Exploring an Energy-Status-Data Set from Industrial Production

Bachelor's Thesis

Arved Hansen

September 30, 2019

Kiel University
Department of Computer Science
Software Engineering Group

Advised by: Prof. Dr. Wilhelm Hasselbring
Sören Henning, M.Sc

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Kiel, 30. September 2019

Abstract

In this thesis we explore an energy status data set from an industrial production environment. This is done to learn more about industrial energy consumption, which helps enterprises to reduce their costs and can assist in reducing the overall energy consumption for ecological reasons. The exploration contains a visual analysis of the data and a forecasting of the future energy consumption. In order to do this we use the Titan monitoring platform, which has a component to visualize energy consumption. This component delivers several possibilities for a visual analysis. To bring the data into the Titan platform we design a tool which in general processes an energy consumption data set but is built in a way that it functions as a part of the Titan platform. Furthermore, we use the infrastructure of the Titan platform for the forecasting. This way we can deploy the monitoring and the forecasting as one product. However, due to the microservice architecture of the Titan platform the monitoring and the forecasting can be deployed individually.

The processing of the data can be adjusted to different modes in order to improve visual analysis and live demonstrations of the Titan platform. In the visual analysis we discuss different patterns in the data. Furthermore, these patterns serve as a base for the forecasting as the patterns should be recognized by a forecasting. For the forecasting we implement two different prototypes, one based on a statistical model and the other one is using neural networks. However, only the statistical approach delivered results. This is due to the fact that the forecasting in this thesis is not a finished implementation but serves as a prototype for future implementations.

Contents

Chapter 1

Introduction

1.1 Motivation

The efficient use of energy is becoming more and more important for society and economy. To help raise this efficiency we need to know more about the energy consumption.

This bachelor's thesis has the goal to explore an energy status data set from an industrial production environment. The data set we explore is the HIPE data set. To explore this data set we first have to read and filter it and then we need a visualization of the data. Based on this we implement a prototype for a forecasting of the energy consumption which is used for gaining a further insight in the data.

To visualize the data set we use the Titan platform [Henning et al. 2019a]. The Titan platform is a real time monitoring platform for industrial production environments to gather an insight in energy usage. However, to use it we need to design a microservice which can read, filter, and translate previous recorded data, such as the HIPE data, so it can be stored by the Titan system. Afterwards, the Titan system creates statistics about the stored data and visualizes both, the statistics and the data.

An additional way to explore an energy status data set is to implement a forecasting of the energy consumption of the monitored machines. The forecast is needed due to four "business-critical goals" [Henning et al. 2019b]. Three of these goals are supported by a forecasting and for one goal a forecasting is rated as necessary. These first three goals are reporting, optimization and fault detection. The reporting supports to gain an ISO50001 [*Energy management systems – Requirements with guidance for use*] certification for which a revelation of all energy consumption is necessary. A forecasting of the energy consumption can also be reported in order to present more extensive data. There are two aspects in the optimization which can be supported by a forecasting. Firstly, there is the minimization of the overall energy consumption to reduce costs and for ecological reasons. Secondly, enterprises with a high energy consumption pay extra for the highest peak in a billing period. Both of these aspects are supported by a forecasting as with an insight in future energy consumption production processes can be scheduled more efficient. Furthermore, when there is a predicted load peak, consumers which are not necessary at that time can be rescheduled. The fault detection is supported by a forecasting, as the forecasted values and the real time values can be compared. When there is a gap between these values, which can not be explained with a higher or lower workload than expected, a fault in the machine might be the reason. For the fourth goal, predictive maintenance, they rate forecasting as a requirement. Therefore, the predictive maintenance is the main use case for a forecasting. With predictive maintenance the regular used time-based intervals for maintenance can be optimized, since maintenance can be scheduled when the machine is expected to have no load at that time. Closely connected to fault detection, maintenance can be scheduled when the forecasted values differ from the real values in a way that suggests that an error might occur soon.

It is reasonable to use the HIPE data set [Bischof et al. 2018] for this exploration, since it contains data with the energy consumption of an industrial production environment and has a resolution and number of monitored machines suitable for the Titan system.

1.2 Goals

In this section the goals for this bachelor's thesis are presented.

1.2.1 G1: Designing a Tool for Data Exploration

The first goal is to design and implement a tool which can be used for exploring an energy status data set. This tool should feature four different modes for the data processing: a real time mode, a speed up mode, a mode for instant data transmission, and a demo mode. These four modes aim at the goals presented in the following.

Real Time Processing

In the first mode we want to achieve a processing in real time, which means in this case that we feed the data into the database in the original recording speed.

Speed Up Processing

Additionally, we want the option to speed the real time processing up, while maintaining a behavior similar to the real time monitoring. The real time and the speed up mode can be used for evaluating the incoming data and for providing a live demonstration of the Titan platform without having a real production environment which feeds the platform with live data. This demonstration would be more expressive than feeding random data into the system.

Instant Transmission

Furthermore, we want a feature to process the data as fast as possible in order to handle large data sets in a short period. In this mode there will be no correlation between the timestamp of a measurement and the time this measurement is sent out.

Demo Mode

The demo mode focuses on assisting live demonstrations of the Titan platform. The timestamps from the data set will be substituted by the current timestamp. This mode will be compatible with all three modes mentioned above. However, using it with the real time mode is the most applicable option, as then the behavior is similar to a real production environment connected to the system.

1.2.2 G2: Visually Exploring the HIPE Data Set

When we can process the HIPE data set we want to explore it by using the visualization of the Titan platform. This is done in order to gain an insight into the information such a data set can give us. The gained information will be used later in the energy consumption forecasting to choose fitting models. Moreover, we investigate how the Titan platform can work with this data and in which way the system could be optimized. This includes different types of visualization and the creation of statistics for the data.

1.2.3 G3: Prototyping a Forecasting for Energy Consumption

The third goal is to design a system which forecasts the energy consumption, in order to explore the data further. After making the concept for this forecasting, the goal is to implement a prototype of it. To implement the forecasting together with the HIPE record bridge is reasonable since we can evaluate its functionality with a large and detailed data set from a real world production environment. The advantage of this data set over other data sets is the high resolution and that it comes from a industrial production environment and not from residential or office buildings [Bischof et al. 2018]. We also could use data gathered by ourselves, but we would have to implement the sensors for data recording in an enterprise large enough and collect the data for three months to have a data set this large, which is not necessary with a provided data set this detailed. Furthermore, Bischof et al. [2018] already propose a load forecasting with their data set as a future research. This is due to the data set featuring different production schedules and therefore it has a high variety of values for the energy consumption.

1.2.4 G4: Evaluation of the Forecasting

After implementing the load forecasting we want to evaluate it. One important point is how close our approach can get to the real data. Another interesting point we can evaluate is how predictable the data for the active power is. For this evaluation we use graphs containing the forecasted values and calculate the mean squared error of the forecasting.

1.3 Document Structure

In Chapter 2 there is information about concepts and technologies this thesis is based on. After that in Chapter 3 the realization of goal G1 is presented. Afterwards in Chapter 4 we explore the data set using the tools created before. Chapter 5 contains the concepts and implementations regarding the load forecasting. In Chapter 6 we evaluate the implementations and findings of Chapter 3, Chapter 4, and Chapter 5. After looking at related work in Chapter 7 we draw our conclusions and show possible future work in Chapter 8.

Chapter 2

Foundations and Technologies

In the following the most significant concepts and technologies this thesis is based on are presented.

2.1 Microservice Architecture

With a *microservice architecture* [Hasselbring 2016] the main goal is to make the software more scalable in comparison to a monolithic architecture. To achieve this every individual functionality of the software obtains a full-stack implementation of its functions. With this implementation dependencies are avoided and technologies, such as databases, can be chosen more fitting for their purpose. Scalability and fault tolerance are supported by this architecture as new instances of the services can be started easily and therefore a high availability can be reached. A microservice architecture can also benefit to DevOps, since the individual microservices can be integrated and deployed separately.

2.2 Industrial DevOps

DevOps [Bass et al. 2015] is used as a term for the combination of development and operation of a software system. Its intent is to speed up the process of changing the system and deploying these changes while keeping a high quality. This includes automated testing before and monitoring after the deployment. However, the process does not end with that. It also includes the operation and the constant improvement of the system in a cyclic process.

Industrial DevOps [Hasselbring et al. 2019] integrates the DevOps values into industrial systems. There are three core elements. The first is a cyclic process as explained above. Secondly, there is a lean organizational structure. This organizational structure improves adapting the software to the enterprise, rather than the enterprises having to adapt its production process to the software. The third element is a "Customer-Centric Value Generation" [Hasselbring et al. 2019], which means that there is no market research before the production process starts, but measurements about new features are taken continuously after releasing them to customers.

2.3 The Industrial DevOps Platform Titan

The *Titan monitoring platform* [Henning et al. 2019a] is a platform for monitoring the power consumption of devices, machines, and production plants. One can install many different sensors which report to the platform. The platform collects the data and then stores and visualizes them. The Titan platform is built on the Industrial DevOps approach. Furthermore, it is a microservice based architecture, which fits well for the organizational structure of adapting the software easily. Figure 2.1 shows the four microservices *Record Bridge*, *History*, *Statistics*, and *Configuration* and the *Visualization* component. The Record Bridges are used for obtaining the data from the sensors, converting them to another format, and forwarding them to the other components via the messaging system. They are built with the concept of a flow engine, which is presented below. The History microservice has the purpose to store the data in a database and to handle the access to it. Furthermore,

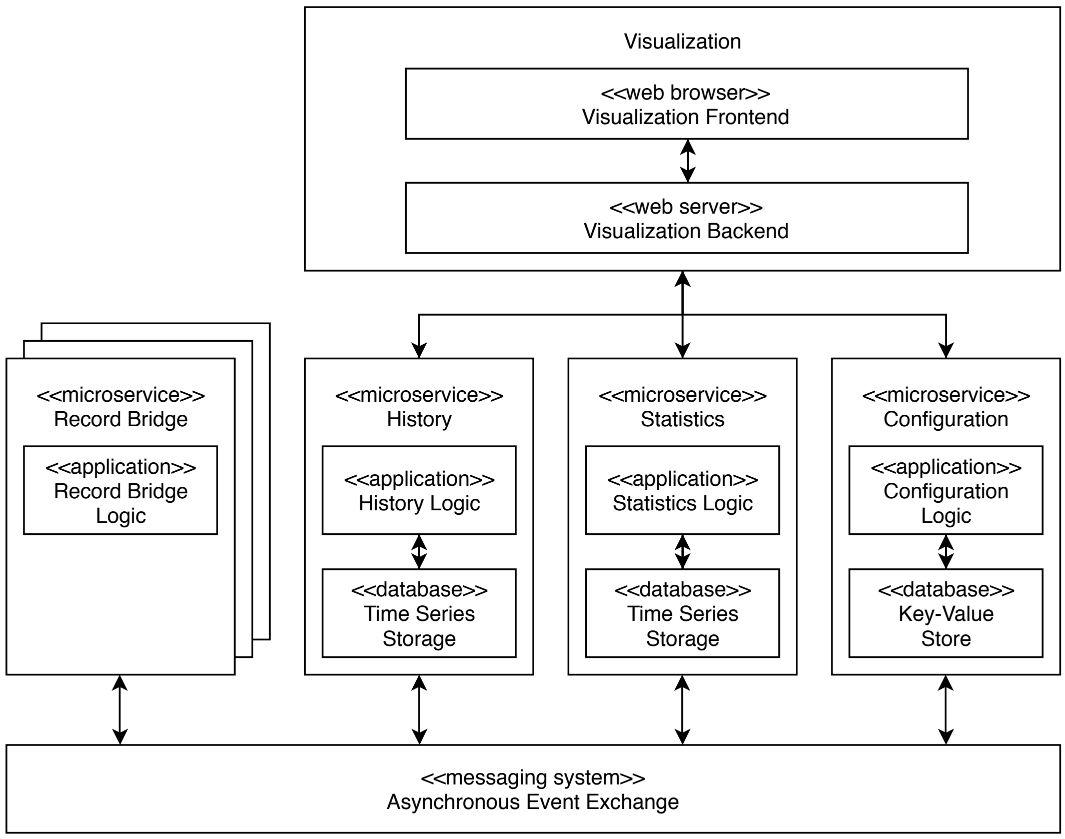
Visualization
<<web browser>> Visualization Frontend
<<web server>> Visualization Backend
<<microservice>> Record Bridge
<<application>> Record Bridge Logic
<<microservice>> History
<<application>> History Logic
<<database>> Time Series Storage
<<microservice>> Statistics
<<application>> Statistics Logic
<<database>> Time Series Storage
<<microservice>> Configuration
<<application>> Configuration Logic
<<database>> Key-Value Store
<<messaging system>> Asynchronous Event Exchange

Figure 2.1. Titan microservice-based architecture [Henning et al. 2019a], updated version

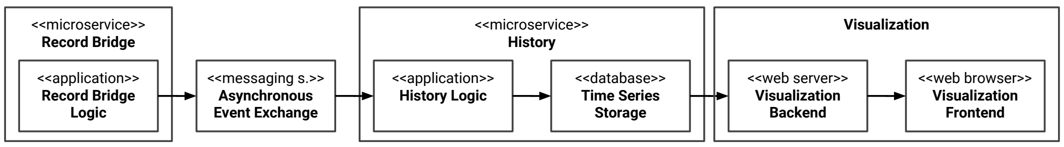

Figure 2.2. Data flow in the Titan platform [Henning et al. 2019a]

it aggregates the data for groups of sensors in real time. In order to gain a better insight in the data the Statistics microservice provides different kinds of charts and indicators. These indicators show whether the energy consumption rose or fell in the last hour, day, or week. The charts vary from a pie chart which presents the percentage each machine contributes to the overall energy consumption, to graphs which present the average energy consumption for every hour of the day or day of the week. The Configuration's purpose is to handle settings which do not belong to the individual services but are system-wide. The Visualization contains a frontend and a backend. The frontend provides an interactive web page and the backend forwards requests to the other microservices and hides this architecture from the user.

Henning et al. [2019a] also present the data flow shown in Figure 2.2. It shows how the data comes from the Record Bridge, is stored in the History, and then is forwarded to the Visualization. The data is sent from one microservice to the next one using the asynchronous event exchange. The asynchronous event exchange system used in the Titan platform is Kafka.

Kafka

Kafka [Apache Software Foundation 2017] is a distributed streaming platform. One use case for Kafka is the operation as a message broker, as it is used in the Titan platform. For the message distribution Kafka has a *publish-subscribe* [Eugster et al. 2003] architecture. In this architecture there are producers and consumers. The producers send the messages which are then stored in a buffer. The consumers task is to read the messages from this buffer. Kafka allows these consumers to be grouped together. When a message is sent to the buffer it is read exactly once by every group which subscribed to this buffer. This has two effects. The first effect is that if consumers send more messages than one consumer can process, other consumers can join this group so all messages are processed. This is useful in a microservice architecture such as the Titan platform. For example, when the Record Bridges send too many measurements for one History instance, the History can be scaled up and the different instances can work together on the incoming data stream. The second effect is that one message is read multiple times when different consumer groups subscribe to the same buffer. Therefore, two different microservices can read the measurements sent by the Record Bridges in the Titan platform.

Flow Engine

The Record Bridges are built with a Flow Engine. This Flow Engine consists of two components: Bricks and flows [Hasselbring et al. 2019]. These bricks are modular software components and one system is built with multiple bricks. The bricks are connected with flow-based programming [Morrison 2010]. Connected bricks are called a flow. The bricks in one flow are connected in a predetermined order and the flow engine handles the transmission of flow packets, which are used for data transmission from one brick to the next one.

2.4 The HIPE Energy Status Data Set

The *HIPE data set* [Bischof et al. 2018] is a data set with energy-related data from a real world production environment. HIPE is the acronym for High-resolution Industrial Production Energy. The data was collected over three months, from October the 1st until December the 31st 2017. However, there is a second set of files containing one week of these three months, starting at October the 23rd. One measurement was taken every 5 seconds with 114 or 362 quantities (depending on the monitored machine) measured every time. For this exploration we are only interested in 3 of these quantities: The timestamp, the machine name and the active power in kilowattss.

The data was collected in an electronics production site operated by the Institute of Data Processing and Electronics (IPE) of Karlsruhe Institute of Technology (KIT) and there were 11 machines monitored. These machines are: A pick-and-place-unit, which places components on a printed circuit board (PCB), a soldering oven to solder the components onto the PCB and a washing machine to clean the PCBs afterwards. Moreover, there is a screen printer which prints material to connect components, two vacuum pumps to create a vacuum for the other machines, a high temperature oven which heats up to 1200 °C and an oven with a vacuum chamber. Additionally, there is a chip saw which cuts a silicon wafer of the chips and a chip press for heat treatment under high pressure. The rest of the building is combined as the main terminal which includes some smaller machines, the air conditioning, and the energy consumption of the offices.

2.5 The Machine Learning Interface TensorFlow

TensorFlow [Martin Abadi et al. 2015] is an implementation and an interface for machine learning algorithms. It is an open source project developed by Google. The system is built to run on a variety of different platforms. This includes desktop operating systems as well as mobile platforms, such as Android or iOS. Furthermore, it supports different sizes of hardware, ranging from of the shelf products to highly specialized machines containing thousands of GPUs. This flexibility aims at the goal to simplify the usage of machine learning in real-world implementations. Although, TensorFlow is mainly used for machine learning and deep neural networks, it is assumed that other domains, for example numerical computations, can profit from TensorFlow applications.

2.6 The Prophet Forecasting Model

Prophet [Taylor and Letham 2017] is a concept and an open source implementation of a time series forecasting model developed by Facebook. A time series is a list of pairs containing a timestamp and a corresponding value. In contrast to TensorFlow, Prophet is a statistical model, which means it does not improve itself. It uses a decomposable time series model which has three components. These components are trend, seasonality, and holidays.

Trend

The trend models non-periodic changes in the data. As an example, regarding the energy consumption in an industrial production environment a trend could be a growing enterprise which produces more goods or a rising degree of automation shifting human workforce to electrical powered workforce.

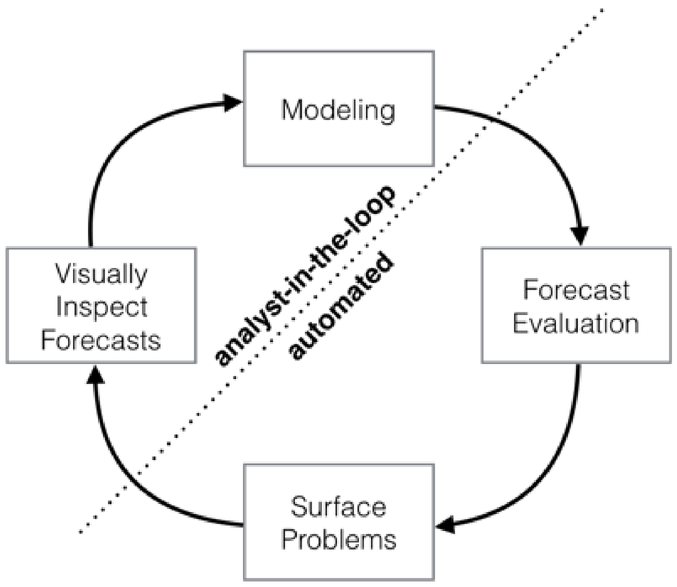

Figure 2.3. Analyst-in-the-loop approach at Prophet [Taylor and Letham 2017]

Seasonality

The seasonality models the periodic changes in the data. These can be a lower energy consumption on weekends or rising sales every year before Christmas. The possibility to model yearly, weekly, and daily seasonality is built in. Furthermore, there is the opportunity to add custom seasonalities one would expect in the data.

Holidays

With the holidays irregular schedules are represented. This component can model holidays as well as holidays which are enterprise specific. Prophet offers the possibility to add the holiday calendar of given countries to the model as well as an interface to add custom holidays.

For the optimization of the forecasts Prophet relies on the analyst-in-the-loop approach shown in Figure 2.3. In the automated section Prophet fits the model with the given data, forecasts future data, and then returns the model to the analyst. The analyst then has the opportunity to analyze the model and adapt the parameters of the components explained above.

Chapter 3

Designing a Tool for Data Exploration

While designing a tool to explore the data set there are several significant considerations coming into account. These considerations regard the goals which the tool should fulfill, as well as technical limitations which are a result from the architecture of the Titan platform. Therefore, we have different approaches which can be used for achieving the same goals. In this chapter we discuss three approaches in Section 3.1 and present an implementation of one of these approaches in Section 3.2. These approaches and the implementation are not using the flow engine as it was not fully implemented at the time the concepts were made. However, to use the concepts with the flow engine, the changes presented in Section 3.3 could be applied.

3.1 Architecture

In the following we discuss the general concept and three different approaches we have for the Record Bridge, along with their advantages and disadvantages. These approaches are based on using them with the Titan platform, but as the output of each is a stream of measurements they can be adapted easily for other use cases.

3.1.1 General Concept

In general, the concept to process the data set is to separate the tasks of the Record Bridge into two components. The first component is the HIPE Reader. The HIPE Reader's task is to handle the initialization, the filtering, and the translation of the data. The input for the HIPE Reader is a directory, which can contain multiple CSV-files with energy status data stored in them. The HIPE Reader outputs a stream of triples with one triple per measurement consisting of the machine name, the timestamp when the measurement was recorded, and the energy consumption in kilowatts. The second component is the Transmission Mode Manager. It reads the output from the HIPE Reader and sends it out to the Control Center Adapter. The Transmission Mode Manager does not manipulate the data itself, but handles the stream according to the different processing modes defined in Section 1.2. Both of these components can be designed as either one larger microservice that features the whole implementation or two separated microservices with split up tasks, standing next to the other Record Bridges.

In order to implement this concept we have three different approaches: A generic approach, an approach using multiple threads, and an approach to access all files in parallel. These approaches do not necessarily exclude each other, but can partially be used at the same time.

3.1.2 Generic Approach

Figure 3.1 visualizes the generic approach. In the figure we see existing parts of the Titan platform highlighted in grey. The Raritan PDUs Bridge and the Frako KN Bridge directly send their data to the Titan Control Center through the Control Center Adapter. With white background the figure shows the HIPE Reader and the Transmission Mode Manager. The goal of this approach is to separate the Record Bridge into the HIPE Reader and the Transmission Mode Manager not only logically but as two different microservices or respectively as different bricks if the flow engine is used. There are two advantages of this approach. Firstly, the Transmission Mode Manager could benefit from the microservice architecture and the advantages that come with this, which are explained in Chapter 2. Secondly, the generic approach would improve the reusability of the code. In our design the structure of the data sent to the Transmission Mode Manager would be the same as for the data sent to the Control Center Adapter, so the Raritan PDUs Bridge and the Frako KN Bridge could send their data to the Transmission Mode Manager to allow a slower processing. Other Record Bridges which read data from files, could use the Transmission Mode Manager the same way as the HIPE Reader.

The disadvantage and the reason we did not implement this approach is that the Transmission Mode Manager would have to buffer all the incoming data which is problematic for the amounts of measurements in the HIPE data set or other, possibly larger, data sets. A possibility to overcome this buffering would be that the Transmission Mode Manager actively requests data from the HIPE Reader. However, this would be against the design of the Titan system.

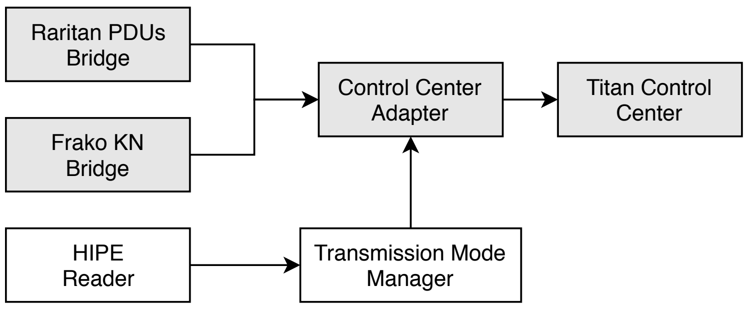

Figure 3.1. Generic approach for the Record Bridge

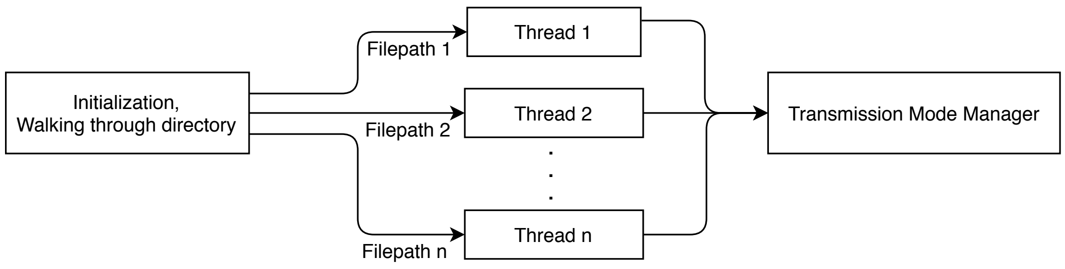

Figure 3.2. Multithread approach for the Record Bridge

3.1.3 Multithread Approach

Another approach is to use multiple threads for the data processing. This could be used with either a generic Transmission Mode Manager or a Record Bridge which combines the HIPE Reader and the Transmission Mode Manager. The concept for this is, as shown in Figure 3.2, to create multiple threads and pass one file path to each of them. The files can then be processed in parallel and can profit from a multicore CPU. This concept again led to the problem of buffering either the corresponding file in each thread or the whole data set in the Transmission Mode Manager. Furthermore, we would have to make assumptions on the underlying scheduler. The scheduler could manage the threads in a way that a thread only starts when the thread before has ended. When the instant transmission is selected this would not be problematic as it does not increase the overall computation time. The problems come with the real time mode as its purpose is to have the behavior of all data streams coming in parallel. This problem led to the parallel reading approach.

3.1.4 Parallel Reading Approach

In order to avoid the problems with buffering or scheduling there is the idea of only reading the currently needed measurement from every file. This does not work together with the generic approach as the Transmission Mode Manager would have to request a new value from the HIPE Reader each time it sends out a message to the Control Center Adapter. A disadvantage of this approach is that the files stay opened through the whole processing.

Figure 3.3 shows the concept with two lists, one containing the files of the data set and the other containing one measurement of each file. Due to this architecture the program can send the measurements from the second list to the Control Center Adapter in a predetermined way. Therefore, all four transmission modes, specified in Section 1.2, can be implemented.

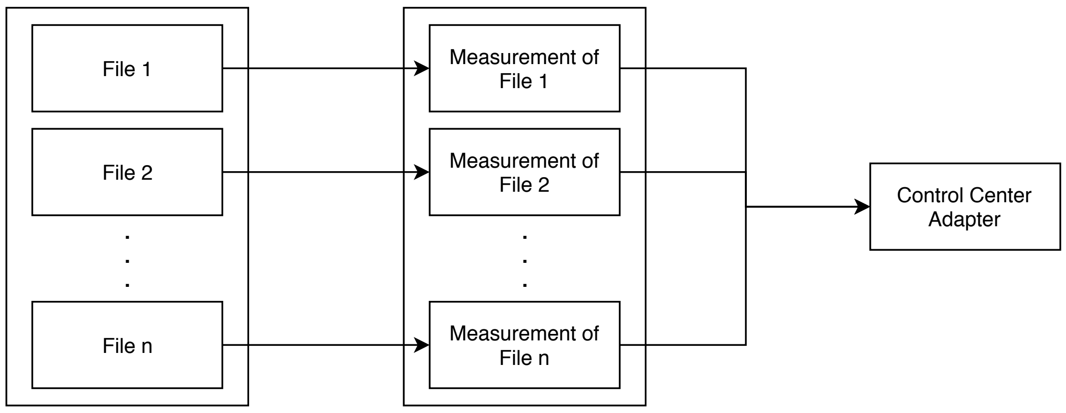

Figure 3.3. Parallel reading approach for the Record Bridge

3.2 Implementation

The generic approach does not work together with the current architecture of the Titan platform and the multithread approach has the problem with the assumptions made on the scheduler. Therefore, we implement the parallel reading approach.

3.2.1 File Management

Due to the parallel reading we need a way to handle all the files concurrently as opening and closing each file for each measurement would be very inefficient. To avoid this ineffiency we use the context manager ExitStack. The ExitStack is part of the Python library contextlib [Python Software Foundation 2019] and allows us to maintain a list containing one CSV-reader for each file. The CSV-readers are iterators returning one line of the file it belongs to at a time, starting with the first line of the file. The ExitStack also closes all opened files when the processing is finished or when an exception is raised.

3.2.2 Data Processing

When all files are loaded the data can be processed. The data flow, how this processing is done, is shown in Figure 3.4. It is divided into three major parts: The data set, the python script hipe_reader.py, and the Control Center Adapter. The data set is the input and the Control Center Adapter is the output for the data, while the hipe_reader.py processes the data. The internal states and transitions in the hipe_reader.py are explained below. The four transitions are the filtering of the data, the filtering for the lowest timestamp, the data sending, and the updating of the lists. This process of filtering and sending out data is repeated in a cycle until all file are processed completely.

Data Filtering

When the file management has provided the list of file readers, the data is filtered as we only process the timestamp, the machine name, and the value of the active power from each measurement. Even though, in the first iteration over all files we read the first row which does not contain a measurement but the column names. As there are different amounts of columns in the files, we determine the column indexes of our desired values for each file and save this information in a dictionary. Afterwards, the next row is read and the values of the timestamp, the machine name, and the energy consumption are read. The timestamp is stored as a string containing date and time. However, the Titan platform needs the timestamp in milliseconds since 01/01/1970. Consequently, we parse and convert it. After this step we created a list containing the earliest timestamp of a record of each file.

Timestamp Filter

In the next step the earliest timestamp in that list is determined and another list is created. This new list contains the measurements taken at this timestamp, as it is possible that two distinct machines were measured at the exact same point in time.

Data Sending

After creating the list with all measurements for one point in time the data is sent to the Control Center Adapter. For this data transfer the measurement is converted according to the UJO scheme, which is used for data transfer between components of the Titan platform. The UJO Data Object Notation is used to serialize data to work with limited resources [WOBE-SYSTEMS GMBH 2016]. After this the UJO map is sent to the Control Center Adapter.

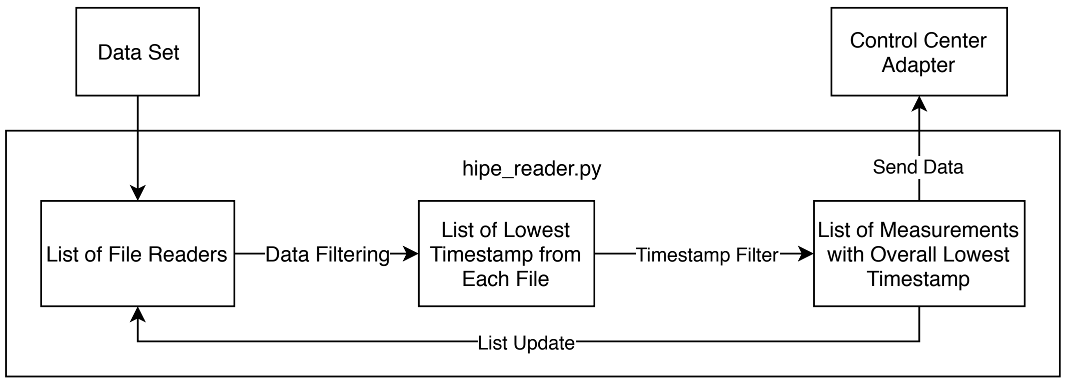

Figure 3.4. Data flow in the Record Bridge

List Updates

The next step is to update the list with the filtered data. Therefore, we fetch a new row from each file whose data got sent out in the step before. These rows are filtered the same way as before and the data replaces the data from the sent out timestamp in the list containing all timestamps.

3.2.3 Transmission Modes

In the step when the data is sent to the Control Center Adapter the four different transmission modes (real time, speed up, instant transmission, and demo mode) can be used. They are handled by one parameter indicating the selected transmission speed. When the instant transmission is selected, the data is sent out as soon as it is processed completely. When real time or speed up transmission is selected we calculate an interval for which the program sleeps before sending out the data. This interval is calculated with the following equation:

$$sleepTime = \frac{timestamp - lastTimestamp + currentTime - lastTime}{speedup}$$

The difference $timestamp - lastTimestamp$ is the time between the timestamp of the last sent measurement and the timestamp of the measurement to be sent. We add $currentTime - lastTime$, with $lastTime$ being the time when the last measurement got sent out, to make the sleeping interval more accurate. Otherwise, the program would sleep for some milliseconds too long, as the processing of the data needs time too, which does not have to be waited for. Even though it is only some milliseconds, these would add up with millions of measurements. The $speedup$ is the factor by which the processing speed should increase. To achieve this we divide the sleep time by the speed up.

The demo mode can be used together with the three other modes. In the demo mode the data processing and the calculation of the sleeping time are the same. However, when a measurement is converted according to the UJO scheme, the timestamp of that measurement is swapped out with the current time.

3.2.4 Configuration

In order to switch between the different transmission modes, the program can be configured via a configuration file in the YAML format. A YAML file contains key-value pairs assigning the selected values to the parameters. These parameters are a path to the data set, the speedup, and a Boolean for the demo mode. If the speedup is set to a number greater than one, the data will be processed faster, whereas setting it to 0 enables the instant transmission mode. The demo mode is enabled when the demo parameter is set to true. Furthermore, there are two parameters, which belong to Kafka, indicating the bootstrap server address and the topic.

3.3 Adjustments for the Flow Engine

Even though the flow engine is not used for the concepts and the implementation, we also provided a version which is adapted for the use with the flow engine. However, it is not functional. In order to adapt the HIPE Reader to the flow engine there are two changes to make which are shown in the following. The first change is to provide a *do_brick_processing* function which calls the *main* function of the HIPE Reader instead of calling the *main* function directly. This allows the flow engine to send a flow packet to the HIPE Reader in order to start its execution. After processing the data, the second change is made at the point when the data is sent to the Control Center Adapter. Instead of calling the *publish_message* function on the instance of the Control Center Adapter, we call *brick.send_output* to send a flow packet without defining the destination where the measurement is sent. In order to deliver the packet correctly we specify the flow in the flow-file. This flow contains the HIPE Reader followed by the Control Center Adapter, so all packets from the HIPE Reader are delivered to the Control Center Adapter. When an approach is used where the HIPE Reader and the Transmission Mode Manager are separated, the Transmission Mode Manager would be placed between the HIPE Reader and the Control Center Adapter in the flow.

Chapter 4

Visually Exploring of the HIPE Data Set

In this chapter we explore the HIPE data set with the visualization of the Titan platform. Some of the following observations can be explained without further knowledge about the data set. In contrast, to explain other observations we would need further insight into the factory. Firstly, we discuss different seasonalities we can observe in the energy consumption, then we discuss the influence of holidays on the energy consumption, and finally we compare the different machines.

4.1 Seasonalities

In the following we have a look at different sections of the HIPE data and discuss three observable seasonalities. These are a weekly, a daily, and a sub-hourly seasonality.

4.1.1 Weekly Seasonality

The first seasonality we discuss is a weekly seasonality. Figure 4.1 shows one month and one week of the HIPE data. The weekends are highlighted with a grey background. On the weekdays the energy consumption peaks vary between 20 kilowatts and 52 kilowatts, whereas on the weekends the maximum values are more homogeneous, with peaks between 13 kilowatts and 18 kilowatts. Independent of the day of the week the energy consumptions

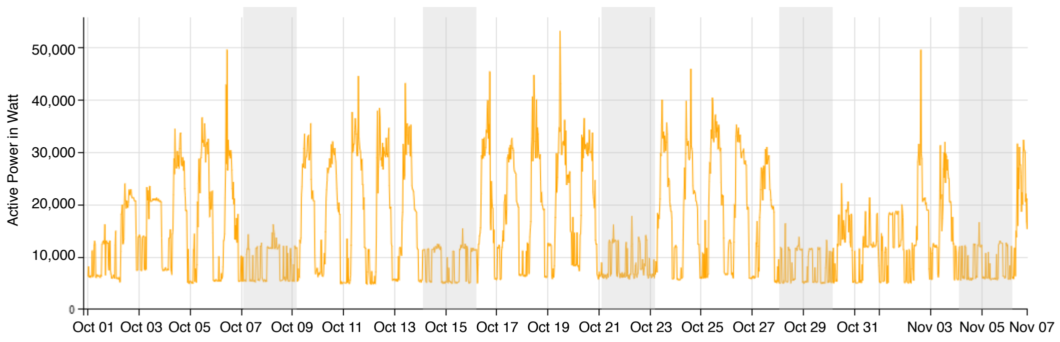

Figure 4.1. Weekly seasonality in the HIPE data

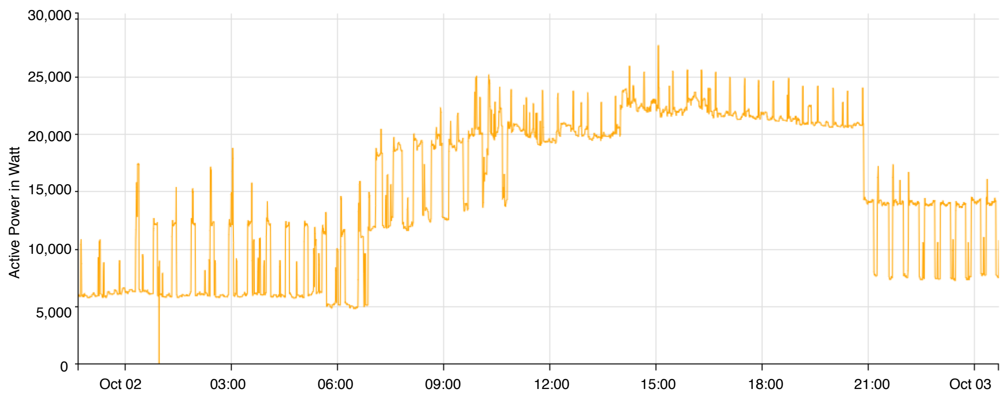

Figure 4.2. Daily seasonality at the main terminal

lowest points are at about 8 kilowatts. One explanation for this value is a base load. Every building has a base load, which is independent of the time, day, or presence of people. This base load can be caused by energy consumers such as cooling or surveillance. The higher energy consumption on weekdays implies a production which is dependent on human control, as otherwise on weekends the energy consumption could be higher as well. Furthermore, this conclusion is supported by the daily seasonality.

4.1.2 Daily Seasonality

Figure 4.2 shows one day of the energy consumption of the main terminal. It displays a difference between the interval of 07:00 am to 09:00 pm and the interval of 09:00 pm to 07:00 am. This pattern can be seen on every day from Monday to Friday. The difference in the energy consumption can be explained with the working hours of the employees and therefore supports the conclusion of the production being dependent on human supervision. Notable is that even during night time the energy consumption ranges from 6 kilowatts to 12 kilowatts, with peaks up to about 18 kilowatts.

4.1.3 Sub-Hourly Seasonality

Figure 4.3 shows the data of one weekend of the main terminal and a zoomed in view of a 7 hour interval. In this zoomed in view we can observe a pattern, marked with blue lines, which occurs over the whole weekend and other weekends as well. The energy consumption rises up about 6 kilowatts, stays at that level for 9 minutes and goes down 6 kilowatts again. This pattern repeats itself every 29 minutes. For explaining the reason of this regular pattern we need further information about the machines that are part of the main terminal.

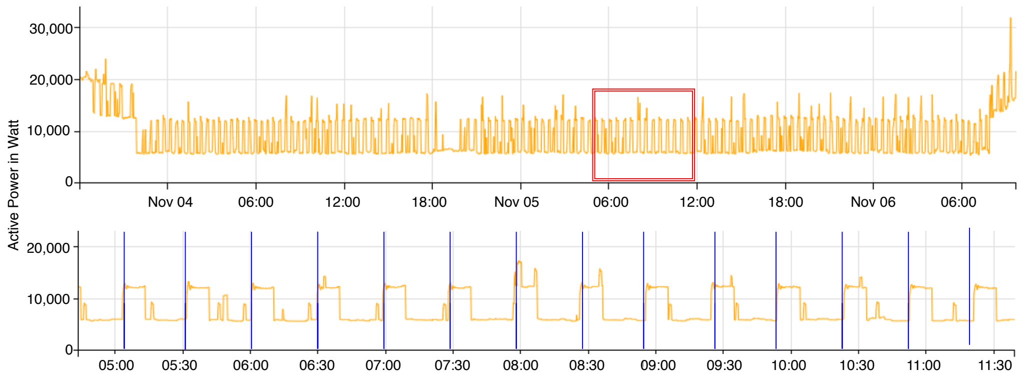

Figure 4.3. Sub-hourly seasonality at the main terminal

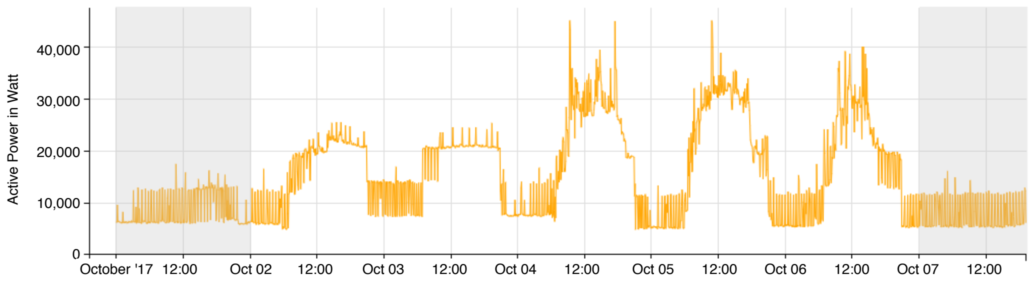

Figure 4.4. Data with a holiday and a bridging day

4.2 Holidays

In the following we analyze the energy consumption on holidays and bridging days. As an example, we look at October the 3rd 2017, which is the Day of German Unity and therefore a public holiday in Germany. In Figure 4.4 it is observable that on October the 3rd the energy consumption is on a level not as high as on a workday but higher than on the weekend which is highlighted in grey. The reason for this is that on a holiday noticeably less people work in the offices, thus the energy consumption is lower. Remarkable is the energy consumption on October the 2nd. It is slightly higher than on the holiday the day after and way lower than on normal weekdays. As October the 3rd was a Tuesday, October the 2nd was a bridging day. Due to this reason many people take an extra day off, which results in an energy consumption similar to a holiday, as the production seems to be dependent on humans, as we concluded in Section 4.1. Based on the fact that on both, holiday and bridging day, the energy consumptions still follows the patterns of the seasonalities, we can make assumptions to explain this. One is that despite the day being a holiday people work in the factory. Another explanation would be machines which start automatically, even without people being present.

4.3 Comparison of All Machines

Figure 4.5, Figure 4.6, and Figure 4.7 combined show the energy consumption of all machines on two days. There are three figures to visualize all machines, as the scale is different in all figures, ranging up to 60 kilowatts in Figure 4.5, 8 kilowatts in Figure 4.6,

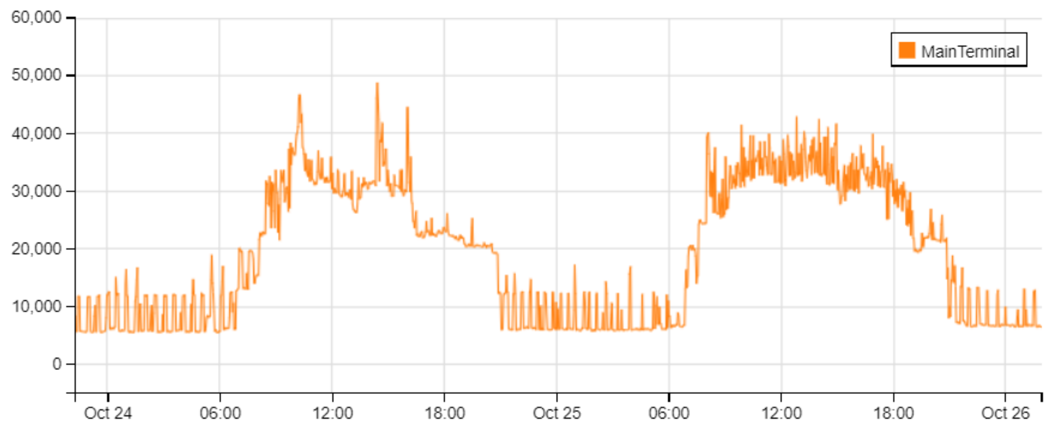

Figure 4.5. 2 days of the main terminal

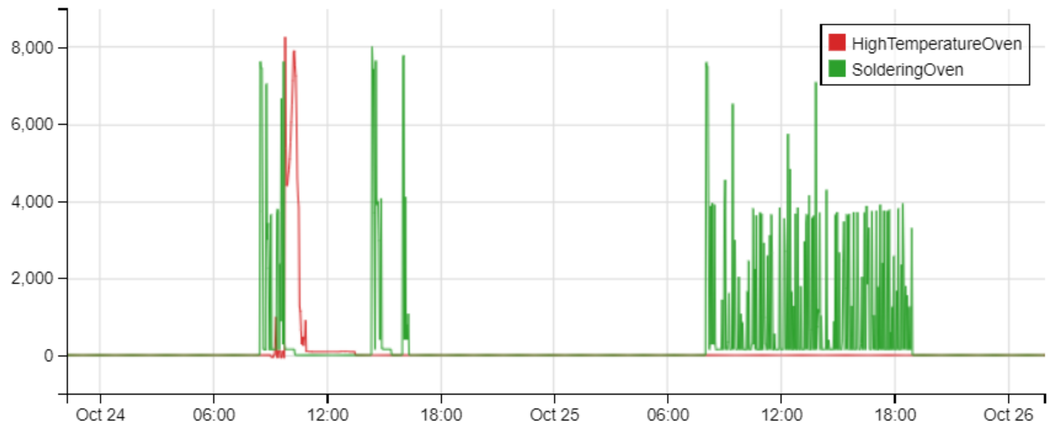

Figure 4.6. 2 days of the high temperature oven and the soldering oven

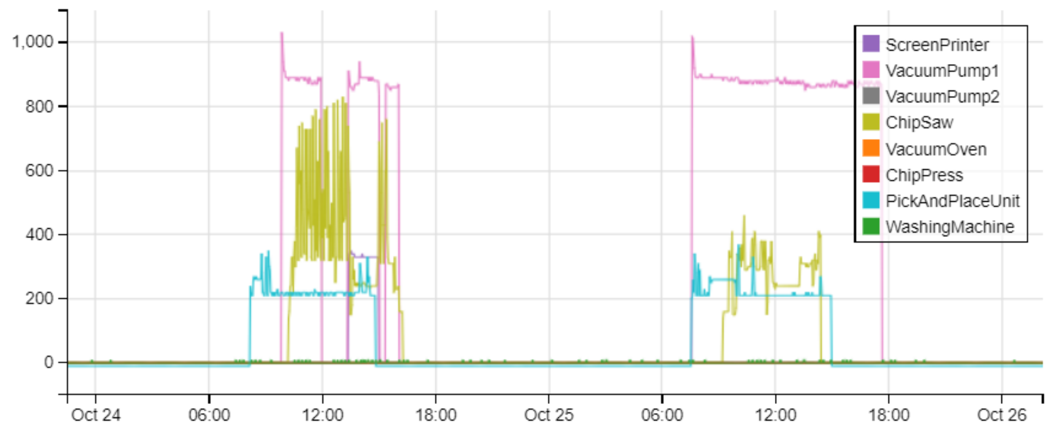

Figure 4.7. 2 days of the rest of the machines

and 1 kilowatt in Figure 4.7. This shows the huge differences in energy consumption of the machines. The overall energy consumption is dominated by the main terminal. However, the other machines are interesting as well, as we can use the following findings in Chapter 5.

Figure 4.5 shows the seasonalities we discussed before on the main terminal, which are observable through the whole three months. The machines in Figure 4.6 and Figure 4.7 behave differently. They have in common that they are used between 09:00 am and 07:00 pm and thus have a smaller time of usage as the main terminal. However, when we explore them in more detail we see large differences in their energy consumption patterns. For example, the high temperature oven in Figure 4.6 is used on one day for a short time and not used on the other day. In contrast to that the soldering oven is used on both days, but not in a regular pattern like the main terminal. The machines in Figure 4.7 show these variations as well. Vacuum pump 1 and the pick and place unit have a quite regular pattern, vacuum pump 2 and the chip saw are used unregularly, and the vacuum oven, the chippress and the washing machine were not used at all during these two days.

The presented findings are representable for larger timescales in the data, but can be observed in more detail when zoomed in this far.

Chapter 5

Prototyping a Forecasting for Energy Consumption

After exploring the data set visually we can use the findings as a base to not only get an insight in the history, but to predict the future of the energy consumption. This chapter contains information about concepts on how to forecast energy consumption. Furthermore, we discuss two prototypes for these concepts, one with Prophet and the other one with TensorFlow. These two prototypes have a fundamental difference in their concepts, which is the reason why both are presented. While one of the prototypes takes the data of one machine at a time and forecasts that machine, the other prototype can take the whole data set and thus can identify relations between different machines. Nevertheless, both of the prototypes have their advantages and disadvantages over the other.

5.1 Concept

The concept of the forecasting is split into two parts. Firstly, we have the concept on how to forecast energy consumption in general and secondly, we fit this concept in the context of the Titan platform.

5.1.1 General Concept

Firstly, the data has to be loaded into the forecasting and then be preprocessed to fit the requirements of the chosen technology. This preprocessing includes a possible resampling of the data and converting it into the required data structure. Resampling means that values are aggregated to have even spaced timestamps. The resampling can be done for two reasons. The first reason is to reduce the number of data points in order to increase computation speed. The disadvantage of this is a loss of accuracy. The second reason to resample the data can be a requirement of the forecasting technology. This is discussed in more detail in Section 5.3. In addition to the preprocessing, there is the need to specify how much time in the future the data shall be forecasted. This depends on the reasons the forecasting is implemented. In order to gain an overview this period can be longer so trends in the data are shown. For higher accuracy a shorter period is more suitable. Depending on the chosen technology the next parameter which needs to be set is the time looked into the past to forecast one value. This parameter depends on the data itself. When the data shows long time trends, a longer period would probably benefit the accuracy, whereas a shorter period suits better when the trends are not seen for long but only for short periods.

5.1.2 Fitting the Concept into the Titan Platform

In this concept the forecasting is another type of microservice next to the Record Bridges, the history, the statistics, and the configuration as seen in Figure 5.1. They communicate to the other microservices via Kafka. In the beginning this is used to get all the recorded data from the history and later to get the data coming in live from the Record Bridges. For this the forecasting has to subscribe to the topic in which the Record Bridges send the data as a new consumer group. The forecasting is built out of its logic and its own database. The database is needed to ensure the independency from the history in order to function as a microservice working on its own. This database stores the preprocessed data as well as the forecasted values. The forecasted data is also sent to the visualization to be displayed in the Control Center. Additionally, one deployment of the Titan platform has the possibility to feature more than one forecasting implementation. This is due to the reason that different technologies in this area have their advantages and disadvantages. For that reason two

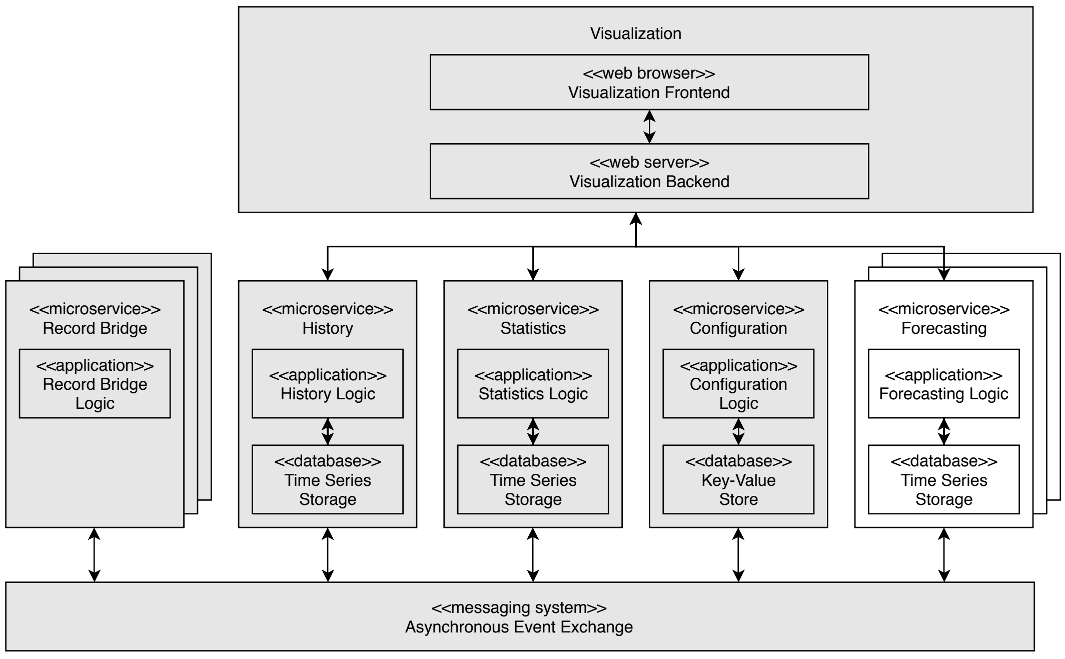
Visualization
<<web browser>> Visualization Frontend
<<web server>> Visualization Backend
<<microservice>> Record Bridge
<<application>> Record Bridge Logic
<<microservice>> History
<<application>> History Logic
<<database>> Time Series Storage
<<microservice>> Statistics
<<application>> Statistics Logic
<<database>> Time Series Storage
<<microservice>> Configuration
<<application>> Configuration Logic
<<database>> Key-Value Store
<<microservice>> Forecasting
<<application>> Forecasting Logic
<<database>> Time Series Storage
<<messaging system>> Asynchronous Event Exchange

Figure 5.1. Titan architecture with forecasting

prototypes with very different foundations are provided.

As the forecasting shall function as a prototype for future implementations there were some simplifications made. One simplification is that the data is not requested from the history but read in from the data set. This is due to the reason that for testing purposes reading in the data with the Record Bridge and then requesting it from the history would cost to much time. The next simplification regards the storing of the data. For this prototype the data is not sent to the visualization but visualized as a graph by Prophet and then saved as a picture.

5.2 Prototype with Prophet

With Prophet we can process the data of one machine at a time. The input to make this forecast is the data which was recorded on that machine.

5.2.1 Preprocessing the Data

Prophet needs the data in a shape which is different to the HIPE data set and to the way the Titan platform stores the data. Therefore, we need to preprocess the data. In the following the concept and the implementation of that are presented.

Concept

For preprocessing the data there are several steps to take. Table 5.1 shows the data structure, as it is filtered out of the HIPE data set, in the top table, whereas in the bottom table the shape of the data after the preprocessing is presented. Firstly, the column containing the machine name is dropped, but we have to make sure the machine name is saved in order to attach it onto the forecast later. The next step is to cut off the milliseconds and the time zone information of the timestamps as Prophets smallest intervals are seconds. An optional step is the resampling of the data.

Table 5.1. Conversion of the HIPE data to the format needed by Prophet

Timestamp	Machine	Value
2017-10-23T00:00:03.727+02	MainTerminal	12.75
2017-10-23T00:00:09.120+02	MainTerminal	12.78

↓

ds	y
2017-10-23 00:00:03	12.75
2017-10-23 00:00:09	12.78

Implementation

As the preprocessing of the data takes some time, which is impractical when we want to test multiple models, the preprocessing is split into two different components: The filtering of the data, in the way Table 5.1 shows and the resampling of the data.

In the filtering the following steps are taken for each file of the data set. The first step is to read the file. However, while reading the file only the columns containing the machine name, the timestamp of the measurement and the value of the energy consumption are stored in a DataFrame, which is provided by the python library pandas [McKinney 2010]. The other columns are dropped. The DataFrame is a multidimensional list which contains one list for each of the three columns we use from the data set. These three lists contain the values of the corresponding column. After creating this DataFrame, the machine name is stored in a variable. To get the machine name the DataFrame can be accessed at the machine column at index 0, considering that at each index the machine name is stored. Afterwards, the machine column is dropped since Prophet does not need it. The next step is reducing the accuracy of the timestamps from microseconds to seconds. To achieve this the microseconds are cut off, as the difference of a value being recorded in one specific second or in the next one has no remarkable effect on the outcome of the forecasting. Subsequently, renaming the columns is necessary. The timestamp column is renamed to *ds* and the column containing the energy consumption becomes y. This is due to the reason that Prophet needs the columns with these names. Finally, the DataFrame is saved in a CSV-file which is named after the machine name. This CSV-file can then be opened and the data can be stored in a new DataFrame without filtering the data every time a new forecasting is calculated.

To speed up the process of creating different forecasts even more, the resampling to different resolutions can be saved as well. To achieve this we define a list which contains the different resolution we want the data to be resampled to. The pandas function resample is then called on the DataFrame and takes one of the resolutions as an argument. Furthermore, we have to choose between different functions, provided by pandas, as a metric how the data is resampled. We choose to resample the data by the mean. After resampling, the DataFrame is saved in a CSV-file. This process is repeated for every resolution in the list.

5.2.2 Creating a Forecast

As stated in Chapter 2, Prophet uses an analyst-in-the-loop approach. Consequently, we set parameters for the model based on our observations from Chapter 4. Due to this we choose to forecast the weekly seasonality and the daily seasonality. A yearly seasonality can not be detected in the HIPE data, due to the fact that the data was not recorded for a sufficient time. Considering that the data set was recorded in Germany we added the German holidays. The results of the forecast are discussed in Chapter 6.

5.3 Prototype with TensorFlow

As TensorFlow is an interface for machine learning, the goal is to implement a prototype for a neural network which takes the data of all machines as an input and forecasts the energy consumption of one machine. In this chapter we discuss one possible way of working with the data to create a neural network. Firstly, we discuss the preprocessing and then the creation of a training and a validation set for the neural network. The neural network is not presented, due to the fact that we could gather no results. This issue is evaluated in Chapter 6.

5.3.1 Preprocessing the Data

In the following we discuss one way of preprocessing the data for TensorFlow. This preprocessing is different from the preprocessing for Prophet. In the preprocessing the shape of the data is transformed and afterwards, the data is normalized. As this implementation is a prototype for future implementations we do not implement different ways of preprocessing the data, even though this can lead to different results.

Transforming the Shape of the Data

TensorFlow needs the data in a different structure than Prophet. The main difference between the data structure for Prophet and the one for TensorFlow is that TensorFlow takes the data of all machines at the same time. Table 5.5 shows the final shape after transforming the data from the individual files to one DataFrame containing the whole data set. For reference we call this the main DataFrame. The vertical dots represent columns containing the energy consumption of the other machines.

The first step to achieve this shape of the data is the filtering. The filtering of the data is done in a similar way as in the preprocessing for Prophet. However, the difference is that the timestamp column keeps its name and the column containing the energy consumption values is named after the corresponding machine name, as Table 5.2 shows.

Table 5.2. Structure of the data of one machine in the HIPE data set

Timestamp	Chip Press
2017-10-23T00:00:03.727+02	12.75
2017-10-23T00:00:09.120+02	6.37

Table 5.3. Structure of the main DataFrame after joining all machines

Timestamp	Chip Press	Chip Saw	...	Washing Machine
1508709603	12.75	2.53	...	21.94
1508709609	6.37	25.56	...	58.00

Table 5.4. Main DataFrame with the future column attached

Timestamp	Chip Press	Chip Saw	...	Washing Machine	future
1508709603	12.75	2.53	...	21.94	6.37
1508709609	6.37	25.56	...	58.00	

Table 5.5. DataFrame after normalization

Timestamp	Chip Press	Chip Saw	...	Washing Machine	future
1508709603	0.234245	0.0446	...	0.345	0.1
1508709609	0.1254	0.4144	...	1.0	

The second step of shaping the data is to resample the data. For TensorFlow this is a requirement, due to the fact that the measurements of all machines need a joint index. The resampling is done the same way as it is done for Prophet. There are different ways to combine resampling the data and creating the DataFrame. In the implementation for this forecasting the data from one machine is resampled at a time. After resampling the data is added to the main DataFrame. To do this we use the pandas function *join*. When two DataFrames are joined they need a common index column. This state is ensured by the resampling. When the data is joined with the main DataFrame, the main DataFrame is extended by the column containing the energy consumption values. When both of these steps are completed for every machine, the DataFrame shown in Table 5.3 is created. It contains the resampled energy consumption of every machine.

The last step is to add the future column which is shown in Table 5.4. The future column holds the information which value should be forecasted based on the other entries in that row. For creating this column we specify the machine which is getting forecasted and a number of steps. This number of steps multiplied with the resolution is the amount of time which will be forecasted. Afterwards, the column containing the energy consumption values for the specified machine is copied to the future column. Then the future column is shifted by minus the number of steps. For instance, if the data has a resolution of 10 minutes and we decide to forecast 1 hour, the entry on index 6 will be shifted to index 0.

Normalization of the Data

After shaping the data, the next step of preprocessing is to normalize the data. For this we need the maximum value for energy consumption in the whole data set. In order to normalize the data we divide every value by the maximum value. This results in every entry being in the interval $[0, 1]$. To have a better performance the future column is classified to values with a resolution of 0.1. The goal of this is to reduce the number of different forecasted values to 10. This is done to reduce the number of nodes in the neural network and therefore to have a faster processing. When this is completed, the DataFrame shown in Table 5.5 is completed.

5.3.2 Creating the Training and the Validation Set

In order to enable TensorFlow to create the neural network and to test its created model we provide two DataFrames. One DataFrame contains the data for training and the other DataFrame the data for validation. Both the training and the validation set are a subset of the main DataFrame. The training set contains 95% of the data, whereas the validation set contains the other 5%. It is important that the intersection of both sets is empty as otherwise the neural network would memorize the values to predict.

The next step to prepare the data for TensorFlow is to specify the length of the history which is used to predict one value. With this value we create sequences. Each sequence contains pairs, whose first component is the data of the history the prediction is based on and the second component are the values which should be predicted. For example, if we forecast the data with a resolution of 1 hour based on a 1 day history the first sequence would contain the data of 2017-01-10-00:00:00, 2017-01-10-01:00:00, ..., 2017-01-10-23:00:00 and the second sequence would start at 2017-01-10-01:00:00 and range to 2017-02-10-00:00:00. These pairs are used as the input for the neural network.

Chapter 6

Evaluation

In this chapter we evaluate the concepts and realizations presented in Chapter 3, Chapter 4, and Chapter 5.

6.1 Data Processing and Exploration

In the following we evaluate the implementation of the HIPE Reader and the visual exploration of the data. For the HIPE Reader we analyze the performance of the implementation. The evaluation of the data exploration focuses on aspects which are related to the Titan platform. These aspects are features of the Titan platform which can and can not be used in the context of this data set.

6.1.1 Performance of the Instant Transmission Mode

The performance of the HIPE Reader was measured on a laptop with an Intel Core i5-6200U 2.3GHz with Turbo Boost up to 2.8GHz and 8 GB DDR4 memory. For this evaluation we define performance as sent measurements per second. The instant transmission mode is the only mode of the four modes presented in Section 3.2.3 which has the goal to send the measurements as fast as possible. Therefore, it is the only mode where the performance is evaluated. However, with the number of measurements sent per second we can calculate the maximum possible speed up for the speed up mode.

The test measures the time from the first sent record to the 1000th sent record. Then we divide 1000 by the amount of time needed and get the sent records per second as the result. To get more reliable results we run the performance test three times. This is for minimizing the influence of the utilization of the machine by other programs. Furthermore, the test is run twice, with three cycles each, as we make a change to the Kafka connection in between the two tests. We do not make this change permanent taking into consideration that it could cause problems when set up with the flow engine. This is due to the reason that the change would not be part of the HIPE Reader, but be a part of the Control Center Adapter. Therefore, this change has an influence on other bricks as well and we can not ensure that this would not cause problems with these other bricks. The change is to remove a call of the Kafka function *flush*. The function flush ensures that every message is delivered, thus calling it after every sent message makes the communication synchronous, which impairs

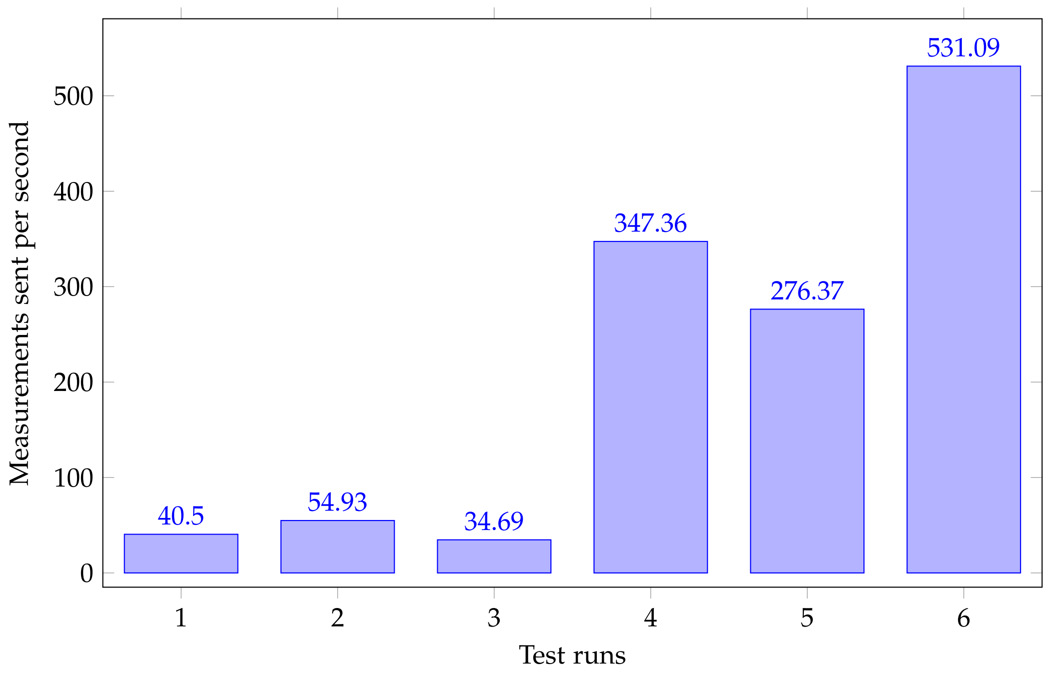

Figure 6.1. Performance of the HIPE Reader in different setups

the performance for the instant transmission. However, when not flushing we do not have the guarantee that the messages are delivered.

Figure 6.1 shows the performance of each test run. They can be divided into two categories. The test runs 1, 2, and 3 where made with the flushing enabled, whereas the test runs 4, 5, and 6 had flushing disabled. With the flushing enabled we reached an average of 43.37 sent records per second whereas without the flushing the HIPE Reader had a 8.87 times better performance, with an average of 384.94 records sent per second. This would result in an overall processing time of 4.4 days with the flushing and 11.9 hours without the flushing. Furthermore, Figure 6.1 shows huge differences within the categories, for example between test run 5 and 6. These would probably balance each other out when the whole data set is processed, but can potentially cause a difference of ±50% in the overall run time.

6.1.2 Features of the Titan Platform

Considering that the Titan platform is currently not optimized for historical data, but for live installation, there were some problems. One problem was the amount of data. The amount of data was too big to be visualized as the http request timed out after 4.5 minutes.

This led to the biggest interval we could visualize, which is presented in Figure 4.1, with an interval of 1 month and 1 week.

One feature of the Titan platform we can not use were the indicators for the trend change. These trend changes are based on the data coming in over the last hour, the last day, and the last week. As the data is from 2017, there is no data from these intervals. However, in the live demo the indicators show the trend correctly as the incoming measurements have the timestamp of when the system is running. The visualization of the incoming data worked in all four transmission modes and one can observe the data coming in such as it would from a real production environment.

6.2 Forecasting

In order to evaluate the forecasting we will first discuss both technologies separately. For Prophet we analyze the figures, which it provides, visually and calculate the precision of the forecast with the mean squared error. Due to the fact that the forecasting with TensorFlow did not deliver results, we discuss possible reasons for that problem.

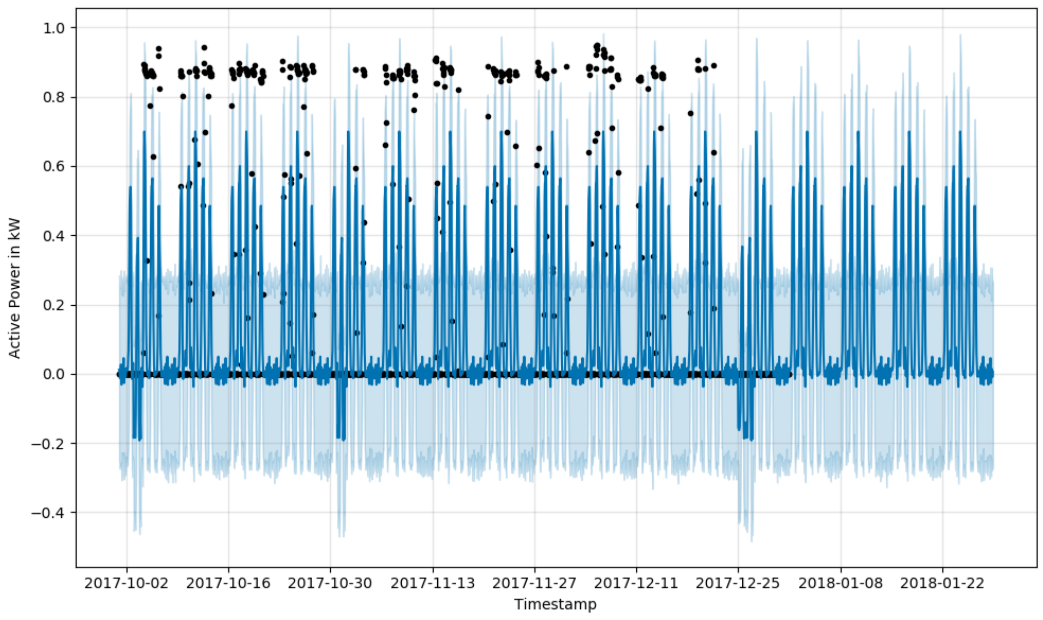

Figure 6.2. Forecasting for vacuum pump 1 with a 1 hour resolution

6.2.1 Prophet

We used Prophet to quantify the observations from Chapter 4. The reason for this is the fact that the model is not optimizing itself but is optimized by the analyst instead. Consequently, the analyst specifies the patterns Prophet has to find. Figure 6.2 shows the forecast of vacuum pump 1 and Figure 6.3 presents the forecast of the chip press. In all forecasts the black dots are the data on which the model was fitted, the blue line is the forecasting and the light blue area is the uncertainty interval. On the x-axis the timestamp is plotted and on the y-axis the energy consumption in kilowatts is plotted. The forecasting for vacuum pump 1 is an example for a time series, where we were able to mimic the findings from Chapter 4, whereas the vacuum oven is a counter example. The findings from this section can as well be seen in the figures for the other machines, which are displayed in the appendix. For vacuum pump 1 the data was resampled to a resolution of one value per hour. Figure 6.2 shows the weekly and daily seasonality as well as the influence of the holidays on 3rd October, 31st October, and the Christmas days. Even though this forecasting is reasonably accurate it has three main problems.

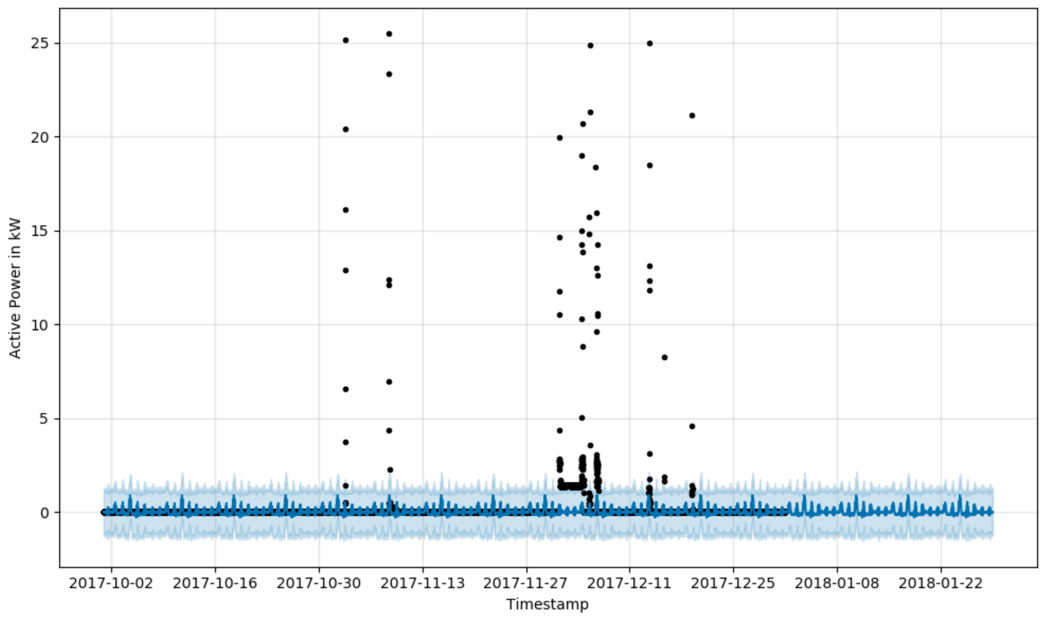

Figure 6.3. Forecasting for the chip press with a 10 minute resolution

Problems with the Forecasting

The first problem are values below zero which do not occur in the data set. These values below zero are seen on the holidays, October the 3rd, October the 31st, and the Christmas days, in Figure 6.2. To predict values lower than on other weekdays is correct, however, as seen in Chapter 4 the energy consumption is not lower than zero. This problem is also a reason for the resampling of the data. In Figure A.7 the forecast of vacuum pump 1 with a resolution of 10 minutes is shown. It can be seen that the model goes down to about -100 watts whenever the actual data is at 0. The second problem are periods when the machine is used. Figure 6.4 shows a 12 hour section of Figure 6.2. The section is part of a regular working day. From 11:20 am to 04:50 pm the energy consumption is varying between 815 watts and 900 watts. However, the forecast ranges up to 410 watts as its maximum. This problem is seen in the figures in the appendix as well. This difference between the forecast and the real energy consumption occurred in all different configurations we tried for Prophet and thus we were not able to achieve a better precision.

The third problem is the overall trend in the data shown in Figure 6.5. At the end the trend decreases which results in forecasts lower than the standby value of the main terminal. This may be caused by the fact that the last week of the data is around Christmas and new year. Not all days of that week are holidays but the energy consumption is lower than at normal workdays. This results in the model calculating it into the overall trend,

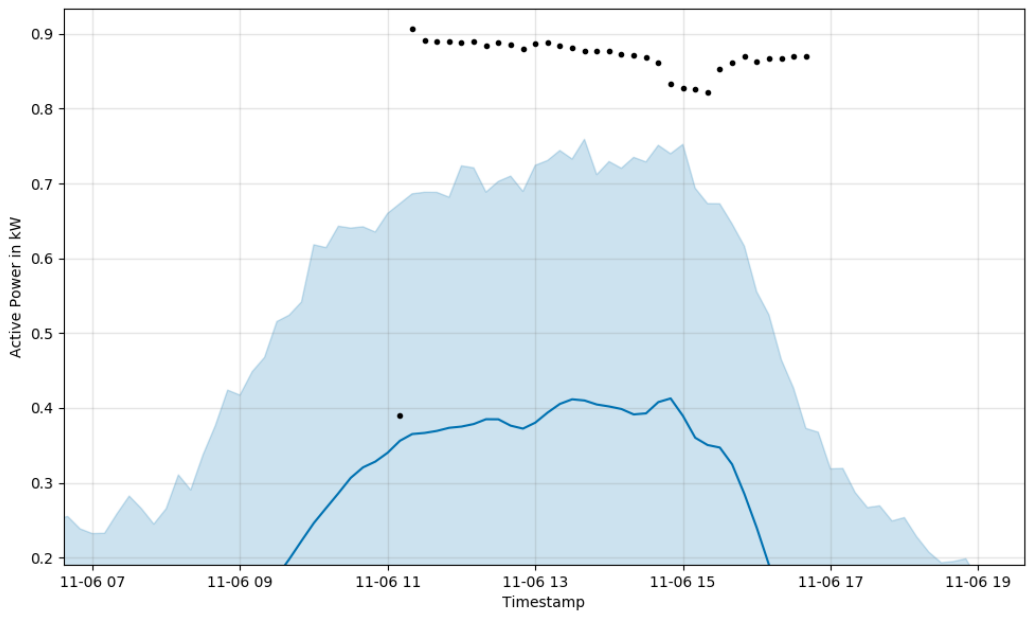

Figure 6.4. 12 hour section of the forecasting for vacuum pump 1 with a 10 minute resolution

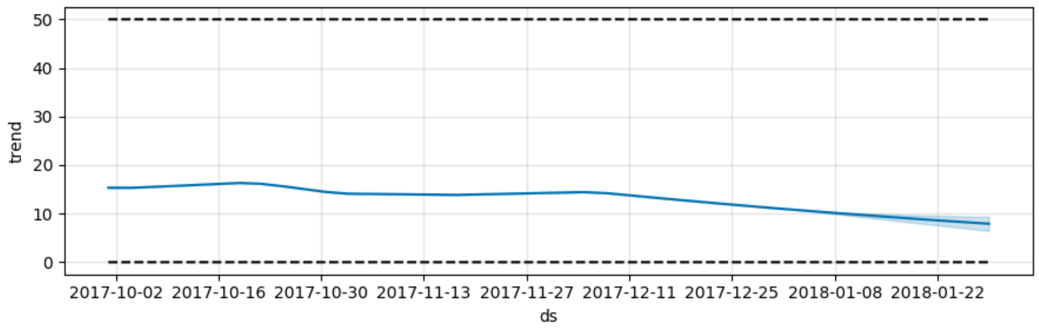

Figure 6.5. Trend of the main terminal

bringing it down further in the future. This issue could also have an influence on the overall prediction on other places. This is due to the bridging days mentioned in Chapter 4. To handle this problem one could manually add the bridging days as holidays. As these manually added holidays are no real holidays it could lead to another problem. On the bridging days not everybody takes a day off so there is still production going on, which results in an energy not as low as on the weekend but not as high as on a normal workday.

Quantifying the Evaluation

To measure the accuracy of the forecast we use the *mean squared error* [Harville and Jeske 1992]. This measurement can be a reference for future implementations of a forecasting. To calculate the mean squared error we separate the data into two parts. The first part is used as the input for Prophet to forecast the interval of the second part. This second part, which we chose to be the last 4 weeks, is used as a comparison to the forecast. Figure 6.6 shows the interval as it was recorded and as it is predicted using the rest of the data. The blue graph represents the recorded data and the orange graph visualizes the forecast. The mean squared error is then calculated between these time series. This implementation of a forecasting with Prophet reached a mean squared error of 0.05264. It has to be tested in comparison to other implementations how good this value is.

6.2.2 TensorFlow

In contrast to the implementation with Prophet, the implementation with TensorFlow did not deliver results. This has two possible reasons: The preprocessing and the architecture of the neural network. In the preprocessing there are multiple points which can lead to different results. These are the interval used for the resampling, the time looked in the past to predict a value, the amount of time how long the predicted values are in the future and classification of the predicted values. Considering that the goal for the implementation was

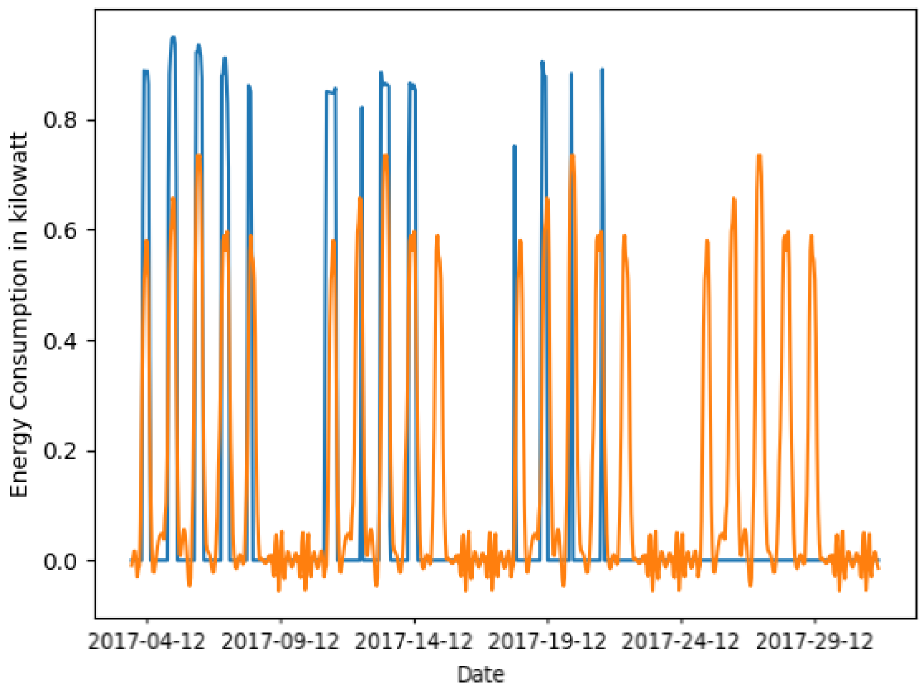

Figure 6.6. 4 weeks of the data of vacuum pump 1 and the same interval predicted from the rest of the data

to create a prototype which can function as a base for future implementations, we decided not to optimize this implementation, but to implement the forecasting with Prophet.

6.3 Threats of Validity

There are three threats for the validity of the performance of the HIPE Reader. The first is the measurement of the performance as we edited the code for the HIPE Reader. This has an influence on the performance, even though it is only incrementing a variable by 1 and comparing if that variable is below 1000, whenever a measurement is sent out. The second threat to validity is the usage of the host machine. For minimizing this effect we ran the test 3 times on each machine, however, with a machine not used by other programs the performance could be better. The third threat to validity is the change to the Control Center Adapter. We traded the reliability that a message is delivered for a performance 8.87 times better than with the flushing. It has to be tested if due to this change other effects, such as losing messages, come up.

The threat to the validity of the forecasting is the fact that there are different ways of preprocessing the data, especially for the intervals of the resampling. The problem is that we do not work with the original data set anymore and therefore, different results can be achieved with different ways of preprocessing the data. This applies for both prototypes but especially for TensorFlow as the preprocessing probably is a main reason why the prototype did not deliver any results.

Chapter 7

Related Work

In Chapter 2 we stated that TensorFlow is an interface for machine learning algorithms, especially for deep neural networks. However, we did not elaborate how the neural networks were built in Chapter 5. This is due to the fact that our neural networks did not work properly and to elaborate all possible strategies to build such a neural network is a too broad topic for this thesis. Zhang et al. [2001] simulated a computer experiment for time series forecasting with neural networks and analyzed how a neural network can be built for this task.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

In Section 1.2 we defined four goals for this bachelor's thesis, which had the common target to explore an energy status data set from an industrial production environment. In order to fulfill these goals we implemented the HIPE Reader, which processes the HIPE data set in a way that the Titan platform can utilize it. This data processing contains four different modes which serve different purposes, such as using the HIPE Reader as a tool for live demonstrations or to process the data as fast as possible. Goal G2 targeted the visual exploration with graphs and other visualizations. This visual exploration was done with visualizations from the Titan platform. We observed some characteristics in the data, such as seasonalities. These observations then were transferred into the realization of goal G3, the prototyping of an energy consumption forecasting. We designed two prototypes, as well as a more general concept, on how to bring a forecasting into the Titan platform. The two prototypes were based on different technologies, which can provide different outcomes in the forecasting.

Overall, we matched our expectations regarding the different goals. With the HIPE Reader we made the different transmission modes work and were able to visualize their outcome with the Titan platforms visualization. Furthermore, we were able to observe patterns in the data, which revealed information about the come about of the energy consumption. Moreover, the information assisted us in the configurations for the forecasting. However, the results of the forecasting were not accurate. Nonetheless, this is not a huge problem as the intent was to create prototypes, which do not have to function perfectly, but rather serve as a foundation for future implementations.

8.2 Future Work

In the following approaches for future work, which are based on the conclusions, are presented.

8.2.1 Optimization of the Titan Platform for Historical Data

As stated in Chapter 6 the Titan platform is currently not optimized for historical data such as the HIPE data set. To optimize this we suggest changes of the connection from the Record Bridges to the Control Center. The change we used in the performance test could work as a starting point for development in that direction. Another part of the platform which could be optimized is the visualization. The visualization had problems with the amount of data, as we discussed in Chapter 4. To solve this problem the data could be requested dynamically and more aggregated to the time interval which the user selects.

8.2.2 Improvement of the HIPE Reader

The HIPE Reader, presented in Chapter 3, could be modified for a better performance. Especially, the Instant Transmission Mode qualifies for an optimization, considering that its purpose is a fast performance. Furthermore, the live demo mode could be altered to produce a more realistic outcome.

Performance Optimization of the Instant Transmission Mode

We did not optimize the performance of the instant transmission mode, due to the performance currently being limited by the connection to the Control Center Adapter. If that problem is solved, an increase in the performance of the instant transmission mode will be achieved and it will enable further optimizations. One way to further optimize the performance is to implement the instant transmission mode separated from the other modes. This works due to the fact that the instant transmission mode does not rely on the data being sent in chronological order.

Live Demo Mode Optimization

As well as the instant transmission mode, the live demo mode could be the target of optimizations. Currently, there is no relation between the time when a measurement was made and the timestamp it gets in this mode. To make the outcome of this mode more like data coming from a live production environment, the timestamp could be changed in a way which correlates with the real timestamp. This could be done on different levels. These levels could be a daily basis, with mapping a 12 am timestamp in the data set to a 12 am timestamp in the live demo, or a weekly basis, synchronizing the days of the week. For other data sets this could even be done by synchronizing full years, but the HIPE data set has a too small interval for this.

8.2.3 Forecasting

For the forecasting a future goal is to connect it to the Titan platform as presented in Section 5.1.2. This is independent from the actual technology used for future forecasting implementations. For a future implementation another approach which utilizes the relations between the machines would be reasonable. This would most likely lead to results different from the Prophet prototypes results.

Bibliography

[Apache Software Foundation 2017] Apache Software Foundation. *Kafka*. Accessed: 2019-08-16. 2017. URL: `https://kafka.apache.org`. (Cited on page 7)

[Bass et al. 2015] L. Bass, I. Weber, and L. Zhu. *Devops: a software architect's perspective*. Addison-Wesley Professional, 2015. (Cited on page 5)

[Bischof et al. 2018] S. Bischof, H. Trittenbach, M. Vollmer, D. Werle, T. Blank, and K. Böhm. Hipe: an energy-status-data set from industrial production. In: *Proceedings of the Ninth International Conference on Future Energy Systems*. e-Energy '18. Karlsruhe, Germany: ACM, 2018, pages 599–603. (Cited on pages 2, 3, 8)

[Eugster et al. 2003] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM computing surveys (CSUR)* 35.2 (2003), pages 114–131. (Cited on page 7)

[Harville and Jeske 1992] D. A. Harville and D. R. Jeske. Mean squared error of estimation or prediction under a general linear model. *Journal of the American Statistical Association* 87.419 (1992), pages 724–731. (Cited on page 38)

[Hasselbring 2016] W. Hasselbring. Microservices for scalability: keynote talk abstract. In: *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*. ICPE '16. Delft, The Netherlands: ACM, 2016, pages 133–134. (Cited on page 5)

[Hasselbring et al. 2019] W. Hasselbring, S. Henning, B. Latte, A. Möbius, T. Richter, and M. Schalk Stefan und Wojcieszak. Industrial devops. In: *IEEE International Conference on Software Architecture Workshops (ICSAW)*. Hamburg, Germany, 2019. (Cited on pages 5, 8)

[Henning et al. 2019a] S. Henning, W. Hasselbring, and A. Möbius. A scalable architecture for power consumption monitoring in industrial production environments. In: *2019 IEEE International Conference on Fog Computing (ICFC)*. June 2019, pages 124–133. (Cited on pages 1, 6, 7)

[Henning et al. 2019b] S. Henning, W. Hasselbring, H. Burmeister, A. Möbius, and M. Wojcieszak. Goals and measures for analyzing electricity consumption in manufacturing enterprises. In: 2019. (Cited on page 1)

[*Energy management systems – Requirements with guidance for use*]. *Energy management systems – Requirements with guidance for use*. Standard. Geneva, CH: International Organization for Standardization, Aug. 2018. (Cited on page 1)

[Martin Abadi et al. 2015] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: large-scale machine learning on heterogeneous systems*. 2015. URL: `https://www.tensorflow.org/`. (Cited on page 8)

[McKinney 2010] W. McKinney. Data structures for statistical computing in python. In: *Proceedings of the 9th Python in Science Conference*. Edited by S. van der Walt and J. Millman. 2010, pages 51–56. (Cited on page 28)

[Morrison 2010] J. P. Morrison. *Flow-based programming: a new approach to application development*. CreateSpace, 2010. (Cited on page 8)

[Python Software Foundation 2019] Python Software Foundation. *Contextlib — utilities for with-statement contexts*. [Online; accessed 11-July-2019]. 2019. URL: `%5Curl%7Bhttps://docs.python.org/3/library/contextlib.html%7D`. (Cited on page 14)

[Taylor and Letham 2017] S. J. Taylor and B. Letham. Forecasting at scale. *PeerJ Preprints* 5 (Sept. 2017), e3190v2. (Cited on page 9)

[WOBE-SYSTEMS GMBH 2016] WOBE-SYSTEMS GMBH. *Ujo data object notation*. [Online; accessed 22-July-2019]. 2016. URL: `%5Curl%7Bhttps://www.libujo.org/%7D`. (Cited on page 15)

[Zhang et al. 2001] G. P. Zhang, B. E. Patuwo, and M. Y. Hu. A simulation study of artificial neural networks for nonlinear time-series forecasting. *Computers & Operations Research* 28.4 (2001), pages 381–396. (Cited on page 41)

Appendix A

Appendix

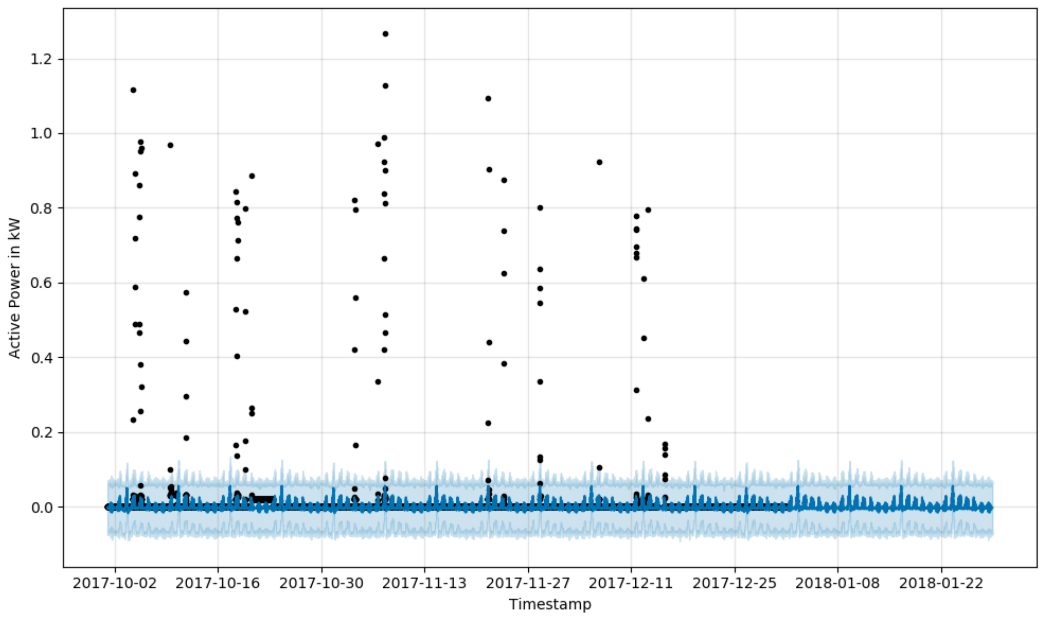

Figure A.1. Forecasting for the vacuum oven with a resolution of 10 minutes

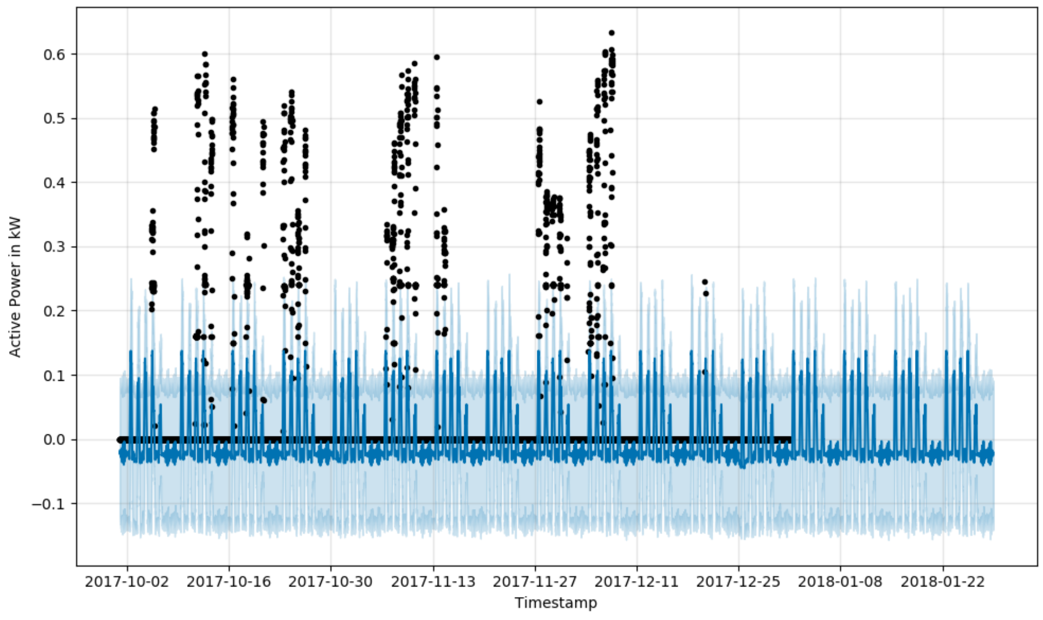

Figure A.2. Forecasting for the chip saw with a resolution of 10 minutes

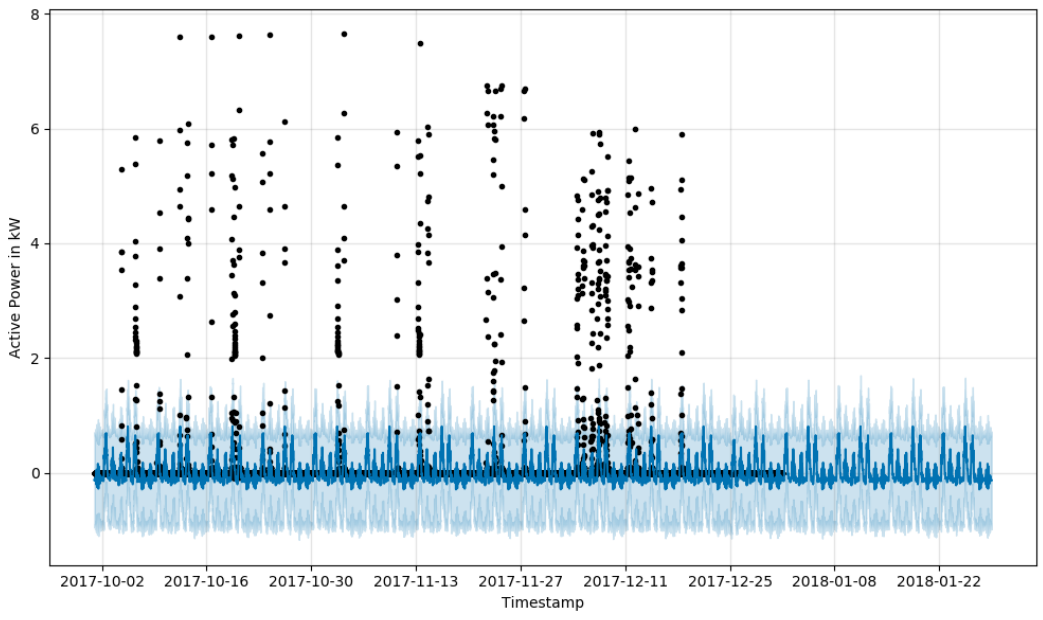

Figure A.3. Forecasting for the high temperature oven with a resolution of 10 minutes

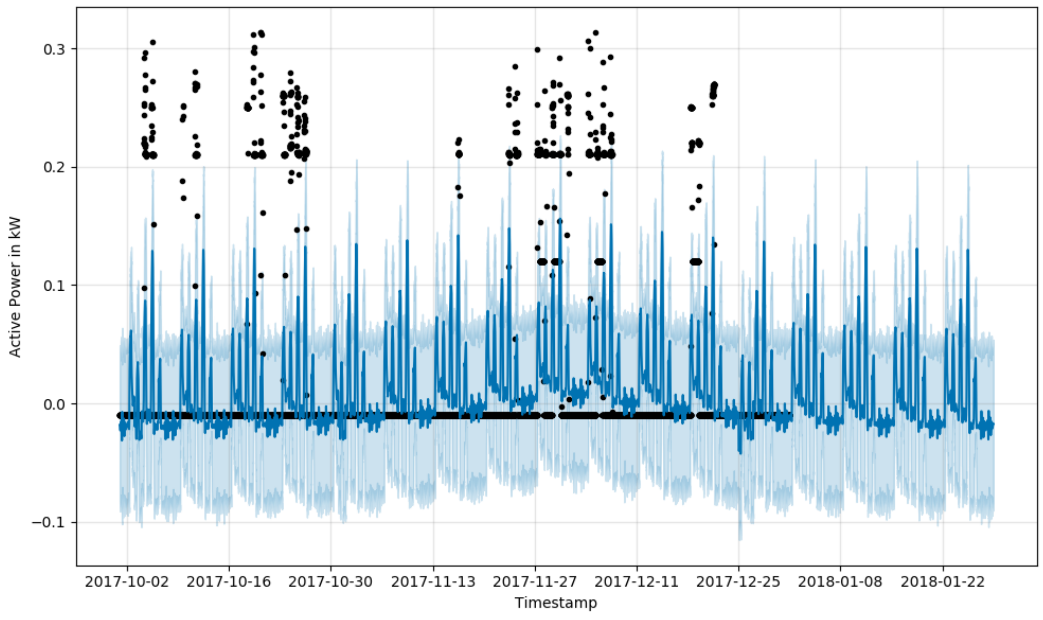

Figure A.4. Forecasting for the pick and place unit with a resolution of 10 minutes

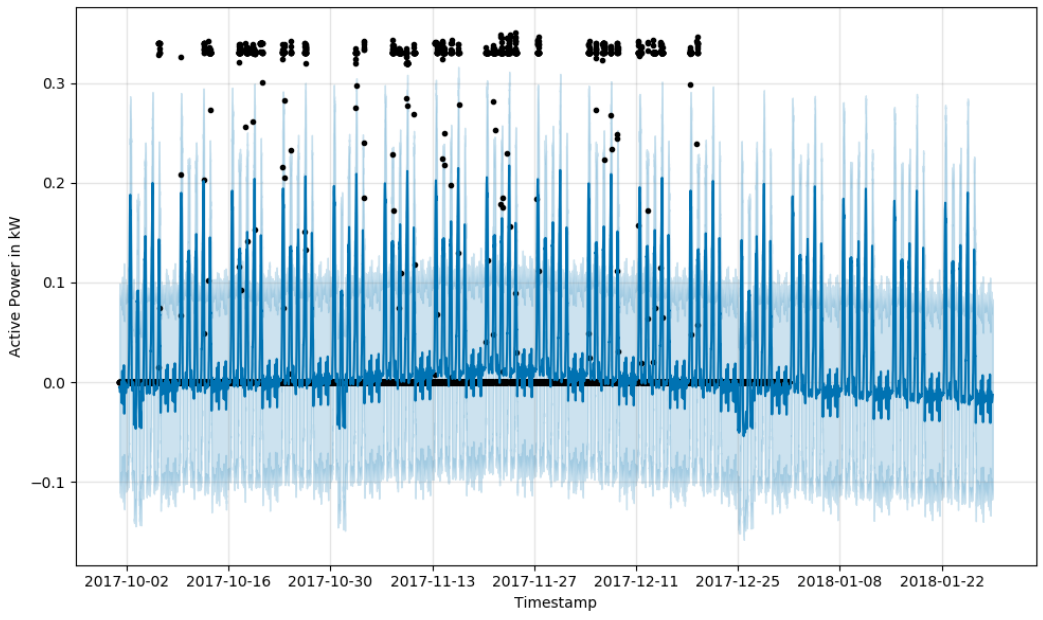

Figure A.5. Forecasting for the screen printer with a resolution of 10 minutes

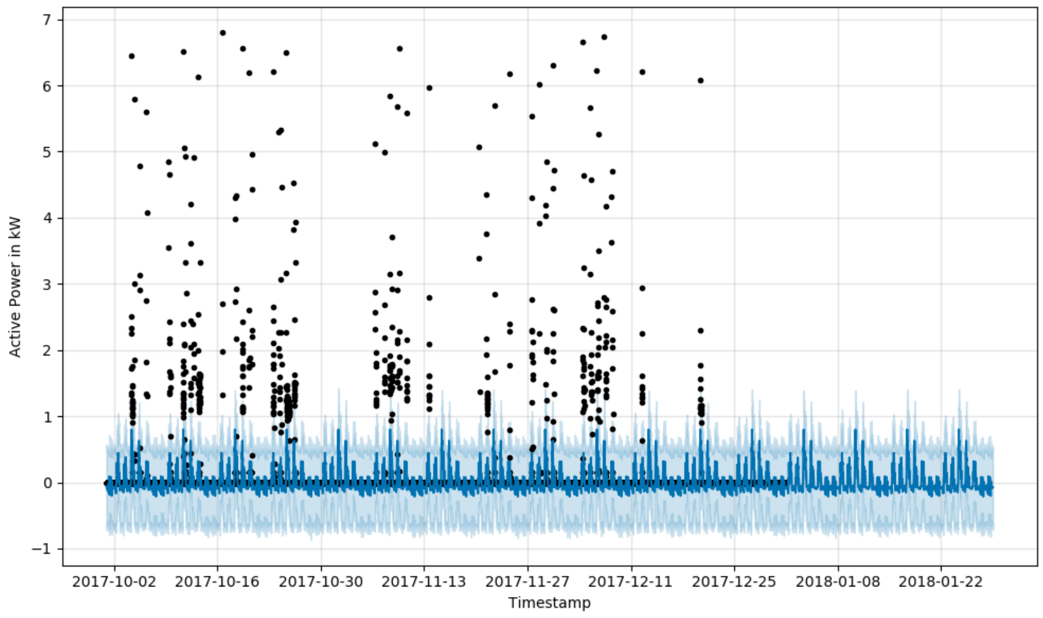

Figure A.6. Forecasting for the soldering oven with a resolution of 10 minutes

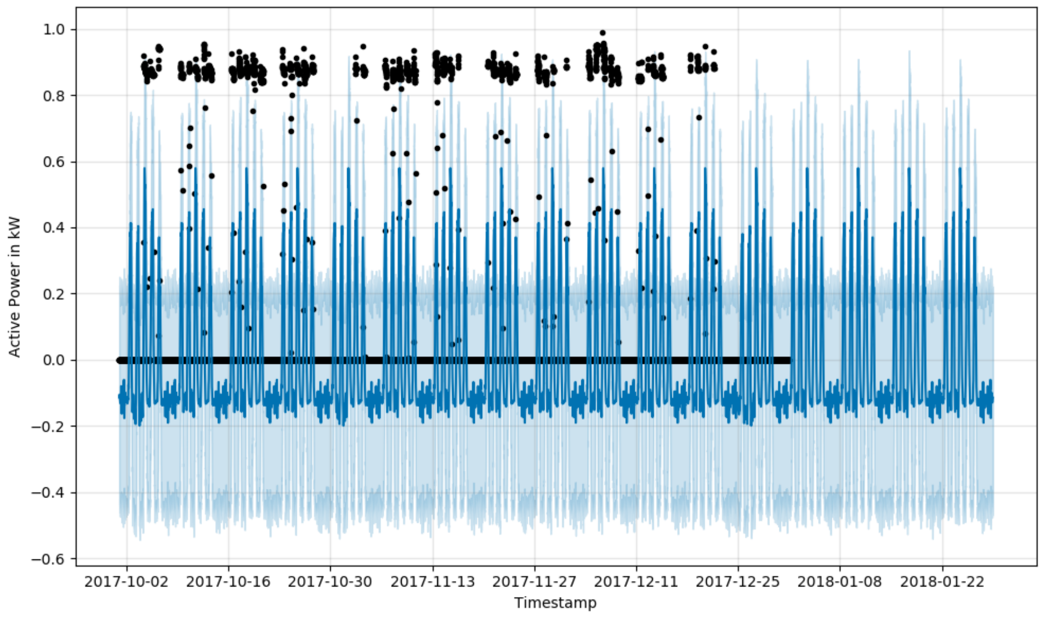

Figure A.7. Forecasting for vacuum pump 1 with a resolution of 10 minutes

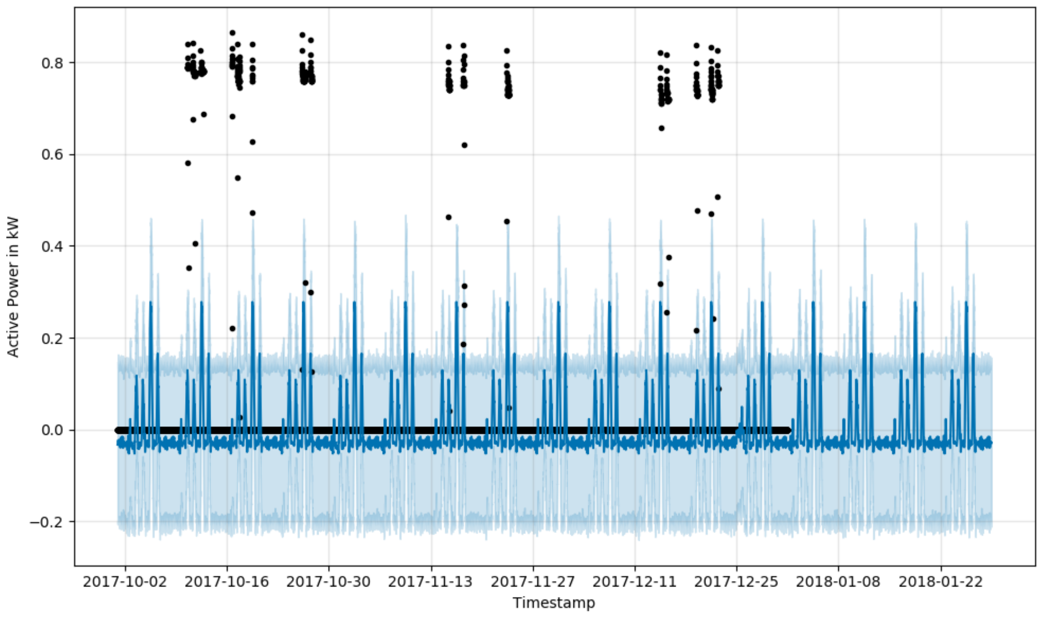

Figure A.8. Forecasting for vacuum pump 2 with a resolution of 10 minutes

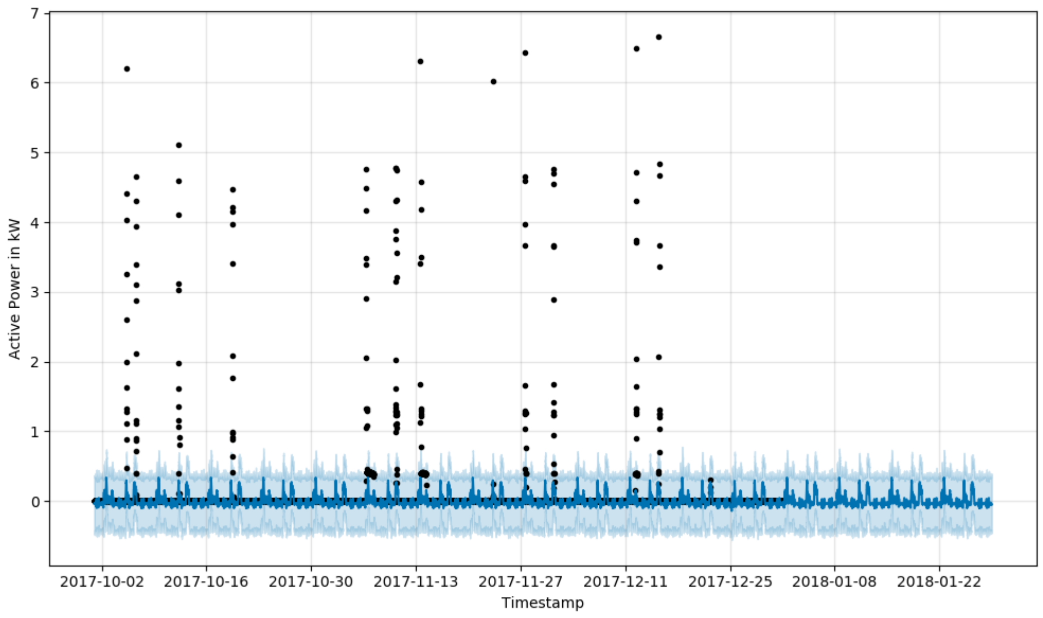

Figure A.9. Forecasting for the washing machine with a resolution of 10 minutes

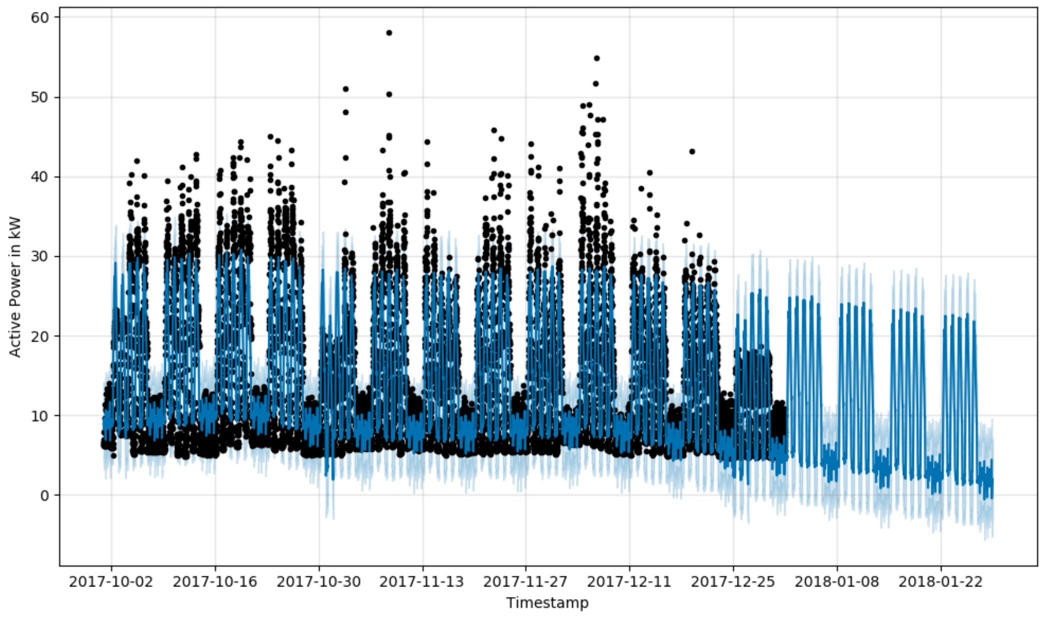

Figure A.10. Forecasting for the main terminal with a resolution of 10 minutes