Real-time measurement and estimation of factory CO_2 emissions

Master of Science in Technology Thesis
University of Turku
Department of Computing
Software Engineering
2021
Conny Karlsson

UNIVERSITY OF TURKU
Department of Computing

CONNY KARLSSON: Real-time measurement and estimation of factory CO_2 emissions

Master of Science in Technology Thesis, 117 p.
Software Engineering
September 2021

The motivation for the thesis was to investigate the feasibility of creating a dynamic system for displaying caused CO2 emissions from a factory environment. Nordkalk's factory in Parainen, Finland, was used to provide real process data to aid in the research. This research topic required the thesis author to acquire deep knowledge and understanding of the methods to obtain the necessary data and of the technologies required to implement such as system.
The information was collected from sources such as literature, factory process data, technology manuals and reference data. Based on this information, key components of the system were created, which made use of the collected process data. Research was done regarding the use of both measured and estimated power usage and caused CO2 emissions data and the feasibility of estimation.
The implemented system components made use of technologies such as .Net Framework, C# language, SQL, OPC UA and SVG graphics. The components implemented the factory process data recording and visualization of the data. Manual analysis was done to evaluate the possibility of estimating CO2 emissions.
The result of the thesis research, shows that it is possible to create a dynamic open system, which could provide detailed information on caused CO2 emissions in a factory context, that could be used together with other systems such as MES systems to create further added-value.

Keywords: CO2, emissions, data collecting, measurement, estimation, MES, factory, process, production, climate change

Contents

Chapter 1

Introduction

The age of industrialization can be considered to have started in Great Britain around the middle of the 18th century, divided into two industrial revolutions, ending in the early 20th century. This industrial revolution then spread to the rest of the world during the latter part of the 20th century. This is the time in history where society moved away from being focused primarily on agriculture and handcraft work and instead started to be dominated by machine manufacturing and factories. The use of machinery required energy to run, energy produced from sources such as coal, petroleum and electricity. New inventions during this time were among others the steam engine, electric generators, the light bulb, electrical engines and the internal-combustion engine used in the emerging automobile industry. [1]

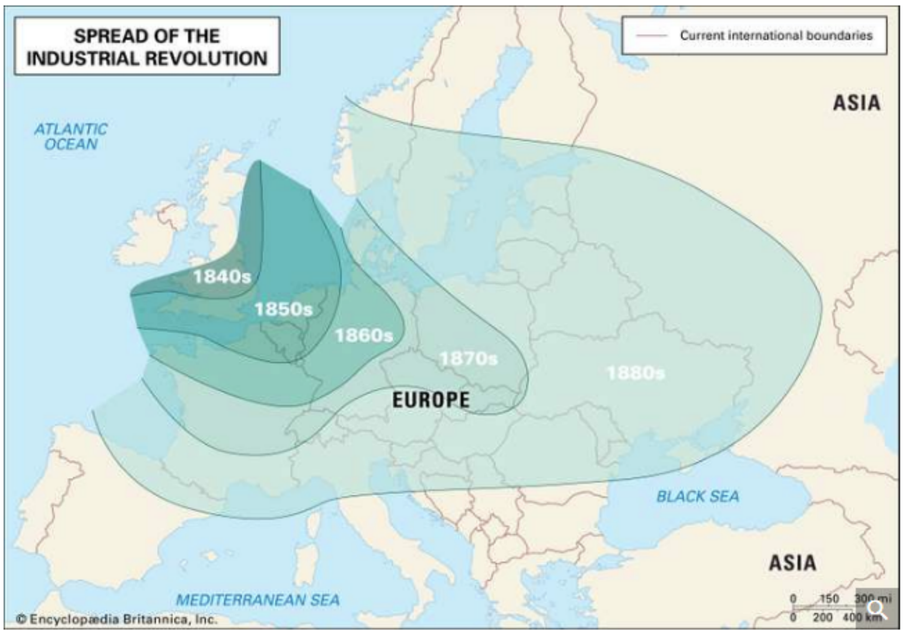


Figure 1.1: The first part of the industrial revolution (Encyclopaedia Britannica, 2019, Industrial Revolution)

1.1 Challenge and problem description

The use and need of energy has gradually caused the increase of air and other pollution, as a result of the processes required for generating energy. The pollution, causing the greatest concern today, is the generation of CO_2 emissions. There is widely an agreement today that a climate change is occurring in the world and that this is one of the most defining challenges encountered in the 21st century (others include for example the increasing global population and great poverty in some parts of the world). There is an on-going global debate regarding the cause for the climate change, where the most widely accepted theory is the increasing level of CO_2 in the atmosphere. The gas CO_2 (Carbon dioxide) is a ”greenhouse-gas”, which means that it is contributing to trap heat inside the earth’s atmosphere, and so forth causing the earth to warm. This effect is scientifically provable. Some do not however believe that this effect is strong enough on a global scale to cause climate change and suggests other reasons for it, like increased activity of the sun.[2]

Climate change and a warming world lead to problems in areas such as global economics, geopolitics, sociology and health. Problems can manifest as large scale migration of people from more affected areas to less affected ones, new wars breaking out because of lack of degrading natural resources such as water and more deaths because of for example heat waves. All of these problems have already been seen in the world and is most likely to increase in the future as the global average temperature keep rising.

More and more measures are being taken to combat the climate change, which focus on limiting the amount of globally produced CO_2 into the atmosphere. Currently the most ambitious measure is the ”Paris Agreement”, where several nations have agreed for the first time to take great measures to combat climate change (mitigation) and also to adapt to its effects (adaption) [3]. The Paris Agreement's main goal is to keep the global temperature's rise below 2 degrees Celsius, above the level seen in pre-industrial times. It is however not easy for nations to agree on matters needed to achieve this goal, as national interests are often considered more important than providing of the economic funds and other resources needed for combating climate change. The prime example of this problem can be seen as USA (the second most polluting country in the world, after China) pulled out of the Paris Agreement in 2019, with the president stating that the agreement was a disadvantage for the country and only beneficial for other countries taking part. [4] [5]

In spite of what measures are being taken, the emission of green house gases keep rising. The level of carbon dioxide in the atmosphere surpassed 400 parts per million for the first time in 2013, which is a level that earth had last time some three to five million years ago, during the Pliocene Epoch [6]. Ways to combat climate change include reducing of the produced levels of greenhouse gases and to increase and enhance ”sinks” that absorb and store the gases from the atmosphere. These measures can help to stabilize the greenhouse gas levels, by which time can be given to the world's ecosystems to naturally adapt.

The following research challenges were defined:

RQ1: is it possible to create a system to display the caused CO_2 emissions from a factory environment?

RQ2: how to design a dynamic system, that can be used to model any factory with the aim to display caused CO_2 emissions?

RQ3: how to technically implement a dynamic system based on the proposed model?

RQ4: what kind of possibilities are there to tie the caused CO_2 emission data to produced product, such as at the level of a production line or a specific produced product?

1.2 Motivation for the work

Carbon dioxide gas is released when energy is created from fossil fuels, like oil, coal and gas. Besides the effect of an escalating climate change, some of the processes creating energy with fossil fuels also releases harmful particles into the air. Particularly PM2.5 particles are harmful for human health, which are particles that have a diameter of less than 2.5 micrometers.

The air pollution is often a more constrained and local problem for a specific geographic area, while the CO_2 emissions contribute to the global problem of climate change.

It can be argued that industries could benefit from knowing as specifically as possible what their CO_2 emissions are. Benefits would include

- a provable method for displaying the impact on climate and air quality caused by the factory processes
- new insights into the contribution for the emissions, for individual parts of the production process

  Earl Baines and Paul Appleton state the following: ”allows companies and consumers to understand the CO_2e implications” [7]

- a verifiable way of using the emission data for positive marketing purposes (as a way of proving low emissions)

  This benefit is similar to the following statement by Earl Baines and Paul Appleton: ”allows consumers to choose lower carbon emission products and services (the potential and implications to a company’s sales and profits of these choices will promote investment in cross supply chain efficiencies and investments in i.e. energy reductions)” [7]

- use of the emission data, to improve their processes to limit the caused CO_2 emissions

- information to enable sustainable development and an environmentally friendly industry

  These two benefits are also listed by Earl Baines and Paul Appleton: ”Awareness can promote sustainable development”. [7]

Chapter 2

Factory & related processes

Section 2.1 aims to give an overview of the concept of an industrial factory. Topics touched include a factory's purpose, the structure of factories and the technology needed to operate factories. Typical IT-systems for handling the data required for the operation of factories are also briefly explained.

The connection between operating of industrial factories and the usage of energy and caused CO_2 emissions, is explained in Section 2.2.

Section 2.3 contains information about the type of data that is typically available from factory processes. Also explained, is the typical ways of saving factory process related data, which are measurement data and event type data.

2.1 Overview of factory elements and processes

A factory's purpose is to produce products. Each factory is made to produce a specific type of product, for example soap, cars or components for other factories. The main process structure of a factory's production lines are set in the planning phase before it is constructed, changing the process at a later stage is always more difficult and costly, although sometimes necessary because of change in demand for the type of product produced.

Factories are made up of several different areas, common to most factories, regardless of what kind of product is produced. It is common for factories to have designated areas for different operations such as receiving of raw-material and supplies needed for production, storage areas for the purpose of having a buffer of raw-material, the main production area of the factory, storage areas for ready made product and a dispatching area for outgoing ready made products.

A factory contains one or several production lines, which usually all produce the same type of product or variants of a product. The production process can be continuous or batch based. A continuous process keep on producing a specific product (as output) for as long as the required raw material is supplied (as input). A batch process, is creating batches of a product, where each batch is made up of a certain pre-planned amount or number of products (based on a production order). Both types of production processes leave a carbon footprint, because both are using the same kind of mechanical and electrical automation hardware. Although factories contain several areas (as mentioned earlier), it is usually the production area that causes most of the factory's produced emissions and therefore the area to mostly consider in regards to emissions and energy use.

Production processes transform the supplied raw-materials into products, based on a pre-defined process that can be adjusted and fine-tuned during each production run, according to supplied run parameters. Raw-materials are worked on with different kinds of machinery, such as for example kettles, pumps and packaging machines in a food producing plant or transport conveyors, mills and suction pipes in a mineral processing plant. The elements that make the machinery work are automation instruments such as electrical motors, controllable valves, frequency modulators and many different sensors such as temperature, pressure, weight, position, level and flow sensors. A concept of key parts in a manufacturing process can be seen in Figure 2.1.

A factory wide automation network system is used to combine all the machinery together into a process. The process itself is then controlled by one or more programmable

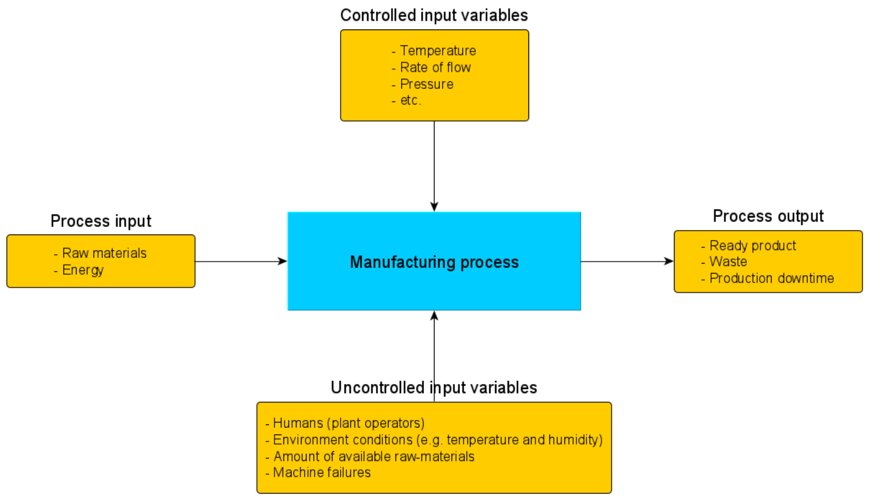


Figure 2.1: Concept of a basic production process's input, output and variables

logic controllers (PLC), which react to all sensor input by controlling the machinery, so that the desired result is reached in any given phase of the production process.

The PLCs are based on computer process units (CPUs) that are designed specifically to have a very fast program cycle time. The fast cycle time is important, making it possible to react in near real-time to sensor input and to issue time-sensitive control commands through the PLCs outputs. As an example, a pallet on a transport conveyor must stop at an exact location within two centimeters, in order for a forklift to able to reach it. An optical sensor might be used to determine when the pallet is at this location and the PLC must monitor the input signal, in order to control the output for stopping the conveyor at the correct time. Typical program cycle times for a PLC are about 5 to 100 milliseconds, depending on the CPU model used and the size of the executed program.

Most factories also have a SCADA system in place. SCADA stands for Supervisory Control and Data Acquisition. This is a type of software used on computers (usually in a factory's control room) for giving the factory operators an overview of all production processes. The operators are able to supervise and control the processes using the SCADA system, as this software is in direct communication with the factory's PLCs. The SCADA software also gathers production execution data from the processes and enables reporting of production events and alarms. Smaller control panels are also often found around a factory, called HMI (Human Machine Interface) systems, these are used for supervision and control of a sub-system in the factory. The HMI-systems are similar to the SCADA system but limited in functionality and connectivity, and only made to serve a certain part of the factory.

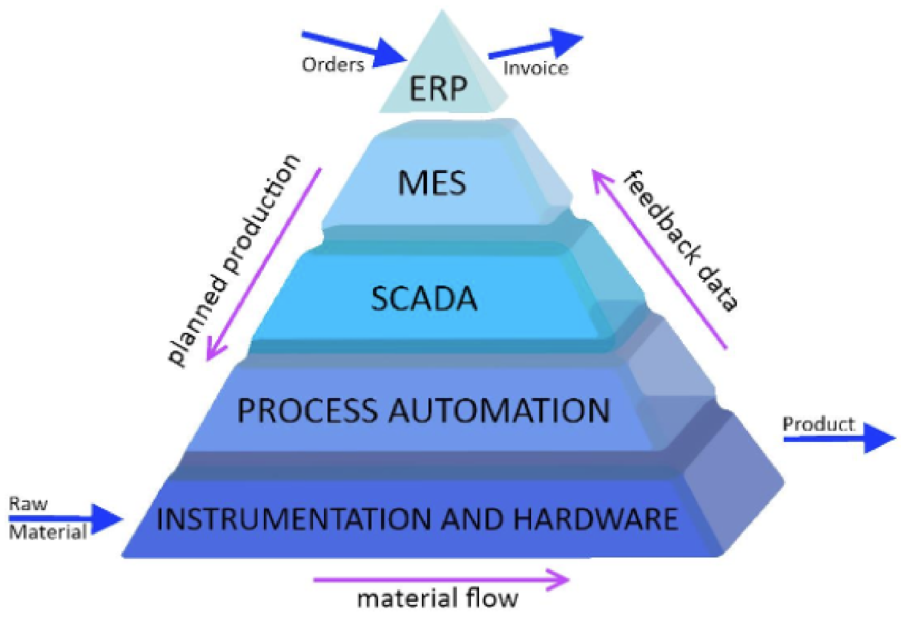


Figure 2.2: A pyramid description, showing the relations of the different systems used for running a company and its factory. [8]

Many factories also implement a MES system, short for Manufacturing Execution System. Before going into what a MES is needed for, some words should be mentioned about ERP systems that companies make use of and which usually are used together with MES solutions. Figure 2.2 shows the relations and levels between the different systems used to run a company and its factory processes.

ERP is short for Enterprise Resource Planning. ERP systems are software solutions used by companies to manage their business at a high level. ERP systems are used for concentrating all of the vital business data into one system and to manage the business based on the knowledge gotten from connecting these separate data sources to each other.

Some of the data saved into an ERP system are customer data, supplier and purchasing data, warehouse inventory data, sales orders, resource consumption data and production data. An important feature of an ERP system is to avoid using several different overlapping software solutions for handling business matters and to have all data saved only once in one location, to avoid data duplication and validity issues. Another important reason for using an ERP system, is that it enables the business to be run efficiently. An example of improved efficiency is when the sales orders and product inventory data is in the same system, as it can then be determined and forecast when production of products must take place, in order for the company to be able to supply the customer with the ordered product amount, at the right moment, without keeping unnecessary high volumes of inventory. Some of the most used ERP systems in the world, include:

- SAP ERP [9]
- Microsoft Dynamics 365 [10]
- IFS ERP [11]
- Oracle ERP [12]
- Epicor ERP [13]

Without going into more details about all functionality that an ERP solution provides, it can be said that a typical ERP solution does not contain the specific functionality of data handling and production control that is needed at the factory level. This is where a MES solution is needed, positioned in between the ERP solution and the automation SCADA solution that enables the production at a factory. More information regarding ERP systems can for example be found from the thesis by Jaakko Kotiranta [14].

A MES system is needed for handling the detailed needs for production in a factory. These are matters related closely to the production procedures, which means that the MES system must be configured separately for each factory, it is never a ”plug-and-play” software solution that would work ”out-of-the-box”, without manual configuration. A MES system typically handles the recipes and versioning of the data needed for producing the products and is used together with the SCADA system to send production orders and recipe data to the automation system. Production orders typically originates from the ERP system but the order data is supplemented with the manufacturing related parameter data by the MES system, before the order is sent to production. A MES system is also typically responsible for receiving and handling of production related report data from the automation and SCADA system. This data can be used to create and show reports from production events for the factory operators, but is also usually sent in an unprocessed format to the ERP system, for reporting of for example the amount of produced product and consumed raw-material.

A MES system can also be responsible for handling other tasks at the factory level, such as handling of production scheduling and recipe related data and the user interface for material receiving and product dispatch, data which also often originates as receiving/dispatch orders from the ERP system and which then are reported to the ERP system once the receiving/dispatch event has taken place. More complex MES systems have traditionally been tailor-made to enable the specific requirements of a customer’s production process but today there are however attempts to create more general ready made MES applications, which include:

- Aveva MES [15]
- Leanware MES [16]
- Epicore Advanced MES [17]

A MES system might be suitable for storing caused emission measurement data for production events, manufactured products and production batches, in order to be able to report for example the CO_2 emissions for a certain production batch.

2.2 Relation to energy consumption and CO_2

All of these factory elements require electrical energy to work, some more than others. The most energy intensive elements are usually the ones that are responsible for physical movement, such as belts, conveyors and material processing equipment. These elements often make use of electrical engines. Processes that make use of elements such as ovens or kettles are often the most energy requiring of all, as these require a high temperature to work, which is only achieved by use of a relatively high amount of energy. On the other hand, these elements have a near zero need for energy at times when production is halted, while elements such as the control system and sensors always stay in a on-mode consuming energy.

The amount of energy consumed by factories matters because of reasons such as energy costs (higher costs lessens the profit), climate and law issues that need to be followed and the overall image of the company being environmentally friendly or CO_2-neutral. Measurement of energy use and the conversion of energy data to CO_2 emissions can be followed at different levels in the company, starting from individual machines on the production line, to a specific production area, to a specific production line or to the whole factory. The obtained data for consumed energy, will be more valuable if it is obtained from the lowest level available (e.g. based on an individual component such as an electric engine instead of a measurement reading for an entire production line). This is because, it will enable more precise and detailed reports and analysis.

2.3 The kind of data available from the processes

SCADA systems always include functionality for collecting of data from the automation processes. The collected data is usually categorised into two groups, measurement data and event and alarm data. The main difference between these are how the data is collected. Measurement data is usually collected at set time intervals, examples are:

- temperature measuring of a cold storage, one collected data point per minute
- surface level measuring of silos, one collected data point per 10 minutes
- pressure reading from a pipe transporting a chemical substance, one collected data point each 500 ms
- a weight scale reading for the end product on a production line, one data point collected each 5 seconds

Collection of alarm and event data differs from the collection of measurement data, in the way that the data is not collected at certain predefined time intervals, instead the data is collected at the moment that an event takes place. Examples are:

- a temperature reading going higher than a predefined level, resulting in an alarm event being generated
- an operator acknowledging that the raw-materials have been added for a certain step in production while producing a product

Event-based logged data can also save additional secondary data, that is related to the event itself. An example of such a situation is an alarm triggered when the temperature rises over a set limit in a mixing silo. The alarm data would include the ID for the sensor triggering the alarm, the current temperature reading, the timestamp for when the alarm took place but could also include secondary information like for example the current recipe values in use for controlling the mixing process. Another example of an event in a process is the change of state. A change of state can for example be when an electrical engine is controlled from a state of off to on, or when a valve is set from state closed to state open. A defining difference between alarm and event data is that an alarm is something that an operator often should react to, in a timely manner, and then acknowledge when the cause of the alarm has been resolved. A common event on the other hand, does not typically require the operator to react to it, it is instead information that can be valuable as history data when for example production log data is examined or for reporting needs.

Reporting of caused CO_2 emissions could possibly make use of both measurement and event type of stored process manufacturing data. Measurement data could be valuable, in cases where industrial machinery and equipment would be able to report measured or calculated data, on CO_2 emissions and energy use. Measured data might in such cases be considered to have a high degree of accuracy, as the source of the data originates from the same point where the needed functionality takes place. Event type data might on the other hand be useful in estimating the caused CO_2 emissions, for example in the form of knowing when and for how long a type of machinery has been operated. Event type data could possibly be used as a source, which by analysis, refining and processing could reveal estimated data on caused CO_2 emissions and energy use.

The magnitude of measurement, alarm and event data can reach quite big quantities in many factories. Measurement data typically makes up the bigger portion compared to alarm and event data, as it is most often stored at pre-defined intervals which usually occur more often than individual alarms and events. Alarm data can also occur as repetitive, in situations where a sensor reading is fluctuating over and under the set limit for the alarm (a situation that can be eased by use of alarm hysteresis settings). The stored information itself, for a certain measurement point or event, does not require much storage space - but the number of stored measurement points or events can quickly add up. The storage capacity of a common industrial PC is today however usually not a limitation, while the processing, analysis and usage of the stored data capability can be a limitation, because of the large amount of saved data points and the limited processing power of one PC. This data can be considered to be the Big Data of the industrial manufacturing business.

Current trends regarding industry automation world-wide include what are called Industry 4.0 and Industrial Internet of Things (IIoT). These are terms for new technologies and new ways of thinking about how to execute production. Industry 4.0 (known as the 4th industrial revolution) aims to improve manufacturing by for example introducing more flexibility and predictability into the production process, delivering of more customized products and an overall increase in production efficiency. IIoT aims to make use of IT-technology as part of the manufacturing process, opposed to having IT-technology as separate connected systems outside of the manufacturing itself. IIoT technologies can enable for example manufacturing machines to communicate directly with each other, to have sensors driving the actuators without externally given commands and machine embedded analytics. In an essence, manufacturing can become smarter, more dynamic and self-controlling. [18] [19]

An example of a platform enabling Industry 4.0 and IIoT is Siemens's MindSphere, which among other things enables the connectivity between the manufacturing floor and the cloud and the processing of collected cloud data and creating of cloud-driven applications for usage of the data. [20] The increased level of sophistication on the manufacturing process level, might also provide more usable, exact and available data for the measurement and estimation of CO_2 emissions.

Chapter 3

Measurement and Estimation of energy use and emissions of CO_2

Energy sources are either renewable or non-renewable. Renewable energy sources include for example electricity from solar panels, water power plants and nuclear energy. The energy produced from these are considered to not create emissions such as CO_2 gases. Non-renewable power sources are for example coal and gas power plants. These cause considerable amount of CO_2 emissions when the raw-material is burnt in order to create energy in the form of heat, of which a part is then converted into electricity.

Buyers of energy usually use the required energy locally, while not all of it is generated locally. This fact enables one to make the distinction of direct emissions and indirect emissions. Direct emissions are emissions caused by producing energy locally, for example by burning of coal for heat generation. Indirect emissions are those, which are caused by power generation off-site, like bought electricity from the national power grid. Indirect emissions are also such which originate from processes not controlled by the user of the process, but still somehow related to the activities.[21]

Direct emissions can be controlled by the end user, as it is possible to exchange a local power generation source from a more polluting to a less polluting one. Indirect emissions can not be controlled locally, as the power is generated by an external partner producing the power.

The use of energy and emissions of CO_2 are directly related to each other, as it is the process of generating the required energy that also creates the unwanted CO_2 emissions. The source used for generating the energy establishes the amount of CO_2 emissions, which ranges from completely non-polluting sources to highly polluting sources. Even completely non-polluting sources, such as solar panels, have in most cases a carbon footprint, as pollution is produced in the processes that extract the raw-materials needed, from the process of manufacturing the panels and also to a lesser degree from the transportation of the raw-materials to the factory and from transportation of ready made solar panels to the shop and forward to the buyer of them. This latter type of indirect emissions is excluded from the type of emissions considered in this thesis.

Measuring and estimating the amount of used energy and CO_2 emissions can be of value and use for many applications.[22] Some examples include:

- knowing where to invest funds to see the greatest reduction in energy use and caused emissions (for less pollution and lowered costs; combating climate change)
- improving energy efficiency (monitoring and follow-up)
- predictive maintenance as some early indication of equipment failure can be seen through increased energy use
- combining of CO_2 emission and business data, for example to show the amount of CO_2 emissions caused for production of a given product

3.1 How to measure

Energy is available in different sources, such as electricity, gas, heat and by fossil fuel such as coal. There are many reasons why a stakeholder would be interested in knowing the amount of energy used. The primary reason is often economic, as the energy used must be bought and paid for. The understanding, on top of how much energy is used, in addition to why, when and where the energy is used, can lead to new insights on how to decrease the amount of energy needed. Measured energy consumption data can be used to improve energy efficiency, which in return means lower costs. Energy efficiency is defined by Jonas Lodén from Chalmers University in IEE's Path-To-Res report "Methodology for efficiency analysis", as "*a ratio between an output of performance, service, goods or energy, and an input of energy.*", while energy efficiency improvement is "*an increase in energy end-use efficiency as a result of technological, behavioural and/or economic changes.*". The report also points out the distinction, that should be kept in mind, that a reduction in energy use is not the same an improvement in energy efficiency - although both cases can be measured as a reduction in energy use.[23]

Another reason to measure energy consumption, is to be able to identify targets needing maintenance, at a stage before a total break-down occurs. This predictive maintenance is especially valuable in a factory context, to be able to identify maintenance needs in time and to then schedule the maintenance during times of planned stops in production. The gradual increase in energy use by certain machinery will indicate a demand for maintenance.

When measuring energy, it must also be taken into account where measuring is worth doing. Energy measuring devices and systems cost stakeholders in terms of paying for the system itself, costs of installing and maintaining the system and costs from training operators to use it. Therefore, it should be considered to use energy measurement for example on only high-energy demanding devices or in a factory context to measure the whole energy consumption for a production line, instead of individual components.

3.1.1 - energy

Ready working energy measuring devices and systems are widely available for purchasing. One such system, specifically aimed for industries, is Siemens' "Simatic Energy Suite" [24]. The system consists of more than merely energy meters, as it can be integrated with other industry automation equipment in the Simatic series. The energy meters for consumed electricity, available from this system are "SIMATIC S7-1200 SM 1238 Energy Meter 480VAC" and "SIMATIC ET 200SP AI Energy Meter 480VAC ST". Other solutions from other industrial automation providers are also available, such as the PowerMonitor devices from Rockwell Automation [25] or the smart power monitors from Omron Industrial Automation [26].

Figure 3.1: Simatic electric energy meters [24]

Figure 3.2: Rockwell [25] and Omron [26] electric energy meters

The meters can be used to measure the energy consumption on individual devices, machines or for a whole production line.

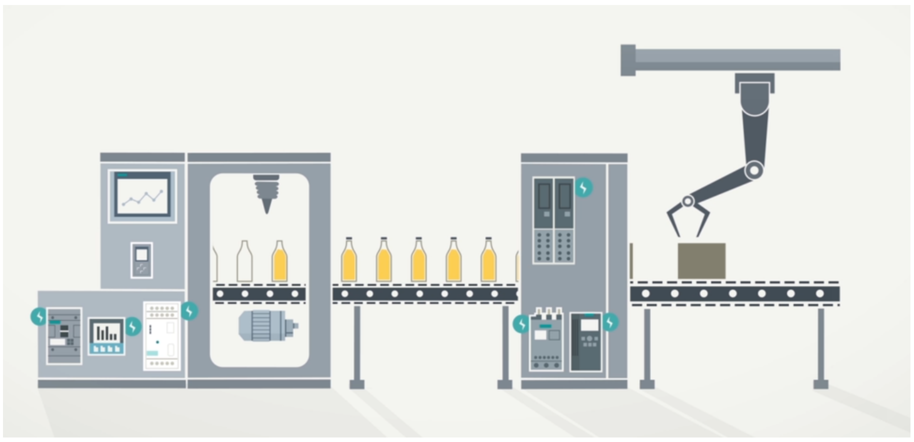


Figure 3.3: Simatic electric energy meters installed on a bottle packing line [24]

A packing line for filling bottles, as illustrated in Figure 3.3, is typically made up of several components. These include an electrical engine for running the conveyor belt, a filling pipe for filling each empty bottle passing by on the belt, a packing module for packing several bottles into one carton package and a robotic arm for removing the packages from the conveyor belt. All energy use for the packing line uses electric energy. In this case, the manufacturer of the packing line, has installed electric energy meters separately on each of the packing line's modules, which are the filling module, the packing module, the conveyor belt shared between the modules and the robotic arm. This level of energy measurement gives an enhanced level of detail, compared to the optional way of installing only one energy measuring system covering the entire packing line.

Other production processes make use of more than electrical energy, such as energy in the form of gases or liquids. Figure 3.4 displays one example of a measuring equipment used to measure the amount of used gas, made by Siemens [27]. Steam is an example of an energy source commonly used in many production processes, where the process requires heating. Heating is used for example while creating products such as chemical detergents, sauces for the food industry and feed products for animals. Liquid flow meters are often used for measuring material flow within a process but depending of the area of use, they can also be used to measure energy usage. An example for measuring energy use by a liquid flow meter, is when an oil burner is used for creating heat and the amount of energy use is measured by the amount of consumed oil.

Figure 3.4: Siemens Sitrans FC300 gas and liquid flow meter [27]

3.1.2 - CO_2

Measuring of caused CO_2 levels is possible by use of for example measurement instruments created by Vaisala company. These measuring instruments work by using infrared light (IR) sensors, as a gas like CO_2 is made up of atoms in such a way, that the gas absorbs IR radiation in a unique detectable manner. A filter is used to only let the wave length of the IR light through to the sensor, which is used to detect the CO_2 gas. The intensity reading of the IR light is then converted into a comparable gas volume concentration value, measured in parts per million (ppm).

The surrounding environment's temperature and pressure must also be taken into account, as they have an influence on the measurement of the of the CO_2 gas concentration. The correctness of the CO_2 measurement value is therefore compensated by use of the Ideal Gas Law. The effects on the CO_2 readings from changes in temperature and pressure can be seen in Figure 3.5 [28].

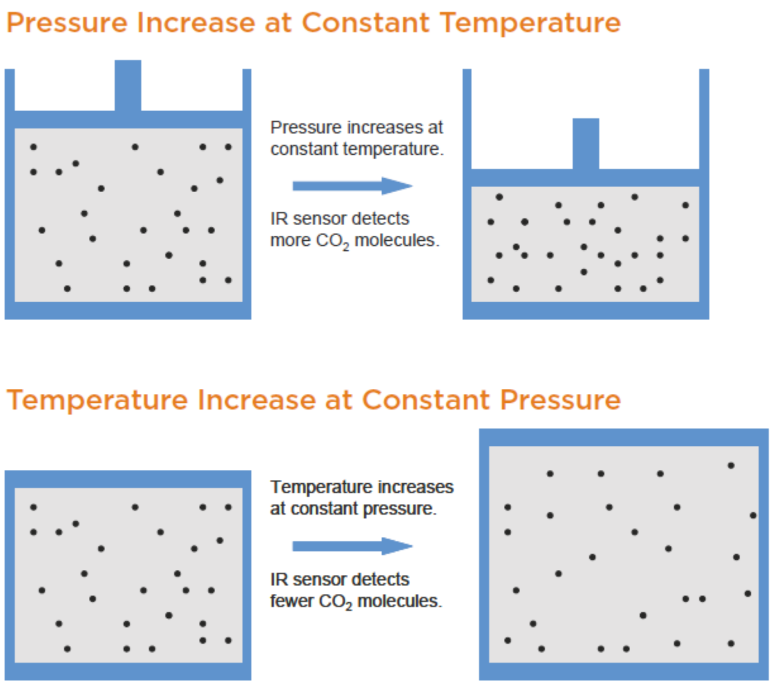

Pressure Increase at Constant Temperature
Pressure increases at constant temperature.
IR sensor detects more CO_2 molecules.
Temperature Increase at Constant Pressure
Temperature increases at constant pressure.
IR sensor detects fewer CO_2 molecules.

Figure 3.5: Effects of temperature and pressure changes on CO_2 readings [28]

The calculation of gas concentration, while taking the environment's temperature and pressure into consideration, is done according to the following formula:

$$\rho(t,p) = \rho(25C, 1013hPa) \times \frac{p}{1013} \times \frac{298}{273+t}$$

where

ρ = gas volume concentration [ppm or $\%$]

p = ambient pressure [hPa]

t = ambient temperature [°C]

with units used

Celsius ([°C]) [29] and hPa (Hectopascal Pressure Unit) [30]

Certain industries make more use of direct CO_2 measurement in their production, but it is not widely in use. Processes that can benefit from CO_2 measurement are such that make use of extreme heat, as this typically requires considerable use of a local energy source. This energy is typically produced by burning of fossil fuels, like coal or oil. An example of such a process requiring extreme heat, is the burning of lime stone, in order to produce quicklime (CaO). One such industry is Nordkalk's lime kiln factory, located near a lime stone quarry in the city of Parainen, Finland. [31]

Figure 3.6: Lime kiln, Parainen, Finland [31]

Knowing the CO_2 emissions from such a process can help to fine-tune the efficiency of the heat generation process (which require temperatures of around 1000 °C). Accurate CO_2 emission data may also be required, when buying emission allowances from the EU Emissions Trading System (EU ETS). ETS is one of EU's main tools for combating climate change, by use of a carbon market shared by all EU members plus Iceland, Norway and Liechtenstein. [32]

Figure 3.7 shows the proportion of energy sources used on average by manufacturing in OECD member countries. OECD is short for "The Organization for Economic Co-operation and Development" and is an international organization with a goal "*to shape policies that foster prosperity, equality, opportunity and well-being for all*" [33]. IEA is an autonomous international governmental organization within the OECD framework and the data in Figure 3.7 originates from its statistics report on "Energy Efficiency Indicators Highlights" (edition 2020) [34].

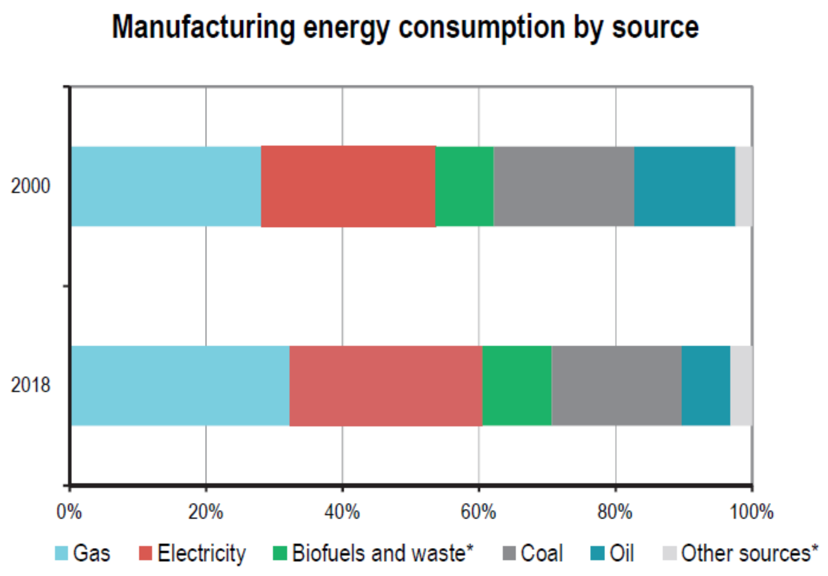


Figure 3.7: IEE's statistics on Manufacturing energy consumption by source, during years 2000 and 2018 [34]

It can be seen from the statistics that around 25% of the energy needed, used by manufacturing is electrical energy, while most of the rest is from sources local to the factory environment. Around 75% of the energy need is produced locally, as fuel such as gas, coal, oil and bio fuel must be delivered on-site to the factory before use by burning. The local energy sources can have their CO_2 emissions measured by installing CO_2 sensors. The direct direct measurement of CO_2 emissions for equipment run by electricity is however not possible, as the electric energy is typically generated hundreds or thousands of kilometers away in electric power plants, such as nuclear, coal and gas power plants.

3.2 How to estimate

Measuring the amount of energy and CO_2 consumed, must be considered to always be a better option, compared to estimation, as measurement by calibrated sensors will provide more accurate readings than what could be achieved by estimation. Measuring however has the downside, that it requires the purchasing of measuring equipment, which will incur costs for the factory stakeholder. Additional costs would also occur from installing and maintaining the measurement equipment. In cases, where the economic impact of purchasing and installing measuring devices, can not be justified, an estimation of consumed energy and caused CO_2 emissions might be a viable option. Other reasons for estimating, instead of measuring, can be situations where it is impossible to measure (e.g. electrical equipment's CO_2 emissions) or when it could be physically impossible to install measuring devices, for example because of limited space.

3.2.1 - energy

SCADA software, used for controlling and monitoring factory processes, have built-in functionality for logging of data. It is common in every factory to log process data such as environmental variables, some examples being temperatures, pressures and tank levels. On top of these, process state data is also logged, for example the time and associated data for when factory equipment is active and running and when equipment is turned off.

Based on the knowledge of when a piece of equipment has started and stopped, the time interval of operation can be determined. This length of time that the equipment has been running, could be used to estimate the total energy it consumed during this time, if one would also know how much energy the equipment needs to run. Figure 3.8 shows a photo of a typical electrical engine, this one manufactured by Siemens [35]. Electrical engines are sold in different sizes, where a larger physical size typically means that the engine is more powerful compared to smaller variants.

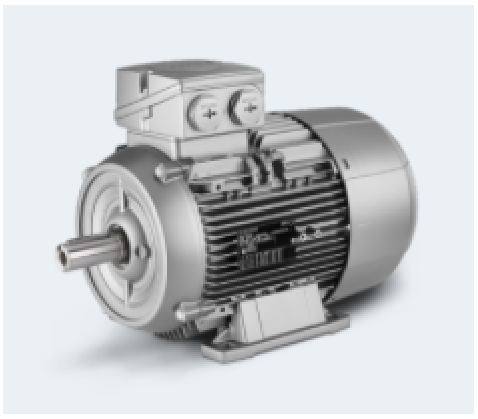


Figure 3.8: Siemens electrical engine [35]

The strength of an electric engine is specified in kW, short for kilowatt. Watt is a measurement of power and kilowatt stands for 1000 watts. The amount of used electrical power is typically measured as kWh, which stands for kilowatt hour. This is a unit of measurement, that equals the amount of energy, that would be used if some equipment that used exactly 1000 watts would be running for one hour. So, as an example, if an electrical device using 100 watts would be on for exactly 10 hours, the total amount of used electric would be 1 kWh, or 1000 Wh (10 hours * 100 watts) [36].

Equipped with the information, regarding the power of an electrical device, and knowing the time of operation, the total used power could now be calculated. As an example, an electric engine in a factory would be of the power of 2 kW and it would have been operating for 30 minutes. The total energy consumed could be calculated as

$$\frac{2000 watts}{(30/60) hour} = 1000 Wh$$

which is the same as 1 kWh.

In reality, this result might not be true. This is because the electrical engine's maximum power is specified as 2 kW, but using it does not necessarily consume the maximum available power. If the load that the engine operates is light, then the power in use is less than 2 kW. A higher load than the engine can handle, would result with the engine stopping. The load should therefore be taken into account when estimating the consumed energy, as the estimation would otherwise almost always be way off, compared to the actual consumed energy. [37]

It depends on the type of electrical equipment if the load must be taken into consideration while estimating the total consumed energy, as for example a light bulb (without a dimmer) would continually consume the maximum power that it is designed to use.

Other energy sources than electric, must have the fuel stored locally on the factory premises. In cases, where the amount of fuel is not measured at the point of usage, an estimation could perhaps be based on knowing the amount of fuel consumed by shift, day, week or some other suitable time period. With this knowledge, the percentage of time that an equipment has been used, could be thought to have used a comparable amount of the total consumed fuel. Here, the load is again a factor that also matters. As for an example, a gas burner could be used for 1 hour, but a different amount of gas would be consumed, depending on if the gas is used to heat up a process to 50 C or 300 C - knowing the operating time is not enough to do a some what accurate estimation.

3.2.2 - CO_2

Carbon dioxide emissions can be estimated based on information about the amount of consumed energy and the type of energy source used. A factor must be applied to the amount of consumed energy, by which the energy is converted into caused CO_2 emissions.

The reliability of the result, is based on how accurate this factor is. For a reliable result, the factor should take into account how much of the energy is produced using renewable power sources and how polluting the remaining non-renewable power sources are.

A standardized protocol has been developed for calculating how much CO_2 emissions a company (or any other entity) causes. The protocol is called the GHG Protocol, which is short for ”The Greenhouse Gas Protocol”. [38] As stated by Henrik Olausson, in his thesis ”A tool for calculating CO_2 emissions in the manufacturing industry – Use of GHG protocol”, the GHG Protocol stands for ”*The Greenhouse gas protocol corporate accounting and reporting standard and the Global protocol for community-scale greenhouse gas emission inventories (GHG protocol) was founded 2004 and 2014 by the World Resources institute (WRI) and the World Business Council on Sustainable Development (WBCSD). Together with help from governments, other firms and NGOs they created this base for greenhouse gas (GHG) emissions reporting (Green 2010).*” [21] [39]

The World Resources Institute states on its web site that ”*Greenhouse Gas Protocol provides standards, guidance, tools and training for business and government to measure and manage climate-warming emissions.*” [38] Henrik states in his thesis, that the GHG reporting is based on three scopes, which together summarize the analyzed entity's total emission of CO_2-equivalents. The scopes are as follows:

- Scope 1: Direct emissions that come from owned sources
- Scope 2: Indirect emissions from purchased electricity. The emitted CO_2 from the generation of electricity.
- Scope 3: Other indirect emissions from processes not owned by the company but related to company activities.

CO_2-equivalents stands for all the different processes an entity has, that contributes to greenhouse gas emissions. Such processes can include for example extracting of raw materials from the earth, processing of the materials, transportation, consumption of energy, manufacturing, waste handling etc. These activities are converted into CO_2 emissions, by using guidelines available from the GHG Protocol in order to create tools fit for the entity's processes and activities. The accuracy of the reports is based on the quality of the reported data available. The data available comes in many forms, depending on the process in question. The data can be for example amount of gas used, amount of electricity used or the length of time some equipment has been in use. Most data available from the processes need a conversion factor, by which the data is converted into CO_2-equivalent.

The GHG Protocol aims to consider each direct and indirect source of CO_2 that an entity is responsible for. This could include such details as CO_2 emissions caused by a company's employees, while travelling to and from work or leakage of gas from Air conditioner units. The guidelines provided by the GHG Protocol could also be suitable when estimating the CO_2 emissions in a production environment, as the same principles of conversion factors and other guidelines part of the GHG Protocol, still holds true also for this level of consideration, and might provide insight into more successful methods of CO_2 estimation.

Chapter 4

Analysis

4.1 Related work for modeling of a factory environment

A factory is a complex environment, holding within its physical construction, a wealth of processing equipment, automation hardware, electrical installations, cabling, control rooms and other facilities needed by the operations of the factory. The process equipment is often not only placed horizontally, but also vertically within the factory. The path of raw-material, proceeding throughout the factory in a process that transforms it and finally produces a ready product, can physically be moving both horizontally and vertically within the factory.

Thinking of the factory's physical layout, could perhaps make one think, that a three dimensional graphic model would be a good way to create a system for displaying caused CO_2 emissions in a factory context. A three dimensional model might however be more fitting for other purposes, such as for planning a new factory's layout or considering physical changes to an existing factory (e.g. new pipes to be installed), for purpose of validation of physical changes.

A process within a factory is typically displayed as a two dimensional graphical model, where the connected parts of the process and the equipment for the production line is shown next to each other. Equipment placed physically vertical to each other can also be shown as such in a two dimensional view. Additionally, the number of equipment installed in a factory is usually so numerous, that the two dimensional view must be divided into several process screens, where each screen is used to show a part of the process. The view of a two dimensional model, will however allow for an efficient way of showing status information (such as CO_2 emissions) by an easily understandable overview, as several factory equipment can fit as graphical objects on a screen at the same time, while showing the connections between the equipment, that as a whole are responsible for the factory process.

A display model using only text information and lists could also work, with the benefit to allow the largest amount of equipment to be viewed simultaneously. The down side of this model however, is the difficulty in understanding the connectivity between equipment and the flow of the process, that a two dimensional graphical view will provide. The interconnections between equipment can often be important to understand and take into account, as the state and performance of one device often has an effect on devices later on in the process chain. An example of an implemented two dimensional model by use of older technology and a not so clear design can be seen in Figure 4.1, while Figure 4.2 shows a two dimensional model created using more modern technology and with a better design.

One such system for creating a two dimensional graphical model, for displaying a factory's process, is Siemens' Simatic PCS 7 [41]. This model uses virtual graphical objects on the screen, to represent the physical equipment in a factory. Simatic PCS 7 belongs to a category of applications known as Distributed Control System (DCS) [42]. A DCS-system is a system where the programmable logic controls have been decentralized and distributed throughout the process control system, but with a central point of operating the system by use of computers. A DCS-system is typically used in larger industry plants, where it offers advantages over traditional systems, which consists of only Programmable Logic Controllers and Human Machine Interface control devices. Other DCS-systems

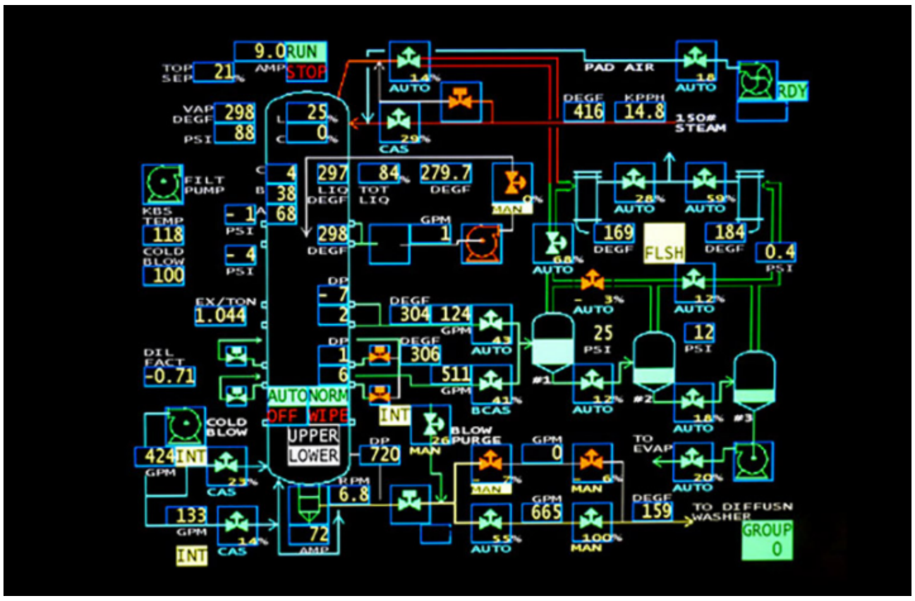


Figure 4.1: Early HMI [40]

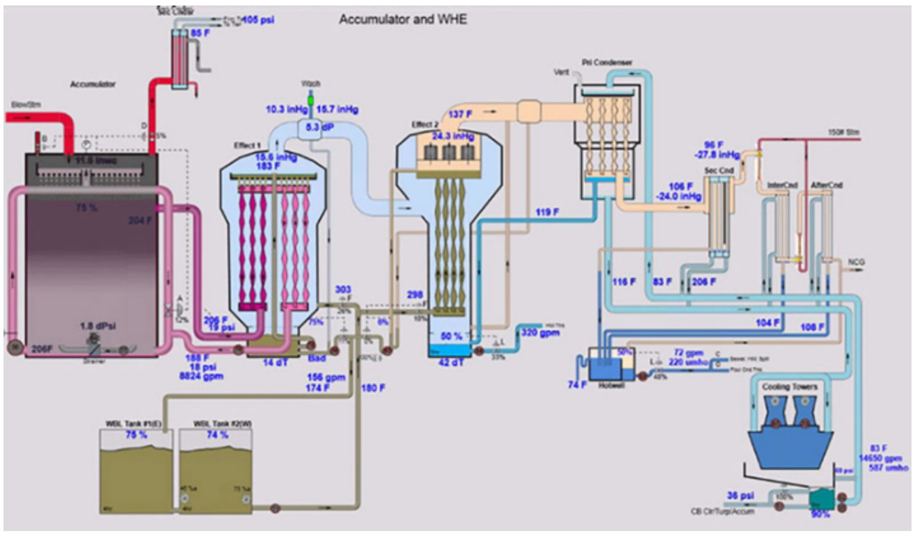
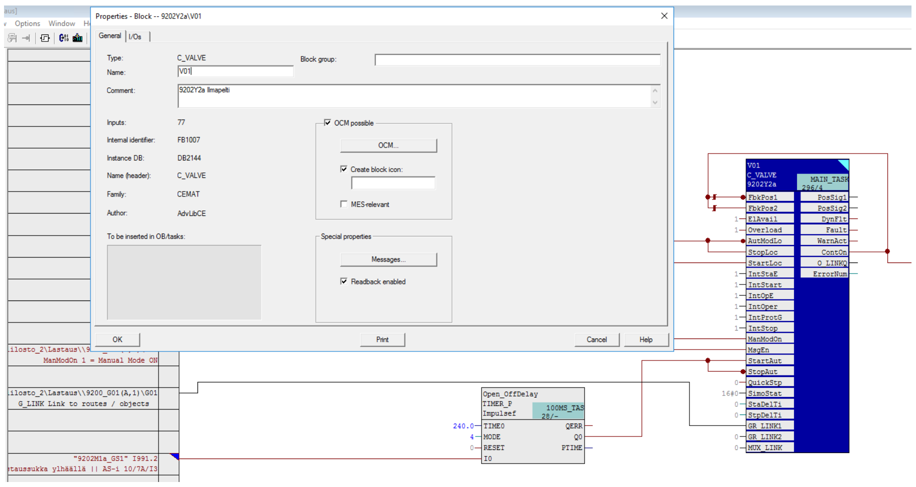


Figure 4.2: Modern HMI [40]

include ABB DCS [43] and Valmet DNA [44]. The ability to graphically configure a two dimensional view for a process is however not only unique in DCS-systems, but instead found in almost all Human Machine Interface and SCADA Software applications.

A difference, regarding the graphical visualization capabilities, between a DCS and a traditional SCADA system, is that the screens of the process can to some degree be automatically generated in the DCS system. In the DCS system, an engineer, creates a hierarchy for the processes and equipment in the factory, and uses template based models for automatic generation, of among other things, the graphical elements that represent the equipment in the process. In other words, a DCS system makes use of an object oriented view, where type definitions are created for factory equipment, and from which instances of the type are then created and configured. This allows for features such as inheritance and re-usability. Figure 4.3 shows an editor called CFC (Continuous Function Chart) from Simatic PCS 7, where an instance of a device is being configured. The type of device shown here as an example, is called ”C_VALVE” and it has the ”Create block icon” check box set as active. This setting will create a graphical icon for the device in a control screen, once the project is compiled. The control screen for the instance is determined by its location in the defined factory hierarchy.


Properties - Block -- 9202Y2a\V01

General | I/Os

Type: C_VALVE
Name: V01
Block group:
Comment: 9202Y2a Ilmapelti
Inputs: 77
Internal identifier: FB1007
Instance DB: DB2144
Name (header): C_VALVE
Family: CEMAT
Author: AdvLibCE
To be inserted in OB/tasks:

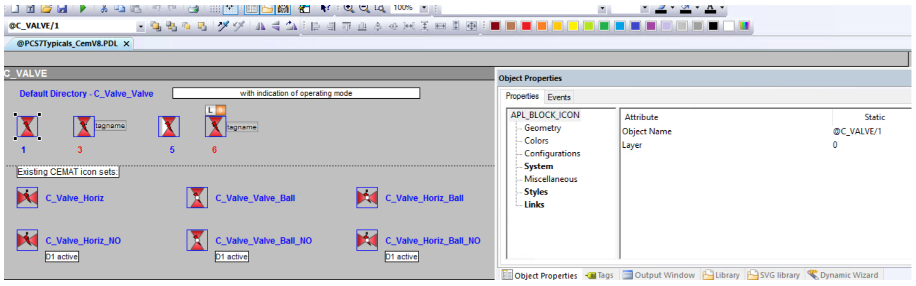
[✓] OCM possible
[✓] Create block icon:
[ ] MES-relevant

Special properties
[✓] Readback enabled

Figure 4.3: Configuring operation of a valve, type C_VALVE

Figure 4.4 shows the graphical icon for type C_VALVE, that will be used as a template, when creating a new icon for the instance of the device, that was being configured in Figure 4.3.

The type of model that a DCS system provides, allows for a more efficient way to model a factory context, compared to other solutions. There is a threshold involved



Attribute	Static
Object Name	@C_VALVE/1
Layer	0

Figure 4.4: Type C_VALVE as a graphical object made for visualization

though, as use of this type of modeling requires more initial work, resources and added complexity, but which will be worth it later on in the project by reduced engineering time. This is why a DCS system is more suitable for use in larger systems and why more traditional SCADA- and HMI-solutions are used in smaller projects.

4.2 How to do estimation in a factory context

A process in a factory is based on a chain of devices that together perform operations that result in a product being created. The performance of one device is therefore dependent on the nearby devices in this chain.

Figure 4.7 shows part of a screen from a typical HMI panel or SCADA screen, used to monitor and control a factory process. In this figure, a conveyor belt (for moving of material) is displayed by a symbol shown in Figure 4.5, which uses an electrical engine to move, displayed by the symbol in Figure 4.6. The conveyor belt can be used as an example for which to do estimation in a factory.

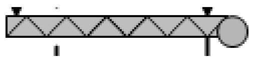


Figure 4.5: Symbol for a conveyor belt

In order to estimate the use of energy and amount of CO_2 emissions for the conveyor belt, one would first need to know the length of time that the belt has been operating. As it is the electrical engine that makes the conveyor belt move, it is in this case the

Figure 4.6: Symbol for an electrical engine

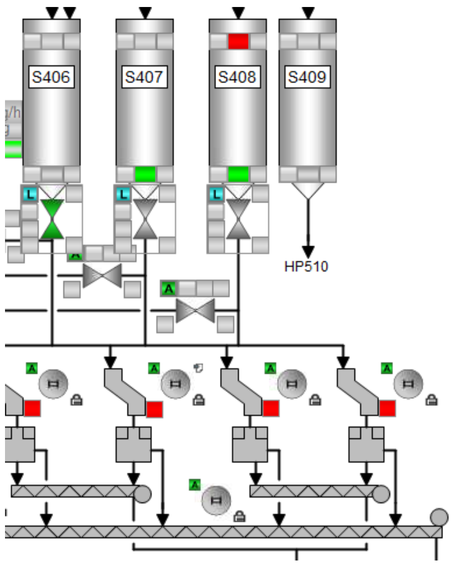


Figure 4.7: Part of a factory process HMI screen

length of time of active operation of the engine that is of interest. The automation system, based on programmable logic controllers, is controlling the process and is connected to the HMI/SCADA application, by use of a network connection. The run-state information for the electrical engine can be made available through the network, for use in estimating energy use and CO_2 emissions.

As stated earlier, on/off information and length of operating time, is not enough to do an accurate estimation - the equipment load must also be known, as a greater operating load consumes more energy, which may cause increased CO_2 emissions. This is why the supplementary data, regarding the load on the electrical engine, must be included in the recorded data for estimating the CO_2 emissions. The available data regarding the load factor, can be available in different forms, depending on the process. For the conveyor belt, the load data may be in the form of knowing if the equipment in the process chain prior to the conveyor belt is running or not, and then used to assume based on this, that the load on the conveyor belt is either 0% or 100%. The input parameters needed to estimate the used energy (which can be converted to caused CO_2 emissions) for the electrical engine would in this case be:

- The electrical engine's specifications needed for estimating energy use, such as the maximum power output of the engine (kW)
- The electrical engine's on/off status (to establish the running time)
- The other equipment's on/off status, prior in the process chain, known to have an effect on the load of the electrical engine (to establish the time frame where this equipment is running and causing a load on the electrical engine)
- The load factors on the electrical engine, at the time when the other equipment prior in the process chain is respectively on and off

In another example case, the conveyor belt could be preceded by a weight scale, used for measuring the passing raw material stream, in roughly equal amount of batches. In this case, the estimation of used energy for the conveyor belt could include the weighted material amount and the timestamps for each scale measurement. This information could be used to accurately know the amount of material passing by on the conveyor belt, in other words the real material load for the electrical engine. The input parameters to estimate the energy use for the electrical engine would be:

- The electrical engine's specifications needed for estimating energy use, such as the maximum power output of the engine (kW)
- The electrical engine's on/off status (to establish the running time)
- The weight scale's material measurements completed with timestamp data.
- A factor or conversion formula, used to convert the amount of material passing by on the conveyor belt, in a given time frame, into a load factor between 0% and 100% for the electrical engine.

The estimation of energy use (and the conversion to caused CO_2 emissions) can be made more exact, based on how much and what kind of supplementary information there is available regarding the use of the equipment, for which the estimation is done.

In the first case above, the most simple and inexact estimation, could be done only based on the on/off data of the electrical engine. The energy use would be considered either with almost no energy usage (off) or energy use based on full maximum output (on).

In the second case, additional data would be available, by knowing when the equipment prior in the process chain is running, and consequently having an effect on the load of the electrical engine. This additional load data, would allow for a more accurate estimation, as it would be known when a load is applied for the electrical engine and when there is no load.

The third example case from above, would enable the most accurate estimation of these cases. Here, the actual load on the conveyor belt and the electrical engine, would be measured as material weight, which would allow, together with the on/off state data, to calculate the variable caused load for the electrical engine over a given time frame. The variable load data, could in these example cases be used to calculate the most accurate estimation of used energy and caused CO_2 emissions.

Each factory process is different and the available supplementary data used to establish the load for an equipment, by which energy use is to be estimated, varies in terms of quantity and quality. This supplementary data has a direct effect on the accuracy of the estimation result.

As the cases above illustrate, a dynamic system to model and display caused CO_2 emissions, must be constructed in such a way, that the input parameters, the method of load calculation etc. are configurable for the equipment. The same kind of equipment type (e.g. a conveyor belt), can have different kind of supplementary data available for load calculation, even within the same factory. Therefore, a true dynamic system for estimating caused CO_2 emissions, must be made possible, by allowing the user of the system to create new equipment types and allowing addition of input parameters and conversion formulas.

4.3 The source, the connectivity to, and structure of the available data, for measurement and estimation of CO_2 emissions

There are a lot of different types of automation equipment and machinery used in factories, that can provide a wide range of data output. What kind of data, depends on the type of machinery and of the implemented monitoring capabilities. Some of the more basic types of equipment might provide output data only in analog format, such as a milli Ampere signal between 4 and 20 mA [45], or basic electrical relay data, by the ability to connect to an on/off relay supplying a voltage level when connected [46]. As examples, a 4-20 mA current can be used as an input signal to indicate the level of liquid in a tank (4 mA = tank is empty, 20 mA = tank is full) and a voltage on/off input can be used to indicate when an electric engine is running (feedback data).

More sophisticated equipment can provide output data in a digital message-based form. Two standardized communication interfaces belonging to the category of field buses, commonly used between the equipment and the PLC, for message-based data, are Profibus [47] and Modbus [48]. Today, field buses are on a decline and are increasingly being replaced by Industrial Ethernet solutions, such as ProfiNet [49]. Both these types of communication interfaces are specifically meant for use in a factory context, emphasising low latency and secure delivery of the data. The Industrial Ethernet technologies are however becoming the standard to use [50], over the field bus technologies, because of their advantages, such as higher bandwidth, compatibility with standard Ethernet technology equipment and more flexible data exchange mechanisms [51].

The PLCs can be equipped with both built-in and optional communication modules, that let them communicate with the automation equipment, over these interfaces. PLC modules are also available for connecting analogue signals such as voltage on/off input or mA input, by use of so called I/O modules.

Part of programming a PLC, is to configure its hardware setup. Analog input signals can for example be configured to scale according to given parameters, while digital on/off input connections need almost no configuration. The communication data received from a networked interface, will require more configuration. The equipment sending the data over the network interface might first need to be configured, for example regarding the quantity of information that it will supply to a PLC. This is because, there is not always a need to send all available data, if not all data is of interest in a certain factory context. Secondly, the PLC's hardware setup must be configured, so that it will know the size of the communication message that it will be receiving from the equipment and the structure of it. The communication message often consists of one or more WORD-constructs (where one WORD equals two Bytes), or an array of WORD-data. The PLC must then be configured to recognize what parts a message is built of, for example that bit 2 of the first WORD in the message, means "Engine Running" [52].

The PLC programmer, the engineer, is the one that must configure this hardware setup, before the data can be used as part of the PLC program. After configuration, the data will be available as easily accessible data points in the PLC's memory. These data points are commonly called "*tags*" and are available as variables for use in the PLC's program code, with their values refreshed in either a cyclic or acyclic manner. The tags have their values defined in the PLC's memory according to the PLC data type that is most fitting. As an example, an analog 4-20 mA signal at source, could be mapped to a tag of data type FLOAT in a PLC, that would contain a decimal value representing the liters of liquid in a tank (based on the scale parameter converting the signal to a float data type). Another example, could be a thermal relay's signal for a triggered state (indicating that it has cut power to an electrical engine, in order to prevent overheating and damage to the engine), which could be mapped to a BOOL variable in a PLC, with the possible values of 1 and 0 (or True and False).

These phases described above is the concept for how the signals from the factory equipment are sampled, communicated, converted to a known data type structure and their values made available for use.

4.4 Accessing the data

External access to the data in a PLC can be accomplished by many different means. The physical medium for access is the first choice to make. Some of the possible ways are:

- memory cards that can be connected to a PLC, after which the PLC can be configured to record data values saved to the memory cards
- network access to the PLC data using standard Ethernet technology [53]
- serial communication, for example RS-232 and RS-485 [54]

Serial communication is fast becoming a thing of the past, regarding external access to a PLC's memory (because of the many limitations compared to Ethernet networking). Saving of data onto memory cards is fully supported by many PLCs, but is inconvenient for a system that displays data in near real-time, as the memory card will need to be physically moved from the PLC to a data receiver, in order to access the data. This leaves the Ethernet connection as the most suitable communication medium, as it offers advantages such as high bandwidth, ability for many client applications to connect simultaneously and no need for physical access to the PLC (once configured).

The second choice to consider is how to communicate over an Ethernet connection. Most PLCs use the TCP/IP protocol to reliably manage the data stream over a network connection. A PLC has a native communication protocol for communicating over the network, which often is proprietary and might not be freely available for public use. Some PLCs provide access to an FTP-server (File Transfer Protocol) [55], where data values can be made accessible through saved files (for example in CSV, Comma Separated Values, formats) [56].

Most larger factories have SCADA applications installed on computers for controlling and monitoring of the factory processes. These applications often use the PLCs' native communication protocol for communicating with the PLC, which often is the most efficient and full-featured protocol for exchanging data. The SCADA applications in turn, usually have other built-in services, which provide access to the PLC data, to other external PC applications. Some older services providing data access, include DDE [57], OPC DA [58], OPC A&E [58], OPC HDA [58] and Suitelink [59]. Some SCADA systems can also be configured to store data from a PLC to a SQL database [60], either through use of some proprietary technology or by script code added to the SCADA project by the engineer.

There are also third-party solutions available, like the remote monitoring and data recording hardware devices, which save the data to their own cloud-based services. Two such vendors are Red Lion [61] and Ixon [62].

To truly be able to construct a dynamic model, that would work in any factory, would require that the method of connecting to the PLC's data points is standardized and widely supported. Problems to support this requirement are present with many of the mentioned methods. Some such problems are:

- DDE is a technology, for which development stopped in 1996 and as such not widely supported any more
- Using data files retrieved from an FTP server does not enable the real-time functionality sought for and would require custom-made parsing of file data for each factory

- Using third-party remote hardware solutions would require each factory to purchase a certain vendor's product and also require the factory to pay monthly fees for the use of this service
- OPC DA, OPC A&E and OPC HDA are still widely supported by many producers of automation equipment but are based on the old DCOM-technology by Microsoft, which is difficult to use over networked connections and usually only supported on Windows operating systems

Today, there is one platform independent architecture standard for exchanging data between devices, that most automation vendors and other interested stakeholders support and develop, which is called OPC UA (short for OPC Unified Architecture). OPC UA is maintained and developed by the OPC Foundation. [63]

The OPC Foundation is an organization with members from major leading companies around the world, such as Mitsubishi Electric, ABB, Siemens AG, SAP and Rockwell. A full list of current members can be found from OPC Foundation's web site. [64]

The mission statement for the OPC Foundation is "*to manage a global organization in which users, vendors and consortia collaborate to create data transfer standards for multi-vendor, multi-platform, secure and reliable interoperability in industrial automation*". [65]

The OPC Foundation was founded in 1996 and has since then been working on specifying a standardized open cross-platform enabling communication technologies. The OPC Foundation was formed originating from a task force, consisting of automation vendors Fisher-Rosemount, Intellution, Opto 22 and Rockwell since 1995. The task force created the first OPC specification version 1.0, based on Microsoft's COM and DCOM technology. The word OPC comes from "*OLE for Process Control*", OLE being Microsoft's Object Linking & Embedding technology [66]. It was soon realised, that an organization would be needed to oversee and manage certification, compliance, interoperability and validation of matters concerning the standards of the data access architecture. The OPC architecture standard developed over the years and OPC UA version 1.0 became available in 2006, as a way to tackle issues present because of the COM/DCOM technology used in the OPC architecture. Today, the OPC Foundation keeps growing by continually adding more members, and keeps up the primary task of developing the standard architecture technology for data access. [67]

OPC Unified Architecture has become a de-facto open standard for exchanging information between automation systems. The architecture uses an approach of many layers, which include

- delivery of the same funcionality as class COM-based OPC specifications
- is a specification for standards, enabling it to be platform independent (anything between small micro-controller devices to cloud based systems can use it)
- enabling security by incorparating encryption, authentication and auditing
- extendable by enabling a way to add new features without breaking existing applications
- ability to model information, in order to be able to define complex data
- different transport protocols are supported, such as binary transport and JSON

The technology specification also enable features such as OPC UA server discovery on a network, hierarchical data presentation for OPC clients, read/write permissions on data, a subscription based data exchange model and event notification based on an OPC client's set criteria.

OPC UA has mostly been used at the SCADA/HMI level, but is with its continued development becoming more suitable for data exchange between automation equipment such as PLCs and other IoT-enabled devices, while at the same time also expanding with functionality for easier access to and integration with cloud-based systems. OPC UA is playing an important part in the ongoing Industry 4.0 transformation, the digitization of manufacturing, as factory owners are looking to connect their plant based machine data to the cloud, which can enable flexibility in production and meeting the increasing expectations of their customers. As a standard secure full-featured data exchange method, OPC UA lowers the threshold for manufacturers to digitize their factories. [68] [69]

For accessing data of automation systems, there is in 2021 no other data exchange method, that would offer any advantage over OPC UA. This is true, at least when connecting to any modern automation system (as some older systems do not and will not be updated to support OPC UA).

This is why, OPC UA was chosen to be the method for accessing the automation data needed for measuring and estimating energy use and caused CO_2 emissions.

4.5 A model for creation of a dynamic system, displaying CO_2 emissions

Displaying CO_2 emissions in real time, from a factory context, is most probably best modeled and shown using a similar configurable system as those used by DCS systems. This type of graphical model for visualization has been proven suitable and successful already dating back decades, in regards of displaying factory process information and for controlling the processes [70] [71].

A conceptual illustration, for the main building blocks used to create a dynamic model for displaying real-time CO_2 emissions, can be seen in Figure 4.8.

The model includes components used for creating and configuring virtual counterparts for physical equipment in a factory, for which CO_2 emissions should be monitored. The equipment types created must have basic configuration parameters set, such as a descriptive type name and parameters added for use in measuring and estimating of consumed energy and caused CO_2 emissions. These parameters will be added and defined during

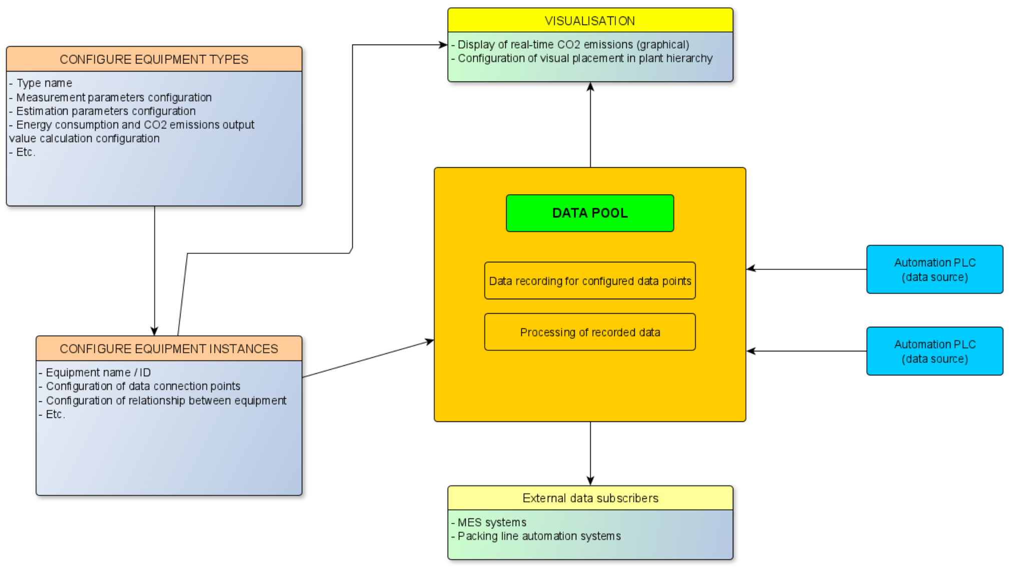

VISUALISATION
- Display of real-time CO2 emissions (graphical)
- Configuration of visual placement in plant hierarchy
CONFIGURE EQUIPMENT TYPES
- Type name
- Measurement parameters configuration
- Estimation parameters configuration
- Energy consumption and CO2 emissions output value calculation configuration
- Etc.
DATA POOL
Data recording for configured data points
Processing of recorded data
Automation PLC (data source)
Automation PLC (data source)
CONFIGURE EQUIPMENT INSTANCES
- Equipment name / ID
- Configuration of data connection points
- Configuration of relationship between equipment
- Etc.
External data subscribers
- MES systems
- Packing line automation systems

Figure 4.8: Building blocks for creating a dynamic display model

the creation of the types. Parameter definitions can include for example the type of data for input signals and the computation of output values, based on the input signals. A graphical representation of the equipment is also included in the type configuration.

After creating an equipment type, the type can be used as a template in order to create an instance of the type. It will be possible to create as many instances of a type, as is necessary to visualise the factory's equipment. Relationships and dependence between equipment can be configured for an equipment instance, by connecting of inputs and outputs between equipment instances. The input data signal points of each equipment instance, are configured to read the input data from a data source, which provides the data from the factory automation system.

The data pool component of the model, will make use of the configured instance data, to record the required data points and to perform computation on the data, according to the type configuration data for output calculations. The data will be recorded by reading live sensor input data from the factory's automation system. The recorded data and the computation result will be made available to the visualization component in the model, while also being available to other third-party systems making use of the recorded CO_2

emission data.

The visualization component will be configured for a two-dimensional layout, that confirms as best as it can, to the physical layout of the equipment in the factory. The visualization component should also provide interactivity for the operator of the system. Detailed information for the displayed equipment instances could be shown in separate ”pop-up”-dialogs, provided by the visualization system. Also, the functionality, to navigate between the different parts of the factory, by selecting screens, would be provided.

The element of the model, where the user creates and configures equipment types, to be used as templates for equipment instances, is not necessary for determining if it is possible to create a system for showing CO_2 emissions in real-time, in a factory context. This part of the model is important, at the phase when the system is proven to be working and usable, and where one is modeling large parts of a factory. Therefore, this part of the model is not considered vital as a study in the research, but more as a part of the model to be developed into a final solution for the system, in order to enable a truly dynamic and efficient system.

Chapter 5

Synthesis of solution

5.1 Modelling of factory energy consumption

Displayed values of caused CO_2 emissions are based on either straight measurement readings, on values calculated from related measurement recordings or on estimated values based on configured rules and formulas. In all cases, the type of equipment, which is monitored, must be created and configured in the system. The system must also allow for a model to be created of the factory context, in order to define the relations between equipment and the influence they have on each other. The relationships will be defined by the connections of inputs and outputs, made between instances created in the model.

The model from Section 4.5 shows the following building blocks:

- Configure equipment types
- Configure equipment instances
- Data Pool
- Visualization
- Automation PLC (data source)

- External data subscribers

From these building blocks, it can be concluded that a tool needs to be constructed for the configuring of equipment types and equipment instances. The Data Pool is a background-type service component, used for storing and processing of the recorded measurement and estimation data, based on the configuration of the system. The visualization component is used for displaying the results of the Data Pool service's operations and the overview and context of the factory processes. The PLCs might also need some configuring, to allow access to the necessary data, but this is outside of the scope of this thesis, as the brand and model of each PLC differs widely, and as such offers different kinds of interfaces used for configuration. Data from the system is made available to third party software, through definition and creation of callable Application Program Interfaces.

All of these components would together as a whole make up the system for displaying caused CO_2 emissions in a factory context. It would make sense to create the system as a group of components, instead of as one single application, as each has its own specialized job description. In practise, the configuration of types and instances, could be handled by a component with a specialized user interface made for the task. The processing of input data and generation of output data is a task, which does not need operator input, and could be handled by a background service component. The Data Pool background service would also host the interfaces for supplying data to third party software. The visualization of the factory context and the caused CO_2 emission values, would show the output result from the component handling the background tasks, and could be handled by a component with a user interface specifically made for this, separate from the configuration component's user interface.

5.1.1 Equipment types and instances

The modelling of a system starts with the creation of equipment types. The word ”equipment”, as used here, can mean a single device, part of the factory process (such as an electrical engine) but can also mean a group of devices, that as a whole make up a single entity, for which the CO_2 emissions should be monitored. It is up to the engineer to decide the level of information detail when configuring an equipment type.

The type definition must allow for a lot of flexibility and adaptability, as each factory will have different kind of equipment. Also, the way the equipment is used in a factory can have an effect on how the type is defined, in order to be able to obtain the necessary values from it. The configuration of equipment types would need to include at least:

- assigning of a unique equipment type name
- assigning of equipment to a certain category of equipment (for convenience when defining of equipment instances)
- separate configuration sets for measurement and for estimation of caused CO_2 emissions
- ability to add and remove equipment property values for both input and output purposes
- ability do define rules and formulas to calculate the output values based on provided input values

The configured equipment types would be used to create equipment instances. The creation of an instance would start by selecting a type. Additionally, a factory process hierarchy should be defined, and the instance added to the correct hierarchy location. The hierarchy would be helpful in maintaining the project structure, as a factory can contain hundreds of equipment. The hierarchy would define the logical structure of the factory. The engineer would make the decisions for how the hierarchy is set up, but a typical hierarchy starts by adding the main factory process areas, then the sub areas where certain defined process operations are being carried out and perhaps additional segments for the sub areas. As many levels can be added, as found to exist physically and logically in the factory context.

The main benefits of a process hierarchy is to configure a structure that helps to manage the project's content but it could also be used to automatically generate relationships between screens for the visualization component, based on the relative location of equipment in the project hierarchy.

5.1.2 Instance: Production batches; energy used & CO_2 footprint

A valuable use of caused CO_2 emissions data in a factory context, would be the possibility to connect this information with data regarding produced product. Production data such as produced orders, produced products, produced quantities and production times is often available for factories. This production data is maintained and reported by MES software systems, which acquire the reported production data by communication with the factory's automation system. There is value in being able to report the total amount of CO_2 emissions caused by the production of a certain batch or even a single product, as for example being able to market a reported low level of caused CO_2 emissions for a certain produced product. This data can for example be marketed by printing the caused CO_2 emissions value on the production package, for example with a label "*Only x grams of CO_2 emissions were emitted while making the orange juice container you now hold in your hand!*".

The information regarding the total amount of caused CO_2 emissions for a particular production batch of a some product is usually not available in most factories. The factory might be able to estimate the total amount of caused CO_2 emissions over a month for all of the factory's operations, but not able to with a high enough accuracy say how much CO_2 emissions a certain production batch has caused.

Connecting the information between a MES system and the planned system for display of CO_2 emissions would most likely be possible. Most MES systems make use of

API's in many forms for communicating with outside systems, such as ERP and automation systems. These interfaces could be used to transfer recorded CO_2 emissions data to a MES system and the link between the production batch and the CO_2 emissions done for example based on timestamp data. The MES system could then create report data, including CO_2 emissions, tied to the produced production batches.

Production of a product is often done separately from packaging of the product, which would allow for recording of produced CO_2 emissions data, that later could be used in the packaging area. The MES system could send the received CO_2 emissions data to the packaging automation system, based on which production batch is being packed. This would allow the packaging equipment to print the caused CO_2 emissions data as labels onto the products.

The caused CO_2 emissions data per products and production batches could also be of value for the ERP system, which could receive the data from a MES system, and use it to produce factory and company wide reports tied to the company's products and production operations. One use of this data on this level could again be for marketing. The data could also be used to show that the company is adhering to goals set for fighting climate change etc.

There are probably many other valuable uses of detailed caused CO_2 emissions data, when connecting the data to produced products and production batches and this could be one the most valuable reasons for a factory to use the system to display caused CO_2 emissions.

5.2 Construction of an application

5.2.1 Data Recording

As concluded in Section 4.4, the technology most suitable for data access is OPC UA. The OPC UA standard is a freely available set of specifications for data access communication. This means, that it is available for anyone to implement as a technical solution. The standard is however comprehensive, covering topics such as different data access methods, discoverability and security matters including authentication and encryption. For these reasons, there are several ready made third-party solutions on the market for shortening the development time by providing OPC UA functionality in forms of components to use in applications. Most of these third-party solutions are not free and are sold under different types of agreements.

Some of the available third-party solutions for OPC UA data access are:

- Integration Objects' OPC UA Client Toolkit (for .Net software developement) [72]
- Unified Automation's OPC UA Client SDK (for .Net, Java, C++ and Delphi software developement) [73]
- OPC Labs' QuickOPC for OPC UA (for .Net, C/C++, PHP, Delphi and Python developement) [74]
- Prosys' OPC UA SDK (for Java, Delphi, C++ and .Net development) [75]
- node-opcua (free open-source OPC UA library for NodeJs) [76]

Besides the open-source option (available on GitHub) for NodeJs development and these commercially available options, for incorporating OPC UA client functionality, exists a project called "UA .NET and UA .NET Standard Stack". This is the official reference for how to build OPC UA applications (built using the .Net Framework in C#), provided by the OPC Foundation. The project includes sample applications and an OPC UA stack for core functionality. The project is maintained by the OPC Foundation and the open-source community. The project is available as a GitHub project, freely downloadable by anyone. [77]

The official reference OPC UA client code was selected as the technique for accessing OPC UA server data, as this represents the official reference OPC UA client and has the code freely available for inspection and public use. The required functionality would be modeled from the reference code, and adapted according to the needs of the system for displaying caused CO_2 emissions. The decision to use the reference code, instead of a third-party solution for data access, was so that the system would not be dependent on a commercial component.

The configuration for the connections to the data sources, the OPC UA DA (Direct Access) servers, should be configured in a general settings editor for the system. This settings editor should be separate from the other configuration functionality of the system, such as configuration of equipment instances or visualization screens.

The data points to sample and record would be defined through the definition of equipment instance types. The engineer should be able to select a data point from a connected OPC UA DA server, for all required equipment instance input data. The allowed data type for a selected data point, would be restricted to those types that were configured for the input, during the creation of the equipment instance type. As for the sample frequency and data recording frequency of the data point, there could be several working solutions, of which some are:

- Sampling and recording of all input data points in the system at a standard interval
- Individual settings for sample and record intervals for each equipment instance's data points
- Sampling and recording of values upon change

It might be difficult even for an engineer to be able to determine the needed sampling and data recording intervals for any given data point. The optimum intervals would be such, that all events of change in an equipment, needed to estimate the caused CO_2 emissions, are sampled and recorded, while still not sampling and recording the states in between where no change has taken place. The engineer might eventually, through observation of the results, be able to find near-optimum sample and data recording frequencies for each equipment instance, if the these interval values could be set separately for each input data point. This would increase the work load for the engineer, also resulting in increased costs for the project.

Another solution might be to sample and record all data points in the system at standard intervals and to then run post-processing on the data, in order to remove unnecessary recorded data points (where no change has occurred). The negative effects of this solution would be an increased load on the system, because of the increased processing demands caused by the high sampling and data recording interval for each data point in the system. The post-processing of the recorded data would further increase the processing power demands. An intelligent system might be able to dynamically adjust the sampling and data intervals for each data point, according to analysis done on recorded data. A dynamic system like this, would still always risk missing out on some change in state, that could lead to a decreased accuracy of the system.

As this is a research project, accuracy is valued more than optimization of system resources, and thus the implementation could be done by doing data input point sampling and recording according to standardized frequency values. Other more optimized solutions could be implemented later, when other main functionality of the system is ready and proven to work.

5.2.2 Data Storage

The type of solution for saving of the recorded data, that will be utilized when measuring and estimating caused CO_2 emissions, must be selected according to the needs of the system and according to what is most suitable. The saved data will need to be queried according to different search criterias, such as the time of the recorded data and the relationship between different equipment. Some of the most common ways of storing data include:

- binary files according to custom created structures

- text files with custom created data structure
- XML files with custom defined schemas [78]
- relational database products (SQL databases) [79]
- NoSQL databases [80]

The first two options would be very time-consuming to use, as they would require the creation of a new custom-made system for saving and querying of recorded data. These two options would be more suitable for a system, that would make use of a small amount of recorded data and where the performance of querying the data is of less importance.

The amount of data recorded for the system, will likely be great, as the sample record frequency might need to be as fast as one value per second, in order to be able to show near real-time calculations of caused CO_2 emissions. The structure of the recorded data will be known, as OPC UA DA is used for communication, and its nodes have a pre-defined data structure, according to the OPC UA standard.

XML files allow the definition of an XML schema, to which the content of an XML file must adherd. This will enable a pre-defined structure for the recorded data. A SQL database works in a similar way, requiring schemas to be pre-defined for the recorded data. A key difference between XML data files and SQL databases, is that SQL databases use a dedicated application for handling the data, and this application is optimized to perform effeciently by use of for example indexing, concurrent data access and relational definitions between data. The use of XML files would require the programmer to by themselves implement an effective system for creating, reading, querying and managing the recorded data, which would require a significant amount of extra work - compared to using a SQL database for the recording of data.

So called ”NoSQL” databases have gained popularity in the last decade or so, and continue to do so. NoSQL databases are suitabale for situations where the structure of the recorded data has frequent changes, or where the amount of recorded data is on the level that can be defined as ”big-data”. NoSQL databases are for example suitable for document type of data, which could be for example a blog post on a social media platform, containing data such as blog text, photos, tag data and location based information. This data is easy to save as a whole in a NoSQL database.

Traditional relational SQL databases are on the other hand suitable for situations where the data structure can be pre-defined to a large degree and where the data itself is used from queries performing operations on it, based on for example a certain time frame or relation between different parts of the data.

There is no advantage in selecting any of the first three listed options for the system that will display caused CO_2 emissions. Binary files and text files, would both require a large amount of work for defining an effective data recording structure and methods for effective handling of the data. There is no point in creating these from scratch, when there are other ready-made solutions available.

This leaves the choice to use a traditional relational SQL database or a NoSQL database solution for storing data. The exact amount of data input points stored is not known in advance for a project, as the engineer will define a variable amount of different equipment instances, with each equipment type containing a different amount of inputs. The type of data stored is however not complex, as the data will mostly be numeric values, such as for example on/off states, straight power consumption readings and time stamps. There might be a need to sometimes associate recorded data points with each other, to form a so called ”snapshot” of a certain situation, but it is more likely that the data points are usually saved as individual data records with some standardized accompanying data such as a time stamp. The saved data for the system is likely to have a structure that does not change often. This known structure of the data and the fact, that the system will run queries and filtration on the data to obtain information for calculating caused CO_2 emissions, would indicate that a traditional relational database will work as a data storage for the system. The system does not either have the immediate need for horizontal scalability across servers or the need to flexibly adjust the type of saved data, both two features that NoSQL databases usually offers. [81]

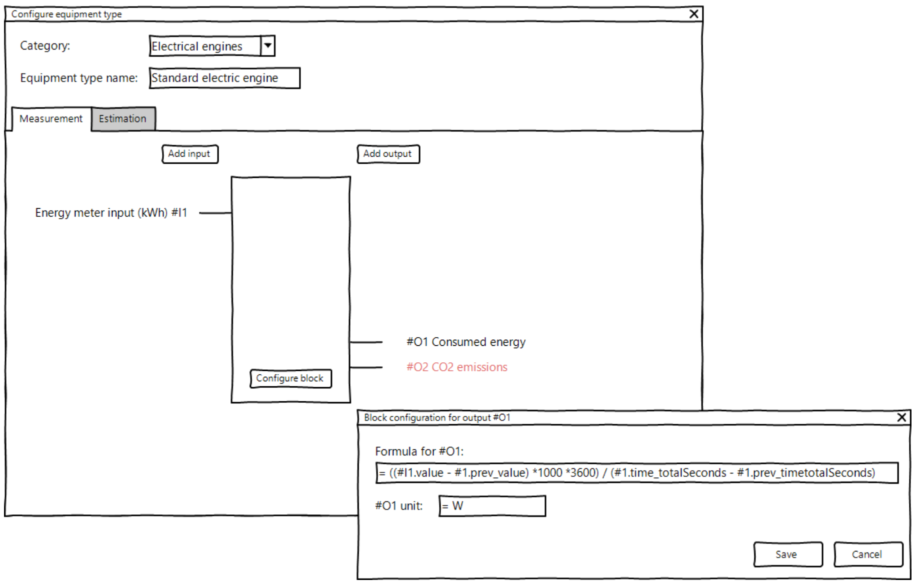
5.2.3 Component communication - exchange of messages and data

The model of the dynamic system in Section 4.5, Figure 4.8, has at its center the main component of the system, named ”Data Pool”. The Data Pool component would contain the core functionality of the system for displaying caused CO_2 emissions, that would tie together all components in the system. The system would be made according to a service oriented architecture (SOA) [82], made up of publishers and subscribers.

The ”Data Pool”-component would act as a subscriber of information from sources with access to the factory’s process data and as a publisher for visualization clients and other external data systems, such as MES systems. Although here named as ”Data Pool”, this component would do much more than just hold recorded data. It would be responsible for storing recorded process data to the data storage and to manage the data but also to perform all necessary background processing, as configured for the system’s equipment instances. The necessary background processing would be determined by the configured equipment instances. The instance configuration would hold data such as what process data points to record and how to use the recorded data to establish the power usage and caused CO_2 emissions for the equipment.

Figure 5.1 shows a mock-up sketch of what functionality a basic dialog window could contain, used for configuring an equipment template. The equipment template would be configured to output the caused CO_2 emissions and if necessary to reach this output by transforming the available process data into the correct format or to estimate the caused CO_2 emissions by use of mathematical formulas, when direct measurement is not available.

The configured data for equipment templates and instances would be stored in the data storage for the system.


Configure equipment type

Category: Electrical engines

Equipment type name: Standard electric engine

Block configuration for output #O1

Formula for #O1: = ((#I1.value - #1.prev_value) *1000 *3600) / (#1.time_totalSeconds - #1.prev_timetotalSeconds)

#O1 unit: = W

Figure 5.1: A sketch of an interface to configure a type of equipment

The reading and saving of data between the configuration editors and the ”Data Pool”-component would be done by establishing a connection, exchanging data according to a custom protocol and then disconnecting once ready. The communication between the ”Data Pool”-component and the sources for process data, the visualization clients and other external clients could be based on both events and polling procedures. Event-based communication would be preferred, as this does not waste resources when no data exchange is needed.

There are many technologies that could be used to implement the communication between publishers and subscribers in the system. Some of the options include:

- Windows Communication Foundation (WCF) [83]
- Microsoft Message Queuing (MSMQ) [84]
- NodeJs [85]
- .NET Core/ASP .NET Core [86]

In today’s Internet-connected world with cloud based services, web technologies have become a way to connect different systems, even systems running on different operating systems. Theses technologies are no longer tied to a certain operating system or server manufacturer and therefore provide more flexibility and compliance with a wide range of systems. This is why native proprietary technologies should be avoided when creating the communication between the components for the system displaying caused CO_2 emissions. From the list above, Windows Communication Foundation and Microsoft Message Queuing are technologies tied to Windows operating systems and should as such not be used here.

NodeJs is based on JavaScript language and interprets code by use of Google's V8 JavaScript engine. .NET Core/ASP .NET Core is created and maintained by Microsoft and C# is used as programming language. Both of these technologies are modern, in continuous development, have excellent performance to scale and serve large amount of simultaneously connected clients and offer all the functionality required to create a publisher/subscriber-based application. Both can also be run on more than one type of operating system and both are also free to use. The company Microsoft is managing and developing .NET Core/ASP .NET Core, while the OpenJS Foundation [87] has a license on NodeJs. NodeJs can also support static programming and other features by use of TypeScript [88], which add additional syntax to JavaScript. TypeScript is developed and maintained by Microsoft. As of 2021, tests have shown that .NET Core/ASP .NET Core has a slight performance lead over NodeJs, at least for larger applications. [89] [90]

Both technologies could be used to implement the ”Data Pool”-component successfully and the choice comes down more to a personal preference, than one technology being superior over the other. A choice to use .NET Core/ASP .NET Core for the system to display caused CO_2 emissions was made, based on the preference to use C# as a programming language and the support of a commercial company behind the selected technology.

5.2.4 Visualization

As discussed in Chapter 4, visualization of real-time data from physical equipment in a factory context, is best visualized through a representation of the data by use of 2-dimensional graphics. The data to visualize must be readily pre-processed, so that the visualization functionality is focused solely on visualization. A first decision to be made, when considering the technology to use for visualization, is if the visualization should use a native or a web-based technology.

Native technologies implement a solution using the functionality made available by native operating systems and platforms, such as for example Microsoft Windows [91], Apple macOS [92], Android OS [93] and Apple's IOS [94]. The other option is to use web-based technology, where the supporting platform is running on top of a web-server. The type of web application that aims to be as usable and user-friendly as possible, both in connected and disconnected environments, is called a "Progressive Web Application" (PWA) [95] and would likely be the most modern and feature-rich type of application available based on web-technology. Progressive web applications provide some access to native features but they still share most of their benefits with traditional web-applications, compared to native applications.

There are several benefits of using web-based technology, such as:

- easy deployment to a central server or group of servers
- no client installations to maintain
- new application versions can be deployed instantly to all clients by issuing updates to the server's code
- mostly device independent
- scalability benefits, as the application can be load-balanced over multiple servers

As for native applications, these are some benefits compared to applications based on web-technology:

- access to low-level code APIs, which could make the visualization run with greater performance
- no need to maintain your own web application server environment or to rent the servers from a cloud-based service provider

The list of benefits contains more points for using web-technology and when considering that more and more applications today are being developed for the web, opposed to native operating systems, then this is also most probably be the best way to do visualization for the system to show caused CO_2-emissions [96]. Special focus would need to be put on functionality to draw graphics in web applications, as representation of the data will use a combination of 2D-graphics, text and numeric output.

Drawing graphics for the web is mainly done by use of one of the following three techniques:

- SVG (Scalable Vector Graphics) [97]
- canvas
- WebGL (Web Graphics Library)

All three techniques have their use cases. [98] [99]

SVG stands for Scalable Vector Graphics and is a language, based on XML [78], for drawing 2D-graphics. Web browsers have built-in support for SVG and the XML definitions can be made part of the HTML code. This technique is probably the most easy to understand and use as it is made part of a web project's Document Object Model [100], which makes it easy to inspect and debug. The fact that it is using XML formatted elements as part of the HTML code, also makes it the worst performing of these techniques. It is in general suitable for graphics where the maximum amount of graphics elements is around a few 1000. SVG has the advantage that its graphics definitions can scale to different resolutions, while still keeping the sharpness of the graphics. The graphics can be manipulated through the web page's DOM and by use of CSS styling [101].

Canvas is an HTML5 element [102] which is used as a container to render graphics on, through the use of a Javascript API. The graphics is dynamically rendered onto the canvas context as a bitmap image and is as such resolution-dependent. This method requires less memory compared to SVG, as it is producing the graphics as rasterized images [103]. Drawing graphics with the HTML5 canvas element provides more performance compared to SVG and allows for drawing of graphics containing several thousands of graphical elements. Using of canvas is more difficult compared to SVG. SVG is made part of the HTML document, and allows for easy inspection and manipulation by applying different styles, while the use of canvas results in a drawn image without any way to inspect how the image result was reached or any way of easily manipulating the finished image. Creating images that provide interactivity for the user is also more difficult when using canvas, compared to SVG.

WebGL produces rasterized images, same as the use of the canvas element. WebGL does this by use of a computer's graphical processing unit (GPU). This provides much more performance compared to using the canvas element and allows for smooth graphics rendering and animations of both 2D- and 3D-graphics. This technique is useful when the amount of data to be visualized is too large for the other methods or when complex computations must be applied in order to draw the graphics. This method of drawing graphics is the most complex and low-level method to use, when compared to SVG and canvas.

Third-party libraries and components are also available for drawing web page graphics. The libraries make use of one or more of the above mentioned web drawing techniques, but provide abstraction methods to make it easier for the user to draw the graphics. One such example is the JavaScript library ”D3.js” [104], which based on data manipulates the DOM of a web page in order to visualize the data. The output graphics is based on SVG and canvas techniques. This library is especially useful for large data sets, as it can enable linking a point in the visualization to a large portion of the data, which will help to present large data sets with great performance. D3.js is not created to focus on any specific type of visualization requirement and can therefore be used with any kind of data requiring visualization. Most of the other third-party visualization libraries [105] are specialized on creating chart graphics (for example Chart.js [106] and Highcharts [107]), which can make them easier to use compared to D3.js, as they provide ready-made functionality but are limited to only drawing chart elements. D3.js can be more difficult to learn, compared to other libraries, because of the fact that it can be used on any kind of data.

The type of graphics required by the system displaying caused CO_2 emissions is 2-dimensional, which requires less resources compared to 3-dimensional graphics. The main purpose of the graphics elements in the system is to display the equipment in the factory for which CO_2 emissions are being monitored. Also, the relationship of the equipment is to be shown graphically, in order to display the the process flow. These requirements do not require a lot of animations or zoom-functionality and the total amount of graphical object elements for an individual screen is likely to at a maximum be counted in a few hundred. Because of the limited graphical requirements and the fact that SVG provides the easiest means to work with, SVG would be the selected graphical technique to use for the system's visualization.

The system's configuration component would need to include a graphical editor, where new equipment type graphics could be created and modified by the engineer. The graphical editor would work in a similar way as most paint applications, such as Microsoft's Paint, providing a graphical user interface for creating the system's graphics. The saved output would be in SVG format, but the engineer should need no knowledge of SVG or of the format the drawn elements are saved in, in order to use the graphical editor.

The equipment template types in the system, could then be assigned a graphical representation, which would be shown in visualization screens for instances of the equipment type. The graphical editor would also be used to create the screens for visualizing the factory processes, by working in either of two modes, one for creating template type graphics and another for creating the screens. The hierarchy defined for the equipment instances could be used to help with the creation of visualization screens, but the engineer would still be responsible for manually creating logical screen layouts (often according to how the factory process is physically constructed in the factory). The graphical elements connecting the equipment instance graphics, would also have to be created manually by the engineer, in order for the process picture to make sense. The system could in theory create connecting lines between different equipment in a screen, but a simple line does not provide information such as if two equipment are connected by for example a liquid pipe, an elevator or a gas line.

Each graphical representation of an equipment instance would at a minimum need be able to display the amount of CO_2 emissions it has caused in a numerical form. This would need to be displayed as a text field next to the graphical object. Some additional information might also be of benefit to be shown graphically, for example a situation where a possible pre-defined or calculated high-limit is exceeded for caused CO_2 emissions. Such information could be displayed most easily by allowing the instance type to have several graphical objects assigned, for example one for normal visualization and another one for situations where the state has changed away from normal. More advanced versions could allow for more detailed graphical manipulation, such as for example a blinking background color when the state changes away from normal state.

The visualized information shown for an instance type would be linked to a defined data field in the system. This would be done by the engineer, when configuring the data of an equipment instance. Additional information about the factory process would not need to be displayed, as this kind of information is already displayed in the HMI-system for the factory (such as for example if the equipment is active, any active alarms etc.).

5.3 End user, usage: Analysis and processing of collected data

Analysis of the recorded data (saved in a Microsoft SQL Server database) in regards to estimation of caused CO_2 emissions, was done manually by use of Microsoft's SQL Server Management studio application. All the operations done manually while analyzing the data, could be programmed into the planned system for displaying CO_2 emissions data, and as such automated to be performed as background tasks. The input for these operations, the data, the formulas for the calculations and the configuration parameters for the calculations would all need to be configured by an engineer, when configuring the equipment templates and instances in the system.

5.3.1 Data queries, processing and calculation to obtain measured and estimated energy usage

The first process data to be recorded was selected from a certain process area in Nordkalk's factory in Parainen, Finland. This process area contains the process equipment required for filling buffer storage silos with limestone, which is the raw-material for the factory process where the limestone is heated up to a temperature over 1000 degrees Celsius (which produces the product called 'slaked limestone' [108]). Figure 5.2 shows the process screen as configured in the factory's automation system. The assigned equipment positions for most equipment can be seen with red labels and have been added manually on top of the screenshot.

The process screen shows two buffer silos in the middle of the screen, of which the right one (labelled ”1006_LI1”) is the primary storage silo for the limestone, supplying

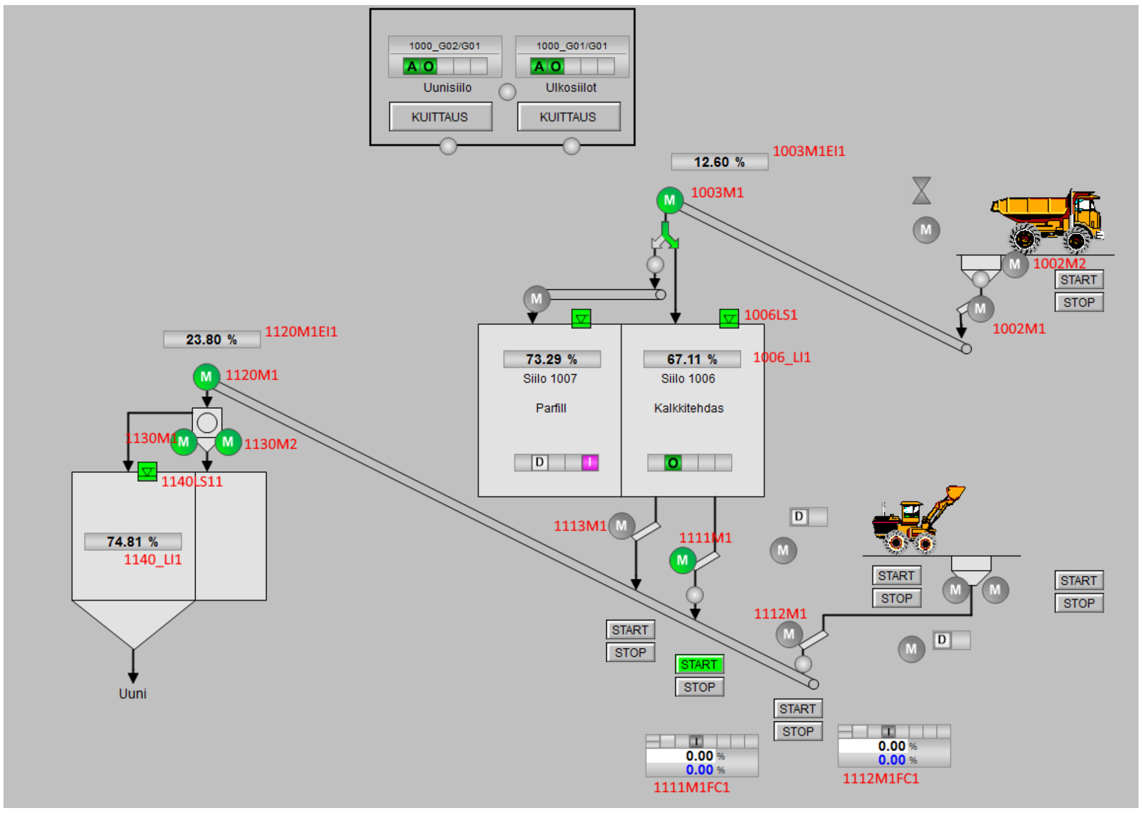


Figure 5.2: Limestone intake process at Nordkalk's factory in Parainen, Finland

the process. The left storage silo in the middle of the screen, is physically located in the neighbouring factory on the same plot of land and is shown here as some of the in-feed equipment is shared between the two factories' processes. The limestone storage silo on the left hand of the screen (labelled "1140_LI1") is the final buffer silo holding the raw-material, from where the limestone is taken and fed into the kiln for processing.

The limestone is transported with wheel loaders and other transportation equipment to feeding intakes, which are connected to conveyor belts, which in turn transport the limestone to the buffer silos. The limestone is always fed to the factory process from the silo on the left hand of the screen, marked by position "1140_LI1" (showing the filling level of the silo). The silo has an upper limit sensor, labeled "1140LS11", that is used for controlling when to start refilling of the buffer silo. Limestone is delivered to the top of the "1140_LI1"-silo by a conveyor belt (filling the silo), which is operated by an electrical engine, with the label position ”1120M1”. The limestone is then supplied to the conveyor belt from the primary limestone silo 1006_LI1 but can also be supplied directly by wheel loaders through a smaller intake facility, which has a vibration motor [109] (labelled ”1112M1”) helping the limestone onto the conveyor belt.

The value shown for conveyor belt engine ”1120M1” through the measurement labeled ”1120M1EI1”, is a percentage of engine power in use, out of the engine’s maximum specified power. The power measurement value is supplied by an electrical frequency inverter [110], which controls the speed of the engine. This percentage value can be directly converted into a sample of the engine’s momentary power usage and the recorded data could then be used to calculate the amount of caused CO_2 emissions by the engine, over a certain time period, by use of a conversion factor.

The direct power usage measurement reading gives a reference value to compare to, when considering estimating the power usage of the engine.

From the process screen, it can be seen that the vibration motor ”1111M1” is running and that the limestone is supplied to the conveyor belt through this path from the limestone silo 1006. Not knowing at what variable speed the conveyor belt engine ”1120M1” is running, one might estimate its power usage through an estimation of the load applied to the conveyor belt, in other words the amount of limestone delivered onto the conveyor belt. There is no measured process data available for this amount of limestone. The fill-level of the primary limestone silo 1006 is measured and could be used to estimate the supplied limestone amount over a given time frame, when knowing the volumetric size of the silo. This is however problematic to do, at the time when this screen shot (Figure 5.2) of the process was taken, because the primary limestone silo is filled with new limestone at the same time as the silo supplies the conveyor belt underneath it with limestone. This means that the fill-level of the silo is most likely both increasing and decreasing during the time frame of interest. Even in the case, that the fill-level would be only decreasing during the inspected time frame, it would be wrong to assume that the decreasing rate is the true rate of the supply of limestone to the conveyor belt under the silo, as it must be remembered that new limestone is constantly filling up the silo (in this case at a slower rate compared to the purging of limestone).

During another time, it was observed that limestone was supplied to the conveyor belt from the path where the vibration motor labeled ”1112M1” is located, while vibration motor ”1111M1” was not running. The limestone here originated from a smaller filling pit, filled up by use of a wheel loader. In this case, the fill-level of a silo is not available as a source of data to estimate the load on, for the electrical engine ”1120M1”. The only known fact is that limestone is most likely supplied to the conveyor belt, but at what rate and if the rate is constant is not known. In this case, there is no information on what to base the estimation of the applied load on the conveyor belt.

A manually done estimation of the power usage of conveyor belt engine 1120M1 was done, based on data from a time frame where silo 1006 was not being filled and while the conveyor belt was running and being supplied with limestone through the process path using the vibration motor ”1111M1”.

The following explanation of the analysis process shows how this was done:

The database table’s structure for holding the recorded process data can be seen in Figure 5.3. Each recorded process data sample consists of a row record of the configured database table columns.

- The RowID column is a unique identifier for each recorded row of data and is being assigned automatically by the database.
- NodeAlias is a unique alias name assigned by the Data Recorder component, to identify the recorded data point.
- Display name is a more user friendly display name for the recorded data point, originating from the OPC UA server’s provided data.
- The value column contains the recorded value. Recorded process data points can

be of different data types, for example whole integer numbers, text data or Boolean on/off bit data. This is why the value is recorded as Microsoft SQL server's varchar() datatype, which is a text type, and can as such hold any kind of data (e.g. numeric and textual).

- ValueStatusCode column hold a status code for the measured data point, provided by the OPC UA server, which can be used to indicate the quality and reliability of the recorded data.
- The time for the recording of the data point, as provided by the OPC UA server, is saved in the ValueServerTimestamp column.
- The Data Importer component tries to convert each recorded data point's value data from text form to numeric form, which is saved in the ValueAsNumber column, if the conversion is successful. The ValueAsNumber is easier to use in analysis operations for sorting, calculations etc. as it is in numeric form, compared to the original Value data, which is is textual form.

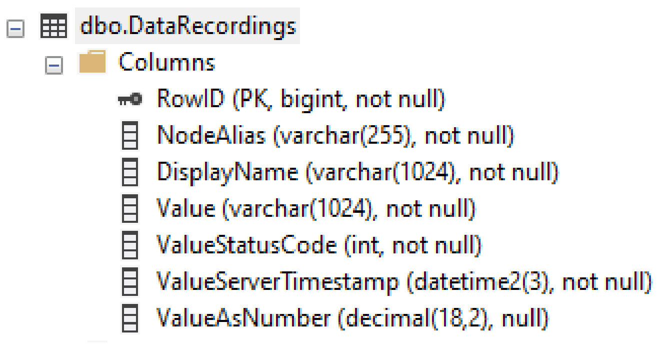

```
dbo.DataRecordings
  Columns
    RowID (PK, bigint, not null)
    NodeAlias (varchar(255), not null)
    DisplayName (varchar(1024), not null)
    Value (varchar(1024), not null)
    ValueStatusCode (int, not null)
    ValueServerTimestamp (datetime2(3), not null)
    ValueAsNumber (decimal(18,2), null)
```

Figure 5.3: Data table structure for holding the recorded process data

A database query was constructed for finding out the time frames where the conveyor belt's electrical engine ”1120M1” was running. The SQL query can be seen in Code Listing 5.1.

```sql

;WITH CTE_Source AS
```

```sql
  (
  SELECT TOP 100 PERCENT NodeAlias, [Value], [ValueServerTimestamp], RowID
    , LAG([Value], 1, 0) OVER (ORDER BY RowID ASC) AS PriorValue
  FROM DataRecordings AS T
  WHERE NodeAlias = '1120M1_OnOffStatus' AND ValueServerTimestamp >= '2021-07-30 0:00:00.000'
  ORDER BY T.RowID, T.[Value]
  )
SELECT NodeAlias, RowID, [Value], [ValueServerTimestamp]
  , LAG([ValueServerTimestamp]) OVER (ORDER BY RowID ASC) AS PriorValueTimestamp
  , DATEDIFF( second, LAG([ValueServerTimestamp]) OVER (ORDER BY RowID ASC)
  , [ValueServerTimestamp] ) AS 'IntervalTimeInSeconds'
  , IIF( [Value] = 'False'
    , 15 / (CAST( DATEDIFF(
      second
      , LAG([ValueServerTimestamp]) OVER (ORDER BY RowID ASC)
      , [ValueServerTimestamp] ) AS float) / 60 / 60)
    , 0
  )  AS kWh
FROM CTE_Source
WHERE PriorValue IS NULL OR PriorValue <> [Value]
```

Listing 5.1: SQL query to find time frames when engine 1120M1 is running

The query also experiments with calculating the amount of power used over each running time frame, in such a case that the engine would be constantly running at full power. This result can be seen in column "kWh". The maximum power for electrical engine 1120M1 is rated at 15 kW and this value is used to calculate the used power for column "kWh". This result can later be compared to the measured and estimated power usage values for the engine, and serves as an example of how it is not possible to calculate the used power for the electrical engine, as this result is not accurate.

NodeAlias	RowID	Value	ValueServerTimestamp	PriorValueTimestamp	IntervalTimeInSeconds	kWh
1120M1_OnOffStatus	2341637	False	2021-07-30 00:00:04.257	NULL	NULL	NULL
1120M1_OnOffStatus	2341701	True	2021-07-30 01:44:56.710	2021-07-30 00:00:04.257	6292	0
1120M1_OnOffStatus	2347925	False	2021-07-30 02:49:48.360	2021-07-30 01:44:56.710	3892	13,8746145940391
1120M1_OnOffStatus	2382501	True	2021-07-30 07:05:54.061	2021-07-30 02:49:48.360	15366	0
1120M1_OnOffStatus	2391045	False	2021-07-30 08:34:56.157	2021-07-30 07:05:54.061	5342	10,1085735679521
1120M1_OnOffStatus	2402581	True	2021-07-30 10:35:09.176	2021-07-30 08:34:56.157	7213	0
1120M1_OnOffStatus	2405397	False	2021-07-30 11:04:29.969	2021-07-30 10:35:09.176	1760	30,6818181818182
1120M1_OnOffStatus	2419509	True	2021-07-30 13:31:33.278	2021-07-30 11:04:29.969	8824	0

Table 5.1: SQL query result for engine 1120M1's running time frames

The result of the query can be seen in Table 5.1. One of the active time frames for engine 1120M1 was from "*30.7.2021 7:05:54.061*" to "*30.7.2021 8:34:56.157*", a time frame of 1 hour, 29 minutes, 2 seconds and 96 milliseconds. This time frame was selected for investigating the feasibility of estimating the power usage for engine 1120M1. The first thing to make sure of, was that the limestone silo 1006 had not been filled with new limestone during this time frame. A query for checking the running time of electrical engine 1003M1 during the selected time frame was executed. The SQL query can be seen in Code Listing 5.2. Engine 1003M1 is running when limestone silo 1006 is being filled. The estimation of consumed limestone from the silo would be impossible if the silo was filled at the same time as limestone was being consumed from the silo.

```sql
-- Check time frame of interest to make sure that silo 1006 is not being filled during this time
-- No returned rows --> filling of silo has not been active
DECLARE @TimeframeStartTime datetime;
DECLARE @TimeframeStopTime datetime;
SET @TimeframeStartTime = '2021-07-30 07:05:54.061';
SET @TimeframeStopTime = '2021-07-30 08:34:56.157';

SELECT * FROM DataRecordings WHERE NodeAlias = '1003M1_OnOffStatus'
  AND ValueServerTimestamp >= @TimeframeStartTime
  AND ValueServerTimestamp <= @TimeframeStopTime
  AND [ValueAsNumber] = 1;
```

Listing 5.2: SQL query for getting engine 1003M1's running state

The result of the query of engine 1003M1's running states can be seen in Table 5.2. The result has no rows, which means that engine 1003M1 was not running during the time frame being investigated.

RowID	NodeAlias	DisplayName	Value	ValueStatusCode	ValueServerTimestamp	ValueAsNumber

Table 5.2: SQL query result for engine 1003M1's running state during time frame

The recorded measure points for limestone silo 1006's fill level were retrieved using the SQL query seen in Code Listing 5.3. The fill level at the beginning of the time frame being investigated was 36,52% and 31,19% at the end of the time frame. These results can be seen from Tables 5.3 and 5.4.

```sql
DECLARE @TimeframeStartTime datetime;
DECLARE @TimeframeStopTime datetime;
SET @TimeframeStartTime = '2021-07-30 07:05:54.061';
SET @TimeframeStopTime = '2021-07-30 08:34:56.157';
```

```

SELECT * FROM DataRecordings WHERE NodeAlias = '1003M1_OnOffStatus'
  AND ValueServerTimestamp >= @TimeframeStartTime
  AND ValueServerTimestamp <= @TimeframeStopTime
  AND [ValueAsNumber] = 1;
```

Listing 5.3: SQL query for getting the fill level of limestone silo 1006

RowID	NodeAlias	DisplayName	Value	ValueStatusCode	ValueServerTimestamp	ValueAsNumber
2382510	1006_LI1_MvPercFillLevel	1006_LI1/MEASURE.MV_Perc	36.5151	0	2021-07-30 07:05:54.470	36.52
2382526	1006_LI1_MvPercFillLevel	1006_LI1/MEASURE.MV_Perc	36.50735	0	2021-07-30 07:06:04.499	36.51
2382542	1006_LI1_MvPercFillLevel	1006_LI1/MEASURE.MV_Perc	36.49944	0	2021-07-30 07:06:14.562	36.50

Table 5.3: Limestone silo 1006 fill level at the beginning of selected time frame

RowID	NodeAlias	DisplayName	Value	ValueStatusCode	ValueServerTimestamp	ValueAsNumber
2390926	1006_LI1_MvPercFillLevel	1006_LI1/MEASURE.MV_Perc	31.27968	0	2021-07-30 08:33:36.590	31.28
2390942	1006_LI1_MvPercFillLevel	1006_LI1/MEASURE.MV_Perc	31.27539	0	2021-07-30 08:33:46.476	31.28
2390958	1006_LI1_MvPercFillLevel	1006_LI1/MEASURE.MV_Perc	31.25963	0	2021-07-30 08:33:56.567	31.26
2390974	1006_LI1_MvPercFillLevel	1006_LI1/MEASURE.MV_Perc	31.2433	0	2021-07-30 08:34:06.688	31.24
2390990	1006_LI1_MvPercFillLevel	1006_LI1/MEASURE.MV_Perc	31.24479	0	2021-07-30 08:34:16.483	31.24
2391006	1006_LI1_MvPercFillLevel	1006_LI1/MEASURE.MV_Perc	31.22408	0	2021-07-30 08:34:26.670	31.22
2391022	1006_LI1_MvPercFillLevel	1006_LI1/MEASURE.MV_Perc	31.20243	0	2021-07-30 08:34:36.611	31.20
2391038	1006_LI1_MvPercFillLevel	1006_LI1/MEASURE.MV_Perc	31.19427	0	2021-07-30 08:34:46.616	31.19

Table 5.4: Limestone silo 1006 fill level at the end of selected time frame

Code Listing 5.4 shows the calculations made for calculating the consumption of limestone from silo 1006 and Table 5.5 shows the results of the calculations.

```
SELECT (36.52 - 31.19) AS LimeSiloLevelPercentDeclineOverTimePeriod
  , (113 * 36.52) / 100 AS SiloStartLevel_m3
  , (113 * 31.19) / 100 AS SiloStopLevel_m3
  , ((113 * 36.52) / 100) - ((113 * 31.19) / 100) AS LimeStoneConsumed_m3
  , (((113 * 36.52) / 100) - ((113 * 31.19) / 100)) * 1000 AS LimeStoneConsumed_kg;
```

Listing 5.4: Limestone usage from silo 1006 - calculations

The fill level of the silo has decreased with 5,33% during the time frame being investigated. It was found from mechanical documents that limestone silo 1006 has a volume of 113 m3. With this information, it was calculated that the silo contained 41,27 m3 of limestone at the beginning of the time frame and 35,24 m3 of limestone and the end of the time frame, which means that a total of 6,02 m3 of limestone was consumed by the factory process during the time frame. This volumetric amount of limestone was then converted to kilograms, which meant that 6022,9 kg of limestone was supplied by the silo during the time frame.

LimeSiloLevelPercentDeclineOverTimePeriod	SiloStartLevel_m3	SiloStopLevel_m3	LimeStoneConsumed_m3	LimeStoneConsumed_kg
5.33	41.267600	35.244700	6.022900	6022.900000

Table 5.5: Consumed amount of limestone during investigated time frame

The conveyor belt that moves the limestone raw-material to the top of limestone silo 1006 is lifting the limestone from a ground level to the top of the silo. The conveyor belt has a length of approximately 100 meters. The force needed for this operation was then calculated using Newton's second law:

$$F = m \times a$$

where

F = force [N]

m = mass of an object [kg]

a = acceleration [m/s²]

The mass to move is known as 6022,9 kg and the acceleration is the gravitational force of the earth, approximately 9,81 m/s². The total force needed was calculated to be 59084,6 N. [111]

The work required was then calculated when knowing that:

$$W = F \times d$$

where

W = the work done; the transfer of energy by a force [J]

F = force [N]

d = the distance that the displacement of a mass must be moved [m]

The required work was calculated to be 5908464.9 J (Joules) when using the length of the conveyor belt (100 m) as the distance. [112]

The amount of work required, could then be converted into required power for doing the work, by use of the following calculation:

$$P = \frac{W}{t}$$

where

P = the rate at which the work is done, measured in watts [W]

W = the energy needed for the work to be done [J]

t = the time in which the work is to be done [s]

The amount of power required was estimated to be approximately 1,11 kW [113] and the total power usage at approximately 1,64 kWh for the duration of time running engine 1120M1.

```sql
DECLARE @TimeframeStartTime datetime;
DECLARE @TimeframeStopTime datetime;
SET @TimeframeStartTime = '2021-07-30 07:05:54.061';
SET @TimeframeStopTime = '2021-07-30 08:34:56.157';

SELECT
  --Force required, Newton
  (6022.9 * 9.81) AS ForceNeededToMoveAmount_N
  --Work required = Force * Distance; 100 m is the approximate length of the conveyor belt, unit Joule
  , (6022.9 * 9.81) * 100 AS WorkToBeDone_J
  --Power required = (Force * Distance) / Time; Unit: Watts [W]
  , ((6022.9 * 9.81) * 100) / 5342 AS PowerNeeded_W
  --Power required as kW = (Force * Distance) / Time; Unit: Kilo Watts [kW]
  , (((6022.9 * 9.81) * 100) / 5342) / 1000 AS PowerNeeded_kW
  -- Power used over the duration of the investigated time frame [kWh]
  , CAST( ((((6022.9 * 9.81) * 100) / 5342) / 1000) AS decimal(18,8) )
    * (CAST( DATEDIFF( s, @TimeframeStartTime, @TimeframeStopTime ) AS decimal(18,8) ) / 60 / 60) AS kWh
```

Listing 5.5: Force, work and power needed for engine 1120M1 - calculations

ForceNeededToMoveAmount_N	WorkToBeDone_J	PowerNeeded_W	PowerNeeded_kW	kWh
59084.649	5908464.900	1106.03985398	1.1060398539800	1.64124024408326820

Table 5.6: Power needed to move the consumed limestone raw-material on conveyor belt by engine 1120M1 during time frame

Actual measured data for the power usage of engine 1120M1 was available for comparison (provided by the frequency inverter that controls the engine). This data was recorded as a percentage of the engine's maximum power, which was 15 kW. The average power percentage usage during the investigated time frame was 33,46%, which is about 5,02 kW. The total use of power over the time frame was around 7,45 kWh. The Code Listing for querying the data can be seen in Listing 5.6 and the result of the query in Table 5.7.

```
DECLARE @TimeframeStartTime datetime;
DECLARE @TimeframeStopTime datetime;
SET @TimeframeStartTime = '2021-07-30 07:05:54.061';
SET @TimeframeStopTime = '2021-07-30 08:34:56.157';

SELECT AVG([ValueAsNumber]) AS AveragePercentOutOfMax
    ,(15 * AVG([ValueAsNumber]) / 100) AS Average_Measured_kW
    , CAST( (15 * AVG([ValueAsNumber]) / 100) AS decimal(18,8) )
      * (
        CAST( DATEDIFF( s, @TimeframeStartTime, @TimeframeStopTime ) AS decimal(18,8) ) / 60 / 60
        ) Average_Measured_kWh
  FROM DataRecordings
  WHERE NodeAlias = '1120M1EI1_PVOut'
    AND ValueServerTimestamp >= @TimeframeStartTime AND ValueServerTimestamp <= @TimeframeStopTime;
```

Listing 5.6: Actual measured data for power usage for engine 1120M1

AveragePercentOutOfMax	Average_Measured_kW	Average_Measured_kWh
33.458239	5.018735	7.44724510277748223

Table 5.7: Measured power usage for engine 1120M1

The difference between the actual measured data result and the estimated result is 3,91 kW (5,02 kW - 1,1 kW) and 5,81 kWh (7,45 kWh - 1,64 kWh), with the estimation providing too low values as a result. The estimated power usage value is 21,91% out of the actual measured power usage value.

At this point, it should be pointed out, that the estimation did not so far take into account certain variables affecting the estimation calculations. An electric engine's performance is never such, that 100% of its power would be converted into the power required for the work load. There are other losses when running an electric engine, which are emitted as heat from the engine. The true efficiency ratio of an electric engine can according to literature be considered to be around 85%. [114]. Code Listing 5.7 takes the engine's efficiency into account by increasing the power requirements by 15% (100% - 85%) and the new estimation results can be seen in Table 5.8.

```sql
DECLARE @TimeframeStartTime datetime;
DECLARE @TimeframeStopTime datetime;
SET @TimeframeStartTime = '2021-07-30 07:05:54.061';
SET @TimeframeStopTime = '2021-07-30 08:34:56.157';

SELECT
  --Power required = (Force * Distance) / Time; Unit: Watts [W] (*1.15 because of engine efficiency ratio at 85%)
  (((6022.9 * 9.81) * 100) / 5342) * 1.15 AS PowerNeeded_WhenEngineEffciencyAt85Perc_W
  --Power required as kW = (Force * Distance) / Time; Unit: Kilo Watts [kW]
  , ((((6022.9 * 9.81) * 100) / 5342) * 1.15) / 1000 AS PowerNeeded_kW
  -- Power used over the duration of the investigated time frame [kWh]
  , CAST( (((((6022.9 * 9.81) * 100) / 5342) * 1.15) / 1000) AS decimal(18,8) )
    * (CAST( DATEDIFF( s, @TimeframeStartTime, @TimeframeStopTime ) AS decimal(18,8) ) / 60 / 60) AS kWh
```

Listing 5.7: Estimation of power usage considering engine power efficiency ratio

EngineEffciencyAt85Percent_W	PowerNeeded_kW	kWh
1271.9458320770	1.271945832077000	1.88742628440548065

Table 5.8: Estimated power usage for engine 1120M1 taking a 85% efficiency ratio into consideration

Consideration of the electrical engine's efficiency ratio increases the estimation of power usage a bit, from 1,64 kWh to 1,89 kWh, but the difference to the measured power usage is still considerable. Another load variable still not considered is the weight of the conveyor belt itself, which adds to the weight that the engine is operating on. An average weight for a typical wear resistant rubber belt, fit for transport of stone material, is about 6,8 kg per square meter [115]. The conveyor belt operated by engine 1120M1 is about 1 meter wide, which would mean that the total area for one side of the conveyor belt is

$$1m \times 100m = 100m^2$$

which would add the following mass to be lifted by the electrical engine:

$$100m^2 \times 6.8\frac{kg}{m^2} = 680kg$$

The weight of the conveyor belt is being lifted multiple times during the course of the investigated time frame. The number of times the conveyor belt needs to be rotated can be estimated based on the length of the conveyor belt, the speed it is moving on and the total time of operation. The calculations can be seen in Code Listing 5.8. The result can be seen in Table 5.9.

```sql
DECLARE @lengthOfConveyorBelt int;
DECLARE @movingSpeedForBelt decimal(18,2);
DECLARE @timeInSecondsForBeltToMove100meters decimal(18,2);
DECLARE @totalRunningTimeInSeconds int;
DECLARE @numberOfTimesToFillAndEmptyBelt decimal(18,2);

SET @lengthOfConveyorBelt = 100;    -- m
SET @movingSpeedForBelt = 0.5;    -- m/s
SET @totalRunningTimeInSeconds = 5342;  -- s

SELECT @timeInSecondsForBeltToMove100meters = (@lengthOfConveyorBelt / @movingSpeedForBelt); -- s
SELECT @numberOfTimesToFillAndEmptyBelt
    = (@totalRunningTimeInSeconds / @timeInSecondsForBeltToMove100meters); -- number of times
```

Listing 5.8: Estimation of number of times the conveyor belt is rotated

NumberOfTimesToRotateTheBelt
26.71

Table 5.9: The number of times the conveyor belt is rotated during the investigated time frame

Code Listing 5.9 shows the estimation calculations when taking the conveyor belt's weight into account and the new results are listed in Table 5.10.

```sql
DECLARE @weight decimal(18,2);
SET @weight = 6022.9 + (680 * 26.71); -- Limestone mass + (conveyor belt mass * number of rotations)

SELECT
  --Power required = (Force * Distance) / Time; Unit: Watts [W] (*1.15 because of engine efficiency ratio at 85%)
  (((@weight * 9.81) * 100) / 5342) * 1.15 AS EngineEffciencyAt85Perc_W
  --Power required as kW = (Force * Distance) / Time; Unit: Kilo Watts [kW]
  , ((((@weight * 9.81) * 100) / 5342) * 1.15) / 1000 AS PowerNeeded_kW
  -- Power used over the duration of the investigated time frame [kWh]
  , CAST( (((((@weight * 9.81) * 100) / 5342) * 1.15) / 1000) AS decimal(18,8) )
    * (CAST( DATEDIFF( s, @TimeframeStartTime, @TimeframeStopTime ) AS decimal(18,8) ) / 60 / 60) AS kWh
```

Listing 5.9: Estimation of power usage considering the conveyor belt's weight

This estimation result is now very near to the measured value for the power usage of engine 1120M1 over the investigated time frame. The difference between the actual measured data and the estimated result is now -0,09 kW (5,02 kW - 5,11 kW) and -0,13

EngineEffciencyAt85Percent_W	PowerNeeded_kW	kWh
5107.65583208505	5.10765583208505	7.57919373440525477

Table 5.10: Estimated power usage for engine 1120M1 taking the conveyor belt weight into consideration

kWh (7,45 kWh - 7,58 kWh), with the estimation showing a bit too large values as a result. The estimated power usage value is 101,8% out of the actual measured power usage value.

5.3.2 Calculations to obtain caused CO_2 emission data

With either an estimated or measured power usage value at hand, the system should be able to show the used energy, as an amount of CO_2 emissions. This is possible by applying a conversion factor to the power usage values. In this case, the energy to run the engine was provided by the connection to the national power grid. Many producers of electricity make yearly data available on CO_2-conversion factors for the energy they produce. For example, the Finnish electricity producer Helen had a CO_2-conversion factor of 99 g / kWh for the year 2020, that is 99 grams of carbon dioxide emissions for each produced kilowatt of electric energy. [116]

Table 5.11 displays Helen's CO_2-conversion factors for years 2015-2020:

Year	Electricity (g/kWh)
2020	99
2019	139
2018	191
2017	191
2016	198
2015	191

Table 5.11: Helen's CO_2-conversion factors for production of electricity

These CO_2-conversion factors provide the average value for each year, as provided by the electricity producer. More detailed information is provided by inspecting the information for a specific electricity product. Two examples are Helen's Basic Electricity package, based on 8% renewable sources, 41% fossil sources and 52% nuclear power with an average CO_2-conversion factor of 232 g / kWh [117] and the Environmental electricity Electricity package, based on 10% solar power, 30% hydro power and 60% wind power with an average CO_2-conversion factor of 0 g / kWh [118].

The CO_2 emissions conversion factor information must be manually entered into the system, as this information can not otherwise be known or estimated by any available process data. Most factories do not produce their own electricity, so the CO_2-conversion factor must be obtained from the electricity producer.

In this case, if it could be assumed, that the factory had been buying their electricity from the Finnish company Helen, by using their product package called "Basic Electricity", the caused CO_2-emissions amount for the investigated time frame for the engine 1120M1 would have been

based on estimation:

$$7,58kWh \times 0,232\frac{kg}{kWh} = 1,76kg$$

of CO_2-emissions, and based on measurement:

$$7,45kWh \times 0,232\frac{kg}{kWh} = 1,73kg$$

of CO_2-emissions.

5.4 End user, usage: Real-time reporting

The concept "*in real-time*", regarding showing of caused CO_2 emissions, could be thought to mean "*new data available within a few seconds*". Real-time reporting could be possible for reporting of equipment's data fields showing caused CO_2 emissions, where the CO_2

emissions are directly measured. The delay from measurement to display would consist of the sampling and recording time for the measured data and the time for the system to query the latest data and refresh the visualization screen.

Real-time reporting for equipment where the caused CO_2 emissions are estimated, at the same refresh rate as for equipment with measured CO_2 emissions, would usually not be possible in the same way. The estimated data is based on some calculations of some process values that indirectly has an effect on the caused CO_2, which means that there in most cases would be a delay between the time an estimated value could be calculated and the time for when the visualization value is refreshed. The estimated value could be refreshed often while being displayed, but the value itself lagging in time compared to the current time. This is because the estimated value's calculations would often rely on data sampled over a longer time period than one sample interval.

The value of being able to observe caused CO_2 emissions in real time from a particular factory equipment might often not be so important. Many processes have cycles in where the load varies on different parts of the process, resulting in fluctuating values of caused CO_2 emissions, but which however is normal when considering the process as a whole. Outside factors can also have an unforeseen effect on the CO_2 emissions, such as for example a situation where the supplying of raw material to the process would be suddenly interrupted, causing a sudden drop in process equipment's CO_2 emissions, because of the decrease in load on the production line. A hypothetical situation where there would be value in real-time reporting of CO_2 emissions, is a scenario where an equipment's operation could be optimized based on the caused CO_2 emissions. This kind of requirement (optimization) is not common in a typical factory context and also not the main purpose for the system displaying caused CO_2 emissions.

There likely would be more value in showing the caused CO_2 emission values, for the configured equipment instances, as computed averages, running averages and other computed and processed values over a longer period of time, compared to the instantly available measured and estimated values. Caused CO_2 emissions values computed from data series over time periods such as hours, days and months, would eliminate the naturally occurring fluctuations in the CO_2 emissions taking place because of sudden change in load requirements and production events such as starting and stopping the production line. Comparing caused CO_2 emission values between equipment instances in a factory, could also benefit of value series obtained for longer time periods, as the resulting data would better highlight differences when fluctuations are eliminated.

Chapter 6

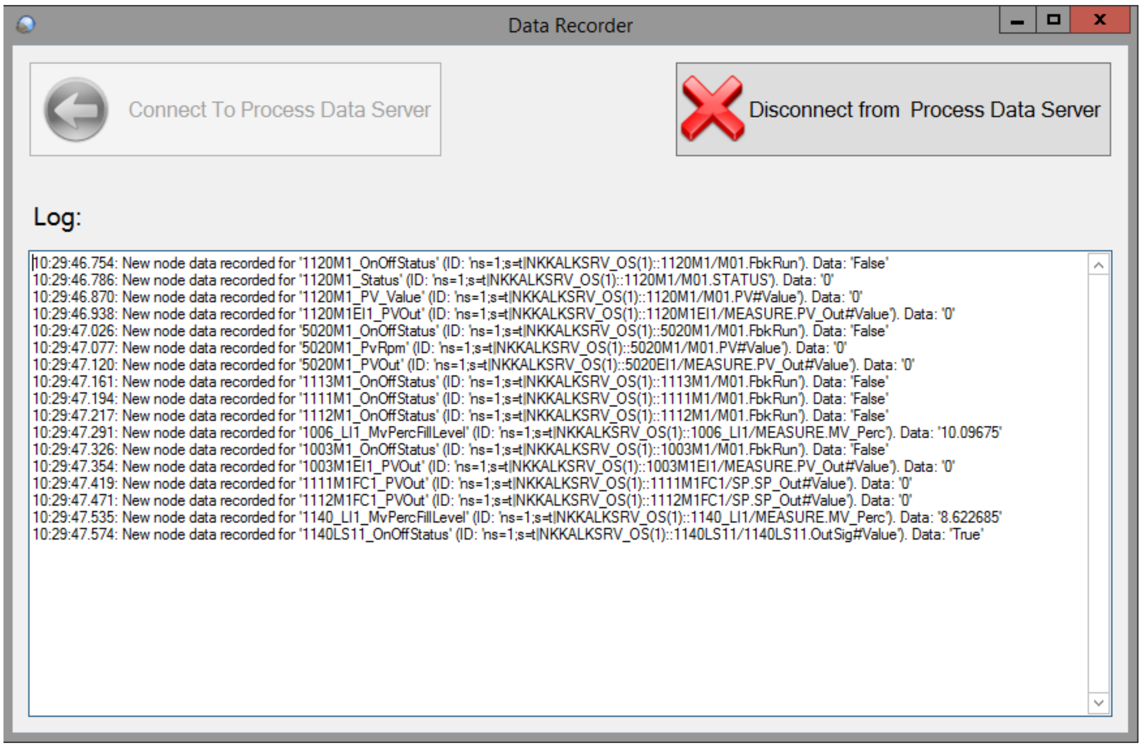
Implementation

The implementation of a complete system for displaying caused CO_2 emissions was found to be outside of the bounds for this thesis work. This is why time was saved by focusing on creating prototypes for some of the key components in the system. This was a way to show a proof-of-concept for how the system could be implemented.

6.1 Data Recorder

An application component was created for sampling and recording of process data. The application was created as a Windows Forms application using Microsoft .Net Framework and C# as programming language. The data communication interface between the application and the process data was made using OPC UA, for reading the process data. Figure 6.1 shows the component's User Interface while it is recording new process data.

Direct access from the development environment to the process data was not possible, because the remote connection to the factory over VPN (Virtual Private Network) [119] allowed only use of a remote desktop type application called VNC (Virtual Network Computing) [120]. This restriction meant, that the data recorder component had to be run locally on the factory's SCADA process computer, and that the recorded data had to be temporarily stored to local files, instead of the Data Pool component's database,


Data Recorder
Connect To Process Data Server
Disconnect from Process Data Server
Log:

```
10:29:46.754: New node data recorded for '1120M1_OnOffStatus' (ID: 'ns=1;s=t|NKKALKSRV_OS(1)::1120M1/M01.FbkRun'). Data: 'False'
10:29:46.786: New node data recorded for '1120M1_Status' (ID: 'ns=1;s=t|NKKALKSRV_OS(1)::1120M1/M01.STATUS'). Data: '0'
10:29:46.870: New node data recorded for '1120M1_PV_Value' (ID: 'ns=1;s=t|NKKALKSRV_OS(1)::1120M1/M01.PV#Value'). Data: '0'
10:29:46.938: New node data recorded for '1120M1EI1_PVOut' (ID: 'ns=1;s=t|NKKALKSRV_OS(1)::1120M1EI1/MEASURE.PV_Out#Value'). Data: '0'
10:29:47.026: New node data recorded for '5020M1_OnOffStatus' (ID: 'ns=1;s=t|NKKALKSRV_OS(1)::5020M1/M01.FbkRun'). Data: 'False'
10:29:47.077: New node data recorded for '5020M1_PvRpm' (ID: 'ns=1;s=t|NKKALKSRV_OS(1)::5020M1/M01.PV#Value'). Data: '0'
10:29:47.120: New node data recorded for '5020M1_PVOut' (ID: 'ns=1;s=t|NKKALKSRV_OS(1)::5020EI1/MEASURE.PV_Out#Value'). Data: '0'
10:29:47.161: New node data recorded for '1113M1_OnOffStatus' (ID: 'ns=1;s=t|NKKALKSRV_OS(1)::1113M1/M01.FbkRun'). Data: 'False'
10:29:47.194: New node data recorded for '1111M1_OnOffStatus' (ID: 'ns=1;s=t|NKKALKSRV_OS(1)::1111M1/M01.FbkRun'). Data: 'False'
10:29:47.217: New node data recorded for '1112M1_OnOffStatus' (ID: 'ns=1;s=t|NKKALKSRV_OS(1)::1112M1/M01.FbkRun'). Data: 'False'
10:29:47.291: New node data recorded for '1006_LI1_MvPercFillLevel' (ID: 'ns=1;s=t|NKKALKSRV_OS(1)::1006_LI1/MEASURE.MV_Perc'). Data: '10.09675'
10:29:47.326: New node data recorded for '1003M1_OnOffStatus' (ID: 'ns=1;s=t|NKKALKSRV_OS(1)::1003M1/M01.FbkRun'). Data: 'False'
10:29:47.354: New node data recorded for '1003M1EI1_PVOut' (ID: 'ns=1;s=t|NKKALKSRV_OS(1)::1003M1EI1/MEASURE.PV_Out#Value'). Data: '0'
10:29:47.419: New node data recorded for '1111M1FC1_PVOut' (ID: 'ns=1;s=t|NKKALKSRV_OS(1)::1111M1FC1/SP.SP_Out#Value'). Data: '0'
10:29:47.471: New node data recorded for '1112M1FC1_PVOut' (ID: 'ns=1;s=t|NKKALKSRV_OS(1)::1112M1FC1/SP.SP_Out#Value'). Data: '0'
10:29:47.535: New node data recorded for '1140_LI1_MvPercFillLevel' (ID: 'ns=1;s=t|NKKALKSRV_OS(1)::1140_LI1/MEASURE.MV_Perc'). Data: '8.622685'
10:29:47.574: New node data recorded for '1140LS11_OnOffStatus' (ID: 'ns=1;s=t|NKKALKSRV_OS(1)::1140LS11/1140LS11.OutSig#Value'). Data: 'True'
```

Figure 6.1: Data Recorder - sampling and saving of process data

which was located in the development environment. The files with the recorded process data was then manually copied to the development environment.

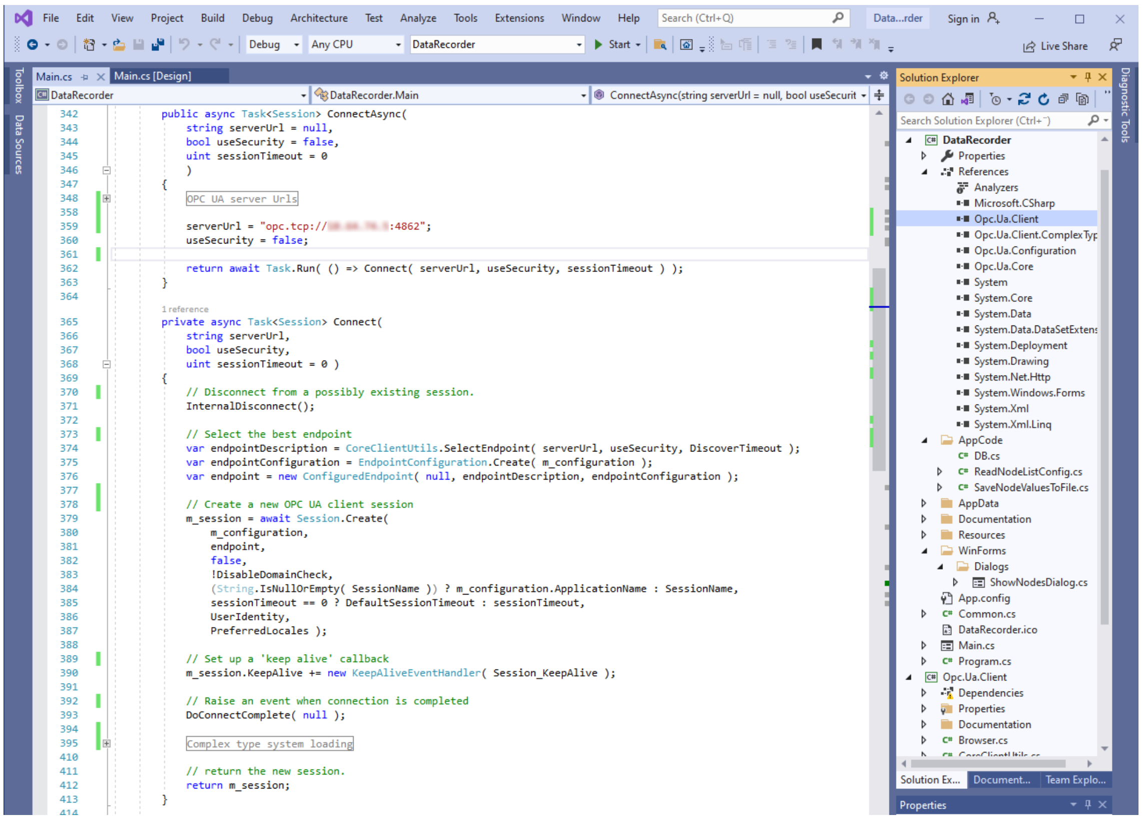
The data recorder component makes use of a configuration file, where the process data OPC UA nodes are listed, for which process data was to be recorded. The configuration file was in CSV format (column data separated by a ',' character), an example of it can be seen in Listing 6.1. The system for displaying CO_2 emissions would in a later version save this configuration data to its database.

```
NodeAlias,NodeId,Datatype
1120M1_OnOffStatus,ns=1;s=t|NKKALKSRV_OS(1)::1120M1/M01.FbkRun,System.Boolean
1120M1_Status,ns=1;s=t|NKKALKSRV_OS(1)::1120M1/M01.STATUS,System.UInt32
1120M1_PV_Value,ns=1;s=t|NKKALKSRV_OS(1)::1120M1/M01.PV#Value,System.Single
```

Listing 6.1: DataRecorder OPC UA Node config file

Figure 6.2 shows one important part of the code for the Data Recorder component, where a new connection and session is being established to the OPC UA server providing access to the factory process data. The OPC UA server was in this case Siemens' WinCC

OPC UA server (part of SCADA application WinCC, running as a background Windows service), which itself has access to the factory process data by use of Siemens' proprietary S7-communication protocol. The OPC UA communication is done using the OPC Foundation's OPC UA .Net Standard Library, which OPC.Ua.Client project can be seen in the figure as a reference for Data Recorder. The connection is here established without OPC UA's security-protocols enabled, but the component could be developed to include connection security through authentication and encryption for future versions.

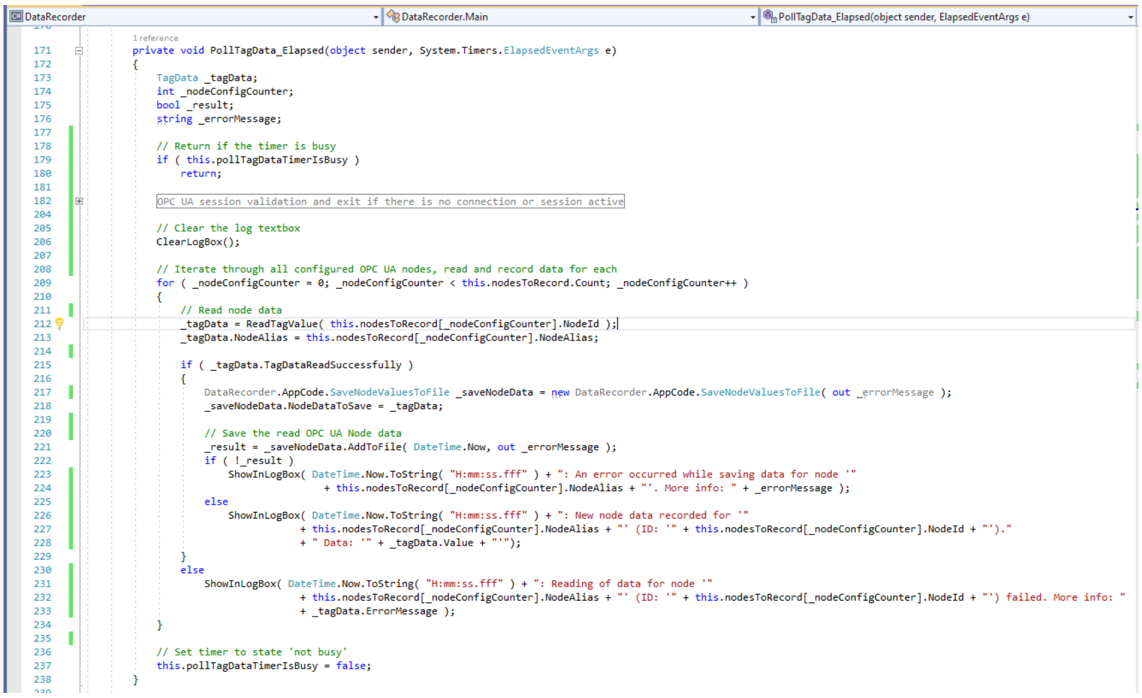


```
public async Task<Session> ConnectAsync(
    string serverUrl = null,
    bool useSecurity = false,
    uint sessionTimeout = 0
    )
{
    OPC UA server Urls

    serverUrl = "opc.tcp://[redacted]:4862";
    useSecurity = false;

    return await Task.Run( () => Connect( serverUrl, useSecurity, sessionTimeout ) );
}

1 reference
private async Task<Session> Connect(
    string serverUrl,
    bool useSecurity,
    uint sessionTimeout = 0 )
{
    // Disconnect from a possibly existing session.
    InternalDisconnect();

    // Select the best endpoint
    var endpointDescription = CoreClientUtils.SelectEndpoint( serverUrl, useSecurity, DiscoverTimeout );
    var endpointConfiguration = EndpointConfiguration.Create( m_configuration );
    var endpoint = new ConfiguredEndpoint( null, endpointDescription, endpointConfiguration );

    // Create a new OPC UA client session
    m_session = await Session.Create(
        m_configuration,
        endpoint,
        false,
        !DisableDomainCheck,
        (String.IsNullOrEmpty( SessionName )) ? m_configuration.ApplicationName : SessionName,
        sessionTimeout == 0 ? DefaultSessionTimeout : sessionTimeout,
        UserIdentity,
        PreferredLocales );

    // Set up a 'keep alive' callback
    m_session.KeepAlive += new KeepAliveEventHandler( Session_KeepAlive );

    // Raise an event when connection is completed
    DoConnectComplete( null );

    Complex type system loading

    // return the new session.
    return m_session;
}
```

```
DataRecorder
    Properties
    References
        Analyzers
        Microsoft.CSharp
        Opc.Ua.Client
        Opc.Ua.Client.ComplexTyp
        Opc.Ua.Configuration
        Opc.Ua.Core
        System
        System.Core
        System.Data
        System.Data.DataSetExtens
        System.Deployment
        System.Drawing
        System.Net.Http
        System.Windows.Forms
        System.Xml
        System.Xml.Linq
    AppCode
        DB.cs
        ReadNodeListConfig.cs
        SaveNodeValuesToFile.cs
    AppData
    Documentation
    Resources
    WinForms
        Dialogs
            ShowNodesDialog.cs
    App.config
    Common.cs
    DataRecorder.ico
    Main.cs
    Program.cs
Opc.Ua.Client
    Dependencies
    Properties
    Documentation
    Browser.cs
```

Figure 6.2: Data Recorder - new OPC UA connection

The configured OPC UA nodes have their process data values recorded according to a reoccurring timer. The code for the event can be seen in Figure 6.3. There is a separate call for reading of each node's current process data, by calling method ReadTagValue() and saving of the received data value by calling method AddToFile(). The recording of the data is done periodically at a static interval of 10 seconds, which means that some changes in process data may be not be recorded or that the same data is recorded multiple times as unchanged. The system could be improved in later versions to make use of OPC UA's subscription model, where the OPC UA server notifies the OPC UA client of changes in node process data values, eliminating the need for continuously polling the node data. The reading of the OPC UA node data again makes use of methods provided by the OPC UA .Net Standard library and these methods are used inside method ReadTagValue().

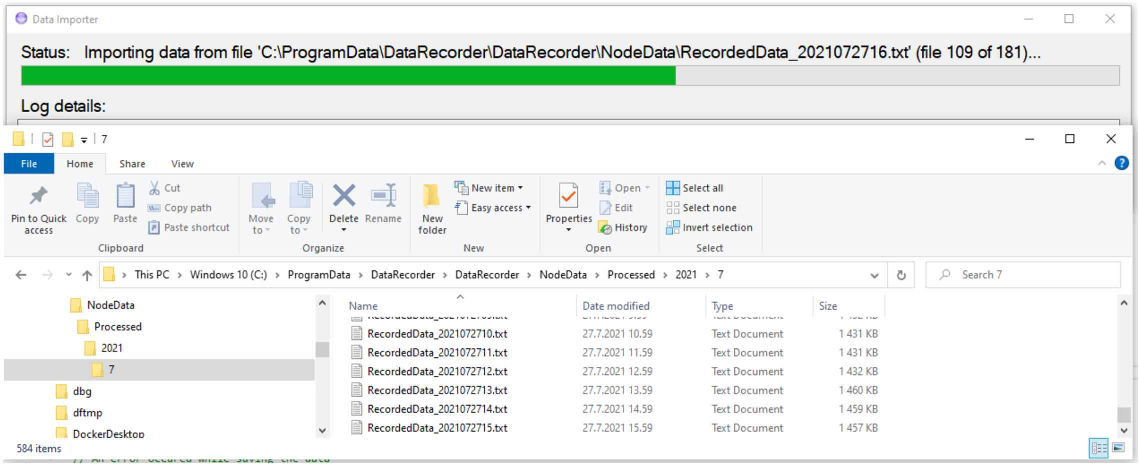


```
private void PollTagData_Elapsed(object sender, System.Timers.ElapsedEventArgs e)
{
    TagData _tagData;
    int _nodeConfigCounter;
    bool _result;
    string _errorMessage;

    // Return if the timer is busy
    if ( this.pollTagDataTimerIsBusy )
        return;

    OPC UA session validation and exit if there is no connection or session active

    // Clear the log textbox
    ClearLogBox();

    // Iterate through all configured OPC UA nodes, read and record data for each
    for ( _nodeConfigCounter = 0; _nodeConfigCounter < this.nodesToRecord.Count; _nodeConfigCounter++ )
    {
        // Read node data
        _tagData = ReadTagValue( this.nodesToRecord[_nodeConfigCounter].NodeId );
        _tagData.NodeAlias = this.nodesToRecord[_nodeConfigCounter].NodeAlias;

        if ( _tagData.TagDataReadSuccessfully )
        {
            DataRecorder.AppCode.SaveNodeValuesToFile _saveNodeData = new DataRecorder.AppCode.SaveNodeValuesToFile( out _errorMessage );
            _saveNodeData.NodeDataToSave = _tagData;

            // Save the read OPC UA Node data
            _result = _saveNodeData.AddToFile( DateTime.Now, out _errorMessage );
            if ( !_result )
                ShowInLogBox( DateTime.Now.ToString( "H:mm:ss.fff" ) + ": An error occurred while saving data for node '"
                        + this.nodesToRecord[_nodeConfigCounter].NodeAlias + "'. More info: " + _errorMessage );
            else
                ShowInLogBox( DateTime.Now.ToString( "H:mm:ss.fff" ) + ": New node data recorded for '"
                    + this.nodesToRecord[_nodeConfigCounter].NodeAlias + "' (ID: '" + this.nodesToRecord[_nodeConfigCounter].NodeId + "')."
                    + " Data: '" + _tagData.Value + "'");
        }
        else
            ShowInLogBox( DateTime.Now.ToString( "H:mm:ss.fff" ) + ": Reading of data for node '"
                    + this.nodesToRecord[_nodeConfigCounter].NodeAlias + "' (ID: '" + this.nodesToRecord[_nodeConfigCounter].NodeId + "') failed. More info: "
                    + _tagData.ErrorMessage );
    }

    // Set timer to state 'not busy'
    this.pollTagDataTimerIsBusy = false;
}
```

Figure 6.3: Data Recorder - OPC UA Node data sampling and recording

6.2 Data Importer

An additional application named "Data Importer" was created for use in the development environment. Its purpose was to import the recorded process data from the data files that the Data Recorder component created, to the local database. This application would not normally be needed, as the Data Recorder component's data would be saved directly to the system's database. This kind of application could however be useful in certain situations where the system is deployed. It might often be the case, that the system's database is located on a different server than where the data recorder component can establish a connection to the process data. A local buffer for the data could be beneficial in this kind of environment, as it would allow data to be continuously saved even in situations where the connection is temporarily lost between the Data Recorder and the database (for example because of network problems). The Data Importer application's user interface can be seen importing new process data in Figure 6.4, together with data record files already processed seen in Windows File Explorer.



Status: Importing data from file 'C:\ProgramData\DataRecorder\DataRecorder\NodeData\RecordedData_2021072716.txt' (file 109 of 181)...

Log details:

This PC › Windows 10 (C:) › ProgramData › DataRecorder › DataRecorder › NodeData › Processed › 2021 › 7

NodeData
Processed
2021
7
dbg
dftmp
DockerDesktop

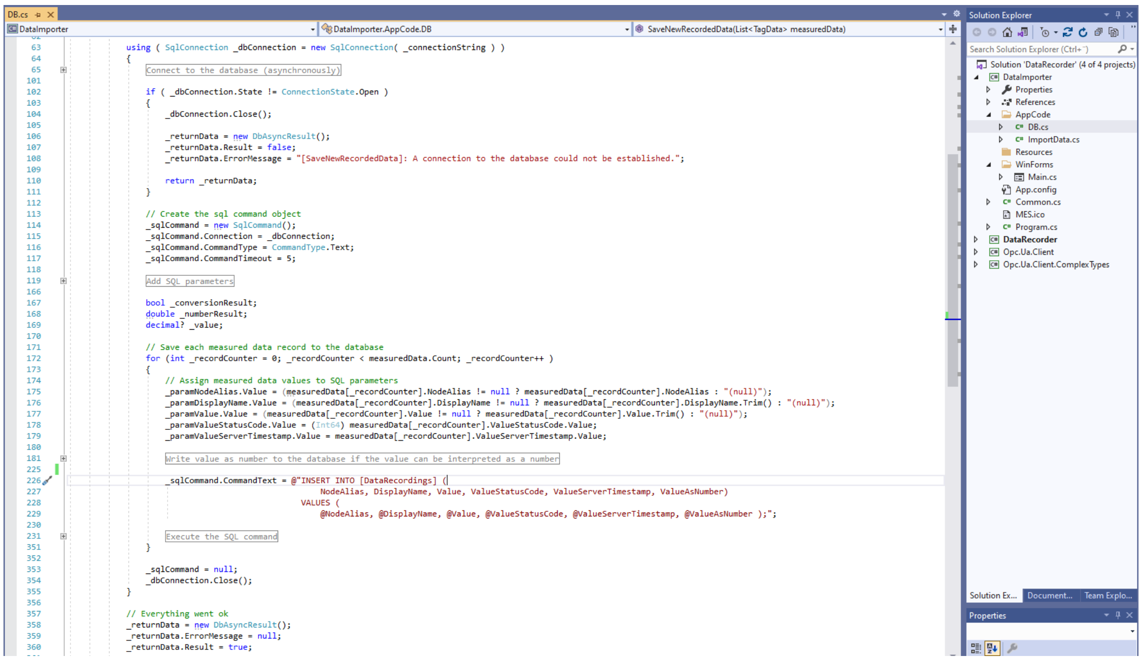
Name	Date modified	Type	Size
RecordedData_2021072710.txt	27.7.2021 10.59	Text Document	1 431 KB
RecordedData_2021072711.txt	27.7.2021 11.59	Text Document	1 431 KB
RecordedData_2021072712.txt	27.7.2021 12.59	Text Document	1 432 KB
RecordedData_2021072713.txt	27.7.2021 13.59	Text Document	1 460 KB
RecordedData_2021072714.txt	27.7.2021 14.59	Text Document	1 459 KB
RecordedData_2021072715.txt	27.7.2021 15.59	Text Document	1 457 KB

584 items

Figure 6.4: Data Importer component - saving recorded data from text files to the database

6.3 Database

Saving of data into the system's database was done using the built-in support in .Net Framework for accessing and using Microsoft's SQL server. The main code for connecting to the database and for inserting new process data can be seen in Figure 6.5. The method SaveNewRecordedData() is called separately for each node's new data and makes use of SQL parameters for saving of the data.



```
using ( SqlConnection _dbConnection = new SqlConnection( _connectionString ) )
{
    Connect to the database (asynchronously)

    if ( _dbConnection.State != ConnectionState.Open )
    {
        _dbConnection.Close();

        _returnData = new DbAsyncResult();
        _returnData.Result = false;
        _returnData.ErrorMessage = "[SaveNewRecordedData]: A connection to the database could not be established.";

        return _returnData;
    }

    // Create the sql command object
    _sqlCommand = new SqlCommand();
    _sqlCommand.Connection = _dbConnection;
    _sqlCommand.CommandType = CommandType.Text;
    _sqlCommand.CommandTimeout = 5;

    Add SQL parameters

    bool _conversionResult;
    double _numberResult;
    decimal? _value;

    // Save each measured data record to the database
    for (int _recordCounter = 0; _recordCounter < measuredData.Count; _recordCounter++ )
    {
        // Assign measured data values to SQL parameters
        _paramNodeAlias.Value = (measuredData[_recordCounter].NodeAlias != null ? measuredData[_recordCounter].NodeAlias : "(null)");
        _paramDisplayName.Value = (measuredData[_recordCounter].DisplayName != null ? measuredData[_recordCounter].DisplayName.Trim() : "(null)");
        _paramValue.Value = (measuredData[_recordCounter].Value != null ? measuredData[_recordCounter].Value.Trim() : "(null)");
        _paramValueStatusCode.Value = (Int64) measuredData[_recordCounter].ValueStatusCode.Value;
        _paramValueServerTimestamp.Value = measuredData[_recordCounter].ValueServerTimestamp.Value;

        Write value as number to the database if the value can be interpreted as a number

        _sqlCommand.CommandText = @"INSERT INTO [DataRecordings] (
                                    NodeAlias, DisplayName, Value, ValueStatusCode, ValueServerTimestamp, ValueAsNumber)
                                VALUES (
                                    @NodeAlias, @DisplayName, @Value, @ValueStatusCode, @ValueServerTimestamp, @ValueAsNumber );";

        Execute the SQL command
    }

    _sqlCommand = null;
    _dbConnection.Close();
}

// Everything went ok
_returnData = new DbAsyncResult();
_returnData.ErrorMessage = null;
_returnData.Result = true;
```

Figure 6.5: Data Importer component - saving recorded data to the database

6.4 Visualization of factory CO_2 emissions

Visualization objects in the system were created based on the Scalable Vector Graphics web-technique. Figure 6.6 shows an example of a drawn object, visualizing an electrical engine in 2-dimensional format.

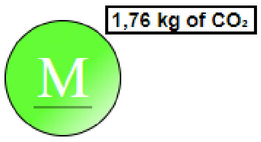


Figure 6.6: A graphical SVG representation of an electrical engine

The defined SVG element and the its inline shape elements, that makes up the picture can be seen in Listing 6.2. The Document Object Model of the visualization screen can be dynamically manipulated by the system to display additional information, such as the latest measured or estimated value of caused CO_2 emissions.

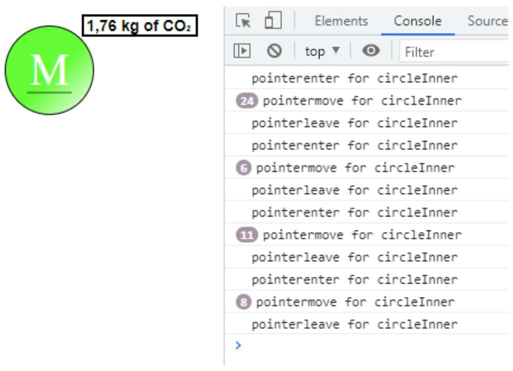
```
<svg width="300" height="100"
     xmlns="http://www.w3.org/2000/svg">

```

```
  <!-- Define a linear gradient -->
  <defs>
    <linearGradient id="bgLinearGradient" x1="55%" y1="55%" x2="100%" y2="100%">
      <stop offset="0%" style="stop-color:rgb(0, 255, 0);stop-opacity:1" />
      <stop offset="100%" style="stop-color:rgb(253, 253, 253);stop-opacity:1" />
    </linearGradient>
  </defs>

  <!-- Define text style -->
  <style>
      .textValueOutput{ font: bold 14px sans-serif; }
  </style>

  <circle cx="75" cy="50" r="40" fill="black" />  <!-- The outer 'border' for the circle -->
  <circle cx="75" cy="50" r="39" fill="url(#bgLinearGradient)" /> <!-- The center circle -->
  <text x="75" y="63" font-size="40" text-anchor="middle" fill="white">M</text> <!-- The text -->
  <line x1="55" y1="70" x2="95" y2="70" stroke="black" stroke-opacity="0.7" /> <!-- Underlined text -->

  <!-- Text field for showing caused CO2 emissions -->
  <text x="108" y="15" class="textValueOutput">1,76 kg of CO2</text>
  <rect x="105" y="1" width="102" height="18" stroke="black" stroke-width="2" fill="none" />
</svg>
```

Listing 6.2: SVG element definition for the representation of an electrical engine

User interaction with the drawn SVG element is possible, by using JavaScript and defining listeners for the object, which enables event handling for situations such as when the operator clicks the object or moves the mouse over it. This functionality enables the engineer to define additional functions to equipment in the system, such as opening modal windows displaying for example more details for the equipment. This configuration would be done in the visualization editor. Figure 6.7 shows an example of a modified version of the visualized engine equipment, where an event handler has been configured to react to mouse movements. The event handler reacts to the mouse movements by printing debug messages, that can be seen in the web browser's console window, seen on the right side in the figure. The engineer of the system would not be responsible for creating the JavaScript code, only for configuring the values to display in the modal window. The JavaScript code is part of the system, enabling operator interaction for equipment instances, but is invisible for the engineer configuring the system.

Listing 6.3 show the SVG object including the JavaScript code, that enables the event handling for the engine equipment.



```
pointerenter for circleInner
24 pointermove for circleInner
pointerleave for circleInner
pointerenter for circleInner
6 pointermove for circleInner
pointerleave for circleInner
pointerenter for circleInner
11 pointermove for circleInner
pointerleave for circleInner
pointerenter for circleInner
8 pointermove for circleInner
pointerleave for circleInner
>
```

Figure 6.7: A graphical SVG representation of an electrical engine

```
<html>
<head>
  <title>Visualization</title>
</head>
<body>

<svg width="300" height="100"
     xmlns="http://www.w3.org/2000/svg">

  <!-- Define a linear gradient -->
  <defs>
    <linearGradient id="bgLinearGradient" x1="55%" y1="55%" x2="100%" y2="100%">
      <stop offset="0%" style="stop-color:rgb(0, 255, 0);stop-opacity:1" />
      <stop offset="100%" style="stop-color:rgb(253, 253, 253);stop-opacity:1" />
    </linearGradient>
  </defs>

  <!-- Define text style -->
  <style>
      .textValueOutput{ font: bold 14px sans-serif; }
  </style>

  <circle cx="75" cy="50" r="40" fill="black" />  <!-- The outer 'border' for the circle -->
  <circle id="circleInner" cx="75" cy="50" r="39" fill="url(#bgLinearGradient)"
          cursor="pointer" pointer-events="visible"></circle> /> <!-- The center circle -->
  <text x="75" y="63" font-size="40" text-anchor="middle" fill="white">M</text> <!-- The text -->
  <line x1="55" y1="70" x2="95" y2="70" stroke="black" stroke-opacity="0.7" /> <!-- Underlined text -->

  <!-- Text field for showing caused CO2 emissions -->
  <text x="108" y="15" class="textValueOutput">1,76 kg of CO2</text>
  <rect x="105" y="1" width="102" height="18" stroke="black" stroke-width="2" fill="none" />
</svg>

<script language="javascript">

      // Get inner circle element
      const _engineSvg = document.getElementById("circleInner");

      if (_engineSvg != undefined && _engineSvg != null) {

        // Events to respond to
        const useEventType = (typeof window.PointerEvent === 'function') ? 'pointer' : 'mouse';
        const listeners = ['click','touchstart','touchend', 'touchmove'
                          ,`${useEventType}enter`,`${useEventType}leave`, `${useEventType}move`];

        // Event handler for 'circleInner'
        const pointerHandler = (event) => {
          event.preventDefault();
          console.log( event.type + ' for circleInner' );
        }

        // Start listeing to events
        listeners.map((etype) => {
            _engineSvg.addEventListener(etype, pointerHandler);
          });
      }
  </script>

</body>
</html>
```

Listing 6.3: SVG element and event handling

6.5 Code

Most of the code for the implementation was created using Microsoft's Visual Studio IDE (Integrated Development Environment). The system's components were implemented as prototypes running on Microsoft's .Net Framework 4.8 runtime and the language used in the implementation was C#. The creation of the Data Recorder component used around 2500 lines of code, while the Data Importer component was made with around 1500 lines of code. The OPC UA client library has around 1500 lines of code, but much of the functionality in the library was not used in the implementation.

Chapter 7

Discussion

With climate change being an increasing worry in the world [121], industries are more and more interested in their carbon footprint (the effect of all greenhouse gases, including CO_2) and how their activity affects the climate. The overall carbon footprint can be estimated based on information from consumed energy, regardless of what source the energy use originates from. The planned system in this thesis, for displaying caused CO_2 emissions, would provide detailed information, allowing the caused CO_2 emissions to be displayed for individual equipment or production lines. Also, the time frame of inspection for CO_2 emissions could easier be constrained to for example a production shift or a production batch. Calculating CO_2 emissions based on data such as raw energy bills, a monthly electric bill or varying quantities of purchased oil, does not allow the factory owner to know, with specific details, how and when the use of the energy resulted in CO_2 emissions.

The value of knowing the momentary amount of CO_2 emissions from a specific type of equipment, might not be so useful either, as it is natural for the need of energy use to vary given a short time frame (such as within a minute), according to fast changing demand in the load of equipment. Data values from such short time frames would better serve for example optimization requirements, which are needs usually specific for only certain equipment in a factory. Optimization would typically also make more use of the energy readings, instead of the caused CO_2 emissions data.

The planned system for displaying caused CO_2 emissions would provide most value, when used to display the the caused CO_2 emissions per factory equipment, over a time period that correlates with meaningful events in the factory, such as a production shift, a single run of a specific production line or a production batch of a certain product. The value of the grouped CO_2 emission data over a longer time period, would enable comparison between events and assigning of CO_2 emission values to supplemental data, such as data originating from a factory's MES system.

Most factories do not individually measure each equipment's power usage, which is why an accurate enough estimation of power usage and caused CO_2 emissions would be beneficial. Through manual analysis of available recorded process data and experimentation with using the data to estimate CO_2 emissions, it was found that estimation is possible, but may sometimes be challenging to achieve. In the manual estimation done in Section 5.3.1, the obtained result for used energy was close to the measured power usage for the equipment over the same time frame, indicating the success of the estimation. When doing estimation, one would always need to have some recorded or otherwise known information, known to be true and accurate, as a base for performing the necessary analysis and calculations to achieve an estimation result with a high enough degree of accuracy.

The manually done estimation in Section 5.3.1, was based on the recorded process values of the amount of the supplied lime-stone used in the factory process, which caused the load on the equipment for which the estimation was done. In this case, this amount of lime-stone extracted from a silo (silo 1006) acted as the one known information that could be used to estimate on, during a time frame where lime-stone was only extracted from the silo. This silo of lime-stone is often being filled at the same time as lime-stone is extracted, meaning that in this situation, a reliable value for the amount of used lime-stone would not be available, and as such, the estimation result would not be accurate.

To highlight the difficulties of figuring out a way to do reliable estimation, the case with silo 1006 being filled and emptied at the same time could be re-examined. Instead of using the process data value indicating the current fill level of the silo, one could analyze the ratio in which the silo level declines, in a situation where lime-stone is extracted and the silo is not being filled. This ratio (for example kilograms / minute), might very well be a constant always true, when lime-stone is extracted. Using this constant value, it would be possible for the estimation to take place also in situations where the silo is being both filled and emptied at the same time.

There are cases where the state of the environment in the factory and the available process data can prevent an estimation from taking place. Although in many cases, estimation might still be possible even though it first looks impossible, as long as one can figure out some static constant or known fact, that alone or together with process data from the surrounding factory equipment, can establish an always known point from where to start the process of calculations and finally reaching an estimation result.

Parts of factory processes can contain equipment, where the load can be thought to always be roughly the same, and where a static load could be assumed to be correct, making the time of operation to be the only variable in the estimation result. In an essence, estimation of used energy and caused CO_2 emissions, is possible, when some variable that is causing a change in load is known reliably. A lot of other factors must also be known and taken into account while estimating, physical factors that has an additional effect on the load, such as the weight of the machinery operated on in addition to the variable load. Such factors are for example the weight of conveyor belts, elevator machinery and rotating metal screws inside transportation pipes. These additional factors can be difficult to determine and must be taken into account in the estimation by use of the law of physics if needed, but once established, they are static configuration data for the equipment template and instance.

In cases where measurement of energy use for individual equipment is available, no additional load variables, calculation formulas and use of physics laws are needed. The measurement values obtained from the various energy meters, can be considered reliable as it is.

In a typical factory, only the most energy hungry equipment and equipment that plays a more important role in the production process, have energy meters installed. By evaluating the process screens of the Nordkalk factory, it was estimated that around 21% of the equipment had direct energy meters installed. The rest of the factory equipment (79%) would need to have the energy usage estimated.

The estimation of energy use and caused CO_2 emissions, based on the process data from the the factory environment and by applying calculation formulas on the data, was found to be the only way that estimation could be done by use of typical available process data. In cases where estimation is not possible, the only alternative is to install direct energy meters. A system based on this kind of typically available process data could not be implemented in other ways than what has been described.

Adding functionality making use of artificial intelligence, could perhaps perfect the estimation results in some cases, but the main problem still remains, which is that the AI-code need some data to learn from and this data is often not easily available. An alternative system would be one based entirely on measured power usage and CO_2 emission reading, but the problem with this system would be cost of procurement, installment and maintenance - all of which was aimed to be avoided by the planned system to display caused CO_2 emissions.

Based on the research and implementation, it was also possible to provide answers for the research questions stated in Section 1.1. The answers are provided as follows:

7.1 An answer to Research Question 1

Research question 1 for the thesis: ”*Is it possible to create a system to display the caused CO_2 emissions from a factory environment?*”

Based on the research done in Chapter 3, the short answer to the question would be ”***Yes***”. It was found, that energy metering and CO_2 measuring devices are available for purchasing on the market, provided by many different vendors. Many of these measuring equipment are specifically made to suite the market of industry and factories, for example by providing standardized interfaces for connecting the equipment to the different processes and automation systems in use. Estimation of energy use and caused CO_2 emissions, can to some level of accuracy, be used to supplement missing measurement values. Measured or estimated consumed energy data can be used to estimate caused CO_2 emissions, by use of conversion factors. All in all, with this available data, it will be possible to process it into a result showing the CO_2 emission caused in a factory environment.

7.2 An answer to Research Question 2

Research question 2 for the thesis: ”*How to design a dynamic system, that can be used to model any factory with the aim to display caused CO_2 emissions?*”

It was found, that dynamic features and enabling of a dynamic system, can be accomplished by a way of thinking in an object-oriented way. A system based on freely configurable templates (or types), which are then used to create instances or objects, on which a system is modeled to fit a real-life environment, can enable a truly dynamic system which could model any factory. The concept of object oriented modeling, will allow an engineer to have the tools to create any necessary elements (for any given factory), in order to be able to reach the goal of displaying the caused CO_2 emissions. The model might not always be easy to create, as it will require an understanding of the elements involved in the factory processes, how they are connected to each other and how they influence each other's operations. Complexity by itself, does not however mean that it is impossible to reach the goal of displaying caused CO_2 emissions. The accuracy of estimated emissions will also most likely be influenced by the amount and quality of available data, of all equipment having an influence on the result. More available data might however lead to more introduced complexity in the calculations required. The template created must allow for addition of rules and calculation formulas, in order to reach a required accuracy and the necessary flexibility for calculating the estimations.

The structure of the available data must be mapped to the instances of the system being modeled. For this requirement, a standardized, secure and widely supported data access architecture was found (OPC UA), which will be able to provide the necessary data for the system.

Finally, a model was created showing the main components needed, in order to create a dynamic software based system, that could be used to model any factory, with the aim to display caused CO_2 emissions.

Based on the research done in Chapter 4, the conclusion would be that it is possible to design a dynamic system, that can be used to model any factory with the aim to display caused CO_2 emissions.

7.3 An answer to Research Question 3

Research question 3 for the thesis: ”*How to technically implement a dynamic system based on the proposed model?*”.

It was found that there are many possible technologies that could successfully be utilized in order to create a dynamic system for the proposed model of the system. Some of these possible options were researched and presented in Chapter 4. Many of the options were easy to eliminate as candidates, for example XML files, because of the limitations compared to databases, and Windows Communication Foundation message communication, because of it becoming an out-dated and non-supported technology. Of the remaining options, one or a few technologies remained for each technology requirement, because of the flexibility, modern technology and public support behind them.

Through research of different technologies, it was found, that all necessary technologies exist to create the proposed dynamic system for displaying CO_2 emissions, technologies such as OPC UA, NodeJs and .NET Core/ASP .NET Core for communication, NoSql and SQL databases for data storage and SVG graphics for visualization. It was found that estimation of caused CO_2 emissions is possible, but that it requires the configuration of the equipment types and instances to include a lot of dynamic functionality. The engineer of the system would need to be able to configure rules, relationships between process equipment and calculation formulas for equipment instances, in order to obtain an estimation result.

These technologies can be utilized to successfully create a system based on components, which would interact with each other according to the plan constructed in Figure 4.8 in Section 4.5 and deliver the the functionality of the system proposed by the model.

7.4 An answer to Research Question 4

Research question 4 for the thesis: "*What kind of possibilities are there to tie the caused CO_2 emission data to produced product, such as at the level of a production line or a specific produced product?*".

The system to display caused CO_2 emissions according to the proposed model (Figure 4.8 in Section 4.5) will be able to record, calculate and store CO_2 emissions with a level of detail that would enable connecting it to the production batches produced in a factory.

The system for displaying the CO_2 emissions does not by itself have the information of production batch runs, for example information about product number and name, order quantity and planned/actual start and stop times for a batch. This information about a factory's production is available in MES systems.

The easiest way to connect the caused CO_2 emissions data to production batch data is by using timestamps. All data regarding caused CO_2 emissions is recorded with actual timestamps of the time of recording. The timestamp data together with the equipment identifier data, could be used to map the caused CO_2 emissions data to individual production batches. The timestamp data would be used to identify all caused CO_2 emissions data from the start to stop event of the production batch and the equipment identifier to filter the data for only equipment belonging to a specific production line (where the production batch was produced), if the factory process is built up of several production lines.

The data communication between the system to display caused CO_2 emissions and a MES system would take place using Application Program Interfaces, based on web technology, by use of a publisher/subscriber data exchange model.

Chapter 8

Conclusion

In this thesis, research was done to investigate the possibility of creating a system for displaying caused CO_2 emissions in a factory environment. Special attention was put on the question if it was possible to also estimate the caused CO_2 emissions for individual factory equipment, instead of always have direct energy measurement readings available. Research was done about how energy and CO_2 emissions are measured and how they could be estimated. Further research was done to find out if this data was available from typical factory processes and how it could be externally accessed. The company Nordkalk's factory (for producing slaked lime-stone) in Parainen, Finland, provided the factory process data used to analyse and to test the implemented components on.

Quantitative research methods were used to collect and analyze the information for the thesis.

Based on the research result, a structure and general plan was then created for the system to display the CO_2 emission values for factory process equipment. The implementation of the system to display caused CO_2 emissions, was constrained to implementing certain key parts of what would make up a complete and fully working system. The implementation consisted of several created components, that would make up the main important parts of the final system to display caused CO_2 emissions. The components were created using several different technologies, including Microsoft's .Net Framework

Runtime with C# as a programming language, Sequel Query Language for interacting with the data store (Microsoft SQL Server) and Scalable Vector Graphics for visualization through a Web user interface. These implemented main parts of the system, serve as proof-of-concept for the planned components of the system. They also helped to test out the technologies researched and considered to be suitable for creating the system. The technologies used did not at any point hinder the development of the components, there were no limitations found in the technologies used. A completed system configured for hundreds of equipment instances would require much more resources than the implemented components and the test runs required, but there is based on research, no reason to believe that the performance of the selected technologies would be problem.

A final completed system for displaying caused CO_2 emissions values, could not be guaranteed to provide the same added value in each factory context. This is because the success and provided value of this system, would depend on the ability of the engineer to configure the system to make maximum use of the available factory process data. A deep understanding of a factory's process, of how production is run and how the parts of the process is connected and relate to each other would be needed to configure a system to display as reliable both measured and estimated values of CO_2 emissions.

A carefully configured system, in a factory environment where both measurement and suitable estimation readings are available, would provide much detail for the factory owner, on where and how the caused CO_2 emissions originate in the factory. Knowledge of the amount of CO_2 emissions that a specific source produces, such as a single equipment, allows for a greater insight, compared to knowing the overall caused CO_2 emissions of the factory.

The detailed information can be used to help fight climate change, by identifying the most polluting sources in a factory's processes and by then focusing on improving efficiency or improving the factory processes, in order to cut down on the caused CO_2 emissions. Possibilities to use the detailed information to enhance marketing is also possible, as the level of detail on reporting of caused CO_2 emissions, allow for a link to be made between the emission data and production data. Positive marketing can be achieved by advertising achieved low levels of CO_2 emissions, for example through media news and by labeling the products with the low emission values caused by production.

This thesis serves as a successful proof-of-concept for a system to display caused CO_2 emissions, whether in near real-time or not. The next step to continue this work, would be to implement the basic functionality of all main components of the system, to produce the first working version of the system.

References

[1] The Editors of Encyclopaedia Britannica. Industrial Revolution. https://www.britannica.com/event/Industrial-Revolution, 2019. Accessed: 2019-09-04.

[2] K. Hamilton, M. Barhmbhatt, and J. Liu. Multiple benefits from climate change mitigation: assessing the evidence. Policy report, Grantham Research Institue on Climate Change and the Environment. Centre for Climate Change and Policy, November 2017. https://www.lse.ac.uk/granthaminstitute/wp-content/uploads/2017/11/Multiple-benefits-from-climate-action_Hamilton-et-al-1.pdf.

[3] United Nations. The Paris Agreement. https://unfccc.int/process-and-meetings/the-paris-agreement/the-paris-agreement, 2015. Accessed: 2020-01-05.

[4] A. Woodward. The Trump administration has started to pull the US out of the Paris climate agreement. Here's what that means and what comes next. https://www.businessinsider.com/trump-pulling-us-out-paris-climate-agreement-2019-10, 2019. Accessed: 2019-11-06.

[5] Union of Concerned Scientists. Each Country's Share of CO_2 Emissions. https://www.ucsusa.org/resources/each-countrys-share-co2-emissions, 2019. Accessed: 2019-10-10.

[6] D. Chow. Uncharted territory: Carbon dioxide expected to peak at levels last seen during pliocene epoch. https://www.nbcnews.com/science/environment/humanity-poised-set-new-record-carbon-dioxide-atmosphere-again 2020. Accessed: 2020-02-25.

[7] Earl N. Baines and Paul G. Appleton. *Industrial Pollution: Oil Spills, Toxicity and Risk Assessment*. Nova Science Publishers, Incorporated, New York, USA, 2008. https://www.waterstones.com/book/industrial-pollution/earl-n-baines/paul-g-appleton/9781604569179.

[8] Asitek Oy. Tehdasjärjestelmät. http://asitek.fi/fi/www/1018_tehdasjarjestelmat, 2021. Accessed: 2021-02-19.

[9] SAP. Enterprise Resource Planning (ERP) and Financial Management. https://www.sap.com/products/erp-financial-management.html, 2021. Accessed: 2021-02-19.

[10] Microsoft. Microsoft Dynamics 365, CRM and ERP Applications. https://dynamics.microsoft.com/en-us/, 2021. Accessed: 2021-02-19.

[11] IFS. Agile Enterprise Resource Planning Software. https://www.ifs.com/corp/solutions/enterprise-resource-planning/, 2021. Accessed: 2021-02-19.

[12] Oracle. Oracle Enterprise Resource Planning (ERP). https://www.oracle.com/erp/, 2021. Accessed: 2021-02-19.

[13] Epicor Software Corporation. EPICOR ECLIPSE - Powerful ERP for Modern Distributors. https://www.epicor.com/en-us/erp-systems/eclipse/, 2021. Accessed: 2021-02-19.

[14] Jaakko Kotiranta. Preparing for ERP implementation, April 2012. https://www.theseus.fi/bitstream/handle/10024/43763/Jaakko_Kotiranta.pdf.

[15] Aveva. Manufacturing Execution System. https://www.aveva.com/en/products/manufacturing-execution-system/, 2020. Accessed: 2020-08-22.

[16] Leanware. LeanwareMES. https://leanware.fi/en/solutions/leanwaremes/, 2021. Accessed: 2021-02-21.

[17] Epicor. Epicor Advanced MES. https://www.epicor.com/en-us/industry-productivity-solutions/modules/production-management-systems/mattec-mes/, 2021. Accessed: 2021-02-21.

[18] Luiz Affonso Guedes Diego Silva Gustavo Leitão Aguinaldo Bezerra, Ivanovitch Silva and Kaku Saito. Extracting Value from Industrial Alarms and Events: A Data-Driven Approach Based on Exploratory Data Analysis. *MDPI*, 19(12):21, June 2019. https://www.mdpi.com/1424-8220/19/12/2772.

[19] Lasse Korhonen. Project leader at Asitek Oy. http://asitek.fi/fi/www/1051_yhteystiedot, 2020. Accessed: 2020-08-24.

[20] Siemens AG. Mindsphere. https://siemens.mindsphere.io/en, 2020. Accessed: 2020-09-13.

[21] Henrik Olausson. A tool for calculating CO_2 emissions in the manufacturing industry – Use of GHG protocol. http://www.diva-portal.org/smash/get/diva2:1446855/FULLTEXT02.pdf, 2020. Accessed: 2021-01-17.

[22] Liikenne ja viestintäministeriö. Ekologisesti kestävällä digitalisaatiolla ilmasto- ja ympäristötavoitteisiin. Report, Liikenne- ja viestintäministeriö, November 2020. https://julkaisut.valtioneuvosto.fi/handle/10024/162562.

[23] J. Lodén. Methodology for efficiency analysis. Report, Chalmers University of technology, March 2009. https://www.studocu.com/row/document/jamaa\%D8\%A9-alkahr\%D8\%A9/sport-medicine/path-to-res-methodology-for-efficiency-analysis-en/15933551.

[24] Siemens. Simatic Energy Suite - Energy transparency in production. https://new.siemens.com/global/en/products/automation/industry-software/automation-software/energymanagement/simatic-energy-suite.html, 2021. Accessed: 2021-02-21.

[25] Rockwell Automation. Energy Monitoring. https://www.rockwellautomation.com/en-us/products/hardware/allen-bradley/energy-monitoring.html, 2021. Accessed: 2021-02-21.

[26] OMRON Corporation. Smart Power Monitor. http://www.ia.omron.com/products/category/energy-conservation-support_environment-measure-equipment/electric-power-monitoring-equipme
smart-power-monitor/index.html, 2021. Accessed: 2021-02-21.

[27] Siemens. SITRANS FC330. https://new.siemens.com/global/en/products/automation/process-instrumentation/flow-measurement/coriolis/sitrans-fc330.html, 2021. Accessed: 2021-02-21.

[28] Vaisala. How to Measure Carbon Dioxide. https://www.vaisala.com/sites/default/files/documents/VIM-G-How-to-measure-CO2-Application-Note-B211228EN.pdf, 2019. Accessed: 2021-01-03.

[29] National Institute of Standards and Technology. SI Units – Temperature. https://www.nist.gov/pml/weights-and-measures/si-units-temperature, 2021. Accessed: 2021-02-21.

[30] SensorsONE Ltd. hPa – Hectopascal Pressure Unit. https://www.sensorsone.com/hpa-hectopascal-pressure-unit/, 2021. Accessed: 2021-02-21.

[31] Nordkalk. Production Process. https://www.nordkalk.com/products/product-information/production-process/, 2021. Accessed: 2021-01-04.

[32] European Union. EU Emissions Trading System (EU ETS). https://ec.europa.eu/clima/policies/ets_en, 2021. Accessed: 2021-01-04.

[33] OECD. OECD - About. https://www.oecd.org/about/, 2021. Accessed: 2021-01-04.

[34] International Energy Agency. Statistics report, energy efficiency indicators highlights, 2020 edition. https://www.iea.org/reports/energy-efficiency-indicators, 2020. Accessed: 2021-01-04.

[35] Siemens. SIMOTICS electric motors for industry. https://new.siemens.com/global/en/products/drives/electric-motors.html, 2021. Accessed: 2021-02-21.

[36] OVO Energy Ltd. What is a kWh? kW and kWh explained. https://www.ovoenergy.com/guides/energy-guides/what-is-a-kwh-kw-and-kwh-explained.html, 2021. Accessed: 2021-01-04.

[37] Mike Sondalini. Effect of Process Changes on Electric Motors. https://accendoreliability.com/effect-process-changes-electric-motors/. Accessed: 2021-01-05.

[38] World Resources Institute. Greenhouse Gas Protocol. https://ghgprotocol.org/. Accessed: 2021-02-07.

[39] Jessica F. Green. Private Standards in the Climate Regime: The Greenhouse Gas Protocol. *Business and Politics*, 12(3):1–37, October 2010. https://www.cambridge.org/core/journals/business-and-politics/article/private-standards-in-the-climate-regime-the-greenhouse-gas-pro
0351883B132F7D44D3FC51536A1C1716#.

[40] dataPARC. HMI Design Best Practices: The Complete Guide. https://www.dataparc.com/blog/hmi-design-best-practices-complete-guide/, 2020. Accessed: 2021-02-28.

[41] Siemens. Distributed Control System SIMATIC PCS 7. https://new.siemens.com/global/en/products/automation/

process-control/simatic-pcs-7.html, 2021. Accessed: 2021-02-23.

[42] Plant Automation Technology. An Overview Of Distributed Control Systems (DCS). https://www.plantautomation-technology.com/articles/an-overview-of-distributed-control-systems-dcs, 2021. Accessed: 2021-02-24.

[43] ABB. ABB Distributed Control Systems - DCS. https://new.abb.com/control-systems, 2021. Accessed: 2021-02-24.

[44] Valmet. Valmet DNA - Distributed Control System (DCS). https://www.valmet.com/automation/distributed-control-system/, 2021. Accessed: 2021-02-24.

[45] Precision Digital Corporation. Back to Basics: The Fundamentals of 4-20 mA Current Loops. https://www.predig.com/indicatorpage/back-basics-fundamentals-4-20-ma-current-loops, 2021. Accessed: 2021-03-13.

[46] Electrical Classroom. Digital I/O and Analog I/O. https://www.electricalclassroom.com/digital-i-o-and-analog-i-o/, 2021. Accessed: 2021-03-13.

[47] PROFIBUS & PROFINET International. PROFIBUS overview. https://www.profibus.com/technology/profibus/overview, 2020. Accessed: 2021-03-13.

[48] Inc. Modbus Organization. The Modbus Organization. https://modbus.org/, 2021. Accessed: 2021-03-13.

[49] PROFIBUS & PROFINET International. PROFINET overview. https://www.profibus.com/technology/profinet/overview, 2020. Accessed: 2021-03-13.

[50] HMS Networks. Industrial network market shares 2020 according to HMS Networks. https://www.hms-networks.com/news-and-insights/news-from-hms/2020/05/29/industrial-network-market-shares-2020-according-to-hms-network 2020. Accessed: 2021-03-13.

[51] PROFIBUS & PROFINET International North America. THE DIFFERENCE BETWEEN PROFIBUS AND PROFINET. https://us.profinet.com/the-difference-between-profibus-and-profinet/, 2020. Accessed: 2021-03-13.

[52] Siemens AG. PROFINET in Process Automation with SIMATIC PCS 7. https://support.industry.siemens.com/cs/document/72887082/profinet-in-process-automation-with-simatic-pcs-7, 2019. Accessed: 2021-03-13.

[53] John Burke Wesley Chai, Alissa Irei. Ethernet. https://searchnetworking.techtarget.com/definition/Ethernet, 2020. Accessed: 2021-04-17.

[54] Swaroop. What is Serial Communication and How it works? https://www.codrey.com/embedded-systems/serial-communication-basics/, 2018. Accessed: 2021-04-17.

[55] Luqmanul M. What is FTP: FTP Explained for Beginners. https://www.hostinger.com/tutorials/what-is-ftp, 2021. Accessed: 2021-04-17.

[56] Techterms. CSV. https://techterms.com/definition/csv, 2020. Accessed: 2021-04-17.

[57] Trend Micro Incorporated. DDE: What It Is, What It Does, and How to Defend Against Attackers Who May Exploit It. https://www.trendmicro.com/vinfo/us/security/news/threat-landscape/dde-what-it-is-what-it-does-and-how-to-defend-against-attacker 2017. Accessed: 2021-04-17.

[58] OPC Foundation. Classic. https://opcfoundation.org/about/opc-technologies/opc-classic/, 2021. Accessed: 2021-04-17.

[59] IPCS Automation. DDE And SUITELINK Explanation. https://ipcsautomation.com/blog-post/dde-and-suitelink-explanation/, 2019. Accessed: 2021-04-17.

[60] Joe Carder. What is SQL Database? https://www.openlogic.com/blog/what-sql-database, 2020. Accessed: 2021-04-17.

[61] Red Lion. Remote Monitoring. https://www.redlion.net/solutions/IIoT/remote-monitoring, 2021. Accessed: 2021-03-14.

[62] Ixon B.V. PLC remote access with IXON Cloud. https://www.ixon.cloud/knowledge-hub/plc-remote-access-with-ixon-cloud, 2019. Accessed: 2021-03-14.

[63] OPC Foundation. Unified architecture. https://opcfoundation.org/about/opc-technologies/opc-ua/, 2021. Accessed: 2021-03-21.

[64] OPC Foundation. Members. https://opcfoundation.org/members, 2021. Accessed: 2021-03-21.

[65] OPC Foundation. Mission Statement. https://opcfoundation.org/about/opc-foundation/mission-statement/, 2021. Accessed: 2021-03-21.

[66] Microsoft. OLE background. https://docs.microsoft.com/en-us/cpp/mfc/ole-background, 2016. Accessed: 2021-03-21.

[67] OPC Foundation. History. https://opcfoundation.org/about/opc-foundation/history/, 2021. Accessed: 2021-03-21.

[68] Colin Masson. Why the OPC UA standard – and what's next? https://cloudblogs.microsoft.com/industry-blog/manufacturing/2018/04/11/why-the-opc-ua-standard-and-whats-next/, 2018. Accessed: 2021-03-21.

[69] Bernard Marr. What is Industry 4.0? Here's A Super Easy Explanation For Anyone. https://www.forbes.com/sites/bernardmarr/2018/09/02/what-is-industry-4-0-heres-a-super-easy-explanation-for-anyone 2018. Accessed: 2021-03-21.

[70] Desney S. Tan. Human-machine interface. https://www.britannica.com/technology/human-machine-interface, 2014. Accessed: 2021-02-27.

[71] David Greenfield. Understanding HMI's Evolution Toward Easier Design. https://www.automationworld.com/products/control/blog/13318548/understanding-hmis-evolution-toward-easier-design, 2018. Accessed: 2021-02-27.

[72] Integration Objects. OPC UA Client Toolkit. https://integrationobjects.com/sioth-opc/sioth-opc-unified-architecture/opc-ua-client-toolkit/, 2021. Accessed: 2021-04-04.

[73] Unified Automation. OPC UA Development. https://www.unified-automation.com/downloads/opc-ua-development.html, 2021. Accessed: 2021-04-04.

[74] OPC Labs. OPC UA Development. https://www.opclabs.com/products/quickopc/opc-specifications/unified-architecture, 2020. Accessed: 2021-04-04.

[75] Prosys. Multiplatform OPC UA Software. https://www.prosysopc.com/, 2021. Accessed: 2021-04-04.

[76] GitHub. node-opcua. https://github.com/node-opcua, 2021. Accessed: 2021-09-06.

[77] The OPC Foundation. Build OPC UA .NET applications using .NET Standard-Library. https://opcfoundation.github.io/UA-.NETStandard/, 2021. Accessed: 2021-04-04.

[78] Brady Gavin. What Is An XML File (And How Do I Open One)? https://www.howtogeek.com/357092/what-is-an-xml-file-and-how-do-i-open-one/, 2018. Accessed: 2021-06-04.

[79] Jacqueline Biscobing. DEFINITION relational database. https://searchdatamanagement.techtarget.com/definition/relational-database, 2020. Accessed: 2021-06-04.

[80] Lauren Schaefer. What is NoSQL? https://www.mongodb.com/nosql-explained, 2021. Accessed: 2021-06-04.

[81] Carey Wodehouse. SQL vs. NoSQL Databases: What's the Difference? https://www.upwork.com/resources/sql-vs-nosql-databases-whats-the-difference, 2019. Accessed: 2021-06-04.

[82] Vidya Vrat Agarwal. What, Why and How: SOA With Microsoft. https://www.c-sharpcorner.com/UploadFile/84c85b/what-why-and-how-soa-with-microsoft/, 2020. Accessed: 2021-09-05.

[83] Microsoft. What Is Windows Communication Foundation. https://docs.microsoft.com/en-us/dotnet/framework/wcf/whats-wcf, 2017. Accessed: 2021-09-05.

[84] Microsoft. Message Queuing (MSMQ). https://docs.microsoft.com/en-us/previous-versions/windows/desktop/msmq/ms711472(v=vs.85), 2016. Accessed: 2021-09-05.

[85] OpenJS Foundation. Node.js is a Javascript runtime built on Chrome's V8 JavaScript engine. https://nodejs.org/en/, 2021. Accessed: 2021-09-05.

[86] Microsoft. ASP.NET documentation. https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-5.0, 2021. Accessed: 2021-09-05.

[87] OpenJS Foundation. Openjs foundation. https://openjsf.org/, 2021. Accessed: 2021-09-05.

[88] Microsoft. Typescript is JavaScript with syntax for types. https://www.typescriptlang.org/, 2021. Accessed: 2021-09-05.

[89] Svitlana Omelia. .NET Core vs Node.js: What Should You Choose? https://inveritasoft.com/blog/net-core-vs-nodejs-what-to-choose, 2020. Accessed: 2021-09-05.

[90] Mike Butusov. Node.js vs .NET: What to Choose in 2021. https://www.techmagic.co/blog/node-js-vs-net-what-to-choose/, 2021. Accessed: 2021-09-05.

[91] Microsoft. Windows. https://www.microsoft.com/en-us/windows, 2021. Accessed: 2021-08-19.

[92] Apple. macOs Big Sur. https://www.apple.com/macos/big-sur/, 2021. Accessed: 2021-08-19.

[93] Android. Android. https://www.android.com/, 2021. Accessed: 2021-08-19.

[94] Apple. iOS 14. https://www.apple.com/ios/ios-14/, 2021. Accessed: 2021-08-19.

[95] Nick McKenna. Progressive Web Apps Explained. https://www.mckennaconsultants.com/progressive-web-apps-explained/, 2020. Accessed: 2021-08-19.

[96] The Teknicks Team. PWA vs Native App: Which is Better in 2021? https://blog.teknicks.com/pwa-progressive-web-app-vs-native-apps-which-is-better, 2021. Accessed: 2021-08-19.

[97] SVG Working Group. SVG Working Group document repository. https://svgwg.org/, 2021. Accessed: 2021-08-21.

[98] Dataquarium Speros. SVG and Canvas and WebGL, Oh My. https://dataquarium.io/svg-canvas-webgl/, 2019. Accessed: 2021-08-17.

[99] Maria Antonietta Perna. Canvas vs SVG: Choosing the Right Tool for the Job. https://www.sitepoint.com/canvas-vs-svg/, 2021. Accessed: 2021-08-22.

[100] MDN WebDocs. Introduction to the DOM. https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction/, 2021. Accessed: 2021-08-17.

[101] Jack Doyle. What on Earth Is CSS? Your Friendly Web Developer Explains. https://careerfoundry.com/en/blog/web-development/what-is-css/, 2021. Accessed: 2021-08-21.

[102] William Craig. HTML5 Canvas Element Guide. https://www.webfx.com/blog/web-design/canvas-element/, 2021. Accessed: 2021-08-17.

[103] Alex Clem. Raster vs. Vector: What's the Difference and When to Use Which. https://www.shutterstock.com/blog/raster-vs-vector-file-formats, 2018. Accessed: 2021-08-22.

[104] Mike Bostock. D3 Data-Driven Documents. https://d3js.org/, 2021. Accessed: 2021-08-22.

[105] Jakub Majorek. 19 JavaScript Data Visualization Libraries in 2021. https://www.monterail.com/blog/javascript-libraries-data-visualization, 2020. Accessed: 2021-08-22.

[106] Chart.js members on Github. Chart.js. https://github.com/chartjs/Chart.js, 2021. Accessed: 2021-08-22.

[107] Highsoft AS. Highcharts. https://www.highcharts.com/blog/products/highcharts/, 2021. Accessed: 2021-08-22.

[108] Nordkalk. Slaked Lime. https://www.nordkalk.com/products/slaked-lime/, 2021. Accessed: 2021-08-06.

[109] BEGE Power Transmission. Venanzetti Electric Vibration Motors. https://www.bege.nl/en/vibration-motors-for-industrial-use/, 2021. Accessed: 2021-08-05.

[110] SEW Eurodrive. Frequency inverters. https://www.sew-eurodrive.it/products/inverter-technology/frequency-inverters.html, 2021. Accessed: 2021-08-05.

[111] Lumen Learning. Newton's Second Law of Motion: Concept of a System. https://courses.lumenlearning.com/physics/chapter/4-3-newtons-second-law-of-motion-concept-of-a-system/, 2021. Accessed: 2021-08-07.

[112] Lumen Learning. Work and Energy. https://courses.lumenlearning.com/boundless-physics/chapter/introduction-2/, 2021. Accessed: 2021-08-07.

[113] Lumen Learning. Power. https://courses.lumenlearning.com/physics/chapter/7-7-power/, 2021. Accessed: 2021-08-08.

[114] Ibrahim Dincer. *Comprehensive Energy Systems*. Elsevier, 2018. https://www.sciencedirect.com/topics/engineering/motor-efficiency.

[115] Roulunds Fabriker A/S. Rubber Conveyor Belts. http://pmural66.ru/uploads/LenTnrasp/CONBELT.pdf, 2001. Accessed: 2021-08-10.

[116] Helen. Specific emissions of energy. https://www.helen.fi/en/company/energy/energy-production/specific-emissions-of-energy-production, 2021. Accessed: 2021-08-16.

[117] Helen. Basic Electricity. https://www.helen.fi/en/electricity/electricity-products-and-prices/basic-electricity, 2021. Accessed: 2021-08-16.

[118] Helen. Environmental electricity – a combination of renewable energy. https://www.helen.fi/en/electricity/electricity-products-and-prices/environmental-electricity, 2021. Accessed: 2021-08-16.

[119] Justinas Mazūra. VPN protocols explained: how do they work? https://cybernews.com/what-is-vpn/vpn-protocols/, 2021. Accessed: 2021-08-16.

[120] Bradley Mitchell. What Is Virtual Network Computing (VNC)? https://www.lifewire.com/vnc-virtual-network-computing-818104, 2020. Accessed: 2021-08-16.

[121] BBC News Matt McGrath. Climate change: Europe's 2020 heat reached 'troubling' level. https://www.bbc.com/news/science-environment-58333124, 2021. Accessed: 2021-09-04.