Valon Gjokaj, BSc.

# A Smart Data Service for Data Collection in a Smart Factory

**Master's Thesis**

to achieve the university degree of

Master of Science

Master's degree programme: Software Engineering and Management

submitted to

**Graz University of Technology**

Supervisor

Dipl.-Ing. Dr.techn. Roman Kern

Institute for Interactive Systems and Data Science
Head: Univ.-Prof. Dipl.-Ing. Dr. Stefanie Lindstaedt

Graz, October 2019

# Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.

| Date | Signature |
| --- | --- |

# Acknowledgments

Many individuals have helped me during the period that this work has been done. For the successful completion of this Thesis, I would like to thank my supervisor Dipl.-Ing. Dr.techn. Roman Kern. Without his expert advice and encouragement, this work would have been hardly achieved. I would also like to thank my sacred family. They helped me in the realization of this work by supporting me in the best ways possible all this time.

# Abstract

With the increasing development of technology nowadays a diverse number of possibilities have arisen but new challenges come into play too. These developments have made it possible to move towards Industry 4.0 and the so-called Smart Factories. It is the new manufacturing system where everything is supposed to be connected. This can have a big impact like in supporting decision making, in shortening the production life-cycle or in enabling highly customizable product manufacturing, which can be achieved by making use of the right data. The data that flows within a Smart Factory can be of an enormous volume, is heterogeneous and they do not come only from a single data source. However, the systems have to bring the created data into play somehow. The challenge here is to transform the created Big Data to the more valuable Smart Data, so that later in the process, analytics like Predictive Maintenance or Retrospective Analysis can be performed successfully on those data. This is also the aim of this Master's Thesis. In order to solve this problem, a prototype service called Smart Data Service has been developed so that the raw incoming data streams are aggregated and put together in a more reduced but valuable format, known as Smart Data. For the testing purposes and the evaluation of the work, it was necessary to additionally develop a Smart Factory Simulator, which is supposed to emulate different scenarios of a manufacturing setup. Two use cases have been taken into consideration for evaluating the Smart Data Service - aggregating data that would be useful for applying Retrospective Analysis and aggregating data that would be useful for Predictive Maintenance. Finally, the results show that the aggregated Smart Data can have considerable value for performing Retrospective Analysis as well as Predictive Maintenance.

# Contents

# List of Figures

List of Figures

# 1 Introduction

It is evident that technology has been taking massive big steps and in recent decades it has moved forward at a very quick rate. It certainly has impacted many areas like communications, medicine [6] and health care [12], journalism, it has impacted how the information is spread, how we socialize [8], our education systems [7], our diverse industries like the food industry, the automobile industry [21], the aviation industry, it has even affected the way we produce goods [17].

One of the areas that have been affected a lot and are of special interest are different industries and the way we manufacture products. This brings us to a word that we hear a lot nowadays - Industry 4.0 or the Fourth Industrial Revolution. What is it exactly? To explain it, it is necessary to understand the history of the industrial revolution.

An industrial revolution, is considered to be a disruption and a fundamental change in the way industries work, and how they affect the economic and social systems. Throughout history, there are four well-known phases of the industrial revolution.

- The first industrial revolution began around the year 1760 until the year 1840. It is characterized by the invention of steamed engines and the construction of railroads [22].
- The use of electricity and assembly lines for mass production symbolize the second industrial revolution which took place between 1870 and 1914 [22]. Assembly lines and mass production got particularly popularized by Ford Motor Company when they made use of the method for the production of Ford Model T.
- The third industrial revolution is known as the computer era which started in the year 1960, and spread with the development of personal computers and continued until the invention of the internet in the 1990s [22].
- The fourth industrial revolution is considered to have begun with the start of the 21st century. This one is distinguished by powerful developed machines

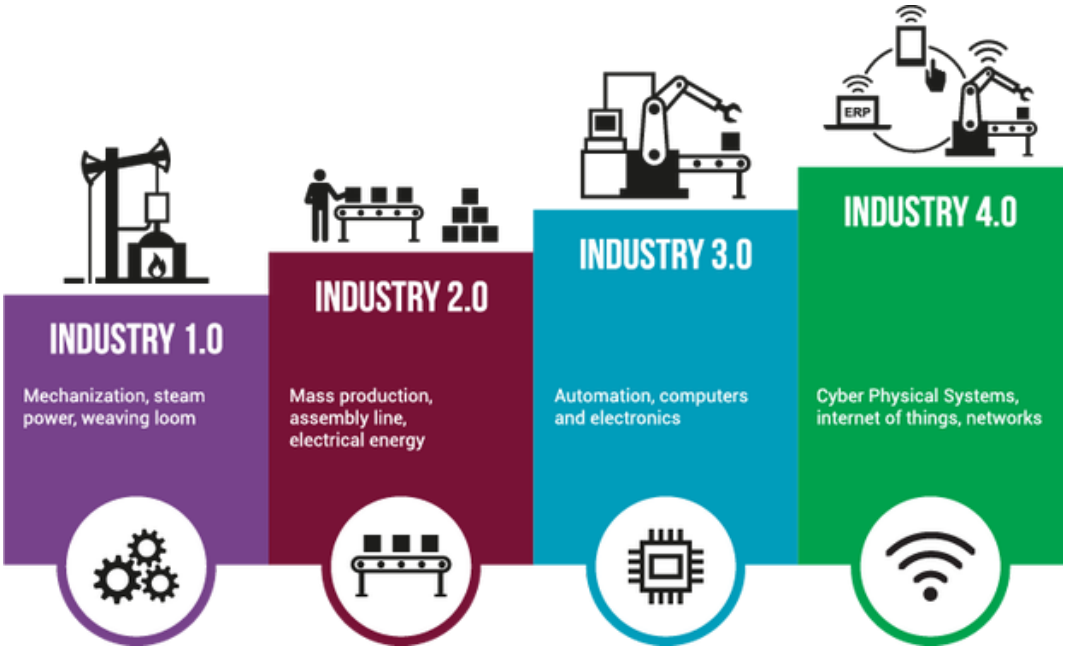


Figure 1.1: Phases of industrial revolutions [1]

and sensors where everything seems to be connected and digitalized and assisted by machine learning and artificial intelligence. This has encouraged to create the so-called Smart Factories, that enable the production of highly customizable products, shorter life-cycles, more efficiency, and incorporation of physical and virtual systems [17]. It is exactly this that makes the difference with the previous industrial revolution.

Now, that we have a brief explanation of Industry 4.0 and the Smart Factories, it is important to note that while Smart Factories are build of multiple connected machines, sensors and systems communicating together, the data that they produce is of great importance so that we can extract valuable information from them by performing different analytics, but before that the data needs to be prepared.

The main purpose of this Master's Thesis is to develop a prototype service that can ingest data from different sources in a Smart Factory and prepare and transform

[1] https://www.netobjex.com/wp-content/uploads/2018/12/1-1.png (Accessed on: 2019-09-23)

them for further processing. This type of prepared data is referred to as Smart Data [24]. In the context of my work, Smart Data is the state of data which are not anymore raw but are transformed and from which we can extract valuable information. This is essential because Smart Data is used together with different predictive and preventive algorithms in order to mainly reduce overall costs and make the whole manufacturing process smoother. An example of a predictive measure is the reduction of unplanned downtimes as a result of regular maintenance to prevent machine failure [11].

Another important part of my work is the Smart Factory Simulator, which generates different sensory information based on a given configuration. It is meant to simulate a real-world Smart Factory and it is necessary for testing the prototype service and for evaluating the results.

In the process of designing and developing the above-mentioned service I will as well try to answer the following research question:

- To what extent is it possible to automate the aggregation of raw incoming data streams?

The outline of the Thesis is as follows:

- In the Background chapter, the main underlying technologies that have been used in this work will be explained.
- Next, in the State of the Art chapter, several research papers related to the above research question will be described and analyzed.
- The next chapter describes the use cases that are of special interest and the derived requirements necessary to develop the system.
- In the Method section, everything related to the work that has been done will be explained. First starting with the general concepts and then diving into more details and describing the implementation as well.
- The Evaluation chapter shows how the work was tested and describes the results as well.
- And finally, in the Conclusions section, the whole work is wrapped up.

# 2 Related Work

## 2.1 Background

This section gives a short description of some of the most important technologies that were used to develop the prototype for collecting data and aggregating them into the so-called Smart Data. In the process of developing the prototype service, the following technologies have been considered important.

### 2.1.1 MQTT Protocol

MQTT stands for Message Queuing Telemetry Transport. It is a lightweight and open publish/subscribe messaging protocol and also an ISO Standard [1]. The publish/subscribe is a messaging pattern, where the publisher decouples itself from subscribers and it sends messages only to specific topics or classes. Subscribers can decide on which topics they want to listen to and receive messages that were published there. The main advantage of this messaging pattern is the loose coupling, where subscribers and publishers do not know anything specific about one another, they are completely decoupled by a message broker, the one that collects the messages from publishers and distributes them to the subscribers [2]. The publish/subscribe pattern is not to be confused with the messaging queue pattern. The main differences are:

- In the message queue pattern, a message is stored until it is consumed by a client, whereas in the publish/subscribe this is not the case.

[1] https://www.iso.org/standard/69466.html (Accessed on: 2019-10-03)
[2] https://en.wikipedia.org/wiki/Publish-subscribe_pattern (Accessed on: 2019-10-03)

- In the publish/subscribe pattern, a message can be consumed by multiple clients, but in the message queue pattern only by one.
- In the message queue pattern, queues have to be created explicitly before they can be used, which can be cumbersome. On the other hand, the publish/subscribe pattern creates topics on the fly.

The MQTT protocol is designed to work on top of the TCP/IP protocols. Because it is so lightweight and has a small footprint - it only requires two bytes of data for a minimal message and the maximum is 256 megabytes [3] - it is considered especially suitable for Internet of Things (IoT) and Machine to Machine communication (M2M) among other usages. It is important to note that, in MQTT the messages that are published, are arranged in hierarchical topics and this gives the possibility to subscribers to listen to the root topic, and thus receive messages from all the underlying topics, or subscribe only to specific ones and ignore the others. This offers us the opportunity to organize the published messages in different groups/topics. Another important characteristic of the MQTT protocol is the so-called Quality-of-Service (QoS), where each client that is connected to the broker can specify it. With the help of the Quality-of-Service, we can set the importance of the messages to be published and received. MQTT offers three types of QoS:

- 0 - At most once: this is also known as fire and forget, where each message is sent only once and there is no acknowledgment if the message has been received successfully.
- 1 - At least once: here the message can be sent multiple times until there is an acknowledgment that the message has been received. This QoS can also lead to duplicates.
- 2 - Exactly once: this is the highest level of QoS offered by MQTT, where the message is guaranteed to be delivered exactly one time.

[3] https://en.wikipedia.org/wiki/MQTT (Accessed on: 2019-10-03)
[4] https://upload.wikimedia.org/wikipedia/commons/8/82/MQTT_protocol_example_without_QoS.svg (Accessed on: 2019-10-03)

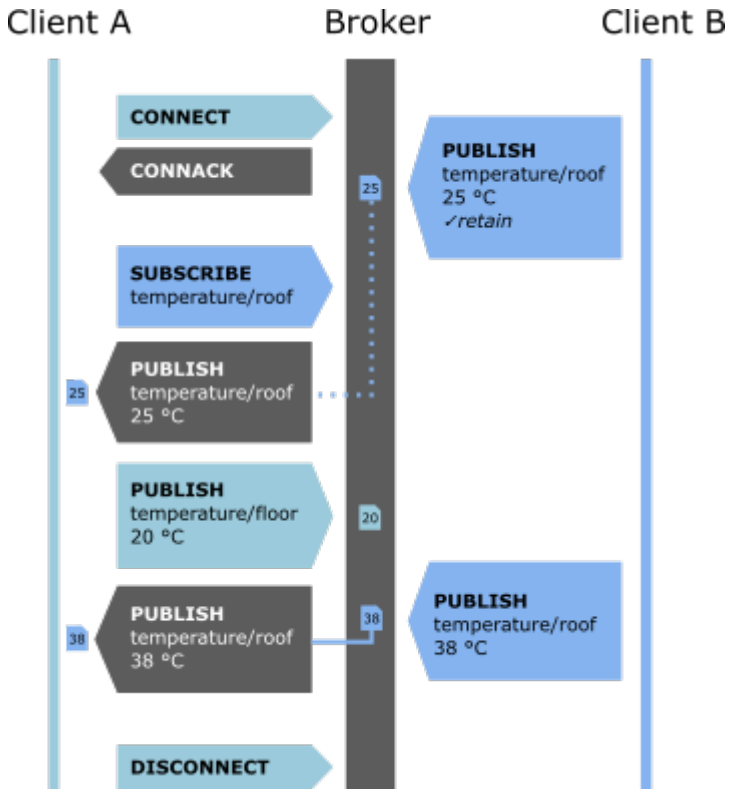


Figure 2.1: MQTT Protocol [4]

### 2.1.2 Apache Kafka

This technology has been developed by LinkedIn, which was open-sourced later - now called Apache Kafka. The main purpose of Apache Kafka is to be used as a reliable real-time data streaming platform. Kafka platform tries to unify the streaming of data by offering high throughput as well as low latency [5]. Apache Kafka also follows the publish/subscribe messaging pattern, but the difference with MQTT here is that it is highly scalable, and it stores the streams of data in a fault-tolerant way [6]. The most important APIs of this platform are:

- Consumer API - is used for ingesting data by subscribing to a topic in a Kafka broker
- Publisher API - is used to publish data to a topic
- Connector API - is a special API that allows Kafka to connect to other systems or applications, the same as publishers and consumers, and make these connectors reusable
- Streams API - is useful when we want to process streams of input data coming from one or more topics and output them to a topic or multiple topics

Data records in Kafka are also organized in topics or categories. When persisted, a data record contains the following structure:

- a key
- a value, and
- a timestamp

In Kafka, a data record is saved in a partitioned log, where each of the partitions is an ordered list of data records, as illustrated in Figure 2.2. When reading from Kafka topics, an offset is kept by each consumer, which can also be controlled by them 2.3. This offset is a small footprint, hence a large number of consumers is not a problem for a Kafka broker.

[5] https://en.wikipedia.org/wiki/Apache_Kafka (Accessed on: 2019-10-03)
[6] https://kafka.apache.org/intro.html (Accessed on: 2019-10-03)
[8] https://kafka.apache.org/23/images/log_anatomy.png (Accessed on: 2019-10-08)
[8] https://kafka.apache.org/23/images/log_consumer.png (Accessed on: 2019-10-08)

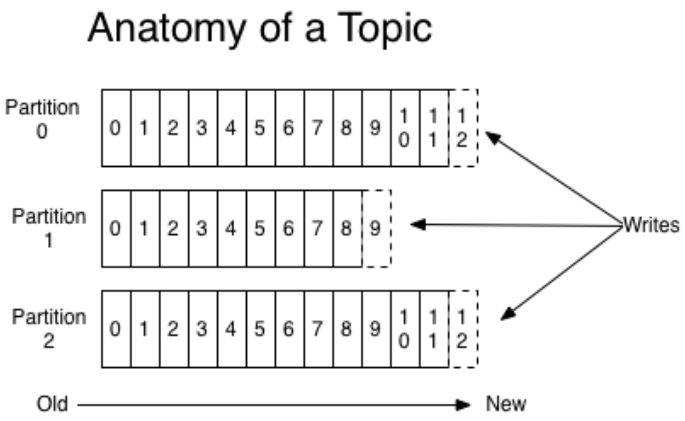


Figure 2.2: Kafka Topic [7]

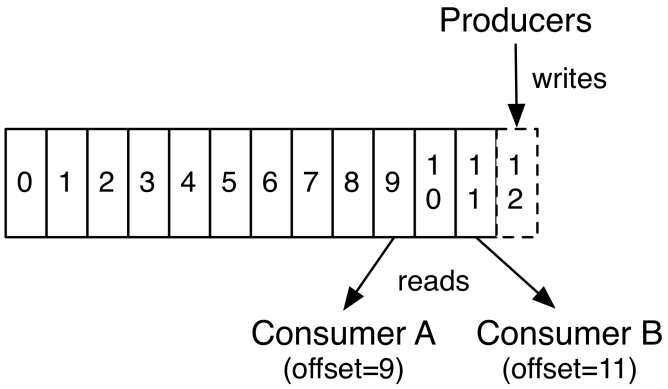


Figure 2.3: Kafka Consumer Offset [8]

Because Kafka is made to run in a cluster, partitions are distributed across multiple Kafka brokers which can run in different servers, thus enabling for high scalability. At the same time, each partition is replicated on other servers, keeping the data safe and making fault tolerance possible. It is important to note that each of the partitions has a master server or the so-called ”leader” which serves as the main server for reading/writing in this particular partition. If the ”leader” fails for some reason then its job is overtaken from another slave partition or the so-called ”follower”. Each of the servers plays the role of a ”leader” for one or more partitions and the role of a ”follower” for other partitions.

#### Apache ZooKeeper

It is also important to emphasize that Kafka platform makes use of Apache ZooKeeper in the background, which is a centralized service for facilitating highly reliable and safe distributed coordination [9].

#### Kafka Connect

Because in this project the data has to be distributed from MQTT broker to Kafka and from Kafka to InfluxDB, it is important to make use of Kafka Connectors so that we can move data from source to destination using reusable components. In Kafka Connect, there are two different types of connectors: a Source connector, and a Sink connector. The job of a Source connector is to ingest data from another system or application and publish them into different topics in Kafka, and the job of a Sink connector is to consume the data records from a Kafka topic and forward them to other applications or systems. In this context, the MQTT Source Connector has been used and a custom InfluxDB Sink Connector has been developed to fulfill the needs of this project.

### 2.1.3 InfluxDB

InfluxDB is a time-series database (TSDB) built by InfluxData. Time-series databases (TSDB) differ from traditional Relational Database Management Systems (RDBM)

[9] https://zookeeper.apache.org/

in that they are specialized and optimized in storing and retrieving time-series data that are organized in pairs of times and values. Time-series data are usually data that comes from constantly generating data sources like application metrics, different kinds of events, or IoT and sensor data. These types of databases can summarize data, compress data, and scan large sets of data faster than traditional RDBMS. Among many other TSDBs, InfluxDB is one of the most popular ones because of its performance [10].

### Structure

The data in InfluxDB are organized in:

- measurements,
- series, and
- points.

A measurement is similar to a table in RDBMS. In InfluxDB, a measurement is constructed from multiple series that are grouped together. Series are formed by a grouping of different pairs of tagsets, which are key-value pairs of tags(differs from a field in that it is indexed). Whereas points are created by grouping multiple pairs of fieldsets and a timestamp. The performance of InfluxDB is optimized mainly by indexing the so-called time ”tag”, by default, as well as other defined tags from the user. Therefore querying data based on tags is faster than fields, which are not indexed. The allowed data types in InfluxDB are integer, float, string and boolean. This is not always the case with other TSDB.

InfluxDB also supports a SQL-like language to write and read the data from databases which is rather familiar when compared with other RDBMS. Another important feature is the InfluxDB API which is a way to query data in InfluxDB [11]. Besides the InfluxDB API there is also a wide range of clients that support different languages in order to communicate with InfluxDB [12].

[10] https://db-engines.com/en/ranking/time+series+dbms (Accessed on: 2019-10-08)

[11] https://docs.influxdata.com/influxdb/v1.7/guides/querying_data/ (Accessed on: 2019-10-08)

[12] https://docs.influxdata.com/influxdb/v1.7/tools/api_client_libraries/ (Accessed on: 2019-10-08)

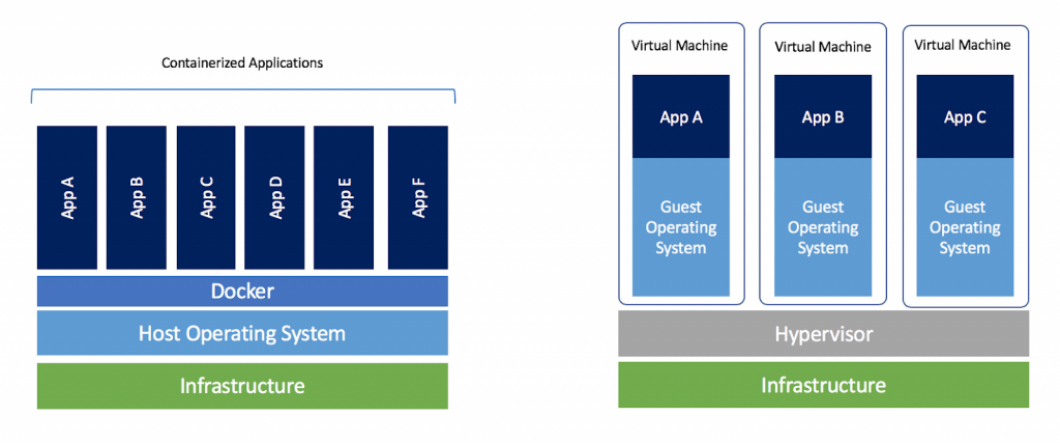


Figure 2.4: Containers vs Virtual Machines [13]

### 2.1.4 Docker

Docker is a platform that is used to organize and deploy applications in packages called containers. It is not the same as virtual machines because Docker runs on top of OS, thus using OS-level virtualization 2.4. A big advantage of Docker is that it is extremely lightweight and uses resources efficiently. In comparison to virtual machines, we can start multiple Docker containers in a single virtual machine while on the other hand to create multiple virtual machines it requires many more resources. With this, Docker is another important factor that makes it possible so that the systems can get highly scalable.

Docker is also flexible in organizing containers in the way that the user wants. By default, containers are isolated from other containers but different ports and communication channels can be enabled and containers can be placed in the same virtual network so that they can interact with one another. Another important part of the Docker platform is also Docker Compose. It is a tool that allows us to configure and run multiple docker containers at once, which is much more convenient than running each container individually. This tool has been used in this project to set up the whole infrastructure and start it with one command - ”docker-compose up”. Apart from this, there is also a tool called Docker Swarm. It

[13] https://www.docker.com/blog/containers-replacing-virtual-machines (Accessed on: 2019-10-08)

is a container orchestration tool, which is mainly used to run docker containers in a cluster - similar to Kubernetes. Nevertheless, Docker Swarm is out of the scope of this project and it has not been used.

### 2.1.5 JSF

JavaServer Faces or JSF is a standard that is part of the Java Enterprise Edition (JEE) which is used to develop component-based user interfaces for different web applications. This technology enables us to develop reusable user interface components and to connect them with data sources on the backend, as well as binding them with different events and triggering method calls on the backend which can be rather practical and time-efficient [14]. In this project, JavaServer Faces has been used together with Primefaces, which is an open-source framework that offers more than 100 different ready-to-use JSF components, to develop the user interface of the prototype, which has been developed for this thesis.

## 2.2 State of the Art

In the process of developing and writing this Master's Thesis, a diverse number of sources and search engines have been used in order to find scientific papers that were considered relevant to this topic. Even though there exist multiple possibilities, mainly the following sources were used:

- Google Scholar [15]
- IEEE Xplore Digital Library [16]
- Microsoft Academic [17]
- World Wide Science [18]
- ScienceDirect [19]

[14] https://javaee.github.io/javaserverfaces-spec (Accessed on: 2019-10-08)
[15] https://scholar.google.com (Accessed on: 2019-10-14)
[16] https://ieeexplore.ieee.org (Accessed on: 2019-10-14)
[17] https://academic.microsoft.com (Accessed on: 2019-10-14)
[18] https://worldwidescience.org (Accessed on: 2019-10-14)
[19] https://sciencedirect.com (Accessed on: 2019-10-14)

## 2 Related Work

In order to come up with relevant search results, the following search keywords were used:

- Smart Data
- Big Data
- Smart Production
- Smart Factories
- Industry 4.0
- Data Aggregation in IoT
- Dimensionality reduction
- Predictive maintenance
- Automated decision making
- Decision making support

Starting with the paper [17], authors define a Smart Factory as a context-aware Factory, which is achieved by data records collected from different sources, that should support people and machines in doing their jobs. The Big Data that is generated and collected from a Smart Factory, as stated in the paper [14], is associated with the following known characteristics:

- Variety
- Volume
- Velocity
- Variability
- Complexity
- Value

Of special interest here, is the Value property. There is value in Big Data, but the value has to be extracted by users running certain queries and acquiring important and filtered information, otherwise, as stated in [9], Big Data in a vacuum can be useless. When there is value or knowledge present in data, which can be gathered from Big Data, then this knowledge is known as Smart Data [16]. In [10], there is a distinction made between Big Data and Smart Data. The job of the Big Data is said to be data processing, whereas Smart Data is involved in the value of the data and the ability to enable decision-making. Smart Data is the way how the data is collected, aggregated, reduced, summarized and brought together in order to get valuable information for decision-making. Similar tasks are also described in the process of transforming Big Data into valuable insights for smart energy

management using Big Data, proposed as a solution by the authors of the paper [27]. As stated above, to support people and machines in doing their jobs, predictive and preventive analytics algorithms can be applied to the Smart Data to reduce costs and potentially avoid machine failures, or on the other hand, the quality of the produced items can be increased by analyzing defects that become visible from Smart Data [11]. All of these needs these days come as a requirement for producing highly customizable products and as a demand for globalization.

In [11], authors describe a guide on how to convert a factory to a Smart Factory. They emphasize four main areas of the IoT architecture in a Smart Factory:

- Manufacturing Applications
- Enterprise Applications
- IoT Platform
- Data Visualization and Control

The IoT Platform area, according to the authors, is considered as the critical part that converts a factory to a Smart Factory. This area, as seen in Figure 2.5, is responsible for enabling connectivity for machines sending sensor data, ingesting them, transforming and persisting them to an RDBMS, or closely related technologies like NoSQL or Hadoop Distributed File System (HDFS). Importantly, Hadoop offers a solution for storing data in a distributed way, in multiple nodes, and enabling parallel processing, which is useful for huge amounts of data [3]. It is stated that data can come in various forms - structured, unstructured, semistructured, events, time-series, logs and that they are converted to the systems standard data format, however, there is no detailed description on how these data are mixed and matched and how they are aggregated and turned into Smart Data. In [13] too, data ingestion is seen from authors as an important requirement for building a Big Data analysis pipeline in a manufacturing environment, where data transformation, noise reduction, and filtering among other tasks take place. Nonetheless, the summarization or aggregation of data is not further described.

The authors of the paper [4] start by stating the importance of the Big Data and the rate of growth, where according to the source, every day around 20 quintillion bytes of data are produced, which can be in any format, structured, semistructured or unstructured data. All of this has become possible nowadays because of the reduction of costs of sensor-enabled devices. When these sensor-enabled devices communicate and are connected with one another and exchange information, then

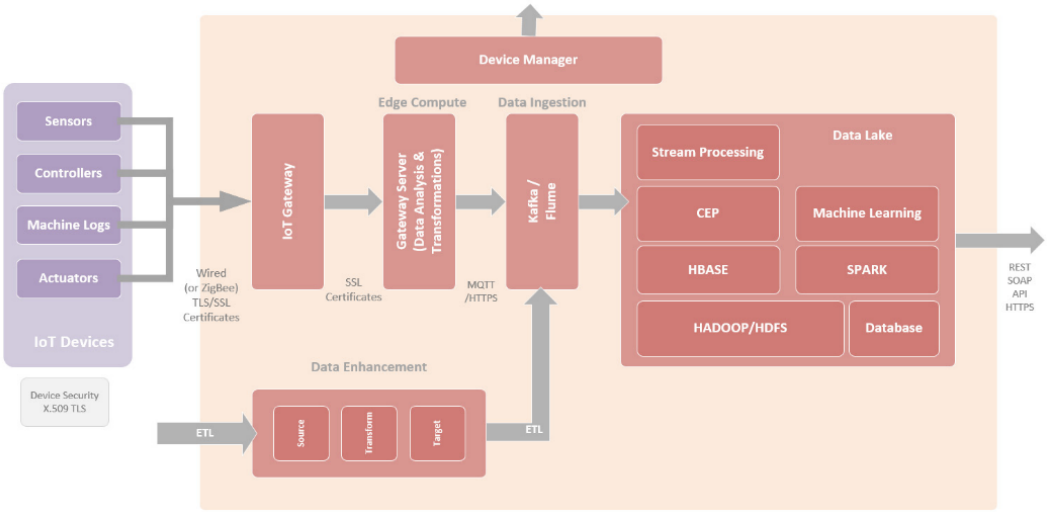


Figure 2.5: IoT and Big Data architecture. [11]

this is known as the Internet of Things or IoT. Sensors can be of different types and can provide all kinds of information, be it temperature, some kind of pressure or anything else. Because of the heterogeneity of the data that is collected, the overall integration and transforming them into the so-called Smart Data becomes a real challenge. Based on this paper, the collected data can be sent as raw or already as aggregated, or there is a possibility so that the aggregation occurs later. It is important to note, that aggregation plays an important role in producing Smart Data. However, according to the paper, specific domain knowledge is required in order to know how the data can be aggregated and extracted to come up with Smart Data that contain value and that make sense in this context. Likewise, in ”Data Mining - Practical Machine Learning Tools And Techniques”, the authors emphasize the importance of a suitable degree of aggregation. They highlight an example where raw telephone communication data is not useful for analyzing the behavior of a customer unless they are aggregated by the user and by a specific period of time, be it by week, month or year. That’s why selecting the appropriate aggregation of data is crucial and requires domain knowledge.

Next, in the paper [19], authors give a summary of Big Data analysis methods in a smart manufacturing system. In their findings, among other important areas, they highlight the importance of saving the data that has been collected from different

sources. The enormous amount of data generated each millisecond from numerous sensors in machines can be aggregated in order to reduce the amount of data that is sent. At the same time, this aggregation is seen as an intermediate format, before feeding them for further processing in data mining, artificial intelligence or machine learning. In other words, this intermediate format is known as Smart Data. It is not specified that the aggregation functions are automatically chosen but it is said that averages, sums, minimums, maximums, or shapes of distribution can be chosen. According to the authors, the importance here is that the aggregated data do not lose worthy information, from which valuable insights can be extracted.

In the paper [15], authors have proposed a platform for Big Data analytics in smart manufacturing environments. They applied the system in a die casting factory in South Korea. In the process of casting, 75 different data parameters have been collected from various sensors and controllers for analytics. All these data information are saved in the so-called legacy system. Data from different sources are merged here, and they are mapped using some identification number. In case there are defects, the quality review outcomes are entered manually by workers. It is important to note, that the prototype service that has been developed for this thesis uses a similar approach for item identification. Following, the data cleaning is performed so that no invalid values are allowed, among them null values are removed as well. Afterward, the authors allow for the selection of the representative value of the collected parameter data such as mean, minimum, maximum in order to come up with an answer as to what the reason for the defect was. Finally, these aggregated data have been analyzed using a correlation matrix and in conclusion, a number of different process parameters like high speed, maximum speed, and low-speed stroke, were heavily associated with defects caused in the production process.

In the paper ”Analyzing Big Data: Social Choice and Measurement” [20], the authors highlight the importance of the summarization of the so-called Big Data by reducing their multidimensionality into something that delivers more value and more insights, a reduced value that tells more and that is easier to comprehend. According to the authors, data is considered as big, not only based on the number of variables and the number of data, but also on the complexity of the data that is being delivered. Social networks, data from brain scans, and genomes can be considered as examples of this complexity in data or examples of Big Data. Such data always needs to be reduced, and this reduction can be achieved through aggregating higher-dimensional data into lower-dimensional models. It is important to note that, with aggregation and data reduction there is a need for a decision to be made

on what to consider as more important and what to leave out of the aggregated values. To argue with all this, authors take as an example a network analysis where they analyze the Florentine Marriage Network, which describes the martial ties between 15 families in Florence in the 15th century. They take into consideration three different centrality aggregation functions, which are mainly supposed to tell the degree of connection and the importance of the nodes (people) in relation to the whole network. The authors conclude that, based on which centrality aggregation function is chosen, then some aspects of the network can be lost. More precisely, it is emphasized that there is no perfect aggregation function that can automatically know and retrieve the most valuable information from the data. They refer here to the knowledge that they received from choosing a centrality aggregation function for the network analysis example.

In a different domain, ”Big Data Reduction for a Smart City’s Critical Infrastructural Health Monitoring” [25], authors propose a platform called BigReduce as a solution that would be used in the so-called smart cities for infrastructure health monitoring by sending different events in case of damage of constructions like bridges or buildings. Here authors use two different schemes to reduce the weight of Big Data processing in IoT, the Big Data reduction scheme and the Big Data decision-making scheme. Of special interest is the Big Data reduction scheme, where authors define it as an important step that helps in deciding which data should be sent over the network for further analytics. In contrast to the solution developed in the prototype service, the data reduction scheme in the above-mentioned paper is achieved by reducing the amount of data that is collected and that will be sent, at the sensor level, basically by using event-based decision, if there is no health-event triggered, then less data is collected.

Complementary to these findings, in the paper ”A Fast Large-size Production Data Transformation Scheme for Supporting Smart Manufacturing in Semiconductor Industry” [23], authors propose a data transformation solution for a Big Data platform in manufacturing environments. The main idea is to insert the data from a local database and convert it into a workpiece-centric pattern, that is supposed to be beneficial for further processing and analytics. The work that is presented, is split into the injection module, partition module, and aggregation module. In the injection module, the data from the local database is injected and persisted into a distributed columnar database in the Big Data platform, thus ensuring scalability. Partition module would partition the persisted data into smaller parts that would contain only records of a particular item. And the aggregation module is meant to bring

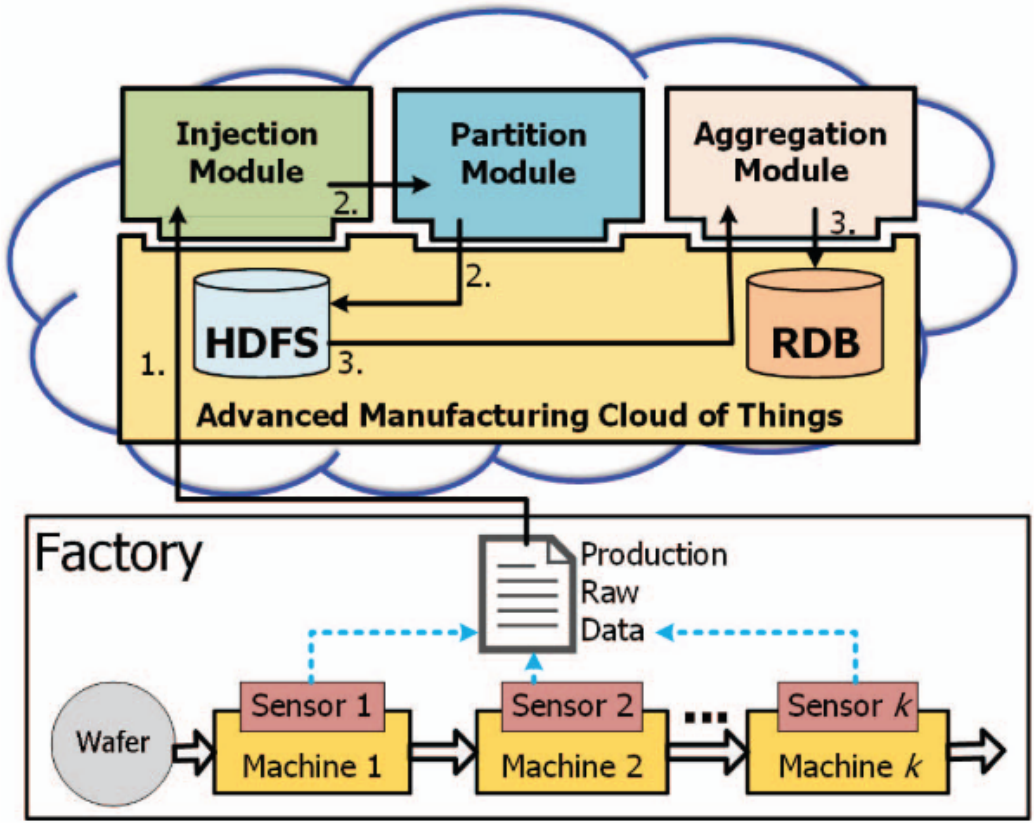


Figure 2.6: The architecture of the Big Data processing-based workpiece-centric transformation. [23]

together all the production data of a workpiece into a single and unified record, all of this using parallel distributed processing to achieve high speed and performance. This scheme is illustrated in figure 2.6. Authors indicate the importance of parallel distributed processing in Big Data platforms, because of the huge amounts of data that systems need to deal with. Parallel distributed processing of data aggregation is out of the scope of this thesis and has not been taken into consideration while developing the prototype service.

In the paper "Big Data and Machine Learning for the Smart Factory—Solutions for Condition Monitoring, Diagnosis and Optimization" [18], authors again as in previous papers, point to the importance of data reduction. Here, they propose different clustering algorithms for anomaly detection and condition monitoring,

however, these clustering algorithms can come to their limits when dealing with huge amounts of data, which is in fact one of the characteristics of Smart Factories in contrast to traditional factories, as highlighted among other points in [26]. Sensors in a Smart Factory usually produce lots of data, so in order to cope with this issue, authors state that the input data has to be preprocessed and reduced in order for the clustering algorithms to deal with them correctly. They state different possibilities for data reduction, like the Multidimensional Scaling method or using the Principal Component Analysis.

The authors of the paper "Highly Scalable and Flexible Model for Effective Aggregation of Context-based Data in Generic IIoT Scenarios" [2] give a proposition of a way to collect and aggregate heterogeneous data coming from multiple sources in the industrial internet of things or IIoT. The approach used in this work is mainly derived from Complex Event Processing methods, where only relevant data is stored and aggregated in order to get meaningful insights, all of it coming from numerous data sources. It is stated once again, that the plenty of data that is generated in the industrial internet of things, makes it unfeasible to analyze each parameter value separately without summarizing data into Smart Data. Authors use a network-centric method, where they consider only actions that flow over the network as important, thus collecting and aggregating only data related to them. They also highlight that the data that is generated from a single host, which can be a machine, can have an impact on cause and effect but they were not taken into consideration as such without looking at the action flow over the network.

In "A Journey from Big Data to Smart Data" [5], like in many other sources, authors start by stating the importance of Smart Data concerning Big Data and the value that has to be generated from them. Nevertheless, in contrast to other papers, authors introduce here a "closed-loop" approach between Big Data and smart data. The idea is that there should be a connection between the two. When Big Data is generated and before they are aggregated into Smart Data, they are compared to the already existing Smart Data - so generally making a real-time comparison of an existing state with an expected state. This should serve the real-time monitoring and decision making, especially in the marketing or sales, which are also part of the use cases of the mentioned paper.

Just as important, the paper "IoT Integration for Adaptive Manufacturing" [1] gives a general architecture of an IIoT system in a manufacturing environment for aggregating data and reacting accordingly to the received results in an automatic

way. Among other significant areas, data collection plays a major role. The authors describe their main idea of data aggregation using two steps. In the first step, a specific software middleware named iIoT-Collect is used. Its main job is to discover devices in a production environment and give the possibility to the user to register new devices and configure them. The job of the second step is to forward the sensor data to a Message Queue (MQ) server, which can then distribute the data further to the systems that make use of them, either storing in a NoSQL database or parsing and converting to information so that the system can react accordingly.

Additionally, in "An architecture for aggregating information from distributed" [28], authors propose a system for collecting data from distributed data nodes. Here, authors assume that the produced data from heterogeneous nodes are first persisted in those nodes and only then their platform comes into play, so that the data is aggregated from a diverse number of sources, which is also the main focus of their work. Besides aggregation and heterogeneity of data, they also consider scalability issues. Nevertheless, according to their platform, each distributed data node should publish the collected data regarding a traced item in a unique URI and persist it in a query resolution server. In this way, when aggregating data for a particular item, a query resolution server knows how to find all the items based on the previously provided URIs and put them together. This way, the data coming from different sources for a particular item are aggregated. It is also important to notice that, there were no data reduction or summarization methods mentioned in the paper.

Finally, taking into consideration all of the readings, to the extent of my knowledge I come to conclude that there isn't a fully automated way of aggregating data from multiple sources and reducing them in order to turn them into Smart Data. Domain knowledge seems to be necessary in order to decide how each parameter should be aggregated and which aggregation function makes more sense in various contexts. There also isn't a fully automated way of aggregating data from heterogeneous sources, without having some prior reference IDs somewhere, or some links between data. Hence, the work presented in this thesis does not represent a fully-automated data aggregation, but it is more directed towards an approach, where a domain expert person is the one that should configure meaningful aggregation functions for Smart Data generation. The intention is to provide ample support to find the right aggregation mechanism.  graphicx

# 3 Use Cases & Requirements

The following is a general description of the use cases and requirements for the development of the prototype service. During the development phase of the service, an imaginary T-Shirt factory has been used as an example for better understanding and explanation, and the same will be used here in order to derive the use cases and requirements.

This imaginary T-Shirt factory may have a different number of machines, all of which are supposed to generate a diverse number of data when a T-Shirt is being produced. As it is illustrated in Figure 3.1, the factory includes the following machines and a production line in the order as shown below:

1. Pattern making machine
2. Cutting machines
3. Edge seaming machine
4. Sewing machines, and
5. Pressing machine.

So, when a T-Shirt is produced, it should go through these five different machines until it is completed. After each finished good, the performance of each machine has to be evaluated, and this is done by the so-called quality gate. In the best-case scenario, all of the produced T-Shirts are shipped to the customers and they are of the best quality, but this is not always the case, thus sometimes it is possible so that some goods are returned because of their low quality. In this situation, it is important to find out what could have caused this drop in quality. The answer can be achieved using retrospective analysis, but only if the data that the selected algorithm is applied to, is aggregated and prepared correctly so that they deliver the right information. This is also the first use case. The second use case has to do with the maintenance of machines in a factory. There is a possibility that some machines might get broken after some time, and this could impair the whole production process. Hence, maintenance is an inevitable task in factories. Nevertheless, the

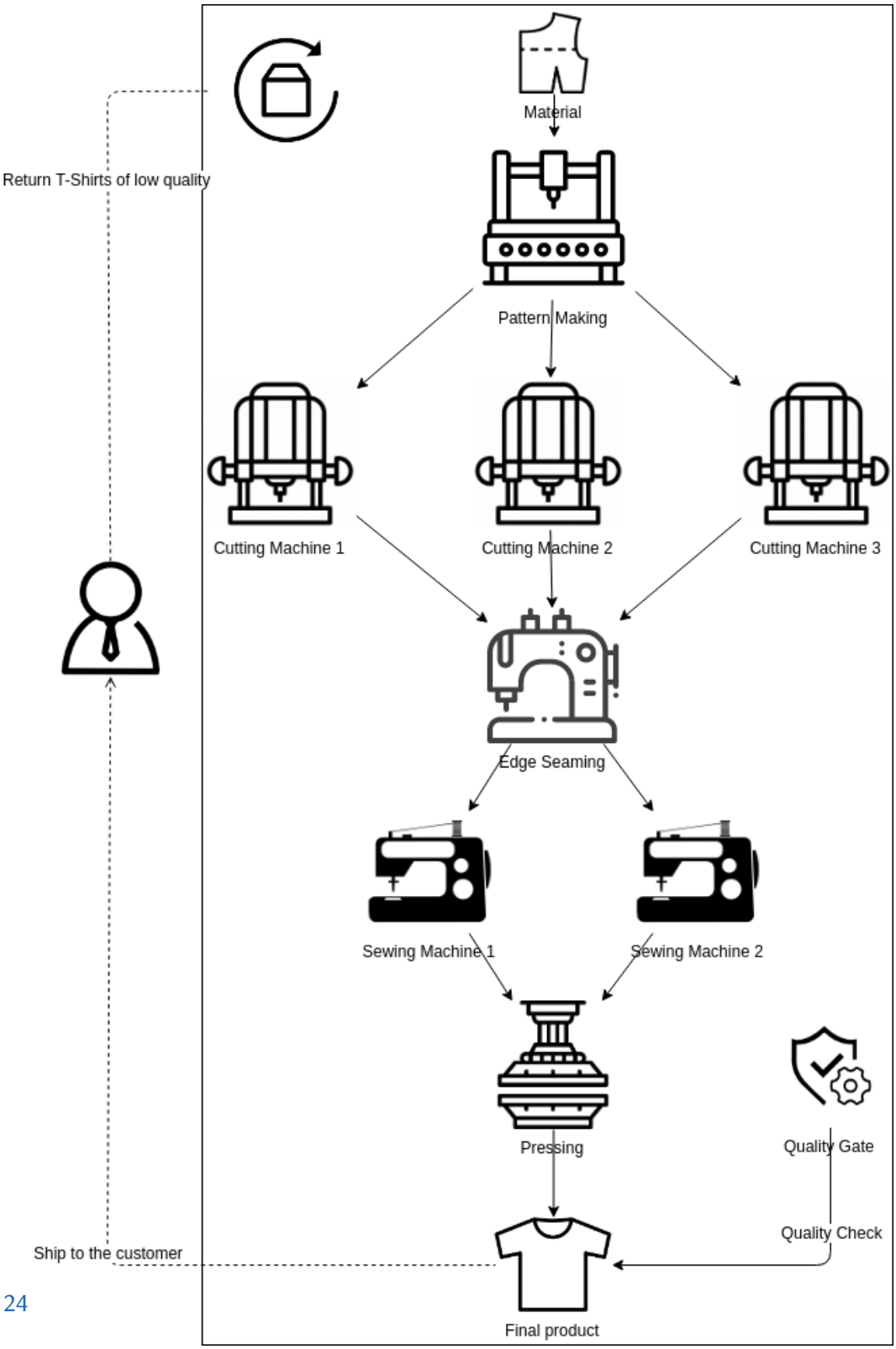


Figure 3.1: The production line of the imaginary T-Shirt Factory

right time to perform machine maintenance is hard to guess - it can be too early, which drives the costs high, or it can be too late, which disturbs the production process and again increases the overall costs. Based on this, the second use case is about making the smart data available so that a prediction when a machine might get broken can be achieved. In this way, the costs can be reduced by performing machine maintenance just at the right time.

Referring to those use cases, the following requirements for developing the system to support data collection and aggregation have been considered necessary:

- Data generation - there should be a way to generate machine data that has to be collected
- Data transportation - in order to collect the generated data, there should be a defined way of transmitting those data to the prototype service
- Data storage - when the generated sensory information arrives, it is obvious that it has to be persisted somewhere, and
- Data aggregation - the last and the most important requirement is to offer a possibility, where the data generated from machines can be aggregated in such a way so that it delivers valuable information for use in further analytics. The requirement here is that the responsible user should have the possibility to decide how each parameter received from the machines should be aggregated. This implies that the user that performs this task, should have domain knowledge and understand which aggregation mechanism makes more sense in a particular context.

# 4 Method

## 4.1 Concepts

What follows is a description of the developed system as a whole, how each component is supposed to communicate and interact with each other. The more detailed description of the most important components that have been developed can be found in the implementation section. There have been many different technologies used in order to achieve the final result, however, the diagram 4.1 shows the architecture of the system and its most important components.

The two main components in this diagram are the Smart Factory Simulator and the Smart Data Service, which in fact are the two endpoints of this architecture. The data is meant to flow from the Smart Factory Simulator to the Smart Data Service. The Smart Factory Simulator can contain a variable number of machines and event sources that generate data while producing items. These machines are configured to send those generated data via a transport protocol in a lightweight JSON format. Using this kind of approach, data coming from heterogeneous sources is simulated.

It is relevant to mention that, the transport protocol does not persist the received messages in its topics, but instead, it immediately forwards the same information to the registered subscribers. Therefore, another message queueing platform is necessary for further ingesting the data and persisting in its topics. Even though the purpose of this work is not scalability, the message queueing platform can be run in clusters. It is also important to emphasize that, the transport protocol decouples the data generation from data collection and can easily be replaced by some other protocol without much intervention, which makes the system flexible.

Following the data flow in this pipeline, which is represented using dashed lines and arrows to show the direction at which the data is flowing, we come to the connectors.

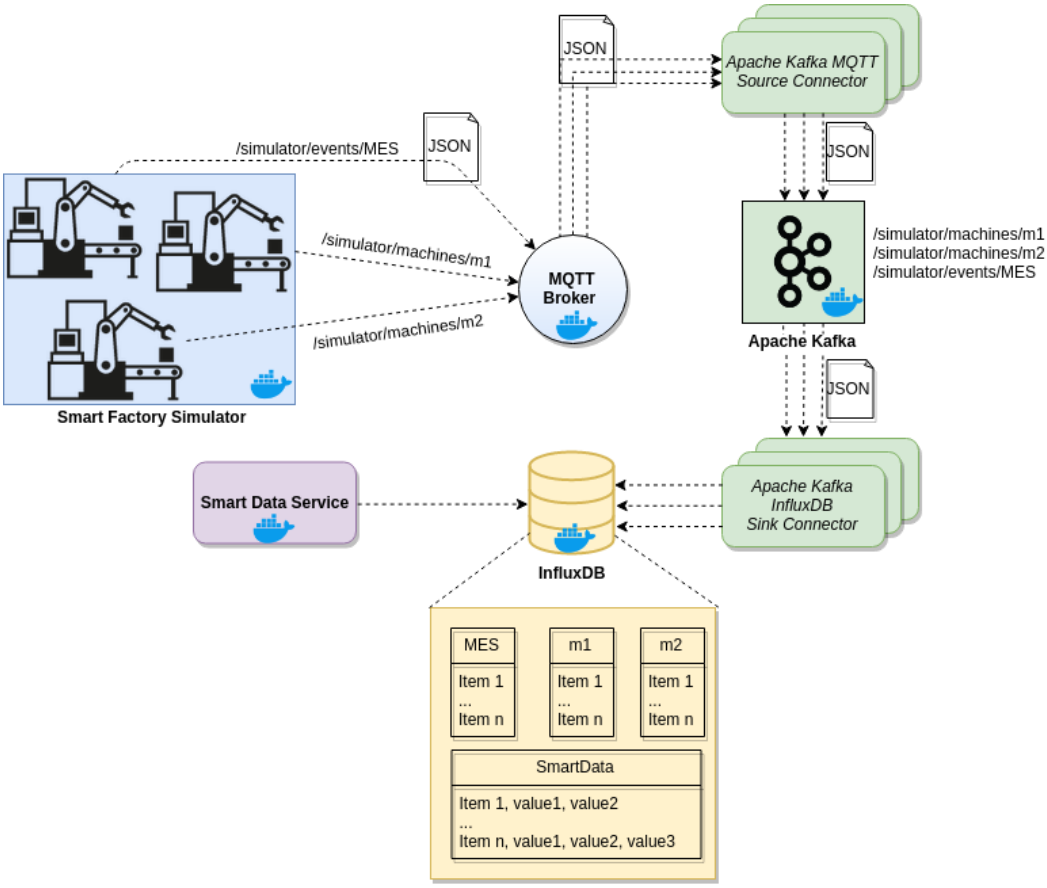


Figure 4.1: System Architecture

These first connectors are in fact subscribers to the topics of the transport protocol and publishers to the topics in the message queueing platform. After the data is published in the message queueing platform, the outgoing connectors play the role of subscribers of those topics, collect the data and persist them in an external database by creating a table per each topic, as it is illustrated in the referenced diagram.

In the end, using the Smart Data Service user interface, the user has access to the raw data collected in the InfluxDB, and by utilizing the aggregation mechanisms of the Smart Data Service new measurements containing Smart Data can be generated.

Besides the described components, it is also important to highlight that each of those components run in their own docker container, as depicted in the diagram with a small docker icon. Docker plays an important role in building this infrastructure because using docker-compose the whole system is configured in a single configuration file and can be started using only one command - ”docker-compose up”.

This summarizes the brief description of the architecture and presents a big picture of the system that has been developed.

## 4.2 Implementation

### 4.2.1 Smart Factory Simulator

In this section, I will explain the details of the implementation of the Smart Factory Simulator. Simulators try to model real-life systems and they enable to perform different tests for different scenarios and apply diverse implementations to observe the outcomes, without the need to test those implementations on real-life systems, which normally takes longer and can be more troublesome based on circumstances.

As already mentioned, the simulator is an important part of my work. It is used to simulate a Smart Factory and generate sensory and other types of information based on the user-defined given configuration. This implies that the Smart Factory simulator is a dynamic one because it can be exactly specified how it should


Figure 4.2: Structure of JSON (JavaScript Object Notation)

behave. Of course, there is always room for improvement, but on this occasion, it is sufficient.

Smart Factory Simulator is split into the configuration part and the implementation part.

### Configuration

The configuration of the simulator is defined inside a JSON (JavaScript Object Notation) file. JSON is a data-interchange open standard that is lightweight and easy to read and write. It is supported by almost all different programming languages because it is easy to parse and only makes use of two structures: attribute-value pairs and an ordered array list [1]. It is already very popular and it is being used extensively and in many cases is serving as a replacement of XML.

With JSON it is possible to define all different kinds of configuration combinations in our simulator. The first part of the configuration file, although the order is not necessarily important - it can only be more readable because of logical order, represents connection properties to an MQTT (Message Queuing Telemetry Transport)

[1] https://json.org (Accessed on: 2019-09-24)

```
"mqttServerURI": "tcp://mosquitto:1883",
"mqttUsername": "admin",
"mqttPassword": "admin",
"rootTopic": "smartfactory",
```

Figure 4.3: MQTT connection properties

broker where the simulator will connect and send the generated data. The following connection properties are defined under the root node:

- The connection URL to the MQTT broker
- Username for MQTT authentication, if present
- Password for MQTT authentication, if present
- The name of the root topic where the simulator will be sending data

The figure 4.3 shows an example of the connection properties.

The next part of the configuration file are the machines and the machine groups. Here it is possible to describe the diverse types of machines that the user wants to have in the simulating Smart Factory. The following JSON properties are available inside a machine node:

- The id of the machine
- The data frequency (milliseconds)
- The total duration of the frequency (milliseconds)
- The amount of data to be produced per frequency
- The type of data to be generated

It is important to note that the id of the machine and the ids of produced data must be unique and cannot contain any spaces. This has implications both in the parsing/processing of the data by the simulator as well as further in the developed infrastructure (MQTT topics, Kafka Connectors, Kafka Topics, and InfluxDB measurements). The data that can be generated are of the following types:

- A fixed predefined curve with a lot of data points to be followed, which have to be defined in another JSON file

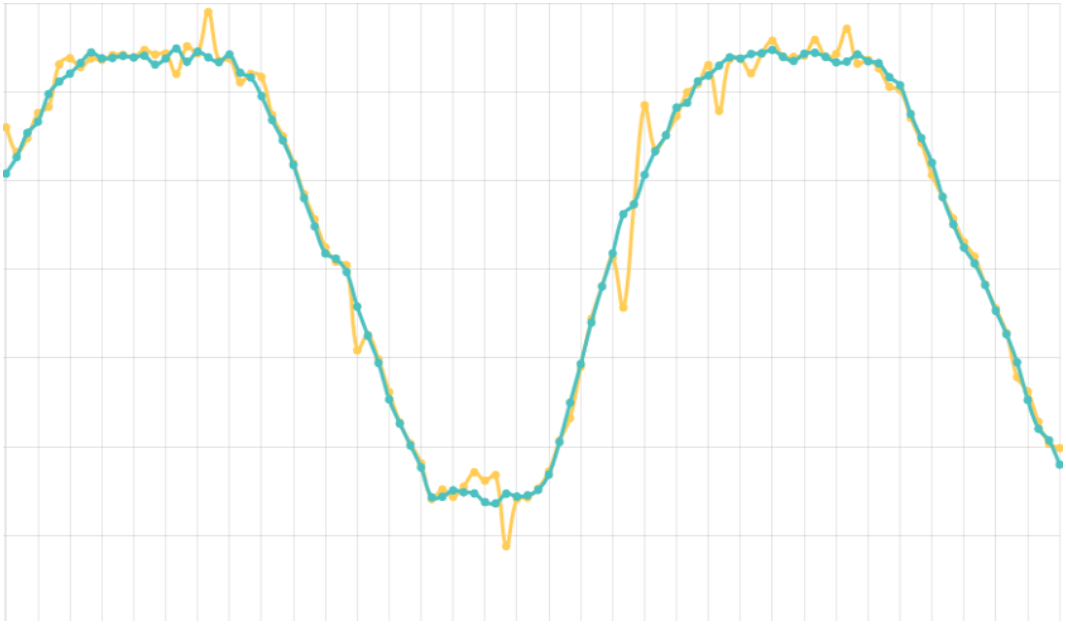

Figure 4.4: An example of random added noise to machine generated data

- A random integer or float that is distributed uniformly on the specified range
- A random value that is generated using a normal or Gaussian distribution, and
- A random value that is generated using a Cauchy distribution

We need different kinds of randomness to approximate the data coming from real-life machines. A more advanced feature of the simulator is to add additional noise to the generated data so that sometimes the produced values can be unpredictable. Figure 4.4 illustrates generated data with and without added noise. Added noise, in this case, is generated using the Cauchy distribution, which adds a wonderful random fluctuation to the graph. This is especially important if we want machines sometimes to be faulty, which is the case in the real-world scenario, where not everything is perfect and doesn't go the way we want. With this in mind, we can also define the quality constraint for each data type and decide what values affect the quality of the produced item. Do the generated values have to be within a range, under a certain value or above a certain value to produce items of good quality. An example of the complete configuration of a machine can be seen in 4.5.

```
"Pressing": {
    "id": "Pressing",
    "data": [
        {
            "id": "p-s1",
            "type": "FIXED_CURVE",
            "value": "curve1.json"
        },
        {
            "id": "p-s1-normal-dst",
            "type": "FIXED_CURVE",
            "value": "curve1.json",
            "noise": {
                "type": "NORMAL_DST",
                "value": "300, 20"
            },
            "qualityConstraint": {
                "type": "RANGE",
                "value1": 260,
                "value2": 380
            }
        },
        {
            "id": "p-s1-cauchy-dst",
            "type": "FIXED_CURVE",
            "value": "curve1.json",
            "noise": {
                "type": "CAUCHY_DST",
                "value": "350, 10"
            },
            "qualityConstraint": {
                "type": "RANGE",
                "value1": 260,
                "value2": 380
            }
        }
    ],
    "dataFrequency": 2000,
    "amountOfDataPerFrequency": 20,
    "totalDurationPerCycle": 6000
}
```

Figure 4.5: Example of the configuration of a machine

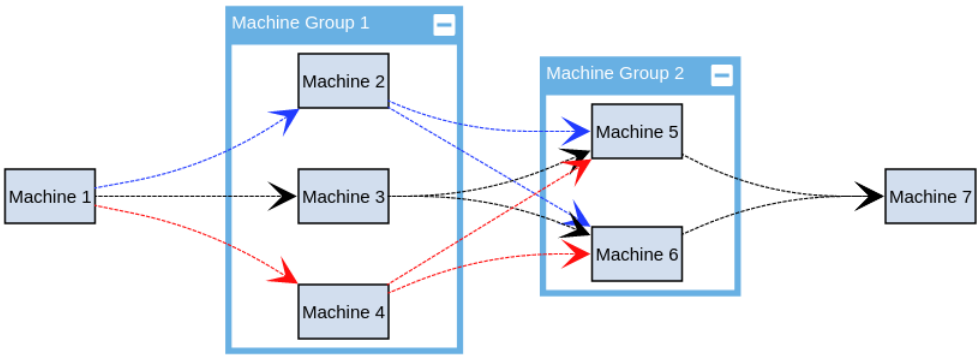


Figure 4.6: An example of machine groups in a production line

There is another extended functionality that the simulator offers. The user has the possibility to define multiple machines and group them. This should normally be done for machines that are similar in nature and that are used to perform the same task in the production line, but the selection of the machine is randomized for each produced item. This happens when there is more demand on a particular task in the production process and the load has to be split on multiple similar machines. So, in reality, different items can go through different lines of production. It can easily happen that one of the machines inside a group doesn't perform as well as the others, thus some of the produced items at the end must be of lower quality. This increases more the overall randomness of the simulator. This kind of production process is illustrated in 4.6.

After specifying the configuration of machines and machine groups, it is possible to define the quality gate as well. Its main purpose is to calculate the quality, how each of the machines involved in a production process has performed. When combining the above-mentioned quality constraints of a machine with added noise, especially using Gaussian distribution or Cauchy distribution, then the generated data-values will vary. The quality is then determined based on these values, it drops proportionally based on how many generated data-values exceeded or were outside of the allowed quality constraint range and how many of them were within the allowed range.

In real-life scenarios, besides a diverse set of machines, we can have other sources

that generate and send data to the MQTT broker, which can play an important role later in analytics. An example of this can be the data coming from an MES (Manufacturing Execution System) which tell more about the item that is being manufactured, or some kind of events stating that a machine is broken or that maintenance has been performed on a machine. Smart Factory Simulator makes it possible to integrate the following types of event sources:

- An event stating when an item has entered a machine in a production process
- An event stating when an item has exited a machine in a production process
- An event stating that maintenance has taken place for a specific machine, and
- An event stating that a machine is faulty

The last two possibilities to specify events can be combined with additional noise (the same way as before in machines) to add randomness when a machine will be broken or when maintenance will occur, e.g. after a specific number of produced items plus a random function or after a defined duration plus a random function. An example of the configuration of events is shown in Figure 4.7.

Finally, the user can set the production line of the Smart Factory, from start to end, thus having the possibility to define the exact order of machines and machine groups if present, where an item will go through in the production process.

### Implementation

For the implementation of the Smart Factory Simulator, the Java programming language together with some useful libraries has been used.

Firstly, the configuration part, where everything is written in JSON format, needs to be parsed in some way so that the values are properly collected. For this, a library called JSON-B or JSON Binding v1.0.1 [2] has been used, so that the configuration can be directly mapped to Java classes.

[2] http://json-b.net (Accessed on: 2019-09-25)

```
"eventSources": [
    {
        "eventSourceId": "eventSource1",
        "events": [
            {
                "machineId": "PatternMaking",
                "eventsToGenerate": [
                    {
                        "id": "Maintenance Event 1",
                        "type": "EVENT_MAINTENANCE",
                        "conditionType": "AFTER_ITEMS",
                        "conditionValue": 2
                    }
                ]
            }
        ]
    },
    {
        "eventSourceId": "eventSource2",
        "events": [
            {
                "machineId": "EdgeSeaming",
                "eventsToGenerate": [
                    {
                        "id": "Machine 1 Broken Event",
                        "type": "EVENT_BROKEN",
                        "conditionType": "AFTER_ITEMS",
                        "conditionValue": 2,
                        "noise": {
                            "type": "CAUCHY_DST",
                            "value": "0, 10"
                        }
                    }
                ]
            }
        ]
    }
],
```

Figure 4.7: Example of events configuration

Secondly, for the generation of random noise using Gaussian distribution or Cauchy distribution, which is ubiquitous in the simulator, a lightweight Java library has been used, namely Apache Commons Math v3.6.1 [3].

Thirdly, because we need to send these generated data to an MQTT broker, then an MQTT client is necessary, and for this, the Eclipse Paho MQTT Client v1.2.1 [4] has been used. This client is helpful and enables easy configuration on how the client should connect to the broker and how should the messages be sent.

Finally, for building and managing the application and its dependencies, the Apache Maven v3.6.0 [5] has been used, and to put everything together and run the simulator in a containerized environment, Docker v.19.03.1 [6] was the preferred option.

Figure 4.8 gives a big-picture of the UML Class Diagram for the most important classes so that the reader can have a better understanding of the whole concept of how the simulator works. Three main colors have been used to group classes of similar functionality - blue, green and gray. SmartFactorySimulator is the main class where the application starts running and therefore has a blue color to distinguish it from other classes. Classes that are colored with green are classes that play a fundamental role in the main functionality of the simulator. Last but not least, classes colored with gray are helper classes that support other classes on meeting their goals.

The SmartFactorySimulator class is illustrated with all of its methods in more detail in figure 4.9. The basic idea of this class is to read the configuration file specified in the arguments when starting the simulator, which in this case, is specified inside the Dockerfile. After reading the configuration file and mapping it to a Config object 4.10, which is done using the JSON-B library, it starts creating machines, machine groups, quality gate, fixed predefined data-points, event sources, etc, depending on the properties mapped in the Config object. Then, the whole simulation starts when the method ”startSimulation” is invoked. In this method, for each item that is produced, it iterates over the production line that has been configured and invokes the appropriate simulation methods of the iterating objects. In case, there is a machine group in the production line, the SmartFactorySimulator class makes

[3] http://commons.apache.org/proper/commons-math (Accessed on: 2019-09-25)

[4] http://eclipse.org/paho (Accessed on: 2019-09-25)

[5] urlhttps://maven.apache.org (Accessed on: 2019-09-25)

[6] https://www.docker.com (Accessed on: 2019-09-25)

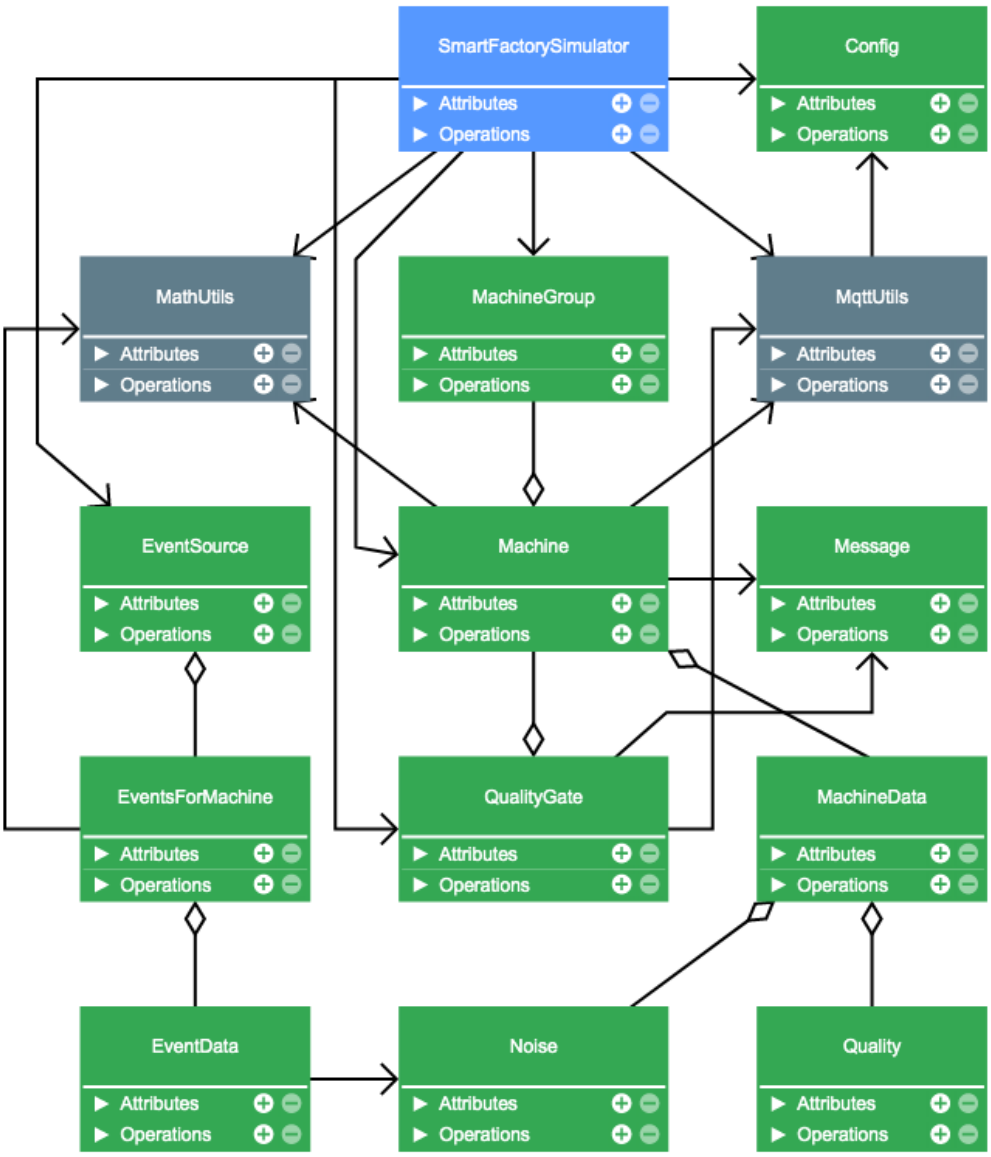


Figure 4.8: UML Class Diagram - Big picture

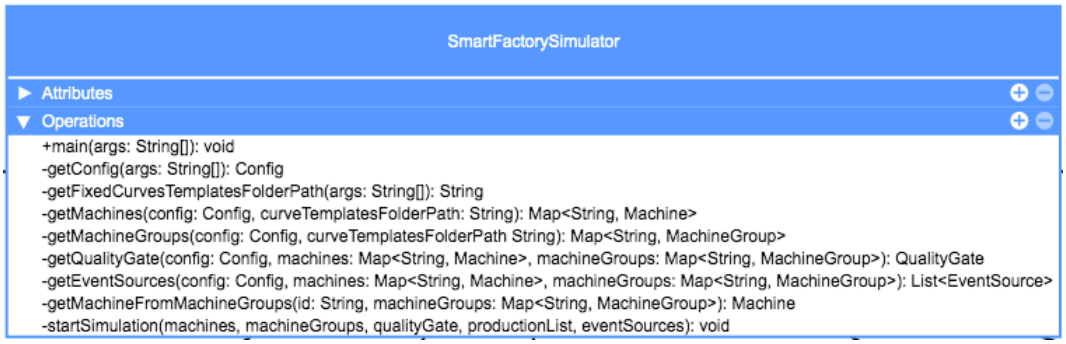


Figure 4.9: SmartFactorySimulator Class

```
private static Config getConfig(String[] args) throws IOException {
    Jsonb jsonb = JsonbBuilder.create();

    String configPath = Constants.CONFIG_DEFAULT_PATH;
    if (args.length > 0) {
        configPath = args[0];
    }

    String jsonString = new String(Files.readAllBytes(Paths.get(configPath)));
    return jsonb.fromJson(jsonString, Config.class);
}
```

Figure 4.10: Mapping the JSON configuration file to Config class

use of the helper class MathUtils to get a randomly generated number in order to make a random decision, which of the machines in the group should be selected. Furthermore, the MqttUtils helper class, which makes use of the above-mentioned Eclipse Paho MQTT Client, is used to set the connection options for the MQTT broker. For each time that a new IMqttClient is created, it uses the same connection options.

All the machine objects are at this point already created and contain the necessary configurations to simulate the machines and generate sensory data-values. The properties 'dataFrequency', 'amountOfDataPerFrequency', and 'totalDurationPerCycle' control how often should data be sent, how many data-points should be generated per one message that is sent, and for how long should a cycle last respectively - meaning how long does it take for the machine to finish one item. Within this period, the method 'generateData' is called many times in order to

```
switch (e.getType()) {
    case RND_INT:
        value = MathUtils.generateUniformRandomInt(e.getValue());
        break;
    case RND_FLOAT:
        value = MathUtils.generateUniformRandomFloat(e.getValue());
        break;
    case NORMAL_DST:
        value = MathUtils.generateNormalDistribution(e.getValue());
        break;
    case CAUCHY_DST:
        value = MathUtils.generateCauchyDistribution(e.getValue());
        break;
    case FIXED_CURVE:
        value = fixedCurvesMap.get(e.getId()).getNextDataPoint();
        if (e.getNoise() != null) {
            value = (Double) value + getNoise(e.getNoise());
        }
        break;
    default:
        value = e.getValue();
        break;
}
```

Figure 4.11: Code snippet from generateData method in Machine class

get the desired data just like it was configured as is shown in the following code snippet 4.11.

If there is already a configured JSON file that contains fixed predefined data-points, then those points will be followed one-by-one from the 'generateData' method and it will add additional noise to it if it is configured so. This gives the user a lot of options and literally it is possible to define different custom paths of data-values, known also as templates, that some sensor in the machine should follow.

After the data is generated it is wrapped in a Message class with the current timestamp and with the help of the IMqttClient it is sent to the MQTT broker. The same is true for the MachineGroup class and the QualityGate class, although the QualityGate instead of generating data, calculates the quality of each performing Machine against the defined quality constraints. The more violations of the quality

constraints there are, the less the quality of the machine will be. The more detailed illustration can be seen in the following UML diagram 4.12.

It is important to note that, machines themselves, normally cannot track the item that is being produced. To simulate real factories, events should send data when an item entered a specific machine and when it exited, when some machine is broken, and when maintenance has occurred. There are of course many other events that can be sent but this should be sufficient for this project. The generation of the last two events should usually be configured with some additional noise, because in real-life scenarios they aren't really predictable, especially the event stating that a machine is broken. Here too, the MathUtils helper class is used to generate the randomness. The figure 4.13 shows more details of the classes that are used to generate and send events and the relationships that they have with other classes.

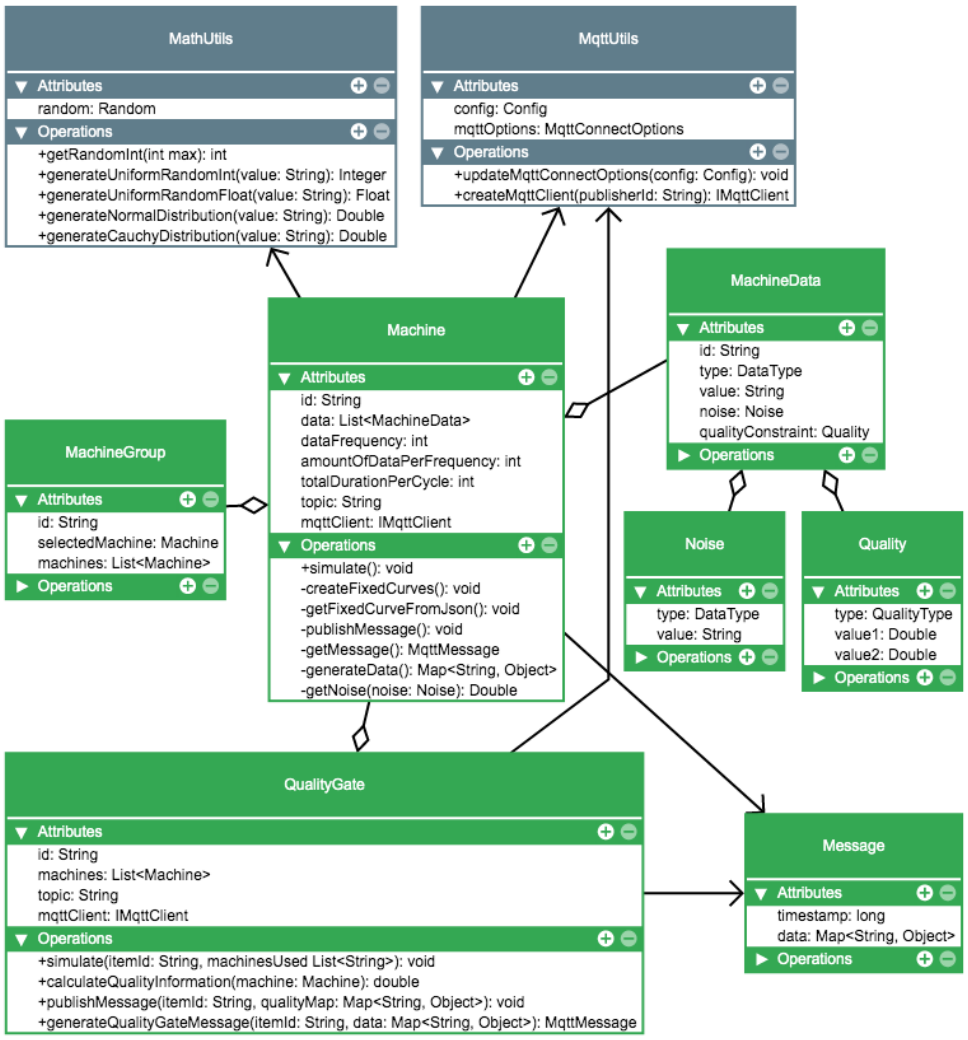


Figure 4.12: Detailed UML Class Diagram of machines and machine groups

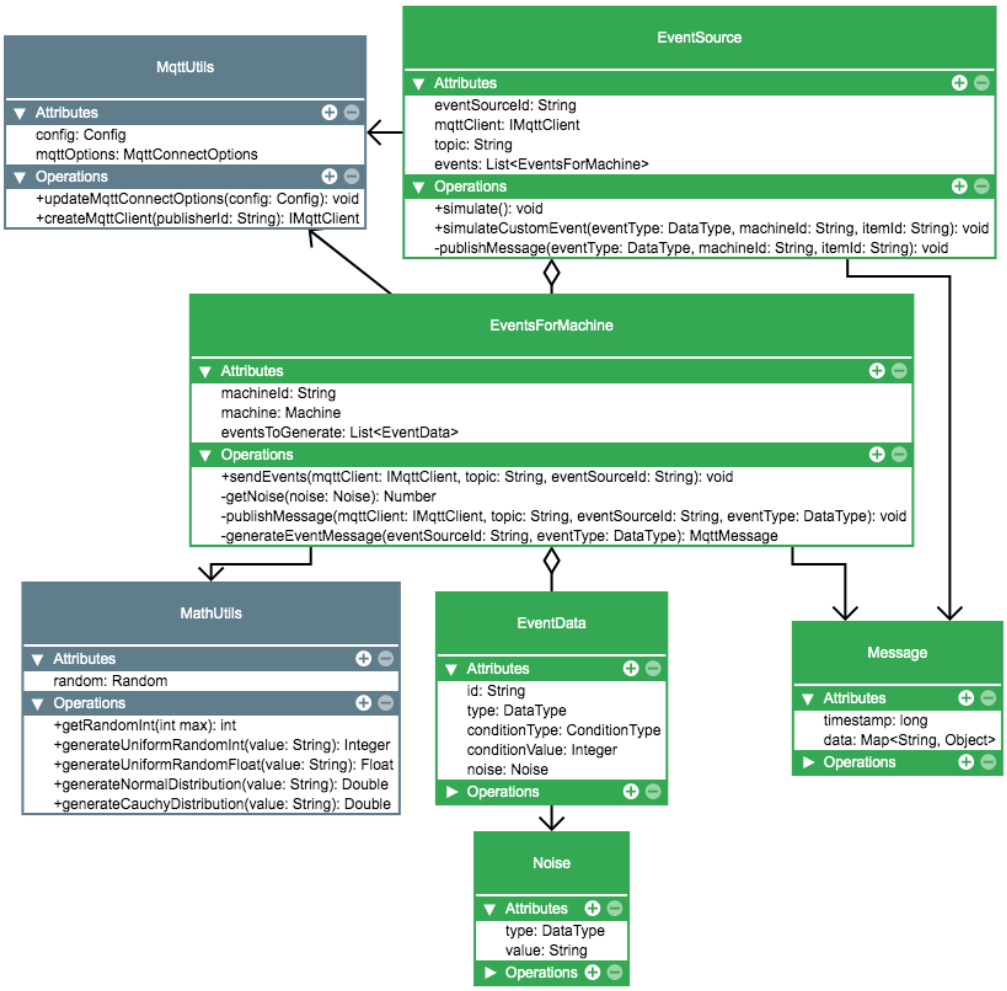


Figure 4.13: Detailed UML Class Diagram of events

### 4.2.2 Smart Data Service

This section describes how the Smart Data Service works in detail and the most important pieces of its implementation. Smart Data Service is a web-based application that is developed mainly using Java EE 8 technology [7]. The reason for this is that the UI is considered more suitable for the user to interact with the service and select aggregation functions in a more convenient way. Another reason is that using a web-based application the need for extra installations is avoided. In order to run this Java EE 8 web-based application, an application server is required, and the Payara Server [8] has been used.

Starting with the index page 4.14 of the Smart Data Service, it is obvious that the user-interface is split into three parts, the top area which contains the title, the left section which contains the main navigation menu, and the central area which contains the content of the currently opened page. All other pages make use of the same format, hence a template was created using the JSF Facelets [9]. When developing using JSF Facelets templates, the template is always derived by a child page and only the parts that need to be changed are updated. This gives us a lot of flexibility. In the content area of the index page, the user has the possibility to choose between two options for data aggregation based on the target analysis, namely:

- Retrospective Analysis, and
- Predictive Maintenance

Those available options are directly linked with the already described use cases in Chapter 3.

[7] https://javaee.github.io/ (Accessed on: 2019-10-20)
[8] https://payara.fish (Accessed on: 2019-10-20)
[9] https://javaee.github.io/tutorial/jsf-facelets001.html (Accessed on: 2019-10-20)

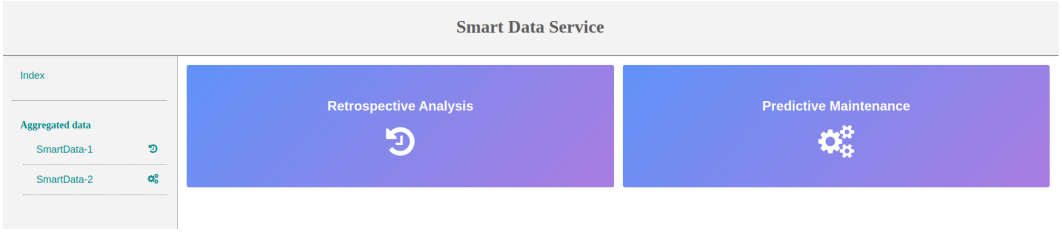


Figure 4.14: Smart Data Service - Index Page

### Retrospective Analysis

This page, which can be seen in Image 4.15, enables the user to aggregate the data in such a way that they will be valuable for finding out what was the reason that caused defects in produced items within a Smart Factory. All of this data that is seen here is referred to as the raw data that has already been collected and stored in the InfluxDB. The machines that sent raw data are displayed in different tabs, as it can be seen the Figure 4.15. In this example, there are nine different machines: Cutting-M1, Cutting-M2, Cutting-M3, EdgeSeaming, PatternMaking, Pressing, Sewing-m1, Sewing-m2, and Sewing-m3. Inside the content of each tab, there is a table that enables the user to select the parameters (Column) that have been sent from this particular machine, an aggregate function that will be used for the aggregation of this particular parameter, as well as a new column name that will be used when creating the newly aggregated data or Smart Data. The currently available aggregate functions are defined within a Java Enum, named AggregateFunction. The following are the available functions:

- Minimum
- Maximum
- Mean
- Median
- Mode
- Count
- Sum
- First
- Last
- Spread
- Standard deviation

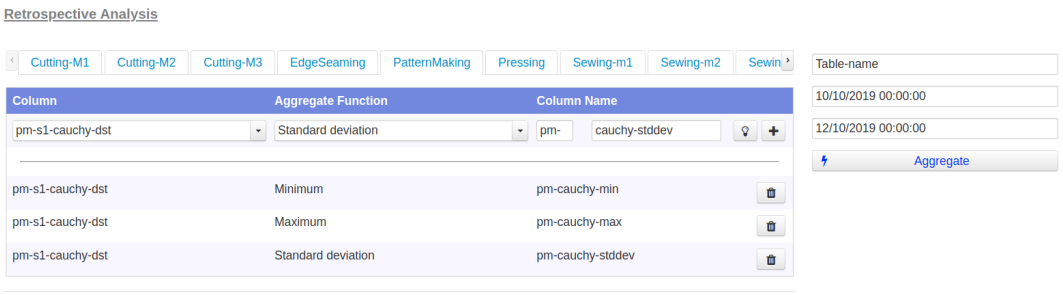


Figure 4.15: Smart Data Service - Retrospective Analysis Page

The combination of those functions gives the user multiple alternatives to generate meaningful Smart Data. After defining all aggregate functions for the machines that are of particular interest for this task, then the user is asked to specify the new table name where the Smart Data will be stored and the period in which the raw collected data has to be aggregated. Most importantly, after clicking the Aggregate button, the Smart Data Service has to collect data from different machines, which are equivalent to different tables in the InfluxDB as it can be seen in the System Architecture Diagram in 4.1, and put them together in a single row that contains multiple columns with values of already specified aggregate functions. Each of these newly created rows corresponds to a single produced item. Because in the raw collected data we have multiple tables for different machines, and because each machine table sends only raw sensory information, there should be a way to distinguish a collection of data for one produced item from the collection of data of another produced item. To solve this issue, it is assumed that the necessary information is sent via another data source called MES (Manufacturing Execution System), thus creating a table named MES. The data stored in the MES table is of the following format:

- Timestamp
- ID of the machine
- ID of the item being produced
- Start production/End production

With all those information available from the MES table, the Smart Data Service automatically knows the production line that an item traversed and the time it entered a specific machine and the time it exited it. This kind of information is

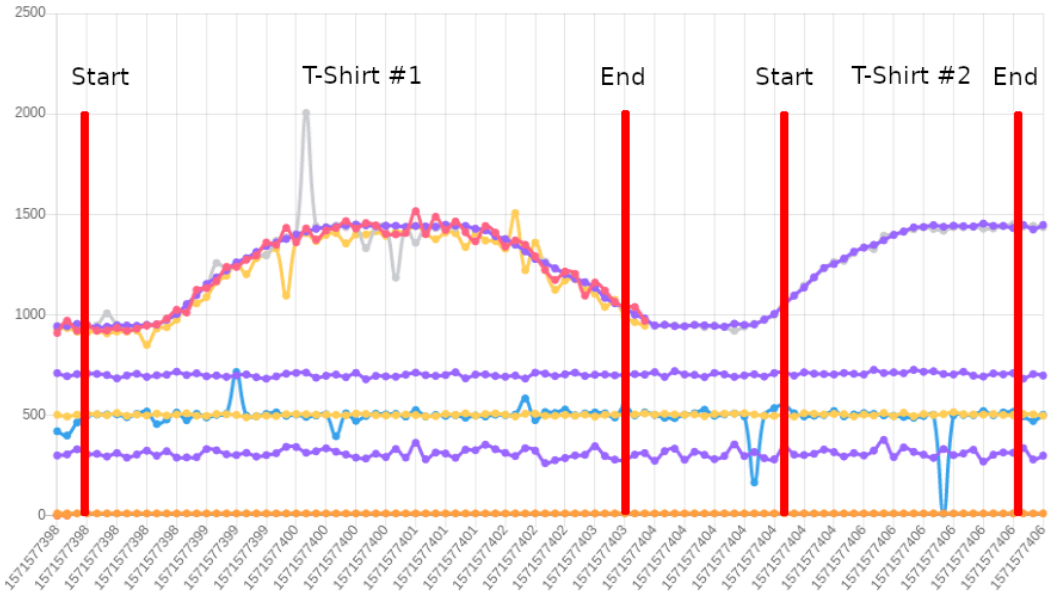


Figure 4.16: Raw Data generated by a Machine

crucial because raw data sent from machines contain only sensor values and no information about the current item that is being produced. That's why it was necessary to assume that this data comes from the MES data source.

With the available information from the MES table, as illustrated in Figure 4.16, the Smart Data Service can now select the right data collection for a specific item in each machine table. After it gets the collection of raw data from each machine table, then it applies the aggregate functions that were already defined by the user. Those aggregate functions reduce these collections of data to a single value, therefore making it possible to have a single row for a single produced item containing valuable information for further analytics. In addition, this row contains also quality information for each machine that produced the item and a boolean value stating if the item was returned from the customer or not.

After the Smart Data table is generated, then it appears in the menu as a link on the left part of the template.

When selecting one of the links from the menu on the left side of the template, then a view that shows Smart Data is displayed, more precisely a table containing

Smart Data

Smart Data - SmartData-1

Columns

(1 of 38) 1 2 3 4 5 6 7 8 9 10 15

| time | EdgeSeaming-quality | PatternMaking-quality | Pressing-quality | itemId | p-min | returnedItem |
|---|---|---|---|---|---|---|
| 2019-10-10T21:17:19.147Z | 1.0 | 1.0 | 0.8833333333333333 | 53790fe8-5ed9-48a6-af06-a3b6143307b2 | 10.0 | false |
| 2019-10-10T21:17:22.754Z | 1.0 | 1.0 | 0.9166666666666667 | f610700c-38a5-4b69-9ca4-d4763f1d1996 | 10.0 | true |
| 2019-10-10T21:17:26.197Z | 1.0 | 1.0 | 0.925 | 64742caf-f473-4f8a-963b-fbda683563b0 | 10.0 | false |
| 2019-10-10T21:17:29.572Z | 1.0 | 1.0 | 0.875 | 031bde11-f14e-40a1-8845-1e4b74c6c94f | 10.0 | false |
| 2019-10-10T21:17:33.092Z | 1.0 | 1.0 | 0.875 | adfb0a9a-d071-444d-a33c-57cf5a1f945a | 10.0 | false |
| 2019-10-10T21:17:36.561Z | 1.0 | 1.0 | 0.9166666666666667 | 1e1e7ed6-d55d-4317-9d7b-c9418b27d064 | 10.0 | false |
| 2019-10-10T21:17:39.929Z | 1.0 | 1.0 | 0.925 | 2b0a5064-48cc-4774-b474-8d74bf57758b | 10.0 | false |
| 2019-10-10T21:17:43.205Z | 1.0 | 1.0 | 0.9 | c63cdaec-9820-4427-8655-6aaa17dd0943 | 10.0 | false |
| 2019-10-10T21:17:46.524Z | 1.0 | 1.0 | 0.8333333333333333 | aad1d1d6-b4a0-4659-8549-d946d207b197 | 10.0 | true |
| 2019-10-10T21:17:49.896Z | 1.0 | 1.0 | 0.875 | c68f0772-f105-4ac6-a1bc-f8095f567192 | 10.0 | false |
| 2019-10-10T21:17:53.325Z | 1.0 | 1.0 | 0.9416666666666667 | 2412d52c-8871-4cb2-9f74-7bcca40e6922 | 10.0 | true |
| 2019-10-10T21:17:56.703Z | 1.0 | 1.0 | 0.9333333333333333 | 3c45feb7-3045-4ec4-a817-17e53e996605 | 10.0 | false |
| 2019-10-10T21:18:00.045Z | 1.0 | 1.0 | 0.9083333333333333 | e5551b96-93e3-44d2-9934-e295716cf6c6 | 10.0 | false |
| 2019-10-10T21:18:03.523Z | 1.0 | 1.0 | 0.8999999999999999 | a0ab05e5-a428-49c8-bfee-61d5faeb67fc | 10.0 | false |
| 2019-10-10T21:18:07.032Z | 1.0 | 1.0 | 0.9666666666666667 | cb7329fa-29ea-4552-9ae5-fa6882517650 | 10.0 | false |

(1 of 38) 1 2 3 4 5 6 7 8 9 10 15

Figure 4.17: Retrospective Analysis - Smart Data

this valuable information. An example of this table can be seen in Figure 4.17. This page view enables the user to see the generated results immediately and it offers as well three different options to extract the Smart Data:

- CSV (Comma-Separated Values) format
- Excel format, and
- PDF format.

## Predictive Maintenance

In the Predictive Analytics page, the user can generate similar Smart Data which will be helpful in predicting future machine failures or appropriate times for performing maintenance on a machine. The page view, as it can be seen in Figure 4.18, has a similar construction like the Restrospective Analysis page.

The first thing that the user should do in this page is to select the machine for which the Smart Data should be generated. The dropdown selection component can be seen on the right side of the page content. After that, it is possible to add additional aggregation functions for the data sent by this particular machine, though it is not necessarily required to do so. After specifying the new table name for Smart Data that will be generated and the timespan in which to collect data, then the data can be aggregated by clicking on the Aggregate button.

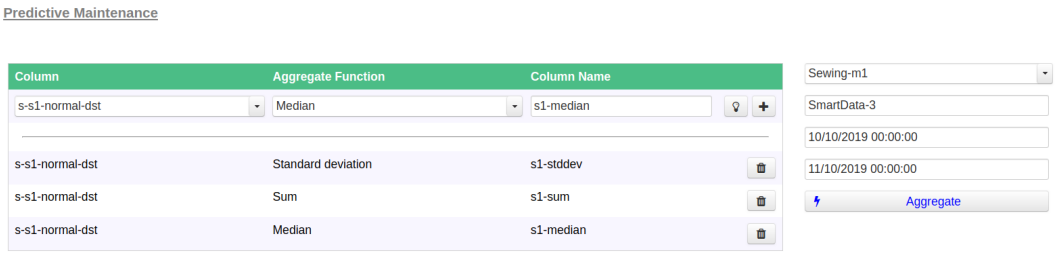


Figure 4.18: Smart Data Service - Predictive Maintenance Page

Most importantly, among data that is aggregated using the defined aggregate functions by the user, there are also generated data about different events that state if the machine has been broken or not, how many items have been produced since the last event, how much time has passed since last maintenance, and how much time has passed since last broken event. To generate these kinds of information, raw data from other data sources have to be collected and calculated. In the T-Shirt factory example, there are two additional data sources that send event data giving information if a machine has been broken or if maintenance has been performed in a machine. Therefore, by combining these two event sources, one can come up with valuable information.

In contrast to Retrospective Analysis, here the aggregate functions are not applied always to the collection of raw data of a single item, but to the collection of raw data of the number of items that were produced since the last event that was sent. Figure 4.19 shows the table view of the generated Smart Data from the Predictive Maintenance function.

The UML diagram 4.20 shows the most important classes of the developed prototype service. The Smart Data Service makes use of the so-called Layered Pattern, and it can be clearly seen from the UML diagram that it has two different layers - the Presentation Layer and the Business Layer. The classes in the Presentation Layer correspond to the page views that the user can see and interact with, whereas on the other hand in Business Layer the main class is SmartDataService, which contains the main logic behind the whole developed service. Other classes in the UML diagram that have gray color are more of helper classes or data transfer classes. It is important to note, that the class InfluxDBConnection produces a connection

Smart Data

Smart Data - SmartData-2

Columns

(1 of 3) 1 2 3 15

| time | broken | itemsProducedSinceLastEvent | timeSinceLastEvent-EVENT_BROKEN | timeSinceLastEvent-EVENT_MAINTENANCE |
|---|---|---|---|---|
| 2019-10-10T21:17:31.035Z | true | 4.0 | 0.0 | 0.0 |
| 2019-10-10T21:18:28.213Z | false | 17.0 | 57.178001403808594 | 0.0 |
| 2019-10-10T21:19:39.096Z | false | 21.0 | 128.06100463867188 | 70.88300323486328 |
| 2019-10-10T21:20:49.604Z | false | 21.0 | 198.56900024414062 | 70.50800323486328 |
| 2019-10-10T21:22:00.363Z | false | 21.0 | 269.3280029296875 | 70.75900268554688 |
| 2019-10-10T21:23:11.487Z | false | 21.0 | 340.4519958496094 | 71.1240005493164 |
| 2019-10-10T21:24:22.516Z | false | 21.0 | 411.4809875488281 | 71.02899932861328 |
| 2019-10-10T21:25:32.287Z | false | 21.0 | 481.25201416015625 | 69.77100372314453 |
| 2019-10-10T21:26:42.369Z | false | 21.0 | 551.333984375 | 70.08200073242188 |
| 2019-10-10T21:27:52.322Z | false | 21.0 | 621.2869873046875 | 69.9530029296875 |
| 2019-10-10T21:29:02.979Z | false | 21.0 | 691.9439697265625 | 70.65699768066406 |
| 2019-10-10T21:30:13.196Z | false | 21.0 | 762.1610107421875 | 70.21700286865234 |
| 2019-10-10T21:31:23.528Z | false | 21.0 | 832.4929809570312 | 70.33200073242188 |
| 2019-10-10T21:32:34.199Z | false | 21.0 | 903.1640014648438 | 70.6709976196289 |
| 2019-10-10T21:32:54.258Z | true | 6.0 | 923.2230224609375 | 20.05900001525879 |

(1 of 3) 1 2 3 15

Figure 4.19: Predictive Maintenance - Smart Data

to the InfluxDB database and makes use of the InfluxDB Java Client Library [10]. This InfluxDB connection is then injected and used by the SmartDataService class.

[10] https://github.com/influxdata/influxdb-java (Accessed on: 2019-10-20)

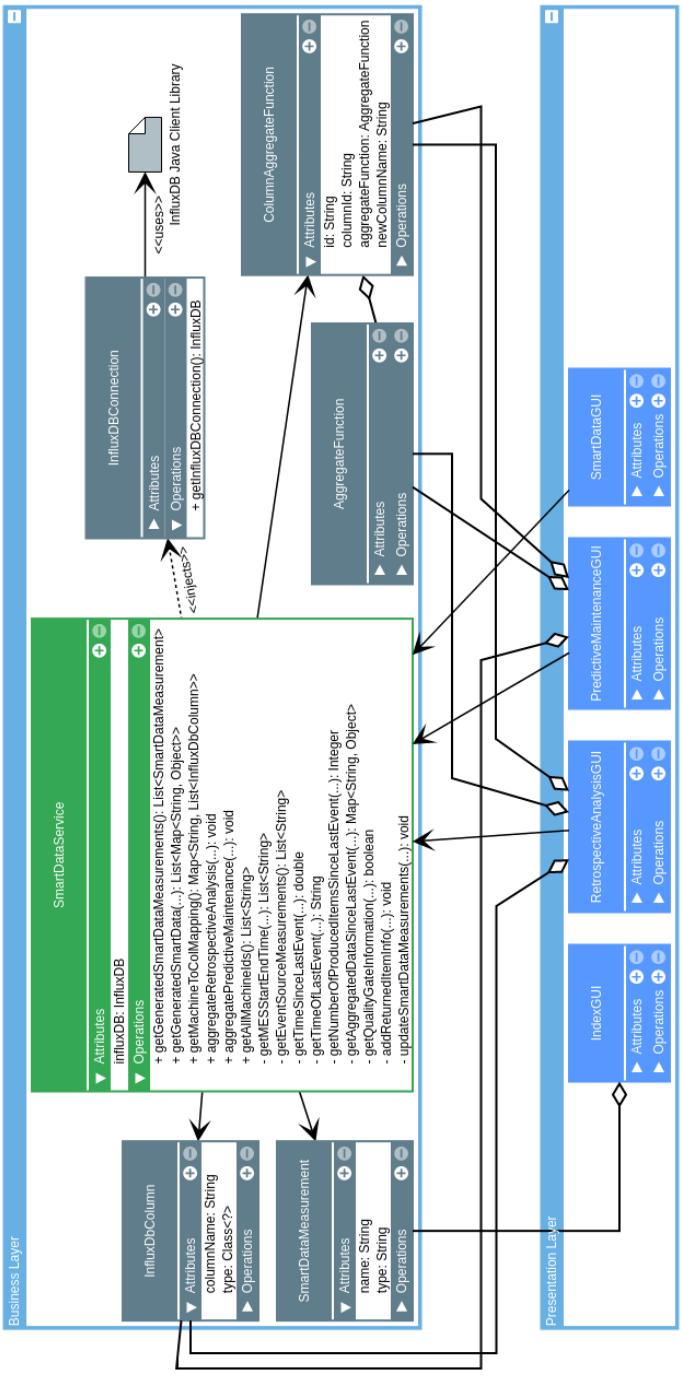


Figure 4.20: Smart Data Service - UML Diagram

# 5 Evaluation

This section explains how the evaluation has been performed and what has been used, as well as what kind of parameters or configurations have been applied.

In order to evaluate the functionality of the Smart Data Service, as already stated before, the Smart Factory Simulator is required for accepting randomly generated data. The results of the Smart Data Service are evaluated as positive if we are able to show that the value in the produced Smart Data suggest that a successful application of:

- Retrospective Analysis, and
- Predictive Maintenance

might be possible. To show this, correlation matrices have been used.

With this in mind, a couple of different configurations were used so that different scenarios can be tested. For the first use case - Retrospective Analysis - three different examples have been taken into consideration.

1. Smart Factory Simulator has been configured in that way so that only the PatternMaking machine causes sometimes that the produced T-Shirts are of lower quality than expected. When the T-Shirts are of lower quality they might get returned from the customers, but this is decided based on probability. The quality of the PatternMaking machine drops based on the number of sensor values that exceed the preconfigured maximum value. Around 110 different T-Shirts were produced from the imaginary factory and around 30 T-Shirts were returned from the customers because of their low quality.
2. In the second example, a group of machines has been taken into consideration - namely machines used for cutting. In this Smart Factory Simulator configuration, we have three cutting machines that might cause a drop in quality of T-Shirts - Cutting-M1, Cutting-M2, and Cutting-M3. When items

are produced, it is randomly decided which cutting machine should be used. Machine quality drops if a sensor value is higher than the maximum configured value. From 126 produced T-Shirts, 34 have been reported as returned items.
3. The third example is similar to the second example but instead, the quality of the cutting machines decreases if the value of a sensor of the machine drops below a minimum configured value. In this case, a total of 100 T-Shirts were produced and 35 were returned.

For the second use case - Predictive Maintenace - a large dataset has been generated where around 500 different T-Shirts were produced. In this process, machines have been broken and maintenance has been performed as well. All of this has been configured using probability when a machine might get broken. The following examples were considered:

1. The PatternMaking machine was configured in that way so that it did not get broken that often. Many more maintenances have been performed - 44 maintenance events and 8 broken events.
2. The Cutting-M2 machine was configured so that it gets broken more often than it gets fixed. In total there were 22 broken events and 10 maintenance events sent.
3. The maintenance events and the broken events sent for the Cutting-M1 machine were as follows - 10 maintenance events and 7 broken events.

## 5.1 Results

### 5.1.1 Retrospective Analysis

Referring to the first example for Smart Data aggregation, Chart 5.1 reveals that the number of returned items is strongly associated with the performance of the PatternMaking machine. It seems that 0.9 is the border of the quality at which most of the items are returned.

In the aggregated Smart Data, a cluster function has been applied, thus grouping values of sensor ”s1” in three categories:

- values between 0 - 190
- values between 190 - 1400, and
- values between 1400 - 5000.

The correlation matrix 5.2 illustrates that there is an association between the last two defined clusters - ”s1-cluster[190,1400]”, ”s1-cluster[1400,5000] - and the number of returned items. More precisely, sensor values between 1400 and 5000 are more likely to cause an item to get returned, whereas on the other hand sensor values between 190 and 1400 not. Important to emphasize, there exists a close relationship as well between those clusters and the quality of the PatternMaking machine. The same is not true for the first cluster.

In the second example of the first use case, faulty produced T-Shirts could be caused by one of the three cutting machines - Cutting-M1, Cutting-M2, or Cutting-M3. Referring to Figure 5.3 it is reasonable to say that there is a correlation between the returned items and the quality of the Cutting-M1 and Cutting-M2 machines, but not with Cutting-M3. The maximum aggregated value of machine Cutting-M2 seems to have an association with the returned items, but the same is not true for the machine Cutting-M1. Nevertheless, using the possibility offered by the Smart Data Service to freely chose aggregation functions seems to improve the situation. In Figure 5.4, the data has been aggregated using the sum function and the cluster function. The relationship between the sum function and the returned items, as well as the cluster function and the returned items, is noticable. Once again, these functions correlate with the quality of the respective machine.

Figure 5.5, which refers to the third example, and which is similar to the previous one, shows once again the connection between the quality of the machines and the items that have been returned. However, in this case, because the quality drops if sensor values fall below a certain point, it is important to notice that by choosing the minimum aggregation function we can deliver more valuable information. If we aggregate the values using the sum function, the correlation is not always accurate, as it can be seen in machine Cutting-M2 and Cutting-M3.

Referring to the illustrations, the valuable information from the above generated Smart Data suggests that those can be suitable for a successful application of Retrospective Analysis.

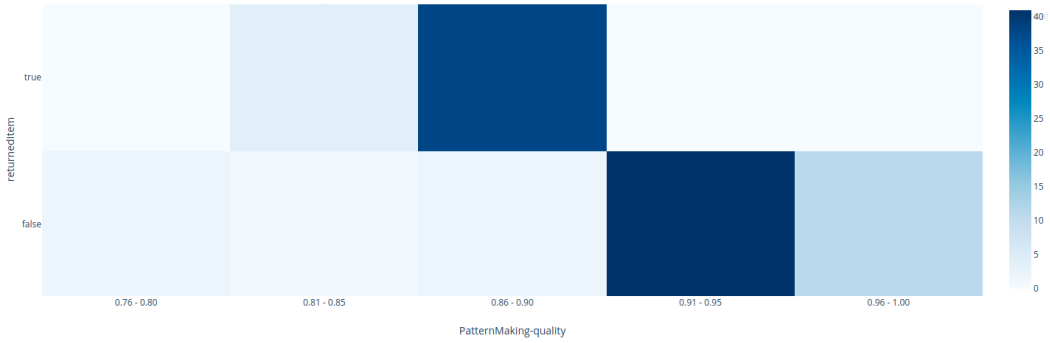


Figure 5.1: Quality of the PatternMaking machine and the returned items

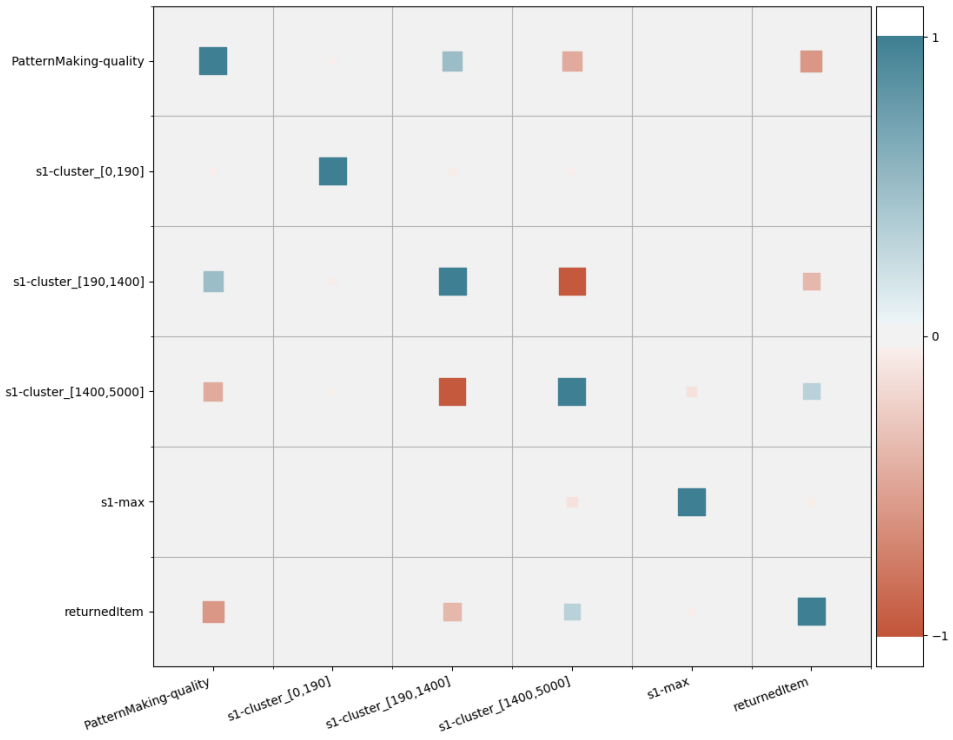


Figure 5.2: Correlation matrix of the first Retrospective Analysis example

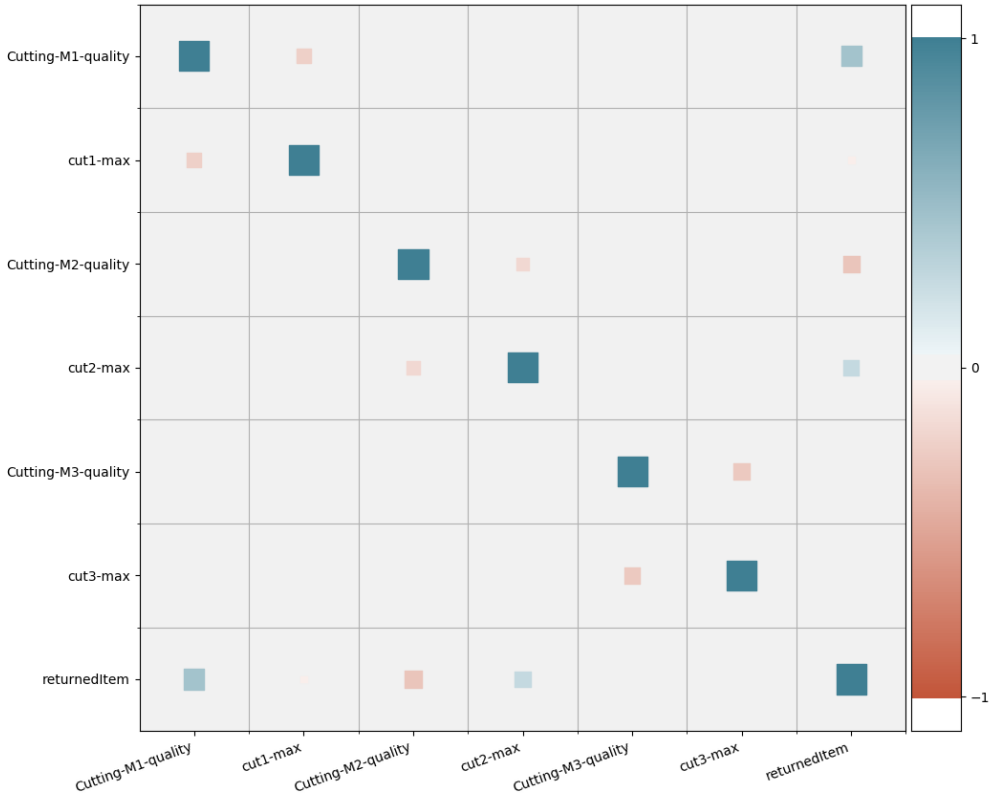


Figure 5.3: First correlation matrix of the second Retrospective Analysis example

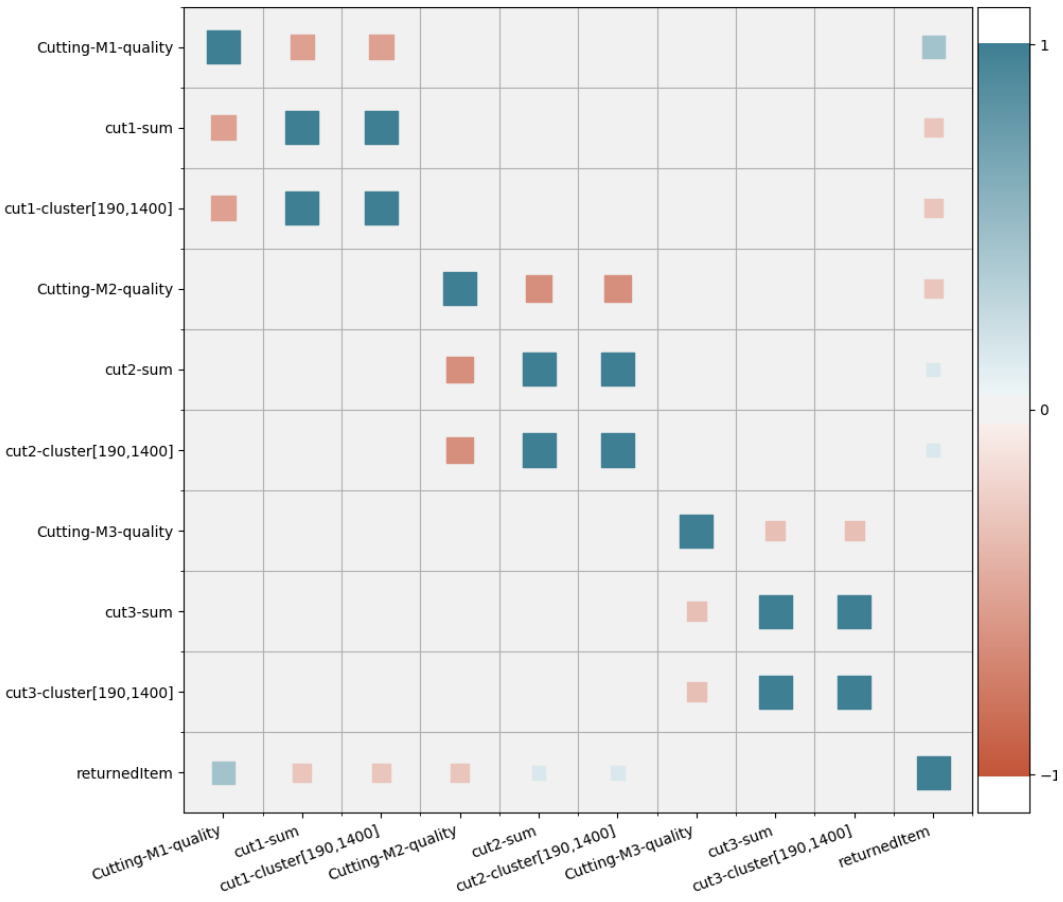


Figure 5.4: Second correlation matrix of the second Retrospective Analysis example

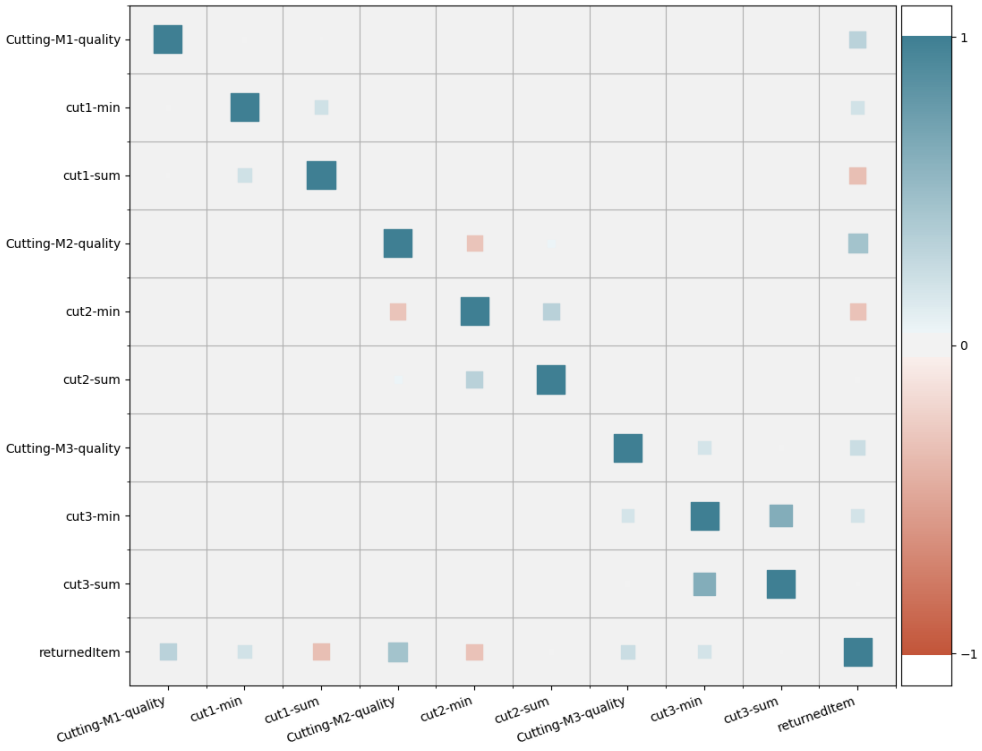


Figure 5.5: Correlation matrix of the third Retrospective Analysis example

### 5.1.2 Predictive Maintenance

Figure 5.6, which refers to the first example of the Predictive Maintenance case, where the number of performed maintenances is higher than the number of broken events, reveals the meaningful correlation between the duration since last performed maintenance and the health of the machine as well as the number of items produced since last event and the health of the machine. The relationship exists also between the number of produced items since the last event and the time since the last maintenance event. The times that the machine is broken are hardly noticeable from the correlation matrix but in order to minimize the downtimes, maintenance intervals have to be slightly shortened.

The same valuable information from the aggregated Smart Data can be noticed in the next two examples. The correlation matrix of the second example 5.7, hints that there is a close relationship between the number of produced T-Shirts since the last event and the broken events. This means that in order to predict the next best maintenance time, the number of produced items of the machine should be taken into consideration.

In the last example 5.8, the number of performed maintenances and the number of times that the machine has been broken, are quite close to one another. Again like in the previous examples, there is a strong correlation between the number of produced items since the last time and the time since the last broken event or the time since the last maintenance event.

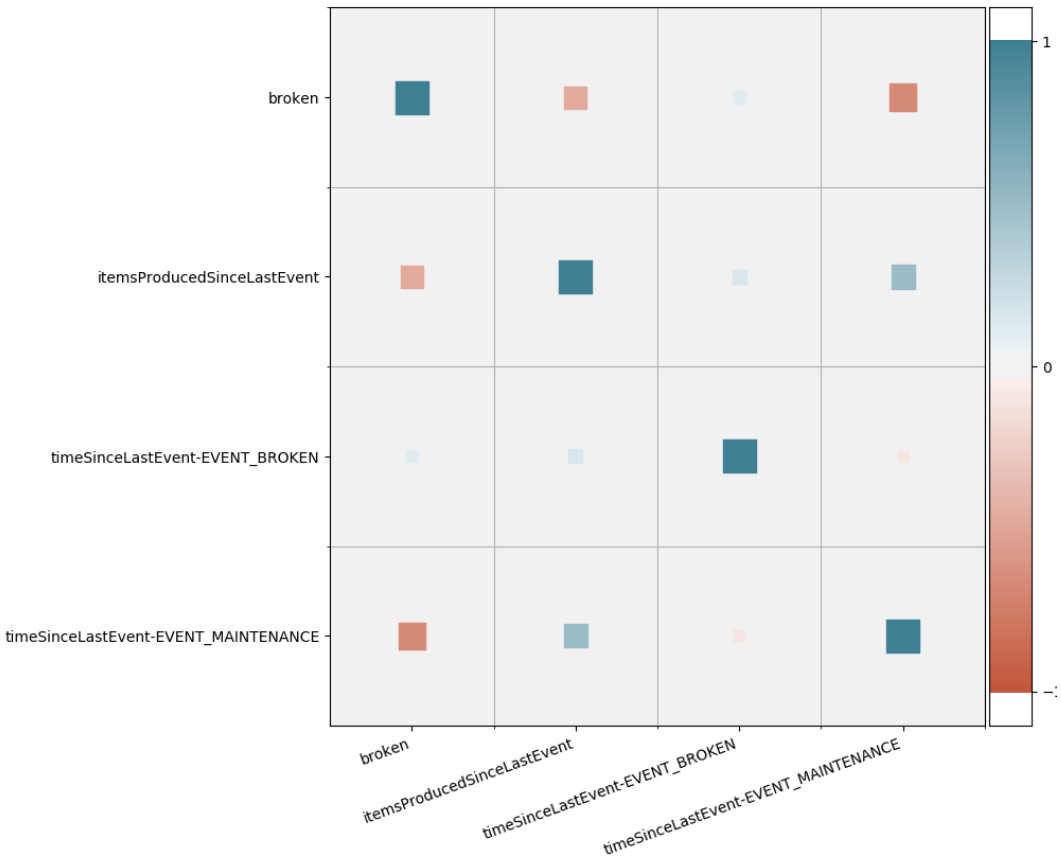


Figure 5.6: Correlation matrix of the first Predictive Maintenance example

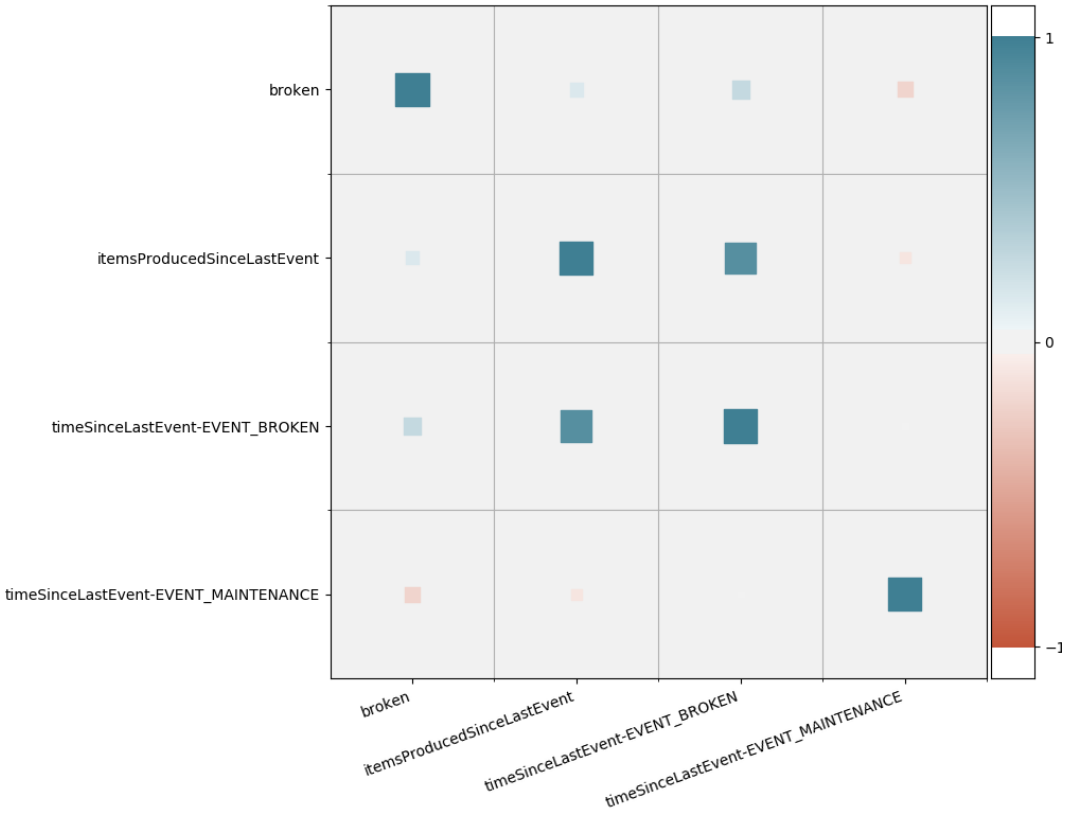


Figure 5.7: Correlation matrix of the second Predictive Maintenance example

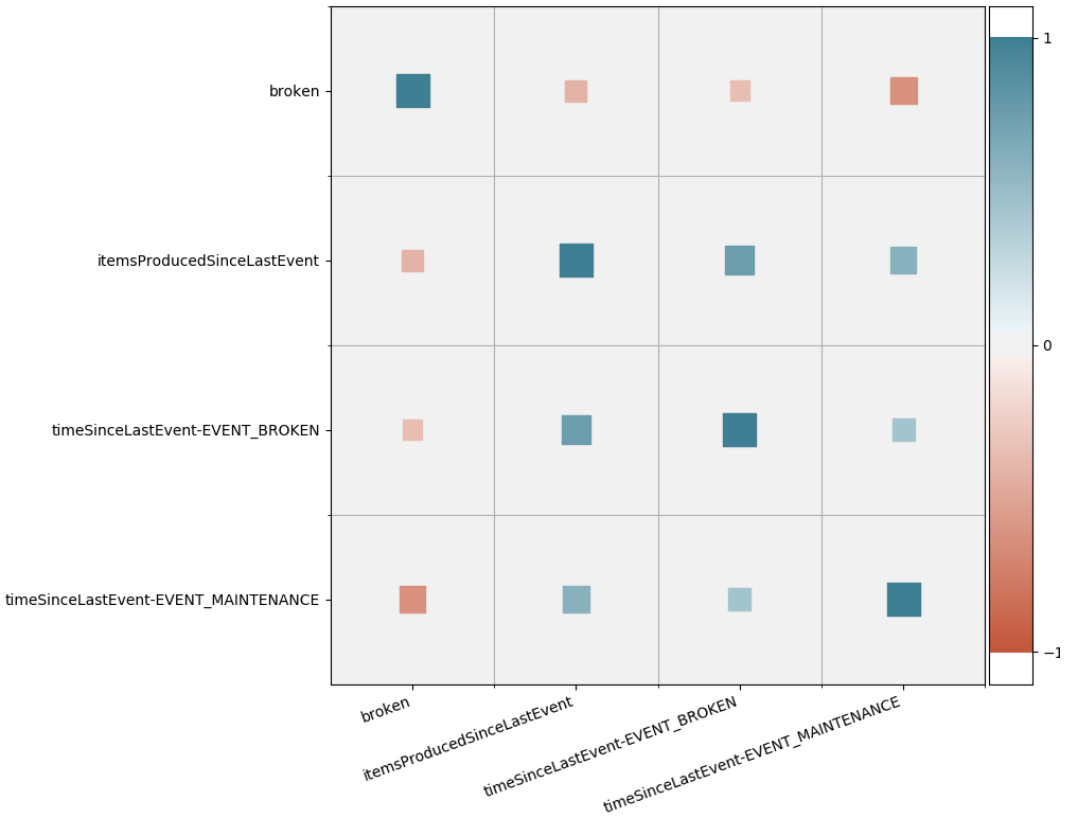


Figure 5.8: Correlation matrix of the third Predictive Maintenance example

## 5.2 Discussion

In this subsection, the results presented above are put into perspective. Even though there are different ways to display results, the method used for the evaluation of the Smart Data Service was considered to be sufficient for showing that those Smart Data contain valuable information and that they suggest that a successful application of Predictive Maintenance or Retrospective Analysis can be feasible. The correlation matrices used in the Results section reveal that the generated Smart Data using the prototype service can carry relevant information. There is the chance that this value in the aggregated data can be considerably increased by the possibility of choosing the right aggregation function as it was shown already from the examples in the Results section. Such solid correlations between the variables suggest that those Smart Data might be suitable for Retrospective Analysis and Predictive Maintenance.

However, it is important to emphasize that this solution has not been tested in a real-world scenario. Some assumptions have been made so that the system would work. An example of such an assumption is that the Manufacturing Execution System (MES) sends information telling which item is being produced in which machine in what period. This information was required to know which raw data should be selected from each machine because normally machines do not send information about which item is being produced. However, when testing this work in a real-world scenario, this might not be the case, and there might be the need to find another workaround. Another important assumption is the way that the events are sent and received. It was assumed that the events for telling if a machine is broken or if maintenance has been performed are not sent from the machine directly but from another source. This can also be different in a real-world scenario and there can be other events, which in this work were not taken into consideration.

Regarding design decisions, the usage of MQTT Protocol, Apache Kafka, InfluxDB, and Docker was considered important, because the whole system can be scalable. Even if the usage of the MQTT Protocol would not be relevant in a real-world scenario, the substitution of it with e.g. OPC UA would not be such a problem because the MQTT is loosely coupled with other systems. An Apache Kafka Source Connector for the OPC UA would be necessary in this case. Another important component is the InfluxDB and in order to use it in a real Smart Factory, it would probably be necessary to run it in a cluster, which is already supported. Nevertheless,

the proposed system does not support parallel processing and this can easily be a bottleneck in a real-world scenario. In order to overcome this, a more suitable solution would be to use the Hadoop platform, which is based on the MapReduce programming model. Hadoop supports distributed storage and processing for Big Data.

# 6 Conclusions

In this Master's Thesis, a prototype service is presented that is designed for collecting raw data from different incoming sources in a Smart Factory and transforming them into more valuable information which can serve for further analytics and processing. Thus, the prototype service transforms the ingested raw data coming from multiple sources into the so-called Smart Data. The end result of the Smart Data is usually a row in the database with multiple columns containing valuable information for a produced item in a Smart Factory.

To the extent of my knowledge and the performed research, there does not exist such a service that is able to collect and aggregate data automatically in that way, so that it always contains the right valuable information that can be used for further analytics. Because the context from where the raw data is coming is crucial, the domain knowledge is necessary for choosing the right aggregation function. This is the reason that the prototype service offers different aggregation mechanisms and gives the possibility to the user to choose appropriate functions.

Finally, the Smart Data Service has been evaluated using the developed Smart Factory Simulator and based on the results and the illustrations used, they show that by giving the responsibility to the appropriate user for choosing the right aggregation mechanism, the generated Smart Data can contain valuable information that might be used for further analytics.

## 6.1 Future Work

The following points were considered relevant for the future development of the Smart Data Service:

- Smart Factory Simulator: Extend the functionality of the Smart Factory Simulator so that it generates more heterogeneous data and it enables better control over the generation of events. The simulator would be easier to use and more understandable if there is a GUI for configuring and visualizing the imaginary Smart Factory.
- Real-life scenario: Put the Smart Data Service to the test using a real-life Smart Factory and analyze the results.
- New aggregation possibilities: Extend the number of possible aggregation functions in the Smart Data Service and thus make the service more flexible. One possibility would be to offer a way so that the user can define aggregation functions using written formulas. Another interesting functionality would be to allow the user to define aggregation conditions based on the values of cross-reference variables.
- Scaling: Extend the Smart Data Service infrastructure so that it has better support for scaling. Apache Kafka running in a cluster with multiple nodes together with the Hadoop platform for parallel processing and data distribution is a suggestion for the solution to this problem. After the right infrastructure is built, stress-test the prototype service using enormous amounts of data.

# Bibliography


[1] C. Alexakos, C. Anagnostopoulos, A. Fournaris, C. Koulamas, and A. Kalogeras. IoT integration for adaptive manufacturing. In *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*. IEEE, May 2018.

[2] S. D. Anton, D. Fraunholz, J. Zemitis, F. Pohl, and H. D. Schotten. Highly scalable and flexible model for effective aggregation of context-based data in generic iiot scenarios. *arXiv preprint arXiv:1906.03064*, 2019.

[3] Y. Bao, L. Ren, L. Zhang, X. Zhang, and Y. Luo. Massive sensor data management framework in cloud manufacturing based on hadoop. In *IEEE 10th International Conference on Industrial Informatics*. IEEE, July 2012.

[4] P. Barnaghi, A. Sheth, and C. Henson. From data to actionable knowledge: Big data challenges in the web of things [guest editors' introduction]. *IEEE Intelligent Systems*, 28(6):6–11, Nov. 2013.

[5] P.-J. Benghozi, D. Krob, A. Lonjon, and H. Panetto, editors. *Digital Enterprise Design & Management*. Springer International Publishing, 2014.

[6] B. Chaudhry, J. Wang, S. Wu, M. Maglione, W. Mojica, E. Roth, S. C. Morton, and P. G. Shekelle. Systematic review: Impact of health information technology on quality, efficiency, and costs of medical care. *Annals of Internal Medicine*, 144(10):742, May 2006.

[7] M. Chelly and H. Mataillet. Social media and the impact on education: Social media and home education. In *2012 International Conference on E-Learning and E-Technologies in Education (ICEEE)*. IEEE, Sept. 2012.

[8] S. Deb. Information technology, its impact on society and its future. *Advances in Computing*, 4(1):25–29, 2014.

[9] A. Gandomi and M. Haider. Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management*, 35(2):137–144, Apr. 2015.

[10] F. Iafrate. *From Big Data to Smart Data (Advances in Information Systems Set)*. Wiley-ISTE, 2015.

[11] P. K. Illa and N. Padhi. Practical guide to smart factory transition using IoT, big data and edge analytics. *IEEE Access*, 6:55162–55170, 2018.

[12] S. M. R. Islam, D. Kwak, M. H. Kabir, M. Hossain, and K.-S. Kwak. The internet of things for health care: A comprehensive survey. *IEEE Access*, 3:678–708, 2015.

[13] A. Ismail, H.-L. Truong, and W. Kastner. Manufacturing process data analysis pipelines: a requirements analysis and survey. *Journal of Big Data*, 6(1), Jan. 2019.

[14] A. Katal, M. Wazid, and R. H. Goudar. Big data: Issues, challenges, tools and good practices. In *2013 Sixth International Conference on Contemporary Computing (IC3)*. IEEE, Aug. 2013.

[15] J. Y. Lee, J. S. Yoon, and B.-H. Kim. A big data analytics platform for smart factories in small and medium-sized manufacturing enterprises: An empirical case study of a die casting factory. *International Journal of Precision Engineering and Manufacturing*, 18(10):1353–1361, Oct. 2017.

[16] A. Lenk, L. Bonorden, A. Hellmanns, N. Roedder, and S. Jaehnichen. Towards a taxonomy of standards in smart data. In *2015 IEEE International Conference on Big Data (Big Data)*. IEEE, Oct. 2015.

[17] D. Lucke, C. Constantinescu, and E. Westkämper. Smart factory - a step towards the next generation of manufacturing. In M. Mitsuishi, K. Ueda, and F. Kimura, editors, *Manufacturing Systems and Technologies for the New Frontier*, pages 115–118, London, 2008. Springer London.

[18] A. Maier, S. Schriegel, and O. Niggemann. Big data and machine learning for the smart factory—solutions for condition monitoring, diagnosis and optimization. In *Industrial Internet of Things*, pages 473–485. Springer, 2017.

[19] K. Nagorny, P. Lima-Monteiro, J. Barata, and A. W. Colombo. Big data analysis in smart manufacturing: A review. *International Journal of Communications, Network and System Sciences*, 10(03):31–58, 2017.

[20] J. W. Patty and E. M. Penn. Analyzing big data: Social choice and measurement. *PS: Political Science & Politics*, 48(01):95–101, Dec. 2014.

[21] E. Qin, Y. Long, C. Zhang, and L. Huang. Cloud computing and the internet of things: Technology innovation in automobile service. In *Human Interface and the Management of Information. Information and Interaction for Health, Safety, Mobility and Complex Environments*, pages 173–180. Springer Berlin Heidelberg, 2013.

[22] K. Schwab. *The Fourth Industrial Revolution*. Crown Business, first edition, 2017.

[23] B. Suryajaya, C.-C. Chen, M.-H. Hung, Y.-Y. Liu, J.-X. Liu, and Y.-C. Lin. A fast large-size production data transformation scheme for supporting smart manufacturing in semiconductor industry. In *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*. IEEE, Aug. 2017.

[24] I. Triguero, J. Maillo, J. Luengo, S. Garcia, and F. Herrera. From big data to smart data with the k-nearest neighbours algorithm. In *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, Dec. 2016.

[25] T. Wang, M. Z. A. Bhuiyan, G. Wang, M. A. Rahman, J. Wu, and J. Cao. Big data reduction for a smart city's critical infrastructural health monitoring. *IEEE Communications Magazine*, 56(3):128–133, Mar. 2018.

[26] X. Xu and Q. Hua. Industrial big data analysis in smart factory: Current status and research strategies. *IEEE Access*, 5:17543–17551, 2017.

[27] K. Zhou, C. Fu, and S. Yang. Big data driven smart energy management: From big data to big insights. *Renewable and Sustainable Energy Reviews*, 56:215–225, Apr. 2016.

[28] T. Zhu, S. Dhelim, Z. Zhou, S. Yang, and H. Ning. An architecture for aggregating information from distributed data nodes for industrial internet of things. *Computers & Electrical Engineering*, 58:337–349, Feb. 2017.