[1], representing context information to the observed values [Riemer 2016]. This description does not change very often over time and can contain further (domain specific) knowledge.

Figure 6.2 depicts a speed event produced by the conveyor belt from the example production line of our motivating scenario. On the left, the stream of speed events is shown, serialized in the JavaScript Object Notation (JSON) format containing three values *timestamp*, *conveyorId*, and *speed*. On the right, the description for each event property of the event schema is presented. For better clarity, it only contains a small part of the model. RDF was chosen because of its flexibility and possibility to add more domain specific knowledge. Each description, contains the runtime name, as well as the property type for each property. The timestamp additionally has a semantic type that specifies it to be a unix timestamp. For the speed property the unit is defined as meter per second and a semantic type is set that specifies the value as the speed of a conveyor. In this example, only a flat hierarchy is shown. However, it is also possible for events to contain nested structures as well as lists.

The structure of an event as defined in [Etzion and Niblett 2010] has three parts. Header properties, payload properties, and an open content. We only use header and payload properties, because we expect that the event produced by one data source has a fixed schema with predefined properties. Additionally, in this work payload properties are split up into *dimension* and *measurement* properties. This information can be specified via the *property scope* for each event property. Dimension properties contain identifiers (usually of real world assets) that can be used to partition or group an event stream. Measurements contain the actual data that usually changes over time and should be observed by algorithms. Figure 6.2 shows an example for each of the property scopes. The timestamp is a *header* property, representing the occurrence time of the event. The conveyor id is a *dimension* property, that can be used for example to calculate the average of *measurement* properties (e.g. speed) over a certain time for each conveyor. This information helps to group event streams in a platform independent way.

[1]https://www.w3.org/RDF/ (accessed on 04/10/2021)



```
{ 'timestamp' : 680961603
  'conveyorId' : 'c3',
  'speed' :  1.9 }
```

```
prop1
   a   :EventPropertyPrimitive
   sp:hasRuntimeName       'timestamp'
   sp :hasPropertyType      so:Number
   sp:hasPropertyScope      'header'
   sp:hasSemanticType       dbpedia:Unix_time .
 prop2
   a  : EventPropertyPrimitive
   sp:hasRuntimeName       'conveyorId'
   sp:hasPropertyType       xsd:String
   sp:hasPropertyScope.     'dimension' .
prop3
   a  : EventPropertyPrimitive
   sp:hasRuntimeName         'speed'
   sp:hasPropertyType         so:Number
   sp:hasPropertyScope        'measurement'
   sp:hasMeasurementUnit    qudt:MeterPerSecond
   sp:hasSemanticType         sp:conveyor_speed .
```

Figure 6.2.: Left: speed events. Right: metadata description of speed events

6.2.2. Virtual Sensor

Different event streams have an independent event schema and so far we have not defined how similar two event streams are, or whether they represent the same information. This is particularly relevant for data produced by different assets, for example when there are two machines of the same type from different vendors. Data produced by those machines can contain similar information, but the representation might differ. In order to unify such event streams and analyze them with the same components, the event schema must be comparable according to its meaning. This helps to achieve the goal of a better interoperability, because not only the representation, but also the semantics is taken into account. *Virtual Sensors*, presented in this section, are a representation of a particular event scheme.

Figure 6.3 shows an example of a virtual sensor measuring the conveyor speed. This sensor can be used by all conveyor belts that have the capability to measure the current speed. For all event streams that have an event schema with all these properties, the predicate *hasVirtualSensor* is added. This information can be used to suggest only suitable analysis algorithms to users (e.g. monitor the speed of a conveyor for predictive maintenance). To ensure that an event schema represents a virtual sensor, all individual event properties must be present in the schema. Therefore, the equality of properties is checked. In order to perform this check, the following conditions must be met:

- *ep1.runtimeType* **equals** *ep2.runtimeType*
- *ep1.hasUnit* **equals** *ep2.hasUnit*
- *ep1.semanticType* **equals** *ep2.semanticType*

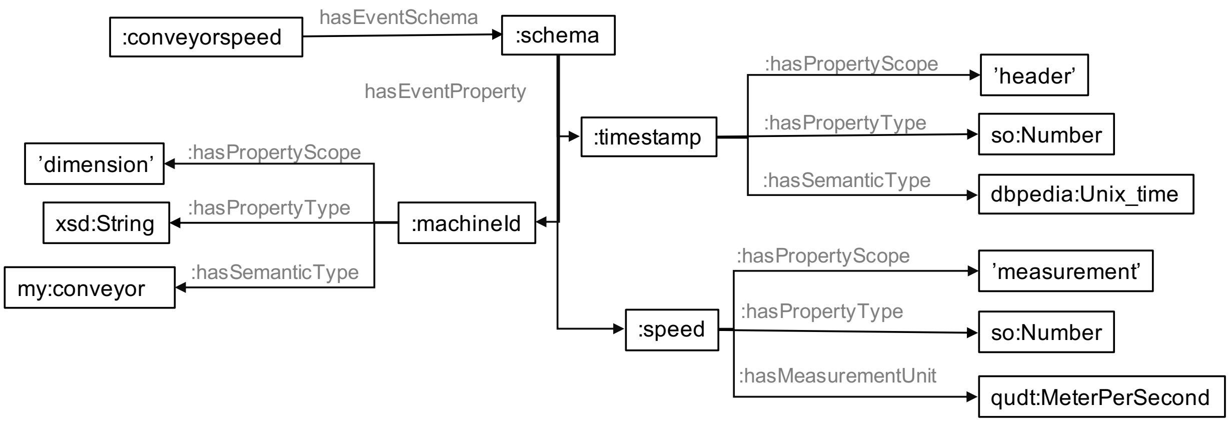


Figure 6.3.: Example model of a conveyor speed event stream

Virtual sensors can also have relations to other virtual sensors. If all properties of a sensor are subsumed by another sensor, the predicate *hasVirtualSensor* is added. For example *:vs2* contains all the properties that *:vs1* has, according to the previously shown rules, then there is a relation *:vs1 :hasVirtualSensor :vs2* between the two virtual sensors. This means in particular, if a component requires *:vs1*, then *:vs2* also fulfills this requirement. With that relation the backward compatibility for algorithms is ensured, for example when newer sensor models differ in terms of provided values. In this work we use this concept for the trained machine learning models. The selected event properties during training are used in the definition of the virtual sensor, to ensure they contain the relevant data for the feature extraction of the machine learning model.

6.2.3. Adapters

As already shown in Section 4.1, adapters are usually used in Industrial Internet of Things (IIoT) systems to ingest data onto a unified middleware. This provides a decoupling mechanism between the heterogeneous data sources and the downstream analytics algorithms, helping to reduce complexity. Adapters can be used to connect and harmonize different, formats, protocols, and data representations (e.g. different units). Therefore, the data has to be transformed to an internally used standard. However, it is not only the form of the events that needs to be adjusted, it is also important to adjust and possibly normalize the data itself. To do this, we present a model for adapters including transformation rules.

For the adapters we distinguish between two different types, *generic adapters* and *specific adapters*. *Generic adapters* are defined for protocols, independent of the format of transmitted events. A generic adapter is able to read data from a generic protocol regardless of the format representation of transmitted events. For many protocols (e.g. Hypertext Transfer Protocol (HTTP), Message Queuing Telemetry Transport (MQTT)) this format is not fixed, therefore it must be specified how to read data from a protocol and how to deserialize the events. *Specific adapters* are defined for specific data sources or types of data sources. This can range from standards using a fixed (proprietary) format to specialized data sources, requiring custom implementations. Both adapter types have in common that it is possible to define individual configuration parameters. This is explained in more detail in the next section.

6.3. Connect

This section presents how streaming data from multiple different industrial data sources can be connected to perform IIoT analytics. At the core of this section is the *adapter model* that is used to describe time-series data sources. Based on that model adapter instances are automatically generated and executed on edge devices in close proximity to the data source. The goal of the model is to enable citizen developers to ingest data of different sources themselves, by providing configuration parameters. In this section, first the adapter model is introduced and it is shown how it can be used to describe different data sources. Then the model for transforming and harmonizing data directly on the ingestion in the adapter is shown. In the last part, the transformation rules that are executed during runtime on the events in the adapter instance are presented.

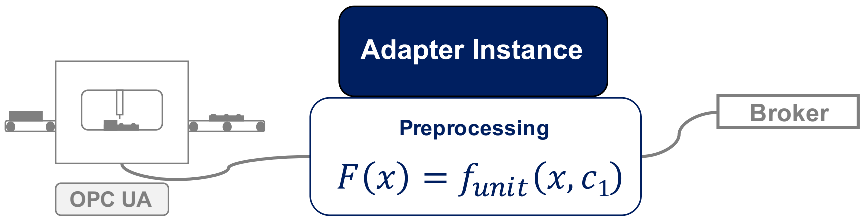


Figure 6.4.: Adapter instance connected to the CNC machine, preprocessing events x and publishing data to the message broker

Figure 6.4 shows an example adapter instance that is connected to the Open Platform Communication Unified Architecture (OPC UA) interface of the CNC machine from our example in Section 3.2. First, data is collected from the machine and preprocessing functions are applied on the events for harmonization. In this case the original unit of the temperature value is represented in degree Fahrenheit and should be transformed into degree Celsius to be comparable with temperature values of other machines. After the preprocessing the events are forwarded to the massage broker. For the description of the adapter model RDF is used because it is possible to reuse already existing vocabularies and extend them. Especially for preprocessing it is helpful to reuse vocabularies and ontologies to leverage already formalized and machine-readable knowledge. Models can be serialized to JavaScript Object Notation for Linked Data (JSON-LD) as an exchange format between different microservices.

6.3.1. Adapter Model

Figure 6.5 shows the semantic adapter model. At its core is the *Adapter* concept, highlighted in grey. On the left is the *StreamGrounding* and *TransformationRules*. Once events are processed they are sent onto a unified message broker, which is described in the *StreamGrounding*. It describes the target protocol and format used by the running adapters. The StreamGrounding can potentially be different for each adapter, but in most cases it will be the same for all adapter instances to provide a unified access to the data for downstream algorithms. *TransformationRules* describe how events from the source should be transformed before forwarding them onto the grounding. Those rules are explained in more detail in Section 6.3.2.

In the model, it is distinguished between *DataStreamAdapters* and *DataSetAdapters* to support both *DataStreams* and *DataSets*. In this work we use data streams as a synonyme for event streams. Both have a fixed *EventSchema* consisting of a set of event properties [Riemer 2016]. For a better overview, 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 point of view, there is no difference between the two types, they are only distinguished during runtime. Data sets are bounded data streams, this means they stop at some point while a stream never ends. Since the modeling process is the same for both types, from now on we will always refer to data streams.

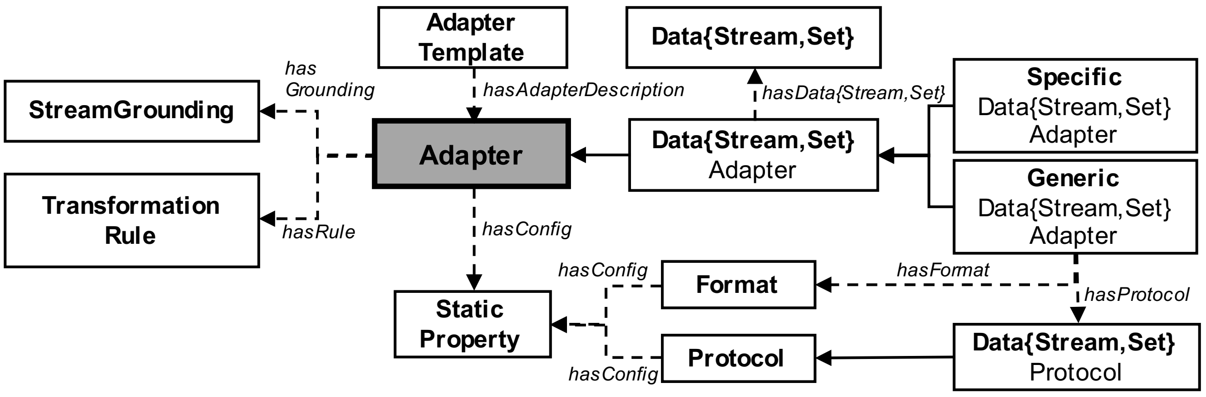


Figure 6.5.: Core of our adapter model

As defined in the previous section, there are two types of Data Stream Adapters, *GenericDataStreamAdapters* and *SpecificDataStreamAdapters*. A GenericDataStreamAdapter consists of a combination of a *DataStreamProtocol* (e.g. MQTT), that describes how the data source can be connected and a *Format* (e.g. JSON) representing the serialization of events. Some machine interfaces can not be separated between protocol and format and require custom solutions (e.g. OPC UA). Therefore, the concept of a *SpecificDataStreamAdapter* can be used. Specific data stream adapters support custom solutions as well as implementations of proprietary data sources.

To be able to reuse adapters in multiple different scenarios they should be as generic as possible and citizen developers must be able to configure them by providing configuration parameters. Therefore, the concept of *StaticProperties* is used as a configuration template [Riemer 2016]. There are several types of static properties, which are automatically validated (e.g. strings, Uniform Resource Locators (URLs), numeric values). *Adapters*, *Formats*, and *Protocols* use *StaticProperties* to define custom configurations. Those configurations can also be stored as an *Adapter Template*. This is a description about the data source that contains all the necessary information that is required to instantiate an adapter. With those adapter templates it is possible to share adapter descriptions and thereby reduce the modeling effort for others.

Next, we present two examples of adapter descriptions which are connecting to a temperature sensor. In the first case, temperature values are read from a MQTT broker and in the second case, the current state of a conveyor is read from a Programmable Logic Controller (PLC).

Generic Adapter

Events transmitted over a protocol like MQTT, can be represented in multiple formats. Therefore, a generic adapter allows to define a combination of a protocol, to read the data from the source, and a format to parse the events into an internal representation. Listing 6.1 shows an instance of a *GenericDataStreamAdapter*, with MQTT as the protocol and JSON as the format connecting a temperature sensor. Such a sensor could for example be used as retrofitting for an already existing production line, to see if the environment temperature influences the production process. The adapter is defined in line 3 with *"Temperature Sensor"* as the name. The MQTT protocol description (line 10) has two static properties, one defining the broker URL (line 19) and one for the topic (line 24). For formats it is also possible to provide configurations with static properties. Line 15 shows the JSON format, which does not need any further configurations in this case.

```
1  @prefix sp: <https://streampipes.apache.org/vocabulary/v1/> .
2
3  <sp:adapter1>
4    a sp:GenericDataStreamAdapter ;
5    rdfs:label "Temperature Sensor" ;
6    sp:hasProtocol <sp:protocol/stream/mqtt> ;
7    sp:hasFormat <sp:format/json> ;
8    sp:hasDataStream <sp:dataStream1> .
9
10 <sp:protocol/stream/mqtt>
11   a sp:DataStreamProtocol ;
12   rdfs:label "MQTT" ;
13   sp:config <sp:staticproperty1>, <sp:staticproperty2> .
14
15 <sp:format/json>
16   a sp:Format ;
17   rdfs:label "JSON" .
18
19 <sp:staticproperty1>
20   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 6.1: Example for a MQTT adapter instance

Specific Adapter

With specific adapters, it is possible to support data endpoints that have specific interfaces. Further, custom solutions can be provided, that do not need many configurations. Often only the endpoint (e.g. IP-address) is required to connect a data source. This can be used by sensor vendors to provide adapters for their sensors, or build custom solutions for individual data sources, that do not need any further configurations. Listing 6.2 shows an example of an adapter connecting to a Siemens PLC. This is the same example as used in Listing 3.1 in Chapter 3. The adapter is described in line 3 with the name *"Light barrier conveyor"* and has three static properties. First the IP address of the PLC must be defined (line 9), then the polling interval for reading the data is specified (line 14). A PLC represents data in registers, therefore a user has to specify the value of the register that should be read. From line 19 to line 32 the configuration for the event property of the entry barrier sensor is defined.

```
1  @prefix sp: <https://streampipes.apache.org/vocabulary/v1/>
2
3  <sp:adapter2>
4    a sp:SpecificDataStreamAdapter ;
5    rdfs:label "Light barrier conveyor" ;
6    sp:hasDataStream <sp:dataStream2> ;
7    sp:config <sp:staticproperty3>, <sp:staticproperty4>, <sp:staticproperty5>, <sp:
         staticproperty6> .
8
9  <sp:staticproperty3>
10   a sp:FreeTextStaticProperty ;
11   rdfs:label "PLC IP-Address" ;
12   sp:hasValue "192.168.188.22" .
13
14 <sp:staticproperty4>
15   a sp:FreeTextStaticProperty ;
16   rdfs:label "Polling Interval [ms]" ;
17   sp:hasValue "1000" .
18
19 <sp:staticproperty5>
20   a sp:FreeTextStaticProperty ;
21   rdfs:label "Runtime Name" ;
22   sp:hasValue "I_entry" .
23
24 <sp:staticproperty6>
25   a sp:OneOfStaticProperty ;
26   rdfs:label "Node Name" ;
27   sp:hasValue "%I1.0" ;
28   sp:hasOption <sp:optionBoolean>, <sp:optionByte> .
29
```

```
<sp:optionBoolean>
  sp:hasName: "Bool" ;
  sp:isSelected: true .

<sp:optionByte>
  sp:hasName: "Byte" ;
  sp:isSelected: false .
```

Listing 6.2: Siemens S7 PLC adapter instance to read the status of a light barrier from a PLC controlling a conveyor

The configurations for the adapters can get quite complex, this is why we provide user support during the modeling process introduced later in this section.

6.3.2. Transformation Rule Model

Production lines in manufacturing companies often use different machines from different vendors. Therefore, it is required to harmonize data before it can be analyzed. To achieve this, the possibility of transforming, reducing, and/or anonymizing directly on ingestion is required. In this section the *TransformationRules* (Figure 6.6) are introduced, that are used by our adapter model in Figure 6.5. There exist three types of transformation rules depending on which part of the event stream or event model needs to be transformed. *SchemaTransformationRules* transform the structure of the event schema. *ValueTransformationRules* change values of event properties and *StreamTransformationRules* change the data stream itself. In the following the different transformation rules for the three types are explained in detail.

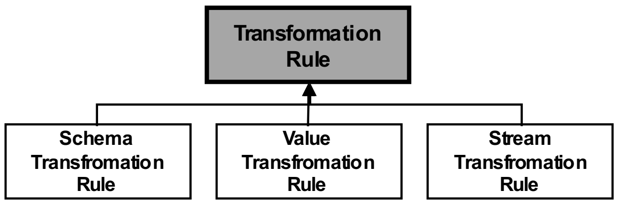

Transformation Rule
Schema Transfromation Rule
Value Transfromation Rule
Stream Transfromation Rule

Figure 6.6.: Model of the transformation rules with schema-, value-, and stream transformation rules

Figure 6.7 shows the model for all *SchemaTransformationRules*. Those rules change the schema of the event by adding, moving, or removing individual properties. On the left is the *AddFixedPropertySchemaTransformationRule*, the *hasRuntimeName* defines the name of the property and *hasValue* specifies the value that is added to that property. This fixed event property will be added to each event in the event stream. Next is the *AddTimestampSchemaTransformationRule*, it will append the current timestamp to the event, when it is processed in the adapter. To rename individual properties the *RenameSchemaTransformationRule* can be used. It needs the old and the new runtime name for the property. The structure of events is not required to be flat, especially nested structures are also supported. The *AddNestedSchemaTransformationRule* appends a new (empty) nested property to the event. With the *hasRuntimeName* predicate it is defined what the name for the property is. The *MoveSchemaTransformationRule* describes how to move individual properties between nested properties within an event. The predicate *hasOldRuntimeName* defines which property should be moved and *hasNewRuntimeName* defines the new location of the property. To delete individual properties the delete rule can be used by specifying the runtime name of the property to delete.

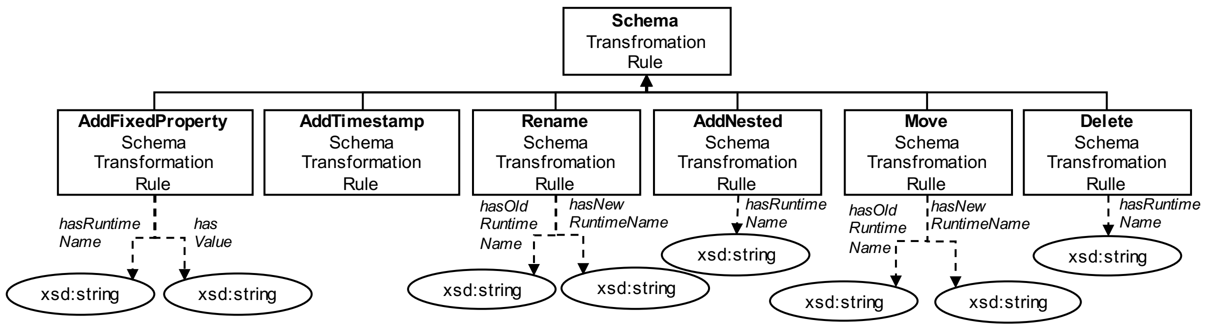

Schema Transfromation Rule
AddFixedProperty Schema Transformation Rule
AddTimestamp Schema Transformation Rule
Rename Schema Transfromation Rulle
AddNested Schema Transformation Rulle
Move Schema Transformation Rulle
Delete Schema Transformation Rulle
hasRuntime Name
has Value
hasOld Runtime Name
hasNew RuntimeName
hasRuntime Name
hasOld Runtime Name
hasNew RuntimeName
hasRuntime Name
xsd:string

Figure 6.7.: Model of schema transformation rules

Figure 6.8 depicts *ValueTransformationRules* that always affect a single *EventPropertyPrimitive* of an event. There are four ValueTransformation rules, the first one is a *NumberValueTransformationRule*, this can perform simple mathematical functions like adding or subtracting a fixed number, to harmonize data.

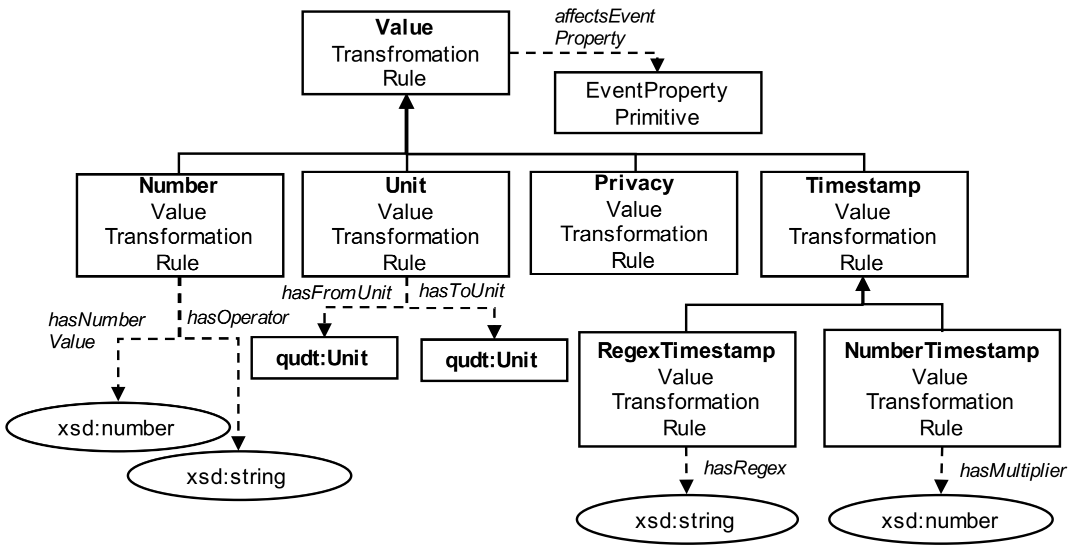

Value Transformation Rule
affectsEvent Property
EventProperty Primitive
Number Value Transformation Rule
Unit Value Transformation Rule
Privacy Value Transformation Rule
Timestamp Value Transformation Rule
hasNumber Value
hasOperator
hasFromUnit
hasToUnit
qudt:Unit
qudt:Unit
xsd:number
xsd:string
RegexTimestamp Value Transformation Rule
NumberTimestamp Value Transformation Rule
hasRegex
hasMultiplier
xsd:string
xsd:number

Figure 6.8.: Model of value transformation rules

The second is a *UnitTransformationRule* that changes the unit of a property value. For the description of the units the QUDT ontology[2] is reused for specifying the unit as well as for conversion between units. The third one is a *PrivacyTransformationRule*, to anonymize values of event properties. The fourth type of value transformation rules are *TimestampValueTransformationRules* that are able to transform different representations of timestamps in the events into an internal representation (UNIX timestamp). It is distinguished between a rule transforming a date string by providing a regular expression (*RegexTimestampValueTransformationRule*) and converting a number into a UNIX timestamp in milliseconds (*NumberTimestampValueTransformationRule*).

The last type of transformation rules are *StreamTransformationRules* shown in Figure 6.9, that influence the data stream and the events within that stream. Data streams can be aggregated either by counting (*CountAggregateStreamTransformationRule*) or by time (*TimeAggregateStreamTransformationRule*) windows. Sometimes, event streams contain duplicates (events were all property values are equal), which can be removed with the *RemoveDuplicatesStreamTransformationRule*. It will remove all duplicates from the event stream and ensure that each event is just published once to the harmonized message broker.

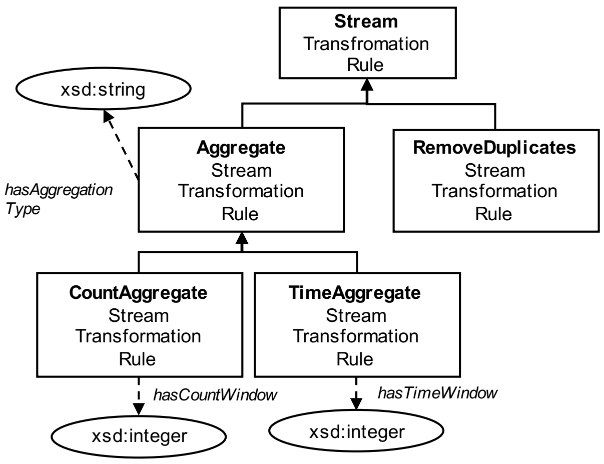

Stream Transfromation Rule
xsd:string
hasAggregation Type
Aggregate Stream Transformation Rule
RemoveDuplicates Stream Transformation Rule
CountAggregate Stream Transformation Rule
TimeAggregate Stream Transformation Rule
hasCountWindow
hasTimeWindow
xsd:integer
xsd:integer

Figure 6.9.: Model of stream transformation rules

The introduced rules were selected to cover a wide range of cases that can occur when connecting and harmonizing data of machines. However, sometimes there are new requirements which require new rules. For such cases the model is designed to be expandable. New rules can be added in the future to accommodate new situations.

Each adapter instance contains a set of rules that are modeled by a user. Listing 6.3 shows an example of a *UnitTransformationRule*. It describes the unit transformation of a temperature sensor (line 3) from degree Fahrenheit (line 4) to degree Celsius (line 5). All instances of rules are basic changes to the events and have a similar structure. A set of

[2]https://www.qudt.org/ (accessed on 04/10/2021)

rules can be used for more complex transformations to be applied by concatenating their execution, which is explained in more detail in the next section.

```
<sp:transformationrule1>
  a sp:UnitTransformRule ;
  sp:runtimeKey "temperature" ;
  sp:fromUnit "http://www.qudt.org/2.1/vocab/unit#DegreeFahrenheit" ;
  sp:toUnit "http://www.qudt.org/2.1/vocab/unit#DegreeCelsius" .
```

Listing 6.3: Unit transformation rule example

6.3.3. (Edge-) Transformation Functions

In the previous sections, the model for adapters and transformation rules was introduced. In this section, it is presented how the transformation model can be used to configure transformation functions that are applied on the actual event streams. Those functions are concatenated into a preprocessing pipeline, which is automatically instantiated. These functions usually run directly at the edge within a worker, leveraging local compute power to only transmit already cleaned and harmonized data. This is especially important for high frequency event streams (e.g. sensors in machines), they can be early aggregated which reduces the load on the network infrastructure significantly.

Functions take an *event e* and *configuration c* containing multiple values as an input and return a transformed *event* e'. Each function has a corresponding transformation rule and adapter instances contain a set of functions to form a preprocessing pipeline. Equation 6.1 shows the definition of the function for the preprocessing pipeline. It is a concatenation of multiple transformation functions, that take the *event e* as an input and calculate the resulting *event* e'. The rules must be applied in a fixed order to ensure the correctness of the transformations, first the schema, then the value, and last the stream transformations are applied.

$$F(e) = f_n(f_{...}(f_1(e, C_1), ...), C_n) = e' \tag{6.1}$$

In the following tables, for each transformation rule type (schema, value, stream) an example transformation function is presented. On the left, the name of the transformation function is given, in the middle an example event instance (serialized in JSON), and on the right the corresponding configuration *C* for the example. The functions are independent of the serialization of events, JSON is used in our examples as a representation because it is human readable.

Table 6.1 shows six *schema transformation functions*. The first one is *add fixed property*, it adds a new property to the event with a fixed value. In the example a new property with the runtime name c_1=*"id"* is added with the value c_2=*"sensor5"* to an empty event. The *add nested* function adds a new empty nested event property, it can be used to add further nested information (e.g. add an object representing a geo coordinate with latitude and longitude values). In the example a new nested property with the runtime name c_1=*"a"* is added to an empty event. With the *move* function the parent property can be changed. It does not change the property itself, it only changes the location within the event schema. For the configuration the names of the properties are used delimited by colons c_1=*"a:b"*. The new location is defined in the second parameter c_2=*"b"*. To add the current timestamp to the event, the *add timestamp function* can be used, which doesn't need any further configurations. Renaming runtime names of event properties can be done using the *rename* function. The individual properties are referenced again by the runtime name, c_1=*"old"* is the old name and c_2=*"new"* the new runtime name. The last function is removing individual properties and is done with the *delete* function, the only configuration is the runtime name of the property to delete c_1=*"a"*.

Table 6.1.: Schema transformation functions

Rule	**Example Function** $[e \rightarrow e']$	**Configuration** $[C]$
Add Fixed Property	{} → {"id": "sensor5"}	c_1="id" c_2="sensor5"
Add Nested	{} → {"a": {}}	c_1="a"
Move	{"a": {"b": 1}} → {"a": {}, "b": 1}	c_1="a:b" c_2="b"
Add Timestamp	{} → {"timestamp": 1575476535373}	
Rename	{"old": 1} → {"new": 1}	c_1="old" c_2="new"
Delete	{"a": 1} → {}	c_1="a"

Functions that affect values of event properties are shown in Table 6.2. With the *number* rule, numerical values can be changed via basic mathematical operations (e.g. add, subtract, multiply with a fix number). The value of property c_1=*"x"* should be changed by adding *(c_2="ADD")* an offset of *(c_3=10)* . To anonymize individual values, the *privacy* rule can be used. In the example the value c_1=*"name"* is anonymized by applying a hash function. With the hashed value it is still possible to perform a grouping, but the original value can no longer be reproduced. Different units often make it difficult to analyze data, because the unification has to be done manually and adds additional complexity. Therefore the *unit* function is introduced to harmonize data at ingestion time. In the example the unit of the property c_1=*"temp"* is changed from degree Fahrenheit c_2=*"unit:DEG_F"* to degree Celsius c_3=*"unit:DEG_C"*. The values are changed according to the transformation factor specified in the QUDT-ontology[3]. Time is often represented in different forms in events, for example as a formatted string (shown in the example) or as a number (e.g. unix timestamp). Similar to the problem with different units, it is much easier to perform calculations on different data streams when the representation of the time is consistent. A unix timestamp is used for the representation, therefore this function transforms the time into a unix timestamp in milliseconds. For the configuration the name

[3]http://www.qudt.org/ (accessed on 04/10/2021)

c_1=*"time"* of the event property is required and a regular expression c_2=*"yyyy/mm/dd hh:mm"* how to interpret the string value.

Table 6.2.: Value transformation functions

Rule	**Example Function** $[e \rightarrow e']$	**Configuration** $[C]$
Number	{"x": 5} $\rightarrow$ {"x": 15}	c_1="x" c_2="ADD" c_3=10
Privacy (SHA-256)	{"name": "Pia"} $\rightarrow$ {"name": "ca9..."}	c_1="name"
Unit ($^\circ C \rightarrow {}^\circ F$)	{"temp": 41} $\rightarrow$ {"temp": 5}	c_1="temp" c_2="unit:DEG_F"[4] c_3="unit:DEG_C"[3]
Timestamp	{"time": "2021/02/25 16:29"} $\rightarrow$ {"time": 1614270540000}	c_1="time" c_2="yyyy/mm/dd hh:mm"

In the last Table 6.3, two different *stream transformation functions* are shown. First the *remove duplicates* function, which removes duplicates from the event stream. Duplicates are defined as events where all properties are equal. Two properties are defined as equal when their values are equal, for primitive properties this equality check is simple because the values have a basic type (e.g. boolean, number, string). For nested event properties this check is performed recursively for each primitive property within the nested property. Only equal events are filtered out of the stream. E.g. when all sensor values are the same, but the timestamp is different events are not filtered. With the *aggregate* transformation rule the frequency of the event stream is reduced by aggregating multiple events into one event. For the aggregation two different window types are possible, time based windows and counting based windows c_1=*"time window"*. The window size c_2=*"5sec"* must be specified for those windows and how the values should be aggregated c_3=*"mean"*. This is especially relevant for the different data types. For boolean and numerical properties it is possible to calculate a single value from the different events. For strings, however, it must be defined how multiple instances within the aggregation can be reduced to a single value.

Table 6.3.: Stream transformation functions

Rule	**Example Function** $[e \rightarrow e']$	**Configuration** $[C]$
Remove Duplicates	{"a": 1},...,{"a": 1} $\rightarrow$ {"a": 1}	
Aggregate	{"a": 2},...,{"a": 1} $\rightarrow$ {"a": 1.5}	c_1="time window" c_2="5sec" c_3="mean"

[4] unit: http://www.qudt.org/2.1/vocab/unit

6.4. User Interaction

In the previous section, the adapter model and the transformation rules were defined. In this section it is shown how such models can be created by citizen developers. For them it can be difficult to specify a model because they often lack the necessary modeling expertise. There is no need of explicitly modeling an adapter instance in a RDF-Editor. Rather, we use an approach that users are more familiar with by providing user input over automatically generated forms that use RDF as a representation format in the background. The user input is directly validated and the system additionally tries to provide useful information, that reduces the complexity of the modeling process. Based on the user interaction with the Graphical User Interface (GUI), the adapter model is derived and the adapter is automatically instantiated.

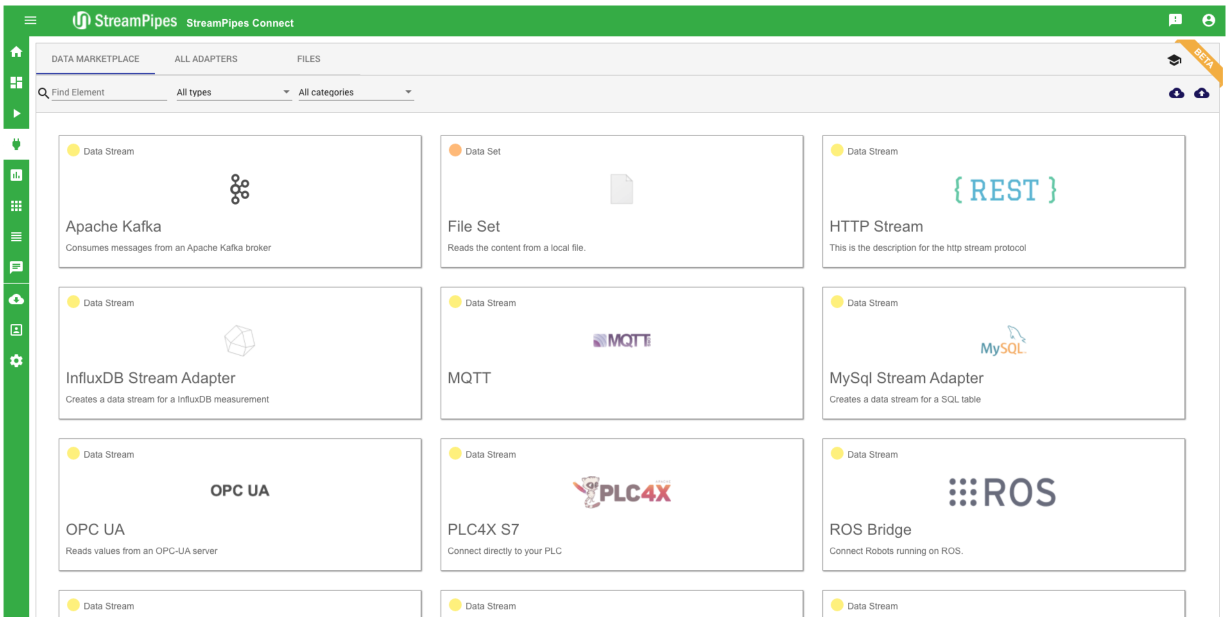
6.4.1. Adapter Marketplace

The adapter marketplace is the central location where users can create and manage their adapters and data sources. Figure 6.10 shows a screenshot of the adapter marketplace in StreamPipes. It lists all the available adapters (generic and specific) as well as running adapter instances. For each of the listed adapters an implementation is provided with the application logic to handle the connection and processing of the events from the respective data source type. It is also possible to store and share user configurations of adapters over adapter templates. These templates also appear in this view and can be used directly by other users. To export and share the description, the adapter information is serialized (e.g. JSON, JSON-LD).

In this work, we developed models and concepts, as well as a SDK to add new adapters. Furthermore, we provided already several implementations that cover a wide range of standard protocols and formats. Additionally, some members of the StreamPipes community have already used the SDK to create more adapters. So far, over 30 adapters are integrated and new types are added regularly. Table 6.4 gives an overview of the current available adapters which are so far integrated into Apache StreamPipes.

Table 6.4.: Overview of implemented adapters

Adapters	Generic	Specific
Stream	MQTT, MySQL, HTTP, InfluxDB, File, Apache Kafka, Apache Pulsar, HDFS, Image Zip's	OPC UA, PLC Modbus, PLC S7, Robot Operating System (ROS), Slack, Random Generator, TI Sensor, Flic Button, NETIO, Coindesk, GDELT, IEX News, Wikipedia Edits, ISS Location, Machine Simulator
Set	HTTP, InfluxDB, MySQL, File, Images Zip's	Random



Data Stream
Apache Kafka
Consumes messages from an Apache Kafka broker

Data Set
File Set
Reads the content from a local file.

Data Stream
{ REST }
HTTP Stream
This is the description for the http stream protocol

Data Stream
InfluxDB Stream Adapter
Creates a data stream for a InfluxDB measurement

Data Stream
MQTT
MQTT

Data Stream
MySQL
MySql Stream Adapter
Creates a data stream for a SQL table

Data Stream
OPC UA
OPC UA
Reads values from an OPC-UA server

Data Stream
PLC4X
PLC4X S7
Connect directly to your PLC

Data Stream
ROS
ROS Bridge
Connect Robots running on ROS.

Data Stream

Data Stream

Data Stream

Figure 6.10.: Overview of the data marketplace

6.4.2. Adapter Modeling Process

Figure 6.11 depicts the four different steps a user has to follow to create a new adapter instance. The first step is to select the adapter type from the marketplace (Figure 6.10). In step 2, the information for the adapter model (Figure 6.5) must be provided and in step 3 and 4 the information for the transformation rule model (Figure 6.6) must be given. The configuration process of the selected adapter differs slightly depending on whether it is a generic or a specific adapter. For generic adapters, the protocol and the format must be configured via static properties. For specific adapters, only the configuration of the adapter itself must be provided. The configuration process is the same for data sets and data streams. Once the adapter model is configured in step 2, the event schema is defined in step 3.

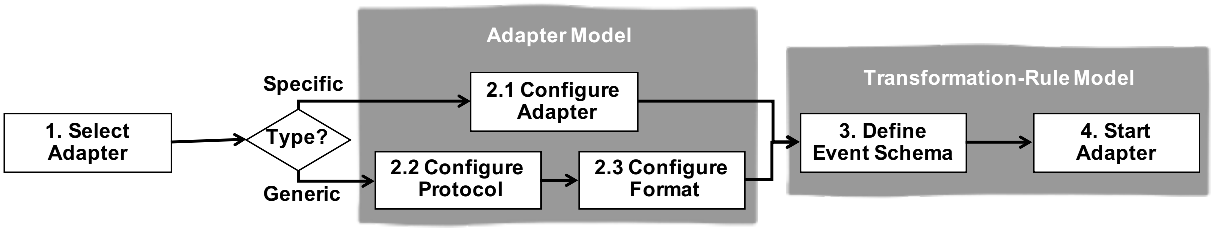


Figure 6.11.: Adapter configuration process

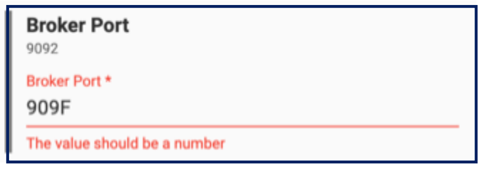
Therefore, sample events are collected from the previously configured data source. Based on those samples an initial proposal for the event schema is provided by the system.

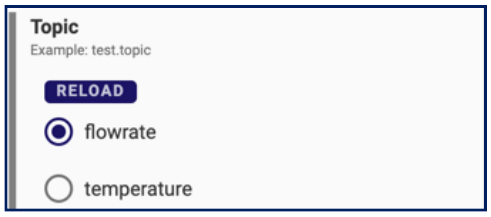
The level of detail of this suggestion depends very much on the selected adapter type. Often some additional meta information (e.g. unit) has to be added. In the fourth and last step, further configurations such as the adapter name or a description can be added. Furthermore, settings can be selected which influence the resulting event stream, such as a maximum event frequency. Once the adapter has been modeled, the adapter model is transmitted to the master, which triggers the instantiation of an adapter instance in the worker. After the successful deployment, the instance automatically starts processing data.

6.4.3. (Semantic) User Guidance

In step 2 in Figure 6.11 a user has to provide information about the data source and how to retrieve data from it. Therefore, values for the defined static properties must be provided. To reduce this configuration effort, the system tries to support the user as much as possible by leveraging the semantic model and meta-data provided by the data sources. Since the data sources are heterogeneous, the availability and quality of this meta-data differs between adapter types. However, the system tries to provide a guided process for the user.

The most basic user support of the system is a validation of all user inputs according to the requirements of the static properties, e.g. ensure correct data type, or formatting of URLs. When the entered value does not conform with the requirements of the static property a warning message is provided to the user (Figure 6.12 on the left).


Broker Port
9092
Broker Port *
909F
The value should be a number


Topic
Example: test.topic

[✓ flowrate]
temperature

Figure 6.12.: Screenshots of configuration with static properties

For most adapters, some initial information like the address of the endpoint (e.g. IP-address) must be provided manually. Whenever possible, the system uses this initial information to collect meta-information from the data endpoint, which is used for input validation or automatic setting of configuration parameters. For example, some message brokers provide an API to pull a list of all available topics (Figure 6.12 on the right). A user can then chose a topic from that list, which eases the configuration process, as well as reduces a possible source of errors compared to entering the topic name manually. If this information is not available, for example in PLC controllers, it must be provided manually. However, for this case it is possible to export the register addresses from other software solutions that are used to program those devices (e.g. the TIA portal, used for programming Siemens S7 PLCs) and import it as a file. This speeds up the configuration process and only the IP address and polling interval for the adapter must be provided manually. Once the information is entered, a connection to the PLC is established to check if the entered values are valid and are readable. Otherwise a warning message is shown to the user that some configuration parameters must be changed. Since each adapter type has different capabilities, our approach is flexible to provide a good user experience for each of those and reduce the configuration effort when connecting new sources.

After configuring all the information of step 2, a connection is established to the source and sample data is extracted. Based on this sample data the event schema is derived and presented to the user in the GUI, as shown in Figure 6.13 on the left. A user can further change the provided event schema via dragging and dropping event properties. New event properties, can be added or unwanted once can be deleted. The example from the figure shows a temperature event. The event has an *id* property as well as a nested *value* containing the temperature value, named *temp*. On the right side is the event after editing, the nested structure was removed and the temperature property is renamed. Additionally, the current timestamp will be added to each event when it is processed in the adapter.

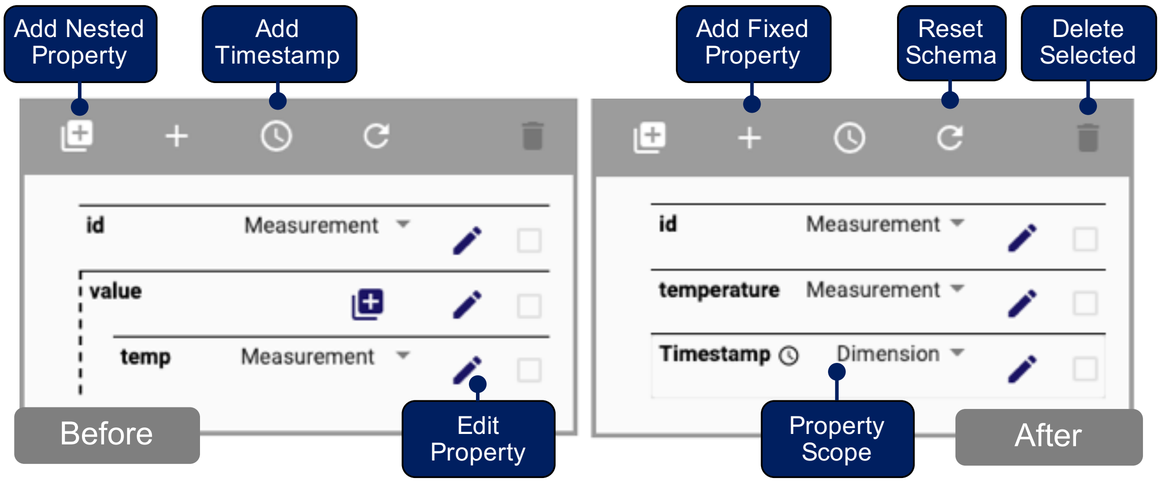
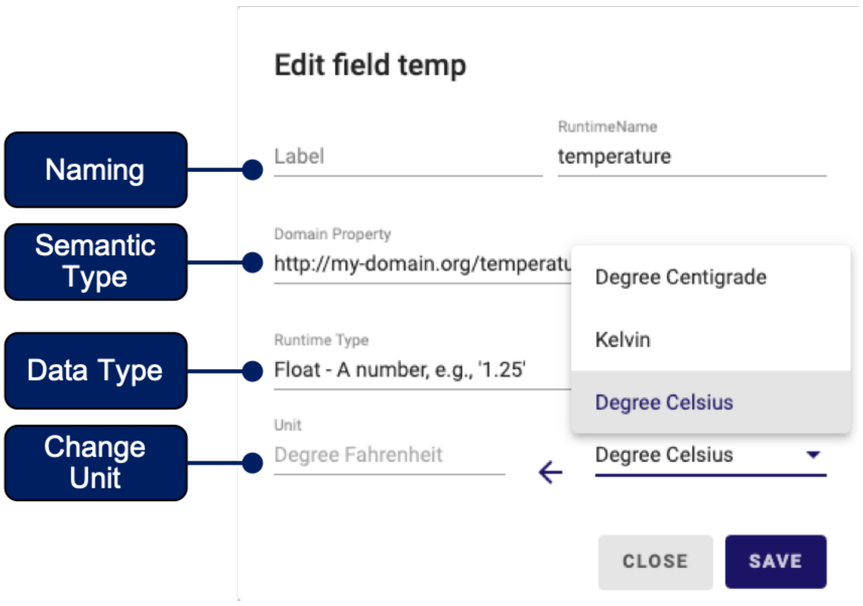


Figure 6.13.: Screenshots of configuration of the event schema

Other meta information can be added as well. For example the property scope for each event property or the data type. Again, the system tries to automatically fill out as many fields as possible, but some information must be provided by the user. An example where almost no information is provided in advance are CSV files read from an FTP server that do not contain a header. Only the amount of properties is known and the data type for each column can be guessed by reading some sample events. All the other missing information must be provided by a user. Another example would be a message broker (e.g. MQTT) sending JSON object, as shown in the example in Figure 6.13. The event schema and data types can be inferred, but no additional meta-information like the semantic type or the unit is available. Other data sources (e.g. OPC UA) already have a rich meta-model that can be used to enrich the event schema with this information, so the user does not need to provide the units manually.

Figure 6.14 shows how the configuration parameters for an individual event property can be provided. First of all, it is possible to change the name, further the semantic type can be added for each property. A domain specific vocabulary can be provided in the background, that provides suggestions for the semantic type. This information can later help when the pipeline is build, by providing suggestions and semantic validations to the user. Additionally, the data type is presented. The system can derive the data type for almost all data sources. Also a value for the unit can be provided in the unit field. When the unit should be changed it is possible to specify it directly in the GUI (on the right side). Only semantically correct transformations are suggested to the user.



Edit field temp

Label

RuntimeName: temperature

Domain Property: http://my-domain.org/temperatu

Runtime Type: Float - A number, e.g., '1.25'

Unit: Degree Fahrenheit ← Degree Celsius

Figure 6.14.: Screenshots of the configuration of an event property

Based on the user interaction in the GUI, the model of the transformation rules is derived in the background. This is done once a user finished editing the event schema. Then the system compares the *'original'* schema that was suggested in the beginning with the *'new'* configured event schema to calculate the transformation rules based on the difference of those schemas. The new schema is later used in the adapter model to describe how the data in the event stream looks like.

6.5. Running Example

In this section, a complete example is presented that is based on the scenario introduced in Section 3.2. It is explained how the methods, models and tools presented in this chapter can be used by citizen developers to ingest IIoT data from industrial assets.

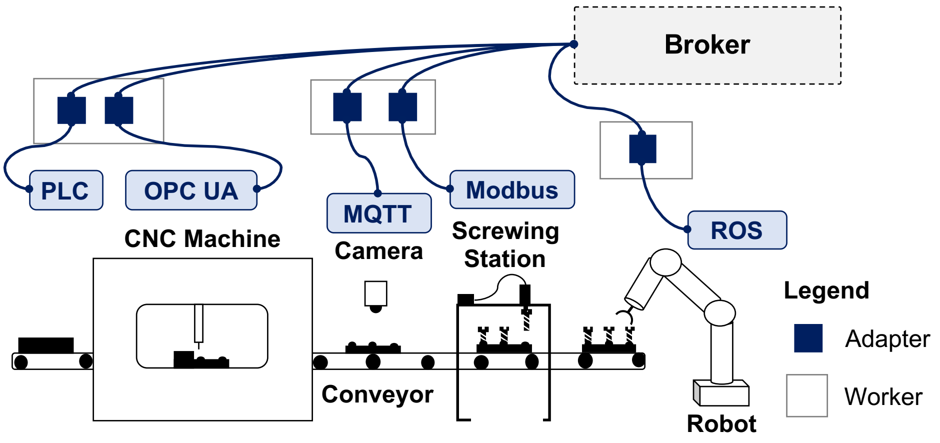
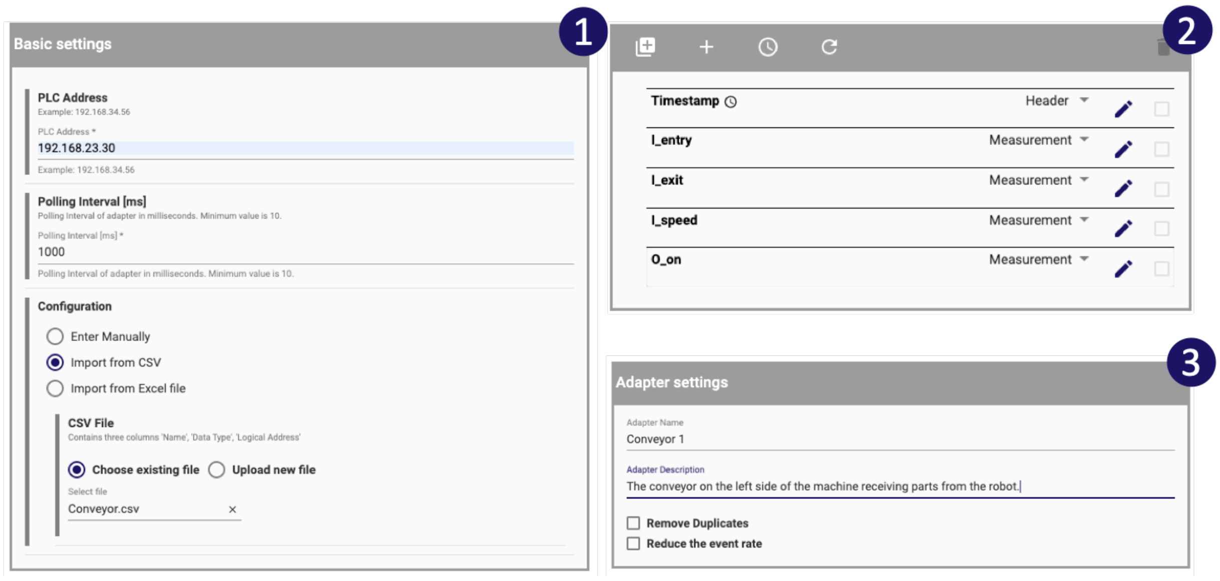


Figure 6.15.: Using adapters to connect the different machines of the running example

Figure 6.15 shows the example production line with the various interfaces of the machines that should be connected as data sources. For each of those sources, an adapter model has to be defined that is used to automatically create the event stream containing sensor values. The model creation is performed with the workflow introduced in Section 6.4 by citizen developers. To start the workflow, a user selects the type of adapter that should be connected in the adapter marketplace and follows the guided configuration process. One example workflow is illustrated in Figure 6.16 for a PLC controlling a conveyor. There are multiple values that can be read from the PLC, like whether the conveyor is running or not *(O_on)*. Furthermore, a boolean value represents when a package is entering or leaving the conveyor *(I_entry, I_exit)*, and the speed *(I_speed)* at which the conveyor is running at. In step ❶, the basic configurations like IP address, polling interval and a file with the variables must be provided. The file contains all the register names and addresses for which values should be included into the event stream. After that, the event schema can be seen and edited, (e.g. remove unused properties or transform the unit etc.). At step ❷, the resulting event schema of the data stream is shown. In step ❸, the final parameters can be set, by providing a name and a description for the adapter. This adapter description is transmitted to the master and forwarded to a worker. There, the adapter is started and directly produces an event stream that is sent to the message broker for further processing.

As previously mentioned, based on the user interaction the preprocessing rules are created in the model. Figure 6.17 shows an example that illustrates how those rules are applied on a sample event *e* from the PLC. On the left is the raw input *event e* that



1

Basic settings

PLC Address
Example: 192.168.34.56
PLC Address *
192.168.23.30
Example: 192.168.34.56

Polling Interval [ms]
Polling Interval of adapter in milliseconds. Minimum value is 10.
Polling Interval [ms] *
1000
Polling Interval of adapter in milliseconds. Minimum value is 10.

Configuration
Enter Manually
[✓ Import from CSV]
Import from Excel file

CSV File
Contains three columns 'Name', 'Data Type', 'Logical Address'
[✓ Choose existing file] Upload new file
Select file
Conveyor.csv

2

Timestamp	Header
I_entry	Measurement
I_exit	Measurement
I_speed	Measurement
O_on	Measurement

3

Adapter settings

Adapter Name
Conveyor 1

Adapter Description
The conveyor on the left side of the machine receiving parts from the robot.

Remove Duplicates
Reduce the event rate

Figure 6.16.: Connecting the Conveyor PLC

is collected from the interface of the PLC and forwarded to the first function f_1. The runtime name of the speed property is changed from *I_s* to *I_speed*. Then the property *I_high* is removed because the user decided that this property is not relevant for the analytics tasks. Since the events do not have a timestamp, it is added within the adapter in function f_3. In the last step of the preprocessing pipeline in function f_4 the unit of the speed property is changed from foot per second to meters per second. After applying all the functions, the resulting *event* e' is sent to the defined unified broker of the adapter. This preprocessing pipeline is performed for each event in the adapter to ensure that all events of the resulting event stream are consistent. By ensuring this within the adapter, it reduces the complexity of the implementation for downstream analytics algorithms.

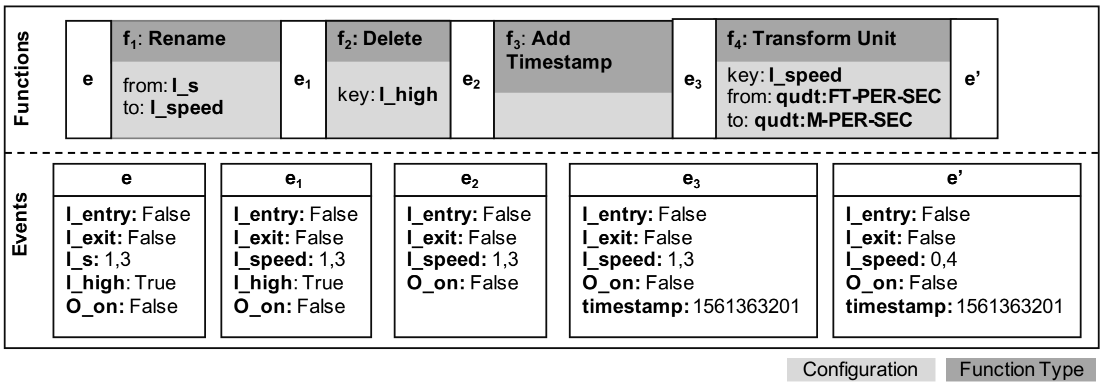


Figure 6.17.: Example of a preprocessing pipeline

After users have created an adapter for each of the different data sources, they are able to analyze the data directly by processing it from the harmonized message brokers. Several different tools can be used to accomplish this analytics task, either graphically-based (e.g. Apache StreamPipes) or code-based (e.g. Apache Flink). The adapters ensure the harmonization of the technical parts (format, protocol) reducing the complexity of the component subscribing to this data. Both the format and protocol are not fix and could be changed in the future by changing the defined event grounding in the adapter descriptions.

Lastly we want to present a simple real-world example were the approach is used. Figure 6.18 shows a production line on the left, which is controlled by several PLCs. On the right is a dashboard that contains the results of the calculations performed on the ingested data. It contains live information about the conveyer speed and temperature within the machine. Additionally, some numbers are presented to monitor the current production performance and scrap rate. Also the image of the last processed part can be seen on the bottom right.

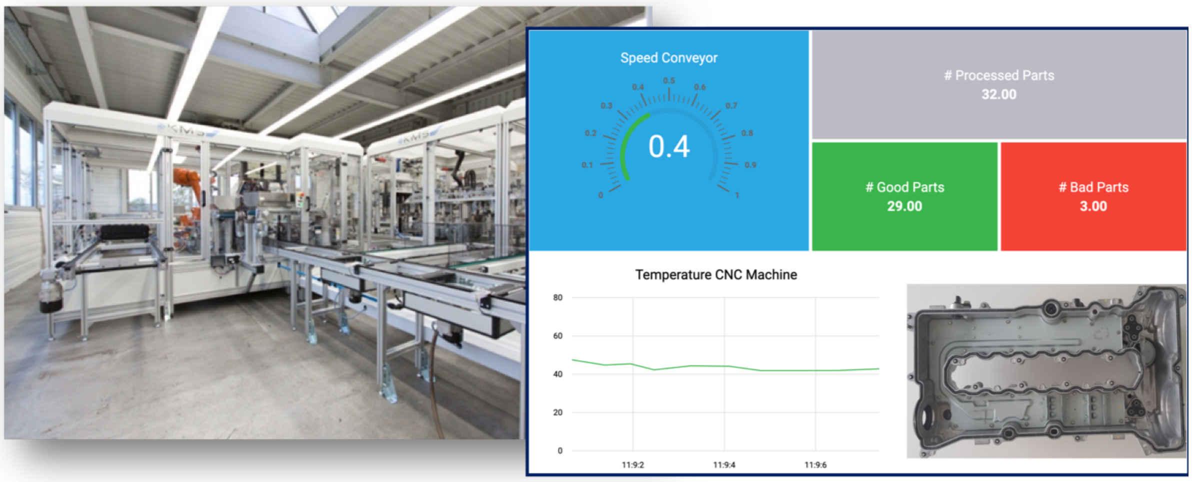


Figure 6.18.: Example dashboard for the example production line

Based on the contribution of this section, citizen developers are able to connect heterogeneous industrial sources themselves. These event streams can then be used to apply further analytic algorithms. As a first step, the data is often made available in a dashboard to users, in this case operators on the store floor, to provide live insights into the production process. The time as well as the complexity is reduced to connect the data sources and citizen developers are enabled to quickly adapt to changing conditions on the shop floor without the need of a software developer. They are able to provide the necessary information to the shop floor workers themselves, without the involvement of other people.

6.6. Summary

In this chapter, we presented how this work enables citizen developers to ingest data from industrial data sources, such as machines and their sensors.

First, basic data models that are relevant for the whole work were introduced in Section 6.2. The *event model* describes the structure of events consisting of a set of event properties. Those event properties can be described with multiple attributes, such as the type, semantic type or property scope. This static information can for example be used to automatically configure analytic algorithms. With the introduced *virtual sensors* it is possible to specify the similarity between different event streams. Therefore, it is defined how the equality of event properties can be checked. Further, the definition for adapters is provided.

Based on this adapter definition an *adapter model* was presented in Section 6.3. Adapter models can be created for *data streams* as well as *data sets* and are distinguished between *generic* and *specific* adapters. Generic adapters describe a protocol and a format. Specific adapters can be created for specific types of data sources. All the configurations are provided with static properties by the user. Additionally, the adapter model uses *transformation rules* to describe transformations to harmonize events at the edge. There exist three different types of rules. The type is defined by the part of the event the rule affects (*schema*, *value*, or *stream*). The model for the rules can be extended to cope with new situations in the future. For example, if there is a need to create new domain-specific rules. Based on those rules, *transformation functions* can be instantiated that are concatenated in a *preprocessing pipeline* at the edge to directly transform events on ingestion. These features ensure that the quality of event schema and property type is guaranteed and that subscribers of the data do not have to deal with inconsistencies. Extensive tool support has been created to validate the developed methods in various real-world settings as shown in Section 6.4. It starts with the *adapter marketplace* that already contains many different adapters. The configuration of them is done in a guided process with four basic steps providing user support in each of them. For this support, the semantics of the adapter model and available meta-data from the data sources is used. Additionally, the vocabulary of existing ontologies is reused.

In the last section an example from the motivating scenario is used to show the whole process of connecting new data sources with the developed approach. Based on that example the preprocessing pipeline for those events is explained in detail and how the individual functions work together. In the end a dashboard based on the connected data shows real-world data to illustrate a basic example how the ingested data can be used to monitor a production line.

7 Integrate

In the previous chapter, data was ingested from multiple heterogeneous data sources onto a harmonized message broker working as the transport layer. This chapter presents how this distributed message broker works, with the main goal to automatically adapt industrial event streams to reduce the network demands on the infrastructure. It is exploited that not all subscribers need every event with all the properties for their calculations. The basic idea is to use local compute resources to dynamically adapt the event stream based on the semantic description of the event stream that was introduced in the previous chapter as well as the requirements of the subscribers.

Section 7.1 gives a detailed overview of the concepts introduced in this chapter and shows how they fit into the context of this thesis. Then, in Section 7.2, event reduction strategies are presented and how they can be applied to different kinds of data. Afterwards, Section 7.3 introduces a wrapper for message brokers. First, the basic approach is explained, then the individual components like publishers and subscribers. At the end of the chapter, the running example is used to illustrate the whole process in Section 7.4, followed by a summary in Section 7.5.

7.1. Walkthrough

In this section, an overview is given by introducing the individual components of this chapter. One result of the previous chapter were adapters that connect different data sources. In this chapter, it is shown how this connected data can be integrated in a message broker, and how the event streams are dynamically adapted. The whole concept of the broker is designed as a wrapper around existing topic-based message brokers. Figure 7.1 shows two different execution pipelines processing data. The first pipeline stores raw data in a time-series database with the goal to persist the original values for offline analytics. In the second pipeline, a machine learning model that does not require all event properties for feature calculation is applied on the event stream. Therefore, not all information must be transmitted. This illustrates that different subscribers might have different requirements on the data and that the broker dynamically adjusts the event streams according to those requirements. How these event streams can be reduced is shown in the next section.

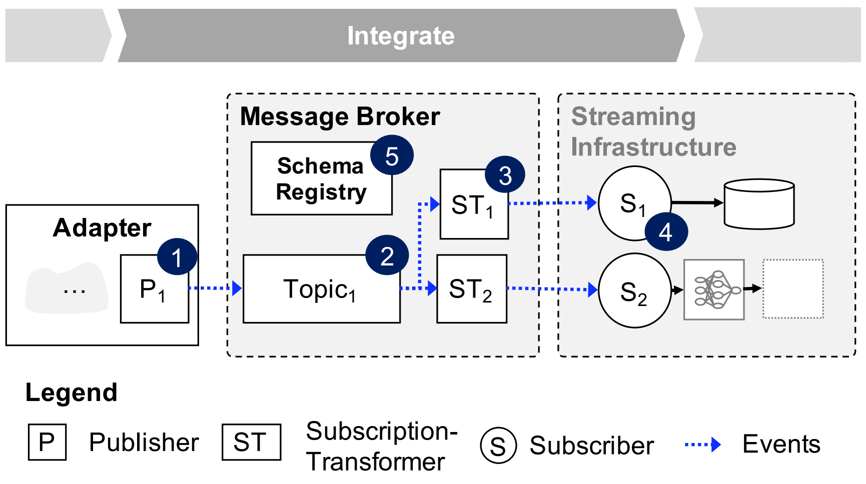


Figure 7.1.: Overview of the message broker and its individual components

On the left of Figure 7.1 is the adapter that connects the physical assets and publishes the data in the *publisher* P_1 ❶ to the broker. It takes the original events and reduces the content of those events. Only *partial events*, explained later in more detail, are transmitted onto a topic in the message broker ❷. Within the broker the events are prepared for the individual subscribers in a *subscription-transformer* ❸. At ❹ there is a *subscriber* receiving the *partial events* and reconstructing so called *virtual events* that are then forwarded to the application logic. The application logic is not aware of the transformations that happen within the broker. The broker has a high-level overview of existing publishers and subscribers and the event schema, stored in a *schema registry* ❺.

7.2. Event Reduction

In this section, data reduction techniques are presented, which allow to reduce the amount of data that needs to be transmitted when working with event streams. These techniques are especially useful when processing industrial event streams as they have a high volume and are often transmitted in a high frequency. First, different reduction strategies are presented which reduce the size as well as the frequency of events. Then, different reduction rules that can be applied to different types of data are discussed.

7.2.1. Reduction Strategies

There are various approaches to compress data, but these are usually only partially applicable to event streams, where events are not processed in a batch, but one after the other (event-at-a-time). In this section, we present different data reduction methods that exploit the structure in events or semantic event schema in order to optimize the transmission. Figure 7.2 gives an overview of the different reduction methods which are divided into two main categories.

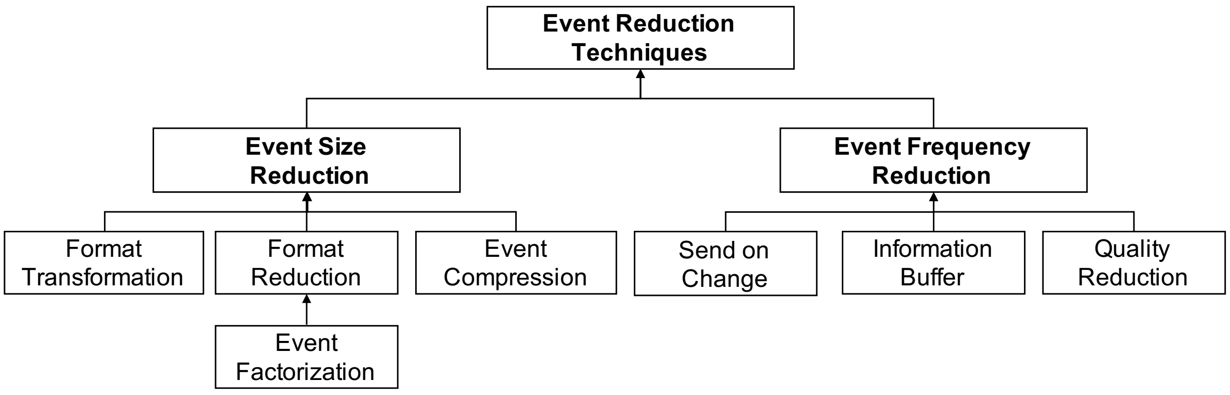


Figure 7.2.: Hierarchy of reduction strategies

First, the *event size* can be reduced, meaning the amount of bits transmitted for each event, secondly the *frequency* of events can be reduced, meaning that less events are sent. For both reduction strategies, there are different techniques. The main advantage of an event stream are that measurements and header information do not always change, or patterns in the data can be exploited. In the following, techniques are explained which reduce the *event size* directly at the publisher.

- **Format Transformation** Different formats can be used for serialization of events. These have different characteristics and often differ in the size of the resulting events. In cases where the size is particularly relevant, binary formats can be used, but these have the disadvantage of not being human-readable. The following example (Figure 7.3) shows the representation of a simple event as Extensible Markup Language (XML) on the left side and in JavaScript Object Notation (JSON) on the right side. It can be seen that the information is the same, namely the "id" and the value for the "temperature", but XML serializations tend to require more characters because it contains more boilerplate with the opening and closing tags.

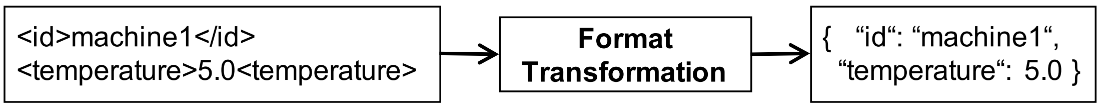


Figure 7.3.: Example format transfromation from XML to JSON

- **Format Reduction** With *Format Reduction* the size of events is reduced, by leaving parts out that are not transmitted. This is especially useful for parts of the event that have not changed or are not relevant for subscribers. In this case only values that have changed could be transmitted. Thus the receiver knows what is the new value and less data can be exchanged. In the example in Figure 7.4, the "humidity" value of the event is deleted and not transferred.

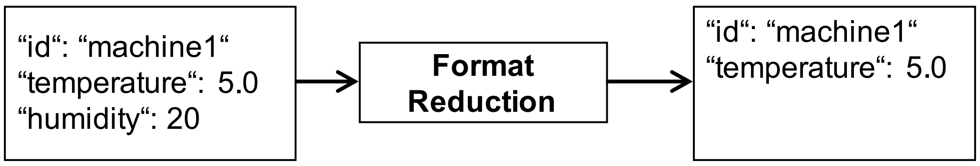


Figure 7.4.: Example of format reduction, the "humidity" value is not transmitted

- **Event Factorization** A special case of *Format Reduction* is *Event Factorization*. Instead of transmitting all of the information, data is factored out of the event and stored at a central location that can be accessed by a receiver. This information is then dynamically added to the event without transmitting it. The example (Figure 7.5) shows how meta-data of machines (e.g. its location) is stored in a central storage. The *id* of the event is used to find the appropriate meta-information and this is then added to the event payload. This also reduces the size of the event that is transmitted without loosing information. Event factorization can either be applied as a static process where it is defined offline which values are loaded from the central location, but it can also be applied dynamically, in a way that the event stream is monitored and information that does not change is extracted.

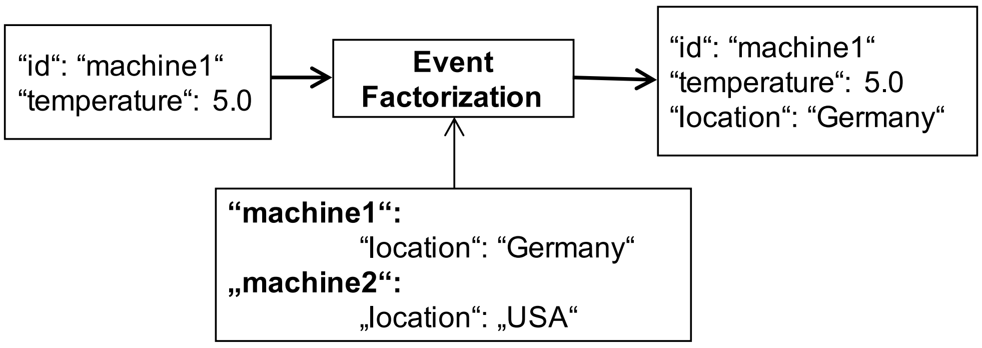


Figure 7.5.: Example for event factorization, the location is not transmitted, but appended to the final event

- **Compression** Another technique to reduce the size of data is to compress it before it is transmitted and decompress it at the receivers side. Several different compression techniques exist for that (e.g. GZIP[1]), but also techniques designed for publish-/subscribe systems are proposed [Doblander et al. 2016]. Those compression techniques can be used in addition to the other presented methods in this

[1]https://www.ietf.org/rfc/rfc1952.txt (accessed on 04/10/2021)

section since they mostly leverage the structure of the serialized data to reduce the size of the message.

So far, we have presented various reduction techniques to reduce the size of events, next, we focus on reducing the frequency of events. This can be particularly efficient for high frequency industrial event streams especially in situations when a machine is idle or performing standard cycles. In the following we present different techniques for frequency reduction:

- **Send on Change** Often, data is transmitted at a static sampling rate that generates one event after another at a fixed time interval. Thus the receiver iteratively receives an event with the current value of the sensor instead of only state changes. This also means that redundant information is transmitted. Therefore, instead of sending at a fixed sampling rate, events can be emitted when the value of the sensor changes. This is especially efficient in inertial systems where values do not change very quickly. In the following example (Figure 7.6), it can be seen that only at time t=1 and t=4 the event is transmitted instead of all five measurements. This can be easily realized for simple events, but becomes more complicated for more complex event structures (e.g. nested properties). Then it must be decided which data to transmit. In such scenarios, the next technique *information buffer* can be helpful.

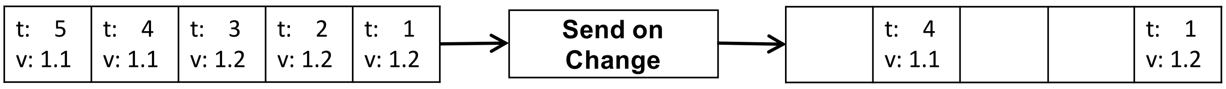


Figure 7.6.: Example for send on change, event is only transmitted when the value v changes

- **Information Buffer** With an information buffer, an event can be reconstructed at the receiver without sending all information by using previously transmitted values stored in a buffer. This buffer is located at the receiver and usually holds the last transmitted value for each field. It is also possible to use more complex buffers and reconstruction techniques. The example in Figure 7.7 uses the same sensor values as before for *send on change*, but this time with an additional *information buffer* at the receiver. As before only the events at time t=1 and t=4 are transmitted, but the resulting event stream contains values for each time step because of the used buffer. This technique allows an event stream to be processed at a constant frequency while reducing data, especially if the values change infrequently.

- **Quality Reduction** A third technique to reduce the frequency is *quality reduction*. So far, the receiver got the same information as the sender, but in many cases this is not necessary. Therefore, the quality of the event stream can be reduced by not transmitting all the information. Figure 7.8 depicts such a situation, where the frequency of the event stream is halved by only transmitting every second event. Simple strategies can be used as shown in the example, or more complex ones like

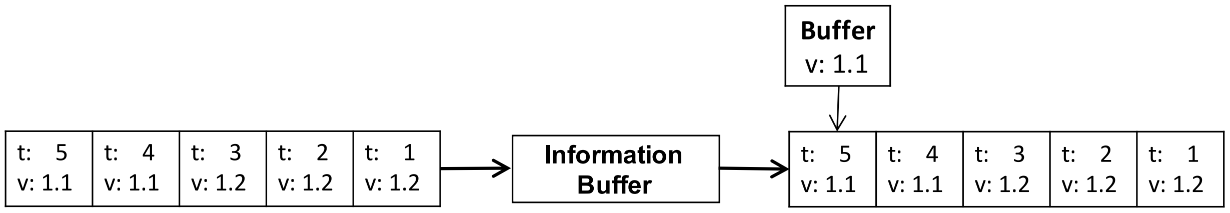


Figure 7.7.: Example for information buffer, the receiver contains a buffer to reconstruct values that are not transmitted

aggregations or thresholds. It depends on the performed algorithm whether the reduction of data quality has an impact on the result or not.

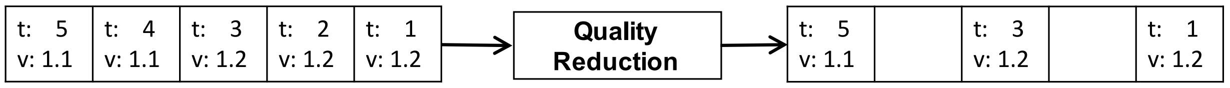


Figure 7.8.: Example for quality reduction, the frequency is reduced by only sending every second event

In this section, different techniques of data reduction on event streams were presented. Those are used as a basis for the stream adoptions within the message broker in Section 7.3 to adapt the event stream. In the following, we present special reduction rules for different data types and semantic types. This is a representative set of reduction rules that can be extended.

7.2.2. Reduction Rules

In this section, it is shown how to apply rules on events to reduce their size without losing relevant information for the subscriber. An event always consists of multiple event properties as introduced in Section 6.2.1. To reduce and reconstruct an event, a reduction rule has to be applied on all of those properties when transmitting the event. First, four different data type rules are introduced which are used as a basis if no information about the semantic type is provided. For all of the four types, a loss-free reduction is possible. If additional information about the semantic type is available this can be leveraged to apply specific adaptations to this event property. All those rules define whether a value must be transmitted or not. A receiver uses a previously introduced *information buffer* to reconstruct the values.

Data Type Rules

Four primitive data types are supported (*Boolean*, *Number*, *Enum*, and *String*) which are explained in more detail in the following:

- A **Boolean** property can take two different values (*true* or *false*). Since there are only two possible values, it must only be transmitted if it changes.
- For **Strings**, reduction is more difficult, because often they do not contain redundant information. Usually strings are compressed before they are sent, as suggested in [Doblander et al. 2016], where a shared dictionary is used within a publish-/subscribe system. However, if some domain knowledge is available, this can also be leveraged for reduction. One example would be if there are a fixed number of static error messages of a machine. Instead of transmitting the error string a mapped status code could be used.
- **Enums** have a fixed number of possible values, i.e. they can be mapped to numeric values. If new enum values can be added during runtime, it must be possible to communicate these new values to both publisher and subscriber.
- The greatest potential for data reduction within industrial event streams exists for **Numerical** values, since they occur most frequently and they can be approximated. This is especially promising when values do not change much and the accuracy of the values is not important. Numerical values are not always transmitted, instead a receiver is able to reconstruct them on the basis of previously transmitted values with a prediction function.

Semantic Type Rules

The previously presented data types only indicate how data is represented and not what meaning it has or how it should be interpreted. For instance, the representation of a timestamp can vary greatly. It can be represented as a string (e.g. ISO 8601) or as a number (e.g. UNIX time[2]). To provide this information, the event model contains the information about the semantic type of a property. Based on those semantic types, new reduction rules can be defined that can then be applied on the event streams. These are able to interpret the data and treat it differently depending on the application. Thus, further structures can be exploited to optimize the data transmission for the respective use cases. In this section, three examples for such rules based on the semantic type are introduced:

- **Geo coordinates** are represented as numerical values. These values have different meanings depending on the used coordinate system. This can be defined via the semantic type, for example the latitude value is uniquely determined by the identifier wgs84:lat[3]. When geo-positions are used in event streams, they often

[2]dbpedia:Unix_time

[3]http://www.w3.org/2003/01/geo/wgs84_pos#lat

represent moving objects. The movement patterns can be exploited to predict where the object is likely to move. If this prediction is accurate enough, the value does not have to be transmitted, but can be calculated by the receiver based on the old values. Several data compression techniques and algorithms for geo data are presented in [Meratnia and By 2004] that leverage the characteristics of geo data to reduce its size. The example in Figure 7.9 shows the altitude values of a vehicle driving up a hill. On the left side, the original values are shown and it can be seen that they initially increase constantly by 0.1. On the right side are the values that are transmitted. Since the increase between t=1 and t=4 is constant, data does not have to be transmitted, instead, the receiver can interpolate those values. Only at t=5 the value is sent again, because this value does not correspond to the previous pattern.

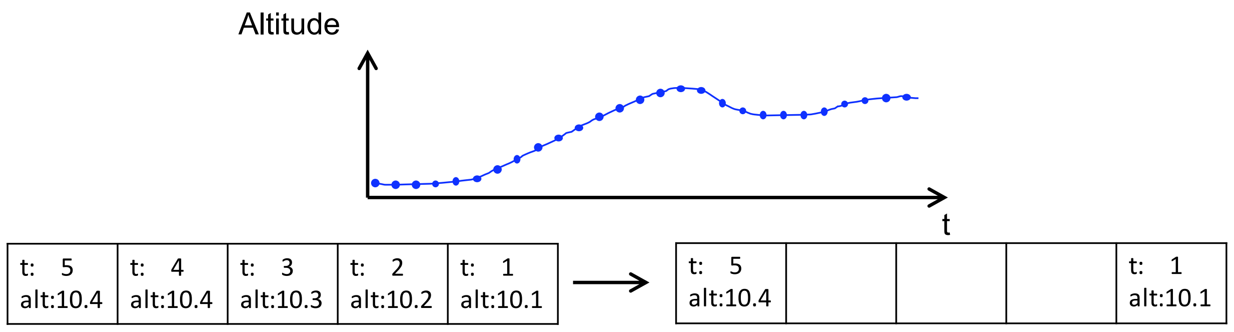
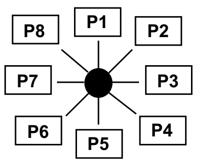


Figure 7.9.: Example for data reductions on geo data

- **Lidar** sensors are used to measure the distance to surrounding objects, often used within robotics for example to identify obstacles. Figure 7.10 shows a simple lidar on the top, that measures the distance to the next obstacle at eight different points. Each of those measurements is represented as a numerical value within an array, as illustrated at the bottom. The position within the array represents the measurement of a certain angle. There may be uncertainties between two measurements, for example due to dust particles in the air. However, measurements are often performed at a high frequency, so several measurements can be used to compare whether the value has actually changed before it is transmitted. In this example, only a very simple lidar was presented which scans two dimensions, however other models have a much higher resolution and can also cover three dimensions.

- **Power consumption** is also usually represented as a numeric value. For this number, it is often not important to recognize small fluctuations, because the consumption typically stays constant for longer periods of time. For example, it can be interesting to recognize how long a certain machine runs or in which production state the machine currently is. Also if the value is almost zero it is most likely that the machine is currently idle or turned off and small changes are not relevant in such cases. For some machines there may be additional patterns in the data which are not relevant for the evaluation, as shown in the example in Figure 7.11. When the



P1	P2	P3	P4	P5	P6	P7	P8
2.2	2.1	3.0	2.9	4.1	3.2	5.8	4.4

Figure 7.10.: Example for representation of lidar data

machine is idle, some small peaks can be seen (t=4 and t=26). Peaks occur because the machine needs to maintain a base temperature. The pattern of the production process is between t=8 and t=21. For the evaluation of this event stream the peaks are not relevant and can be ignored. In this example, it means that only changes that are measured over two time steps will trigger a change in the value. This will slightly increase the latency, but significantly less data will need to be transmitted since the machine is only active for a short time period with long idle times in between.

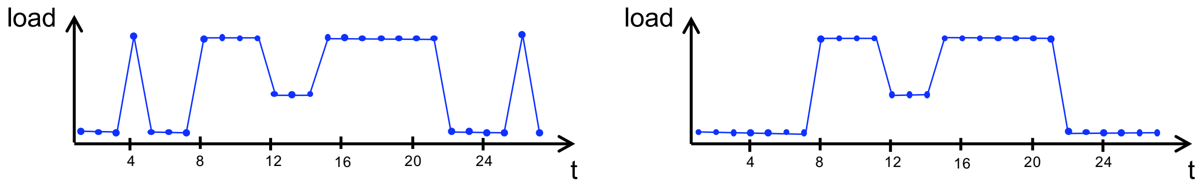


Figure 7.11.: Example for data reductions on power data

7.3. Message Broker

In this section, it is shown how the previously introduced techniques can be used and integrated into a wrapper around existing topic-based message brokers to dynamically adapt industrial event streams to reduce the network traffic.

This approach allows to leverage the functionality of already industry grade message brokers as it was analyzed in [Sommer et al. 2018]. In Chapter 6 (Ingest), a semantic meta-model was introduced. This is used additionally to the configurations provided by a user to automatically adapt event streams. Figure 7.12 shows an overview of the different components which are explained in detail the following subsections. The central component in the figure is the *schema registry*. It contains the current status of the system, i.e. which publishers and subscribers exist and what the respective event schema χ for

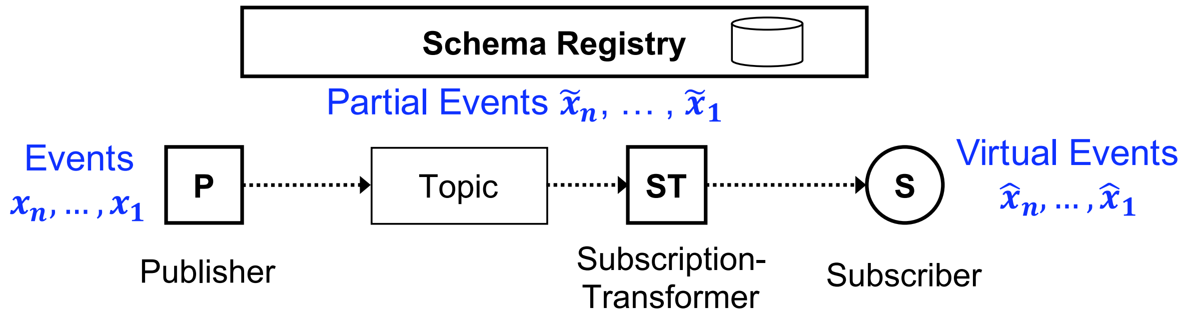


Figure 7.12.: Schematic overview of the different components of the broker

the different event streams looks like. On the left side are the events x_t emitted by the data source at *time t*. Next, there is the publisher reducing the event properties x_t^p and only transmits *partial event* $\tilde{x}_t$ to the topic. These events usually do not contain values for all properties of the event schema and are therefore smaller. The event stream is prepared individually for each subscriber in a *subscription-transformer* by reducing the event properties even further. The *subscriber* on the right takes the *partial events* as an input and reconstructs all (not transmitted) event properties with the *prediction* P. This results in *virtual events* $\hat{x}_t$ which are then consumed by the algorithm that uses the subscriber. The transmitted events and those stored within the topic of the broker are only partial events. This reduces both the bandwidth and required storage space, while exploiting the local computing power and memory of edge devices to calculate the values that are not transmitted. The simplest strategy is to transmit properties only when their value changes. Then the *window size* is $w = 1$ and the *prediction* P uses the last value x_{t-1} for the new value $\hat{x}_t$. With the *error function* e, the error is calculated and it is determined whether this error is bigger then the error *threshold* $\varepsilon = 0$. If that is the case the value is transmitted, otherwise the subscriber performs the same prediction to reconstruct the value. Thus, the algorithm has the data in the original frequency available for evaluation although less data is transmitted. Table 7.1 provides an overview of the notations used throughout this chapter.

7.3.1. Schema Registry

The *schema registry* is the central component of the system that controls all the other components. It stores which subscribers exist for each publisher and how the event schemas look like. This knowledge can be used to make optimizations in the transmission of the event stream that would otherwise not be possible. For example, the data does not have to be transferred as long as there is no active subscriber or in case certain properties are not required by the subscriber, they are also not transmitted. The additional communication that takes place might effect the response time while starting and stopping

x_t	event with all properties at *time t*
x_t^p	actual value of event *property p* at *time t*
$\hat{x}_t$	virtual event at *time t*
$\tilde{x}_t$	partial event at *time t* sent and stored on broker
P	prediction $P(x_t) = \hat{x}_t$
w	window size $\{w \in N \mid w > 0\}$
X_t^p	vector of last w values of *property p*
e	error function $e(x_t^p, \hat{x}_t^p)$
ε	threshold value $\{\varepsilon \in R \mid \varepsilon >= 0\}$
δ	decision rule, returns true or false
χ	semantic schema of event
ζ_{sub}^p	configuration for *subscriber sub (publisher pub)* of event *property p*

Table 7.1.: Notations of variables used throughout this chapter

publishers and subscribers. However, this effect does not have a great impact since these usually run over a long period of time and are not constantly changed. This is managed by the message broker and has no effect on the client code of the publisher or subscribers. So from the client's point of view there is no knowledge about the subscribers as in existing topic based systems.

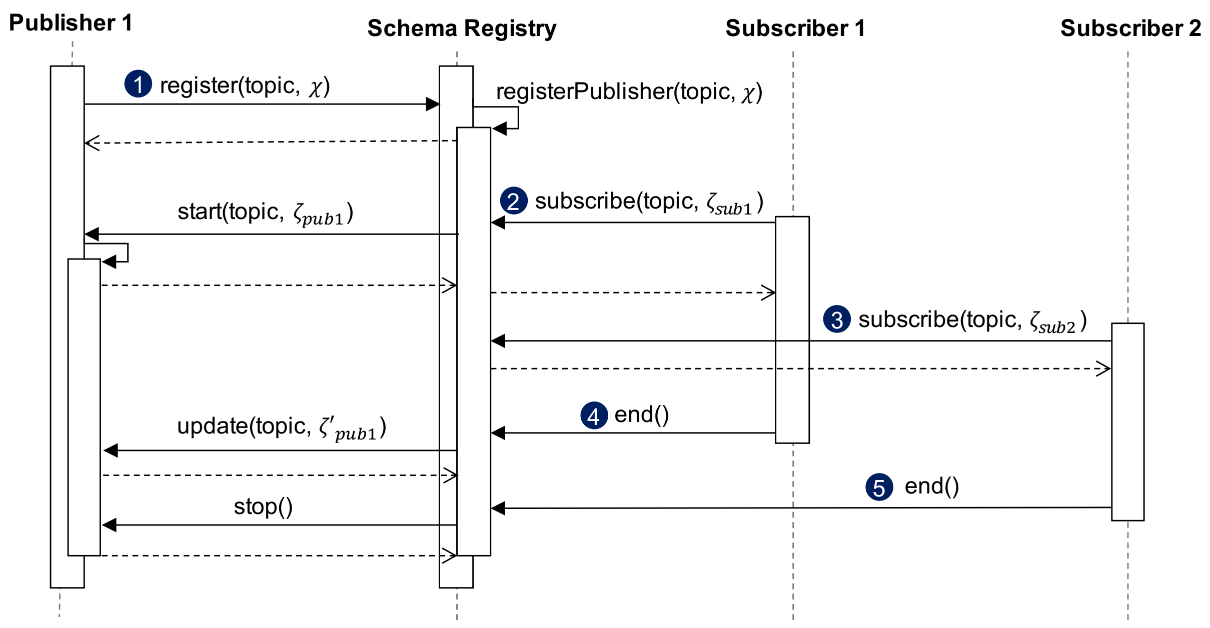

Publisher 1
Schema Registry
Subscriber 1
Subscriber 2
1 register(topic, χ)
registerPublisher(topic, χ)
start(topic, ζ_{pub1})
2 subscribe(topic, ζ_{sub1})
3 subscribe(topic, ζ_{sub2})
4 end()
update(topic, ζ'_{pub1})
stop()
5 end()

Figure 7.13.: Sequence diagram to illustrate the functionality of the schema registry

Figure 7.13 shows the interaction between the different components and the schema registry. The sequence diagram shows how one publisher is started and then two subscribes which consume the events. The schema registry takes care of all the administration and communication while the event stream is transferred via the underlying message broker. A publisher registers itself at the schema registry by transmitting the *topic* and the *event schema* χ ❶. The information of the publisher is stored and it is waited till the first subscriber subscribes to the topic, only then the publisher does send any data. At ❷, the first subscriber is started by providing the *configuration* ζ_{sub1} describing the requirements for the event stream. The schema registry triggers the publisher to start publishing the events using the provided configurations and also a *subscription-transformer* is started. The next section explains how the configuration influences the event stream. After some time the second subscriber is registered ❸ with its configuration ζ_{sub2}. Therefore, a new *subscription-transformer* is started with the new configuration. This does not affect the publisher because subscriber 2 requires the event stream in a lower granularity. When the subscriber 1 is stopped at ❹, also the first subscription-transformer is stopped. Since now only subscriber 2 is subscribing the data of publisher 1, the configurations are updated. Finally, at ❺ the second subscriber is stopped as well. This stops the subscription-transformer at the broker and the data transmission of the publisher. So far, the functionality of the system and the schema registry was explained, next the publishers and subscribers are introduced in more detail and how they use reduction rules to adopt the event stream during runtime.

7.3.2. Publisher / Data Producer

In this section the publisher is introduced that provides a simple API that can be used to transmit events to a broker. A client calls the *publish* method from the API with an event x_t at $time\ t$. Within this method, the *reduceEvent* routine is called, that reduces the event according to the configuration ζ_{pub} provided by the schema registry. Event properties are removed according to the provided configuration parameters, which also defines the precision of the resulting virtual events at the end. The reduction within the publisher sets the minimum quality for the resulting virtual events that are recreated by subscribers. After the event is reduced into a partial event, this partial event is transmitted to the broker. Once the configurations ζ_{pub} change, the producer sends one full event to reset the recreations at the subscriber and then starts applying the new reduction configurations.

In Listing 7.1, the definition of the *reduceEvent* routine can be seen. Within this routine an *event* x_t at *time t* is passed in. Line 2 iterates over all properties of the current event and loads the configurations for this property in line 3. The configuration ζ^{p}_{pub} is a tuple of four consisting of the *decision rule* δ, the *prediction function P*, the *error function e*, and the error value ε. Within the *decision rule* δ, it is defined whether the current value should be transmitted or not. This decision rule is explained in the next subsection in more detail. It compares the result of the *prediction function P* with the real value of the property. The *error function e* defines an error that is calculated between those two values, if this error is higher, then the defined ε for this property the value of the original event is added to the resulting partial event $\tilde{x}_t$. All those variable parameters are transmitted by the schema registry within ζ_{pub}. Once the decision rule is evaluated for all properties the resulting partial event is transmitted over the network.

```
reduceEvent($x_t$)
  FOR ALL  $x_t^p$  IN  $x_t$  DO
    ($\delta$,$P$,$e$,$\varepsilon$)  $\leftarrow \zeta_{pub}^p$
    IF ($\delta$ ($x_t^p$,$P$,$e$ ,$\varepsilon$))
      $\tilde{x}_t^p$.add($x_t^p$)

  RETURN $\tilde{x}_t$
```

Listing 7.1: Reduce events before transmitting them over the network

7.3.3. Decision Rule

For each event property a *decision rule* δ is specified that decides when a value must be transmitted. This is either based on the semantic type (if defined) or based on the data type otherwise. In Listing 7.2 the algorithm for such a decision rule is illustrated. It is the main decision rule mostly used in this work, but it can also be replaced with another one. First, in line 2 the prediction of a value x_t^p of *property p time t* is calculated and stored in $\hat{x}_t^p$. Then, the error function e is used in line 3 to calculate the error between x_t^p and $\hat{x}_t^p$. This can be a simple error function like the absolute difference between the two values, but it can also be a more complex function. When the error value is higher than the threshold ε, the decision function returns true, meaning the property value must be transmitted and false otherwise. Within the prediction, often a state is hold for each property, usually of a fixed window size w, which is used as a basis for the predictions.

```
1 $\delta$ ($x_t^p$,$P$,$e$ ,$\varepsilon$)
2   $\hat{x}_t^p \leftarrow P(x_t^p)$
3   err $\leftarrow e(x_t^p, \hat{x}_t^p)$
4   IF (err >= $\varepsilon$)
5     RETURN true
6   ELSE
7     RETURN false
```

Listing 7.2: Decide whether a property value must be transmitted or not

7.3.4. Subscriber / Data Consumer

The last two components that are missing are the subscription-transformer and subscribers. Both work similarly, which is why they are explained together in this section. In the pseudo code, only the basic functionality is explained, further checkpoints to reconstruct events for new subscribers would be required which are neglected here for the sake of simplicity. For each subscriber, a subscription-transformer is started within the broker to prepare the events for this specific subscriber. The subscriber specifies the configuration parameters ζ_{sub} which are then distributed by the schema registry as described before. Listing 7.3 show the algorithm that is used within the subscription-transformer to prepare the subscriptions. The algorithm runs within the broker and gets the partial events $\tilde{x}_t$ sent by the publisher as an input. First in line 2 the virtual event is reconstructed via the *expandEvent* routine, which is further explained later in this section. Then, the event is reduced to a partial event $\tilde{x}'_t$ again, according to the configurations of the subscriber. It is important that $\tilde{x}_t$ and $\tilde{x}'_t$ are in general not the same, only if the publisher and subscriber configuration are the same, both events are equal. Once the new partial event $\tilde{x}'_t$, is calculated it is transmitted to the subscriber in the *sendToSubscriber* function.

```
1 prepareSubscription($\tilde{x}_t$)
2   $\hat{x}_t \leftarrow$ expandEvent($\tilde{x}_t$)
3   $\tilde{x}'_t \leftarrow$ reduceEvent($\hat{x}_t$)
4   sendToSubscriber($\tilde{x}'_t$)
```

Listing 7.3: Prepare the events for specific subscribers

Listing 7.4 shows the *expandEvent* routine that is used within the subscribers and subscription-transformers to reconstruct virtual events. Therefore, the event schema χ and the configuration parameters ζ_{sub} are required. In line 3, it iterates over all properties stored in the event schema, provided by the schema registry. For each of the properties it is checked whether they are part of the partial event $\tilde{x}_t$ or not. If the property was transmitted within the partial event the value is added to the resulting virtual event $\hat{x}_t$. When the value was not transmitted the prediction function from the configuration is used on the *property p* to create a value based on the previously transmitted values for this property. This is the same prediction function that was used before to decide whether the property value should be transmitted or not. In line 8, the resulting virtual event is returned consisting of transmitted and predicted values.

```
1 expandEvent($\tilde{x}_t$)
2   FOR ALL $\chi^p$ IN $\chi$ DO
3     IF exists($\chi^p$, $\tilde{x}_t$)
4       $\hat{x}^p_t \leftarrow \tilde{x}^p_t$
5     ELSE
6       $\hat{x}^p_t \leftarrow P_{\zeta_{sub}}$ (p)
7
8   RETURN $\hat{x}_t$
```

Listing 7.4: Reconstructs a virtual event based on the configurations

The last routine is the *subscribeEvent* algorithm, running at the subscriber and shown in Listing 7.5. It receives the partial event created by the *prepareSubscription* at the broker and the virtual event is then created by the previously described *expandEvent* routine. $\hat{x}_t$ is the final virtual event that is returned in line 3 to the algorithm subscribing to the topic. The extent to which this value differs from the original measurement depends heavily on the configuration of the publisher and the subscriber. This ensures that our main goal is achieved, which is to allow different subscribers to receive the data at different quality levels depending on the subscribers processing purpose.

```
subscribeEvent($\tilde{x}_t$)
  $\hat{x}_t \leftarrow$ expandEvent($\tilde{x}_t$)
  RETURN $\hat{x}_t$
```

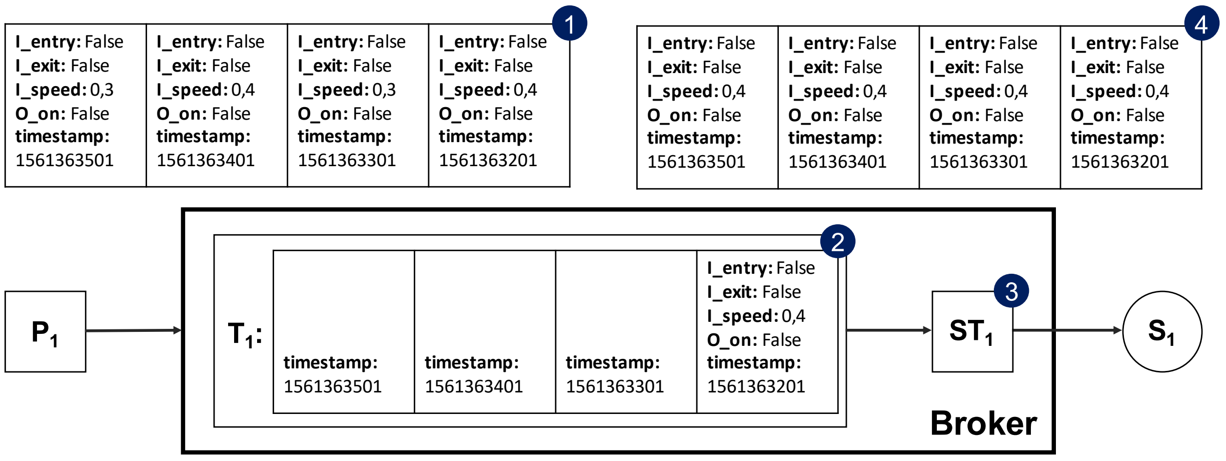
Listing 7.5: The subscriber function receiving the partial events

7.4. Running Example

In this section, two real-world examples are used to illustrate how the presented approach for a message broker that dynamically adapts the event streams works. The use cases are based on the event streams from the running example presented in Section 3.2. First, data produced by the PLC of the conveyor, that was connected in Section 6.5, is monitored. For this purpose, small changes in the values during operation are not relevant, since these can always fluctuate slightly, but it is important to detect larger changes. In the second example, data from the screwing station is transmitted and consumed by two different subscribers. The first subscriber stores the data in a database and the second calculates how long the machine is active. This illustrates how the system supports different levels of quality for multiple subscribers.

For both examples, the timestamp is transmitted in each event. A simple function is used for the prediction that assumes the value did not change $P(x_t) = x_{t-1}$, therefore a state is required to store this value. The decision rule is used from Listing 7.2 and for the *error function* the absolute error between the prediction and actual value is calculated $e(p_1, p_2) = |p_1 - p_2|$. These configurations are the same throughout this section, only the threshold value ε differs between the respective publisher, subscription-transformer, and subscriber. It is assumed that the value ε is the same for all numerical properties within the events. It would also be possible to have different configurations for the different properties.

Figure 7.14 shows the first example using the conveyor data. The data is read with 10 events per second (10 Hz) from the PLC and the timestamp is represented as a UNIX timestamp. On the left, is the publisher P_1 sending the events and on the right, is the subscriber S_1 receiving the events and monitoring the current state. At ❶, the original events can be seen. They are sorted from right to left with the first on the far right. Each event contains three boolean measurements, two representing the state of light barriers (I_entry, I_exit), one showing the state of the motor (O_on), and one numerical value (I_speed) indicating the current speed of the conveyor belt. The error threshold is set to $\varepsilon = 0.1$ for the subscriber. Since it is the only active subscriber, this value is set for all components. At ❷, the partial events can be seen that are transmitted. Since the changes in the values are below the error threshold, only the first event must be transmitted fully, for the other events only the timestamp is transmitted. In the subscription-transformer ❸ the events are not changed, since the configuration parameters are the same as for the publisher. At ❹, the resulting virtual events at subscriber S_1 can be seen. They contain all the properties and introduce a small error to some of the values of the property I_speed.



1
I_entry: False I_exit: False I_speed: 0,3 O_on: False timestamp: 1561363501
I_entry: False I_exit: False I_speed: 0,4 O_on: False timestamp: 1561363401
I_entry: False I_exit: False I_speed: 0,3 O_on: False timestamp: 1561363301
I_entry: False I_exit: False I_speed: 0,4 O_on: False timestamp: 1561363201
4
I_entry: False I_exit: False I_speed: 0,4 O_on: False timestamp: 1561363501
I_entry: False I_exit: False I_speed: 0,4 O_on: False timestamp: 1561363401
I_entry: False I_exit: False I_speed: 0,4 O_on: False timestamp: 1561363301
I_entry: False I_exit: False I_speed: 0,4 O_on: False timestamp: 1561363201
P1
T1:
timestamp: 1561363501
timestamp: 1561363401
timestamp: 1561363301
2
I_entry: False I_exit: False I_speed: 0,4 O_on: False timestamp: 1561363201
3
ST1
S1
Broker

Figure 7.14.: Example data transmission for the data of the conveyor

Figure 7.15 shows the second example with an event stream from the screwing station in the example production line. The data is produced at 1000 Hz and, for a better overview, the timestamp is represented with a counting value called t within the figure instead of a UNIX timestamp. In this use case there is one publisher P_2 publishing on topic T_2 and two subscribers exist. The events at ❶ have five properties, the timestamp t, the load of how much energy the machine currently consumes and x, y, z values from the acceleration sensor. Subscriber S_2 subscribes with $\varepsilon = 0.0$ to store the data in a database for later offline analytics and S_3 uses $\varepsilon = 0.2$. S_3 calculates how long the machine is active, therefore only the load value is relevant and the x, y, and z values must not be transmitted because they are not read from the event. Also small changes are not important, only big changes of the load value are relevant to determine whether the machine is turned on or off, as can be seen in the big increase of the load value between t=0.003 and t=0.004. The publisher uses $\varepsilon = 0.0$ because this is required by S_2 and this already reduces the amount of information that must be transmitted to the broker ❷. At ❸ those partial events are forwarded and the virtual events are recreated at ❹. In ST_3 the $\varepsilon = 0.2$ is used for S_2, which reduces the events even further ❺. The load values of the resulting virtual events at ❻ are slightly different than the original events and those at ❹, but this is sufficient to perform the task to detect whether the machine is running or not. Previous research has shown that data can be reduced at edge nodes without compromising the accuracy of analytical approaches too much [Harth and Anagnostopoulos 2017].

In this section, we showed two different examples for event streams from our motivation scenario and how they are realized with the concepts and approaches presented in this chapter.

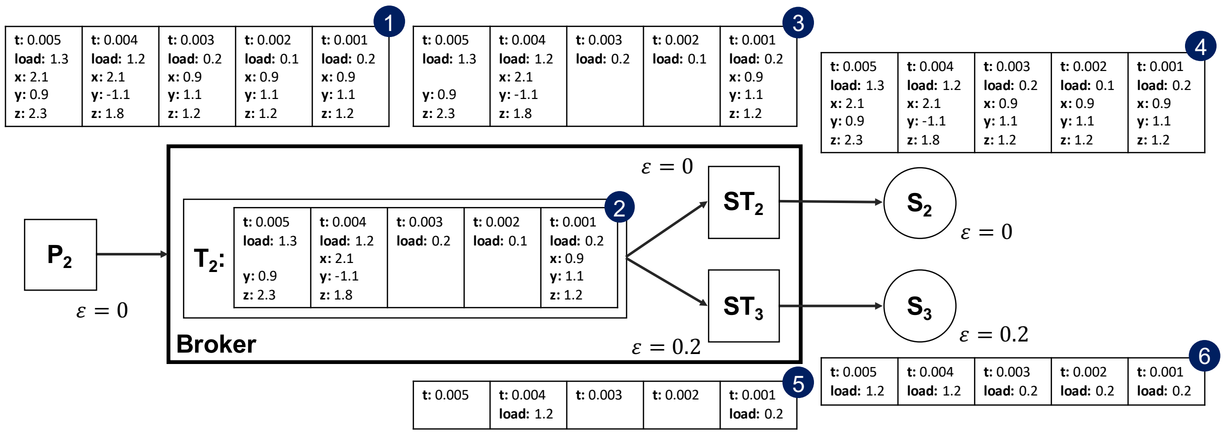

1
t: 0.005 load: 1.3 x: 2.1 y: 0.9 z: 2.3
t: 0.004 load: 1.2 x: 2.1 y: -1.1 z: 1.8
t: 0.003 load: 0.2 x: 0.9 y: 1.1 z: 1.2
t: 0.002 load: 0.1 x: 0.9 y: 1.1 z: 1.2
t: 0.001 load: 0.2 x: 0.9 y: 1.1 z: 1.2
3
t: 0.005 load: 1.3 y: 0.9 z: 2.3
t: 0.004 load: 1.2 x: 2.1 y: -1.1 z: 1.8
t: 0.003 load: 0.2
t: 0.002 load: 0.1
t: 0.001 load: 0.2 x: 0.9 y: 1.1 z: 1.2
4
t: 0.005 load: 1.3 x: 2.1 y: 0.9 z: 2.3
t: 0.004 load: 1.2 x: 2.1 y: -1.1 z: 1.8
t: 0.003 load: 0.2 x: 0.9 y: 1.1 z: 1.2
t: 0.002 load: 0.1 x: 0.9 y: 1.1 z: 1.2
t: 0.001 load: 0.2 x: 0.9 y: 1.1 z: 1.2
P_2
$\varepsilon = 0$
T_2:
2
t: 0.005 load: 1.3 y: 0.9 z: 2.3
t: 0.004 load: 1.2 x: 2.1 y: -1.1 z: 1.8
t: 0.003 load: 0.2
t: 0.002 load: 0.1
t: 0.001 load: 0.2 x: 0.9 y: 1.1 z: 1.2
$\varepsilon = 0$
ST_2
ST_3
$\varepsilon = 0.2$
Broker
S_2
$\varepsilon = 0$
S_3
$\varepsilon = 0.2$
5
t: 0.005
t: 0.004 load: 1.2
t: 0.003
t: 0.002
t: 0.001 load: 0.2
6
t: 0.005 load: 1.2
t: 0.004 load: 1.2
t: 0.003 load: 0.2
t: 0.002 load: 0.2
t: 0.001 load: 0.2

Figure 7.15.: Example data transmission for the data of the screwing station

7.5. Summary

In this chapter, it was shown how industrial event streams can be reduced and integrated via a message broker in a way that they are dynamically adapted according to the requirements of the subscribers. First, a high-level overview of the structure of the content of this chapter was given in Section 7.1 (Walkthrough). Then, several event reduction techniques were presented in Section 7.2. These are divided into two categories, depending on whether they reduce the event size or the event frequency. Next, an extensible set of reduction rules was presented, which can be defined either on the basis of the data type or on the basis of the semantic type. The information about the data type only allows to perform basic reduction strategies to ensure that the values are transmitted correctly. When a semantic type is available, it is possible to leverage additional information about patterns in the data, e.g. removing peaks from power data of a machine. In Section 7.3, concepts for a wrapper around topic-based message broker were introduced. Therefore, an approach was used that leverage knowledge about the event schema as well as requirements provided by each subscribers. The previously introduced concepts for data reduction are used to dynamically adjust events. The schema registry is the central place, which has an overview of all current publishers and subscribers, as well as the schema of the respective event streams. The publisher receives the configuration parameters and the data is adjusted accordingly. This also ensures that data is only transmitted if there is a subscriber that subscribes to the event stream. A publisher only forwards partial events that do not contain the values of all properties. Based on previously received events, a subscriber is able to reconstruct virtual events. Values that are not transmitted are predicted based on previously transmitted values. We introduced a schema registry and explained how publishers and subscribers register themselves. Then the routine reducing the data and running within the publisher was explained, therefore it uses a decision rule, whether the value of a property should be transmitted or not. After that, it is explained how the subscribers work and how they reconstruct events. The subscription-transformer is running within the broker and prepares the events for a specific subscriber, which creates the resulting virtual events. In the end of the chapter two use cases from the running example were used to illustrate how the whole approach works in different settings. With the method presented in this chapter, it is possible to dynamically adjust event streams and reduce the amount of transmitted data, while ensuring that the analytics algorithm has sufficient information. In the next chapter it is shown, how supervised machine learning models can be trained automatically based on the integrated data.

8 Analyze

In the previous chapters, we introduced methods to ingest and integrate data of multiple industrial assets. In the following, we explain how we enable citizen data scientists to automatically train and deploy Machine Learning (ML) models for industrial event streams. The goal is to apply supervised ML methods even with little data science knowledge.

Section 8.1 presents a walkthrough that gives a high-level overview of the approach and the developed concepts and components. After that, we specify the concept of a Machine Learning Pipeline Agent (MLPA) and define different types of ML tasks that can be trained to analyze industrial event streams (Section 8.2). Further, it is explained how a guided workflow facilitates citizen data scientists to specify the training tasks. Based on this task description, a ML service can be triggered that is capable of automatically training ML models (Section 8.3). No additional preprocessing is required for the training and the resulting models can directly be deployed on industrial event streams. In the end of this section, different tool support realizing the introduced concepts is illustrated. Section 8.4 showcases, based on data of the CNC machine, how a ML model can be trained and deployed. In the end of this chapter (Section 8.5) the main contributions are summarized.

8.1. Walkthrough

In this section, it is illustrated how ML models can be trained on previously ingested and integrated event streams. Figure 8.1 presents an overview of the different components that we use to facilitate citizen data scientists to analyze industrial event streams themselves.

In the beginning, a pipeline is used to collect data and continually persist it in a time-series database. The training for new models usually starts with the creation of different visualizations of the collected data ❶. Additionally, it might be necessary to provide labels manually that can later be used for the training of supervised ML models. Once the data is prepared, the *training service* ❷ can be used to describe the ML task. This is accomplished in a guided process, where the citizen data scientist configures the learning task. Once the configuration is complete, a training job is triggered automatically. During

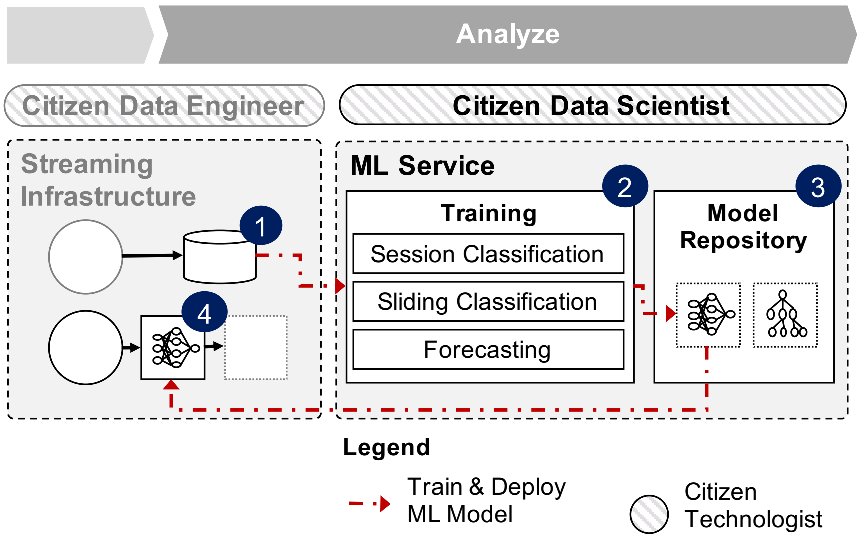


Figure 8.1.: Overview of the analyze approach targeted at citizen data scientists

the training, it is possible to monitor the status of the training job. After the training is completed, the best models are stored in a model repository ❸. A citizen data scientist can evaluate these models visually, e.g. by checking how well they performed or by inspecting the model parameters. The best model can be selected to be deployed on event streams in a pipeline as illustrated ❹. Changes in the event streams might require the reconfiguration or retraining of a model, to ensure high-quality results [Rehman et al. 2019]. Therefore, the event stream is constantly monitored and the citizen data scientist will be notified in case a change is detected and a model might need additional training. With this ML Service and the guided approach, many ML models can be trained without much manual effort and little technical knowledge. In the following, the details of the individual concepts are explained.

8.2. Machine Learning for Industrial Event Streams

In this section, it is introduced how ML models are integrated into Event Processing Networks (EPNs). First, three different types of ML problems in the context of stream processing are presented. Then, the workflow for citizen data scientists to define training tasks is explained.

In event processing, events are processed by Event Processing Agents (EPAs) in an EPN, as introduced in Chapter 2 (Foundations). In order to apply ML in such systems, events must be transformed into feature vectors, which are used as an input for ML models. Results of the model must then be retransformed into events to further process them. To realize this, we introduce Machine Learning Pipeline Agents (MLPAs) in this work.

These are used as wrappers around machine learning models to integrate them as generic agents into an EPN. The problem is, that there is not a single generic ML approach, rather many different algorithms exist (e.g. Decision Trees, Deep Learning, ...). To choose the best one depends strongly on the available data, as well as on the application. As an example, condition monitoring requires different approaches depending on whether it is performed on continuous or discrete manufacturing processes [Maier et al. 2017]. In order to apply ML models, they must be created individually for their respective tasks. Usually, data scientists are required and they apply for example the Cross-Industry Standard for Data Mining (CRISP-DM) cycle. To enable citizen data scientists to build such models, we present different types of MLPAs. They can be configured in a guided workflow and can directly be deployed in an EPN.

8.2.1. Definition of MLPA Types

To reduce the complexity of defining models, we support three types of ML problems on event streams as depicted in Figure 8.2. The first one focuses on *forecasting*, e.g. predicting the value of a numerical event property in the future. The second one is a *sliding window classification*, where a sliding window is applied on the event stream and a classification is performed for each window, e.g. monitoring the current state of a machine. The third one is called *session window classification*, where a classification is applied on a session window, e.g. classify a discrete processing step within a machine. Those three types were selected, because in our experience they are commonly needed when developing applications based on industrial event streams. In the evaluation (Section 9.3.2), the experts confirm that those types are reasonable to them from a citizen data scientist point of view. This work focuses on those three problem types, but it is possible to extend it with further learning tasks in the future (e.g. anomaly detection). One requirement for ML in streaming settings is a limitation of memory, meaning not all instances can be stored [Bifet et al. 2018]. Therefore, a window is used to calculate the feature vector for the ML model, which means that the last ω events, where ω is the window size, are used as an input to perform the prediction. This approach considers as well the context of the event.

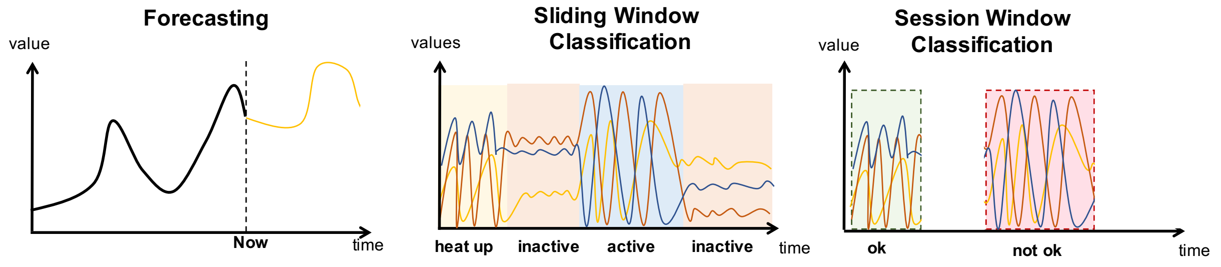


Figure 8.2.: Different types of MLPAs for industrial event streams

Table 8.1 presents the different characteristics of the three types of ML tasks, which are illustrated in Figure 8.2. On the left, the first agent type *forecasting* is shown. A user can select a numeric value from the event schema and define a time offset for this value. Then, a regression model is trained for these parameters. No additional labels are required, because the time-series can be shifted according to the defined *offset*, resulting in a new column with the target variable y. Multivariate time-series predictions are supported too. Once the model is trained, the resulting agent uses the events as an input, calculates a feature vector according to the learned parameters and applies the model. The result is a new event property containing the predicted value that is appended to the event. The *sliding offset* defines the frequency of predictions. If this is set to one, an output is emitted for each event, meaning the output and input frequency of the agent are the same. The next type, *Sliding Window Classification*, is similar to forecasting. Instead of a numerical value, a categorical event property is predicted for the incoming events. This can for example be used to classify the current state of a machine (e.g. heat up, inactive, active) based on a retrofitted acceleration sensor. To train a model for this type, a label must be provided for each event. As before, the output frequency depends on the sliding offset of the sliding window. The last ML type is *session window classification*. In this work, a session is defined as a period in which a categorical value does not change. Additionally, it can be defined which sessions are relevant (e.g. to predict the quality of a produced product for a completed production step). In the example before, the window size for the prediction was fixed, whereas in this case the window size varies according to the session size. Moreover, the data labeling is slightly different, as only one label per session and not per event is necessary (e.g. ok / not ok). When applying the trained agent on an event stream, all events of a session are buffered in memory. Once the session is over, the prediction is performed resulting in one output event.

Table 8.1.: Characteristics of the MLPA types

	Forecasting	**Sliding Window Classification**	**Session Window Classification**
Window Type	Sliding Window	Sliding Window	Session Window
Prediction Type	Regression	Classification	Classification
User Configuration	Prediction Offset, Sliding Offset, Target Property, Feature Properties	Sliding Offset, Target Property, Feature Properties	Target Property, Session Label, Feature Properties
Label	No	Event	Session
Example Application	Forecast Scrap Rate	Classify Vibrations	Monitor Tool Wear

Next, it is shown how the training for those different MLPA types can be configured by a citizen data scientist.

8.2.2. Configuration of MLPA Training Tasks

So far, we introduced three different types of MLPAs. In this section, the process of defining a new ML model by citizen data scientists is explained, which is depicted in Figure 8.3. The operation starts with the selection of the data that should be used to train a model ➊, which is followed by choosing the model type. Depending on the type, different configurations must be provided ➋. In case of forecasting, the forecast offset must be specified, while for the different classifications labels must be provided. Next, the target property for prediction needs to be selected ➌. For classification, this is the property containing the class, and for forecasting it is the numerical property to forecast.

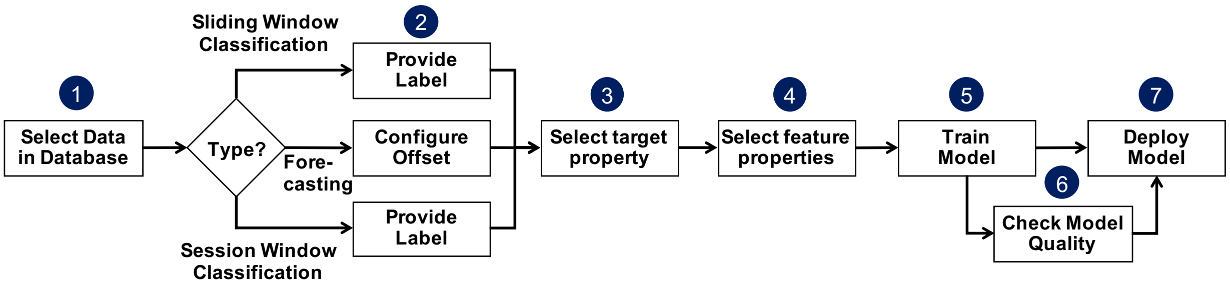


Figure 8.3.: Process steps to configure the training for a new machine learning model

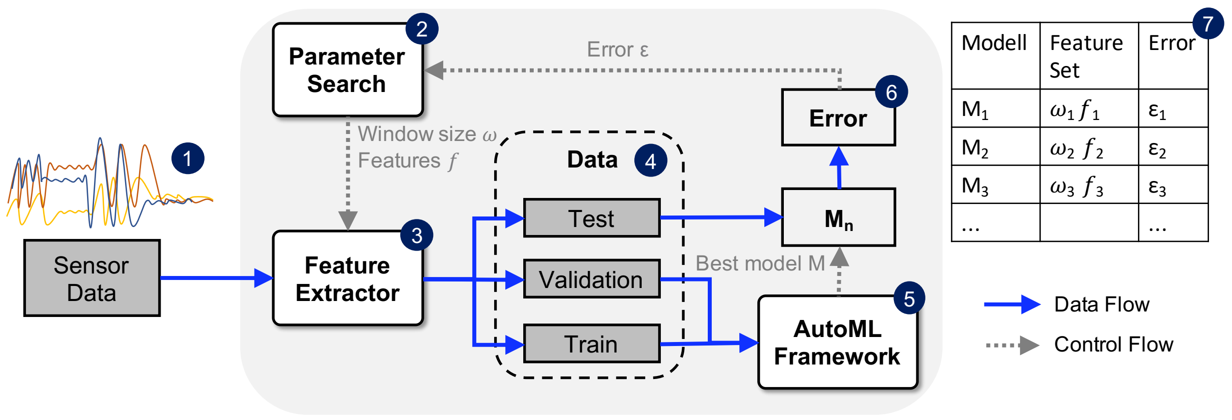
In step number ➍, the event properties must be selected that will be considered for the feature extraction during the training process. At that point, a citizen data scientist can leverage the knowledge about the domain, by only selecting the relevant properties. This step reduces the search space for the parameter search during the training phase. After all configuration parameters are set, the model is trained ➎. The training is completely automated and the user is notified upon completion. Normally, the models found are then ready to be used, however, the results should ideally be reviewed to select the best model ➏. Model parameters, as well as the training performance, can be inspected in detail by the citizen data scientist to verify, if the models work according to the requirements of their use case. Once the results are satisfactory, the model can be deployed ➐ and used on industrial event streams. The goal is to automate most of the configuration, however, for more experienced users there is still the possibility to change advanced configuration parameters manually. This approach enables less experienced users to directly use ML models, while keeping the flexibility for more advanced users.

8.3. Machine Learning Service

So far, the different ML model types for industrial event streams and the configuration of their training tasks were introduced. In this section, it is presented how this configuration is used to train and deploy new models. After explaining how the ML service automatically trains a new model, the concept of deployment within an EPN is shown. In the end of this section, the developed tool support for these phases is presented.

8.3.1. Training Phase

In this subsection, the workflow of the Automated Machine Learning (AutoML) framework for industrial event streams is introduced. As an input, it takes raw sensor values of industrial event streams and generates a ML model for a MLPA. We decided to create a modular framework that uses individual building blocks for the different tasks. Algorithms for the individual building blocks can be exchanged, especially, if new techniques are developed in the future.



Modell	Feature Set	Error
M_1	$\omega_1 f_1$	ε_1
M_2	$\omega_2 f_2$	ε_2
M_3	$\omega_3 f_3$	ε_3
...		...

Figure 8.4.: AutoML workflow to train a machine learning model for event streams

An overview of the training approach is depicted in Figure 8.4. The basic idea is to learn the best features, both the window size and the parameters, for a given event stream and leverage existing AutoML frameworks to perform the model selection. Therefore, existing AutoML pipelines, as introduced in Section 2.3.2, are used once the data is transformed into a tabular form containing the features and the target variable. We have chosen this design, because existing AutoML frameworks (auto-sklearn [Feurer et al. 2019], TPOT [Olson and Moore 2016], or H2O AutoML[1]) lack the ability to directly train and deploy the different types of MLPAs on events. The raw sensor values have to be transformed, before the model can be trained. However, we did not want to realize a new generic AutoML framework, we rather wanted to use existing approaches (e.g. for model selection, hyper parameter tuning) and extend them with the capabilities to use industrial event streams. Therefore, an interface for existing AutoML frameworks was designed, which allows their exchange. It is very likely that new and better frameworks will be developed, since this is still an active research area. Further, there is no single best

[1]http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html (accessed on 04/10/2021)

AutoML system that outperforms all others, as presented in [Gijsbers et al. 2019]. Next, we will explain the seperate steps in the training process depicted in Figure 8.4. The time-series of the sensor values is used as an input ➊. The training-process starts with the *Parameter Search* ➋. The features for the prediction within the AutoML framework do not only contain values from single events, but are rather buffered in windows. Based on these windows, features are extracted and subsequently used as input for the prediction. Since it is not clear in advance which the best features are for a given time-series, the goal is to learn those parameters. The parameter search service is used to find the best window size ω and parameters for the feature calculation f for a given problem. Different algorithms, from basic ones like random search or grid search, to more learning-based approaches, can be used for this parameter search. The results are then used as inputs for the *Feature Extractor* ➌ that extracts the features from the raw sensor data and splits the data into three different data sets ➍. One data set for training of the model, a validation set used by the AutoML framework to validate the found models, and one test set used to rank the best models to find the best overall model. Features are calculated for both, time and frequency domain [Figo et al. 2010]. All available features as well as the window size, define the search space. As the best features for a given time-series are not clear in advance, the goal is to learn these parameters. The feature extraction from the time-series data is executed in parallel on multiple cores. This is possible, because the individual windows are independent from each other, as long as the resulting order is the same as the input order. Once the features are calculated, the train and validation sets are used to trigger an AutoML framework ➎ that requires the training data in a tabular form. For this search, a budget is configured, either based on the available training time, or on the amount of models that should be trained. As explained before, this framework can be exchanged. Once the training of the model is completed, the best model is evaluated using the test set to calculate an error value ε ➏, which is forwarded to the parameter search as a score to rank the selected parameters. All models, as well as selected feature sets are stored ➐ and presented to the citizen data scientist in order to perform a final visual validation of the model. Based on these configurations, it is then possible to initialize a model and perform it on an event stream, which will be explained later in this chapter. To create a generic MLPA and ensure that the event streams contain all the relevant properties, a virtual sensor is defined as an input requirement for the processing agent. This means when using this MLPA, it is ensured that the required values are represented in the event stream and the features can be calculated.

8.3.2. Deployment Phase

This subsection explains how the previously trained models can be deployed and events of the event stream are processed. First, the deployment processes of the model and the interaction of the different services is explained. Afterwards, we focus on the MLPA design and in the end describe the processing of the events for the individual types in pseudo code.

Figure 8.5 shows the process of deployment of a trained model into a streaming service. A citizen technologist, usually a citizen data scientist or a citizen data engineer, can select the model and apply it in a pipeline (EPN). During the modeling process of the analytics pipeline, it is possible to validate if the schema of the event stream is compatible with the requirements of the trained model. If the requirements of the virtual sensor are met, then the model can be invoked on pipeline execution. Therefore, the actual model parameters M, as well as all the other configurations, like the window size ω and features f, are loaded into the execution environment in the *streaming service*. From this point onward, the events of the event stream are analyzed and resulting events are forwarded to the next agent within the EPNs.

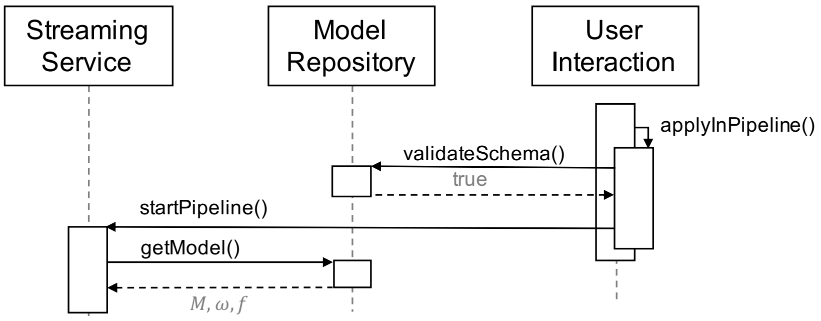


Figure 8.5.: Sequence diagram to deploy a ML model

Figure 8.6 depicts a pipeline that contains a MLPA. The pipeline has two event producers, three agents (including the MLPA), and one consumer in the end. The MLPA takes an event as an input, performs a prediction, and emits an event containing the new property. As shown before, once the agent starts up, the model parameters are loaded from the model repository. For the prediction, context information is buffered within the memory of the agent. The size of this *event buffer* is defined by the parameter ω. Based on these buffered events, features are extracted, according to the configuration defined by the parameters f in the *feature extractor*. These features are then used as input for the ML model M. The result is included in the event, that is forwarded to the next agent in the pipeline. Subsequently, one can choose between two output strategies for the resulting event schema. Either the result is attached to the incoming event or a fixed schema, containing the timestamp and the prediction value, is used for the output.

As the characteristics of event streams can change over time (e.g. data drift), it should be monitored if these changes affect the quality of the predictions. Online learning is often used for training in streaming scenarios, which allows to continuously update the model. However, these cases assume that during runtime, it is possible to find labels that map to the ground truth of the data [Bifet et al. 2018]. In this work, we have chosen to use batch learning techniques because it is not always feasible to constantly create new labels at runtime. However, in order to be able to detect changes the event streams are constantly monitored. When a change occurs, a citizen data scientists is notified and can provide

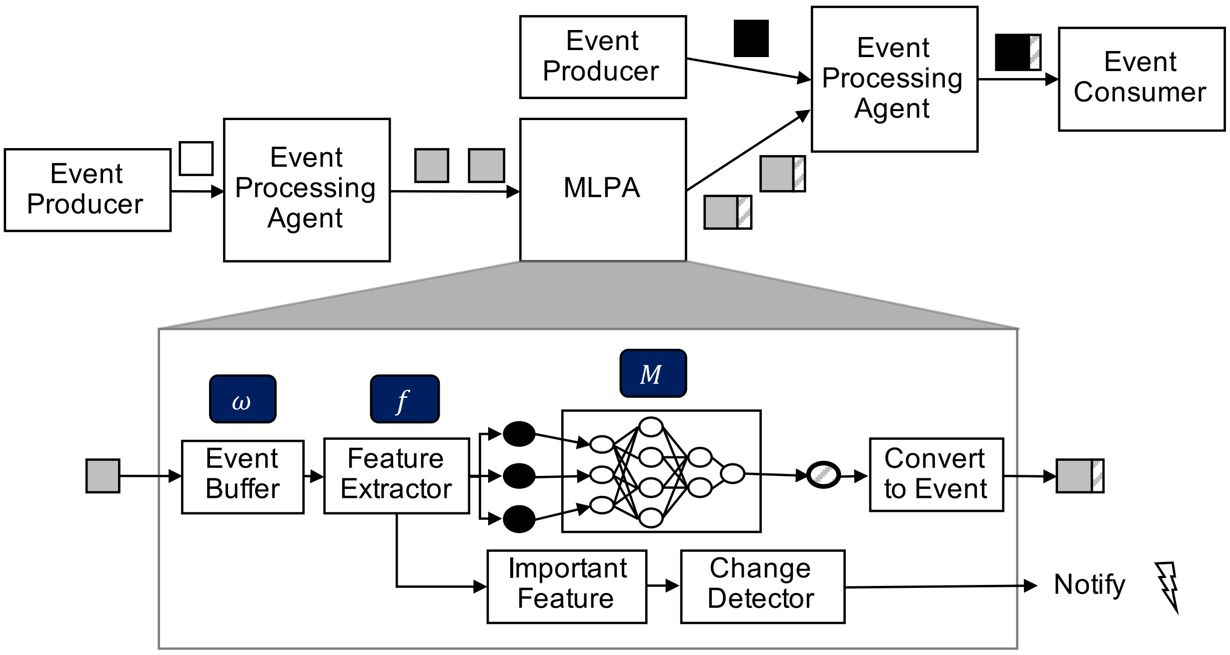


Figure 8.6.: Streaming pipeline containing a ML pipeline agent

x_t	event with all properties at *time t*
x'_t	resulting event containing prediction result
ζ_M	Configuration of machine learning task
M	Machine learning model parameters
ω	window size
f	feature parameters
σ	window slide offset
ρ	property representing session state
β	value of relevant session state

Table 8.2.: Notations for algorithms running in machine learning pipeline agents

new labeled training data and initiate the training for new models with little manual effort. Therefore, it is still possible to quickly update the model, even if online learning is not used. To detect changes, a *change detector* that monitors the most *important features* used by the model is integrated.

The following listings show pseudo code to illustrate how the prediction is performed during runtime and Table 8.2 represents an overview of the different notations that are utilized. The interface has three methods: When the pipeline is started to initialize the agent, the *onInvocation* is triggered. Then, for each event x_t the *onEvent* method is called.

Once the pipeline is stopped, the *onDetach* method is called to clean up everything and stop the agent. Listing 8.1 shows the algorithm for the sliding window classification. It is similar to forecasting, the main difference is the model type used for the prediction method. At invocation time, first the model parameters are loaded from the model repository in line 2. Those parameters include the information about the fitted model M, the window size ω, the selected features f, and the sliding offset σ, which defines how often the prediction is performed. If σ=1, a prediction is performed for each event. If σ=10, it is only done for every tenth event. In line 3, the internal buffer is initialized to store ω events and the offset counter is initialized to 0. For each event, first, the change detection is called (line 7). Then, the event is added to the internal buffer and the slide offset counter is increased. Next, it is checked whether a prediction should be performed (or not) (line 10). If the condition is *true*, first the features are calculated (line 11). Then, the feature parameters are forwarded to the model (line 12) and an inferencing takes place to calculate the label. This label is used to create the final event x'_t (line 13), which is then returned (line 15). Before, the offset counter is reset to 0.

```
 1 onInvocation( $\zeta_{M_1}$ )
 2   $(M_1, \omega, f, \sigma) \leftarrow loadModel(\zeta_{M_1})$
 3   slideBuffer.init( $\omega$ )
 4   slideOffset = 0
 5
 6 onEvent( $x_t$ )
 7    changeDetection( $x_t$ )
 8    slideBuffer.add($x_t$)
 9    slideOffset = slideOffset + 1
10    IF(slideOffset == $\sigma$)
11      features = calculateFeatures( $f$, slideBuffer )
12      label = predictLabel(features)
13      $x'_t$ = createEvent( $x_t$, label)
14      slideOffset = 0
15      RETURN $x'_t$
16
17 onDetach()
18    cleanUp()
```

Listing 8.1: Functionality of prediction and sliding window classification

Listing 8.2 shows the algorithm for the session window classification. This is similar as before, but this time there is no sliding window, rather a session window is applied on the data before the prediction is performed. Configuration parameters are loaded again from the model repository in the *onInvocation* in line 1. The first three parameters M, ω, f, are the same, but instead of the sliding offset, ρ is specified, which is the property of the event representing the state and β, defining the value of the state that should be evaluated.

```
1 onInvocation( $\zeta_{M_2}$ )
2    $(M_2, \omega, f, (\rho, \beta)) \leftarrow loadModel(\zeta_{M_2})$
3
4 onEvent( $x_t$ )
5   changeDetection( $x_t$ )
6   IF ($x_t^\rho == \beta$)
7      sessionBuffer.add($x_t$)
8   ELSE IF ($x_{t-1}^\rho == \beta$)
```

```
 9     features = calculateFeatures( $f$, sessionBuffer )
10     label = predictLabel(features)
11     $x'_t$ = createEvent( $x_t$ , label)
12     sessionBuffer.clear()
13     RETURN $x'_t$
14
15 onDetach()
16    cleanUp()
```

Listing 8.2: Functionality of session window classification

Again, the change detection is used (line 5), to evaluate if a data drift happened. To perform a prediction, all events of the session are buffered. The session is marked as active as long as the property value of ρ is equal to β. Once the state changes, the session window is evaluated. First, the features are calculated in (line 9) and the label is predicted (line 10). The final event is created (line 11), before the buffer is emptied (line 12) and the resulting event x'_t is returned.

This section showed, how the previously trained models can be deployed on an event stream. The next section presents the tool support that we implemented to realize the introduced methods and concepts.

8.3.3. Tool Support

In this subsection, we introduce the developed data explorer that can be used to manually label data. Additionally, it is shown how we implemented ML services that can train machine learning models based on this data.

The main goal of the data explorer is to ease the initial visual exploration of ingested and integrated data. It can be used by citizen technologists to browse through historic data in order to create different data views with distinct visualizations. In addition, it is possible to label the data within the data explorer as a preparation for the learning tasks. The added labels can be used for both classification types (sliding window classification and session window classification). Figure 8.7 shows a screenshot of the data explorer containing the data of the CNC machine.

On the top right, the time window can be configured. In the figure only one visualization is shown, but it is possible to create several in one view. Labels can be structured via label categories. Each category is customizable and can contain multiple labels. To label time-series data the line chart visualization is used. On the left, labels that can be used to classify the wear of the milling head are depicted. The labeling is performed in a drag-and-drop-like interface by selecting the time period the label should be assigned to. In addition, shortcuts to switch between labels and to step through the event stream were added to speed up the labeling process. Once the labeling is complete, the model can be trained with the training service, which is explained next.

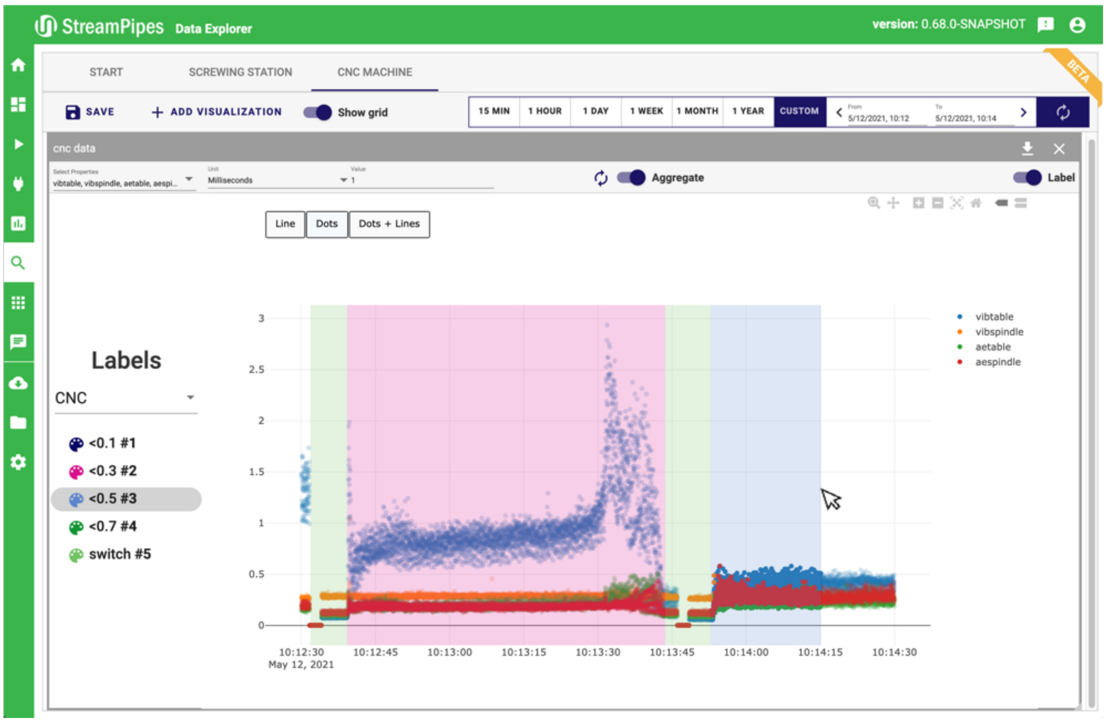


Figure 8.7.: Data explorer to visually analyze data and manually label event streams

Figure 8.8 shows the different components involved in the training phase and how these interact with each other. A user (citizen data scientist) interacts over a Graphical User Interface (GUI) with the *user interaction* service, that is mainly responsible for coordination. First, all event streams are fetched from the data storage to provide a suggestion to users of the different data sets. Once the training task is configurated, the *training service* is initialized. In a first step, the training service downloads the data from the data storage and executes the training loop. We provided an implementation for all the individual building blocks of the conceptual framework introduced in Figure 8.4. For the *parameter search*, an approach based on genetic algorithms was used. It is optimized to find good results at an early stage, compared to random search or Reinforcement Learning (RL) [Real et al. 2019]. This algorithm has three different configuration parameters. First, the amount of learning cycles C. Second, the amount of configurations that are kept in the population P. And third, the parameter S, representing the sample that is drawn from the population based on the highest fitness. Members of this sample are used for the mutation. Additionally to the features that affect all events in the window, we added bucket features for the *feature extractor*. This means the window is further divided into buckets and the features are calculated for each bucket. The amount of buckets is also learned as part of the parameter search. H2O AutoML was used for the *AutoML Framework* because it provides a good integration in Python and Java. All of these algorithms are provided as software components with a standardized interface implemented in python, that allows their replacement with different implementations. During the training, status

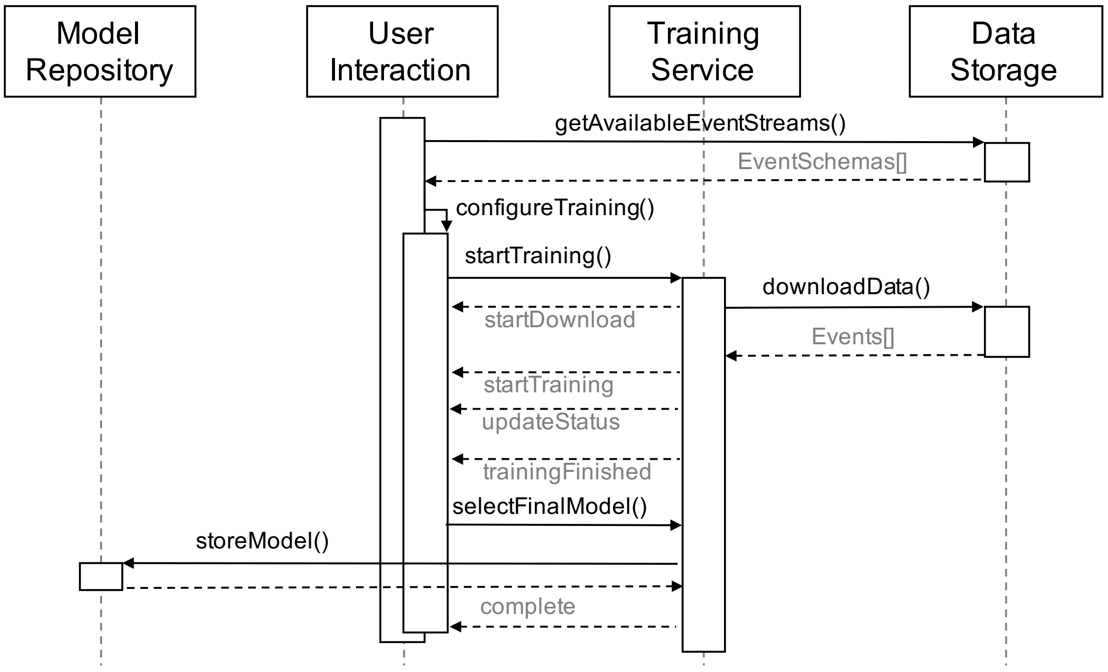
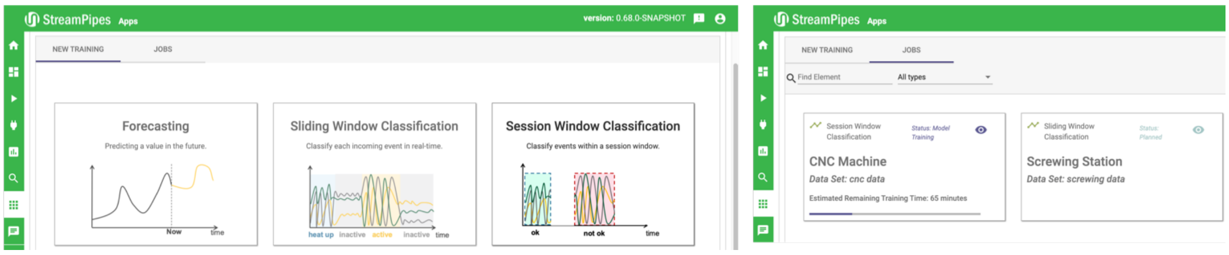


Figure 8.8.: Sequence diagram to train a new ML model

updates are sent to the *user interaction* service to notify the user about the progress and potential problems. Once the training is complete, the results can be reviewed by the citizen data scientist and the best model (e.g. ranked by accuracy) can be selected. This model is then stored in the *model repository* from where it later can be applied on event streams. Storing the model completes the training step. Several trainings can take place simultaneously, if enough computational resources are available. Otherwise, the training job will be stored in a queue and started automatically once sufficient compute resources are available. The *user interaction* service schedules these tasks.


StreamPipes Apps
version: 0.68.0-SNAPSHOT
NEW TRAINING
JOBS
Forecasting
Predicting a value in the future.
Now
time
Sliding Window Classification
Classify each incoming event in real-time.
heat up
inactive
active
inactive
time
Session Window Classification
Classify events within a session window.
ok
not ok
time
StreamPipes Apps
NEW TRAINING
JOBS
Find Element
All types

Session Window Classification
Status: Model Training
CNC Machine
Data Set: cnc data
Estimated Remaining Training Time: 65 minutes

Sliding Window Classification
Status: Planned
Screwing Station
Data Set: screwing data

Figure 8.9.: Machine learning service. On the left new trainings can be defined. On the right the currently running training jobs can be monitored

Figure 8.9 shows the GUI for the ML service. On the left is the selection menu, to select the MLPA type. Once one is selected, the guided process as explained in Section 8.2.2 is started. After initialization, a user can monitor the training progress, which can be seen on the right side of the figure. Here the expected remaining time it takes to train the model is displayed. It is also possible to stop a training process early and to monitor the progress and the quality of the results in a dashboard.

In this section, we introduced the developed tools to support citizen data scientists in training and deploying ML models. The next section, presents a full example that shows how the concepts of this chapter can be used in our motivating scenario.

8.4. Running Example

The running examples in the previous chapters showed, how sensor data of machines from the motivating example in Section 3.2 was ingested and integrated. In this section, it is shown how the overall goal depicted in the motivating scenario, to train a ML model that monitors the milling head wear, can be realized.

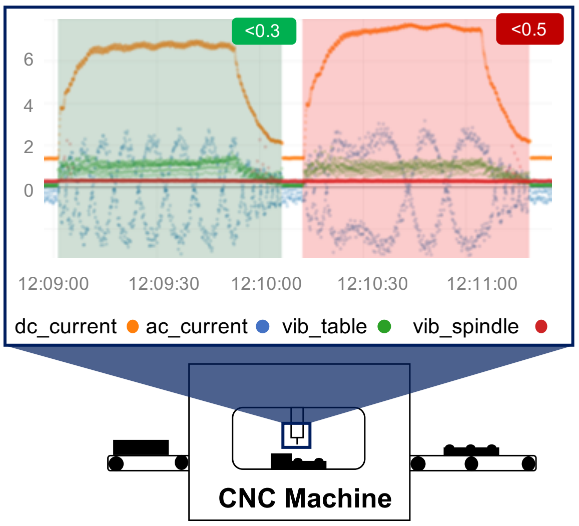
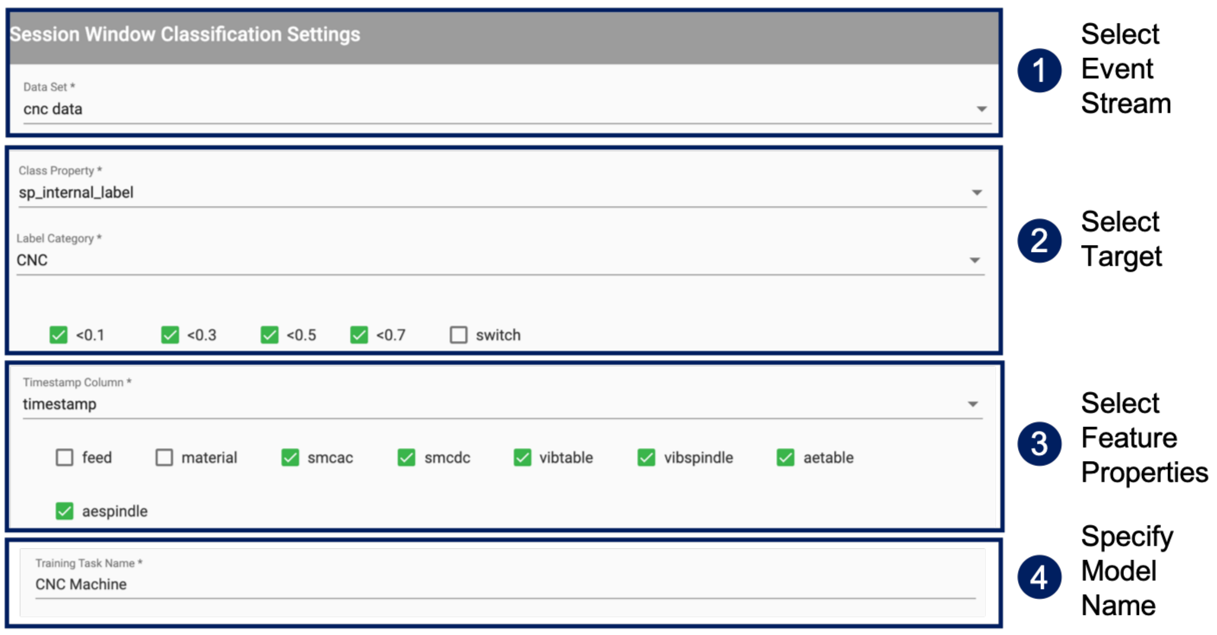


Figure 8.10.: Data from a CNC machine. On the left the milling head is ok, on the right it is worn

Figure 8.10 shows the stored event stream for the CNC machine. To get a better understanding, the citizen data scientist usually inspects the data first. For this purpose, the developed data explorer, enabling a quick exploration of the stored data, can be used. As soon as a basic understanding of the data is achieved, labels can be provided. This can be done in multiple ways. The data can either be manually labeled via a drag-and-drop-like interface in a data explorer, or other Human Machine Interfaces (HMIs) can be used to provide labels with little manual effort. Figure 8.10 depicts two runs of a milling process. The data was already labeled with labels representing the state of the milling head. A label '<0.3' indicates that the wear of the tool is less than 0.3 mm compared to a new tool and '<0.5' less than 0.5 mm, respectively. Once the labelling is completed, the model training can be configured in a step by step guided process, which is depicted in Figure 8.11.


Session Window Classification Settings

Data Set *
cnc data

Class Property *
sp_internal_label

Label Category *
CNC

☑ <0.1 ☑ <0.3 ☑ <0.5 ☑ <0.7 ☐ switch

Timestamp Column *
timestamp

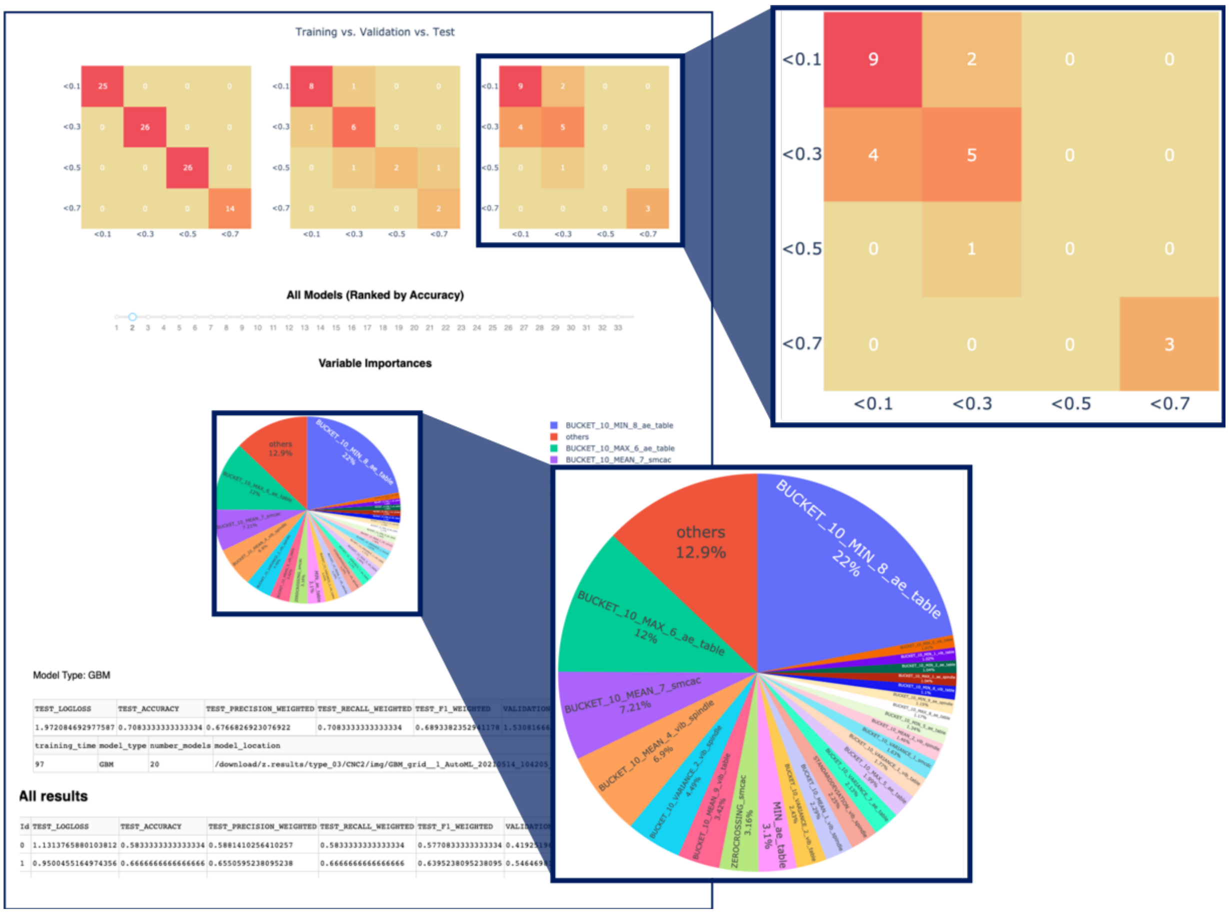
☐ feed ☐ material ☑ smcac ☑ smcdc ☑ vibtable ☑ vibspindle ☑ aetable ☑ aespindle

Training Task Name *
CNC Machine

Figure 8.11.: Configuration dialog to specify the machine learning task

In our example (Figure 8.11), a session window classification configuration is used. The configuration is similar for the other classification types, but can differ for individual steps. First, data of the event stream that is stored in the time-series database must be selected ❶. Then, the target property representing the class value must be specified ❷, as well as the labels that are relevant for the training. This is done using the "label category" drop down menu. If a label is not important for the use case, it can be ignored. Afterwards, the timestamp column is chosen and all relevant properties that should be considered during the parameter search are selected ❸. The selection of those helps to reduce the search space and increases the chance of finding a good model for the problem. In the last step ❹, a name for the training process is set and the training can be executed. This will trigger the training service that downloads the data and starts the training. The progress of the training can be monitored in the UI.

Once the training is completed, the results of the best models can be inspected as depicted in Figure 8.12. The best-rated models, as well as all supporting information and learned parameters, are presented. A more experienced user can have a detailed look at the different metrics. Moreover, we provide two simple visualizations for less experienced users, to support them selecting their favorite model. On the top, confusion matrices for the training, test, and validation sets are shown. By comparing the values from the training (train, validation set) with the values of the test set it can be checked that the model does not overfit, meaning that the results for the training and validation set are


Training vs. Validation vs. Test
<0.1
<0.3
<0.5
<0.7
All Models (Ranked by Accuracy)
Variable Importances
BUCKET_10_MIN_8_ae_table
others
BUCKET_10_MAX_6_ae_table
BUCKET_10_MEAN_7_smcac
others 12.9%
BUCKET_10_MIN_8_ae_table 22%
BUCKET_10_MAX_6_ae_table 12%
BUCKET_10_MEAN_7_smcac 7.21%
BUCKET_10_MEAN_4_vib_spindle 6.9%
ZEROCROSSING_smcac 3.16%
MIN_ae_table 3.1%

Model Type: GBM

TEST_LOGLOSS	TEST_ACCURACY	TEST_PRECISION_WEIGHTED	TEST_RECALL_WEIGHTED	TEST_F1_WEIGHTED	VALIDATION
1.972084692977587	0.7083333333333334	0.6766826923076922	0.7083333333333334	0.689338235294117 8	1.53081666

training_time	model_type	number_models	model_location
97	GBM	20	/download/z.results/type_03/CNC2/img/GBM_grid__1_AutoML_20210514_104205

All results

Id	TEST_LOGLOSS	TEST_ACCURACY	TEST_PRECISION_WEIGHTED	TEST_RECALL_WEIGHTED	TEST_F1_WEIGHTED	VALIDATION
0	1.1313765880103812	0.5833333333333334	0.5881410256410257	0.5833333333333334	0.5770833333333334	0.4192519
1	0.9500455164974356	0.6666666666666666	0.6550595238095238	0.6666666666666666	0.6395238095238095	0.546469

Figure 8.12.: Dashboard to visually inspect the best models from the training process

much better than the results of the test set. Also sometimes it might be relevant to have a high accuracy for a specific class instead of an overall accuracy. Additionally, to the the training results it is shown how big the influence of the individual features is in calculating the results. This is an additional indicator for a citizen data scientist when selecting the best model. Once the model was selected, it is available in the pipeline editor to be applied on a live event stream.

The implementation of the concepts introduced in this chapter are integrated as an extension into StreamPipes. Figure 8.13 shows the StreamPipes pipeline editor with a pipeline using a MLPA. It has an event stream as a data source, followed by a session window classification using the previously trained model and a dashboard as a sink to visualize the results. Different algorithms can be used for the change detection within the MLPA, for example cumulative sum (CUSUM) [Page 1954]. The predictions of the MLPA can be integrated in the dashboard shown in the running example of Chapter 6. This can be placed next to the machine so the operator gets immediate feedback whether a milling head must be replaced or not.

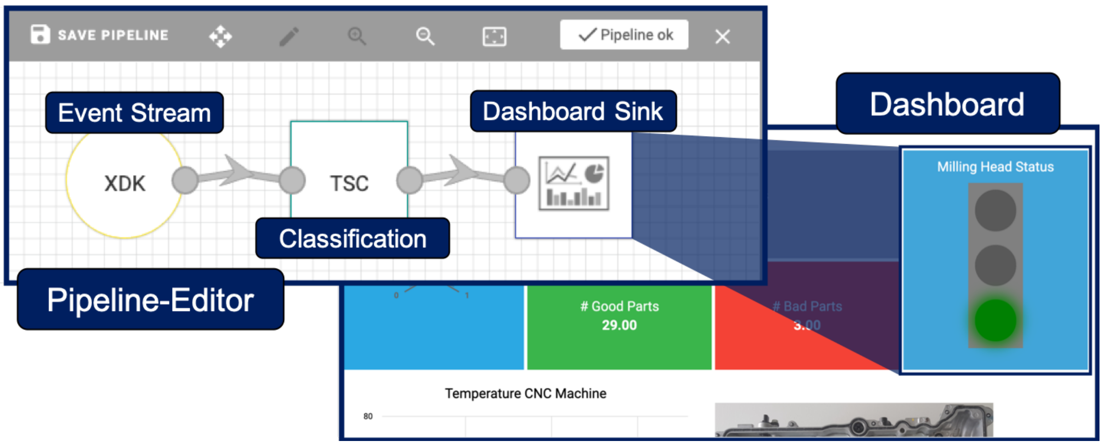


Figure 8.13.: Deployed model in a pipeline and the results are presented in a live dashboard

In this section, the whole analytics process was shown. In the first step, it is explained how training data can be explored, which is followed by the configuration of a supervised ML task. This is used to automatically train a model. Since the training is completely automated, the resulting models should be visually analyzed for consistency by a citizen data scientist. Once this is done, the model is deployed and applied on an event stream produced by the CNC machine. The results are presented in a dashboard for visualization of the output.

8.5. Summary

In this chapter it is shown, how citizen data scientists are enabled to train supervised ML models. Three different MLPA types were introduced that can be used to define standardized training tasks for different applications. Based on this training definition, a framework for an AutoML training service was presented that can train ML models directly on stored industrial event streams without any additional preprocessing. First, Section 8.1, presented a high-level walkthrough introducing the main concepts of this chapter. Then, in Section 8.2 three different types of ML problems on event streams were introduced. Further, it is shown how a ML task can be defined based on the three types and some initially provided labeled data. In Section 8.3, the ML service performing the training, and the automated deployment of the trained models on event streams are shown. In the last section of this chapter, a running example based on the motivating scenario was presented. It shows the training of a model to monitor the quality of the milling head and its deployment with the developed concepts. Additionally, the simplicity of the workflow for a citizen data scientist to train and deploy a model is shown.

Part III.

Finale

9 Evaluation

In this chapter, the evaluation of this work is presented by assessing the concepts introduced in Chapter 6 (Ingest), Chapter 7 (Integrate), and Chapter 8 (Analyze). Section 9.1 presents the evaluation strategy that was realized in this thesis. Three main aspects were evaluated. First, the fulfillment of the requirements was evaluated (Section 9.2). Second, user studies were conducted in Section 9.3, to verify if the approaches are applicable for less technical users (citizen technologist). Third, performance tests were executed (Section 9.4), to evaluate if the developed prototypes are capable of processing data efficiently.

9.1. Evaluation Strategy

In this work, the research methodology of design science was applied [Hevner et al. 2004]. Therefore, the introduced methods and approaches were implemented in software artifacts that were used for the evaluation. The intention of this section is to assess, if the introduced methods, models, and tools fulfill the elicited requirements, can be applied by citizen technologists, and are performant when running in edge computing environments.

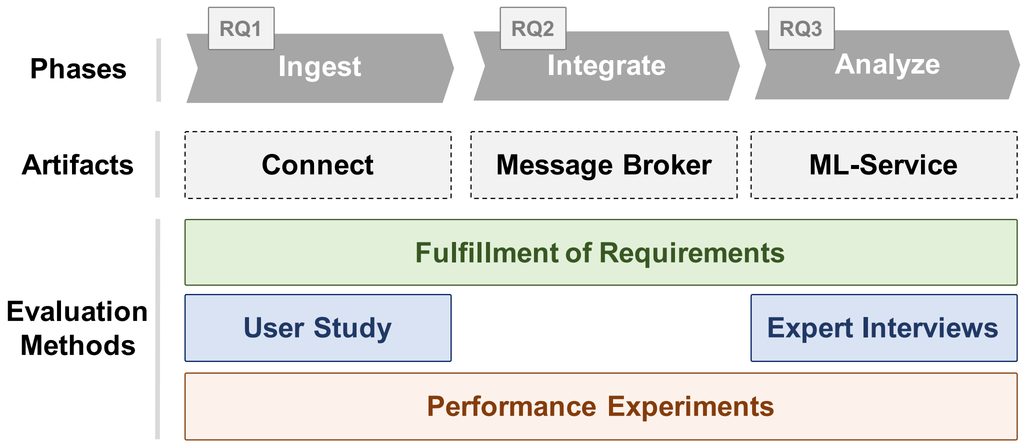


Figure 9.1.: Overview of evaluation strategy

Figure 9.1 gives an overview of the overall evaluation strategy that was chosen in this work. Depicted on the top are the three phases and associated research questions that were answered in the main chapters. In each chapter, one or more artifacts were provided that realize the conceptual contributions and were used as a basis for the evaluation. The prototypes for those software artifacts were integrated into StreamPipes to offer a holistic solution that can be used by citizen technologists. For the ingest phase, the *Connect* framework can be used by citizen developers to ingest data from industrial machines. In the integration phase, a *Message Broker* was developed, realizing the presented event stream reduction techniques. For the *ML Service*, the user interaction layer and the background services were implemented, that include the workflow of configuring a Machine Learning (ML) task and automatically train and deploy models. The *fulfillment of the requirements* is checked for all the phases. In the following, we present two different user studies that were performed to evaluate if the approach is applicable for citizen technologists. The first targeted citizen developers in the ingest phase and the second focused on expert interviews of citizen data scientists for the analyze phase. In the end, several performance experiments show how efficient the data is processed by the prototypes. In addition, several datasets from the Industrial Internet of Things (IIoT) domain were used to show that the presented approaches are generally applicable.

Additionally to the conducted evaluations, we have taken the opportunity to engage with the open source community to obtain feedback from both academia and industry. The developed prototypes were deployed in various industrial settings, to assess if the requirements in real-world scenarios are met. The extensive tool support is provided as open source and is already being used by various external users. We have also endeavored to ensure that our approach is extensible. This can be seen for example in [Jacoby et al. 2021], where the authors introduce an integration of the Asset Administration Shell (AAS) into StreamPipes. It is described, how our adapter model was used to enable none-expert users to connect AAS-based digital twins. They leverage the flexibility and reusability of our model to provide a specific adapter designed for this task. They also make use of our introduced transformation rules to preprocess data on ingestion (e.g. unit transformations).

9.2. Fulfillment of Requirements

In this section, the fulfillment of the requirements described in Section 3.4 is discussed. We focused on the research questions that are affected by these requirements. Figure 9.2 presents an overview of the requirements, as well as the research questions. In the following, it is shown how the developed concepts of this work address the requirements.

We begin with the two ***overall*** requirements tackling the main research question:

(R1) Accessibility for Citizen Technologists

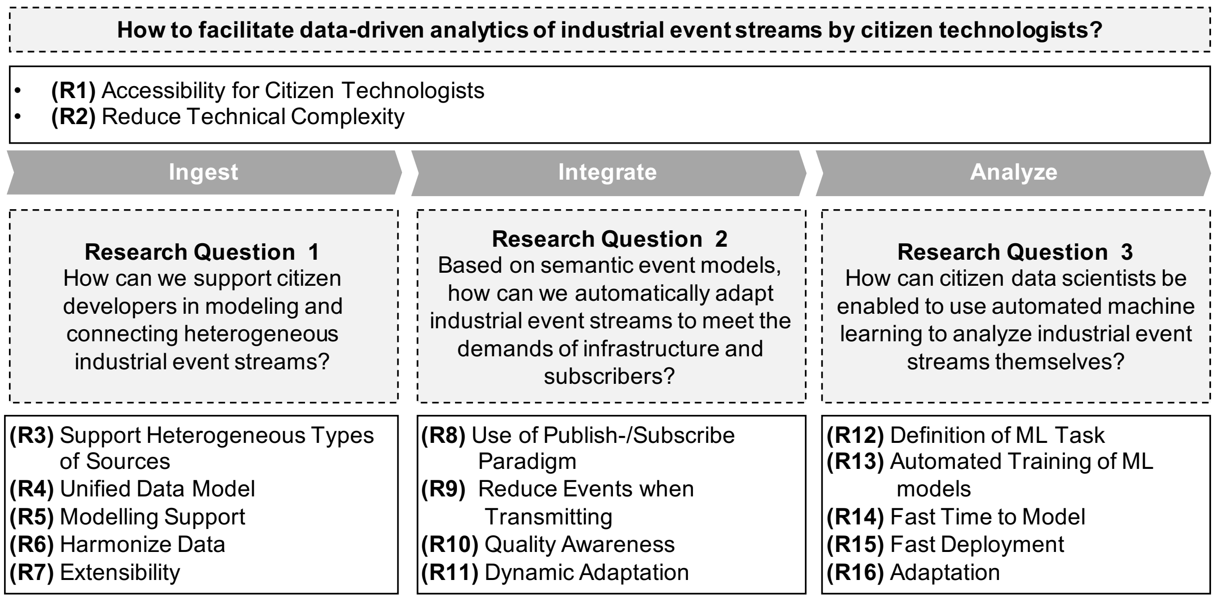

How to facilitate data-driven analytics of industrial event streams by citizen technologists?
(R1) Accessibility for Citizen Technologists
(R2) Reduce Technical Complexity
Ingest
Integrate
Analyze
Research Question 1
How can we support citizen developers in modeling and connecting heterogeneous industrial event streams?
Research Question 2
Based on semantic event models, how can we automatically adapt industrial event streams to meet the demands of infrastructure and subscribers?
Research Question 3
How can citizen data scientists be enabled to use automated machine learning to analyze industrial event streams themselves?
(R3) Support Heterogeneous Types of Sources
(R4) Unified Data Model
(R5) Modelling Support
(R6) Harmonize Data
(R7) Extensibility
(R8) Use of Publish-/Subscribe Paradigm
(R9) Reduce Events when Transmitting
(R10) Quality Awareness
(R11) Dynamic Adaptation
(R12) Definition of ML Task
(R13) Automated Training of ML models
(R14) Fast Time to Model
(R15) Fast Deployment
(R16) Adaptation

Figure 9.2.: Research questions & requirements

In Chapter 3 (Motivation) it was shown that technical specialists are usually required to analyze industrial data. In the three main parts, Chapter 6 (Ingest), Chapter 7 (Integrate), and Chapter 8 (Analyze) we introduced concepts, methods, and models to automate tasks that previously required technical specialists. With these automations, citizen technologists are enabled to analyze the data themselves. The focus lies on the support of the user as shown in Section 6.4 and Section 8.2.2. In the implemented prototypes, this user support was integrated in a way that citizen technologists could interact with the system over a Graphical User Interface (GUI). This approach ensures that no code has to be written and no model must be explicitly created. These prototypes were subsequently evaluated in user studies. These were already integrated in Apache StreamPipes. The concepts are used by both, developers extending the functionalities and citizen technologists using the software.

(R2) Reduce Technical Complexity

In Section 3.3 (Problem Statement) and Chapter 4 (Related Work), we showed problems of state-of-the-art solutions. Further, it was explained why the technical complexity must be reduced if targeting citizen technologist to perform industrial event stream analytics. Therefore, Chapter 6 (Ingest) focused on citizen developers and Chapter 8 (Analyze) on citizen data scientists. In the user studies (will be presented in Section 9.3), it is shown that users can quickly apply the developed approaches. Additionally, the user studies show that even less technical users were able to analyze industrial event streams.

After discussing the requirements tackling the overall approach, we will now discuss the requirements for the three individual research questions. We start with requirements focusing on ***model & ingestion***:

(R3) Support Heterogeneous Types of Sources, (R4) Unified Data Model

To support heterogeneous types of sources, a semantic adapter model was introduced in Section 6.3. This model distinguishes between generic and specific adapters to cover a wide range of possible industrial data sources. The model was used as the basis for the implementation of the StreamPipes Connect framework. It contains more than 30 different protocols and specific adapters. This number keeps growing, because we provided a Software Development Kit (SDK) that can be used to integrate new protocols and adapters. Data from the various adapters is harmonized at runtime on a message broker in a uniform form using the presented event model. This allows downstream algorithms to process data of multiple data sources.

(R5) Modelling Support

Citizen developers are supported during the modeling process as presented in Section 6.4.3. This was evaluated in the user study and participants were able to connect new data sources in a couple of minutes. Further, it was stated that even users without programming knowledge were able to use the introduced concepts to connect data sources.

(R6) Harmonize Data

To be able to harmonize data directly on ingestion, a model for transformation rules was introduced in Section 6.3.2. These rules are inferred based on the users interactions with the system, so there is no need to explicitly create the model. Based on these rules, functions are invoked directly within the adapters on an edge node during runtime. This ensures that only harmonized data is ingested for further processing. The impact of these preprocessing rules on the throughput was evaluated within the performance experiments and showed that it is sufficient for industrial event stream analytics.

(R7) Extensibility

The model was designed in a way that it is extensible. For example new protocols, formats, specific adapters, or transformation rules can be added. Using the provided SDK users can introduce new models and integrate new implementations into the Apache StreamPipes project. Several new extensions were already developed by the community. Further, it is possible to store and share adapter descriptions (e.g. for a special kind of sensor) to facilitate reuse.

In the following, we focus on research question two and the requirements tackling the ***integration***.

(R8) Use of Publish-/Subscribe Paradigm

In Section 7.3, a wrapper around topic-based message brokers was introduced. This uses the functionalities of the underlying message brokers and extends them with the possibility to dynamically adapt the event streams. Apache Kafka[1] was used as a basis for the developed prototype applied in the evaluation. The wrapper contains a client that can be used to publish events (publisher), as well as a subscription-transformer preparing the events for the individual subscribers that receives them and reconstructs the resulting virtual events.

(R9) Reduce Events when Transmitting

To reduce events when transmitting, several reduction strategies were introduced in Section 7.2. Both, the event frequency and the size of individual events were reduced. The developed message broker applies those strategies and only transmits partial events that are reconstructed by the subscribers. The evaluation shows that with this approach it is possible to reduce the amount of transmitted data, while only slightly increasing the memory and CPU usage (Section 9.4.2).

(R10) Quality Awareness, (R11) Dynamic Adaptation

To meet requirement R10, subscribers have the possibility to define the quality requirements for receiving event streams. The logic of the message broker ensures that those quality requirements are met. Since the conditions in publish-/subscribe systems can dynamically change (e.g. new subscriber registers with different quality requirements), the broker is able to dynamically adapt to those changes. The schema registry keeps an overview of the whole system and is capable to change individual publisher or subscription transformers, as shown in Section 7.3.

In the last part of this section, we look at ***analytics-related requirements*** focusing on research question three. Citizen data scientists are the targeted role in this research question and the goal is to enable them to train supervised ML models:

(R12) Definition of ML Task

We introduced three types of Machine Learning Pipeline Agents (MLPAs) that use ML models to analyze event streams. The participants of the expert case study stated that all the supported types make sense to them. To configure these ML tasks, a workflow is presented in Section 8.2.2. This workflow can be applied by users with little data science knowledge, which was shown in the user study, where even users with little experience in applying ML stated they would be able to use the system.

(R13) Automated Training of ML Models

The training service introduced in Section 8.3 is capable of training a ML model directly on (labeled) data. Data of industrial event streams that was previously ingested and integrated, can directly be used as the foundation for the training task. No further preprocessing is required, only the training task must be specified and the training itself is then completely automated.

[1]https://kafka.apache.org/ (accessed on 04/10/2021)

(R14) Fast Time to Model

Since the training and parameter selection is automated, only a few configurations must be provided. The training itself might take a while, however, the manual effort to define the training and analyze the results only takes minimum time effort. This means a citizen data scientist must only inspect the results once the training is completed. It was also shown in the user study, that participants were able to train two different models in a couple of minutes, even when using the system for the first time.

(R15) Fast Deployment

Once the models are trained, they are available to deploy them directly within a streaming environment. For the prototype, an integration with Apache StreamPipes was implemented, where users can use the models in the pipeline editor. This was also part of the user evaluation and participants used the previously trained models on new data. The participants stated that they like the usage of ML models in such pipelines, because it allows them to have new analytics possibilities on their data.

(R16) Adaptation

To detect changes in the event stream, a change detector is integrated into the MLPA as described in Section 8.3.2. A citizen data scientist is notified as soon as a change is detected. With this knowledge, it is possible to decide whether the model needs to be updated or not. Further, it might be necessary to prepare some new training data by the citizen data scientist before a new model can be trained automatically.

9.2.1. Discussion

This work facilitates industrial event stream analytics for citizen technologists. Therefore, different requirements for each research question were elicited and their fulfillment is presented in this section. First, the overall requirements were discussed. Then, the requirements for each research question were presented. The main contributions for the three parts are illustrated in the three main chapters of this work building upon each other.

9.3. User Study

We performed two different user studies to evaluate whether the developed concepts can be used by citizen technologists. The first study targeted less technical users who had to ingest data from different data sources. In the second study, we evaluated the ML service to see if the developed methods enable citizen data scientist to analyze event streams. In both cases, the participants got a brief introduction, then they had to use the software prototype to perform multiple tasks. In the end they were asked several questions about the usage of the system.

9.3.1. Connect

In this user study, we evaluate if citizen developers with little technical and programming knowledge are enabled to connect new data sources. 19 students from the fields of industrial engineering and management as well as computer science were recruited randomly from a voluntary student pool of the Karlsruhe Institute of Technology (KIT) using hroot [Bock et al. 2014]. First, the design of the study is introduced, then the results are presented and at the end the results are discussed.

Design

For the user study the the Karlsruhe Decision & Design Lab (KD^{2}Lab)[2] at the KIT was used. The overall goal for the participants was to create a live air quality index, similar to the one presented in ['Air Quality in Europe' 2017], based on data from two different data sources. Since no technical background knowledge or previous experience with sensor data was required, we started with a 10 minute introduction explaining the approach and the domain. Also, the data sources were introduced as well as the types of data that had to be connected (e.g. particulate matter PM2.5 /PM10, nitrogen dioxide NO_2, …). Further, it was explained how an air quality index might be calculated. Once the introduction was finished, the participants were assigned to isolated cabins. Each cabin was equipped with a computer that had access to the software prototype running on a centralized server. The software provided instructions, on how to use it and how to perform the tasks. Instructors were still on site but did not provide further assistance. The students had to solve the task on their own.

The first task was to configure a specific adapter to connect data from the openSenseMap API [Pfeil et al. 2015]. This is an online service for environmental data based on sensors distributed over the whole world. In the second task, a generic adapter had to be created to connect environmental data from official institutions. For this an endpoint provided by the 'Baden-Wuerttemberg State Institute for the Environment, Survey and Nature Conservation' was used. Once both tasks were completed, an online questionnaire, targeting the user experience, had to be filled out. Three standardized questionnaires were used to access the usability of the system. To get specific feedback to the functionalities of the adapter concept and how they are created, we added further questions related to the approach. In addition, control questions were added to ensure that the participants answered the questions carefully. Three of the participants answered those control questions wrong, resulting in a total of 16 valid answers.

[2]http://www.kd2lab.kit.edu/(accessed on 04/10/2021)

Results

In this section, we show the results from the standardized questionnaires as well as the supplementary questions. First, the results of the System Usability Scale (SUS) [Brooke et al. 1996] are shown. The SUS measures the usability of a software system by comparing results to the average scores of 500 websites[3]. This was used because the user group is familiar with interacting of websites or web applications. Therefore, we expect a meaningful result. When the score is above 71.4 it is considered a good result. In this work we also use colors to indicate how well the score is compared to the other systems. When the value is above 71.4, it is marked in green.

Overall	
Correct control question	72,2

Connect Sensor with Programming Language			
Yes	67.5	No	75.0

Technical affiliation	
Quantil > 0.75 (high)	63.1
Quantil > 0.25 < 0,75	76.9
Quantil < 0.25 (low)	71.9

Figure 9.3.: Results of SUS

Figure 9.3 shows the results of the SUS. On the left, is the overall result of 72.2, considering all participants that answered the control questions correctly. Our aim was to target less technical users with the introduced concepts and techniques. However, there was a high variance in the technical expertise of the participants. So we grouped the results according to the technical understanding of the participants. In the middle of Figure 9.3, the results are grouped according to whether participants are able to connect sensors with a programming language of their choice. Our goal of being useful for less technical users was confirmed here, as participants without programming skills find the system more useful (good system, mean: 75.0) than participants who could also use a programming language for the connection (acceptable system, mean: 67.5). On the right, we grouped the participants according to their technological affinity. To assess this, we used the items of the Technology Readiness Index (TRI) [Parasuraman 2000] and adopted them in order to formulate questions about expertise in using programming IDEs and data tools. On the top, are the results of participants with a high technology affinity (quantile > 0.75). They find the system not as useful as less technology-affine participants, but still acceptable (mean: 63.1). In the middle are participants with an average technology affinity. The results show that they 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 than the participants of the middle group (mean: 71,9). This also confirms our assumption, that such a tool is especially useful for non-technical users. The SUS gives the tool a rating of a good system. It must also be noted that participants used the system for the first time for only 15 to 20 minutes. Taking this into

[3]https://www.trymyui.com/sus-system-usability-scale (accessed on 04/10/2021)

account, it is already a very good score and it is likely that more experienced users would rate the system better.

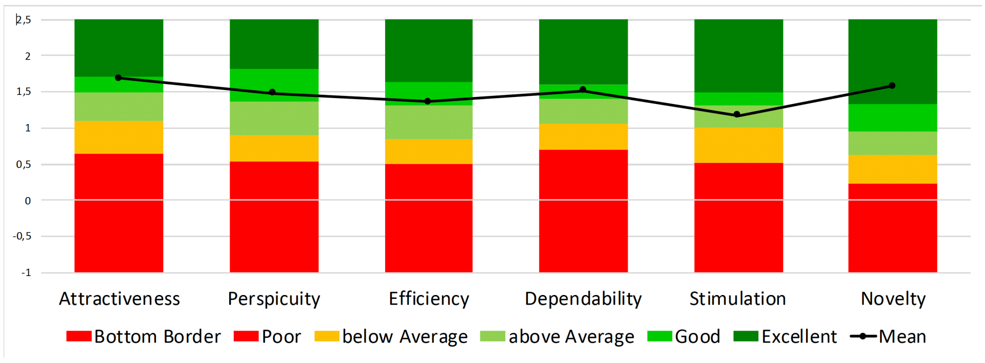


Figure 9.4.: Results of User Experience Questionnaire (UEQ)

The UEQ [Schrepp et al. 2014] was used for the second questionnaire and the results can be seen in Figure 9.4. The figure shows all the six categories: Attractiveness, Perspicuity, Efficiency, Dependability, Stimulation, and Novelty. A Likert scale is provided for each category, indicating how well the system compares to others rated with the UEQ. The colors indicate how well the system is rated and the black points represent the score for each category. As can be seen in the figure, the results of all categories are above average. The system is rated as good in the categories Attractiveness, Perspicuity, Efficiency, and Dependability. The Novelty of the system is even rated as excellent. Additionally, it can be seen that the system is rated equally good over all categories, meaning there is not an individual category which needs particularly improvement. However, it suggests that there is still room for improvement, but for a first user study based on a prototype the results of the UEQ are already very promising.

Finally, Table 9.1 presents the results of some supplementary questions that provide additional information about the usage of the system.

Table 9.1.: Overview of implemented adapters

Question	Yes	No
I have the confidence to connect real-time sensor data with a programming language of my choice?	5	11
I trust myself to connect real-time sensor data with the system?	16	0
Do you think that if you are trained to use the system, you could connect new data sources in under a minute?	14	2

To evaluate how technical the participants were, we added the question, that asks if they could use a programming language of their choice to connect sensor data. Only 5 of the participants answered with yes and 11 stated that they are not able to connect sensor data with a programming language. This shows that the participants had a good mix of technical experience. The system focuses on less technical users, however it was also interesting to get feedback from more technical-oriented users. The second question shows that all participants think that they are able to connect real-time sensor data with the system. Especially, the ones that answered the previous question with no, also stated that the system enables them to connect sensor data. This shows that the software achieves the designed goal. Additionally, we asked if they thought they would be able to connect new sources in under a minute, once they were trained on the system. 14 stated that they would be able to do that and only 2 stated that it would take them longer. This also shows that the approach is efficient. Without the provided concept the task is quite technical and time-consuming. The evaluation shows that, even less technical users are able to quickly connect new data sources in a short period of time with the provided approach. Also, we monitored the time of the interaction of the users with the system to see how long they needed to complete the individual tasks. For each task, it took them about 3 to 5 minutes. As they saw the system for the first time and also had to understand the instructions, it is likely to assume that their estimate of being able to connect the data quickly once they are familiar with the system is realistic.

Discussion

With this performed user study the goal was to evaluate how easy it is for citizen developers to use the methods and models to ingest sensor data. Therefore, a prototypical implementation of the concept was provided to participants who used the system on their own. Thereafter, they rated the system in a questionnaire. Both the UEQ and SUS indicate that the system is not only usable (i.e. fulfills its purpose) but also provides a good experience when using it (i.e. fun experience). Overall, the results of the user study show that the system for the ingest phase (StreamPipes Connect) is already rated as a good system, which can be used by citizen developers to quickly connect new data sources.

9.3.2. ML Service

In this user study, the ML service introduced in Chapter 8 is evaluated. Participants were not randomly selected as before, but experts from the relevant domains were specifically chosen. In the evaluation, they solved two tasks based on two different use cases in which ML models had to be trained, evaluated, and deployed. The goal was to find out if the developed approach makes sense and is applicable from a domain expert point of view.

Design

For this study six experts from the fields of IIoT and ML from industry and academia were selected. They were chosen because most of them are familiar with stream processing (e.g. Apache StreamPipes) and the basic concepts of ML. This knowledge is needed for citizen data scientists. The prototype was integrated into StreamPipes, so it was necessary for the participants to understand the basic concepts and functionalities. They acted as citizen data scientists and had to define two different ML tasks. Therefore, two examples were chosen based on real-world sensor data.

In the beginning of each interview a few slides with an introduction were presented. First, a brief introduction about IIoT and Apache StreamPipes was given. Then the drawbacks of rule-based analytics were outlined and how ML models could be used instead. Based on the Cross-Industry Standard for Data Mining (CRISP-DM), the long model development cycles for ML models were illustrated to show what must be done to integrate ML into stream processing applications. After that, the approach of this work was outlined by introducing the three MLPA types from Section 8.2 and how such models can be trained on previously collected and labeled data. For the evaluation, the participants had to work on two use cases: First, a state detection model for an acceleration sensor box had to be trained (e.g. whether the sensor is shook or thrown). This could be done using a *sliding window classification*. For the second use case, a *session window classification* was trained to classify the drink produced by a coffee machine (e.g. coffee, espresso, tea, ...) based on acceleration and power sensors. These examples were chosen because they are easy to understand, especially for participants from different domains and the concepts can be adapted to other scenarios in production environments. For both cases some training data was available, as well as an event stream with live data from the sensors to test the resulting models. To reduce the training time during the evaluation, already pretrained models with identical configurations were provided. This means the participants did the whole workflow, but instead of waiting for the model training to be finished (about one hour for each model) they used a previously trained model for the deployment. The system ran on a server and was accessible over the internet. During the evaluation the participants shared their screen and started by browsing through the previously recorded data in the data explorer before they defined the ML tasks. The whole evaluation was guided by an instructor, who could answer questions directly. After the use cases were finished, some questions were asked by the instructor and the participants filled out a questionnaire. Each of the interviews took about one hour and the results of those questions are presented in the next section.

Results

In this section, the results of the user study are presented. First, the results of the questionnaires are introduced then the feedback that was given during the interviews is discussed.

For each question, a Likert scale was used and the answers of the participants for each question are presented. The results are color-coded and each column represents an individual participant. First, Figure 9.5 shows the results of questions regarding the technical background of the users. The goal was to learn about the technical skill level of participants and how much experience they already have in the field of ML and industrial event stream analytics.

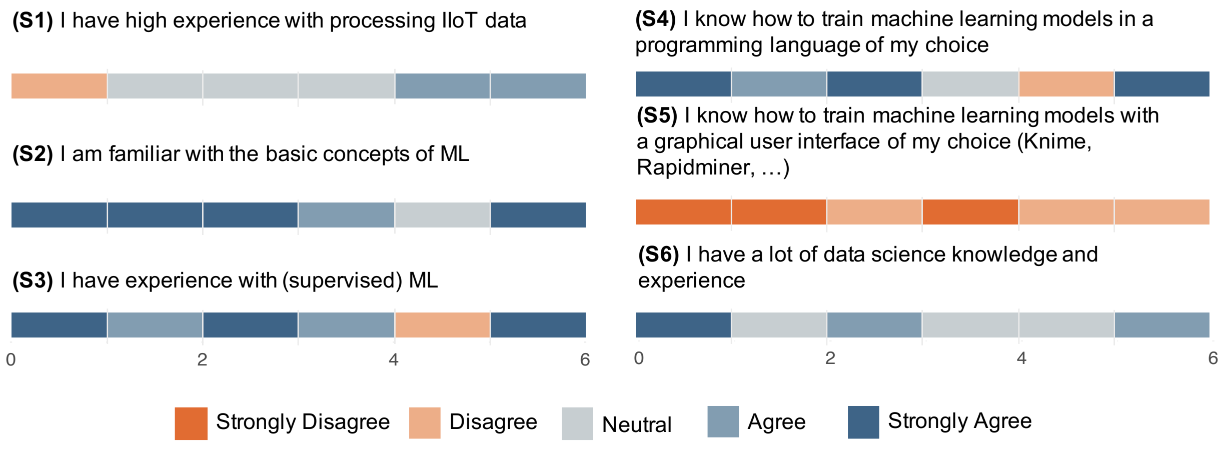


Figure 9.5.: Answers about the technical background knowledge of the individual participants

The first statement **(S1)** was about the experience in processing IIoT data. Here, it can be seen that not all participants work regularly with IIoT data in their job. However as stated before, we ensured that all of them were familiar with the basic concepts of stream processing and StreamPipes. The next statements **(S2)** and **(S3)** are about the experience regarding the concepts of ML. All of them are familiar with the basic concepts and except for one they are also generally familiar with supervised ML. This is important to know, because we assume that users act as citizen data scientists that already have a basic understanding. In **(S4)** and **(S5)**, it can be seen that the participants are more familiar with programming languages than graphical tools to define ML models. The last statement of this category **(S6)**, shows that even if the participants stated that they are familiar with ML, most of them would not consider them as very experienced data scientists. This shows that we selected a representative group of participants that are able to evaluate the approach from a citizen data scientist point of view.

Figure 9.6 depicts the answers regarding the statements about the introduced approach in Chapter 8. Overall, it can be seen that the participants liked the approach and also considered it as useful. In statement **(S7)** and **(S8)** all participants (strongly) agree that they understood the different types of ML problems and that they make sense from their domain knowledge point of view. Most of the participants stated that they think they would be able to generate new insights based on own data **(S9)**. In **(S10)**, all of them confirmed that they understood what they were doing when executing the use cases on the system. In **(S11)**, four of the participants confirmed that they think in their

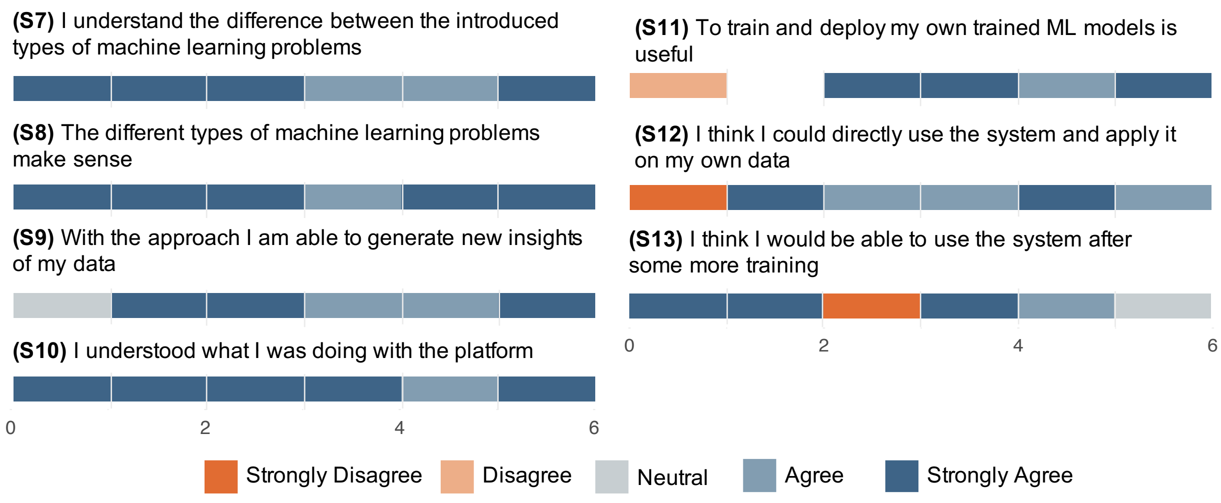


Figure 9.6.: Answers to the statements regarding the approach introduced in this work

particular domain it would be useful to train ML models, one participant did not answer this question and the first one stated that the approach is not applicable in their domain. Statements **(S12)** & **(S13)** had the goal to discover how intuitive the developed solution is from a user point of view. Most of the participants stated that they would directly be able to use the system without any further assistance. Only one participant stated that some more training would be required, but then would be ready to use the system.

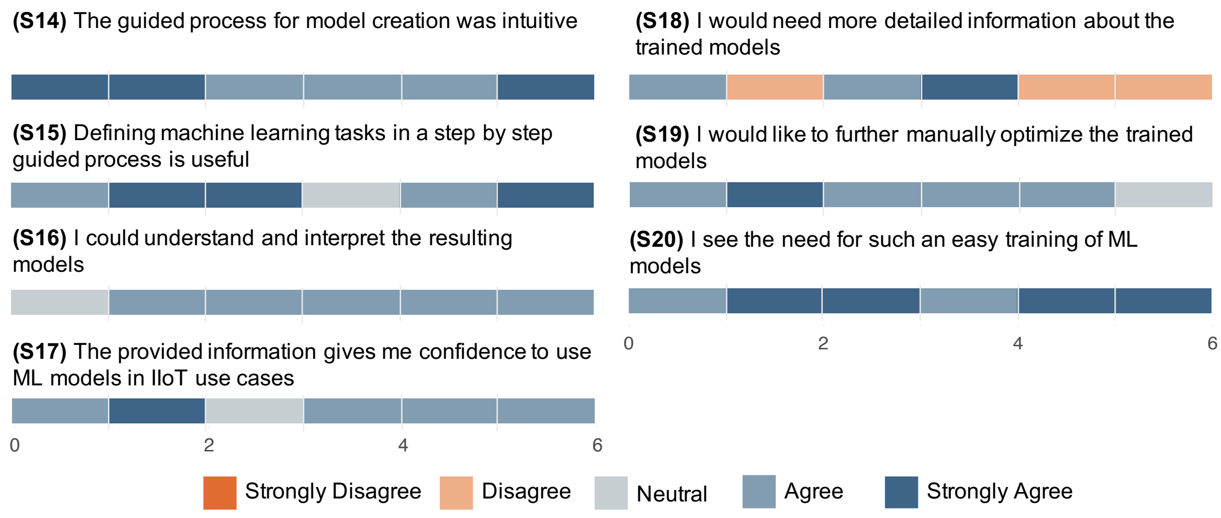


Figure 9.7.: Answers to the training process for the ML model

Figure 9.7 shows the answers about the statements regarding the process of training new models. As before, it can be seen that the participants agreed with most of the statements, only in **(S18)** half of them stated that they would not need more detailed information about trained models. In statement **(S14)**, all mentioned that the guided process to create a new model was intuitive. Additionally, they liked the approach of a step-by-step process to define the training task **(S15)**, except one participant who marked it only as neutral. Most could understand the results of the model **(S16)**. However, compared to the other answers of this section the average agreement is lower. This could also be seen in the statements during the questioning in the interview. Most of the participants would have liked more detailed information about the training parameters. In **(S17)**, the statement was whether the user has confidence in the results of the model and most of the participants agreed with this statement. Lastly, in **(S19)** it can also be seen that more experienced users would like more flexibility in addition to the simple process to train models **(S20)**.

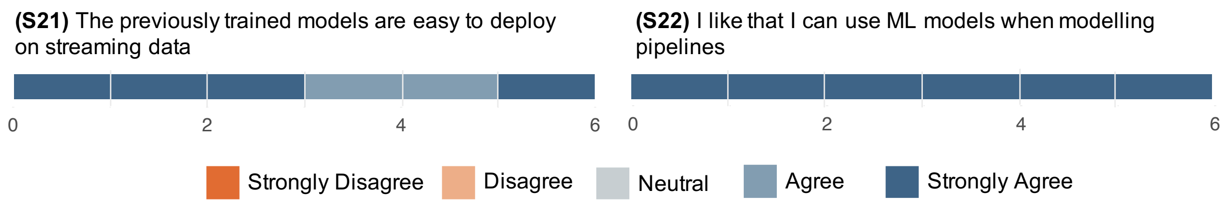


Figure 9.8.: Answers regarding the deployment of the previously trained models

In the last part of the user survey, the participants had to rate statements regarding the deployment of the previously trained models, see Figure 9.8. Four participants could strongly agree with statement **(S21)** that the previously trained models are easy to deploy and the other two agreed with it. In the last statement **(S22)**, all participants liked that they were able to also add ML models into pipelines. This category has the highest overall agreement.

The overall results of the questionnaire is very positive. This could also be confirmed by the feedback that was given during the interviews. In addition, the participants also provided valuable input that we will use for further improvements. They liked the simplicity of the approach. Most feedback was related to advanced features that they thought would be helpful for more experienced citizen data scientists. For example, we asked if it was useful to display relative importance of the individual features. Most of the participants stated that this information would be very helpful. Additionally, they mentioned it would be good to provide further information, especially for new users, like an in-depth explanation how the features are calculated. Also they liked that it is easy to train new models and that the information is clearly presented. To provide further functionalities for more advanced citizen data scientists some suggested to also add a more advanced view that contains all the detailed information of the learning process. This can simply be integrated in the GUI, because all this information is already available in the current implementation. A few participants also wanted to change some of the configurations during the training process. In the prototype this had to be done in configuration files, but based on that feedback we will include this functionality in the GUI as well.

Discussion

In this user study, six participants who acted as citizen technologists were asked to evaluate the implemented prototype. First, a short presentation was provided to introduce the basic concepts and the two use cases. Then, two different models were trained and deployed using the prototype integrated into StreamPipes. In the end, questions from a questionnaire were answered. The results of this study showed, that the participants were able to train and deploy ML models, further the results showed that they liked the approach and that the concept made sense to them. Overall, it can be said that they thought it is useful, in particular it was easy to use. Most of them stated that they could directly use the software without any additional instructions. It also showed that additional advanced information (like information about the ML models and data processing / feature extraction) should be integrated in the future to further improve the system.

9.4. Performance Experiments

In this section, the performance is evaluated based on implemented modules for the individual phases as shown in the overall overview in Figure 5.1. To show that the presented approaches are universal, data from different domains was used in these evaluations.

First, the performance of the connect module is evaluated (Section 9.4.1), then the performance of the developed message broker (Section 9.4.2). In the end of this section the ML Service is evaluated (Section 9.4.3).

9.4.1. Connect

This section focuses on adapter instances in the connect module. To evaluate the performance of those adapters, they were instantiated on two different edge nodes and the throughput was measured to see how many events can be processed by a single adapter. The performance was measured for two different scenarios. First, the extraction of the data from the source and second the preprocessing functions that harmonize the events on ingestion.

Design

The performance tests ran on two different setups. The first was a robot arm from Universal Robots (UR), connected via the ROS protocol. In addition, to see what the maximum throughput of an adapter is, an event simulator was used. In both setups, the

StreamPipes core ran on a central server. For the worker node, two types of commonly used edge nodes were selected. One was a Raspberry Pi 4 and the other one was an Intel NUC shuttle computer. From the robot arm we monitored the events of the joint states with an event frequency of ~500 Hz and with an event size of about 800 bytes. For the simulator, temperature events were randomly created with three different properties: *timestamp*, *temperature* value, and an *identifier*. The events were created as fast as the simulator could generate them. For each edge node we ran the test in 6 different setups, each with a different amount of preprocessing functions to see how they impact the throughput.

Results

First, the performance of the joint state events from the robot arm was measured. Those could be processed without any delays on both edge nodes. Therefore, we used a data simulator to see the maximum performance of an adapter. To avoid individual outliers, each test ran 15 times and the results are plotted in Figure 9.9. The y-axis represents the mean events per second and the x-axis shows the length of the preprocessing pipeline. At *None*, the test was performed with no preprocessing function. Then a *Rename* functions was added, after that a *Move* function was used. Then a property was deleted *(Delete)* and with *Add Timestamp*, the current timestamp was added. The last function was the *Transform Unit* to change the unit of the temperature property. Each function was appended to see how well an adapter performs with more preprocessing steps. To test the performance on the NUC 10.000.000 events were created per test and for the Raspberry Pi 5.000.000 events.

The figure shows that the NUC performs is significantly better than the Raspberry Pi. For many real-world use cases a Pi might still be sufficient, since it processed ~54.000 events per second with no preprocessing function. The results show that, the longer the preprocessing pipeline, the less is the throughput of the adapter, except for the delete function. This seems to improve the throughput because the size of the event is reduced. It can also be seen that different functions have a different impact on the throughput. The functions *Add Timestamp* and *Transform Unit* seem to have the biggest impact. It can also be seen that the functions on both devices have about the same impact on the relative throughput.

Discussion

In this subsection, the performance for adapters was measured. It was shown that the edge node type has a big influence on the performance. However, even edge nodes with limited resources (e.g. Raspberry Pi) might already be sufficient for many use cases. Also, the preprocessing functions have an impact on the performance. However, usually one

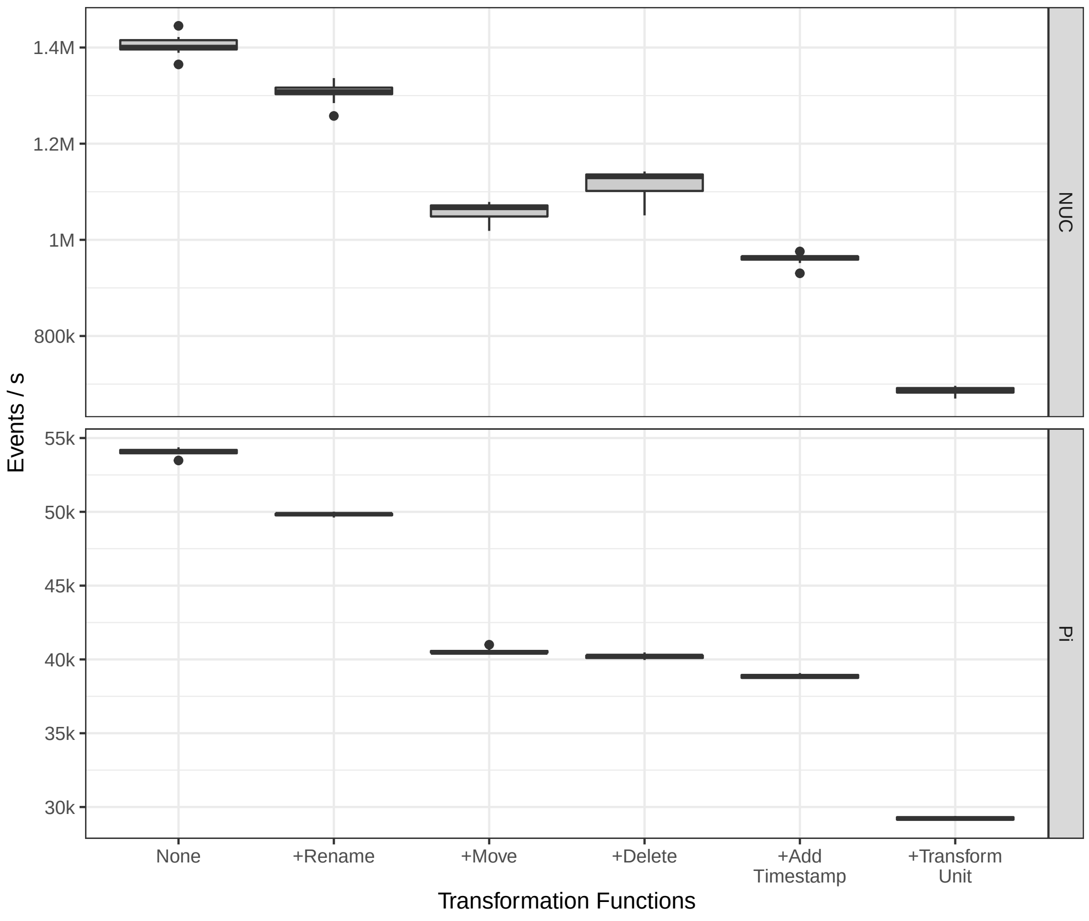


Figure 9.9.: Results of the performance tests. The top chart shows the results for the adapter running on the NUC and on the bottom on a Raspberry Pi

adapter only connects one or a couple of machines, so the reduction in throughput does not significantly impact the deployment in industrial settings.

9.4.2. Broker

In this section, the performance of a prototypical implementation of the message broker is evaluated. Data from the domain of smart buildings was chosen and a real-world data set[4] was used. It contains data of a Solar Thermal Climate System (STCS) that uses solar heat to heat the building in winter and cool it in summer.

[4]This dataset was provided by the Faculty of Management Science and Engineering of the Karlsruhe University of Applied Sciences

Design

The implementation of the broker is realized in Java as a wrapper around Apache Kafka[5]. Kafka was selected for its high scalability and ability to handle large amounts of events. For the producer, a wrapper around the kafka-client was implemented. Before this producer transmits events, the reduction strategies introduced in Section 7.3 are used to create partial events. On the server side, the partial events are prepared for the individual subscribers, potentially further reducing the size. For the subscription, we implemented a subscription interface that receives the partial events and creates virtual events with values for each property. Those virtual events are then forwarded to the application logic using the subscriber. From the clients point of view, the subscription is similar to any other topic-based message broker as the reduction operations are carried out transparently.

The data set used for the evaluation consists of approximately 1,000,000 events and the size serialized in JavaScript Object Notation (JSON) is about 855MB. For the tests one Kafka instance was started additionally to the three developed components, all deployed as Docker containers. As a server, we used a machine with 4 cores and 32 GB of memory.

During a test, the data of the STCS system is read from a file and send to the broker. To test how big the reduction is without loosing any information in the events, the value of ε was set to 0. This means only values that change are transmitted and the reconstructed virtual events are equal to the original events. For the subscriber, the variable ε was set to 1 to see the impact of introducing a small error in the resulting virtual events. For both strategies, the same prediction rule, that uses the previous value to see whether it changed or not, was chosen. We compared different serialization configurations to measure the impact on the size of transmitted data. Therefore, we choose Apache Avro[6], JSON, and JSON events compressed with GZIP. All three types were chosen, because they allow to remove individual properties from the serialized events. With JSON, the resulting events are human readable, if that is not required Avro or JSON compressed with GZIP can be used.

To measure how much data is transmitted, we added four logging stages into the system. Two at the publisher, to get the size of the raw events, as well as the partial events that are transmitted. In addition, a logging stage was added at the broker to monitor the size of the partial events for the subscriber. The last one was added at the subscriber for the resulting virtual events. Since we expected higher demands on the compute resources for the reduction and reconstruction, we also measured the utilization of the CPU and the memory. As a baseline, we first ran the test without reducing events before running it again with active reduction algorithms.

[5]https://kafka.apache.org/ (accessed on 04/10/2021)
[6]https://avro.apache.org (accessed on 04/10/2021)

Results

In this section, the results of the performed evaluation experiments are discussed. The first test checks how much the data can be reduced by applying our approach and Figure 9.10 presents the baseline with the used data. It shows the size of the introduced data set serialized in the different formats. On the left is JSON with about 885 MB. In the middle is JSON compressed with GZIP with the size of ~390 MB and on the right is the size of the data set serialized in Avro (~392 MB). Compressing the data reduces the size about of its half, the Avro serializations has about the same effect.

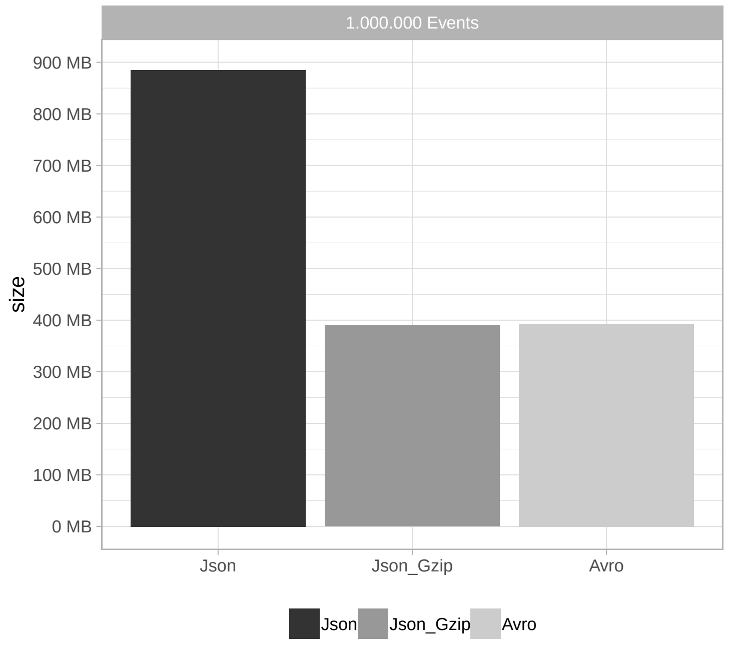


Figure 9.10.: Baseline of the evaluation data set without reductions

After presenting the baseline that shows how much of an impact different serializations of events have on the size of data we present the results of our approach shown in Figure 9.11. For each of the serializations, it is shown how big the size of the data is at the different steps of the data transmission and how much of this size can be reduced. On the left of each serialization the original data at the publisher can be seen. Then there is the amount of data that is transmitted to the message broker after the reduction strategies are applied (*To Broker*). In the evaluation scenario the configuration for the subscriber has a higher value for ε and the events are further reduced. The result of this reduction is shown at (*From Broker*), those are events transmitted from the broker to the subscriber. In the end, the size of the events at the subscriber is depicted, which is the same size as in the beginning at the publisher, showing that the events were reduced and reconstructed again.

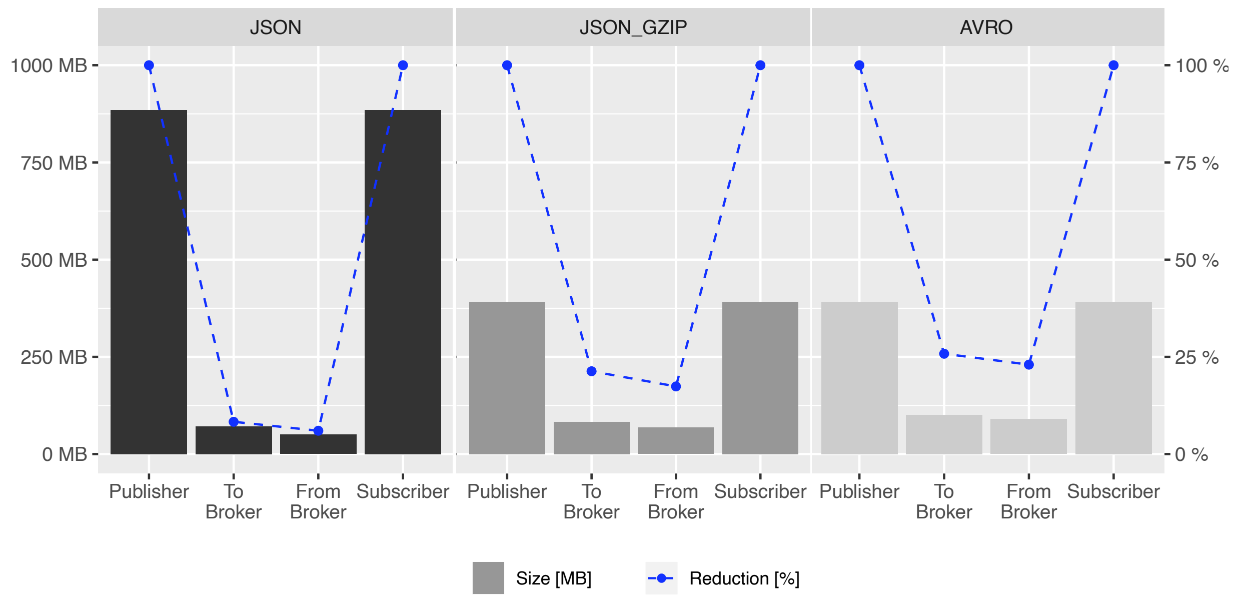


Figure 9.11.: Applying our approach on the data set

The previously described figure shows that the transmitted data is significantly reduced. The result of the JSON events is less than a tenth of the original event size. An explanation for this big reduction is the high sampling frequency, compared to the low inertia of the STCS system. Data is sampled with 1 Hz, however, for example, the measured value of a room temperature is not changing such frequently. Usually, a high frequency ensures that changes are detected quickly, but also has the drawback that a lot of data must be transmitted. The advantage of our approach is that the timely resolution of the data remains high, because the subscriber gets each virtual event, but the amount of transmitted data can be significantly reduced. The results also show that if the required quality of the subscribers decreases, even further reductions are possible.

So far, it was demonstrated how the size of data can be reduced by applying the approaches introduced in Section 7.3. To achieve those reductions, it is necessary to apply calculations at the publisher, the broker, and in the subscriber. The calculations of partial events and reconstruction to virtual events requires both CPU and memory resources. For this evaluation, the logging mechanism to get the size of the events was deactivated to only measure the overhead of the developed algorithms. The evaluation setup consisted of five containers, all running on a single virtual machine. Two containers were responsible for the Kafka environment (Kafka + Zookeeper). Then, there was one publisher (third container) reading the data of the file, reducing the events and transmitting it to the broker. At the broker, a fourth container was deployed that acted as a wrapper around Kafka to reduce the events of the subscribers. Finally, the fifth container ran the subscriber receiving the data. All the following measurements were obtained from the Docker statistics API.

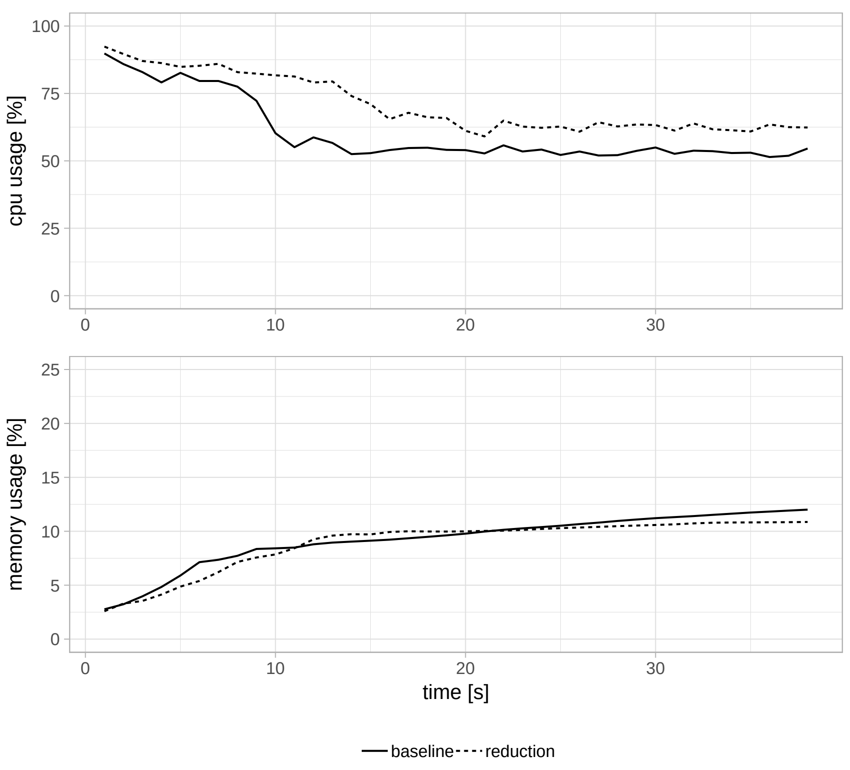


Figure 9.12.: Computation performance of our approach

Figure 9.12 depicts the results of the performance measures. Each test setup ran 10 times and the results were averaged. On the top of the figure, the CPU usage is shown and on the bottom, the memory usage can be seen. Once the publisher started to transmit events, the measurements were started. After all events were sent, it was stopped. For the baseline we ran the same test, but without our approach activated, a publisher sent the data to the broker and a subscriber read it. In the figure, the results of all containers are summarized. For the CPU consumption it can be seen that our approach uses slightly more CPU than the baseline. The reason is the higher computational cost to perform the reduction and reconstruction of the events. This overhead is relatively stable over the whole experiment and only marginally exceeds the baseline. The memory consumption is on the bottom of the figure and shows that over time, our approach requires less memory. A reason for this is that events are reduced. Those reduced events are also stored within the Kafka cluster, requiring less memory compared to processing and storing all the complete events.

Discussion

In this subsection, we presented the evaluation results for the concepts of the message broker introduced in Chapter 6. Therefore, a data set from the smart building domain was chosen and the results were compared to a baseline setup without any reductions. It can be said that our approach works particularly well in scenarios where the data does not change frequently. Even if data changes constantly, the information of the events is transmitted correctly, however the rate of reductions is reduced. It was also shown that the CPU usage is slightly higher compared to the baseline. This effect is somewhat reduced in a production deployment, as the CPU consumption is distributed across many different edge nodes. The memory usage can even be further reduced, because less information is transmitted and must be buffered at the individual components. With this evaluation, we showed how well the introduced approach works. Furthermore, the impact it has on the compute resources was measured.

9.4.3. ML Service

In this section, the performance of the ML Service is evaluated by training a model to detect the milling head degradation as described in Section 3.2. In Chapter 8, a modular framework to automatically train supervised machine learning models was introduced. This framework was realized as a microservice having an interface to manage the training jobs. In StreamPipes, an extension was implemented that provides a GUI that can interact with this service.

Design

For the evaluation, an open source milling data set was chosen [Agogino and Goebel 2007]. It contains multiple measurements of sensors attached to the table and spindle of a CNC machine. The sensors include acoustic emissions, vibrations, and motor current. For the prediction, the flank wear is provided as a label. This should later be automatically detected by the model. New milling heads have a value of zero and the higher this value gets, the blunter the tool is. The goal of the citizen data scientist is to detect if the tool wear is too high and provide feedback to a worker on the shop floor, whether it must be exchanged or not. In the data set, two different kinds of materials were processed, but we did not consider this distinction for the training. A classifier was trained that classifies a whole production run and calculates if the milling head must be replaced or not. We decided to split the data into four classes depending on the value of the flank wear (’<0.1’, ’<0.3’, ’<0.5’, ’<0.7’). If the result is within the first two classes the tool does not need to be replaced and if it is in class ’<0.5’ or ’<0.7’ we decided that it should be replaced. It would be also possible to only use two classes for this problem, but we decided to do it like this to provide a more granular feedback to the worker on the shop floor. For the evaluation we used a server with 8 cores and 32 GB of memory and we measured the training time and analyzed the results of the trained model.

Results

For data preprocessing, we have halved the frequency, additionally we removed several runs with irregular sensor values. In doing so, we took the same approach as described in [Kißkalt et al. 2020] and removed runs with too strong irregular deviations. Also, all runs that did not contain a label were removed from the data set, resulting in 138 samples. For the training all those samples were concatenated adding a short time-break between them and were afterwards uploaded into StreamPipes using a data set adapter. The training was then configured over the GUI and the ML service trained the model as described in Section 8.4. The overall training took about 3 hours and Figure 9.13 depicts a box plot with the time for the individual steps. In the training process, 20 iterations were performed and each took ~520 seconds. Of these, ~420 seconds were needed to extract the features and ~100 seconds to train the models. The resulting best model was a Gradient Boosting Machine (GBM). In total, the best models for the iterations were 16 times a GBM, 2 times extremely randomized trees (XRT), and 2 times Deep Neural Networks (DNN).

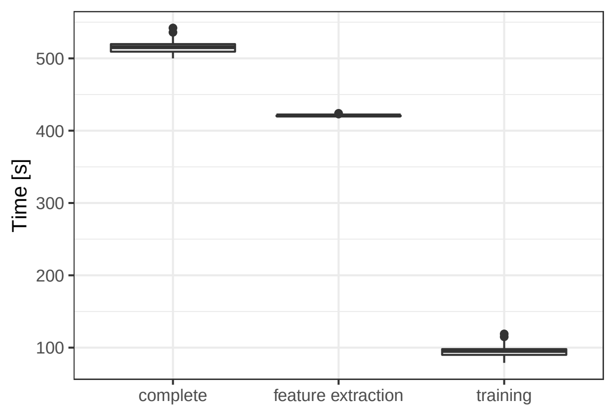
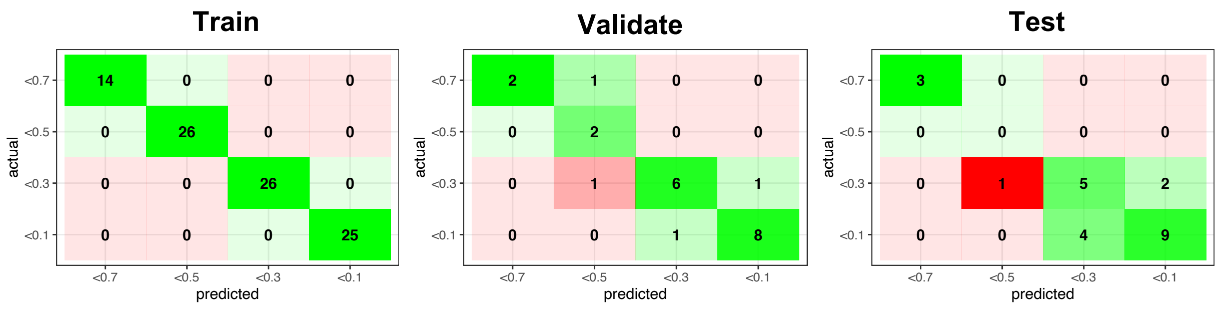


Figure 9.13.: Training time for the CNC milling head classification models

Figure 9.14 shows the results of the training for the best overall model. It depicts three confusion matrices, one for the training set, one for the validation, and one for the test set. The order of the individual milling runs were randomized in the beginning and the split of the three data sets is performed automatically. Within the framework it can be seen that for the train set all predictions were made correctly with no errors. The accuracy for the validation set is about 81.2%. Further, it can be seen that for the decision if the tool should be replaced or not, only one sample was misclassified. This is indicated by the red and green colors in the confusion matrix.

For the test set, the accuracy is about 70.8%, however, only one classification would result in a wrong suggestion for a tool change. One tool was predicted into the class ’<0.5’, which would mean it should be replaced, but was still in class ’<0.3’ (do not replace). It



	<0.7	<0.5	<0.3	<0.1
Train				
<0.7	14	0	0	0
<0.5	0	26	0	0
<0.3	0	0	26	0
<0.1	0	0	0	25
Validate				
<0.7	2	1	0	0
<0.5	0	2	0	0
<0.3	0	1	6	1
<0.1	0	0	1	8
Test				
<0.7	3	0	0	0
<0.5	0	0	0	0
<0.3	0	1	5	2
<0.1	0	0	4	9

Figure 9.14.: Training result for CNC milling head classification

can also be seen that all the misclassifications are only misclassified by one class, which is an indication that the model is capable of deriving the wear based on the sensor values.

Discussion

In this subsection, we evaluated the ML service by creating a session window classification to monitor the tool wear of a milling head in a CNC machine. Therefore, an open data set was selected to train and evaluate a model. It is shown how long the training took for the individual steps and how well the resulting model performs. The results demonstrate that the introduced use case can be solved with the presented ML service and that a model could be trained to monitor the tool wear based on sensor data. Our goal was to provide a solution for even less technical citizen data scientists. Therefore, the results of the ML service were created with the standard configurations and no manual modifications were applied. More advanced citizen data scientist would have the chance to change some of the configuration parameters to further improve the results (e.g. increase training time).

9.5. Summary

In this chapter, we evaluated the concepts that were developed in this work. First, an evaluation strategy was presented that shows how the overall approach is assessed by applying different methods. In the beginning, a conceptual evaluation was performed to validate that all requirements are met. Since one major focus of this work was to enable citizen technologists to analyze industrial event streams, we examined the usability of the system. Therefore, two different user studies were performed. First, a user study targeting citizen developers to connect new data sources, and secondly an expert questioning with citizen data scientists to train new ML models. Both user studies showed that the technical complexity for the performed tasks is reduced and the systems are usable after a short introduction, even for less technical users. For data ingestion, users stated that they will be able to connect new data sources under one minute and during evaluation for the analyze phase it was shown that it is easy to apply new models on event streams. In addition, to the user studies, performance experiments were conducted to evaluate the performance of the individual artifacts. Therefore, the performance for the adapters running on different edge nodes, as well as the reduction capabilities of the broker were tested. Furthermore, the accuracy of the ML service was evaluated. The results show that the methods can cope with the characteristics of industrial event streams. In this evaluation, we showed that the developed approach is both usable, especially for citizen technologists, and that the algorithms are performant enough to perform industrial event stream analytics.

10 Conclusion

In this thesis, we enable citizen technologists to perform industrial event stream analytics without programming effort. To achieve this, they must be enabled to ingest and integrate data produced by sensors in industrial machines, before this data can be analyzed. Thereby, it must be possible to process the large amounts of data that can arise in such Industrial Internet of Things (IIoT) scenarios. This is achieved by introducing methods, models, and tools that automate technical tasks and provide user support, targeting citizen technologists. We distinguish between citizen developers and citizen data scientists. Citizen developers are supported during ingesting data and connecting data sources. The resulting industrial event streams are processed in an optimized way using a distributed message broker, leveraging compute resources of edge devices. Based on this data, citizen data scientists are then enabled to analyze it by training and deploying supervised Machine Learning (ML) models.

This chapter is divided into three sections. First, a summary of this work is given in Section 10.1. Then, the significance of our results is discussed in Section 10.2 and in the end, an outlook for future research topics is provided (Section 10.3).

10.1. Summary

In this section, an overview of the developed approaches and concepts is provided. To facilitate this, the research questions (Section 1.1) are listed below. For each research question, a summary is given that explains how the research question is solved and evaluated in this work. Next, we present the main research question that targets citizen technologists:

How to facilitate data-driven analytics of industrial event streams by citizen technologists?

The goal of this thesis is to enable citizen technologists to perform data-driven analytics. This main research question is subdivided into three specified questions. The first tackles the *ingestion* of industrial data, the second focuses on *integrating* previously connected data. In the last research question, the *analyze* phase is targeted. A high-level overview of the approach was given in Chapter 5. There, it can be seen which roles are involved in the individual phases, what kind of artifacts were developed, and how these interact with each other. The evaluation of this work was provided in Chapter 9.

Research Question 1 (Ingest)**.** *How can we support citizen developers in modeling and connecting heterogeneous industrial event streams?*

This research question focuses on how data is extracted from industrial data sources and ingested in a harmonized middleware. The goal is to enable citizen developers to perform this task by modeling the data source and automatically instantiate *adapters*. A master- / worker paradigm was chosen, where the master manages the system and workers are executing adapters on edge nodes in close proximity to the data sources. In Chapter 6, terms and definitions were presented that are used throughout this thesis, like the underlying *event model*, *virtual sensors*, and *adapters*. To solve the task of ingesting data, the *connect* framework was developed. It consists of a *semantic adapter model* that distinguishes between *specific* and *generic adapters*. The model describes the data source and is then used as a basis for the automatic adapter instantiation. It also contains *preprocessing rules* that are used to harmonize data on ingestion, to ensure a good quality of the data. Based on these rules, functions are automatically derived and executed within adapters. Additionally, support on different levels is provided for citizen developers, in order to reduce technical complexity and configuration effort. For example, a *guided user interaction* was introduced with the adapter marketplace at its core. Users can interact over a Graphical User Interface (GUI) with the marketplace to create new adapter models in a guided step-by-step process. A semantic based user guidance assists citizen developers during the configuration process. Suggestions are provided based on meta-information read from the data source. If no information is available, it can also be provided manually by a user (e.g. semantic types). Various examples show how the approach can be applied in a real-world production line, consisting of multiple machines. In the evaluation, a user study and performance tests were conducted. In the user study, 19 students evaluated the introduced concepts using a prototype. The participants rated the user experience as good and all participants (with a wide range of technical knowledge) were able to connect new data sources. Most of them even stated that they would be able to connect new sources within under one minute. In the performance tests, it was evaluated how the throughput for an adapter is on different edge nodes, with different preprocessing pipelines. The resulting performance was completely sufficient for most use cases on industrial event streams.

Research Question 2 (Integrate)**.** *Based on semantic event models, how can we automatically adapt industrial event streams to meet the demands of infrastructure and subscribers?*

When processing industrial event streams, not all algorithms require data in the highest quality. For example, only one property of the event is relevant for the calculation, or features for ML models are calculated over a window. These cases can be exploited to dynamically adjust the industrial event streams and reduce the demand on the infrastructure. The main objective of this research question is how this can be achieved automatically (Chapter 7). A wrapper around existing topic-based message brokers was provided that leverages both, the schema of events (Chapter 6) and the requirements of data consumers. First, *event reduction strategies* were introduced that can be leveraged to reduce the size or frequency of events. Those strategies are then applied by the presented wrapper for topic-based message brokers. Depending on the type or semantic type of individual event properties, it is possible to decide dynamically, what kind of information must be transmitted. This is possible with the concepts of *partial events* (not containing all property values) and *virtual events* (reconstructed events based on predictions). During runtime the *schema registry* is the central place that manages all the components of the broker (*publishers*, *subscription-transformers*, and *subscribers*). It has knowledge about all event schemas and provides the reduction configuration for the individual components. These decisions are made at runtime to react to new situations, e.g. a new subscriber registers. In Section 7.4, it is shown how events from a typical production line are processed by the broker in different scenarios. For the evaluation, a wrapper around Apache Kafka was implemented and data from the smart building domain was used. It is shown how much data can be reduced and what the computational impact of the algorithms on the infrastructure is.

Research Question 3 (Analyze)**.** *How can citizen data scientists be enabled to use automated machine learning to analyze industrial event streams themselves?*

Research question three focuses on the goal to enable citizen data scientists to leverage techniques from the field of ML to analyze previously ingested and integrated data (Chapter 8). Citizen data scientists should be able to specify the training for supervised ML models, which are then executed automatically. Three types of Machine Learning Pipeline Agents (MLPAs) were introduced that support different learning tasks: *session window classification*, *sliding window classification*, and *forecasting*. A standardized guided workflow was developed to ease the configuration of those training tasks. This workflow was prototypically implemented in a GUI that can be used by citizen data scientists. To automate the training process, we proposed a conceptual framework consisting of multiple building blocks that is capable of training ML models automatically. Integrated event streams can be used as an input without any further manual preprocessing. This framework was realized as a ML Training service that takes the (labeled) training data and automatically performs a training job. During the training, a feature extraction is performed where both, the selected features and window size are learned. To avoid the implementation of the complete ML pipeline, existing Automated Machine Learning (AutoML) frameworks are reused (e.g. for model selection) as a building block within our framework. The best models of a training job are stored in the model repository and a dashboard is provided for the citizen data scientist to evaluate the results of these models. Based on these results, the best model can be selected and deployed as an agent within an Event Processing Network (EPN) to process industrial event streams. The running example shows, how a model can be trained and applied on data produced by a CNC machine to predict the degradation of the milling head. In the evaluation, we performed expert interviews in addition to performance tests. For the interviews, we chose participants that could act as citizen data scientists. The results of the interviews confirmed that the models are simple to train and that it is easy to deploy the resulting models within a pipeline. In addition to the interviews, we conducted performance tests that evaluated the approach based on an open data set containing sensor measurements of a CNC machine.

In summary, we presented how our approaches fulfill the research questions and how we enable citizen technologists to analyze industrial event streams. All of the concepts were prototypically implemented and integrated into StreamPipes.

10.2. Significance of Results

In recent years, progress in the field of Artificial Intelligence (AI) has expanded its use in more and more domains. Since, specialists are often required to create custom implementations, such projects can be very expensive. Therefore, there is a trend towards no code AI solutions that enable less technical users to leverage techniques from the field of ML. This is often applied to image data to train deep learning models. However, also in IIoT environments, more and more data is available, that should be connected and processed. As stated in Section 3.3, several problems exist when processing this industrial data. In this work, methodological foundations were established to create no code AI solutions for industrial event streams, with a major focus on reduction of technical hurdles that often have to be overcome when integrating AI in IIoT scenarios. By reducing the technical barriers, our approach enables more people to benefit from data analytics. Thus, we have created a solution that allows citizen technologists to cover the complete cycle of IIoT analytics - from the connection of (raw) sensor values, over the training of ML models up to the deployment of these models for industrial event streams. By realizing an open source strategy, the developed concepts are disseminated in industry, academia, and the open source community. This also led to an integration for the emerging standard of the Asset Administration Shell (AAS) that was provided in [Jacoby et al. 2021]. In addition, the author of this thesis is a co-initiator of the StreamPipes project, which transferred to the Apache Software Foundation. With a growing user base and developer community, the goal is to ensure a sustainable dissemination. Further, the feedback of the open source community supported the development and improvement of the presented concepts, as well as their alignment with real-world requirements.

10.3. Outlook

In this work, we demonstrated how industrial event streams can be analyzed by citizen technologists. A holistic approach was chosen, which covers the complete cycle. In the following we suggest potential ideas for further research, which are based on the work presented in this thesis.

Adapter Auto Discovery: The technical complexity of ingesting data from industrial data sources is reduced by introducing an adapter model that automatically instantiates adapters. Even if extensive support by the system for the configuration is already provided, it is still a manual task to create these adapters by providing configuration parameters. With the growing amount of descriptions for existing adapter instances, techniques from the field of AI could be leveraged to systematically search for new configurations. For instance new PLCs or new topics within an already created message broker could be detected. The goal would be to automatically detect new data sources and instantly create their adapter model. These could then be suggested to a user, who only needs to confirm the resulting event stream.

Support for Labeling: For the training of supervised ML models, it is necessary to provide labels. In this work, we developed a labeling editor for time-series data that can be used to provide custom labels for event streams in a graphical editor. This simplified data labeling, but still requires manual labor and can be time consuming. Therefore, a promising (future) research approach is to further reduce this manual effort, either by supporting users in labeling or in reducing the required amount of labeled data. The field of few-shot learning could be a promising extension, where existing models are adapted with very few training samples. These resulting models can then be used for the creation of new labels that a user can confirm or reject. Additionally, it would be good if users would only need to manually inspect situations in which the model is uncertain. This could result in a semi-automated approach, where the machine assists the user in quickly providing labels for training.

Online Transfer Learning: We used techniques from the field of batch ML to train models and included a change detector during the deployment to see, if data changes and the model should be retrained. In the future, it could be interesting to include online learning algorithms to constantly update models. This would require a way to regularly provide new labels, especially in production environments. Also, it must be ensured to update the model constantly to leverage the new training data. For this purpose, it would be feasible to use the architecture presented in this thesis for the ML service and extend it with the necessary functionalities.

In this work, we have shown how to automate industrial event stream analytics by introducing methods, models, and tools. The focus is on citizen technologists that are enabled to perform the complete analytics cycle from the sensor data to the analytics results. An extensive implementation was presented that provides an all-in-one solution for citizen technologists. Moreover, the presented concepts form a basis for further research in the field of automated industrial event stream analytics.

Appendix

A Expert Questionnaire

Table A.1.: Results Expert Questionnaire

Id	Question	#1	#2	#3	#4	#5	#6
1	I have high experience with processing IIoT data	2	3	3	3	4	4
2	I am familiar with the basic concepts of ML	5	5	5	4	3	5
3	I have experience with (supervised) machine learning (ML)	5	4	5	4	2	5
4	I know how to train machine learning models in a programming language of my choice	5	4	5	3	2	5
5	I know how to train machine learning models with a graphical user interface of my choice (Knime, Rapidminer, …)	1	1	2	1	2	2
6	I have a lot of data science knowledge and experience	5	3	4	3	3	4
7	I understand the difference between the introduced types of machine learning problems	5	5	5	4	4	5
8	The different types of machine learning problems make sense	5	5	5	4	5	5
9	With the approach I am able to generate new insights of my data	3	5	5	4	4	5
10	I understood what I was doing with the platform	5	5	5	5	4	5
11	To train and deploy my own trained ML models is useful	2		5	5	4	5
12	I think I could directly use the system and apply it on my own data	1	5	4	4	5	4
13	I think I would be able to use the system after some more training	5	5	1	5	4	3
14	The guided process for model creation was intuitive	5	5	4	4	4	5
15	Defining machine learning tasks in a step by step guided process is useful	4	5	5	3	4	5
16	I could understand and interpret the resulting models	3	4	4	4	4	4
17	The provided information gives me confidence to use ML models in IIoT use cases	4	5	3	4	4	4
18	I would need more detailed information about the trained models	4	2	4	5	2	2
19	It would like to further manually optimize the trained models	4	5	4	4	4	3
20	I see the need for such an easy training of ML models	4	5	5	4	5	5
21	The previously trained models are easy to deploy on streaming data	5	5	5	4	4	5
22	I like that I can use ML models when modelling pipelines	5	5	5	5	5	5

Bibliography

Abburu, Sailesh; Berre, Arne J.; Jacoby, Michael; Roman, Dumitru; Stojanovic, Ljiljana; Stojanovic, Nenad (2020). 'COGNITWIN - Hybrid and Cognitive Digital Twins for the Process Industry'. In: *2020 IEEE International Conference on Engineering, Technology and Innovation, ICE/ITMC 2020, Cardiff, United Kingdom, June 15-17, 2020*. IEEE, pp. 1–8. DOI: 10.1109/ICE/ITMC49519.2020.9198403.

Adolphs, P; Auer, S; Bedenbender, H; Billmann, M; Hankel, M; Heidel, R; Hoffmeister, M; Huhle, H; Jochem, M; Kiele, M, et al. (2016). 'Structure of the administration shell. continuation of the development of the reference model for the industrie 4.0 component'. In: *ZVEI and VDI, Status Report*.

Agogino, A; Goebel, K (2007). 'Milling Data Set'. In: *BEST lab, UC Berkeley. NASA Ames Prognostics Data Repository*.

'Air Quality in Europe' (2017). In: ISSN: 1977-8449. DOI: 10.2800/850018.

An, Kyoungho; Khare, Shweta; Gokhale, Aniruddha S.; Hakiri, Akram (2017). 'An Autonomous and Dynamic Coordination and Discovery Service for Wide-Area Peer-to-peer Publish/Subscribe: Experience Paper'. In: *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, DEBS 2017, Barcelona, Spain, June 19-23, 2017*. ACM, pp. 239–248. DOI: 10.1145/3093742.3093910.

Ari, Ismail; Olmezogullari, Erdi; Çelebi, Ömer Faruk (2012). 'Data stream analytics and mining in the cloud'. In: *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings, CloudCom 2012, Taipei, Taiwan, December 3-6, 2012*. IEEE Computer Society, pp. 857–862. DOI: 10.1109/CloudCom.2012.6427563.

Armbrust, Michael; Fox, Armando; Griffith, Rean; Joseph, Anthony D; Katz, Randy; Konwinski, Andy; Lee, Gunho; Patterson, David; Rabkin, Ariel; Stoica, Ion, et al. (2010). 'A view of cloud computing'. In: *Communications of the ACM* 53.4, pp. 50–58.

Baheti, Radhakisan; Gill, Helen (2011). 'Cyber-physical systems'. In: *The impact of control technology* 12.1, pp. 161–166.

Bahri, Maroua; Veloso, Bruno; Bifet, Albert; Gama, João (2020). 'AutoML for Stream k-Nearest Neighbors Classification'. In: *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, pp. 597–602.

Berger, Christoph; Nägele, Juliane; Drescher, Benny; Reinhart, Gunther (2017). 'Application of CPS in machine tools'. In: *Industrial Internet of Things*. Springer, pp. 375–400.

Berthold, Michael R.; Cebron, Nicolas; Dill, Fabian; Gabriel, Thomas R.; Kötter, Tobias; Meinl, Thorsten; Ohl, Peter; Thiel, Kilian; Wiswedel, Bernd (2009). 'KNIME - the Konstanz information miner: version 2.0 and beyond'. In: *SIGKDD Explor.* 11.1, pp. 26–31. DOI: 10.1145/1656274.1656280.

Bhowmik, Sukanya; Tariq, Muhammad Adnan; Grunert, Jonas; Rothermel, Kurt (2016). 'Bandwidth-efficient content-based routing on software-defined networks'. In: *Proceed-*

ings of the 10th ACM International Conference on Distributed and Event-based Systems, DEBS '16, Irvine, CA, USA, June 20 - 24, 2016. ACM, pp. 137–144. DOI: 10.1145/2933267.2933310.

Bifet, Albert; Gavaldà, Ricard; Holmes, Geoff; Pfahringer, Bernhard (2018). *Machine learning for data streams: with practical examples in MOA*. MIT Press.

Bifet, Albert; Holmes, Geoff; Pfahringer, Bernhard; Kranen, Philipp; Kremer, Hardy; Jansen, Timm; Seidl, Thomas (2010). 'MOA: Massive Online Analysis, a Framework for Stream Classification and Clustering'. In: *Proceedings of the First Workshop on Applications of Pattern Analysis, WAPA 2010, Cumberland Lodge, Windsor, UK, September 1-3, 2010*. Vol. 11. JMLR Proceedings. JMLR.org, pp. 44–50.

Bock, Olaf; Baetge, Ingmar; Nicklisch, Andreas (2014). 'hroot: Hamburg Registration and Organization Online Tool'. In: *European Economic Review* 71, pp. 117–120. ISSN: 0014-2921. DOI: https://doi.org/10.1016/j.euroecorev.2014.07.003.

Bose, Ranjit (2009). 'Advanced analytics: opportunities and challenges'. In: *Industrial Management & Data Systems*.

Boyes, Hugh; Hallaq, Bil; Cunningham, Joe; Watson, Tim (2018). 'The industrial internet of things (IIoT): An analysis framework'. In: *Comput. Ind.* 101, pp. 1–12. DOI: 10.1016/j.compind.2018.04.015.

Brooke, John et al. (1996). 'SUS-A quick and dirty usability scale'. In: *Usability evaluation in industry* 189.194, pp. 4–7.

Bröring, Arne; Ziller, Andreas; Charpenay, Victor; Thuluva, Aparna Saisree; Anicic, Darko; Schmid, Stefan; Zappa, Achille; Linares, Mari Paz; Mikkelsen, Lars Møller; Seidel, Christian (2018). 'The BIG IoTAPI -Semantically Enabling IoT Interoperability'. In: *IEEE Pervasive Comput.* 17.4, pp. 41–51. DOI: 10.1109/MPRV.2018.2873566.

Cao, Longbing (2017). 'Data Science: A Comprehensive Overview'. In: *ACM Comput. Surv.* 50.3, 43:1–43:42. DOI: 10.1145/3076253.

Celik, Bilge; Vanschoren, Joaquin (2020). 'Adaptation Strategies for Automated Machine Learning on Evolving Data'. In: *arXiv preprint arXiv:2006.06480*.

Chaves, John; Curry, Chris; Grossman, Robert L.; Locke, David; Vejcik, Steve (2006). 'Augustus: The Design and Architecture of a PMML-Based Scoring Engine'. In: *Proceedings of the 4th International Workshop on Data Mining Standards, Services and Platforms*. DMSSP '06. Philadelphia, Pennsylvania: Association for Computing Machinery, pp. 38–46. ISBN: 159593443X. DOI: 10.1145/1289612.1289616.

Commission, International Electrotechnical et al. (2016). 'IEC TR 62541-1: 2016-OPC unified architecture-Part 1: Overview and concepts'. In: *IEC, Geneva, CH, Technical Report*.

Cook, Darren (2016). *Practical machine learning with H2O: powerful, scalable techniques for deep learning and AI*. " O'Reilly Media, Inc."

Davenport, Thomas H; Patil, DJ (2012). 'Data scientist'. In: *Harvard business review* 90.5, pp. 70–76.

Doblander, Christoph; Ghinaiya, Tanuj; Zhang, Kaiwen; Jacobsen, Hans-Arno (2016). 'Shared dictionary compression in publish/subscribe systems'. In: *Proceedings of the*

10th ACM International Conference on Distributed and Event-based Systems, DEBS '16, Irvine, CA, USA, June 20 - 24, 2016. ACM, pp. 117–124. DOI: 10.1145/2933267.2933308.

Etzion, Opher; Niblett, Peter (2010). *Event Processing in Action*. Manning Publications Company. ISBN: 978-1-935182-21-4.

Eugster, Patrick Th; Felber, Pascal A; Guerraoui, Rachid; Kermarrec, Anne-Marie (2003). 'The many faces of publish/subscribe'. In: *ACM computing surveys (CSUR)* 35.2, pp. 114–131.

Feurer, Matthias; Klein, Aaron; Eggensperger, Katharina; Springenberg, Jost Tobias; Blum, Manuel; Hutter, Frank (2019). 'Auto-sklearn: efficient and robust automated machine learning'. In: *Automated Machine Learning*. Springer, Cham, pp. 113–134.

Feurer, Matthias; Klein, Aaron; Eggensperger, Katharina; Springenberg, Jost; Blum, Manuel; Hutter, Frank (2015). 'Efficient and Robust Automated Machine Learning'. In: *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., pp. 2962–2970.

Figo, Davide; Diniz, Pedro C.; Ferreira, Diogo R.; Cardoso, João M. P. (2010). 'Preprocessing techniques for context recognition from accelerometer data'. In: *Pers. Ubiquitous Comput.* 14.7, pp. 645–662. DOI: 10.1007/s00779-010-0293-9.

Foundation, OPC; Institute, MTConnect (2019). *OPC Unified Architecture for MTConnect*. URL: https://www.mtconnect.org/opc-ua-companion-specification (visited on 06/05/2021).

Fowler, Martin (2010). *Domain-specific languages*. Pearson Education.

GE (2015). *Industrial Internet Insights Report - For 2015*. URL: https://www.ge.com/digital/sites/default/files/download_assets/industrial-internet-insights-report_1.pdf (visited on 01/19/2021).

Gijsbers, Pieter; LeDell, Erin; Thomas, Janek; Poirier, Sébastien; Bischl, Bernd; Vanschoren, Joaquin (2019). 'An Open Source AutoML Benchmark'. In: *CoRR* abs/1907.00909. arXiv: 1907.00909.

Glinz, Martin (2011). 'A glossary of requirements engineering terminology'. In: *Standard Glossary of the Certified Professional for Requirements Engineering (CPRE) Studies and Exam, Version* 1, p. 56.

Gölzer, Philipp; Cato, Patrick; Amberg, Michael (2015). 'Data Processing Requirements of Industry 4.0 - Use Cases for Big Data Applications'. In: *23rd European Conference on Information Systems, ECIS 2015, Münster, Germany, May 26-29, 2015*.

Gosewehr, Frederik; Wermann, Jeffrey; Borsych, Waldemar; Colombo, Armando Walter (2017). 'Specification and design of an industrial manufacturing middleware'. In: *15th IEEE International Conference on Industrial Informatics, INDIN 2017, Emden, Germany, July 24-26, 2017*. IEEE, pp. 1160–1166. DOI: 10.1109/INDIN.2017.8104937.

Gröger, Christoph (2018). 'Building an Industry 4.0 Analytics Platform - Practical Challenges, Approaches and Future Research Directions'. In: *Datenbank-Spektrum* 18.1, pp. 5–14. DOI: 10.1007/s13222-018-0273-1.

Gröger, Christoph; Kassner, Laura; Hoos, Eva; Königsberger, Jan; Kiefer, Cornelia; Silcher, Stefan; Mitschang, Bernhard (2016). 'The Data-driven Factory - Leveraging Big Industrial Data for Agile, Learning and Human-centric Manufacturing'. In: *ICEIS 2016 - Proceedings of the 18th International Conference on Enterprise Information Systems, Volume 1, Rome, Italy, April 25-28, 2016*. SciTePress, pp. 40–52. DOI: 10.5220/0005831500400052.

Guazzelli, Alex; Zeller, Michael; Lin, Wen-Ching; Williams, Graham, et al. (2009). 'PMML: An open standard for sharing models'. In: *The R Journal* 1.1, pp. 60–65.

Han, Jiawei; Kamber, Micheline; Pei, Jian (2011). *Data Mining: Concepts and Techniques, 3rd edition*. Morgan Kaufmann. ISBN: 978-0123814791.

Harth, Natascha; Anagnostopoulos, Christos (2017). 'Quality-aware aggregation & predictive analytics at the edge'. In: *2017 IEEE International Conference on Big Data, BigData 2017, Boston, MA, USA, December 11-14, 2017*. IEEE Computer Society, pp. 17–26. DOI: 10.1109/BigData.2017.8257907.

Hevner, Alan R.; March, Salvatore T.; Park, Jinsoo; Ram, Sudha (2004). 'Design Science in Information Systems Research'. In: *MIS Q.* 28.1, pp. 75–105.

Huber, Steffen; Wiemer, Hajo; Schneider, Dorothea; Ihlenfeldt, Steffen (2019). 'DMME: Data mining methodology for engineering applications–a holistic extension to the CRISP-DM model'. In: *Procedia Cirp* 79, pp. 403–408.

Hueske, Fabian; Kalavri, Vasiliki (2019). *Stream Processing with Apache Flink: Fundamentals, Implementation, and Operation of Streaming Applications*. O'Reilly Media.

Hufnagel, Johann; Vogel-Heuser, Birgit (2015). 'Data integration in manufacturing industry: Model-based integration of data distributed from ERP to PLC'. In: *13th IEEE International Conference on Industrial Informatics, INDIN 2015, Cambridge, United Kingdom, July 22-24, 2015*. IEEE, pp. 275–281. DOI: 10.1109/INDIN.2015.7281747.

Ismail, Bukhary Ikhwan; Mostajeran Goortani, Ehsan; Ab Karim, Mohd Bazli; Ming Tat, Wong; Setapa, Sharipah; Luke, Jing Yuan; Hong Hoe, Ong (2015). 'Evaluation of Docker as Edge Computing Platform'. en. In: IEEE, pp. 130–135. ISBN: 978-1-4673-9434-5. DOI: 10.1109/ICOS.2015.7377291.

Jacoby, Michael; Jovicic, Branislav; Stojanovic, Ljiljana; Stojanović, Nenad (2021). 'An Approach for Realizing Hybrid Digital Twins Using Asset Administration Shells and Apache StreamPipes'. In: *Information* 12.6, p. 217.

Jeschke, Sabina; Brecher, Christian; Meisen, Tobias; Özdemir, Denis; Eschert, Tim (2017). 'Industrial internet of things and cyber manufacturing systems'. In: *Industrial internet of things*. Springer, pp. 3–19.

Jiang, Hongbo; Jin, Shudong; Wang, Chonggang (2011). 'Prediction or Not? An Energy-Efficient Framework for Clustering-Based Data Collection in Wireless Sensor Networks'. In: *IEEE Trans. Parallel Distrib. Syst.* 22.6, pp. 1064–1071. DOI: 10.1109/TPDS.2010.174.

Jirkovský, V.; Obitko, M.; Mařík, V. (2017). 'Understanding Data Heterogeneity in the Context of Cyber-Physical Systems Integration'. In: *IEEE Trans. Ind. Informatics* 13.2, pp. 660–667.

Kanawaday, Ameeth; Sane, Aditya (2017). 'Machine learning for predictive maintenance of industrial machines using IoT sensor data'. In: *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*. IEEE, pp. 87–90.

Khalajzadeh, Hourieh; Abdelrazek, Mohamed; Grundy, John; Hosking, John G.; He, Qiang (2018). 'A Survey of Current End-User Data Analytics Tool Support'. In: *2018 IEEE International Congress on Big Data (BigDataCongress)*. IEEE Computer Society, pp. 41–48. DOI: 10.1109/BigDataCongress.2018.00013.

Khalajzadeh, Hourieh; Abdelrazek, Mohamed; Grundy, John; Hosking, John; He, Qiang (2019). 'Bidaml: A suite of visual languages for supporting end-user data analytics'. In: *2019 IEEE International Congress on Big Data (BigDataCongress)*. IEEE, pp. 93–97.

Kirmse, Andreas; Kraus, Vadim; Hoffmann, Max; Meisen, Tobias (2018). '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, ICEIS 2018, Funchal, Madeira, Portugal, March 21-24, 2018, Volume 1*. SciTePress, pp. 175–182. DOI: 10.5220/0006776701750182.

Kißkalt, Dominik; Mayr, Andreas; Lutz, Benjamin; Rögele, Annelie; Franke, Jörg (2020). 'Streamlining the development of data-driven industrial applications by automated machine learning'. In: *Procedia CIRP* 93, pp. 401–406.

Kotsiantis, Sotiris B. (2007). 'Supervised Machine Learning: A Review of Classification Techniques'. In: *Emerging Artificial Intelligence Applications in Computer Engineering - Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*. Vol. 160. Frontiers in Artificial Intelligence and Applications. IOS Press, pp. 3–24.

Kuppevelt, Dafne E. van; Meijer, C.; Huber, F.; Ploeg, A. van der; Georgievska, S.; Hees, Vincent T. van (2020). 'Mcfly: Automated deep learning on time series'. In: *SoftwareX* 12, p. 100548. DOI: 10.1016/j.softx.2020.100548.

Lade, Prasanth; Ghosh, Rumi; Srinivasan, Soundar (2017). 'Manufacturing Analytics and Industrial Internet of Things'. In: *IEEE Intell. Syst.* 32.3, pp. 74–79. DOI: 10.1109/MIS.2017.49.

Landset, Sara; Khoshgoftaar, Taghi M.; Richter, Aaron N.; Hasanin, Tawfiq (2015). 'A survey of open source tools for machine learning with big data in the Hadoop ecosystem'. In: *J. Big Data* 2, p. 24. DOI: 10.1186/s40537-015-0032-1.

Laverman, J.; Grewe, D.; Weinmann, O.; Wagner, M.; Schildt, S. (2016). 'Integrating Vehicular Data into Smart Home IoT Systems Using Eclipse Vorto'. In: *2016 IEEE 84th Vehicular Technology Conference (VTC-Fall)*, pp. 1–5. DOI: 10.1109/VTCFall.2016.7881049.

Lehmberg, Oliver; Bizer, Christian; Brinkmann, Alexander (2017). 'WInte.r - A Web Data Integration Framework'. In: *Proceedings of the ISWC 2017 Posters & Demonstrations and Industry Tracks co-located with 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 23rd - to - 25th, 2017*. Vol. 1963. CEUR Workshop Proceedings. CEUR-WS.org.

Liu, Chao; Vengayil, Hrishikesh; Lu, Yuqian; Xu, Xun (2019). 'A cyber-physical machine tools platform using OPC UA and MTConnect'. In: *Journal of Manufacturing Systems* 51, pp. 61–74.

Luckham, David (2002). *The power of events*. Vol. 204. Addison-Wesley Reading.

Madrid, Jorge G.; Escalante, Hugo Jair; Morales, Eduardo F.; Tu, Wei-Wei; Yu, Yang; Sun-Hosoya, Lisheng; Guyon, Isabelle; Sebag, Michèle (2019). 'Towards AutoML in the presence of Drift: first results'. In: *CoRR* abs/1907.10772. arXiv: 1907.10772.

Maeda, Kazuaki (2012). 'Performance evaluation of object serialization libraries in XML, JSON and binary formats'. In: *2012 Second International Conference on Digital Information and Communication Technology and it's Applications (DICTAP), Bangkok, Thailand, May 16-18, 2012*. IEEE, pp. 177–182. DOI: 10.1109/DICTAP.2012.6215346.

Mahapatra, Tanmaya (2019). 'High-level Graphical Programming for Big Data Applications'. PhD thesis. Technical University of Munich, Germany.

Maier, Alexander; Schriegel, Sebastian; Niggemann, Oliver (2017). 'Big data and machine learning for the smart factory—Solutions for condition monitoring, diagnosis and optimization'. In: *Industrial Internet of Things*. Springer, pp. 473–485.

Meratnia, Nirvana; By, Rolf A. de (2004). 'Spatiotemporal Compression Techniques for Moving Point Objects'. In: *Advances in Database Technology - EDBT 2004, 9th International Conference on Extending Database Technology, Heraklion, Crete, Greece, March 14-18, 2004, Proceedings*. Vol. 2992. Lecture Notes in Computer Science. Springer, pp. 765–782. DOI: 10.1007/978-3-540-24741-8_44.

Michalczyk, Sven; Nadj, Mario; Azarfar, Darius; Maedche, Alexander; Gröger, Christoph (2020). 'A State-of-the-Art Overview and Future Research Avenues of Self-Service Business Intelligence and Analytics.' In: *ECIS*.

Mitchell, Tom Michael (1997). *Machine Learning -*. New York: McGraw-Hill Education. ISBN: 978-0-070-42807-2.

Morales, Gianmarco De Francisci; Bifet, Albert (2015). 'SAMOA: scalable advanced massive online analysis'. In: *J. Mach. Learn. Res.* 16, pp. 149–153.

MTConnect, Institute (2021). *MTConnect Standard 1.6.0*. URL: https://www.mtconnect.org/ (visited on 04/05/2021).

Neumann, Peter (2007). 'Communication in industrial automation—What is going on?' In: *Control Engineering Practice* 15.11, pp. 1332–1347.

Olson, Randal S; Moore, Jason H (2016). 'TPOT: A tree-based pipeline optimization tool for automating machine learning'. In: *Workshop on automatic machine learning*. PMLR, pp. 66–74.

Oltrogge, Marten; Derr, Erik; Stransky, Christian; Acar, Yasemin; Fahl, Sascha; Rossow, Christian; Pellegrino, Giancarlo; Bugiel, Sven; Backes, Michael (2018). 'The Rise of the Citizen Developer: Assessing the Security Impact of Online App Generators'. In: *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. IEEE Computer Society, pp. 634–647. DOI: 10.1109/SP.2018.00005.

Page, Ewan S (1954). 'Continuous inspection schemes'. In: *Biometrika* 41.1/2, pp. 100–115.

Parasuraman, Ananthanarayanan (2000). 'Technology Readiness Index (TRI) a multiple-item scale to measure readiness to embrace new technologies'. In: *Journal of service research* 2.4, pp. 307–320.

Patel, Pankesh; Cassou, Damien (2015). 'Enabling high-level application development for the Internet of Things'. In: *J. Syst. Softw.* 103, pp. 62–84. DOI: 10.1016/j.jss.2015.01.027.

Pauker, Florian; Frühwirth, Thomas; Kittl, Burkhard; Kastner, Wolfgang (2016). 'A systematic approach to OPC UA information model design'. In: *Procedia CIRP* 57, pp. 321–326.

Pedregosa, Fabian; Varoquaux, Gaël; Gramfort, Alexandre; Michel, Vincent; Thirion, Bertrand; Grisel, Olivier; Blondel, Mathieu; Prettenhofer, Peter; Weiss, Ron; Dubourg, Vincent; VanderPlas, Jake; Passos, Alexandre; Cournapeau, David; Brucher, Matthieu; Perrot, Matthieu; Duchesnay, Edouard (2011). 'Scikit-learn: Machine Learning in Python'. In: *J. Mach. Learn. Res.* 12, pp. 2825–2830.

Pfeil, Matthias; Bartoschek, Thomas; Wirwahn, Jan Alexander (2015). '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. 1, p. 39.

Pfisterer, Dennis; Römer, Kay; Bimschas, Daniel; Kleine, Oliver; Mietz, Richard; Truong, Cuong; Hasemann, Henning; Kröller, Alexander; Pagel, Max; Hauswirth, Manfred; Karnstedt, Marcel; Leggieri, Myriam; Passant, Alexandre; Richardson, Ray (2011). 'SPITFIRE: toward a semantic web of things'. In: *IEEE Communications Magazine* 49.11, pp. 40–48. DOI: 10.1109/MCOM.2011.6069708.

Plazas, Julián Eduardo; Bimonte, Sandro; Schneider, Michel; Vaulx, Christophe de; Corrales, Juan Carlos (2020). 'Self-service Business Intelligence over On-Demand IoT Data: A New Design Methodology Based on Rapid Prototyping'. In: *New Trends in Databases and Information Systems - ADBIS 2020 Short Papers, Lyon, France, August 25-27, 2020, Proceedings*. Vol. 1259. Communications in Computer and Information Science. Springer, pp. 84–93. DOI: 10.1007/978-3-030-54623-6_8.

Portugal, Ivens; Alencar, Paulo; Cowan, Donald (2016). 'A preliminary survey on domain-specific languages for machine learning in big data'. In: *2016 IEEE International Conference on Software Science, Technology and Engineering (SWSTE)*. IEEE, pp. 108–110.

Rausch, Thomas; Dustdar, Schahram; Ranjan, Rajiv (2018a). 'Osmotic Message-Oriented Middleware for the Internet of Things'. In: *IEEE Cloud Comput.* 5.2, pp. 17–25. DOI: 10.1109/MCC.2018.022171663.

Rausch, Thomas; Nastic, Stefan; Dustdar, Schahram (2018b). 'EMMA: Distributed QoS-Aware MQTT Middleware for Edge Computing Applications'. In: *2018 IEEE International Conference on Cloud Engineering, IC2E 2018, Orlando, FL, USA, April 17-20, 2018*. IEEE Computer Society, pp. 191–197. DOI: 10.1109/IC2E.2018.00043.

Real, Esteban; Aggarwal, Alok; Huang, Yanping; Le, Quoc V (2019). 'Regularized evolution for image classifier architecture search'. In: *Proceedings of the aaai conference on artificial intelligence*. Vol. 33. 01, pp. 4780–4789.

Rehman, Muhammad Habib Ur; Yaqoob, Ibrar; Salah, Khaled; Imran, Muhammad; Jayaraman, Prem Prakash; Perera, Charith (2019). 'The role of big data analytics in industrial Internet of Things'. In: *Future Gener. Comput. Syst.* 99, pp. 247–259. DOI: 10.1016/j.future.2019.04.020.

Riemer, Dominik (2016). ‘Methods and Tools for Management of Distributed Event Processing Applications’. PhD thesis. Karlsruhe Institute of Technology, Germany.

Rose, Karen; Eldridge, Scott; Chapin, Lyman (2015). ‘The internet of things: An overview’. In: *The Internet Society (ISOC)* 80, pp. 1–50.

Runkler, Thomas A. (2016). *Data Analytics - Models and Algorithms for Intelligent Data Analysis, Second Edition*. Springer. ISBN: 978-3-658-14074-8. DOI: 10.1007/978-3-658-14075-5.

Rüth, Jan; Schmidt, Florian; Serror, Martin; Wehrle, Klaus; Zimmermann, Torsten (2017). ‘Communication and Networking for the industrial Internet of Things’. In: *Industrial Internet of Things*. Springer, pp. 317–346.

Salehi, Pooya; Zhang, Kaiwen; Jacobsen, Hans-Arno (2017). ‘PopSub: Improving Resource Utilization in Distributed Content-based Publish/Subscribe Systems’. In: *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, DEBS 2017, Barcelona, Spain, June 19-23, 2017*. ACM, pp. 88–99. DOI: 10.1145/3093742.3093915.

Satyanarayanan, Mahadev (2017). ‘The Emergence of Edge Computing’. In: *Computer* 50.1, pp. 30–39. DOI: 10.1109/MC.2017.9.

Schilberg, Daniel; Hoffmann, Max; Schmitz, Sebastian; Meisen, Tobias (2017). ‘Interoperability in smart automation of cyber physical systems’. In: *Industrial Internet of Things*. Springer, pp. 261–286.

Schleipen, Miriam; Gilani, Syed-Shiraz; Bischoff, Tino; Pfrommer, Julius (2016). ‘OPC UA & Industrie 4.0-enabling technology with high diversity and variability’. In: *Procedia Cirp* 57, pp. 315–320.

Schrepp, Martin; Hinderks, Andreas; Thomaschewski, Jörg (2014). ‘Applying the User Experience Questionnaire (UEQ) in Different Evaluation Scenarios’. In: *Design, User Experience, and Usability. Theories, Methods, and Tools for Designing the User Experience - Third International Conference, DUXU 2014, Held as Part of HCI International 2014, Heraklion, Crete, Greece, June 22-27, 2014, Proceedings, Part I*. Vol. 8517. Lecture Notes in Computer Science. Springer, pp. 383–392. DOI: 10.1007/978-3-319-07668-3_37.

Sculley, D.; Holt, Gary; Golovin, Daniel; Davydov, Eugene; Phillips, Todd; Ebner, Dietmar; Chaudhary, Vinay; Young, Michael; Crespo, Jean-François; Dennison, Dan (2015). ‘Hidden Technical Debt in Machine Learning Systems’. In: *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pp. 2503–2511.

Shearer, Colin (2000). ‘The CRISP-DM model: the new blueprint for data mining’. In: *Journal of data warehousing* 5.4, pp. 13–22.

Shi, Weisong; Cao, Jie; Zhang, Quan; Li, Youhuizi; Xu, Lanyu (2016). ‘Edge Computing: Vision and Challenges’. In: *IEEE Internet Things J.* 3.5, pp. 637–646. DOI: 10.1109/JIOT.2016.2579198.

Shi, Weisong; Dustdar, Schahram (2016). ‘The Promise of Edge Computing’. In: *Computer* 49.5, pp. 78–81. DOI: 10.1109/MC.2016.145.

Sommer, P.; Schellroth, F.; Fischer, M.; Schlechtendahl, Jan (2018). ‘Message-oriented Middleware for Industrial Production Systems’. In: *14th IEEE International Conference* *on Automation Science and Engineering, CASE 2018, Munich, Germany, August 20-24, 2018*. IEEE, pp. 1217–1223. DOI: 10.1109/COASE.2018.8560493.

Tantik, Erdal; Anderl, Reiner (2017). 'Integrated data model and structure for the asset administration shell in Industrie 4.0'. In: *Procedia CIRP* 60, pp. 86–91.

Trunzer, Emanuel; Calà, Ambra; Leitão, Paulo; Gepp, Michael; Kinghorst, Jakob; Lüder, Arndt; Schauerte, Hubertus; Reifferscheid, Markus; Vogel-Heuser, Birgit (2019). 'System architectures for Industrie 4.0 applications'. In: *Production Engineering* 13.3-4, pp. 247–257.

Trunzer, Emanuel; Kirchen, Iris; Folmer, Jens; Koltun, Gennadiy; Vogel-Heuser, Birgit (2017). 'A flexible architecture for data mining from heterogeneous data sources in automated production systems'. In: *IEEE International Conference on Industrial Technology, ICIT 2017, Toronto, ON, Canada, March 22-25, 2017*. IEEE, pp. 1106–1111. DOI: 10.1109/ICIT.2017.7915517.

Xu, Hansong; Yu, Wei; Griffith, David W.; Golmie, Nada (2018). 'A Survey on Industrial Internet of Things: A Cyber-Physical Systems Perspective'. In: *IEEE Access* 6, pp. 78238–78259. DOI: 10.1109/ACCESS.2018.2884906.

Zehnder, Philipp; Riemer, Dominik (2017). 'Modeling self-service machine-learning agents for distributed stream processing'. In: *2017 IEEE International Conference on Big Data, BigData 2017, Boston, MA, USA, December 11-14, 2017*. IEEE Computer Society, pp. 2203–2212. DOI: 10.1109/BigData.2017.8258170.

Zehnder, Philipp; Riemer, Dominik (2018). 'Representing Industrial Data Streams in Digital Twins using Semantic Labeling'. In: *IEEE International Conference on Big Data, Big Data 2018, Seattle, WA, USA, December 10-13, 2018*. IEEE, pp. 4223–4226. DOI: 10.1109/BigData.2018.8622400.

Zehnder, Philipp; Wiener, Patrick; Riemer, Dominik (2019). 'Using Virtual Events for Edge-based Data Stream Reduction in Distributed Publish/Subscribe Systems'. In: *3rd IEEE International Conference on Fog and Edge Computing, ICFEC 2019, Larnaca, Cyprus, May 14-17, 2019*. IEEE, pp. 1–10. DOI: 10.1109/CFEC.2019.8733146.

Zehnder, Philipp; Wiener, Patrick; Straub, Tim; Riemer, Dominik (2020). 'StreamPipes Connect: Semantics-Based Edge Adapters for the IIoT'. In: *The Semantic Web - 17th International Conference, ESWC 2020, Heraklion, Crete, Greece, May 31-June 4, 2020, Proceedings*. Vol. 12123. Lecture Notes in Computer Science, pp. 665–680. DOI: 10.1007/978-3-030-49461-2_39.