SHOPLFOOR ORGANIZATION AND MANAGEMENT SYSTEM

Developing a software system for a matrix-production system

Abstract

The production industry is undergoing rapid change. A more volatile market together with a shorter-lived window of opportunity put pressure on the sector to adapt and become more flexible. A new trend emerged, which some call industry 4.0, which is marked by big data collection and digitalization. The Fraunhofer Project Center is planning on creating a matrix-production shopfloor which implements these new technologies. Many contemporary systems are not built for the high flexibility of matrix-production; therefore, a new type of system is required. In this thesis a new software system is developed that can support a matrix production shopfloor. The integration between planning, organizing, tracking and initiation is the main focus of the new system. The result of this thesis is a prototype which forms the basis of the envisioned end product. Some features of this end product are outlined and evaluated. The prototype uses a novel approach to modelling the production process which was received well by the users. With this foundation different charts and statistics can be generated. These statistics help the planner to optimize the matrix-production system beforehand, resulting in a more efficient system. The prototype and envisioned end product can serve as a starting point for further development as they were received positively by stakeholders and users.

Emiel Rous

Creative Technology

Date: 02-07-2021

Supervisor:
Job Zwiers

Critical observer:
Demitriana Minassian

TABLE OF CONTENTS

CHAPTER 1 – INTRODUCTION

A recent trend in the consumer market is the higher demand for customized products. While globalization increases the size of the market, it also creates a higher demand for customizability since one product has to appeal to a wide range of buyers from many different cultures. This, together with a more volatile market and a shorter-lived window of market opportunity, creates a new challenge for the manufacturing industry [1], [2]. Consequently, a new trend in industry emerged, which some call the fourth industrial revolution, or industry 4.0. The fourth revolution is marked by digitalization and big data collection allowing for highly flexible and customizable mass production. It uses the latest developments in the digital world to create decentralized production facilities that are self-regulating, self-configuring, self-optimized, and intelligent [3].

This new development in manufacturing is what the Fraunhofer project center wants to incorporate in the new building they will occupy near the University of Twente campus. They aim to show the latest developments in manufacturing technology by creating a matrix production shopfloor. A matrix production shopfloor is an implementation of industry 4.0 principles. Highly configurable machines are placed in a matrix structure setup to allow for high flexibility due to the elimination of equal cycle time [4]. The transportation between machines is done by automated guided vehicles (AGV) to create a decentralized system that is self-organizing and self-optimizing [5].

Matrix production systems allow for higher throughput by minimizing the machine idle times, however, this requires more meticulous planning since there are more factors to consider compared to dedicated production lines. Increasing production is the main goal of any manufacturing plant. Different methods can be used to increase the system throughput, like identifying opportunities for adding products to an existing production line [6], simulating the system beforehand [7], using an interactive planning tool to configure the production data [8], and using proper machine execution software (MES) to initiate and track the production process [9]. However, none of these approaches focus on the product selection beforehand nor do they incorporate all daily management tasks that a planner is faced with such as process tracking and

process initiation. There is a need for an MES that covers all the features of industry 4.0 The requirements for such an industry 4.0 compliant MES are that it interconnects all the equipment seamlessly to enable automated decision making, provides contextual information, and allows for high flexibility and modularity [10].

Improving the throughput of a matrix-production system is one goal of supporting software but flexibility is the main difference between a dedicated production plant and a matrix production plant. When a shopfloor planner manages a dedicated production line there needs to be only one good solution for the planning and scheduling of one product needs to be developed. Inefficiencies can be progressively resolved making the process more optimized. Each machine needs to be configured once and is likely only tweaked now and again. On the other hand, in matrix-production, the production system produces multiple products simultaneously and products are added and removed constantly. This requires more planning and organizing and calls for a tool that focuses on flexibility. An added factor in this specific case of a matrix-production system is that it acts as an exemplary prototype. This means that producing many products very efficiently is less of a priority. The main selling point of the new production technique is flexibility so this should be reflected in the software system supporting it too.

To summarize, there are a few requirements for industry 4.0 compliant MES. Selecting the right products and ensuring the system is operating at full capacity are difficult tasks, consequently, there is a need for an MES that helps the planner organize, plan, initiate and track the production processes. In line with the requirement put forth by Jaskó et al. [10], the system must keep track of each product, what processes are required for this product and which machines are capable of executing these processes. Additionally, the system must also be able to interact with the machinery to initiate the production process and track its progress. Creating such a system is out of the scope of this bachelor thesis and therefore this thesis will focus on creating a prototype to help the planning and organizing of the production processes and creating an outline of the envisioned product. The prototype will be the foundation of the eventual end product.

Therefore, the goal of this thesis is divided into two main parts: the prototype and the envisioned end product. The first research goal of this thesis is *to develop a software system that helps the*

planner organize and plan products with their required processes which can be executed by different machines. This first goal can be divided into two sub-goals. The first sub-goal is *to create a database structure that represents products, their processes, machines, and their dependencies,* which would serve as the backbone of the entire application. The second sub-goal *is to create an interface in which the planner can manage products, processes, and machines in the database.* The second goal of this thesis is *to outline what features the envisioned end product could have*.

The first chapter of this thesis contains some background research and state of the art review. The conclusions of this research will be the starting point of the ideation chapter. The ideas of this chapter are then put into requirements in the specification chapter. The fifth chapter describes the iterative design process, and the evaluation chapter thereafter tests the viability of the realized prototype and envisioned product. Lastly, there is a conclusion and recommendations for future work.

CHAPTER 2 – BACKGROUND

For the organization and planning of matrix production systems, or any industry 4.0 compliant production system [7], [11], simulation and modeling are crucial. Modeling can be very useful to maintain an overview of the system, its inputs, and its outputs. The large, often risky, investment to create a matrix production system can be made more attractive by simulating the system beforehand. For tracking and initiation, MES software can be used [9]. In this chapter, a few of the approaches to modeling and simulation and some MES systems are analyzed.

Preparatory to this chapter, a distinction has to be made between reconfigurable and matrix production systems. Reconfigurable production systems are production systems that are built for scalability and flexibility [1]. Hence, matrix production can be seen as an implementation of reconfigurable production. Research towards simulation and modeling of matrix-, or more generally, industry 4.0 manufacturing systems has only very recently been done [11], therefore some literature regarding reconfigurable production is also analyzed.

CLASSICAL MODELLING AND SIMULATION METHODS

One tempted approach to model a production system is with the use of Petri Nets (PN). A PN is a directed graph with two types of nodes: places and transitions. The PN consists of a set of places, a set of transitions, an input matrix (which defines the weight of the arcs from the places to the transitions), and an output matrix (which defines the weight of the arcs from transitions to places). Each place can also contain any number of markings. These markings are transferred for each evolution of the PN. These markings can be seen as parts and the transitions as processes [12]. An approach to apply this to the modeling of a flexible manufacturing system is proposed by Colombo et al. [12]. They use a Temporized Petri Net (TPN). A TPN is a PN with an additional stochastic firing delay on each evolution of the PN. With their method, they can model, perform qualitative and quantitative analysis and implement a real-time control structure of a flexible production system. However, Colombo et al. [12] have developed their method for a flexible production system, which is essentially different from a reconfigurable manufacturing system since the latter is highly prone to, and by design, build for rapid change [1]. Additionally, TPNs use a random firing delay, not a predetermined cycle time. This cycle time is what determines a transition in a matrix production system.

There are also similar PN models used for reconfigurable production systems [13]–[15]. One proposed by Aping and Nan [13] uses an extended time-place Petri Net (ETPN) and a genetic algorithm (GA) to create a scheduling solution. However, this is not applicable for a decentralized system as the scheduling is variable in such a system. Therefore, the use of PNs is not very suitable for modeling the matrix production system as the robustness of PNs does not allow for real-time, stochastic analysis of the system.

Another method for modeling a production system is using unified modeling language (UML). The use of UML is part of the large-scale emergence of the object-oriented design approach of manufacturing systems [16]. The advantage of this approach is that UML is easily understood by both professionals as well as non-professionals [17]. One approach utilizing UML is proposed by Ismail et al. [17]. Their method is used to model the information and material flow as well as

collecting relevant information, however, their approach lacked flexibility. Another, UML-specific disadvantage of their approach is that it is very different from what is standardized in the production sector. This makes UML not a great modeling or simulation tool.

Combining the beforementioned PNs and UML diagrams is proposed by Li and Chao [18]. They use an object Petri Net (OPN) in combination with UML diagrams to model the entire reconfigurable production system. The advantage of this method is that the logical model is done with UML and the physical model by OPN which means that the entire process is modeled. A downside of this is, however, that the two modeling techniques are not integrated with one another. They model different parts of the system but do not depend on each other, thus, the downsides of UML and PNs are still present.

RECENT MODELING AND SIMULATION METHODS

A more recent development in terms of standardization of a model for a reconfigurable production system is proposed by Tavola et al. [19]. In their approach, they use different types of entities in combination with Finite State Automata to model the production system. They then continue with simulating some of the processes using Excel. An advantage of their approach is the fact that they attempted to standardize the representation of a production system, which could be very useful in future research. A downside is, however, that their approach is done mostly manually, and a software implementation of the approach is yet to be made.

Simulating matrix production cannot be done with previously used simulation software. The software that has been developed so far has not been able to simulate the flexibility and individual cycle time [7]. Therefore Schönemann et al. [7] have proposed a simulation method using an agent-based structure. This allows for a good simulation of a matrix production system since the agents have a simulated control logic comparative to the one used in the actual system. Schönemann et al. have simulated the system using AnyLogic software. There has yet to be an application which is dedicated to the simulation of a matrix production system, but a lot more

research has to be done to incorporate more of the production process[7]. Nevertheless, their approach seems very promising.

MES

MES is software that combines production planning and equipment control. The definition given by the Manufacturing Execution Association (MESA) is software that guides, initiates, responds to, and reports plant activities [20]. This means it can track the progress of products, initiate the production process and display live machine status. However, most of the MESs are not suitable for matrix production since they do not support a decentralized system [10].

There are a few examples of cloud-based MES that cover some of the tracking and planning. One example of tracking MES is made by Barata et al. [21]. Their software integrates methods like QR-code scanning to facilitate traceability of products. This allows for a constant knowledge of the stage of the product. A downside of this approach is, however, that it does not integrate with the equipment. Additionally, the product stages are sequential which is not always the case in a matrix production system.

Another example that covers the planning and statistics generation is from Sun et al. [8]. They also developed a cloud-based application, but in contrast to traceability, their main focus was planning. Their application allows the planner to allocate resources, use historical data to generate optimization charts, and maintain an overview of all the orders. This approach does not, however, focus on the products themselves, mostly on the resources. One of the advantages of matrix production is the ability to quickly change what products are being produced, but this approach does not allow for rapid changes since it is more focused on mass production.

CONCLUSION

A lot of research has already been done towards modeling and simulation, planning and organization, and tracking and initiation. For modeling and simulation, there are some older methods, used for reconfigurable production systems. These methods attempt to simulate the

system in a very systematic way, like with Petri Net models, and others which model the logic of production using UML diagrams. Neither of these approaches are sufficient to capture the stochastic nature of a matrix production system. Therefore, new object-oriented approaches were introduced but these are still in their development phase and no implementation of these methods exist. There exist different approaches to MES but most of them are inadequate to control and track the complexities of a matrix production system. Additionally, no MES takes into account the highly changeable product portfolio which is, especially in the case of a demonstrative shopfloor, very important.

This emerging field of research still has a long way to go before there are industry-wide standards and dedicated software. Since a lot of the industry change has happened very recently, and the modeling and simulation techniques of previous manufacturing systems are lacking in terms of flexibility, there is a gap for an MES-like system that can cover the most important features of matrix production. This thesis is focused on creating such a system. The approach will be more product-centric, meaning the product is the starting point for modeling and simulation. The envisioned system will be a new kind of MES, which allows for control of a decentralized system and can model the synchronicity of product production. The goal is to create a system in which the planner can identify new opportunities in the products they can produce, with a similar outcome to the method proposed product generation method by May et al. [6]. The prototype system created does not cover all parts of the system but will be a foundation from which tracking, simulation, and initiation will be possible.

CHAPTER 3 – IDEATION

PROBLEM STATEMENT

The problem can be divided into three sub-problems: Planning and organizing, progress and machine status tracking, and machine instruction. Finding solutions to the first two sub-problems

can help identify bottlenecks or inefficiencies in the shopfloor while the third sub-problem is part of the daily management of the shopfloor planner.

In this chapter, all sub-problems will be analyzed more in-depth and possible solutions will be presented. The last section will contain a concluding statement about the chosen solution and why this was chosen.

PROBLEMS AND SOLUTIONS

PLANNING AND ORGANIZING

Different from a dedicated production line is the high flexibility of matrix-production. This is even more important when considering that the shopfloor for which this software will be developed would serve as a prototype. This will mean that high throughput is less of a priority and that flexibility should be the main focus of the shopfloor and should also be reflected in the software system. The shopfloor will be producing many different products so adding and removing products will be important tasks for the shopfloor manager. Additionally, a matrix-production plant can produce many products at the same time and the processes required for these products can be done on multiple machines. The first problem is therefore to maintain an overview of the shopfloor, what it produces, which machines are used the most given the set of products, and which processes can be executed by which machines.

SOLUTION 1A: ASSEMBLY GRAPH CREATOR

One way of representing a product is by way of an assembly graph. An assembly graph is a directed graph that consists of nodes and these nodes represent processes. When two nodes are connected that means that the starting node is the process

that precedes the end node. This way processes that can be executed in parallel can easily be spotted since they would be in different branches. Also, nodes can have properties such as a weight to indicate how long they take. The idea is to create a diagram drawing tool with which the planner can create such a diagram. An example is shown in Figure 1.

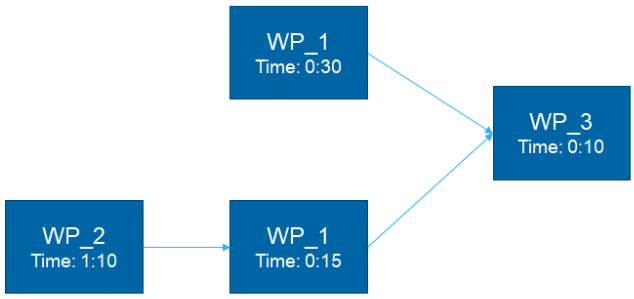


Figure 1: Solution 1a: assembly graph creator (WP=work package)

Each node represents a work package that is the same as a process. In each node, the time the process will take is displayed.

SOLUTION 1B: ASSEMBLY GRAPH CREATOR WITH PRODUCT STAGES

Similar to solution 1a, this idea also involves creating an assembly graph. The difference is that in this graph there are two types of nodes: process nodes and product stage nodes. An example can be seen in Figure 2.

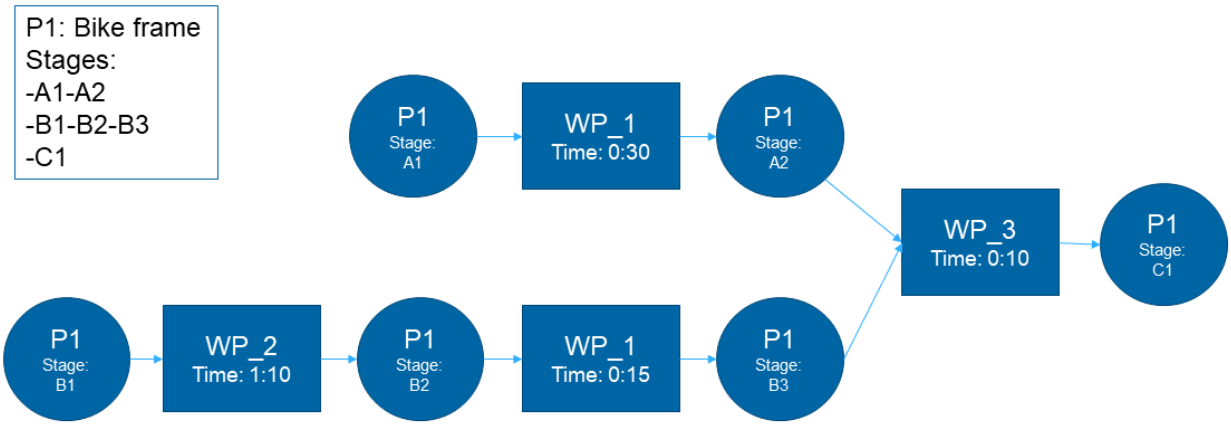


Figure 2: Solution 1b: assembly graph creator with product stages (P1=product 1, WP= work package)

Here the rounded nodes are the stages the product can be in. Stage *A1* can for example be metal powder. Then, if f *WP_1* was additive manufacturing, stage *A2* could be a printed steering wheel. This way of creating a diagram would allow you to see an overview of all the stages a product is in, like the one that is seen on the top left of Figure 2. Another slight variation on this method could be approaching it from a product stage point of view. In that case, the planner can create a product by simply specifying which stages the product can be in and the assembly graph would be generated based on the differences between the stages. So, if we use the previously mentioned example where stage *A1* is metal powder, then all the planner has to do is specify that stage *A2* is a steering wheel, and the system could fill in the work package.

SOLUTION 1C: VARIATION ON ASSEMBLY GRAPH CREATOR WITH PRODUCT STAGES

One of the features of a matrix-production system is left out in the previous two solutions and that is the fact that some processes can be executed on different machines. In the diagrams of solutions 1a and 1b, there is no information about which machines can execute each process. Another problem is that each machine might have a different completion time for a process. A solution to this is to replace the processes (or work packages) with the individual machines. See Figure 3 for an example.

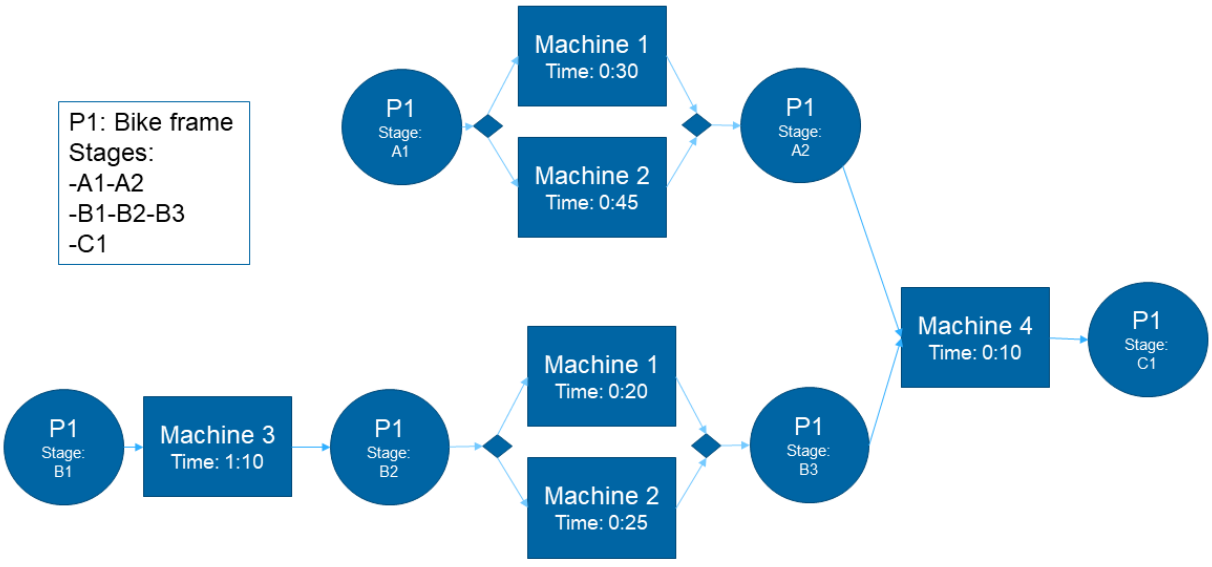


Figure 3: Solution 1c: variant assembly graph with stages (p1= product 1)

In this diagram, all of the problems of the first two solutions are solved at the cost of higher complexity. In the example, only two possible machines have been given for two of the processes but there might be more machines.

PROGRESS- AND MACHINE STATUS TRACKING

Knowledge of the current status of the products and the machines is important information for the shopfloor planner. Knowing where a product is in its production process and knowing which machines are working and which are idle is important information for identifying inefficiencies. In a dedicated production line, the important information is the status of the machines. Every product is dedicated to a certain production line and the most important information is the throughput and the maintenance status. This is different in a matrix-production system. Here there is no dedicated path for the products and the knowledge of where a product is in its production cycle becomes an extra variable in the system.

Next to this, machines can be idle if the produced products are not selected properly. This is another variable introduced to the planner. The total duty time of a machine in a given timespan can be of great importance when deciding which product should be produced by the system. As an example: if three different products are continuously produced and they only use two out of the four machines, there is a clear inefficiency. The planner should look for products that are produced using idle machines to maximize the throughput of the entire system. Keeping track of machine occupancy rates is therefore important.

Lastly, in a matrix-production system, the pathing can be very different for each product. Each product has its own set of processes it needs to go through. Next to this, the pathing can be different for the same product based on the state of the system. An AGV can choose the next machine based on factors like which machine is available. This makes the pathing dynamic and thus very hard to predict.

SOLUTION 2: PRODUCT PROGRESS TRACKER

This solution is focused on tracking the product progress. An example can be found in Figure 4. The way this would work is that all product information from the machines would be collected in one location and the diagram would update its visuals to correspond to the current stage of the progress. In addition to tracking the products, this visual representation can also be driven by a simulation. This would require integrating with external software or the creation of simulation software but it would help the planner to identify chokepoints beforehand.

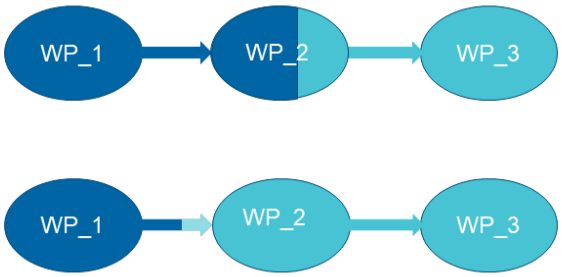


Figure 4: Solution 2: Product progress tracker

Additionally, it has to be remarked that this solution is very easily combinable with solution 1 since the graph-like structure of the diagram is very similar in both cases.

SOLUTION 3A: MACHINE STATUS TRACKING

This solution is based on integrating with the machines on the shopfloor to display live data about their status. It is similar to solution 2 in that it requires integrating with the machines and collecting data about the product they are working on with the addition of a status or maintenance indicator. See Figure 5 for an example.

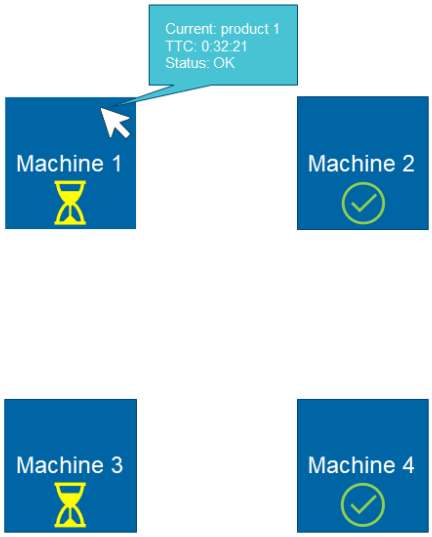
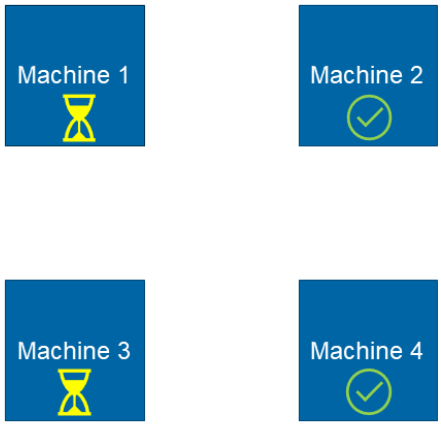


Figure 5: Solution 3a: Machine status tracking

In this interface, the hourglass indicates a machine is working on a product and the checkmark indicates a machine is idle. There would be the option to hover over a machine to see more details about which product it is working on, how long it will take to complete the product (ttc), and the status of the machine.

SOLUTION 3B: MACHINE STATUS TRACKING WITH DATA ANALYSIS

This solution builds upon solution 3a. It would have the same functionality with an additional data analysis feature. The idea is that all occupancy data will be collected and can be analyzed. The percentual duty time of a machine can for example be calculated for one day as seen in Figure 6. Alternatively, machine uptime can be calculated.



Machine	Occupancy/day
Machine 1	20%
Machine 2	60%
Machine 3	10%
Machine 4	3%

Figure 6: Solution 3b: Machine status tracking with data analysis

SOLUTION 3C: MACHINE STATUS TRACKING WITH PRODUCT PATHING

This solution is also an overview of the machines of the shopfloor, but it displays the pathing for all the products. This solution could be an extra feature of solution 3a, but it does not necessarily have to be. An example can be seen in Figure 7. There are two ways this can be implemented. One is similar to solutions 3a and 3b. With this method, the progress of each product can be displayed as a location on the shopfloor. In the example in Figure 7 *product 1* is being produced simultaneously two times where one product is at edge 1 and the other at edge 3 since the machines succeeding these edges are in progress. Another possible use of this method is

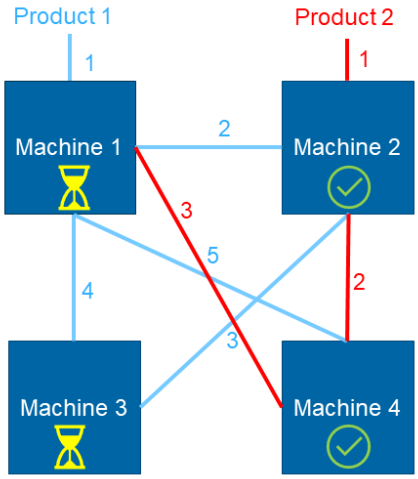


Figure 7: Solution 3c: Machine status tracking with product pathing

to get a preferred path for a product by, for example, making the edges that are traversed the most often thicker.

The second method of using this diagram is in combination with the product assembly graphs of solution 1. If the product diagram is made then the pathing for each of the products can be generated onto a grid view of the machines. Implications of this method are again the problem of pathing when a product can choose a machine. If all these alternative paths are to be displayed then this diagram becomes very cluttered.

MACHINE INSTRUCTION

The last problem that is considered in this paper is the problem of machine instruction. Every machine needs to execute certain instructions based on the product that is presented before them. For each product, there are likely instructions in some file format but distributing these can be a difficult task.

SOLUTION 4: MACHINE INSTRUCTION FILE DISTRIBUTION

This solution consists of an interface and a database. The interface can be used for creating a process list each with its own instruction file. An example of this can be seen in Figure 8.

The different numbers indicate the different steps of the system. The first step is creating a list of processes that are required for a product in the interface. Here the time to completion can be added, as well as the instruction file that each machine needs. This list and each of the instruction files is then uploaded to a database. The second step is also done by the interface and that is to generate a QR code that can be given to either the AGV or be put on or near the product. The third step happens on the actual shopfloor. The machine scans the QR code of the product that the AGV delivers to it. It then requests the file that is required for that product at that time and executes this instruction. Note that the database should in this case also track where the product is in its progress.

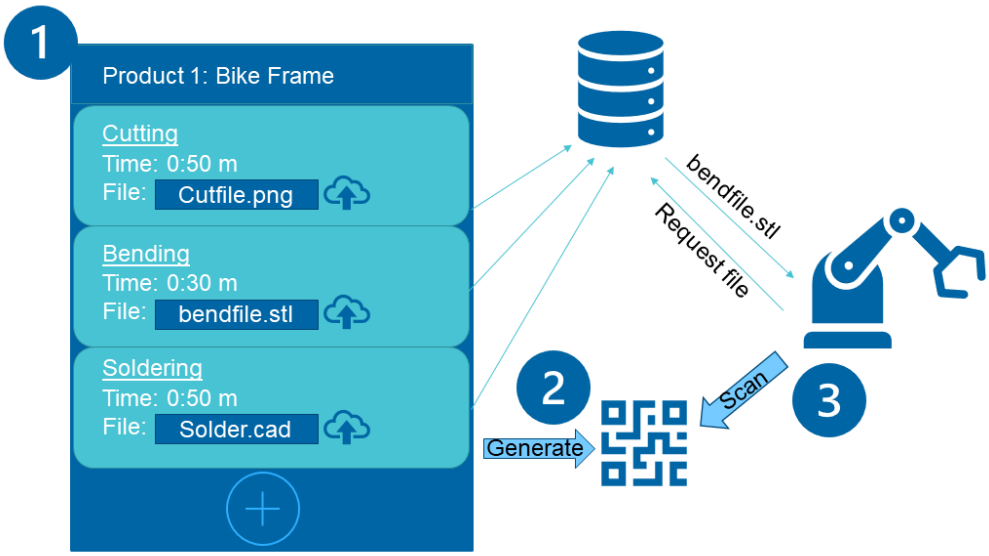


Figure 8: Solution 4: Machine instruction file distribution

CHOSEN SOLUTIONS AND ENVISIONED PRODUCT

The envisioned product is a combination of some of the solutions that are described earlier. The biggest problem that the stakeholders mentioned they had was the instruction distribution. They gave an example of a modern shopfloor where notes were passed around and each person would set up each machine manually. Therefore, the basis of this project is solution 4. However, the envisioned end product, which would be out of scope for this project, would also include parts from solutions 1, 2, and 3. Most products can be represented as an assembly graph and therefore the simple list format of solution 4 would not suffice to capture the synchronicity of the processes. Also, the material, completion time, and dimensions become important when planning a product and therefore some combination of the solutions 1a-c would be preferable. Live product tracking such as the one proposed in solution 2 would also be very nice to include in the product as well as simulation capabilities.

Lastly, the gird view of the shopfloor is very easily combined with solutions 1 and 2 and would be a very useful addition to solution 4. If the machine can communicate with a database, then updating the system with live data would be a logical next step.

CHAPTER 4 - SPECIFICATION

In this chapter, the specification of the product will be given. The first part will be the prototype outline in which the core functionality and value of the system are given. The second part will be about the requirements which are divided according to the *MoSCoW* method [22]. The last part will discuss the first wireframe of the product and the feedback from the stakeholders.

PROTOTYPE OUTLINE

The system's main functionality is the creation of links between products, processes, and machines. The system has three places for input. The first is the machines input. The machines can be put into the system with their required details such as name, manufacturer, dimensions but also location on the shopfloor. The next place for input is the process type input. Here all the process types can be put into the system with details such as the material that can be processed and which machines are capable of executing that process type. This will create a link between the machines and the process types. The last place for input is the product input. Here the user can add a new product with some details like name, description, and the material. The user can also build a production flow consisting of the different process steps required for creation. Process steps are the execution of process types and have the extra option to specify a time to completion. A schematic overview of this design architecture can be seen in Figure 9.

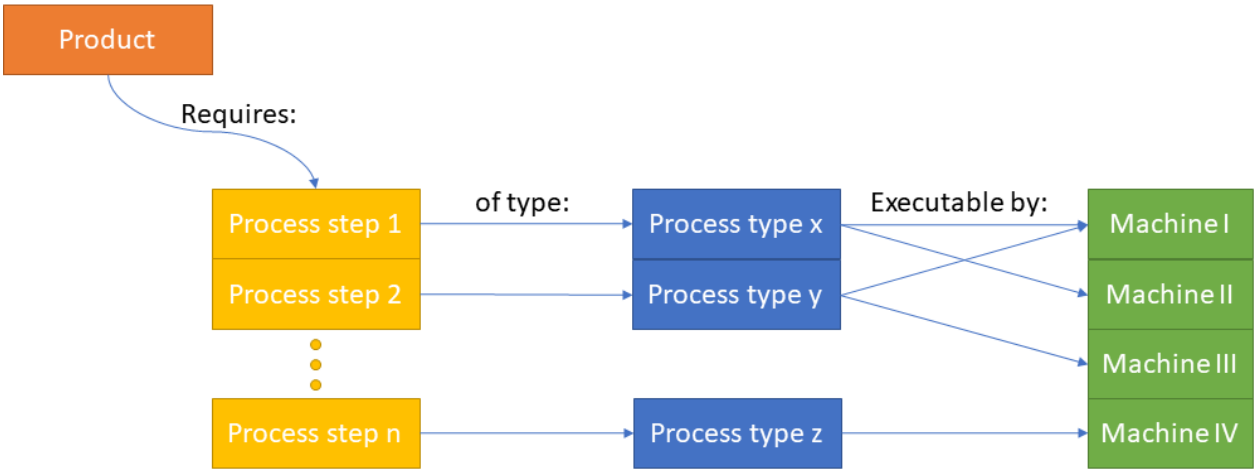


Figure 9: A schematic overview of the design architecture

REQUIREMENTS

Now that the core functionality of the prototype has been outlined, the requirements can be established. To do this the *MoSCoW* method [22] is used. This method divides the requirements into ones that must be included, should be included, could be included, and will not be included. A brief reasoning for the requirements in each category will be given after each list. Furthermore, all the requirements that are not met by the prototype, will be part of the envisioned end product.

THE NEW SYSTEM **MUST** INCLUDE THE FOLLOWING

- Ability to create a production flow for a product using process steps
- Ability to add process types
- Ability to add machines
- Ability to assign process types to machines
- Ability to get an overview of all the machines
- Ability to get an overview of the products

The requirements above are needed to create the core system functionality. Completing these requirements would mean the first main goal of this thesis is reached. The adding of the process types, products, and machines but also creating an overview is included.

THE NEW SYSTEM **SHOULD** INCLUDE THE FOLLOWING:

- Ability to get a graphical representation of the machines
- Ability to get a graphical representation of the products with their flows
- Ability to generate a chart of the machines with their process steps over time

These requirements will give the planner some more insight into the entire system. The ability to create a graphical representation gives insight into things such as which process steps take the longest and which machines are used most often. Generating charts such as Gannt charts for the

process steps, pie charts for the machine occupancy, and burn down-charts gives the planner an even better understanding of how the entire system functions over time.

THE NEW SYSTEM **COULD** INCLUDE THE FOLLOWING:

- Ability to integrate with the machines
- Ability to generate statistics of the shopfloor
- Ability to add cost per process
- Ability to generate an overview that could be displayed on a big screen

Integrating with the machines means receiving data about their current status and progress. Once this data is collected statistics can be easily generated. The ability to integrate with the machines would tie in very well with the stakeholders' desire to use the software system in real-time on the shopfloor since it combines the functionality of a predictive overview and a real-time overview of what is going on in the production system.

THE NEW SYSTEM **WILL NOT** INCLUDE THE FOLLOWING:

- Ability to simulate the production process
- Ability to group processes steps

Simulation is an important step to optimize a shopfloor but adding this functionality, even if it is just illustrative, is out of the scope of this thesis. Simulation requires the modeling of the AGVs and their ability to make decisions and this is something that requires a lot of research.

The ability to group process steps that are required for one component would require a complex frontend tool and one that would likely have to be developed from scratch. This is not something that fits in the timeframe of this thesis.

WIREFRAME

The basic concept of the application with the requirements is now known. The next step is to create a mockup product on which the stakeholders can give feedback. The different inputs of the first part of this chapter are machine, product, and process type input. These inputs are divided into three different pages in the first wireframe sketch.

The first input page is the machine input page which can be seen in Figure 10. The arrow denotes a pop-up window. A list of machines is given on the left and once a machine is selected the preview will be next to it. On the far right, the preview options are shown which would replace the 'preview task' list. When a new machine needs to be added the plus button will open a pop-up window where the details of the machine can be input.

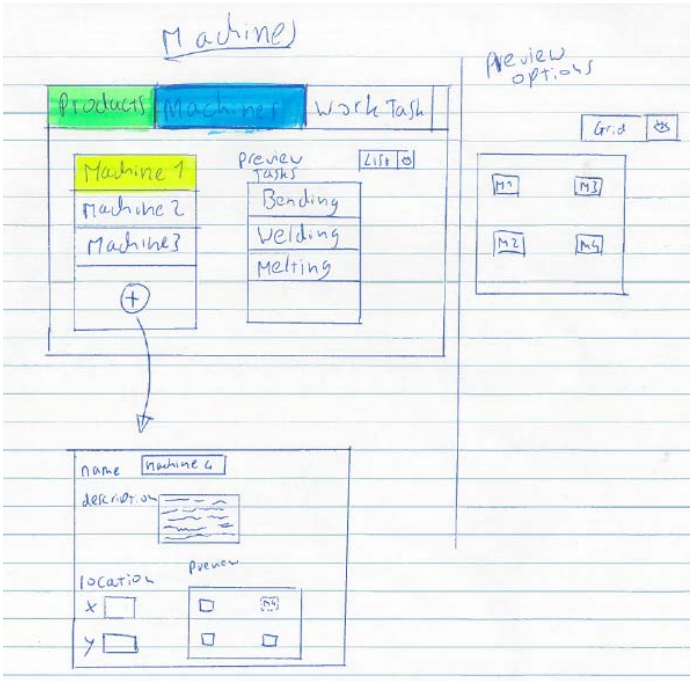


Figure 10: Machine input page

The second input page is the process types page which can be seen in Figure 12 (in the sketch process types are called 'work tasks'). A list of the different process types can be seen on the left of the main page. Once a process type is selected, the list of machines that can execute the

process is shown next to that on the machine preview list. When a new process type is added a name, description, machine list, and material list can be added.

The third input page is the product page. This can be seen in Figure 11. Here there are a few more preview options like generating a Gantt chart or a timeline. When adding a new product flow the process steps can also be added. These have a process type and a time to completion (ttc).

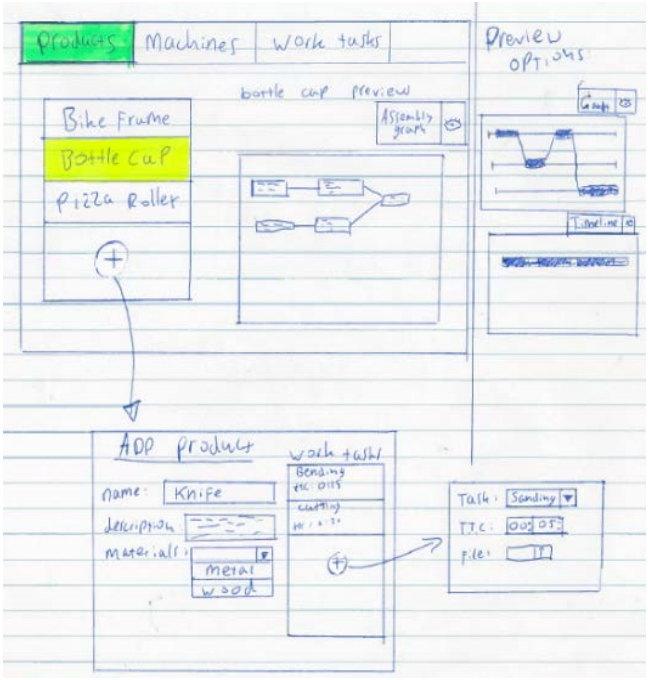


Figure 12: Product input page

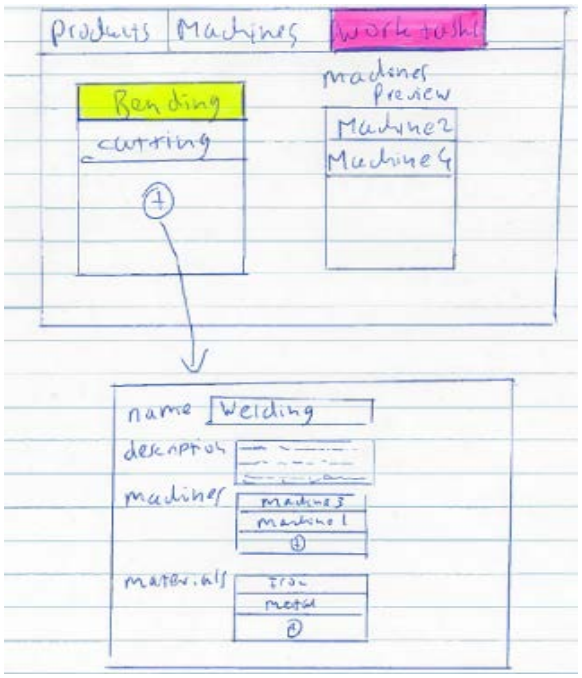


Figure 11:Process types page

FEEDBACK

These wireframes were presented to the stakeholders and their feedback was solicited. They were positive about the idea and the outline of the product. In their opinion, the hardest part of planning and organizing a shopfloor is keeping an overview. There often is one person who defines the process flow and passes this around to the people responsible for certain process types. This transfer of information is very inefficient and is very prone to error. Therefore, the prototype that was proposed to them seemed very useful. They especially liked the ability to

generate a Gantt chart since this would allow them to plan new products with an insight into the restrictions of the availability of the different machines. They agreed that the product would be mostly practical in nature but also suggested the option to show a part of the software system on a large display next to the shopfloor. This last feature was added to the 'could' requirements.

CHAPTER 5 – REALIZATION

This chapter will cover the realization phase. It describes the iterative process of implementing requirements, getting feedback, adjusting the requirements, and implementing the adjustments. This phase was made up out of one preliminary phase, and three iteration phases of one or two weeks. After each iteration, there was a meeting with the stakeholders where a demo was presented on which they could give feedback and could suggest new requirements.

In the preliminary phase, decisions were made in determining the right tools, conceptualizing the structure of the software system, and designing a database structure. Next were the iteration phases. The first iteration phase consisted mostly of familiarizing with some new tools and creating an outline of the product with minimal functionality. The second phase consisted of creating the first minimum viable product which included all of the functionality prescribed by the basic requirements. The third iteration was used for making the final adjustments and fixes so that the prototype could be tested by the stakeholder. This phase also contained the addition of placeholder pages to outline the envisioned end product. In the next sections, each of the phases is described in detail.

PRELIMINARY PHASE

PRELIMINARY DATABASE DESIGN

The initial idea, which can be found in Chapter 4 - Specification, was to create a product consisting of sequential process steps. The planner could create a new product by adding process steps to a list and specify the time to completion for that process step. A flaw in that concept is that some

products are created by combining two materials. This means that two process steps can be performed simultaneously. The list structure disallows any synchronous process step configuration. Therefore, it was decided that this structure was insufficient in capturing the complexity of a product. In the ideation phase, the idea for an assembly graph builder was mentioned. The choice was made to merge this idea into the system and make an assembly graph builder that is used to create the complex assembly graph structure. An assembly graph is a directed graph in which each process step is a node, and each connection specifies which process steps are next in the sequence. In the database design, this structure is maintained by an abstract class node, which has a list of parents and a list of children. As an example, consider that two processes are executed simultaneously and the components are then combined in the next process step, as seen in Figure 13.

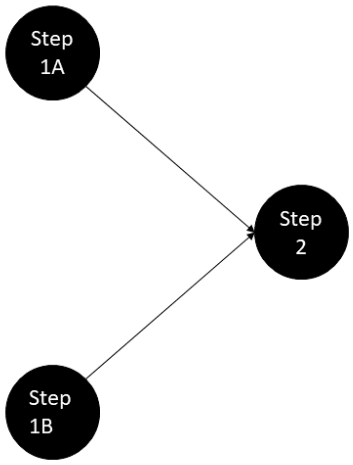


Figure 13: Example assembly graph

The steps 1A and 1B can be executed simultaneously and step 2 can only be done once step 1A and 1B are both finished. In the database, this structure would be saved as follows: step 1A has an empty list of parents and a list of children with exactly one element: step 2. The same goes for 1B. Step 2 would have an empty list of children, and a list of two parents: 1A and 1B.

This parent-child data structure is very similar to the abstract data type *tree*. However, because a node can have multiple parents it is formally called a graph. There is no option for any cycle since no two operations are ever the same instance, i.e. they each have a specific operation that can not be repeated on the same component. This also means the graph is directed since an operation can not be reversed. The reason this parent-child data structure for the graph is chosen is that it allows for easy traversing of the graph. One can start at any of the nodes and recursively traverse all the parents and children to perform operations.

The product consists of a list of process steps, but as was described in Chapter 4 - Specification, the process step is an implementation of a process type. Therefore, the database should also

contain a process type. This process type can be executed by different machines and therefore should have a list of available machines. The preliminary database design is shown below in Figure 14.

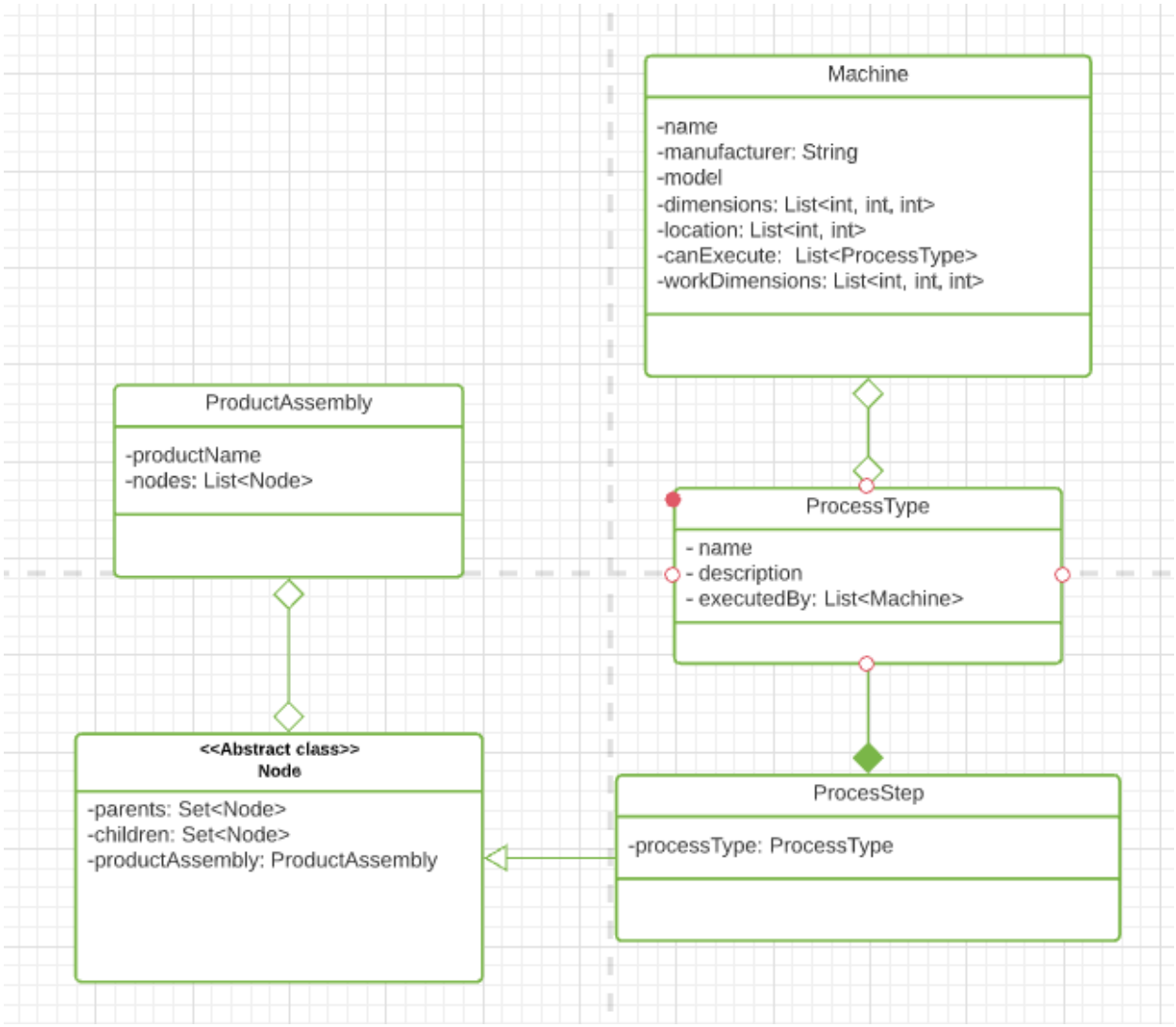


Figure 14: Preliminary database design

Notice that *Node* is an abstract class. The reason is that in the future, more kinds of nodes could are required, and by creating an inheritance like this makes the database design is more flexible. Notice also that most vector-like attributes in *Machine* are lists. The reason that the machine dimensions, work dimensions, and location are lists instead of vectors is that Java has little built-

in support for mathematical vectors especially when it comes to serialization and deserialization[1]. The green borders on the classes indicate the classes are entities, which means that are persisted in the database.

PRELIMINARY TOOLING DECISIONS

WEB-BASED APPLICATION

The decision was made to make the software system web-based for a few reasons. The first is that a web-based application is easily accessible on any device. This is useful because the planner might start planning in his office, but he could also want to check on the machines in which case he has to access the system from a mobile device. The second reason is that there is a lot of tools available for web-based technology [23]. The use of these frameworks facilitates rapid application development since a lot of the boiler plate code and common design patterns are provided.

FRONTEND FRAMEWORKS

After deciding for a web application the next choice is the frontend server and framework. A logical choice is to use a *Node.js* server since it is abundantly used by many big companies and across the internet as a whole [24]. Next to this, it was decided to use *React.js*[25] as the frontend framework. The reasons for this are similar to those of the decision to make a web application.

The next framework that is used in the frontend is a *React.js*-based framework called *Material UI* [26]. This framework makes designing an appealing user interface (UI) a lot easier because there are a lot of standard components, like menu bars, icons, and tables which can easily be used.

[1] Serialization is when a class object is translated to some string format, such as json, and deserialiation is the conversion of a string to a class object.

The last frontend framework that is needed is one that can draw diagrams. The main benefit of the software system is that the dependencies between the machines, process types, process steps, and the product it comprises are maintained and therefore a flexible tool is needed to draw such diagrams. There were only two available tools built specifically for *React.js* so the choice was very limited. The choice was made to use the *Beautiful-react-diagrams*[27] library because the tool seemed easier to use and required less time to learn, which is beneficial for rapid prototyping.

BACKEND FRAMEWORK

Another decision based on the abundance of support, information, and experience is the choice of using *Java* as the programming language and *Spring* [28] as the backend framework. The *Spring* framework has a tool for writing a lot of boiler plate code which allows for an almost immediate start after downloading it. Additionally, the framework also contains useful tools such as *Hibernate* [29] which make persisting data simpler. Another notable dependency is the bundle *spring-boot-starter-web*. This bundle contains useful libraries such as *Jackson-databind* used for class serialization and deserialization and *spring-boot-starter-tomcat* which provides the embedded tomcat server the application runs on. The database that was chosen for this project is *PostgreSQL* [30]. An overview of all the frameworks with the reason for their choice can be seen in Table 1.

	Framework/tool	Reason
Frontend	Node.js [24]	Availability of tools, support, and information
	React.js [25]	Ease of use, prior knowledge, availability of tools, support, and information
	Material UI [26]	Simple web-based UI tool
	Beautiful-react-diagrams [27]	Gentle learning curve
Backend	Spring [28]	Ease of use, access to useful tools
	Hibernate [29]	Easy persistence of data
	PostgreSQL [30]	Prior knowledge

Table 1: Overview of the used framework

DEMO AND FEEDBACK

The preliminary phase was concluded with a demo of the diagram tool and a feedback session on the design choices. The stakeholders agreed that an assembly graph structure would be better suited to represent the complex sequences of process steps required for a product compared to a list structure. Next, the stakeholders were shown the tool which was chosen to create the assembly graph diagram with, and the stakeholders were asked what they would like to see in the different visual representation of the nodes. They suggested a few ideas such as displaying the attack time (time to completion) in the nodes, displaying how many parts have gone through a machine, the usage time, the most popular processes, the change time[2]. They also suggested that it would be useful if the constraints for the input of the machines were also considered. For example, if a process step can be executed on two different machines, but one machine can only handle materials with a maximum thickness of 5 mm, then this machine should not be an option to the planner when they put a material in with a greater thickness. In that case, the machine that could not process said material would not appear in the list of available machines for that process step. Finally, the stakeholders reiterated that the idea of simulation would be extremely useful.

After this feedback session the following requirements were deemed viable and important and were therefore added to the list of 'must' requirements:

- Ability to specify a time to completion to each process step
- Ability to alter the list of available machines for a process step based on the input constraints of the machines

[2] The time for a machine to prepare for the next process i.e., changing of a blade

ITERATION PHASE 1

PROGRESS

This was the first implementation phase. A large part of this phase was learning the new tools and trying to persist the complex structure of the nodes using hibernate. Once the database design was translated into java classes and could be persisted as entities in the database, the frontend main structure was built. The frontend layout consists of three pages: the product diagram page, the process type page, and the machine page. These pages can be seen in Figure 15, Figure 16, and Figure 17. Placeholders were inserted in both the process type and machine page as an example. The product page was only able to fetch data from the backend. The way it was implemented was by manually storing an assembly graph with a few nodes in the backend which were fetched by the frontend. In the frontend, the assembly graph structure was converted to the JSON object required by the *beautiful-react-diagram* library. The JSON object is used by the library to render all the nodes and the links in the diagram.

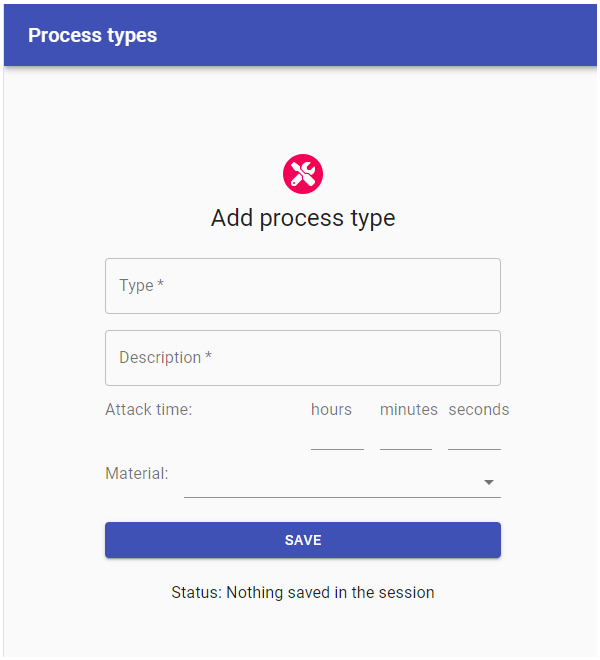


Figure 16: Process type page (phase 1)

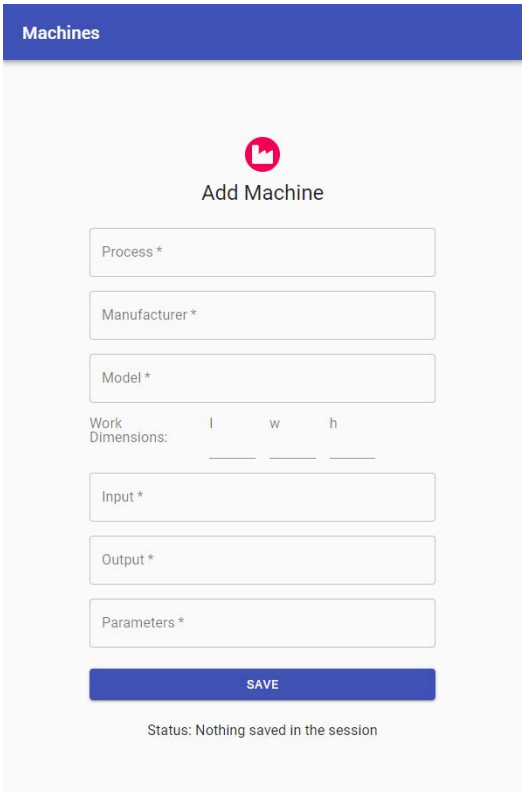


Figure 15: Machine page (phase 1)

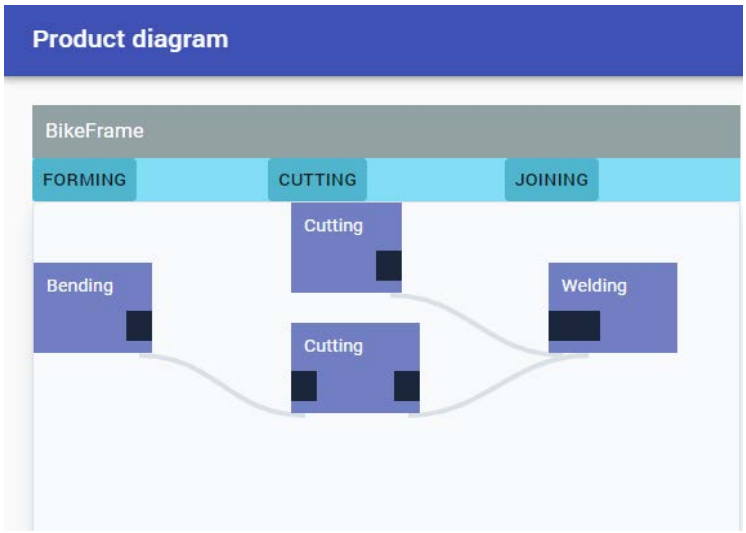


Figure 17: Product page (phase 1)

DESIGN DECISIONS

To justify the decision to convert the data in the frontend a few things need to be explained about the *beautiful-react-diagram* library and the data structure in the backend.

As explained above, in the database a product has an assembly graph structure that consists of a list of nodes, and each of these nodes has a list of parents and children. The data structure that is used by the diagram library consists of a *Schema* JSON object which contains two lists: one with the *schema nodes* (which each have a list of in- and output ports) and a list of schema links consisting of all the in- and outputs that are connected to each other. So, as an example, consider two *schema nodes A* and *B*. *A* has an output list consisting of one port called *Out-A* and *B* has a list of input consisting of only one port: *In-B*. If these *schema nodes* are connected, then there would be one element in the list of links which is an object with input *In-B* and output *Out-A*. Both the list of links and the list of *schema nodes* are part of one *Schema* object. This is very similar to the mathematical definition of a graph since the lists in the *Schema* objects are essentially the set of vertices and the set of edges.

The *schema nodes* in the *Schema* object have some extra notable properties. As said before, they have a list for inputs and outputs, each *schema node* also has a list with two elements that specify their coordinates on the diagram, they have a *data* object and a *render* object. The *render* object contains a function reference which is called by the library. The *data* object is a data transfer object (DTO) that passes variables to the *render* function.

The decision to convert the node structure in the frontend was based on two reasons. The first is reason is the fact that the serialization in the backend is easier since it does not require any additional DTO: the classes are serialized as is. The second reason is that the *schema nodes* in the *Schema* contain function references. Since the references are to *JavaScript* functions it is not possible to transfer them over an HTTP connection. Also, the functions references cannot be converted to *Java* objects, nor can they be stored in the database.

FEEDBACK

The stakeholders were happy with the progress and the direction the product was going. They had no new requirements. They restated the fact that process types can be executed by different machines and therefore the input constraints of the machines are an important factor in the planning of a product. They want to be able to specify the input of a machine and have some validation as to whether this machine can handle this type of input based on at least the material and the dimensions. They also mentioned their desire for a grid view of all the machines on the shopfloor, which was already mentioned in Chapter 4 - Specification and therefore put on a higher priority in the planning of the project. Another remark they had was related to the altering of machines. They mentioned that when using machines on a shopfloor that they are not as static as might be expected. There are a lot of add-ons and different tools for the machines which could be added or removed, so being able to easily change this is very useful. This strengthens the choice of creating a separate page for machines, even though they change relatively infrequently.

After this meeting, the requirements remained the same, however, the priority of the following requirement was increased:

- Ability to get a graphical representation of the machines

ITERATION PHASE 2

PROGRESS

SYSTEM DESIGN

This was the longest phase consisting of two weeks. At the beginning of this phase, the conversion of the assembly graph to a JSON *Schema* was still done in the frontend. Relatively soon the decision was made to redesign the backend. Because of the frontend conversion, the system was not designed using the model-view-controller (MVC) pattern. This was changed so the database structure and system design would adhere better to the MVC pattern. The backend was refactored, and the conversion layer was moved from the frontend to the backend. The conceptual differences in system design between phase 1 and phase 2 can be seen in Figure 18 and Figure 19.

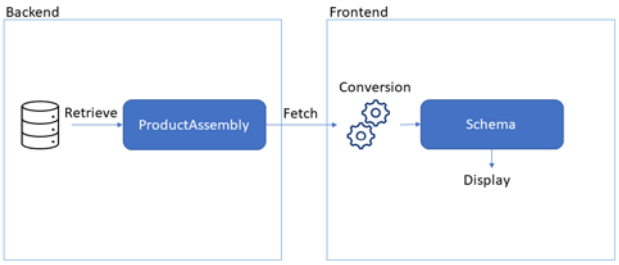


Figure 19: Conceptual system design phase 1

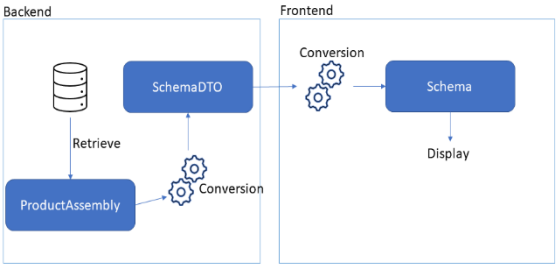


Figure 19: Conceptual system design phase 2

The black text near the arrows indicates a programming process and the blue rounded boxes indicate the datatype. Notice that in the new design in Figure 19 there is also a new datatype: *SchemaDTO*. This type contains the id of the assembly graph, the product name, and a *Schema* object. Because this *SchemaDTO* is part of the backend the *Schema* object is the *Java* equivalent of the JSON *Schema* object. Many of the parameters of the *Java Schema* and the JSON *Schema* can be mapped one to one, except for the function reference. A solution to this was to store the

function name as a string in the *Java Schema* and look up the corresponding function after it was fetched and converted in the frontend. This is why there is still some conversion needed in the frontend.

INTERFACE PAGES

During this phase, the pages for machines and process types were implemented. They both contain a list of each respective element and the functionality to add, edit and remove elements. The machine list can be seen in Figure 20. The edit page is very similar to the dummy page with

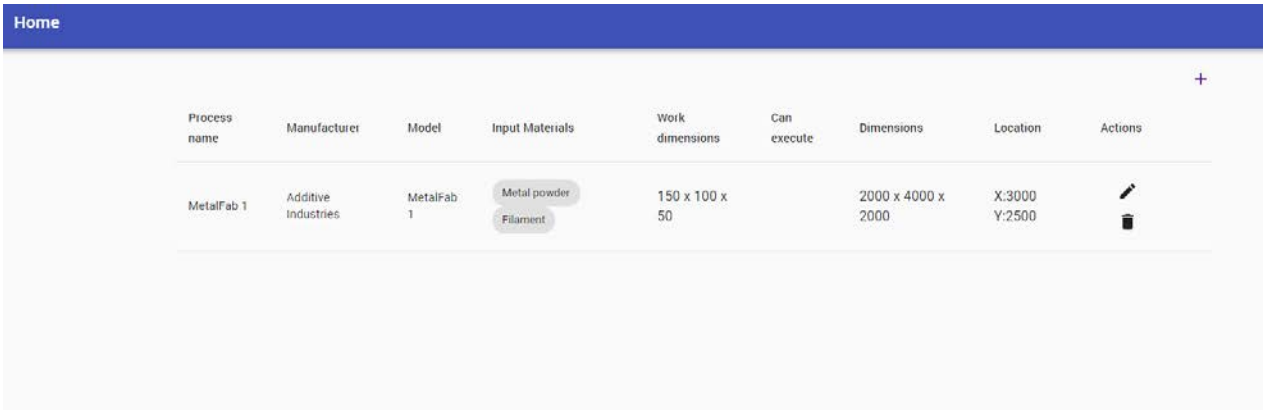


Figure 20: Machine list page (phase 2)

some slight changes in variables. Notable is the feature to add materials to the database. This can be seen in Figure 21.

The product diagram has been changed as well. Firstly, all the products are displayed in a list similar to the process types and the machines: the products can be edited, deleted, and added. Secondly, the layout and rendering of the nodes have been changed. There now is a list of machines that can execute the process type. With this, there is also the option to specify a time to completion for each of the machines. An example can be seen in Figure 22.

Also added this phase were the component nodes. The stakeholders have said from the beginning that input constraints for the machines, such as material or dimensions, are useful to consider when planning. Not all machines can use every material and not all of them have the same work dimensions. Therefore the components were added as a different type of node. These components each have dimensions and a material. Before each process step node, a material node should be added. This material node needs a name, a material, and dimensions before it can be placed on the diagram. An example of adding a component node can be seen in Figure 23.

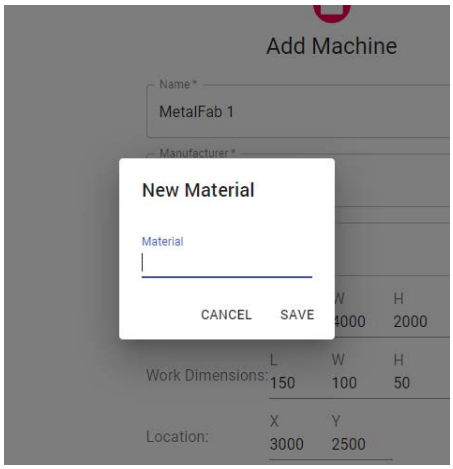


Figure 23: Adding materials to the database (phase 2)

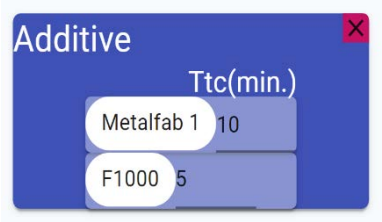


Figure 23: Process step node (phase 2)

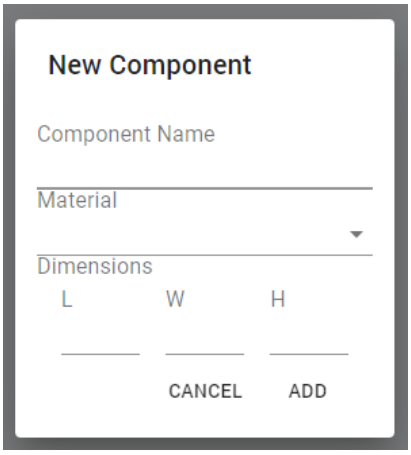


Figure 23: Adding component node (phase 2)

With the new layout of the process step node and the addition of the material node, the product diagram has been changed to what can be seen in Figure 24.

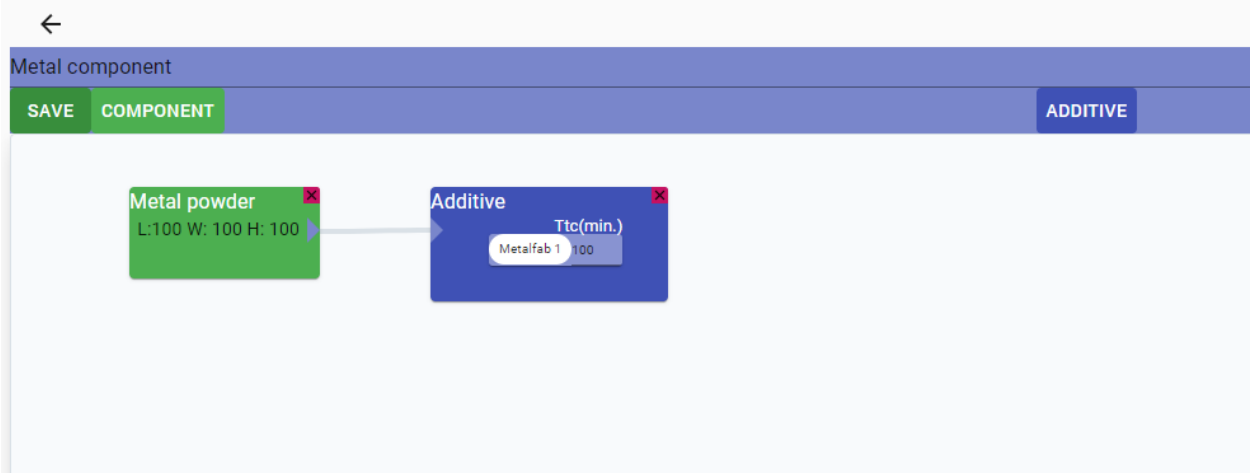


Figure 24: Product diagram (phase 2)

Note that at this point, the constraints for the machines are not taken into account yet. The list of machines in the proceeding process step does not change if the component node violates any of the input constraints of any of the machines.

DESIGN DECISIONS

SYSTEM DESIGN

The decision to move the conversion from the front- to the backend is based on two reasons. One is the fact that the *beautiful-react-diagram* library did not allow for altering the *Schema* object directly. The only option is to create a JSON object beforehand and initiate the diagram with this. This means that every time any changes are made, the entire *Schema* needs to be updated, converted to the node structure of the backend, sent back and be validated in the backend, sent back to the frontend, reconverted back to a *Schema,* and then be rendered again. This is very inefficient. Also, one of the biggest downsides of this frontend handling was that the conversions, such as traversing the graph recursively, had to be done in the frontend, which made it more prone to errors partly due to the fact *JavaScript* is a weakly typed language.

The second reason the system design was changed was due to the lack of scalability and adaptability of the architecture. It did not adhere to the MVC pattern, and this pattern is known to be very scalable and adaptable. The reason the previous design was not adhering to the MVC pattern is that the view and controller (the frontend) contained business logic (i.e., conversion of data). The problem with this is that now the view and the controller are very hard to separate. When another iteration of the product is made where a new view is chosen, both the view and controller have to be rewritten. When the business logic is done in the model, the view and the controller are easily interchangeable. An important remark on this is that even though this change does make the system more in line with the MVC pattern, the ideal system should have the same data type in the frontend and the backend. Due to the choice of the frontend diagram library and the choice of the parent-child data structure, there will always be a need for some undesirable conversion.

DATABASE REDESIGN AND CLASS DIAGRAM

The database and class structure were redesigned, and the new design can be seen in Figure 25. (a higher resolution copy can be found in Appendix A.

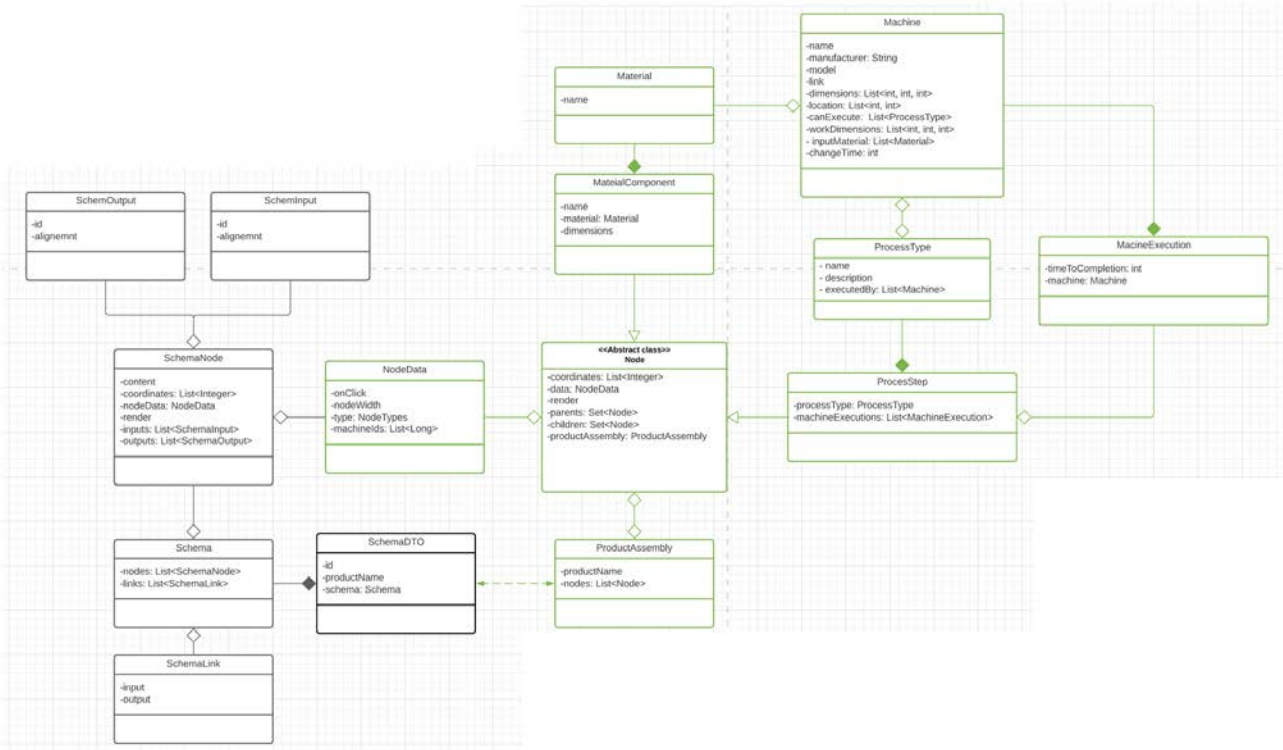


Figure 25: Database design (including classes) (phase 2)

Note that the green classes are entities that are persisted in the database and the black classes are part of the model but not persisted[3]. The green classes are converted into the black classes as part of the conversion in the backend.

[3] Strictly speaking this is a class diagram and not a database diagram. However, a database diagram and class diagram are very similar. Additionally, with *Hibernate* the separation between classes and the database becomes obscured. Therefore, representing the class and database structure in one diagram was deemed the logical choice for clarity.

A few notable changes to the persisted entities are the addition of the classes *Material*, *MaterialComponent*, *NodeData, MachineExecution,* and the addition of the *render* and *data* attribute in the *Node* class. First of all, *Material* is added to the database. This material can be attached to a machine as was mentioned before. Second of all, the *MaterialComponent* class. This class extends the *Node* class, so it has the same properties as the *ProcessStep* class. This class is used to represent the material nodes in the frontend with dimensions and a material.

There was an important choice to be made as to whether a material component should be a separate node in the diagram or not. The alternative was to incorporate a material component into the process step node since every process step has a material component as output. This would mean fewer nodes and reduce clutter in the diagram when the products become more complex. However, the stakeholders were asked during this phase if a process step always has only one output, and the answer was no. One process step can create two different material components. Therefore, the choice was made to make material a separate node, to maintain flexibility and allow multiple outputs for one node.

Thirdly, the attributes *render* and *data* are added to the *Node* class. The reason for this is that the *SchemaNode* class also has a *render* and *data* attribute. These attributes are used in the frontend for rendering the node and passing variables to the render function. These attributes would be lost if they were not persisted when the *Schema* was converted to the assembly graph structure and therefore the choice was made to persist *render* and *date* in the *Node* class. Strictly speaking, this violates some of the principles of the MVC pattern, since the node structure model does not need these attributes; they are only needed in the frontend. The alternative is to persist an additional entity, which stores this extra information and also stores which node the data is a part of. The choice for storing everything in the *Node* class was based on the reduced complexity of the conversion.

Lastly, there is the addition of the *MachineExecution*. The reason this was added is that there needs to be a separation between process type and time to completion. A process type does not have a time to completion itself since every machine can do the same process in different amounts

of time. Therefore, the relation is: a process type has a list of machine executions, and each machine execution has its own machine and time to completion.

FEEDBACK

The stakeholders were pleased with how the product looked. They thought it looked very clean and modern. They were excited to see the addition of material nodes since this was an added requirement in the preliminary phase. However, they did stipulate the fact that including and validating the machine constraints was a core feature of the envisioned product and they would very much like to see this implemented for the next iteration.

They also introduced some new requirements. The first was the addition of standard sequences. Certain process steps always proceed each other. For example, additive manufacturing is always proceeded by part removal and cleaning. The second requirement was to add efficiency in terms of cost or time for each path the product could take. The last requirement was to be able to manually select a path to see how long it would take.

Due to the short period of this project and the complexity of implementing such features these requirements are not included in the list of requirements.

ITERATION PHASE 3

PROGRESS

PROTOTYPE

This was the last phase of the realization. The biggest addition this phase was the working machine constraints. In the last phase, there were machine execution visible in the process step nodes, but these machines were not removed from the list if the material that was input did not satisfy the machine constraints. Now, only the machines which can process the material that is put into the process step will appear. The machines are removed from the list after the diagram

is saved. See Figure 26 and Figure 27. This was the last functionality that was added to the prototype.

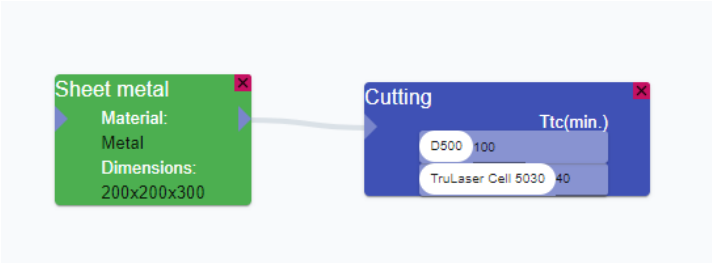


Figure 27: Machine list before saving

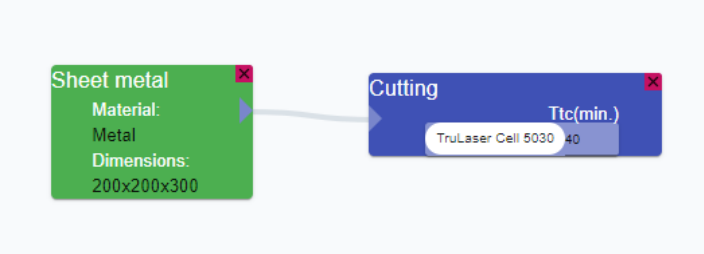


Figure 26:Machine list after saving

ENVISIONED END PRODUCT

This next section is about the features that are not part of the prototype, but the envisioned end product. They were implemented as simple placeholder images in the prototype to illustrate what the envisioned end product could look like. The placeholders are the requirements that could not be completed in the timeframe of this thesis.

The ability to create an overview of the machines on the shopfloor is an important 'should' requirement. The first page of the envisioned end product is called *Machine overview*. The idea is that with the data that is input in the machine page such as dimensions and location on the shopfloor, a map can be generated that displays all the machines at their correct places. This map can then be used with different overlays to track the status of the machines, see the product pathing, and generate statistics about the shopfloor. The overlays that were included as placeholders were the same as the solutions *3a-c* seen in Figure 5, Figure 6, and Figure 7. One important difference between solution *3b* (which is the machine occupancy percentage) and the page that is part of the envisioned product is that in the initial idea the occupancy was calculated

using the data that is collected throughout the day, but much easier would be to create these statistics predictively. This would require no integration with the machines. The way it would work is that the time to completion for each machine in each process step is added together and an estimate of the total duty time can be calculated for some time frame. If a process step has two or more options, however, this would not work since the AGVs make stochastic choices between the machines. A solution for this problem is that if a process step is executable by multiple machines, the time for completion is divided for each machine by the total amount of machines, resulting in the average duty time. As an example: if process *A* can be executed by machines *1* and *2* and take 8 and 10 minutes respectively, then the average time on each machine is 4 and 5 minutes, respectively.

The ability to generate charts is another 'should' requirement. Two charts were included in the prototype: Gantt and timeline. The Gantt chart can be seen in Figure 28 and Figure 29. The buttons above the Gantt chart are to select which paths to display.

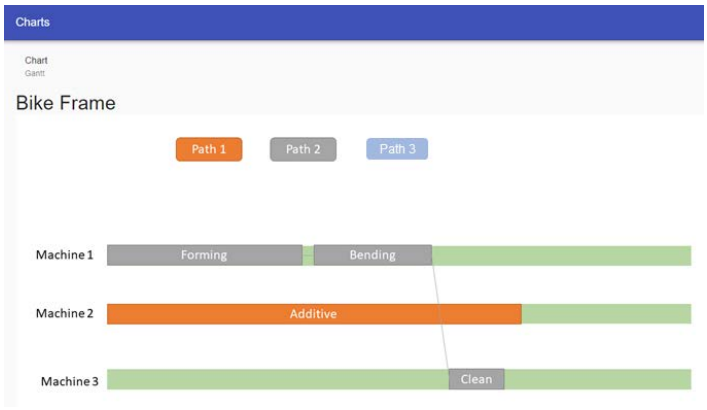


Figure 28: Gantt chart with 2 selected paths

The second chart that was added is a timeline. An example can be seen in Figure 30. Each product path is a separate timeline and after each timeline, the total time and cost are displayed. Also, since the pathing is stochastic, the average cost and time are useful data and are therefore displayed below the time and cost column. From this diagram, the average machine usage can also be calculated for a given timeframe.

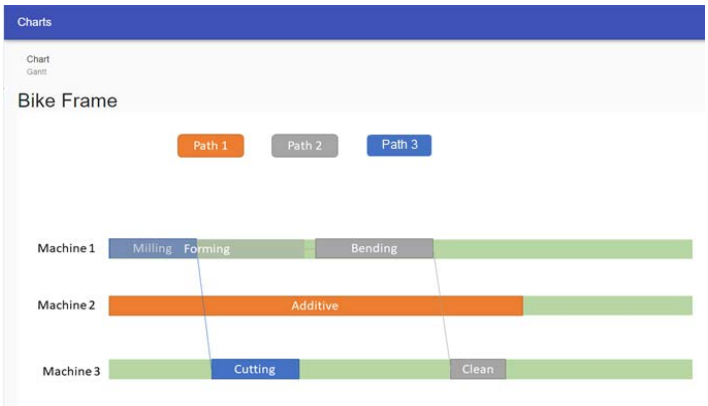


Figure 29: Gantt chart with 3 selected paths

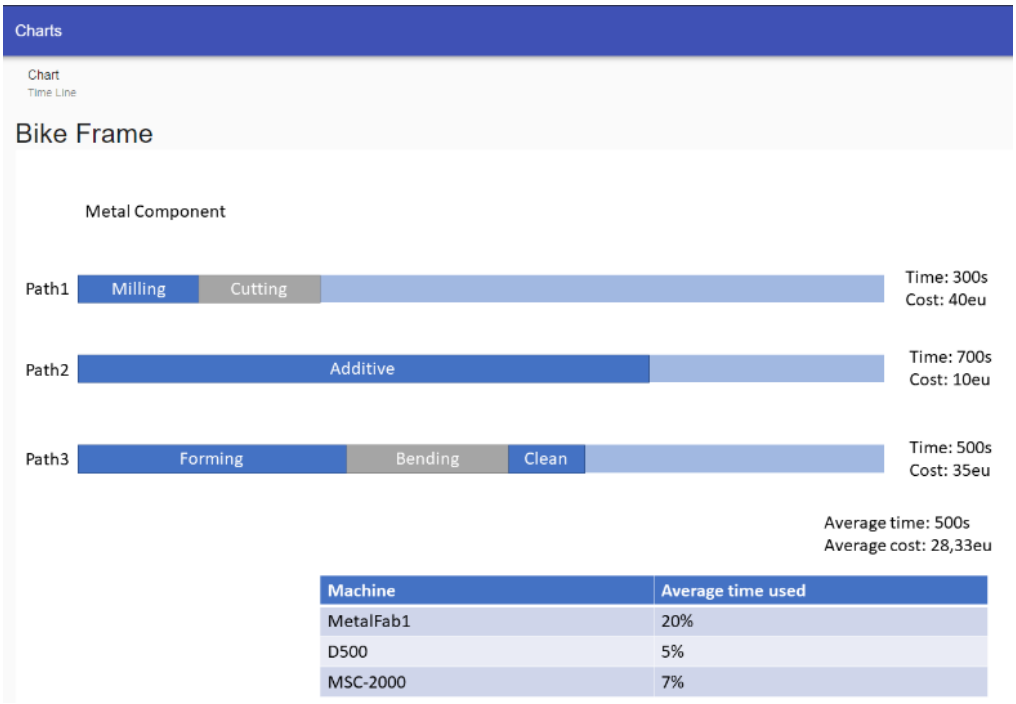


Figure 30: Timeline chart

Lastly, the file distribution placeholder was added. The files that are required for each of the machines need to be uploaded with their respective process step. Therefore, the option to upload an instruction file could be added to the product diagram page. An example can be seen in Figure 31.

This chapter will not have a feedback section. Instead, the feedback from the stakeholders after this phase is evaluated in the next chapter: Chapter 6 – Evaluation.

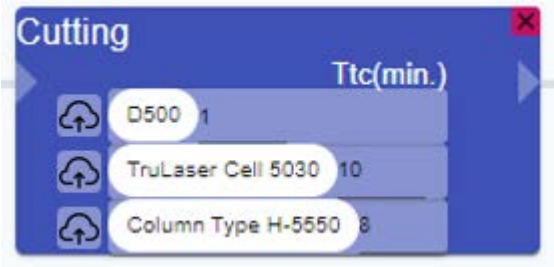


Figure 31: Upload option in diagram

CHAPTER 6 – EVALUATION

In this chapter, the system is evaluated. This is done by firstly looking at the requirements from Chapter 4 - Specification and evaluate which requirements were and were not met and why. This will be done for both the prototype and the envisioned end product since not all requirements could be implemented in the prototype. The envisioned product has some placeholder pages and images to illustrate what it may look like, so if one of these pages meets a requirement it will be marked as met. After evaluating the requirements, the results from the feedback sessions of the last iteration, as well as the results from the user tests, will be evaluated.

REQUIREMENTS EVALUATION

EVALUATING THE **MUST** REQUIREMENTS

- Ability to create a production flow for a product using process steps
- Ability to add process types
- Ability to add machines
- Ability to assign process types to machines
- Ability to get an overview of all the machines
- Ability to get an overview of the products

Added during realization:

- Ability to specify a time to completion to each process step
- Ability to alter the list of available machines for a process step based on the input constraints of the machines

All of these requirements were met by the prototype. There are pages to input machines, process types and to create a diagram. The machines can be attached to process types, and these are displayed within a process step. Additionally, the time to completion can be specified with the machines in the process step node and the list of machines is updated based on the material that is given as its input.

EVALUATING THE SHOULD REQUIREMENTS

- Ability to get a graphical representation of the machines
- Ability to get a graphical representation of the products with their flows
- Ability to generate a chart of the machines with their process steps over time

The first requirement is met by the envisioned product. The data that is required to get a graphical representation of a machine, like location and dimensions, are added to the machine input page. For now, the data is not actually used, and a simple placeholder image is put into the envisioned product. The second requirement is also met, but it has to be stipulated that it is a vague requirement. The graphical representation is the same as the product diagram, so this requirement is very similar to the first 'must' requirement. The last requirement is met through the placeholders of the Gantt chart.

EVALUATING THE COULD REQUIREMENTS

- Ability to integrate with the machines
- Ability to generate statistics of the shopfloor
- Ability to add cost per process
- Ability to generate an overview that could be displayed on a big screen

The requirement concerning the integration with the machines is somewhat met with the machine overview placeholder page, but it has to be stated that this is very minimal. The actual integration and displaying meaningful information are very complex. Generating statistics is possible in the form of the Gantt and timeline charts. The last two requirements concerning adding costs and creating an overview are not met.

USER EVALUATION

The user evaluation was done using a small questionnaire that covered the features of the system. The user evaluation will be split into feedback on the prototype and feedback on the envisioned product.

PROTOTYPE

Firstly, the machine pages. The list was liked, especially with the materials displayed as chips for each machine. The column order was a point of critique since the first column displayed the process name, but it was suggested to be more intuitive to have the machine name first. The process types were not linked to this page, and this was something that was suggested by one of the users as well. Furthermore, for the parameters of the machines, there was a suggestion to add department, the software version, and the main responsible person for this machine.

Secondly, the process type page. There was little feedback on this page, other than the addition of the parameter: person able to perform the process type.

Lastly, the feedback for the product diagram. Creating the product using an assembly graph like this was received positively because it required a different way of thinking about the process. One user did find it hard to create a product this way because starting from the end product it can be hard to work back towards single components. Additionally, the user rightfully asked the question of how to dimensionalize water or electricity. Other feedback was the lack of units in the time to completion field as well as a suggestion to specify the unit. Some suggestions were to have a scrollable window when dragging a connection to the edge of the screen, the addition of optional paths, and the total time it takes to produce a product.

ENVISIONED PRODUCT

The users gave feedback on each of the placeholder pages of the envisioned product. The first page was the machine overview page. The usefulness of this page was disputed because the user interface was unintuitive. It was unclear to some users that there was an option to change the overlay of the overview. The overview of the machines on the shopfloor without any additional information was deemed not useful, but if the user found the overlays they were perceived as useful. Additional information for the machine status overlay was suggested such as overall equipment effectiveness (OEE), capacity per hour, and energy usage.

The feedback on the chart pages was that cost is an important variable which perhaps should get a separate page. Charts that could display costs such as the price of producing on a certain machine and the average price of materials were also suggested.

Lastly, the feedback on the control page was relatively negative. The idea that a production process can be started with one press of a button is perceived as oversimplified. Making this work in the first place would require a lot of difficult integration with the machines in addition to being somewhat risky because the planner could bypass the operator inadvertently. The progress timelines on the page were deemed useful if operational.

CONCLUSION OF EVALUATION

To conclude, all of the 'must' requirements, including the ones that were added during realization, are part of the core functionality of the system. These requirements were all met with the prototype. The users were pleased with this functionality but had some suggestions for adding details such as unit specification, the addition of cost as a parameter, and optional product pathing. These are details that should be included in any future development of the system.

The 'should' requirement of getting an overview of the product is also met with the prototype, but this was not a good requirement to begin with as it was vague. The other 'should' requirements were implemented in the envisioned product. The users liked the charts but

suggested more focus on costs since this is an important variable. The 'could' requirements were not all covered with the envisioned product. The addition of costs was one of these requirements and looking at the feedback this is something that requires some focus. The integration with the machines is, as was said before, something that is a good requirement for MES but is very difficult to implement.

CONCLUSION

The goal of this thesis was to develop a new kind of MES that is specifically useful for matrix production. The focus of this particular matrix production system is to serve as a demonstrative example. Therefore, flexibility becomes even more important, and the MES has to reflect this too. The main requirements of the new MES are that it helps the planner organize, initiate, and track the production process. Due to the limited scope of a bachelor thesis, this goal was split up into the creation of a functional prototype and an envisioned end product.

Different ideas were proposed in the ideation phase. The idea of creating an assembly graph to represent a product and its processes was the basis of the other features. This structure would be the basis of the envisioned end system. This envisioned system would have the option to generate statistics, create overviews of product pathing, and distribute files. In the specification phase, this core idea was put into requirements and two more requirements were added during realization. These were the addition of time to completion for each machine in a process step and the addition of functional machine constraints and became part of the prototype. The database structure was changed during realization too since the initial structure was not compliant with the MVC pattern and the tools used required very specific usage.

The first goal of this thesis was to create a software system that helps the planner organize and plan products. All the 'must' requirements were met with the prototype. The requirements describe a system in which a planner can manage their machines, process type, and products and this was essentially the first goal of this thesis. The users that tested the product agreed that the system would be useful if it were part of a functional MES as it helps with organizing and planning.

However, there were some remarks about the difficulty of creating a product and the problem of displaying persistent input. Since the requirements were met and overall, the users were positive towards the prototype the conclusion is that the first goal of this thesis was met.

The second goal was to outline what the eventual product could look like. For this, the ideas of the ideation phase were converted into placeholder web pages and feedback on these ideas was solicited. The other requirements were in line with the ideas that were chosen in the ideation phase. Not all of the requirements were met, but the generation of charts, which were both Gantt and timeline as well as the creation of an overview with product-specific pathing were. These can also be considered to be more important requirements since they were repeated by the stakeholders in more than one feedback session. The second goal of this thesis is therefore met. However, it must be stated that this goal was very open-ended, and deciding that has been reached is somewhat subjective. The goal was also not reached with a prototype, but with simple images. Thought has been put into how these images would be a logical result of the prototype, but when they are developed there will undoubtedly be new problems.

To conclude, the prototype and envisioned system that were developed in this thesis serve as a good example of what a new MES might look like. The new way of thinking about processes, process types, and materials allows for the creation of a complex structure to represent a product. This new philosophy fell in well with the users and stakeholders and allowed them to rethink their process. Although there are things that can be improved such as the clutter of the diagrams, the prototype did meet the requirements. The envisioned end product displayed the most important features the stakeholders would like to see and would therefore both the prototype and the envisioned end product would be a good starting point for future work.

FUTURE WORK

The first recommendation is to choose the right diagram tool or create a new one. The tool that was used in this thesis caused more problems than it solved. For example, it required a big change in the database to store the required data. Ideally, one would want a frontend tool that matches

the data types and data structures of the backend. The conversion that was needed for this prototype was a difficult task.

If the choice for another tool is not possible, and there still needs to be additional data stored because the front- and backend data types do not match, I would suggest using an additional database table. This table would contain all the data that is required by the frontend, but not the backend, and vice-versa. Changing the model to adhere to the view data structure is bad practice and should therefore be avoided in the future.

The second recommendation is that the stakeholders did seem very keen on a live overview of the shopfloor with a useful overview. Particularly the product pathing was a liked feature, as well as the creation of different charts and statistics such as the machine occupancy.

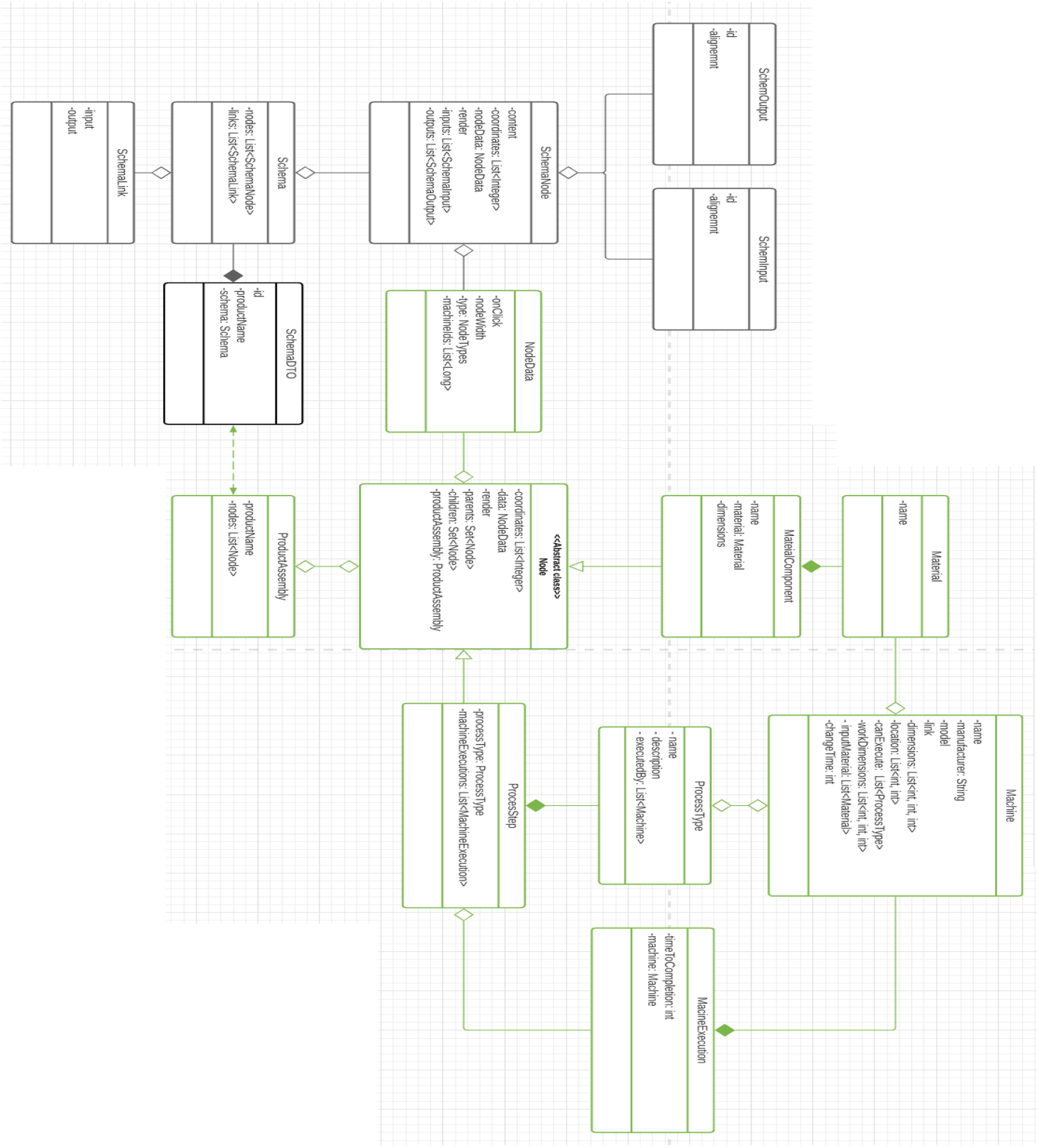
The third recommendation is to add cost to the system. This was another parameter that came back during the feedback sessions as well as the evaluation session. Cost is an important variable and any tool that can predict and reduce it will be well received. Adding cost would not require much work since it would simply be associated with a machine, in the same way, the time to completion is. In addition to this recommendation, there is another feature that the stakeholders requested and that was the addition of average cost and time to completion for one product.

The fourth recommendation is to rethink the diagram. One reason the product diagram looked the way it did was because of the tool that was used. Two recommendations are to add the option to create a component by combining several nodes. It could be something like a box that contains several process steps which are required to create that component. During the feedback sessions and the evaluation, there was a suggestion to create optional paths. These would be slightly different from the pathing choice between machines as it would be the choice between different series of steps. An example of an optional path is the following: when we consider some component that can be made by manipulating some basic material, like metal this would require several process steps. This would be one optional path. The second path could be only one process step which is an additive process. The end result might be the same, but the time and cost might differ. A slight misconception made in this thesis was that there was only one choice

and that was between the machines. In reality, there are more choices and one of them is the choice between two sets of process steps that produce the same component.

The fifth recommendation is to rethink the persistent input. In the prototype, all the input is grouped. This is both persistent input, like water, air, and oil, in addition to material input, like metal, wood, or plastic. These are actually two different types of input and should be handled in this way. One problem that was highlighted in the evaluation was the fact that some of the inputs that were specified for a machine could not be added properly with the product diagram page as it is. Each material would require its own node, but this is illogical. The material node in the product diagram is actually a component node, and the persistent input should not be a separate node. It might be an added piece of information for something like a hover function, but the first question that has to be asked before changing this is: should persistent input be in the product diagram in the first place?

APPENDIX A


SchemOutput
-id
-alignemnt
SchemInput
-id
-alignemnt
SchemaNode
-content
-coordinates: List<Integer>
-nodeData: NodeData
-render
-inputs: List<SchemaInput>
-outputs: List<SchemaOutput>
Schema
-nodes: List<SchemaNode>
-links: List<SchemaLink>
SchemaLink
-input
-output
SchemaDTO
-id
-productName
-schema: Schema
NodeData
-onClick
-nodeWidth
-type: NodeTypes
-machineIds: List<Long>
<<Abstract class>>
Node
-coordinates: List<Integer>
-data: NodeData
-render
-parents: Set<Node>
-children: Set<Node>
-productAssembly: ProductAssembly
ProductAssembly
-productName
-nodes: List<Node>
MaterialComponent
-name
-material: Material
-dimensions
Material
-name
Machine
-name
-manufacturer: String
-model
-link
-dimensions: List<int, int, int>
-location: List<int, int>
-canExecute: List<ProcessType>
-workDimensions: List<int, int, int>
- inputMaterial: List<Material>
-changeTime: int
ProcessType
- name
- description
- executedBy: List<Machine>
ProcessStep
-processType: ProcessType
-machineExecutions: List<MachineExecution>
MacineExecution
-timeToCompletion: int
-machine: Machine

REFERENCES

[1] Y. Koren *et al.*, “Reconfigurable Manufacturing Systems,” *CIRP Annals*, vol. 48, no. 2, pp. 527–540, Jan. 1999, doi: 10.1016/S0007-8506(07)63232-6.

[2] Y. Koren, X. Gu, and W. Guo, “Reconfigurable manufacturing systems: Principles, design, and future trends,” *Front. Mech. Eng.*, vol. 13, no. 2, pp. 121–136, Jun. 2018, doi: 10.1007/s11465-018-0483-0.

[3] Y. Cohen, M. Faccio, F. G. Galizia, C. Mora, and F. Pilati, “Assembly system configuration through Industry 4.0 principles: the expected change in the actual paradigms,” *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 14958–14963, Jul. 2017, doi: 10.1016/j.ifacol.2017.08.2550.

[4] P. Greschke, M. Schönemann, S. Thiede, and C. Herrmann, “Matrix Structures for High Volumes and Flexibility in Production Systems,” *Procedia CIRP*, vol. 17, pp. 160–165, Jan. 2014, doi: 10.1016/j.procir.2014.02.040.

[5] S. Wang, J. Wan, D. Zhang, D. Li, and C. Zhang, “Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination,” *Computer Networks*, vol. 101, pp. 158–168, Jun. 2016, doi: 10.1016/j.comnet.2015.12.017.

[6] M. C. May, S. Schmidt, A. Kuhnle, N. Stricker, and G. Lanza, “Product Generation Module: Automated Production Planning for optimized workload and increased efficiency in Matrix Production Systems,” *Procedia CIRP*, vol. 96, pp. 45–50, 2021, doi: 10.1016/j.procir.2021.01.050.

[7] M. Schönemann, C. Herrmann, P. Greschke, and S. Thiede, “Simulation of matrix-structured manufacturing systems,” *Journal of Manufacturing Systems*, vol. 37, pp. 104–112, Oct. 2015, doi: 10.1016/j.jmsy.2015.09.002.

[8] D. Sun *et al.*, “PlanningVis: A Visual Analytics Approach to Production Planning in Smart Factories,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 1, pp. 579–589, Jan. 2020, doi: 10.1109/TVCG.2019.2934275.

[9] M. Younus, C. Peiyong, L. Hu, and F. Yuqing, “MES development and significant applications in manufacturing -A review,” in *2010 2nd International Conference on Education Technology and Computer*, Jun. 2010, vol. 5, pp. V5-97-V5-101. doi: 10.1109/ICETC.2010.5530040.

[10] S. Jaskó, A. Skrop, T. Holczinger, T. Chován, and J. Abonyi, “Development of manufacturing execution systems in accordance with Industry 4.0 requirements: A review of standard- and ontology-

based methodologies and tools," *Computers in Industry*, vol. 123, p. 103300, Dec. 2020, doi: 10.1016/j.compind.2020.103300.

[11] W. de Paula Ferreira, F. Armellini, and L. A. De Santa-Eulalia, "Simulation in industry 4.0: A state-of-the-art review," *Computers & Industrial Engineering*, vol. 149, p. 106868, Nov. 2020, doi: 10.1016/j.cie.2020.106868.

[12] A. W. Colombo, R. Carelli, and B. Kuchen, "A temporised Petri net approach for design, modelling and analysis of flexible production systems," *Int J Adv Manuf Technol*, vol. 13, no. 3, pp. 214–226, Mar. 1997, doi: 10.1007/BF01305873.

[13] L. Aiping and X. Nan, "A robust scheduling for reconfigurable manufacturing system using petri nets and genetic algorithm," 2006, vol. 2, pp. 7302–7306. doi: 10.1109/WCICA.2006.1714504.

[14] N. XIE, "Modeling and analysis of reconfigurable manufacturing system by extended stochastic petri nets," *Chinese Journal of Mechanical Engineering - CHIN J MECH ENG*, vol. 42, Dec. 2006, doi: 10.3901/JME.2006.12.224.

[15] N. Xie and A. Li, "Modeling and analysis of reconfigurable manufacturing system by extended stochastic petri nets," *Jixie Gongcheng Xuebao/Chinese Journal of Mechanical Engineering*, vol. 42, no. 12, pp. 224–231, 2006, doi: 10.3901/JME.2006.12.224.

[16] S. Narayanan, D. A. Bodner, U. Sreekanth, T. Govindaraj, L. F. McGinnis, and C. M. Mitchel, "Research in object-oriented manufacturing simulations: an assessment of the state of the art," *IIE Transactions*, vol. 30, no. 9, pp. 795–810, Sep. 1998, doi: 10.1080/07408179808966526.

[17] H. S. Ismail, V. S. Tey, L. Wang, and J. Poolton, "A UML approach for the design of reconfigurable manufacturing simulation models," in *2011 IEEE International Conference on Industrial Engineering and Engineering Management*, Dec. 2011, pp. 1690–1694. doi: 10.1109/IEEM.2011.6118204.

[18] A. Li and Chao Lv, "Reconfigurable manufacturing system modeling method based on object-oriented technology," in *2009 International Conference on Mechatronics and Automation*, Aug. 2009, pp. 3420–3425. doi: 10.1109/ICMA.2009.5246353.

[19] G. Tavola, M. Taisch, and F. Boschi, "A standard approach to production systems modelling based on Finite State Automata," in *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, Jul. 2017, pp. 1117–1122. doi: 10.1109/INDIN.2017.8104930.

[20] J. Fraser *et al.*, "MES explained: a high level vision," *MESA White Paper*, no. 6, 1997, Accessed: Jun. 09, 2021. [Online]. Available: https://mesa.org/

[21] J. Barata, P. R. da Cunha, A. S. Gonnagar, and M. Mendes, “Product Traceability in Ceramic Industry 4.0: A Design Approach and Cloud-Based MES Prototype,” in *Advances in Information Systems Development*, vol. 26, N. Paspallis, M. Raspopoulos, C. Barry, M. Lang, H. Linger, and C. Schneider, Eds. Cham: Springer International Publishing, 2018, pp. 187–204. doi: 10.1007/978-3-319-74817-7_12.

[22] D. Clegg and R. Barker, *CASE Method Fast-track: A RAD Approach*. Addison-Wesley Publishing Company, 1994.

[23] “Single-page application,” *Wikipedia*. May 19, 2021. Accessed: Jun. 09, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Single-page_application&oldid=1024072473

[24] “64 Node JS Stats that Prove Its Awesomeness in 2021,” *HostingTribunal*, Jan. 16, 2021. https://hostingtribunal.com/blog/node-js-stats/ (accessed Jun. 09, 2021).

[25] *Reactjs*. Menlo Park, California: Facebook, 2021. Accessed: Jun. 09, 2021. [Online]. Available: https://reactjs.org/

[26] *Material-UI*. Paris, France: Material-UI SAS, 2021. Accessed: Jun. 09, 2021. [Online]. Available: https://material-ui.com/

[27] *beautiful-react-diagrams*. Beautiful Interactions, 2021. Accessed: Jun. 09, 2021. [Online]. Available: https://github.com/beautifulinteractions/beautiful-react-diagrams

[28] *Spring boot*. Palo Alto, California, U.S.: VMware, 2021. Accessed: Jun. 09, 2021. [Online]. Available: https://spring.io/

[29] *Hibernate*. Raleigh, North Carolina, U.S.: Red Hat, 2021. Accessed: Jun. 09, 2021. [Online]. Available: https://hibernate.org/

[30] PostgreSQL-Community, *PostgreSQL*. 2021. Accessed: Jun. 09, 2021. [Online]. Available: https://www.postgresql.org/