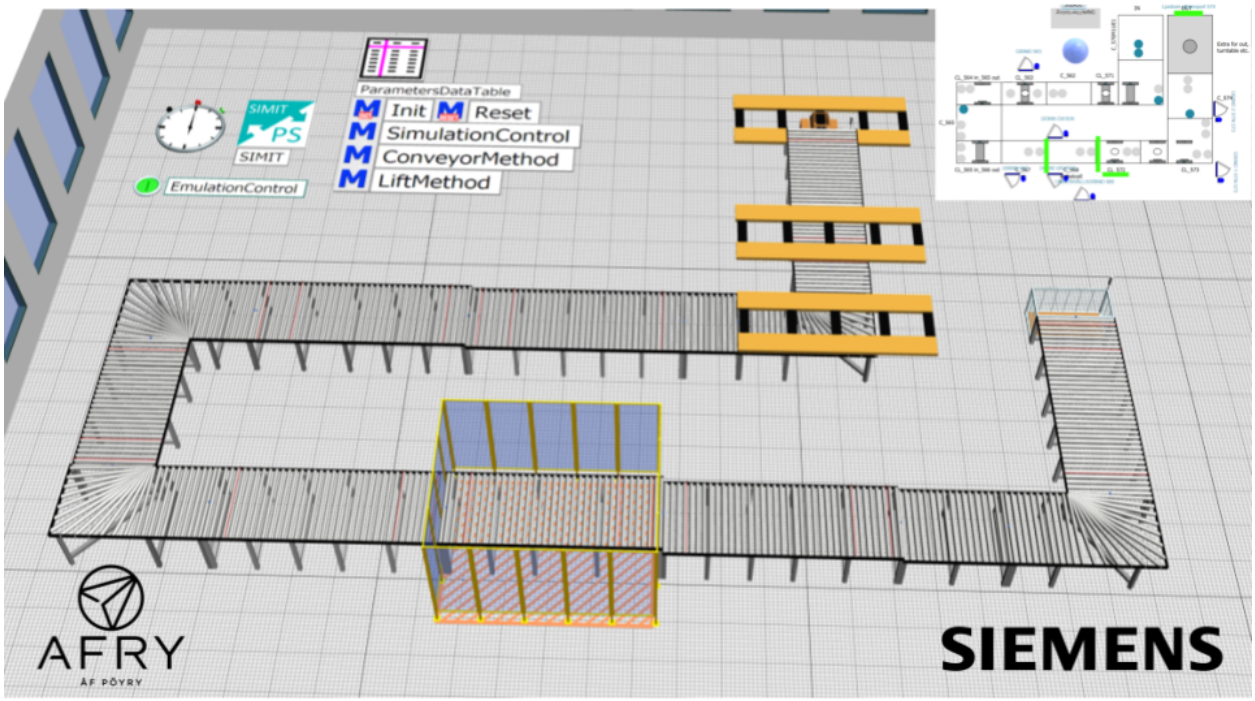
CHALMERS
UNIVERSITY OF TECHNOLOGY



Enabling Digitalization with Emulation Prepared Simulation

Developing a Framework for Hybrid Discrete Event Simulation Models

Master's thesis in Production Engineering

BRITTA OTTESJÖ & SOPHIA WETTERBERG

DEPARTMENT OF INDUSTRY AND MATERIALS SCIENCE

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Enabling Digitalization with Emulation Prepared Simulation

Developing a Framework for
Hybrid Discrete Event Simulation Models

BRITTA OTTESJÖ
SOPHIA WETTERBERG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Industrial and Materials Science
Division of Production Systems
Chalmers University of Technology
Gothenburg, Sweden 2021

Enabling Digitalization with Emulation Prepared Simulation
Developing a Framework for Hybrid Discrete Event Simulation Models
BRITTA OTTESJÖ & SOPHIA WETTERBERG

Supervisors:
Camilla Lundgren, Department of Industrial and Materials Science
Sanna Älegård, AFRY, Department of Advanced Manufacturing
Jessica Andersson, AFRY. Department of Supply Chain Management

Examiner:
Marcus Franzén, Department of Industrial and Material Science

Master's Thesis 2021
Department of Industrial and Materials Science
Division of Production Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Hybrid simulation model in Siemens Plant Simulation with corresponding emulation model in Siemens SIMIT in the upper right corner.

Typeset in LaTeX
Printed by Chalmers Digitaltryck
Gothenburg, Sweden 2021

Enabling Digitalization with Emulation Prepared Simulation
Developing a Framework for Hybrid Discrete Event Simulation Models
BRITTA OTTESJÖ & SOPHIA WETTERBERG
Department of Industrial and Materials Science
Chalmers University of Technology

Abstract

Virtual commissioning is a significant development area in today's industry. Virtual commissioning is a way for companies to stay competitive in today's rapidly changing market, using technological development and digitalization paradigms. Virtual commissioning consists of computerized models of real systems used to test the systems before physical implementation. Today's models have limited functionality, which means that different models are needed to test various parts of a system. This thesis aims to investigate how to create a hybrid Discrete Event Simulation (DES) and emulation model for improved virtual commissioning by extending a model's functionality and life cycle. This thesis conducted a literature study in combination with qualitative and quantitative analysis. The qualitative and quantitative parts were divided into two phases; a development phase used to investigate software connections and create a hybrid framework, and a case phase where the hybrid concept was tested and evaluated. The result was a creation of a hybrid model framework. The framework consists of a simulation object library prepared for emulation and best practice descriptions on building a hybrid DES model using the object library in the software Plant Simulation. A functioning hybrid model with the dual function of DES and emulation can be built using the developed framework. The hybrid model extends the functionality of simulation models, thus increases its life cycle. Further, a dual-functioning hybrid DES and emulation model enables advanced system testing and better visualization during virtual commissioning, testing systems in a way that has not been done before.

Keywords: Digitalization, Hybrid model, Virtual Commissioning, Discrete Event Simulation, Emulation, Digital Twin, Industry 4.0.

Acknowledgements

We want to give special thanks to our supervisors Camilla Lundgren, Chalmers, Sanna Älegård, AFRY, and Jessica Andersson, AFRY, for invaluable support and guidance throughout the project. We also want to give a large thank you to Marcus Franzén, Chalmers, who has not only been our examiner but also provided us with support and guidance. We also want to thank Marcus Andersson and Johan Nordling, Siemens Digital Industries Software for great technical support that has helped the project forward many times. We also want to acknowledge and thank Atle Zvantesson, AFRY, and other team members at AFRY that have helped us and supported the development of our project. Lastly, we would like to thank AFRY and Siemens for giving us this opportunity.

Britta Ottesjö & Sophia Wetterberg, Gothenburg, June 2021

Table of Contents

List of Figures

List of Tables

List of Abbreviations

CPS Cyber Physical Systems.

DES Discrete Event Simulation.
DT Digital Twin.

HMI Human Machine Interface.
HSEM Hybrid simulation and emulation model.

I4.0 Industry 4.0.
IoT Internet of Things.

PLC Programmable Logic Controller.

TBL Triple Bottom Line.

1
Introduction

This thesis will begin with an introduction describing the areas of interest and the issues to be investigated. Purpose, aim, and limitations will be presented, leading up to the research questions for this thesis. This chapter ends with a description of the report structure.

1.1 Background

Digitalization is nowadays a prerequisite for production companies to stay competitive [1]. Industry 4.0 and digitalization bring many cost-saving and quality increasing possibilities for the production industry [2]. One of those is the simulation and emulation software that can be used, among other things, to shorten ramp-up times as new implementations are tested, verified, and validated virtually in advance of the physical implementation [3], [4]. Simulation and emulation tools enable virtual commissioning to become a rapidly growing field within digitalization and Industry 4.0 for the manufacturing industry [5]. Virtual commissioning requires building simulation and emulation models to test specific systems and model different environments [6] (the difference between simulation and emulation is described in Section *2.4 Simulation and Emulation*). Today simulation and emulation models are only built for particular purposes with limited functionalities, so many additional models need to be created to meet different needs during the commissioning phase. The main issues with simulation in virtual commissioning are finding the right level of detail, which affects the life cycle of the simulation models [6]. Since different models are created to simulate various aspects of a system, there is seldom any direct integration done between real-time emulation or Discrete Event Simulation (DES) in one model [6]. When separate models are used within these fields, the models' life cycles are often short. They are seldom designed to enable the possibility of transitioning the use of the model to different areas or ultimately enabling the creation of a Digital Twin (DT) in later development stages.

Today there are many industrial simulations- and emulation software available on the market, ranging from the simulation of production flows to a complete emulation of the industrial control system [7]. Combining DES and real-time emulation in one hybrid model would make it possible not only to perform tests during the virtual commissioning but also in all stages of a project pre-planning, improvement work to production control. Thus, extending the life cycle of the simulation model by increasing its functionality [3]. Commissioning time in the industry has been

reduced mainly with the introduction of virtual commissioning [7]. Commissioning can be further improved by reducing the required time spent on model building and improving the models' functionality [7].

Previous research proposes Plant Simulation as a tool for virtual commissioning [8]. There is a need for a real-time simulation platform and a new way of utilizing simulation modeling to take advantage of DES software for virtual commissioning. Therefore, AFRY's team that is working towards digital twins claims that an investigation is necessary into connecting DES and emulation creating a hybrid model to see what potential benefits a DES model with dual-functionality could have for virtual commissioning and DTs [7]. Further, to investigate if a hybrid model can impact the life cycle of a simulation model, possibly reducing the need for creating multiple complex models, thus increasing the model's life cycle, saving both time and money.

1.2 Purpose

As described in the background, many models are needed for virtual commissioning to test different systems or environments, such as simulation and emulation of various systems. Each model represents a highly time-consuming process to build, verify and validate before it can be used. It may be possible to reduce the time and money spent building the models by creating a hybrid model with dual-functionality. Hybrid models could extend the life cycle of early DES models or reduce the number of simulation models needed in the commissioning stage, thus saving time and money. The purpose of this thesis is to investigate what possibilities and potentials a hybrid model with dual-functionality could have for virtual commissioning and the industry as a whole; enabling companies to stay competitive and at the forefront of today's rapid digital development using hybrid models.

1.3 Aim

This thesis aims to investigate how to integrate real-time emulation in a DES model, thus exploring how to build a DES model to enable this integration of dual-functionality to create a hybrid model. With this investigation, evaluate the possible benefits and drawbacks of combining DES with real-time emulation in a single model, and if a model with this dual-functionality could extend the life cycle of the DES model and increase its functionality.

1.4 Specification of Issue Under Investigation

Investigation into how emulation can be used to control a DES model using Siemens software Plant Simulation and SIMIT, aiming to create a hybrid model. Further, to investigate what is needed in the model-building stage in this software to enable this connection of real-time elements in the DES environment.

Research questions:

1. How can an emulation model control a DES model? What are the prerequisites and methods for building a hybrid DES model with this dual-functionality?
2. What are the benefits and drawbacks of using a hybrid DES model with dual-functionality compared to using two separate models for simulation and emulation when it comes to virtual commissioning and the creation of a digital twin?
3. How can a hybrid DES model with dual-functionality extend the life cycle of a simulation model?

1.5 Limitations

This thesis will only investigate and evaluate the possibilities of dual-functioning simulation-emulation hybrid models. The thesis will not aim to create the software connection, thus limit itself to using software with existing external connections between the simulation and emulation software. The software used is Siemens software Plant Simulation for the DES model and Siemens SIMIT for the emulation model.

This thesis will not create a new emulation model in SIMIT, only use an existing SIMIT model of a production cell created by AFRY. Some modifications will be done to scale down the emulation model to fit the scope of the thesis. The focus is on evaluating and testing the hybrid model function, not to mimic the system exactly. The production cell used for modeling will not be visited by the students, which means that this thesis will not verify or validate the results on-site. The data used for the case model was collected in an earlier project by AFRY. Therefore, no new data collection was made in this study.

This thesis will limit its research on virtual commissioning to a virtual plant with real controllers, see Section *2.3 Virtual Commissioning.* Further, comprehensive stress tests will not be researched in this thesis.

1.6 Report Structure

This first chapter presents the background, purpose, aim, and limitations of this project. The structure of the rest of the report is as follows; **Chapter 2** presents the theoretical framework surrounding the topic. **Chapter 3** presents the methods used, how they were used, and the theory behind them. **Chapter 4** presents the results from the quantitative and qualitative development and case phases, followed by a discussion of the findings connecting the results to literature and areas for future research in **Chapter 5**. **Chapter 6** concludes the report.

2
Theoretical Framework

In this chapter, the theoretical framework for the project will be presented to provide the reader with sufficient knowledge of the topic without further previous knowledge.

2.1 Industry 4.0 and Digitalization

Industry 4.0 (I4.0) and digitalization are today well-known topics and key ideas for driving industry innovation [9], [10]. I4.0 can be defined as the fourth industrial revolution that brings digitalization of industry [10], Figure 2.1. I4.0 includes technologies like the Internet of Things (IoT), Cyber Physical Systems (CPS), Big Data, simulation, etc. which opens up for the development and use of virtual commissioning and DTs [9]. These innovations and new technologies have become a must for companies to stay competitive in today's market [11].

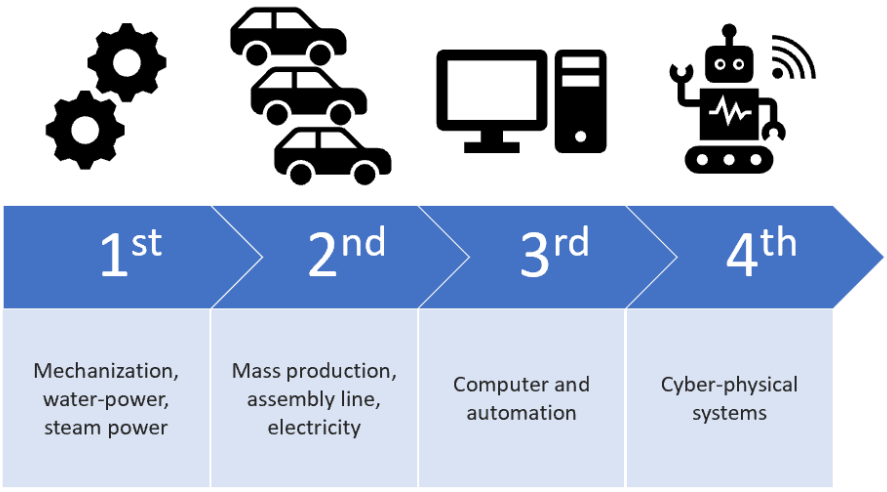


Figure 2.1: The four industrial revolutions. Adapted from Roser [12].

Digitalization is about the evolution and innovation of technology and creating new business values with the technology as a driver [1]. Looking at the life cycle of a technology, the evolution of the specific tool needs to go hand in hand with how it is used so that the use is value-creating for the company [1]. This is why concepts and technologies need to be evaluated and developed with a life cycle perspective concerning their usability [13]. Among the technologies driving digitalization and I4.0, is the technology of simulation, which can be seen to capture many of the design

principles of I4.0, especially that of simulation-based approaches [5]. Simulation has grown with the introduction of the IoT, CPS, Big Data, etc., which are drivers for the development and use of simulation approaches [5]. Hybrid simulation models and DTs seem to be promising approaches to achieve a digital industry, and research into these concepts is of great value to the industry [5].

The development and evolution of Industry 4.0 will come at a cost to develop, but the benefits of the transition into a digitalized industry will outweigh the costs [9]. Thus research into how to use these digital concepts and further optimize them is needed to cover the challenges of lacking competencies and IT maturity in industry and development today [9].

2.2 Digital Twins

A DT can be defined as a virtual model of real objects in a virtual setting that simulates the real-world or reflecting the life of the real-world [14]. A DT is built in four levels: geometries, physics, behaviors, and rules [5]. The DT should represent all the information existing in the physical system that can be obtained by studying it, represented as data or algorithms [15]. Kritzinger et al. [15] propose that three levels of integration can be seen within the definition of a DT and that they should be categorized under the digital model, digital shadow, and DT. The different categories represent the integration level between the physical and digital object, from manually modeled objects that are not connected to the physical object (digital model) to fully integrated digital objects with real-time data exchange (DT) [15]. Figure 2.2, shows the data flow for each categorization.

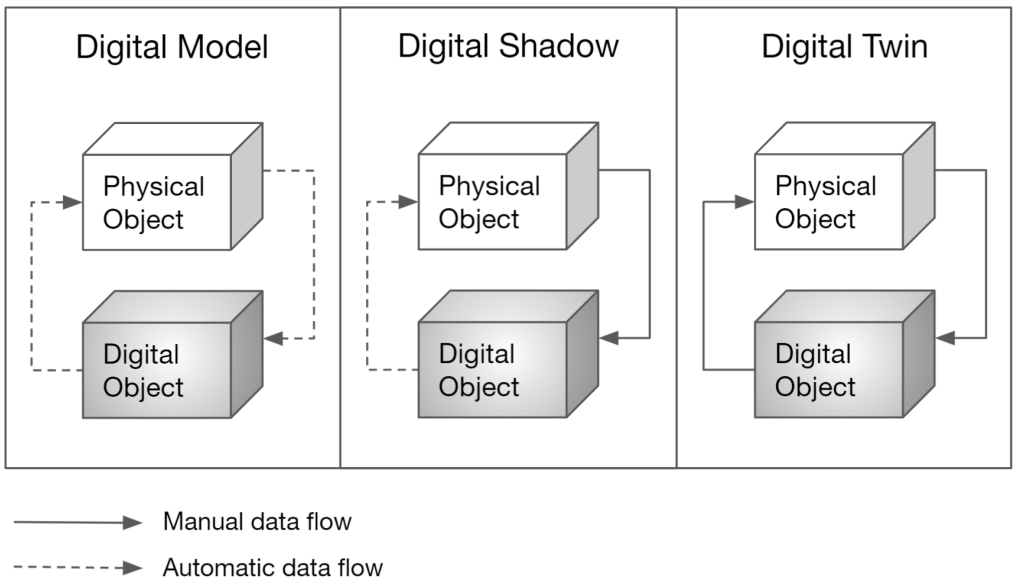


Figure 2.2: Three subcategories of data integration. The figure is adapted from Kritzinger et. al [15].

The DT can be seen as the most promising enabler to achieve smart factories, with seamless integration between the physical world and its digital copy [16]. For the production industry, a DT can potentially create a more reliable, flexible, and predictable system environment [16]. Building a comprehensive virtual representation of a system would require huge data volumes and is not feasible. Instead, architecture for building and enabling DT is needed [17]. Simulation models can serve as a base for implementing a DT in practice [16]. A DT can be made up of simulation models, which link the relevant digital artifacts in the system and create the base model. For the DT to represent reality and not be classified as a digital model or digital shadow, it needs direct feedback to and from the digital object [15]. This enables the digital twin to evolve with the real system [16].

For a DT to be relevant early in a process, it should preferably be designed in parallel with the physical realization of the system, i.e., commission phase [17]. Today's virtual commissioning could enable this since many simulation models are built in the early stages of development projects [6]. Thus, there are many opportunities for developing DTs in an early stage of a project. However, for the DTs to stay relevant through the project's life cycle, certain parameters and ways of building the later-on introduced parameters must be taken into account as early as when the data is collected and when the early models are built.

When going from a digital model to a digital shadow or a digital twin, automatic data from the real world is needed [15]. This step to extend the model's life cycle must be taken in the creation phase to enable future model development and evolve it. Bi-directional automated information needs to be integrated into the digital model or digital shadow to create a DT. With this, and as Lim, Zheng & Chen [13] describes it, DTs can then be beneficial in most stages of a product's life cycle, from design to end-of-life, in various ways.

2.3 Virtual Commissioning

Virtual commissioning is a method to speed up the actual commissioning through testing in a virtual environment with real controllers first to reduce the time required for debugging and making corrections on the real plant [5], [6]. Virtual commissioning can aid in the actual implementation, detect errors in design or plans, test improvements or optimization ideas, run several executions of test programs to identify many different scenarios [18]. It can bring instant simulation-run feedback leading to faster and better control development [18]. Using simulation models and emulation signals (virtual commissioning is most often done with emulation in a virtual Programmable Logic Controller (PLC) to best mimic real machine signals and behavior [18]), the commissioning can be visualized and validated virtually [5]. This is one large focus of the concept of digital factories [18]. There are four types of commissioning; virtual plant with virtual controllers, real plant with virtual controllers, virtual plant with real controllers, and real plant with real controllers, see Table 2.3, [6].

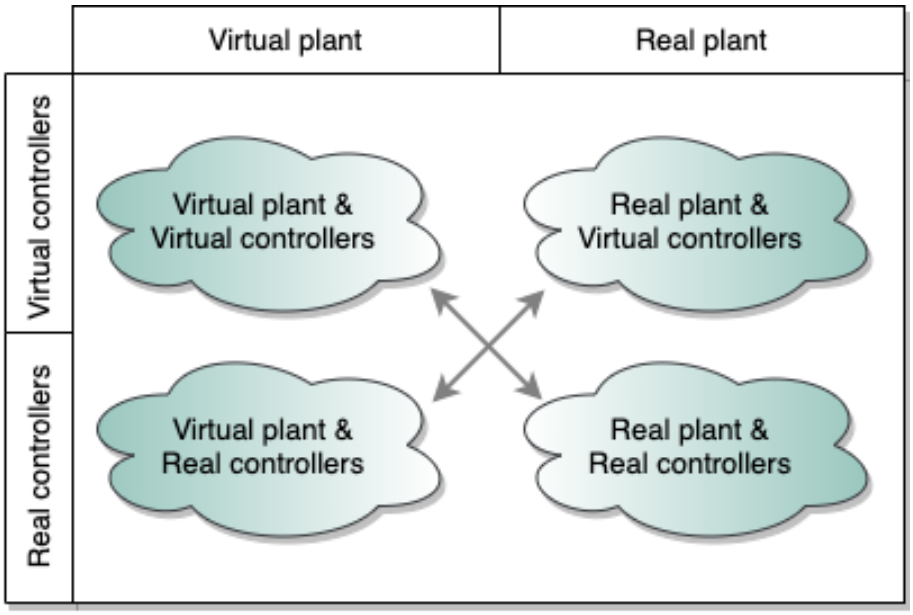


Figure 2.3: The different types of virtual commissioning. The figure is adapted from Lee and Park [6].

Creating simulation models of the factory layout and the new implementations before implementing the changes can help save time and money, especially in the ramp-up time [5]. Without virtual commissioning, the actual commissioning entails making corrections, debugging, and stabilizing on the real plant, which is time-consuming [6]. During the time the line is occupied making the commissioning, there will be no production thus expensive for the company [6]. Simulation models can further help to evaluate risks, implementation barriers, and impacts on operational performance [5]. This can provide directions for the company's road towards I4.0 [5].

Studies show that the major drawback of virtual commissioning is that it takes a lot of time and effort to construct accurate virtual models [6]. Most models consist of several objects like transport lines, machines, robots, fixtures, etc., which all need to have correct data, geometry, and kinetics [6]. Further, since most models should communicate with real controllers, the virtual objects in the model must be correctly programmed on the level of sensors and actuators, which could be hard for control engineers as they do not always have deep knowledge in simulation [6].

Emulation on simulation models is central in virtual commissioning [6]. If a simulation model could also be used for emulation of the system, then the model's life cycle could increase [3]. Increasing the functionalities and operating ranges of simulation models would lead to a lower need for new models. The same model could be used in several ways and for a longer time [3]. This would decrease the time needed to create simulation and emulation models to virtually commission their system or create a DT. Further, if a minimal level of detail could be established in the creation of a model like this, even more time could be saved to avoid excess details and data in the simulation model. The details are one of the most time-consuming parts of creating a simulation or emulation model [6].

2.4 Simulation and Emulation

Simulation and emulation models are computerized interpretations of a real system [19]. Regardless of the model being a simulation or emulation model, there will be differences in performance compared to the real system [19]. Simulation models are based on "good enough" data to represent the system. In contrast, emulation models aim to minimize the differences to the real system to be implemented in the real system or that a real part of the system could be integrated into the emulation model [19]. Virtually modeling a system can make it possible to learn about the whole process, identify bottlenecks in a production line, or test new solutions to a system without or before implementing it to the real system. Virtual modeling can save both time and money as implementing ideas to the real system is very timely and costly [3].

Simulation can be described as a process of designing a model of a real-world or an idea. The simulation model should describe the system in a way that its behaviors can be analyzed [15]. Ingalls [20] describes simulation as: *" simulation is the process of designing a dynamic model of an actual dynamic system for the purpose either of understanding the behavior of the system or of evaluating various strategies (within limits imposed by a criterion or set of criteria) for the operation of the system."*. Simulation has, according to Tao et al. [16], gone through three stages, starting with the simulation of specific devices or areas, moving to the simulation of generic devices, and now simulation is taking place in multilevel and multidisciplinary levels; enabling the possibility to simulate the product or process through its life cycle. Simulation models are normally used for experimentation, testing new ideas, improving throughput times, or testing production volume changes to find the most suitable solutions [19], [7]. These models are often flexible, and it is easy for modelers to make changes [19]. Simulation models are seldom run a single time but iteratively to evaluate numerous outcome scenarios [7].

Emulation models are not used for experimentation of new ideas to the same extent as simulation models [19]. Emulation models are most often used to test the control systems under different loading conditions and for virtual training for operators providing trustworthy analysis [19], [7]. Emulation models must comply with the real-world signals and functionalities more than a simulation model, and it often executes in real-time [19]. Emulation models are very specific and detailed, and they are based on real hardware, complying with the laws of physics and interactions, making it very realistic and possible to run by the real system's controls [7], [3], see Figure 2.4 for a simple illustration. Emulation models are often programmed with PLC code to make it as like the real system as possible [3]. Being able to test the solution onto the "real" system in an emulation model can largely decrease the ramp-up phase in the real world when the idea is finally implemented [3], [4]. Studies have shown that ramp-up time can be reduced by up to 50 % with the use of simulation and emulation [3].

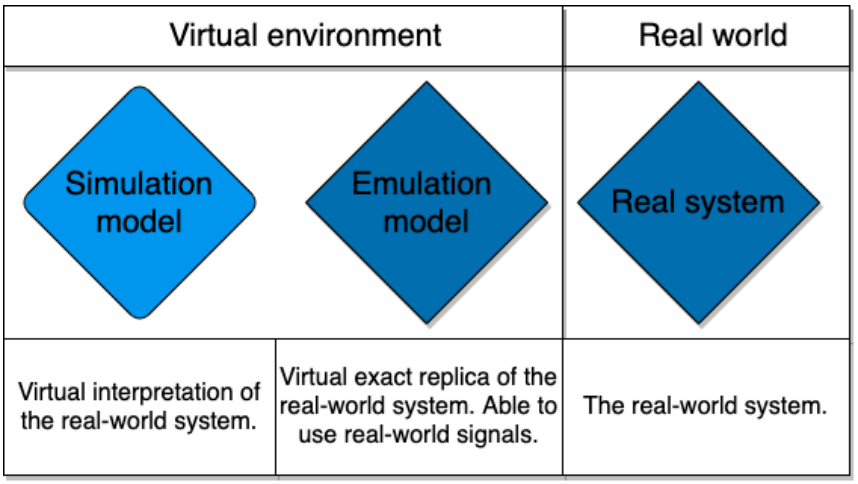


Figure 2.4: Simulation and Emulation compared to a real-world system.

Simulation is a key technology of I4.0 and digitalization. It can be seen to support the implementation of I4.0 as well as support the process from designing to the optimization of a system [5]. Some research indicates that simulation techniques like DES and emulation are applicable in an I4.0 environment even if the complexity of such systems has increased today [5]. Also, hybrid simulation models and DTs will have a more central place in the future of simulation. Simulation can be seen to capture many if not all design principles of I4.0 implementation, strategically, tactical, operational, etc. Whereas, as mentioned briefly, hybrid models with more functionality and DTs seem to offer more value for the digitalization of industry in the future [5]. Even if traditional approaches like DES will still stay valid, the underlying assumption is that they need to evolve. It is possible by emulating a simulation model to increase the value and life cycle of the simulation model, as it can then be used for more than visualizing the factory and in more phases of the project [3]. This would potentially create an environment where initial experimentation can be carried out as usual in the simulation model, but where emulation parts or layers can be added to make a more thorough test or in final stages before implementation to finalize a solution.

2.4.1 Discrete Event- vs. Real-Time Simulation

There are two types of simulation models used in industry today, non-real-time simulations, DES, and real-time simulation, Continuous Simulation [21], see Figure 2.5. In a DES model, time does not pass by like in the real world. The simulation time advances at discrete points in time, at the time when events occur [22], [21]. It is assumed in a DES model that the system state is unchanged between these points in time [21]. DES is a computer-based modeling system [23], and there are three methods to program DES models; activity-based, event-based, or process-based simulation model [21]. The activity-based system's time advances with a set time step, while the event-based system's time advances at the point of events [21]. The process-based approach uses parallel software threads and processors to cope with an increased parallel thread computing requirement [21].

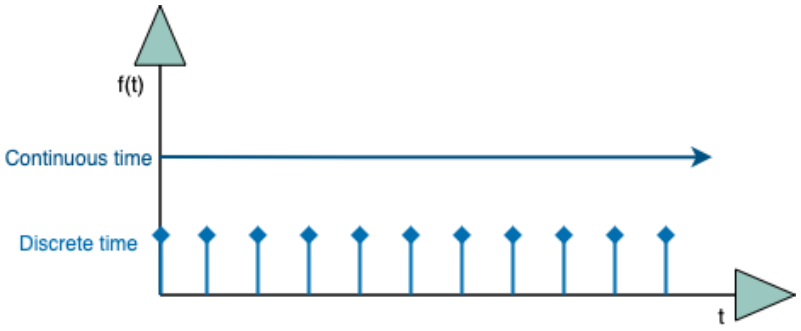


Figure 2.5: Continuous time vs. Discrete time.

An example to explain DES easily can be a train ride between Gothenburg and Stockholm. The train stops at several stations on the way where passengers enter or leave the train. A DES model visualizes each stop as events occur at these points in time, but it does not simulate the ride between the stops since new events do not occur [22]. DES can be seen as jumps in time because the simulation only runs at the time these events happen, making the simulation run take a shorter amount of time compared to if it would simulate the wait between events [22]. DES models are widely used within industries today, e.g., to model manufacturing plants, distribution centers, or service facilities [21]. DES models can be used to validate the performance of materials handling systems or order picking and product location strategies, etc. [23]. DES is one of the most popular approaches today for verification of manufacturing systems [6].

Continuous, or Real-time simulation, on the other hand, simulates events on what seems to be a normal time scale, as they would occur and behave in the system in the real world [21]. Even real-time simulation is executed discreetly, but it is based on algorithms to backlink the normal, continuous, time [21]. It can be said that real-time simulation simulates everything in comparison to DES because it simulates even the wait between the events, making it more like the real world's time [21]. Returning to the train example, a real-time simulation would simulate the entire trip and not only the stops when things happen. Real-time simulation is widely used within industries today, e.g., aircraft, automobiles, robotics, and continuous flows like weather forecasting.

2.4.2 Verification and Validation

It is crucial to have a good system for verification and validation of any simulation system to create a simulation model accurately representing the actual system [24]. A simulation model is created to mimic the phenomenon of the actual line as much as possible [24]. Verification and validation of simulation models are often very time-consuming and costly, but creating a simulation model that does not reflect the real world properly would be a waste of time, as changes or optimizations in the model would not apply to the actual line [25].

The concepts of validation and verification are often mixed up as they are both used for confirmation [24]. Verification can be said to be the "establishment of truth" [26]. It means that if a simulation model has been verified, then the truth has been demonstrated in the model [26]. Verification can, however, only be done on closed systems because, in open systems, there are unknown parameters that cannot be truth checked, e.g., a weather condition equals a human activity [26]. Even if the weather conditions are approved for taking a swim in the lake, that does not mean that all people will take a swim, even if one person does. Good weather equals swims can not be verified, but dropping a thin glass on concrete from 2 meter's height equals the glass breaking can be verified; it can be said to be true.

Validation is an establishment of legitimacy, not necessarily of truth [26]. A validated model is internally consistent and does not contain any obvious logical errors [26]. A model is validated if the results are found acceptably like the intended outcome [27], [25]. Validity is often defined regarding contracts, arguments, or methods [26]. Examples of validation techniques are through the use of animation or graphics, comparison with similar models, stress tests, etc. [25]. An example of validation can be to answer the question, "will the water in the lake be warm on this summer's day?". The true answer depends on the person taking a swim, some will say that it is warm, and some will say that it is cold. The expected outcome may be that it is warm on a summer's day, and if that is the answer, then the question has been validated. Validation can be seen as more subjective than verification.

2.5 Hybrid Models

Hybrid simulation is an emerging research field [28] and can be defined as combining two or more simulation methods [29]. Most hybrid research is focusing on dual-functionality models [5]. Three approaches to hybrid models are defined in the literature; a discrete model with continuous elements, a continuous model with discrete elements, or a model with equal portions of discrete and continuous elements [21], see Figure 2.6.

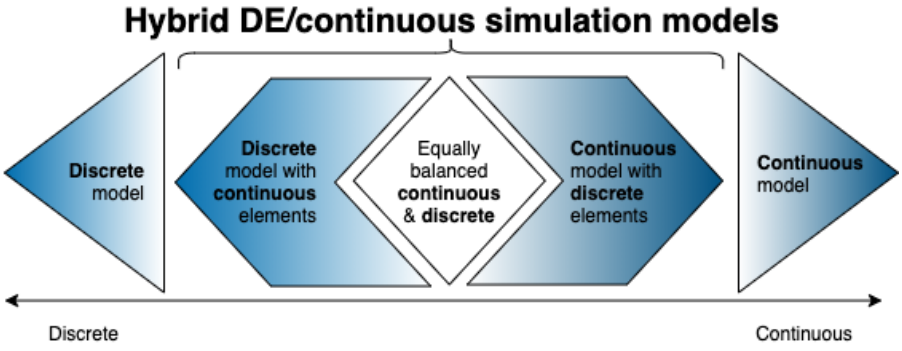


Figure 2.6: Visualization of the relation between Discrete-Event and Continuous time in hybrid models.

Hybrid models can also be defined as [28];

- Sequential - the output of one model is the input to another
- Enriching - one model is dominant and the other supportive
- Interaction - cyclic interaction between equal models
- Integration - merging models that cannot be distinguished from each other

Hybrid models can be categorized by their integration between sub-models [28];

- Automated - software package
- Manual - i.e. copy paste
- Integration - using intermediate tools (i.e. Excel or similar)

For hybrid models, it is important that the software can handle the implementation of the different functions, i.e., functionalities for modular model development and the possibility to add codes and integrate with other applications [30]. The following three simulation methods and combinations of these have been given the most attention in the literature: DES, system dynamics (SD), and agent-based simulation (ABS). These simulation methods have been used to a considerable extent in hybrid simulation research [28]. Many reports mark a communication difficulty between models as time perception differs between models. The perception of time is a reoccurring problem that is being investigated to some extent [31].

Many researchers agree that hybrid models are an interesting topic, especially because of the clear connection to the development of Industry 4.0 [5], [7]. Many types of hybrid models have been researched, for example, DES with continuous models or different types of simulation models with emulation models [30]. Hybrid simulation have the potential to support the evolution of I4.0 and digitalization in many ways (e.g., strategic, tactical, operational), foremost in the area of virtual commissioning and DT development [5], [32].

2.5.1 State-of-the-Art: Hybrid Simulation and Emulation Models

The hybrid simulation-emulation models found in the literature ranges from full-simulation or full-emulation models to either model with elements of the other to equally balanced hybrid models, see Figure 2.7, [21]. Some articles regarding hybrid models can be classified as older due to the rapid technology development of the software used. One of the older articles looked into the integration of emulation functions on the objects in simulation libraries [4]. This is an interesting way of incorporating emulation early in the simulation modeling process since the objects will be created with the emulation functions already there. The motivation behind this is the same as today's view of reducing the complexity of commissioning by enabling thorough testing of not only simulation flow, but control systems [4]. Meyer et al. point out that if one should edit the simulation library, it is beneficial to group the simulation objects from an emulation point of view and focus on creating a standardized way of working from both sides [4].

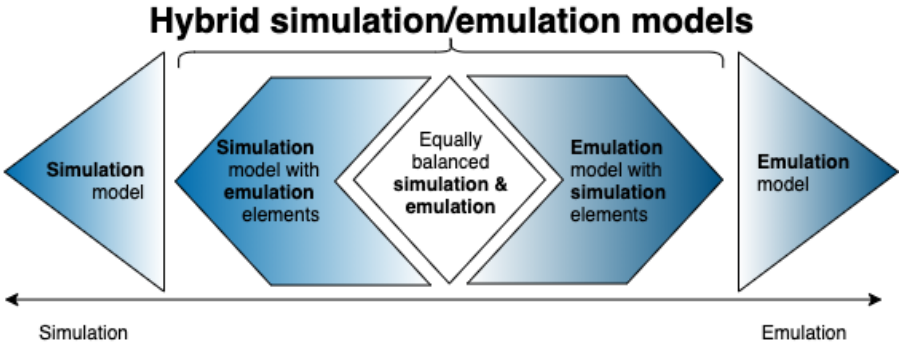


Figure 2.7: Visualization of the relation between Simulation and Emulation in hybrid models.

Researchers have carried out tests to control simulation models with emulation software. One example is using industry-standard communication protocols to control a simulation model with a live PLC [33], proving that hybrid models could extend the life cycle of a model by running the simulation model with real-world controls. What-if stages could be tested using the hybrid models in later development phases if the models are kept up to date [33]. The issue the researchers had at the time was that it was hard to run large systems and that standards were needed to make systems compatible [33].

There are also newer articles that have experimented with controlling objects in simulation software, with virtual commissioning in mind, using Siemens software [34]. The research was done on loading 3D graphics from CAD models into Plant Simulation to simulate and control a realistic simulation model. Enabling extensive testing of control system on a realistic model before installation, to reduce the chance of damaging components and to give an overview of the movement in and around the real objects similar to PLC and robot program testing [34]. Hybrid models enable cross-function testing and open up many possibilities. However, it is essential to state that the benefits have to outweigh the cost of more extensive models and testing [34]. More advanced virtual commissioning will ultimately require more realistic models. Connecting and integrating functions of different types of modeling methods will thus open up for more detailed models moving closer to the realistic DT needed for virtual commissioning [34].

2.5.2 Examples of Research Application

Limited research has been carried out on the prerequisites set on the simulation program for an emulation program to be integrated [30]. In an article from 2005, the prerequisites were based on a methodology of sequential steps [30]. The first step is to create a simple simulation model to be used as the base model, after that to create a detailed emulation model, and lastly to investigate the integration of controllers between the two models [30]. The article states that without the proposed

prerequisites, the models would not be convertible. However, using the mentioned steps, the base simulation model can be converted into an emulation model, running tests using the control system [30].

Some research proposes using emulation as a verification and validation tool for simulation models because ensuring that the simulation model truly represents the real system is generally very hard [3]. A simulation model can be tested and compared to the real system or the intended system using emulation with real controllers. Problems that could occur in the system can, with these tests, be detected early in the simulation phase, thus saving both time and money [3], [32]. The research found is consistent in that the simulation model should be built in the earlier stages and integrated with the emulation at later stages [3]. Previously, the simulation model has been left aside when the execution phase begins, and the use of emulation model starts [3], [32]. Research states that implementing emulation to a simulation model can increase the use and will thus extend the simulation model's life cycle and improve the commissioning time [3].

There has not been much research on the hybrid integration of the simulation program Siemens Plant Simulation and the emulation program Siemens SIMIT. The software Plant Simulation and SIMIT can be connected to a variety of other software, which are described in their respective help guides [35], [36]. Research has been done on the connection between Plant Simulation and, for example, PLCSIM Advanced by the software's developer Siemens [37], and by university theses [38], [39], [32].

2.6 Sustainability

Sustainability can, with benefit, be viewed from the Triple Bottom Line (TBL) perspective [40]. The TBL can be defined as viewing and measuring sustainability from the three pillars: economic aspects, environmental aspects, and social aspects [40], see Figure 2.8.

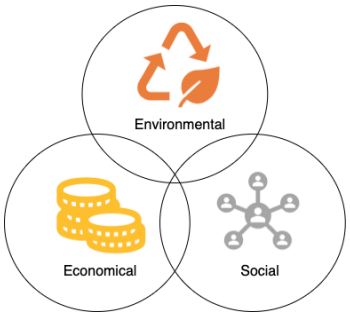


Figure 2.8: The aspects of the triple bottom line.

The economic aspects are connected to long-term profitability. The environmental aspect regards reducing the negative environmental effects towards zero, or even to positive environmental impacts. The social aspect includes that the company should

be providing a workplace where development and learning can take place [41]. Today's manufacturing companies need to adapt to a rapidly changing environment, and this needs to be done with sustainability in mind [41]. The production needs to address each area of the TBL. Benefits of simulation, emulation, virtual commissioning, DTs, and other I4.0 applications can be found under all three aspects of the TBL [42].

2.6.1 Industry 4.0, Digitalization, and Digital Twins

Sustainability can be seen as one of the driving forces for I4.0 and digitalization [42]. I4.0 and DTs enable virtual testing and virtualization as well as visualization of systems and operations of a real system. Companies can gather large amounts of data and optimize their systems in a faster and better way. Optimization can often be done concerning either part of the TBL, economy (e.g., throughput, productivity, or product quality), environment (e.g., monitoring emissions), and social sustainability (e.g., workload, ergonomics) [42]. The correlation is then that I4.0 concepts can increase sustainability in the three different aspects and that there is a lot to gain regarding sustainability by moving to a digitized industry [42]. Further, DTs or other visualization of data can measure and keep track of environmental data that can be used for optimization or improvement.

A drawback on the social aspect of I4.0 can be the perception of Big Data. Big Data is great for making decisions based on statistics or collecting behaviors [43], etc. People may be offended by companies collecting personal data because they get a feeling of being overwatched or even controlled. However, in the long run, people will, because of Big Data, be able to get and produce products that will be popular in the market because they can be adapted in the early design stages to people's opinions. The same idea can be said about cybersecurity [43] because that and Big Data goes hand in hand and could lead to systems becoming more vulnerable and thus less socially sustainable. A social drawback is an uncertainty that a digitized industry can bring to a workforce, of layoffs, that now more competence is needed to do tasks since it is more about serving the machines in the future [43].

The sustainability aspects of I4.0 in an overview seem to be mostly economic gains, as the lowest hanging fruit [42]. Further, a literature review by Kamble [44] indicated the same, that I4.0 concepts contribute to economic and environmental sustainability, but that many concepts also have great potential to contribute to the social sustainability aspect as well.

2.6.2 Virtual Commissioning, Simulation and Emulation

Economic sustainability effects of simulation, emulation and virtual commissioning are similar to that of I4.0 [42]. Systems that can be tested in a virtual environment before the real implementation can save time, money, and staffing before, during, and after installation since most disturbances or miscalculations have already been found. Virtual commissioning enables a reduction of commissioning time of up to

50%, which enables the production to keep going for a longer time [7]. In the long run, virtual commissioning can therefore create large cost savings for the company leading to better economic sustainability.

Virtual commissioning can also affect the environmental aspect. Not only will it save time for commissioning and ramp-up [7], but then it is also possible to test different scenarios and calculate the environmental impact in the virtual environment to optimize the system and reduce its environmental impact as early as in the design stage.

Regarding social sustainability, simulation and virtual commissioning makes it possible to virtually test scenarios for workers and calculate, e.g., the number of steps required to do a certain job, workload, or ergonomics [42]. This can greatly increase the work environment for the employees and lead to greater social sustainability at the workplace if virtually optimized and then implemented in real life.

A potentially negative part of the simulation, emulation and virtual commissioning is the crucial part of finding the right level of detail in the model to correctly represent the actual system [45]. In virtual commissioning, it is crucial to code on a very detailed signal-level, to better replicate the real system [20]. This could be frustrating as it is time-demanding and requires expertise in both simulation and signal processing. Connecting these improvements to the TBL, a system like this could further increase the sustainability aspects of economics, the environment, and social sustainability in the same way that other I4.0 concepts could [42].

3
Methodology

This chapter will present a detailed description of the methodology and work procedure used in this thesis. The methodology was based on a mixed methods approach as a triangulation of the methods; literature review, qualitative analysis, and quantitative analysis [46], Figure 3.1. Mixed methods approaches are suitable when one type of method (e.g., qualitative or quantitative method) is not enough to answer a research question [47]. Since this thesis is comprised of three research questions that all require different types of analyses, and most could not be answered by a single type of analysis, a mixed method approach was suitable. Since the research area of hybrid simulation-emulation models is sparse, a literature study combined with qualitative and quantitative studies using experiments and a case would provide a stable ground for analysis of the research area.

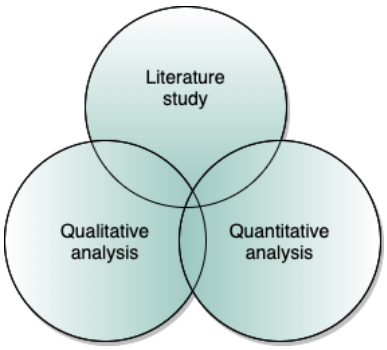


Figure 3.1: Visualization of the triangulation of methods used in this thesis.

There are different types of mixed methods designs, e.g., sequential, parallel, or nested design depending on the order in which the research is made [47]. In a sequential design, one method is performed before the other, in a parallel design, they are done simultaneously, and in a nested design, the methods are embedded [47]. In this thesis, a combination of sequential and embedded design was used. The research started in a sequential manner, where the main literature study was conducted followed by the development phase, and thereafter the case phase, see Figure 3.2. An embedded approach was needed during the development phase for the iterative approach of developing the hybrid model framework and further test it with a case. Follow-up literature review and qualitative and quantitative analyses were combined. The primary purpose of a triangulation of methods in the development and case phase was to verify findings using different methods to increase the trustworthiness of the results [48].

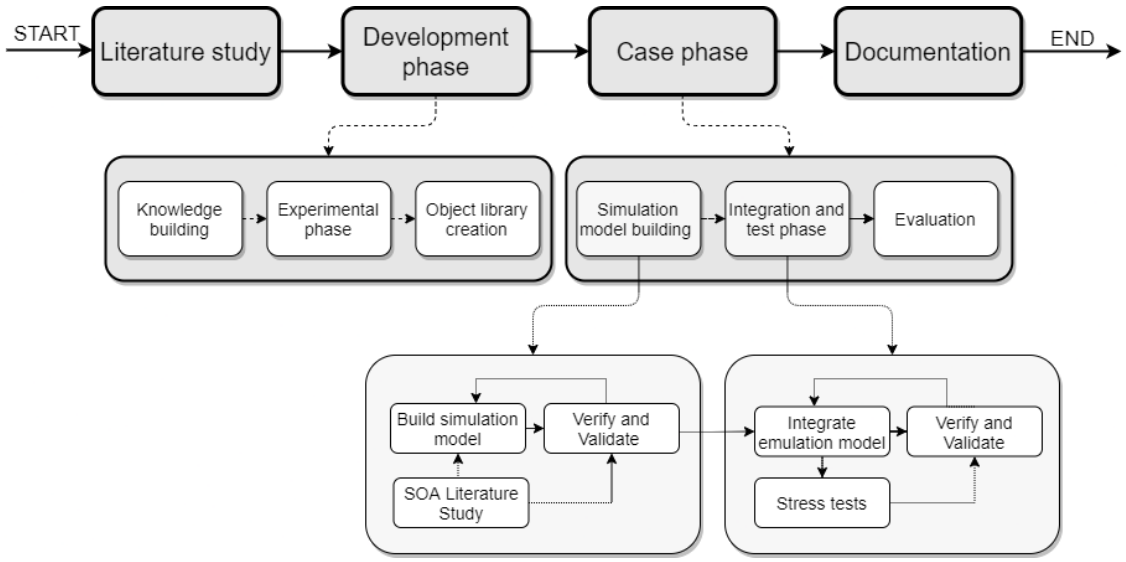


Figure 3.2: The work procedure presented in a timeline perspective.

The data used in the thesis was a mix of primary and secondary empirical data. The literature study accounted for most of the secondary data from previous research into hybrid models and was used to build the theoretical framework and develop the hybrid model. Primary data was retrieved via quantitative and qualitative analyses during the development phase's iterative process and later on in the case. The development phase inductively retrieved information and constructed theories that were passed on to the case. The case was deductive as our theories were tested during the phases; model building, integration test, and evaluation phases [49].

The object-oriented simulation software Siemens Technomax Plant Simulation version 16.0.4 was used for all simulation models. The emulation software Siemens SIMATIC SIMIT SP DEMO version V10.2, with virtual controllers, was used for the emulation models in the knowledge and experimental phase. The full version of Siemens SIMATIC SIMIT SP version V10.1 and V10.2 and Virtual controller software SIMATIC Manager in the VMware Workstation Pro 14.0 was used for the final tests in the case.

3.1 Literature Study

The literature study started with the research into the area of hybrid models, which consisted of searching for, and studying, previous research on the topics covered in Chapter 2 *Theoretical Framework*. The literature study explored what data was available using inductive research and then structuring it down with a description [48]. The purpose of inductive reasoning is to develop a theory [49], which is suitable for building a theoretical framework and for developing a theory of the sparsely explored research area of hybrid models.

The literature supporting this report originates mainly from secondary sources, such as published research papers, articles, handbooks, or books. A time filter of 2010 was used when searching for literature regarding concepts such as I4.0, Virtual Commissioning, and DT's because of the rapid technological development. The literature older than 2010 acquired had to be carefully reviewed and discussed regarding its relevance before use. The databases used for the literature study have mainly been Chalmers Online Library, Science Direct, Google Scholar, and Scopus. Additional literature was provided from peers and our previous work during our studies. The search keywords used were: Simulation, Emulation, Virtual Commissioning, I4.0, DTs, Plant Simulation, SIMIT, Hybrid models, DES, Real-time simulation, and Digitalization, and combinations of these.

A semi-systematic, snowball-structured research approach was used to find and go through literature [46]. The semi-systematic research approach is often used to achieve an overview and to identify themes within a topic under current limitations, e.g., of time and budget [46]. The semi-systematic approach was suitable for topics with many studies published and the snowball effect for the topics where information was sparse or widespread, i.e., hybrid models and their state-of-the-art today [50]. These approaches were helpful in finding the information that was suitable within the thesis scope.

3.2 Development Phase

This section will describe the development phase, which consisted of a knowledge-building phase, an experimental phase, and lastly, the object library creation, see Figure 3.3. The development phase focused on investigating research question one by developing a framework for hybrid DES and emulation model building.

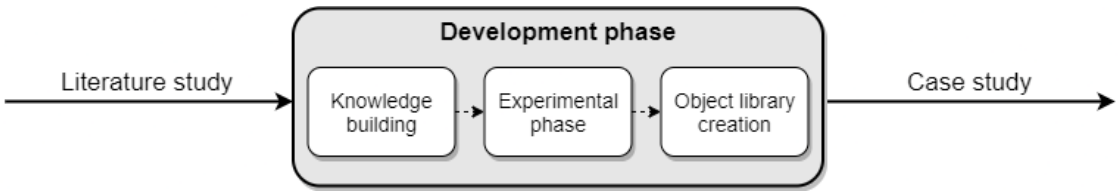


Figure 3.3: The development phase.

3.2.1 Knowledge Building

The knowledge building phase started by building the knowledge foundation to understand the concepts, systems, functions, and programs within this research scope. In the knowledge building phase, as much information as possible regarding the software Siemens Plant Simulation, Siemens SIMIT, the connection between them, and their respective functions was gathered and reviewed. For example, that the executable code in Plant Simulation is coded in "Methods" using the programming language "SimTalk" [35]. "Methods" can be used to create complex and individual

solutions. Also, a Plant Simulation model is created in a "Frame", and those simulation models can be built of several connected frames for modular model building.

The knowledge building phase was a combination of quantitative and qualitative data analysis. The primary sources of information, following the information from the literature study, were Plant Simulation and SIMIT help guides, lab material from Siemens and supervisors. The lab material was acquired for both Plant Simulation and SIMIT. The help guides were read and combined with tests and experiments to familiarize ourselves with each of the software [35], [51]. In this phase, a size-reduced version of the existing emulation model, to be used for the case, in SIMIT was acquired and used for familiarization with SIMIT and initial tests.

3.2.2 Experimental Phase

Following the knowledge building, an experimental study was carried out [52]. The idea was to deepen our knowledge within the software, both in Plant Simulation and SIMIT and their interconnection. The study was done through testing the connection and our modeling ideas of how to best integrate and control Plant Simulation objects with SIMIT signals. An iterative design approach was applied. First, tests and experiments in the software Plant Simulation were formulated and then performed (i.e., retrieving signals from SIMIT into Plant Simulation) to get the connection running. Once the connection was enabled, experiments on how to read and write signals/information to objects in Plant Simulation were carried out, thus learning how to control the Plant Simulation model with signals from SIMIT in the most general way. The experiments were reviewed, evaluated and the process then started again, see Figure 3.4.

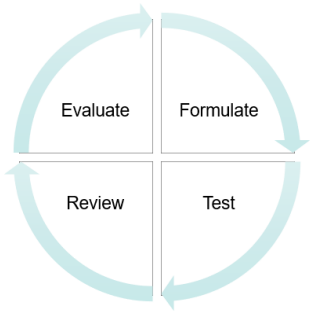


Figure 3.4: The iterative development process.

All tests in this stage consisted of building separate Plant Simulation models for controlling specific objects, such as conveyors, sensors, or machine attributes. Either signals from a scaled-down version of the main emulation model or signals/models created specifically to test a specific type of variable connection, boolean, integer, or real was used. A demo version of SIMIT SP 10.2 was used in this phase, where only 30 bytes of signals were available. Therefore, the best approach was to test specific objects one at a time or in small combinations.

The iterative design process was carried out in combination with further research and support from AFRY and Siemens, following the approach that framework development and experimentation should not be done in one step according to research [53]. A development process should be an iterative process of at least three cycles to integrate the voice of the user fully or to identify the voice of the audience [53]. The integration of feedback should serve as the vital element in model development [53]. Iterative design is connected to user-centered evaluation. The review and evaluation process consisted of sharing modeling solutions with supervisors and engineers working within the area to achieve this user-centered evaluation [53], acquiring input from a company point-of-view on the model requirements and ideas on efficient ways to code the connection solutions in Plant Simulation. The information from the review led to an evaluation of the solution. The input from the engineers and supervisors was combined with further research using literature and help documents. The input was used to further re-formulate and re-design a solution to refine the solutions and document the requirements needed for them to work.

The iterative process in Figure 3.4 continued until no new information or ways to read or write a signal type could be found in Plant Simulation. The solutions that worked well and were within the requirements set by the company, see Appendix B, were documented to be tested and evaluated in later stages. The documentation consisted of preconditions of set solutions, how they worked, and the initial benefits and drawbacks of a specific solution. First versions of stress testing the solutions were also made in this stage to test the dual-functions based on communication between the programs and functionality of a solution.

3.2.3 Object Library Creation

Plant Simulation is an object-oriented simulation software that offers possibilities to create new object libraries with modified objects that can be exported and shared among users. A solution to develop a framework for building hybrid models was, therefore, to create an object library with modified objects prepared for handling the connection with SIMIT. The iterative design used in the experimental process was suitable for developing this software library [53]. Only robust solutions are worthy of further testing, and from the experimental phase, these went onto becoming new library components.

The solutions documented in the experimental phase were thus used to create a new object library. The components used for the hybrid library were derived from the original objects and methods found under the object library in Plant Simulation [35]. The new library could then be exported and loaded into any Plant Simulation model using Plant Simulation version 16.0 or later.

3.3 Case Phase

In this section, the work procedure used in the case part of the project will be described: simulation model building in Plant Simulation, integration of the emulation model (SIMIT), and final tests using virtual controllers to control the simulation model with the emulation model's PLC. Lastly, the evaluation process of the tests is described, see Figure 3.5. The case phase's main focus was to investigate research questions two and three.

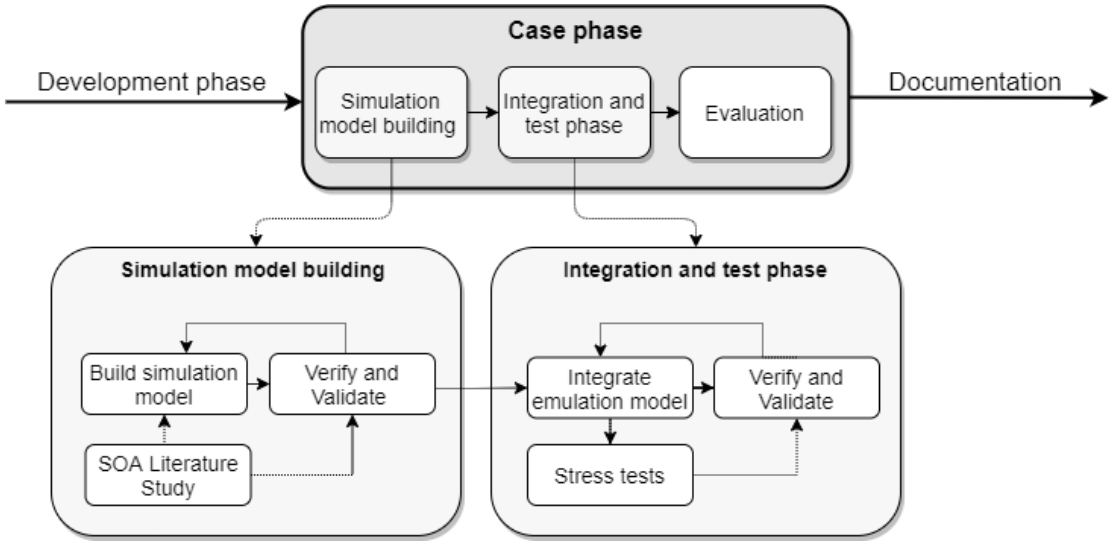


Figure 3.5: The case phase.

3.3.1 Simulation Model Building

The first step in the case phase was building the simulation model in Plant Simulation using the objects from the object library together with the framework developed in the development phase. There are multiple methods developed to carry out DES modeling [54]. They differ in the level of detail, making the available methods range from less to more complex, but the overall structure of the DES modeling methods seems to be pretty similar [54]. A well-known methodology for carrying out a simulation project is Bank's model [55]. Another model that is explicitly designed for simulation and emulation models is Hasan's Hybrid simulation and emulation model (HSEM) [30]. These two models were used to develop a method for carrying out the model building during the case phase, see Figure 3.6. A more in-depth description of the methods can be found in Appendix A. The method in Figure 3.6 was used for building the final simulation model.

The problem definition follows the research questions in this project. Creating the conceptual model was the first step in the model-building process. The conceptual model was created of the specific production line in the software Draw.io, based on the existing model in SIMIT and guidance from a supervisor at AFRY. Since the focus of this case was to test and evaluate the hybrid model framework and software

connection, it did not evaluate the existing system. Therefore, the existing data was analyzed, and where data was missing, assumptions were made together with the team at AFRY.

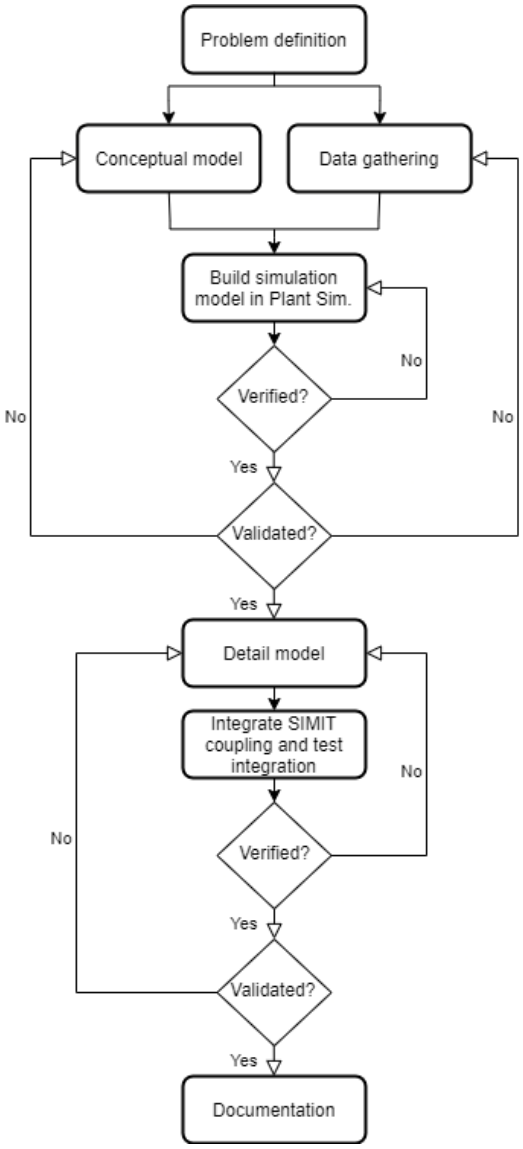


Figure 3.6: Combination of Bank's model and HSEM for simulation model building [30], [55].

Verification and validation were performed iteratively on the simulation model throughout the modeling process, as well as investigating the connection between the simulation model and the emulation model in SIMIT, see Figure 3.6. Verification of the hybrid DES model was performed by comparing its functions to the emulation model, which was a replica of the real system. The model was further face validated in several meetings together with one of the persons who built the emulation model at AFRY [25].

3.3.2 Integration and Test Phase

In the integration test phase, the emulation model in SIMIT was integrated with the simulation model in Plant Simulation. The focus in this phase was to get the connections between the models working and to control the Plant Simulation model with the SIMIT model. The test was used to evaluate how well the object library components and the framework for building a simulation model with dual-functionality functioned.

Entering the integration test phase, the licensed version of SIMIT was delayed, so the integration tests began using an earlier version of the emulation software, SIMIT SP V10.1. SIMIT SP V10.1 was a licensed version of size M. Since the case model was built in version SIMIT V10.2 a copy of the emulation case model with scaled-down functionality was created using our signal standards SIMIT for the initial tests. Since the model had to be rebuilt, limited emulation logic was used and had to be controlled manually during tests. The full-scale model was used when the licensed version of SIMIT 10.2 was acquired, some modifications were still needed in the full-scale model. This model contained graphical components that needed specific licenses, those components were removed. Further, the signals were incorporated in the model and the coupling that would enable software and model connection set up.

The integration and creation of a hybrid case model were tested using virtual controllers to verify the communication and functionalities of the hybrid case model. The virtual controller used the software VMware Workstation to load SIMATIC Manager, which was loaded with the PLC program built for the original system.

3.3.3 Evaluation

The base for the evaluation process was the case model built in the simulation model building phase and adjusted in the integration phase. The model was used to evaluate the framework, the object library, and how the Plant Simulation model and SIMIT connection could potentially affect a model's life cycle. The model was evaluated during the process of building it and in the final integration tests. If the integration and control works, the framework and object library can be seen as successful.

The initial evaluation step looked at the framework as a whole as well as the specifics of the case model. This evaluation was performed through a mix of validation by engineers working with emulation from AFRY and information gathered from integrating the simulation model with the emulation controls in SIMIT. Appropriate changes were made to the object library components and framework if deemed relevant. Quick fixes, or case-specific solutions, were disregarded. The remaining problems and suggestions for improvement retrieved from the engineers working with emulation were documented for further evaluation or marked for future work documentation.

The next step in the evaluation process was running the simulation case model, both in DES mode and in real-time emulation mode, alongside the supervisors to get input on the outcome based on the requirements. A meeting was held with engineers in both the simulation and emulation area, where a presentation of the hybrid model and a discussion about it followed. Feedback from end-users from both a simulation and emulation perspective helped to improve the framework. Based on this feedback, the model and methods were improved and developed further.

The last step in evaluating the hybrid simulation-emulation model included verification using virtual controllers. Virtual controllers enabled the running of the system using the case's original PLC emulation code. The integration with the real code was used to verify and validate how well the hybrid model would work in a real system. The framework could be evaluated based on how well the integration worked.

3.4 Documentation

Documentation is a crucial part of all research projects for external people to be able to follow the work [56], [57]. There is a need to document and explain how the work- and mind-process has evolved during the project for other people to be able to understand, and possibly keep working with, the project as base [56], [57]. Documentation should be done on everything from how to find the relevant literature to how, for example, a programming code is written [58], [56]. For simulation models, it is essential to make comments on everything in the code [57]. Further, it is essential to document how and where findings were made [57].

Documentation should be done in a suitable language and on a suitable detail-level [56]. For simulation software, it is also essential to define suitable variables to ease the understanding [57]. Because a simulation model often takes longer to develop accurately, it must be possible to be understood, updated, re-used, and inherited by others to increase its life cycle [56]. Simulation models are becoming more and more popular within industries. As more and more companies are using simulation, an increasingly competitive factor is the speed of the simulation analysis, and the complexity of the simulation system [57]. According to Svensson and Tehler [57], documentation on simulation models is done to increase the credibility and facilitate validation and verification of the model [57]. Examples of what needs to be documented are the requirements, specifications, plans, expectations, results, and used methods [59]. During the documentation process, these theories were used throughout the development and case phases to create the framework for building and working with the hybrid model concept.

3.5 Reliability and Validity

Reliability and validity are two critical measures for trustworthy research [60]. Reliability represents the stability of the findings and validity of the truthfulness as well as a measure for transparency, and researcher bias [60]. These measurements are essen-

tial when using secondary data, in our case, the literature study, which can enhance the trustworthiness of the research [60]. Reliability and validity can be increased using various methods to collect data or search for information [60], this reduces the risk of biases or interference of tainting the results. These measures need to be discussed and commented on in the documentation to prove the research's integrity and quality. It is essential to be aware of the errors that can occur within measurement and research to minimize the risk of these affecting the work and influencing the results [60]. With this, background verification and validation were integrated into each step of the model building and case tests. This enables the identification of errors before they risk being built into a process [61]. Reliability and validity were also built in the development through its iterative approach, since an iterative process can become self-correcting, moving researchers between design and implementation, running through cycles of data collection, analysis, and development [61].

Because of the increasing complexity of virtual modeling, Balci [58], proposes 20 principles, so-called *golden rules*, for the process of verifying, validating, testing, and certifying simulation models. Three of these rules were used during the thesis work. Golden rule no. 2, model accuracy should be considered on a scale because the virtual model is rarely a replica of reality since there are too many external factors involved in reality [58]. This rule was regarded when models were scaled down during the development phase and to find the appropriate level of detail for the case model. Golden rule no. 3 states that the validity, accuracy, and verity must be for the end-users or the customers. Golden rule no. 7 states that it is the end-users who will be able to say if the model is sufficiently accurate or sufficiently valid for their purposes [58]. Therefore, frequent use of face-validation with simulation and emulation engineers as end-users of the system was performed.

3.6 Ethical Concerns

For findings that another researcher had come up with, he or she always got the credit, using thorough referencing and critical investigation of sources. Regarding own findings, documentation was made on how they were retrieved. Persons and organizations that wished to be anonymous were kept anonymous throughout the report.

The line at the company on which the simulation line was based will not be mentioned in detail to keep any external parties anonymous. Details that are sensitive to companies connected to the project will not be exposed to any extent. AFRY will own all model rights. Sensitive data will be deleted at the end of the project.

4

Results

In this chapter, the results are presented. The first section presents the hybrid model framework that was developed during the development phase, connected to research question one. The second section presents the case phase connected to research question two and research question three.

4.1 Hybrid Model Framework

This section will present the results from the development phase (the hybrid model framework), which is connected to research question 1: "*How can an emulation model control a DES model? What are the prerequisites and methods for building a hybrid DES model with this dual-functionality?*" The framework consists of the requirements for a dual-functioning hybrid DES and emulation model, a name standard, and an object library with emulation prepared simulation objects. See Figure 4.1 for an overview of the framework.

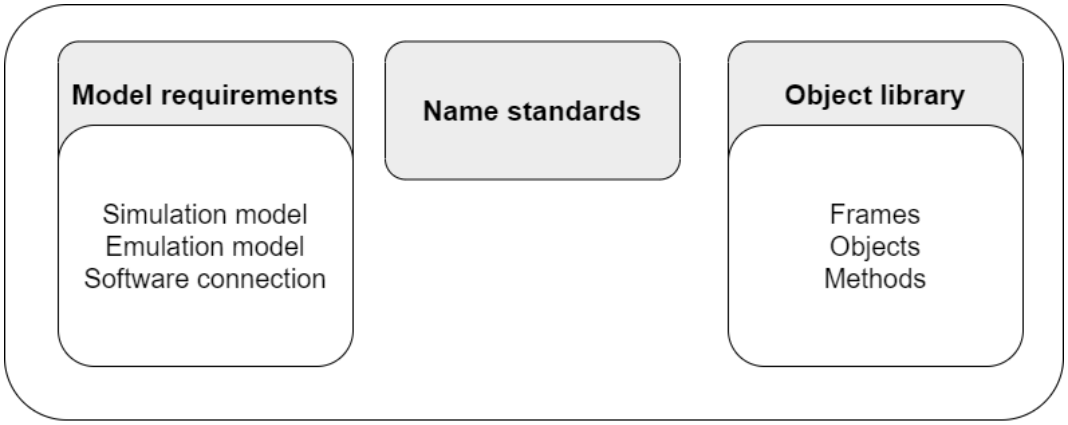


Figure 4.1: Framework content.

The section starts by explaining the requirements for a hybrid model from a simulation, emulation, and software connection point of view, followed by a description of the name standards and the object library. Different integration solutions, test results, best practices, and library components will further be presented.

4.1.1 Model Requirements for Integration

This section will define the specific model and software requirements, such as name standards and the emulation and simulation model requirements, respectively, for the hybrid connection to function. The general requirements for the hybrid concept will also be defined. For the initial requirement list developed together with AFRY used in the development phase, see Appendix B.

In modeling rules, there is a distinct difference between simulation and emulation model building. It is important to note that a simulation model can contain constraints and conditions that approximately mimic functions of a system, while emulations' purpose is to mimic the exact system. The constraints and conditions set for simulation purposes should not have the ability to affect the DES model when emulation control is enabled. Therefore, the general idea for emulation-controlled simulation is that the simulation model should only send and retrieve signals in emulation mode, not control the objects.

An example from the development phase was modeling mechanical constraints, using a system with conveyors in different heights and a lift moving the product from one conveyor to the next, see Figure 4.2. In simulation mode, the lift conditions of when to lift or where the product should stop are coded on the object or method. Here the code should represent reality to the extent needed for the set experiment. In simulation mode, the constraints will stop the product if the lift is in the wrong position or if constraints are set on when and how the product can move. For emulation, this scenario should only have mechanical constraints and code receiving and sending information to the emulation model or PLC. For example, a product should not continue to move if the succeeding conveyor is higher than the current path since that mechanically stops the product, see Figure 4.2. There should not be any conditions coded to stop a product from moving if the path was free in reality. It is up to the emulation model or the PLC to decide the next movement based on the information sent from the simulation model. Finally, in emulation mode there is not allowed to be anything stopping a product or changing its path other than the emulation code, or the PLC, and mechanical constraints reflecting real-world conditions.

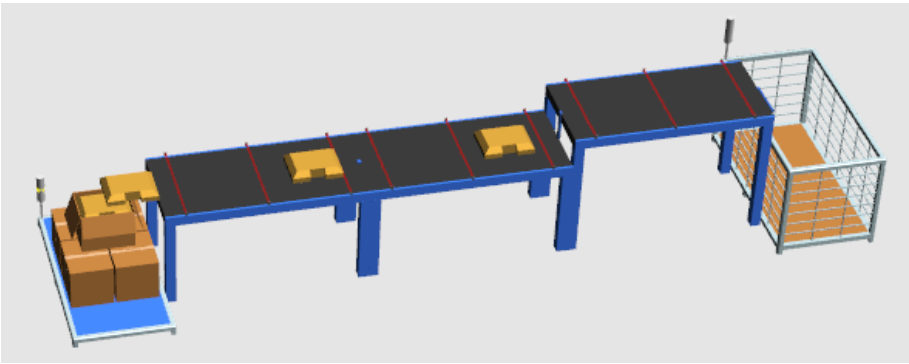


Figure 4.2: Conveyor system with height differences.

To keep the simulation and emulation function separate and thus be able to enable or disable the hybrid function, i.e., choosing between a simulation or emulating the model, a checkbox was developed. The checkbox was used to decide whether the model is in a simulation or emulation mode. Choosing what to emulate can then be done using different simulation frames or using emulation-specific methods in the main frame controlled by the checkbox, these concepts will be described in coming sections.

The general requirement on the software to build a hybrid model is that the DES model needs to be integrated with the emulation model. A connection between the software Plant Simulation and SIMIT is thus needed; Plant Simulation and SIMIT come with this function built-in. The connection uses a SIMIT interface in Plant Simulation and a Shared Memory coupling in SIMIT where the signals that will send and receive information are created. These signals are loaded into the SIMIT interface in Plant Simulation by entering the Shared Memory name and activating the SIMIT interface, see Figure 4.3. For the connection to work, the emulation model controlling the DES model needs to run on the same computer simultaneously as the DES model. The Shared Memory must be named the same in both programs. The coupling of type Shared Memory can be seen as the link between the two programs.

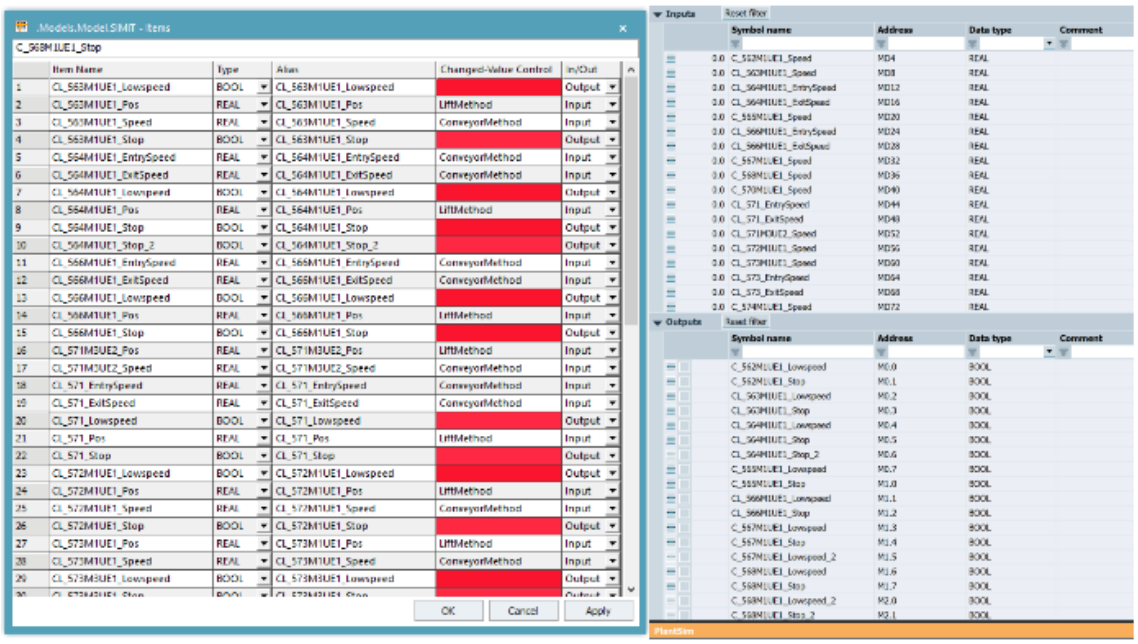


Figure 4.3: The signals in the Shared Memory are shown in Plant Simulation to the left and in the SIMIT coupling to the right.

4.1.1.1 Requirements of SIMIT Models

In SIMIT, a new coupling of type Shared Memory must be created, see "PlantSim" to the left in Figure 4.4. This thesis has used the name "PlantSim" on all Shared Memory connections used in tests, with the intention that an emulation model can have multiple Shared Memory couplings. In this coupling, all signals that should

send or retrieve information from Plant Simulation need to be created and placed, given an address and data type, Figure 4.4. The type of signal created needs to match the information that should be sent or retrieved, signal types can be found in Appendix C. Under properties in this Shared Memory coupling, "Signal description in header" must be checked for Plant Simulation to be able to read the signals, see the bottom of the Figure 4.4.

There are two types of signals in SIMIT, Input signals and Output signals. SIMIT referrers to its signals as Input or Output based on another system's signal direction. Neither the Input nor Output signals must be connected to a SIMIT component for the coupling to work. Plant Simulation will receive the default value, i.e., false for Boolean and 0 for Real, or set value on a signal when signal values are imported or called on in Plant Simulation. For Output signals, the value will be changed in SIMIT if a Plant Simulation code is executed that specifically sets or updates specific signal values.

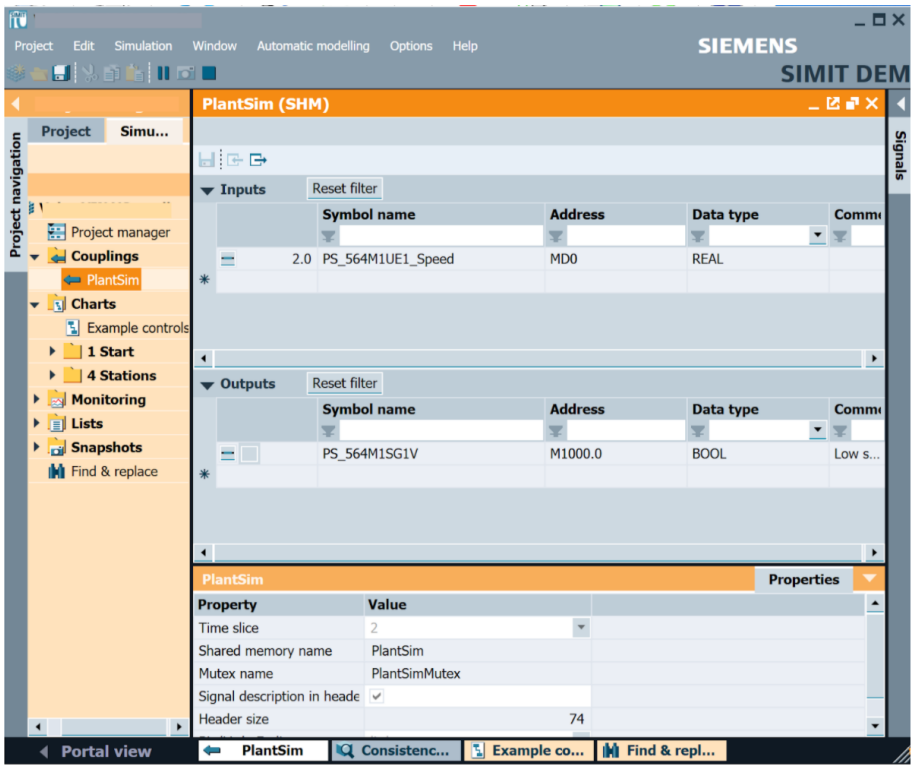


Figure 4.4: The Shared Memory coupling in SIMIT.

When the simulation in SIMIT runs, the background is yellow, see Figure 4.4. For basic test models, emulation logic is not needed, the simulation model can be controlled by manually changing signal values. However, additional logic is beneficial in SIMIT charts to control the signal output or input for more extensive tests without a virtual controller.

4.1.1.2 Requirements of Plant Simulation Models

The main requirement for a hybrid DES model is the required setup of the SIMIT interface and the use of emulation-prepared components presented in Section *4.1.3 Object Library Creation* that enables shifting between simulation and emulation code.

The SIMIT Interface box needs to be added in the frame and is to be connected to the emulation. The interface is placed onto the simulation model "floor". The SIMIT interface is located under "Information Flow". However, it is not a standard library component and can be located under managing the class library.

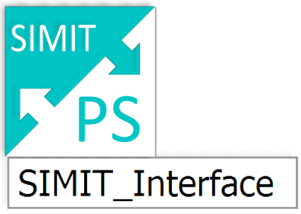


Figure 4.5: SIMIT interface icon in Plant Simulation.

In the interface, see Figure 4.6, the Shared Memory name from SIMIT (the same Shared memory as in the coupling of type Shared Memory in SIMIT) must be written; this is how the model can find the correct SIMIT coupling. When SIMIT runs a simulation, the SIMIT interface box in Plant Simulation can be activated. Setting up the SIMIT interface requires pressing "Apply" after activation, followed by importing signals, done through pressing "Import Items" followed by pressing "Apply" again to make sure that the imported items are saved. Plant Simulation cannot import or update values if the SIMIT interface is deactivated.

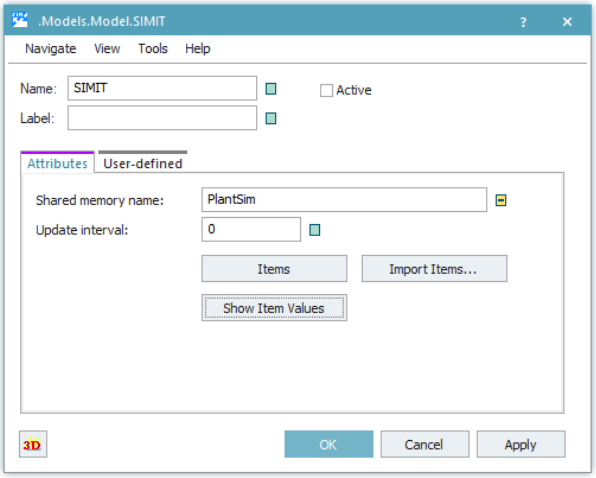


Figure 4.6: The SIMIT interface in Plant Simulation.

When pressing "Items" the imported signals from the Shared Memory coupling can be shown, Figure 4.7. Here an alias needs to be added, which enables code paths to the signal. The Item name by itself cannot be used as a path in Plant Simulation methods. With the developed name standard, the column "Item Name" should be copied and paste under the "Alias" column.

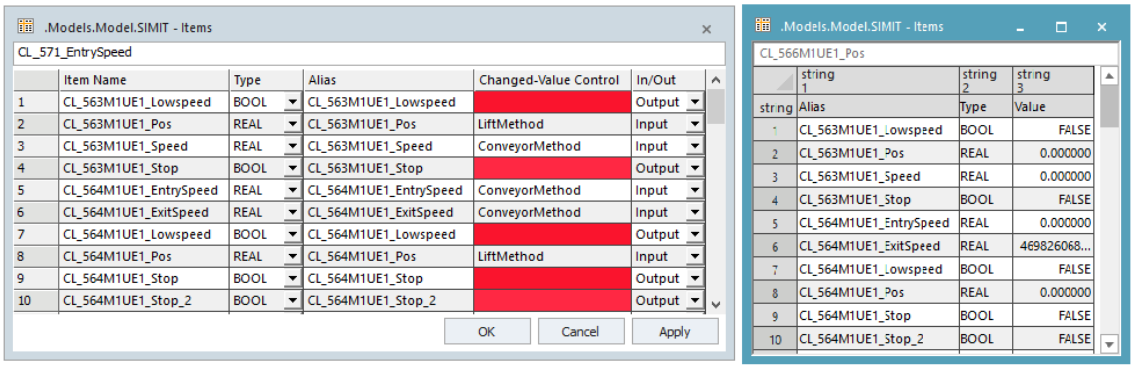


Figure 4.7: The SIMIT interface in Plant Simulation, showing Items in the figure to the left and Item values in the figure to the right.

In the column "Changed-Value Control", the path to, and name of, a method, checkbox, or global variable that should receive the signal must be written, Figure 4.7. When a signal is updated, the Changed-Value control; triggers the corresponding method or updates the value of the corresponding checkbox or global variable. A method retrieves the changed value signal as a parameter which can then be used in the code execution. These methods, checkboxes, or global variables will therefore update their value when a change registers in SIMIT.

In simulation mode any simulation speed can be used. However, to enable real time emulation in Plant Simulation the Event-Controller needs to activate real-time x1, setting the simulation speed to real-time (24h clock), see Figure 4.8.

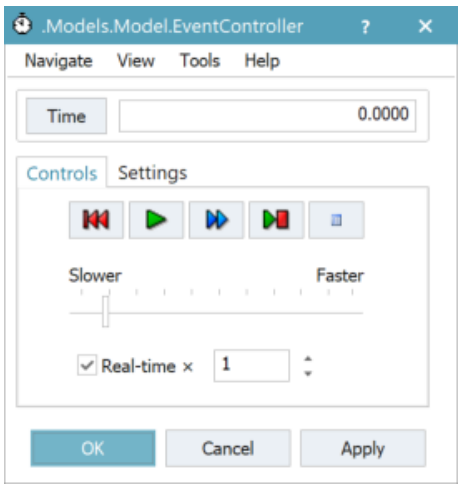


Figure 4.8: The event-controller in Plant Simulation.

4.1.1.3 Requirements on Software Connection

The Changed-Value control column in the Items table in the SIMIT interface, see Figure 4.7, is responsible for "calling" on methods, checkboxes, or global variables in the Plant Simulation model. The Changed-Value control column is only for Input

signals. A change in a SIMIT signal registers as an event, and the signal value is updated and sent to the designated path, see Figure 4.7. The framework developed consists of specific methods created to control object parameters in Plant Simulation. Therefore the methods, checkboxes, or global variables controlling the objects in emulation mode needs to be added as a path under the Changed-Value control. These methods, checkboxes, or global variable paths must be written or dragged to the Changed-Value control box manually. The simulation model will only update the signal values continually if a Changed-Value control path is designated.

We investigated several solutions to generate a list of Changed-Value control paths. First, a general method can be used for every signal, i.e., all speed signals should be controlled by a ConveyorMethod like in Figure 4.7. The ConveyorMethod is then typed once and can be copied and pasted onto all the other relevant rows. Using the same method works for smaller models that only have simple objects controlled by the same method or variable. Secondly, a method, checkbox, or global variable in the model can be dragged by the programmer from the simulation floor to the changed-value control box. Third, a method can be created in Excel where the signal names are pasted in a column from the Items list in Plant Simulation. Then Excel does the math and generates a column of method names which can be pasted into the changed-value control column. See Appendix C for Excel templates and equations. Using the search function in the signal names requires the name standards to be updated to support this function. Fourth, the Item list can be exported as an Excel file which can be edited and then imported in Plant Simulation.

Signal values can also be updated using SimTalk code in Plant Simulation. SimTalk is the programming language of Plant Simulation. The methods of the SIMIT interface are connected to the alias name in the interface by the code in Figure 4.9. Input signals can be retrieved with the GET command, and Output signals can be set using the SET command. The GET code can be used regardless of methods, checkboxes, or global variables called by a Changed-Value control. However, suppose the signal does not have a Changed-Value control path? In that case, the update frequency is set by the update interval in the SIMIT interface, Figure 4.6, i.e., not triggered and update on change in SIMIT. These codes are the only registered way to set values on the signals going out from Plant Simulation to SIMIT, i.e., object locations when passing senors or other values.

4.1.2 Name Standard

As mentioned, the connection between the software is built on the Shared Memory coupling. These signals contain values to or from the emulation model. The objects in the simulation model need to find the matching signal to retrieve its value or set a value on an outgoing signal. This thesis developed a name standard to enable building models that do not need hard-coded paths for each signal and object connection. It will mean that the SIMIT signals that control the simulation model in Plant Simulation must follow a certain name standard to comply with the integration. The same name standard must apply to the objects in the simulation model.

Figure 4.9: Methods of the SIMIT Interface [35].

The standard enables the information in a signal to be directed to the corresponding object. The integration must become as general as possible so that specific code will not be needed for each object and signal. The name standards developed through this phase are explained as follows: The names of the signals must begin with a description of what type of an object the signal should control, e.g., a conveyor or a lift, see 4.1. The part of the name followed is the object ID/object name, and the final part presents what parameter the signal controls, e.g., speed or capacity, see Table 4.2. Therefore, the name standard is constructed as: "[Prefix Indicating the Object Type]_[The Object Name You Want]_[Suffix Indicating the Changing Parameter].", see Table 4.1, and 4.2 for examples of prefix and suffix standards.

Prefix	**Description**
C_	Conveyor
CL_	Lift with conveyor (where you can set speed)
L_	Lift without a horizontal speed

Table 4.1: Example of the prefix name standards for signals and objects.

Suffix	**Description**
_Speed	Horizontal speed of conveyor/lift
_EntrySpeed	Entrance speed of angled conveyor/lift
_ExitSpeed	Exit speed of angled conveyor/lift
_Baseheight	Height of conveyor/lift
_Position	Position (height in meter)
_Acceleration	Acceleration
_Capacity	Capacity (number of MUs on the conveyor)

Table 4.2: Example of the suffix name standards for signals and objects.

The object's name should follow project-specific standards. The suffix of the object must correspond to one of the parameters existing on the object in Plant Simulation, either by predefined attributes or a user-specific created attribute. For the

framework, predefined attributes such as speed for conveyors or entry speed and exit speed for angled conveyors have been created. However, the standard should be evaluated and adjusted at the start of any project to cover the signals needed (see Table 4.2).

4.1.3 Object Library Creation

A new object library was created in Plant Simulation consisting of new objects, methods, and frames prepared for emulation control. This library was created based on the model requirements described in Section *4.1.1 Model Requirements for Integration*. The following sections will describe the contents of the new library with corresponding examples. For a more extensive description of the library contents, see Appendix E.

With the object-oriented design of Plant Simulation, an object library can be created containing modified objects, duplicated from the original standard library, see Figure 4.10. The first step is to create a folder under the class library where the modified objects are placed. To create the library, right-click on the folder in the class library, and under "Make library", add the information, and save. The library will be saved as a .lib file that can be used in other models by loading the library under general preferences in main menu after creating a new model.

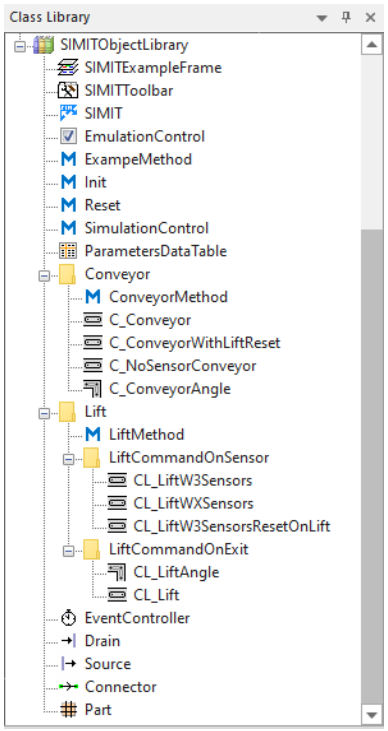


Figure 4.10: The object library containing modified objects.

The objects in the final library were modified with methods and attributes that would enable signals from SIMIT to control the objects with limited to no coding

needed for the emulation mode. The creation of an object library means that these objects will always come with the pre-set method and attributes created on them. This library was saved and exported, enabling other users to use this library in their simulation models. The objects in the new library consisted mainly of the objects necessary to build the case model, creating a base framework for the hybrid model concept. The library consists of more than one version of an object for solutions that needed to be tested and evaluated in the next phase.

Methods that Changed-Value control should control must be placed on the simulation floor. Changed-value controlled signals can not send data to methods created on a specific object. These "on-self" methods can minimize the number of methods on the simulation floor, but it is essential to remember that SIMIT cannot call these. Another benefit of on-self methods, except for keeping the simulation floor clean, is that the objects can be created directly from the library containing all required attributes and methods. Methods that are not on-self must be dragged out separately to the simulation floor.

4.1.3.1 Frames

Frames can be used if, for example, the production line is extensive or if there is a part of the production line that is not interesting enough to have on the main frame or to divide the model into different frames to get a clearer picture. A frame can contain either one object, two objects, or even an entire production line. Frames are often used when building large models, modeling a part of an extensive production line or factory in each frame, making it possible to get a better overview. Frames can also be used if several programmers work on the same model because they can work simultaneously in different frames.

If emulation is preferred in several frames, a SIMIT interface box can be placed either in each frame or only in the main frame, and then the signals needs to be passed on to the right frame, see Figure 4.16. Passing the data onto different frames is done the same way as passing data on to other methods or objects, except that the path to the frame must be included before the receiving method. Using different SIMIT interfaces in each frame will not require the same coding to a new frame. It works like previous examples where the interface is located in the main frame. For example, a frame can be seen as a house with people in it. All people in the house can easily talk to each other, or give each other gifts. If someone in the house wants to give a person in another house a gift, they must first get to the new address.

4.1.3.2 Objects

Among the new objects created for the library, all objects were based on the already existing objects in Plant Simulation. Detailed description of the objects, their variants and figures of their code, see Appendix E. The conveyor must be named correctly while installing it; it must follow the set name standards presented in Section *4.1.2 Name-standards*. The first object created was a conveyor with three corresponding sensors attached. The conveyor can be viewed in Figure 4.11.

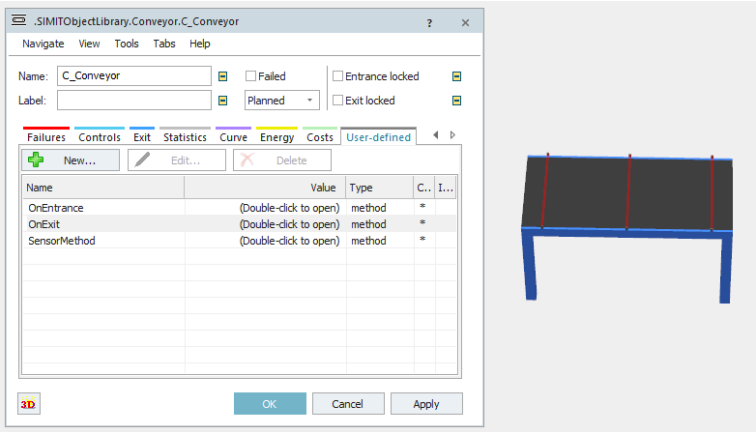


Figure 4.11: Straight conveyor.

The object was set to be a basic conveyor component containing the most basic User-Defined attributes; a method for entrance control, a method for exit control containing a mechanical constraint, and a method for sensor control. If it would have been desirable to have more than three sensors, it would be simple to add a new sensor, connect it to the existing sensor method, and add the extra logic in the method.

The second object that was created was an angled conveyor. The pre-defined angled conveyors in Plant Simulation are not compatible with sensors. Because sensors could not be added, the programming code to send values of sensors to SIMIT was written on "On-Entrance" or "On-Exit" controls. These codes are triggered when a part enters or exits the conveyor, respectively. Therefore, the angled conveyor looks the same as the original angled conveyor, but it contains new attributes and on-self methods. An important thing to remember regarding the angled conveyors is that they, unlike a straight conveyor, have two different speeds, one entrance speed, and one exit speed. This is because the angled conveyor is originally based on two straight conveyors but with new logic and graphics. See Figure 4.12.

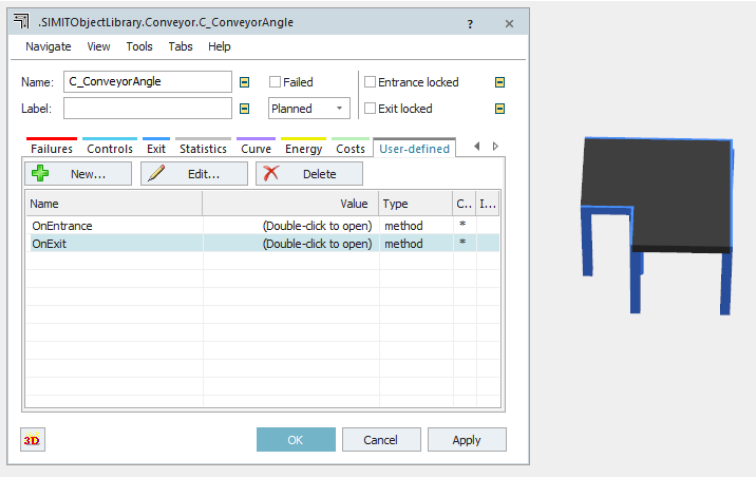


Figure 4.12: Angled conveyor.

The third object that was a straight lift. This was simply a developed version of the conveyor with three sensors but with extra logic. For simulation purposes, the lifts have some extra User-Defined attributes values such as start position, lift method, end position, lift increment in meters, and lift increment in time to simulate a continuous lift sequence. A straight lift can be seen in Figure 4.13.

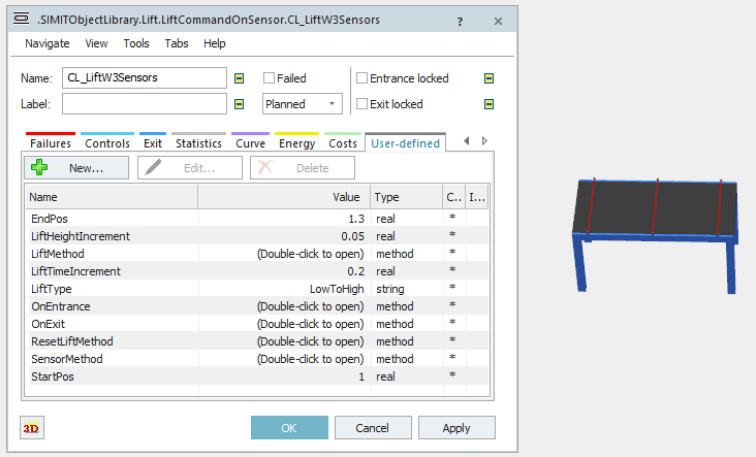


Figure 4.13: Straight lift.

After the straight lift was constructed, angled lifts were also constructed. These were created based on the angled conveyors but with similar logic as the straight lift. The raising/lowering of an angled conveyor in simulation mode was coded in the same way as a straight lift because the table only has one height, unlike the split speed. The graphics of an angled lift is just like the angled conveyor, see Figure 4.14.

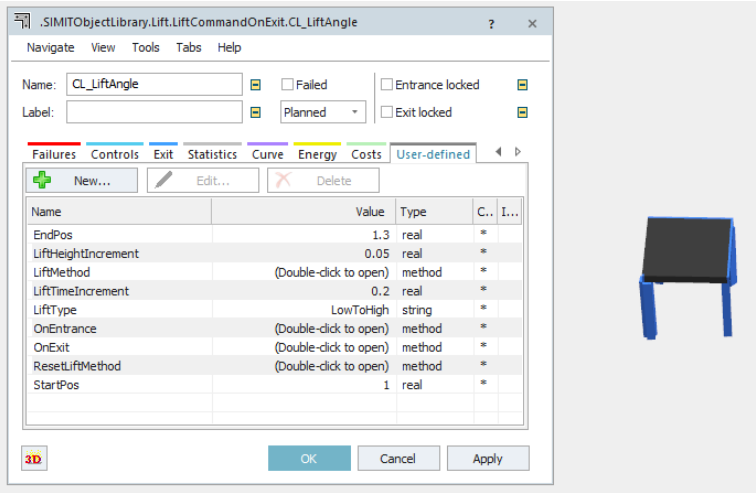


Figure 4.14: Angled lift.

Some variants of the lifts were constructed in this phase regarding the programming. The most suitable and sustainable solutions were chosen in a later stage of the project.

4.1.3.3 Methods for Emulation Control

An initial-run method and a reset-run method was created, from now on called the init- or reset method. The reset method is simply used to change the required parameters regarding simulation/emulation mode. If the "EmulationControl" button is false, the reset method deactivates the SIMIT interface to make the model ready for a simulation run. If the button is true, the reset method sets the SIMIT interface to Active to prepare the model for an emulation run, see Figure 4.15.

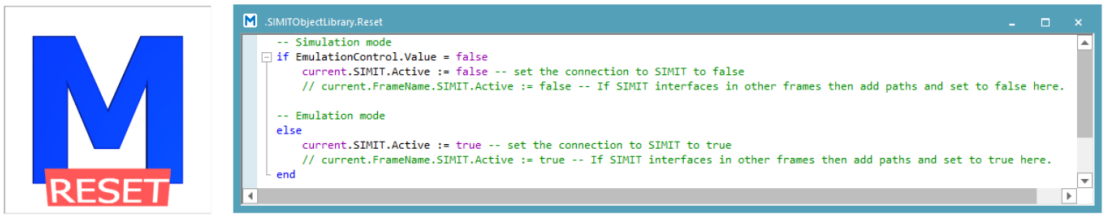


Figure 4.15: The reset method.

Depending on whether Plant Simulation is running in an emulation or simulation mode, old parameter values are saved, like speed and height. It is not desirable to keep the simulation values in an emulation model or the emulation values in the simulation model. Thus the init method was created to reset all parameters, see Figure 4.16. How the values were reset depended on the simulation/emulation mode. If a simulation run is about to happen, speeds were set from a predefined data table using a method reading this data table. If an emulation run is about to happen, the values are retrieved from SIMIT automatically when the SIMIT interface is activated. In emulation mode the init can also be used to reset sensor values, to guarantee that no sensors are active in an empty system.

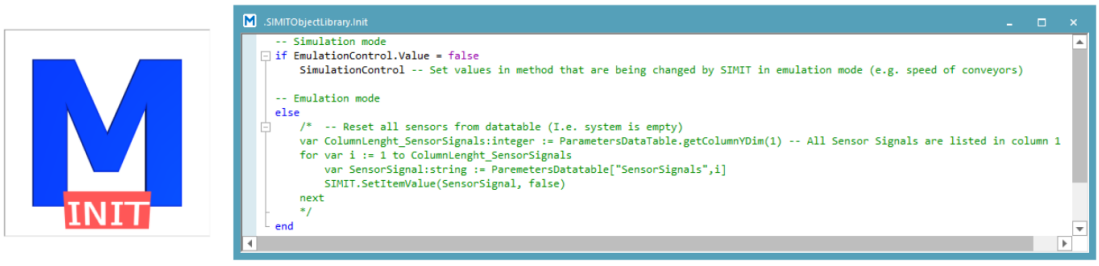


Figure 4.16: The init method.

When switching between simulation mode and emulation mode, the values must be reset to adjust to the mode. Therefore, a simulation control method was created, which updates the values that change during emulation (by changed value control) when entering the simulation mode. The object names and initial values are stored in a table, read by the method, and the object's value is also updated in the method. The simulation control method could also read the values from the table and pass it on to the method "ConveyorMethod", which updates the values in emulation mode.

This requires one extra step, though, when the values are sent from a table via a method to another method.

General methods to control different parameters of objects placed in the same frame were created for the main frame. This enabled the methods to be added under Changed-Value control, and values to be updated automatically on change in SIMIT. The possible parameters must be defined using "param" at the beginning of the code, and the objects must have the same name as the signal (but without the parameter definition). Otherwise, the code can set new parameters to any object in the actual frame. The method is built by the code in Figure 4.17.

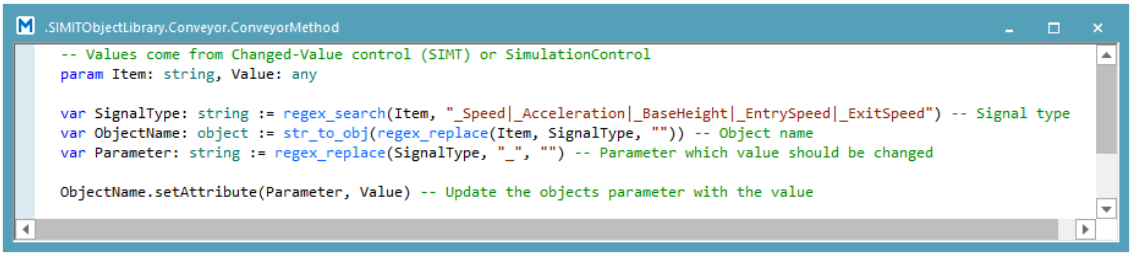


Figure 4.17: The "ConveyorMethod" used for controlling conveyor parameters, for detailed description see Appendix E.

The two general methods that were created were "ConveyorMethod" and "Lift-Method". In "ConveyorMethod", all signals regarding speed, acceleration, capacity, etc., are received, see Figure 4.17. In the method, the signal and its value are retrieved. The method later passes the value of the signal onto the corresponding object in the simulation. The second method, the "LiftMethod", is presented in Figure 4.18. The "LiftMethod" changes the height of the corresponding lift already in the method instead of passing the data on.

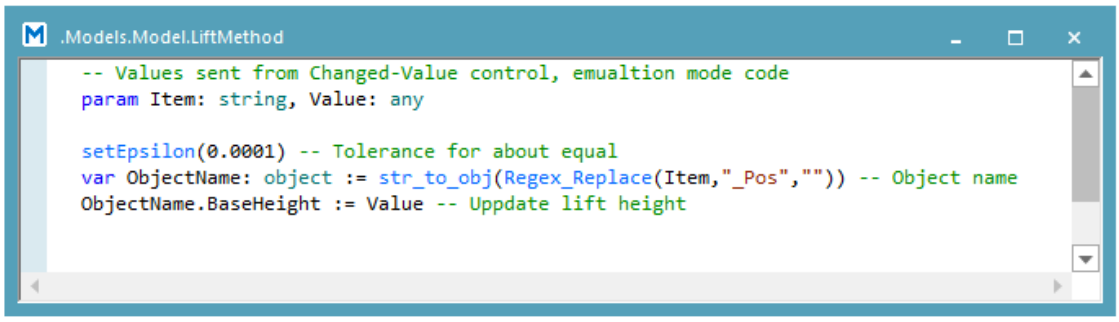


Figure 4.18: The "LiftMethod" used for controlling lift parameters, for detailed description see Appendix E.

Another method was created to control what happens when a products reaches certain positions, simulated as passing by a sensor on conveyors within the system. The method is presented in Figure 4.19. The first goal was to send a signal to SIMIT

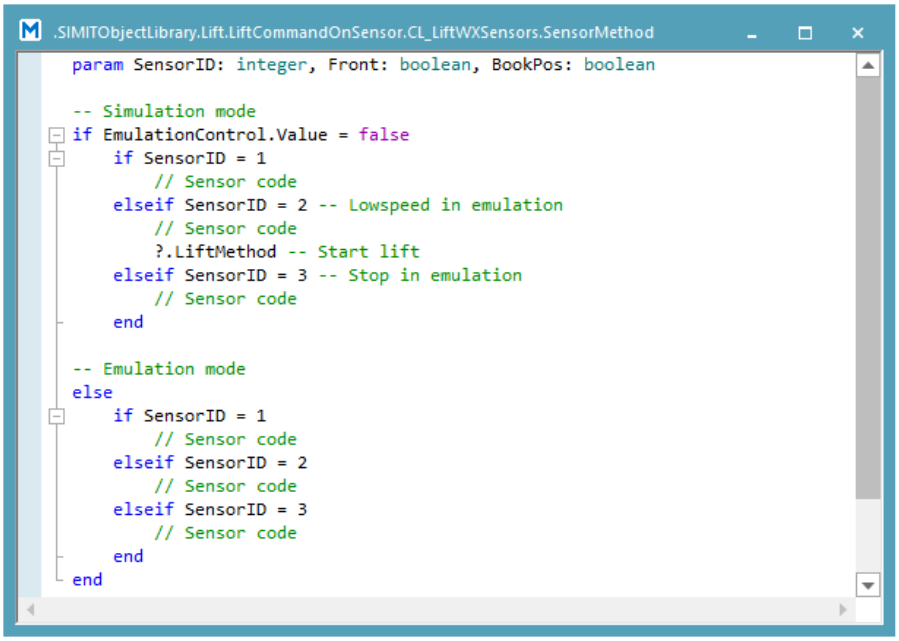


Figure 4.19: Sensor methods used to take actions when reaching a certain position on a conveyor, for code with examples, see Appendix E.

when the product reaches the different sensors. Therefore, the method sends a signal to SIMIT when a product in Plant Simulation reaches the sensor. It is then up to SIMIT to decide on the actions depending on the signal. If the product reaches sensor two, then a low-speed should be set in. SIMIT then changes the speed value, and the model in Plant Simulation will update its values, in this case, the speed of the conveyor.

All on-self methods begin with an if-statement to check if the simulation runs in a simulation mode or an emulation mode. Depending on the mode, different logic should set in. During a simulation run, standard simulation logic should be coded. However, during an emulation run, the logic should only follow the signals and logic from the emulation program slavishly. A coding example in simulation mode is that a product could stop when reaching the end position of a conveyor when the coming conveyor or lift is in a lower position. However, this should not be set by the simulation code in emulation mode but by the emulation code. A sensor should send a signal to the emulation software telling the emulation code that the product is in place, then the emulation controllers should decide to stop the conveyor. For more details of the methods development, see Appendix E.

4.1.4 Preliminary Tests of Hybrid Model Framework

Preliminary tests were carried out to evaluate, verify, and validate the individual results from the development phase. First, tests were done in simulation mode, with a simple simulation demo-model, testing library component's functionality, see Figure 4.20. Secondly, tests were carried out in emulation mode, using basic

emulation components in SIMIT to control objects in the Plant Simulation demo-model. For conveyors a simple slider in SIMIT was used sending values of the type Real to set the conveyor speed, and for the lift a lift component sending position data in meter. Following the integration steps and continuously detecting errors, a connection between Plant Simulation and SIMIT could be made and the test results seemed optimistic. During the tests, problems regarding some technical details i.e. code and their solutions are presented in Appendix D. Face validation was performed with the team from AFRY working with SIMIT, Plant Simulation, and PLC, evaluating the functionalities of the library components and the programming codes. Minor adjustments were made to the library components if they did not work as expected or to make their code more general. Best practices from the evaluation were documented and will be presented in this section.

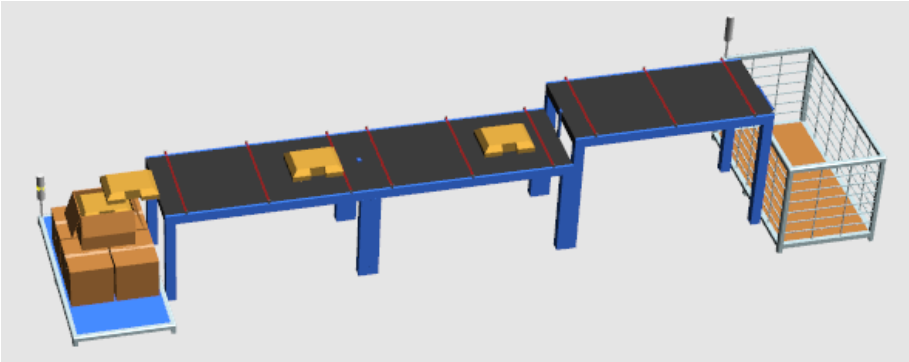


Figure 4.20: The demo version of the production line consisting of two conveyors with sensors and a lift in between.

4.1.4.1 Name Standards

Early tests showed that name standards enable transitioning from a pure DES model to a hybrid model in a few steps. The name standard was a backbone for the requirements of the general methods, and the name standard developed as the methods were tested to adjust to the needs of the general methods. The name standards were tested by continuous discussion with AFRY, evaluating the standard from both simulation and emulation building prospective.

4.1.4.2 Object Library

Preliminary tests showed great results transitioning the predefined straight conveyors with attached sensors and additional logic to emulation mode. It was discovered early that the predefined angled conveyors (conveyor corners) could not have any sensors attached. The angled conveyors had to have the corresponding sensor code on entrance-, or exit controllers. From a simulation perspective it was said to be OK to not have sensors on the conveyor as long as the model can work as intended. From an emulation perspective the team from AFRY were of a similar opinion but more skeptical because the functions of the sensors may be different from that of the real system, the sensors cannot be position at the exact place as in the real world. However, it was decided to keep this version of the angled conveyor, because angled

conveyor enables the product to always be oriented in the same way, only changing direction, "north kept pointing north". They were to be evaluated in later phases. Experimental ideas on other solutions for the function of angled conveyors can be seen in Appendix E.

Preliminary tests of the lift components showed that there was a need to code for a stop and wait during the lifting in simulation mode. For a continuously moving lift component, the lift sequence must be done in steps in a Plant Simulation model because the simulation command to set height immediately changed the height from a to b. Therefore a set incremental step was required. If the product on the lift had not gotten the command to stop and wait during the incremental lifting, the part would keep moving during the lifting session. See Appendix E for the simulation lift code. Different incremental steps were tested, showing that a calculation of the size of the steps must be made to move the lift to the right height. The smaller the steps, the more alike a continuous movement and the closer to the end-value it gets. However, in emulation mode, the emulation model controls the increment and the height; thus, there was no problem stopping the product during the lift or finding the right end-height, while moving the lift in a continuous way. The lift principle thus seemed to work well in both simulation and emulation mode.

4.1.4.3 Hybrid Function from a Simulation Perspective

Potential problems that were found when evaluating the theory of the hybrid framework in that the coding could be a problem if simulation engineers must write emulation code for a system that is not operational yet, i.e., not knowing where to position sensors, or if the simulation model is built without programmers knowing if it should be prepared for emulation since the simulation model often is created first. Using the name standards described in earlier sections and the if statement to control whether to run in simulation or emulation mode seemed to be a suitable alternative for a flexible simulation model. A programmer would then not need to know what parts to emulate at the beginning of the model building, as long as all simulation logic is written under the if statement.

When SIMIT sends signals to Plant Simulation that have a path connected to the signals Changed-Value control, they are received to Plant Simulation in chronological order, i.e., if a value is changed rapidly five times, Plant Simulation will receive five values and will change parameters accordingly five times. In the test model no direct delays were recorded, but for larger systems with multiple variables changing at the same time delays could potentially appear. Larger systems testing is something that needs to be tested further.

4.2 Case Phase

The case phase used a case based test to evaluate the hybrid model framework with the aim to investigate research question two and three; *"What are the benefits and drawback of using a DES model with dual functionality compared to using*

two separate models for simulation and emulation when it comes to virtual commissioning and the creation of a digital twin?", and *"How can a DES model with dual-functionality extend the life cycle of a simulation model?"*.

Since the results from the development phase aimed to create a general framework, the objects used in the case model were modified to the project specific requirements. Therefore, the case specific objects were not added to the object library, only minor modifications were done to the library components if they benefited the framework as a whole. Unique case solutions will be handled in this section. The results regarding changes of methods or procedures to adapt the general framework to a specific case to test the hybrid function using a virtual PLC controller of the actual case system, will further be presented.

In this section, the case phase will be presented in chronological order, starting with the building of the simulation model using the hybrid model framework, adapting and integrating the emulation model, and testing the hybrid model using a virtual controller. Lastly, the results connected to the impacts that a hybrid model can have on the simulation model life cycle are presented.

4.2.1 Building Plant Simulation Model

In this section the building of the Plant Simulation model, using the framework and object library described in earlier sections, are described. Due to the small production line in the case model, frames were not used and their compatibility with the hybrid framework was not evaluated. The hybrid model built in Plant Simulation can be seen in Figure 4.21. All objects created in the case model were named according to the name standard, see Section *4.1.2 Name Standard*.

For this case, straight conveyors, straight lifts, and angled lifts were required. The mechanical stop encoded on all conveyors and lifts, to check if the following station is higher than the current station, was required for the case model and was therefore kept. The basic rule for the conveyors was that at the second sensor, a low-speed signal should be set, and at the third sensor, a stop signal should be set. During the time a product pass by a sensor, for each sensor, the sensor should be activated (set to true), and as soon as the product is out of reach for the sensor, it should be deactivated (set to false). In emulation mode this was coded using the command SIMIT.setItemValue in the product's sensor method. The low-speed signal and the stop signal were the only signals sent from Plant Simulation to SIMIT.

There were also conveyors and lifts required without sensors, as there were conveyors in the system without any sensors. For these conveyors the library object conveyor without sensors was used. On some lifts a second stop sensor was required to make sure the product was in the correct position, in addition to the two general sensors (low-speed and stop), i.e., one sensor on the right side of the conveyor and another on the left. In this case this was solved by sending the same value to both stop signals, i.e., one sensor in Plant Simulation representing two signals. However,

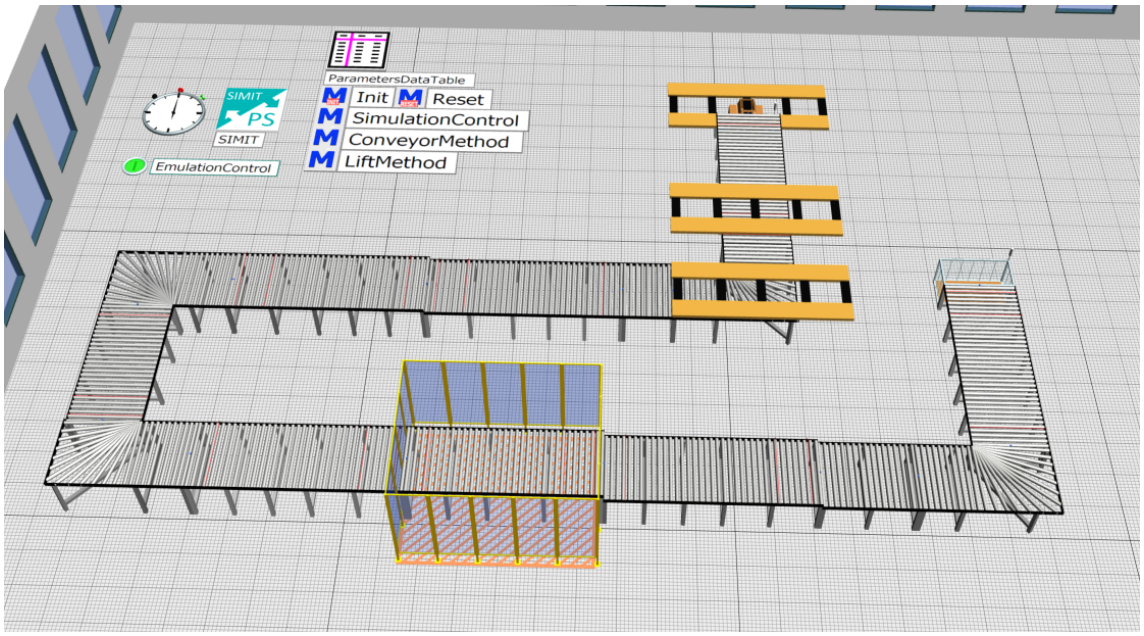


Figure 4.21: The production line in Plant Simulation.

common for all lifts was that they had some signal to check if a product is on the lift or not. Since angled lifts can not have sensors, because they are derived from angled conveyors, the low-speed signal was coded on a front triggered entrance, and the stop signal was coded on a rear triggered entrance. The product was equally broad as the entrance length of the angled conveyor. Therefore the product was graphically placed in the corner of the angled lift when its rear touched the lift's entrance.

The signals to be sent from SIMIT to Plant Simulation were conveyor speed, angled conveyor entrance-, and exit speeds, and lift positions. Switching between simulation mode and emulation mode is done with the init method from the object library which reset necessary signals; sensors, speeds, and heights. All reset data was stored in the library component, "ParametersDataTable". Angled conveyors have, as mentioned, one conveyor towards the corner and another conveyor out from the corner thus two motors and speed parameters. This represents the same function as the real system, where corners have two motors. There is a third motor controlling the vertical changes of the lift for the lifting and lowering of the lift. SIMIT continuously sends the updated height when raising or lowering the lift since the signal is connected to a Changed-Value control path, Plant Simulation reads and sets the new height to the lift continually as the values are coming to Plant Simulation. In SIMIT, this height is changed continuously, while in Plant, it is discretely changed as steps of the values the model retrieves.

The lifts in the first and last corner in the model are in reality divided into two parts retrieving different signals, one lifting the front of the object simultaneously as the other lifts the back of the object. Therefore, an angled lift was combined with a straight lift in our model so that the two objects would work as a two-parted

lift and so that they could receive different signals. This required a particular case solution and coding in both the simulation code and emulation code, as the same "lift" signal should reach both lifts for them to move simultaneously.

4.2.2 Model Integration

The emulation model built by AFRY in SIMIT was constructed with conveyor parts in the control chart requiring a so-called CONTEC license which the thesis did not have access to. Therefore, the control chart of the emulation model in SIMIT needed to be reconstructed for the integration, see Figure 4.22. The Shared Memory was set up using the name standard, see Section *4.1.2. Name-standards*, and signals connected to respective component which should retrieve or send information to Plant Simulation, Figure 4.23. The integration tests were done incrementally, first by simply connecting Plant Simulation with SIMIT, establishing a successful connection where data could be retrieved and sent. Once the signals were loaded into the SIMIT interface, the Changed-Value control in Plant Simulation was set up using Excel templates, see Appendix D. The second step included controlling the speed of the first conveyor in Plant Simulation from SIMIT to verify that it received the right value, followed by the second conveyor, and so on until all objects had been verified. After that, the sensors' functions were tested for each object. Making sure signals were sent from either SIMIT or Plant Simulation and received in the other. Sensors were graphically modeled as light bulbs in SIMIT, so when a product reached a sensor, the light bulb lit up, see Figure 4.22.

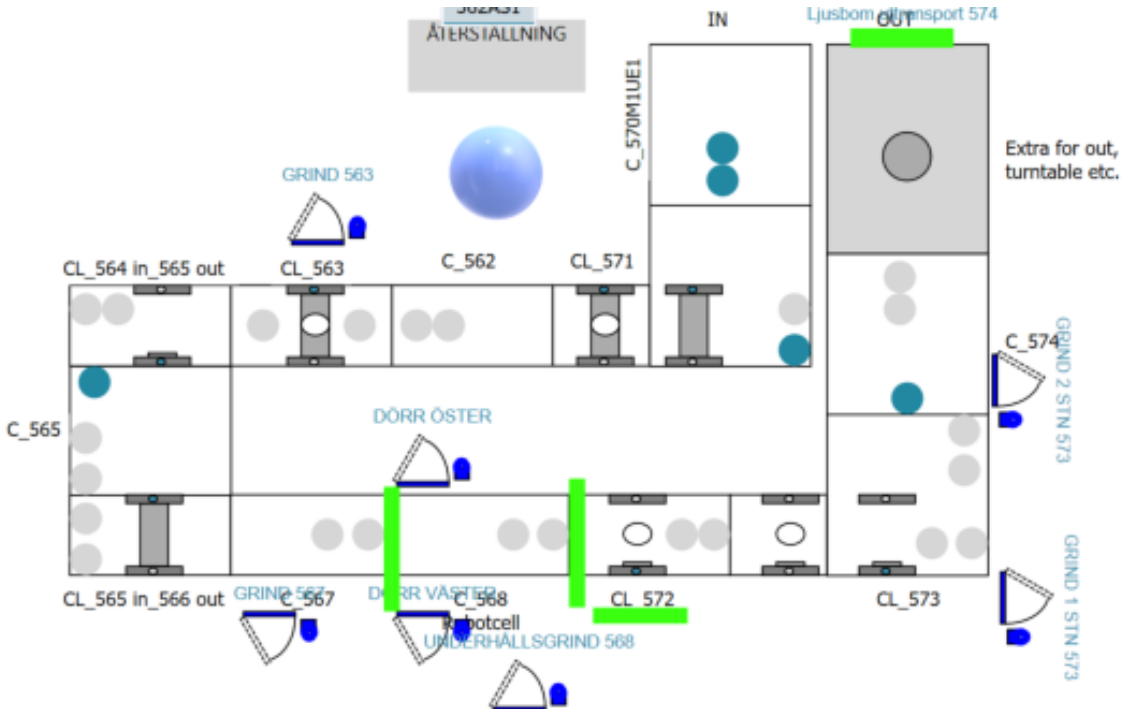


Figure 4.22: The SIMIT interface of the production line.

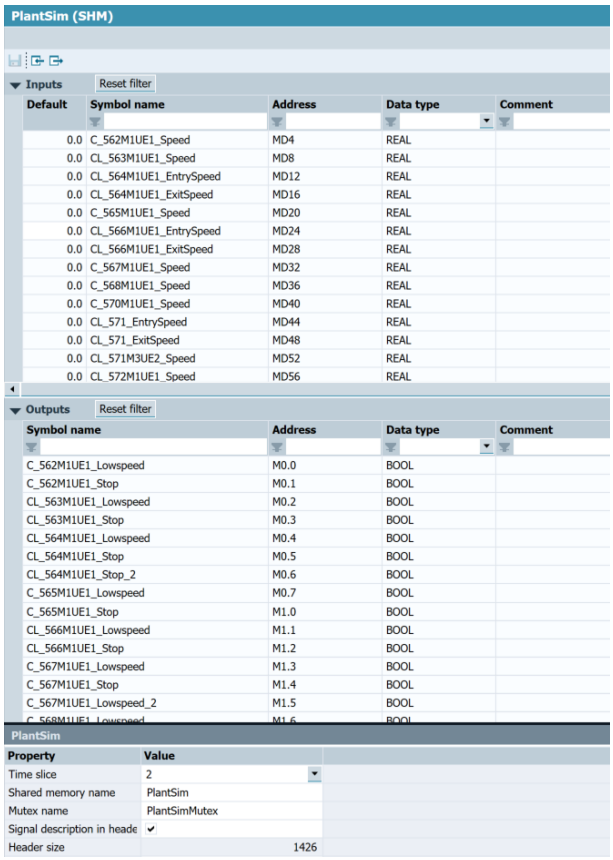


Figure 4.23: The Shared Memory in SIMIT containing the input and output signals of the system.

The changes that were required in the simulation model to integrate the models were mainly re-positioning sensor code so that the information was sent to the right place. SIMIT and Plant Simulation models communicated well when controlling the Plant model by SIMIT signals manually. No direct signal delay was noted at this stage. The model integration could thus be seen as successful using the steps in the framework developed during this thesis. Further, since communication between SIMIT and PLC works fine, theoretically, it should work to control the Plant Simulation model with PLC via SIMIT.

4.2.3 Final Test of Hybrid Model Framework

The final test consisted of controlling the hybrid model in emulation mode with the real PLC using a virtual controller and the emulation model in SIMIT. This was done in two steps, first using the manual mode in the Human Machine Interface (HMI) to start up and resolve any errors, and second setting the HMI to auto and letting it fully control the hybrid model. The HMI in the PLC looks like a map over the production line, physically viewed from the location it is placed/where one stands watching the line. Therefore, the HMI rotates between different locations, showing the same system from different views. The HMI in the PLC controller was programmed in WinCC Flexible Advanced and tested in SIMATIC Manager, see

Figure 4.24. When loaded and installed, the virtual controller appears correctly on the screen loaded with the PLC program. The HMI of the system was divided into three zones, the test was carried out on the first zone of the real system, which represents the line from when the materials come into the model to the box representing a robot cell, see Figure 4.21. The changes needed in the PLC program for the PLC to control the Hybrid model must be updated in the SIMATIC Manager in the Virtual Controller.

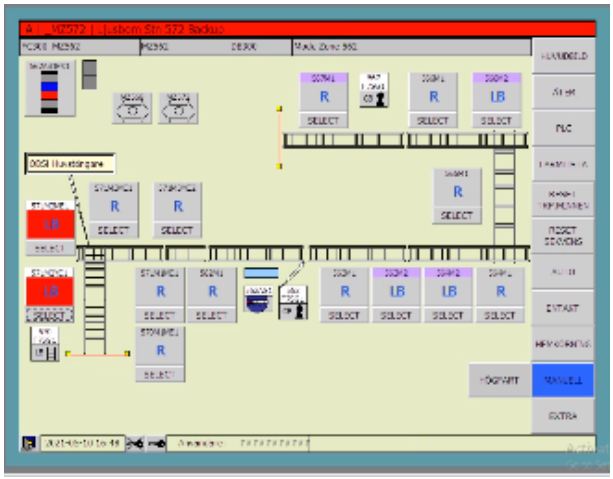


Figure 4.24: A figure of the HMI in the virtual controller. It is blurred to increase anonymity.

Warnings and errors appeared initially that had to be managed. One of the significant errors occurring during the manual PLC run was the negative speed that represented a conveyor moving backward. Plant Simulation is not compatible with negative speeds, and a product can not stop and go backward on a conveyor, for that to happen the direction of the conveyor needs to be changed in Plant Simulation, not its speed. This was solved by making the main direction a positive speed for all conveyors and lifts. If the system would require something to move backward, Plant Simulation will stop the simulation. With the positive speed, Plant Simulation moved the product forward correctly. The low-speed and stop functions worked as intended. When a product in Plant Simulation sends sensor info to SIMIT, the PLC sets a signal to low-speed or stops the conveyor, then both SIMIT and Plant Simulation stopped the conveyor by reducing the speed to zero.

The two parted lifts, which were a particular case solution, were not compatible with the way the sensors were placed in the hybrid model because sensors could not be placed on angled conveyors. The PLC expects a low-speed signal rather quickly after a stop signal. This was experimented with by moving the stop signal in the Plant Simulation code to on-entrance control instead of on-exit control and back until the model sent the signals correctly. This also enabled both the low-speed signal and the stop signal to be activated for the lift to move vertically in the PLC.

Other things that occurred were that the lifts in the HMI sometimes had inverted "set" (end) position or "reset" (start) position compared to the SIMIT model. This

was solved in the SIMIT model so that it matched the PLC. The mechanical stop was a problem during the integration. Since the mechanical stop does not send any signals to SIMIT, thus to the virtual controller, the PLC does not understand that the product is stuck. For the sake of the tests, the mechanical stops were temporarily removed from the code. The mechanical stop is something that should be investigated further. When the errors in the PLC were corrected the HMI was set to auto. The PLC successfully controlled the simulation model through the SIMIT model when it was running in emulation mode. The sensor lit up correctly in SIMIT once an object activated it in Plant Simulation, and the PLC sent signals that slowed down the conveyors at low-speed, stopping them at stop, and moving the lifts. Figure 4.25 shows a picture from the video recording of the emulation controlled simulation model using a virtual controller for the PLC.

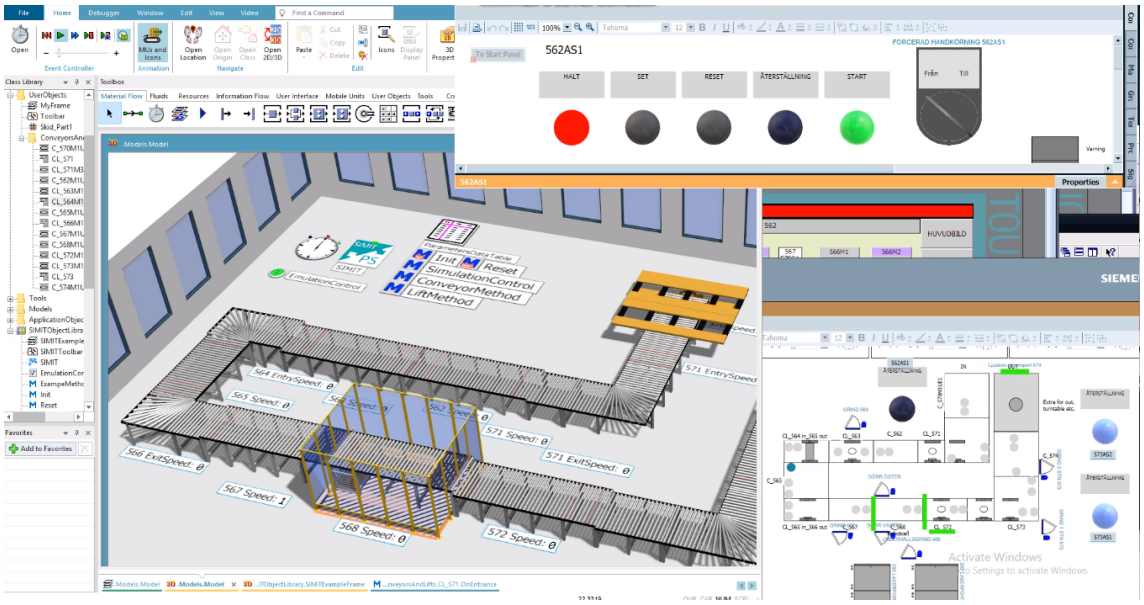


Figure 4.25: The different components in the final test. To the left; the Plant Simulation model, upper right corner; the SIMIT controllers, lower left corner; the SIMIT chart.

The entire test was done with real-time x1 in the event controller in Plant Simulation. Running the simulation model at speeds other than real-time x1 would not enable a real-time simulation connection to SIMIT. When tests were done with real-time times other values than 1, the communication was too slow between the systems, and delays and errors occurred. If the simulation model is not set to run in real-time simultaneously as the emulation model is running, which always runs in real-time, the hybrid function can not be expected to function.

There did not seem to be any extensive delays regarding the communication between the software. It is important to remember, though, that the case is a relatively small production line. Thus, more delays should occur as the production line increases. Further, there were more significant delays in the communication between the PLC and SIMIT than between SIMIT and Plant Simulation.

4.2.4 Simulation Model Life Cycle

The results from the hybrid model's impact on the simulation model's life cycle were hard to verify. It can be seen that the coding part could be a problem if simulation engineers do not know how to code for emulation, which was identified as a potential problem in the preliminary tests. In most projects, the simulation model is created as the first step. However, knowing from the beginning which parts should be emulated in later stages, or how to write the emulation code, may be hard to know, which also was identified as a potential problem in the preliminary tests. Having a simple if-statement at the beginning of all codes could, according to the simulation team at AFRY, make it faster and easier to expand the simulation model to a hybrid model. With the if statement, simulation code can be written as usual in the simulation mode part of the code, see Section *4.1.3.3 Methods for Emulation Control*.

Regarding name standards, the simulation team at AFRY thought it would be a valuable addition to every project. However, it is essential to remember that several simulation engineers work with other software, like AutoMod, where this object-oriented approach does not exist to the same extent. The case model could be swiftly built using the object library, and its drag and drop function made it easy to quickly build up the base for a system. The time-consuming parts of the model building was to make the adjustments to be able to both simulate and emulate the real system, going from one mode to the other. The simulation logic stood for the more time-consuming part of the model building, while the emulation logic in the Plant Simulation model only needed the pre-defined methods connected to Changed-Value control, and the rows of code where Output signal data is to be sent to SIMIT. This opens up the possibility to create an emulation connection to a simulation model that had already been built in the early development stages. As long as the name standard is followed, the transition can be done to create a hybrid model by using the methods from the library which contains the "if-else emulation is active" statement.

5
Discussion

This chapter will discuss the findings from the development and case phase in combination with literature findings. The chapter is divided into four sections, development phase with discussion connected to research question one, case phase with discussion connected to research question two and three, sustainability connecting the work to the TBL and lastly suggestions for future research.

Well-documented findings are the key in any research, as discussed in Section *3.4 Documentation*. This is especially important in research areas where the research is sparse and in areas connected to rapid technological development to enable future research. Hybrid simulation and emulation research can be connected to the rapid technological development and digitalization of industry. However, as mentioned in Section *2.5 Hybrid models* the research into the area of hybrid models is sparse, and the definition of what a hybrid model can be is widespread. Because the research is sparse and the definition of a hybrid model is so widespread, the research strategy chosen in the thesis was a triangulation of literature, quantitative, and qualitative studies.

An iterative triangulation of methods enabled the thesis to retrieve as much information as possible on hybrid models through a thorough investigation of existing research which could be used as a base to develop a new hybrid concept. Since no clear instruction could be found on how to create a dual functioning hybrid model with roots in simulation and emulation, the work started with a development phase aiming to create a hybrid model framework for emulation controlled simulation models, enabling real time emulation in a simulation environment, that later could be used in the case phase where the concept could be evaluated. Thus, the literature findings were combined with both quantitative results from evaluation case experiments and qualitative results from continuous face validation throughout the framework development process.

5.1 Development Phase

It was hard to find literature matching the specific area of hybrid simulation and emulation models. The other hybrid combination research gave a basic understanding of the general hybrid model concept and the potentials of hybrid models, as literature described it as the future of simulation. Experiments with small-scale simulation models in Plant Simulation were used at the start of the project as a

quantitative investigation into the software and simulation model building using DES software. This was essential for understanding how to create the connection between the simulation and emulation software. The iterative development process of a knowledge-building and experimental phase deepened our knowledge in how simulation and emulation worked and thus started to build a picture of how to combine them. Understanding each of the software helped to evaluate the work as the hybrid model framework grew through development and continuous testing using small-scale demo models.

The iterative development process described in Section *3.2 Development Phase* consisted of experiments combined with literature research and qualitative face validation with the team from AFRY. From this iterative process, a hybrid framework could be developed and later tested. The iterative process connected to continuous feedback was crucial for the thesis' success. However, the methods and approach developed in this study may not be directly applicable to other software since the thesis was limited to using software that had the existing external connection between the simulation and emulation software.

Significant findings during the development phase were firstly the name standards enabling stable connections and easy software integration. Secondly, developing general simulation objects and modeling solutions would allow for easy software connection. And third, developing a framework that will enable the user to emulate parts of a simulation system or a whole system. This general component development came as a request from the team at AFRY, requesting general components that can be easily adapted to any project they take on.

At the start of the project, the idea was to create an object library of emulation-prepared simulation components. However, it was not enough to only create a new object library, we also needed to develop a framework that would enable the user to understand how to build a hybrid model and the requirements on both models and their connection. This is because the area of hybrid Plant Simulation and SIMIT models has not been previously investigated in research or documented in a way that could be further built on, making our development and documentation of the developed framework extra important for future research. This is why we chose to start with a development phase to test the foundation of the developed hybrid framework, which could then be tested and evaluated in later stages.

5.1.1 RQ1

How can an emulation model control a DES model? What are the prerequisites and methods for building a hybrid DES model with this dual-functionality?

To control a DES model with an emulation model or with a PLC, a hybrid connection that can separate simulation constraints and emulation logic is needed. Why we call it simulation constraints and emulation logic here is to separate their functions. One of the main development steps was understanding that with a hybrid

model, both model's functions need to be regarded when developing methods for building and working with them, i.e., the emulation and simulation perspectives will set different requirements on a model. It is then essential to cater to both those needs for the model to be a hybrid that can switch between DES and Real-Time emulation. To enable this function, we developed a framework that lets the user enable or disabled emulation mode. This means that the simulation constraint and emulation logic can be kept separate from each other.

This dual-functioning hybrid model requires, as mentioned, that simulation conditions and constraints are kept separated from the emulation code that is used to send and retrieve information that will control the simulation model. The technical details on the connection can be found under Section *4.1.1 Model Requirements for Integration*. When coding the emulation control logic in a simulation model, it is vital to include only emulation logic, no simulation constraints, conditions, or simulation logic. For example, no simulation code should stop an object in the emulation mode when it is up to the emulation model or the PLC to perform the stop. This means that the emulation model should be stopping the conveyor speed which then signals the simulation model to set it to zero, and thus stopping the conveyor and the object on it. However, code for real-life conditions like a mechanical stop should be coded in the emulation mode and will have to be built using simulation logic for them to work. This is why to create these hybrid models there needs to be an understanding of the difference between simulation and emulation model requirements.

The hybrid model framework is therefore developed to be capable of running in both simulation mode (where the simulation model has standard constraints and logic) and in emulation mode (where the simulation model strictly complies with the emulation model signals). This is partly for the flexibility and to create a truly hybrid model but also used to enable the separation of the simulation and emulation code using the switch between the modes via simply pressing a button that is connected to the if-statements that should be in all methods of the hybrid model, described in Section *4.1.3 Object Library Creation*.

We worked a lot on putting as many methods and attributes on the objects as possible in the new object library. Keeping methods and attributes on the objects frees up space on the simulation production floor and enables the methods and attributes to follow the object without having to drag out extra things to the simulation floor. Methods and attributes have a larger potential to be individually set or changed when placed on the objects. Although, the methods and attributes placed on an object cannot be used as Changed-Value control paths in the SIMIT interface which limits their usability. We created finished components to the reach of our understanding of what is needed for a general test model and the objects needed for the case model, but there is always room for improvement and more objects that can be added to the library, following the general idea of the developed if-statement separating the simulation and emulation functions.

A prerequisite for the developed hybrid framework is using the name standard de-

veloped, described in Section *4.1.2 Name Standard*. A reason why name standards are important in hybrid models is to reduce the time searching for and remembering different names. Minimizing the number of different signal names enables the programmer to write general codes for all signals. The name standards developed during this thesis was mainly originated from discussions regarding function and enabling generalization of library components, from both simulation and emulation perspective. Respect was taken to the predefined functions and attributes in Plant Simulation and the compatibility to the model in SIMIT. Because the suffix of the signal name is based on the predefined attribute of an object, a suffix seems to be an exceptionally well fitted solution. Not only is it easy for the programmer to know what signal is handled, it is also an attribute already controlling the specific object. It can be discussed if the name standard is applicable to software other than Plant Simulation. The prefix part of the name could also be further evaluated due to an increased number of components in larger systems. Using C or L to identify a conveyor or lift was suitable for the purpose of this thesis, but it should be investigated how well-fitted this solution is for larger systems.

Lastly, we found that using Excel as a tool for managing and storing data, sorting, and creating signals for either SIMIT or Plant Simulation seems to be today's best alternative. Excel can be used to ease the creation of the model connection and reduce the risk of coding errors. Excel is widely used and it has great compatibility with many software, for example, Plant Simulation and SIMIT, it can be used to exporting and importing files to or from Excel, or even auto-generating Excel files. However, auto-generating data came with its complexities in the development stages, and it showed the importance of developing robust templates for data transfer to work smoothly. Methods and logic can probably be coded in Excel to support the hybrid connection further and ease model building. The possibility of auto-generating data from Excel during simulation is something that could be investigated in the future.

5.2 Case Phase

The case phase was used to verify and validate the framework, name standards, and the object library on an actual project. When developing a new framework, it is essential to test theoretical findings to see how the developed theory works in the real world or close to a real world setting. A more extensive test than the one carried out on the promising demo models was therefore performed before we could evaluate research questions two and three. By using a real world case that we could scale down to an appropriate size for evaluating the framework, the full potential of a hybrid model could be better evaluated, and errors in the framework's components or work method highlighted and improved. It was essential to choose a case that could be scaled down and that consisted of basic components such as the conveyor system to test more functionality of the hybrid connection, not test how well the real system could be represented. The focus in this phase was always on the function of the hybrid model and its connection between the software.

Both manual and automatic tests were carried out as described in Section *3.3.2 Integration and Test Phase*, to step by step test the connection. Testing the system with a real virtual controller was an excellent type of validation for our hybrid model. Some changes were required in the virtual controller during the integration due to its adaptation to an outdated project. Once the connection was working and the PLC could control the model, the results could be documented and evaluated to further improve the model and investigate the potential such a connection can have for virtual commissioning and DT development.

The case phase was meant to simulate a real simulation project, that is why the well-known Banks methodology for simulation project was combined with HSEM to create a methodology for the case phase, and a structure for the test of the framework, could be applied in future projects. Following the methodology for the project as a whole, the aim was to build a simulation model using the framework with little to no code for emulation and later in the integration phase add sensor signals and logic in the simulation model under emulation mode code to create the connection. Simulating a project that would have started with a simulation model to test material flow, and then transitioning this into a hybrid model where the emulation logic and PLC could be visualized and tested in the DES model. It was essential to document the functionalities of the sensors, e.g., how and where signals were sent or retrieved in the system so that it is easy to adjust the simulation model accordingly during the integration to emulation mode. This is also why the methods developed in the framework look like they do. It is to enable easy integration with an emulation model. Making comments on the programming code was essential in this stage for documentation purposes to enable further development so that someone else understands the code and, when the coding gets more complex, to ease the troubleshooting.

When building the simulation case model using the framework during integration with the emulation model, key findings showed that it was a good idea to have the predefined angled conveyors as corners for the system when building the object library. However, it turned out that sensors had to be coded in a particular way (on the entrance or exit controls) because it was impossible to place sensors on the conveyor. This resulted in a pretty complex coding to get Plant Simulation to send signals to SIMIT at the right moment. It eases the function of sending information, but with a component that cannot have a sensor placed where desired, much time needs to be put on correct code placement, which required very in-depth knowledge of how the system worked. It would have been beneficial to create another corner component in the library that worked like an angled conveyor but could contain sensors. With this said, unique case solutions will always be needed to adapt to different projects.

Beyond the coding arrangement during the integration to emulation mode and the adjustments to the SIMIT model when creating the connection between the models were smooth, thanks to the early tests that we had done when practicing and documenting the Shared Memory and the SIMIT integration interface. Using name

standards and Excel templates for generating Changed-Value control was incredibly beneficial for setting up a connection in the minimum amount of steps.

5.2.1 RQ2

What are the benefits and drawbacks of using a hybrid DES model with dual-functionality compared to using two separate models for simulation and emulation when it comes to virtual commissioning and the creation of a digital twin?

The literature describes the need for technical development for Industry 4.0 and virtual commissioning to optimize the industry further. Literature states that the possibilities of technical solutions like hybrid models will outweigh the drawbacks of the development costs in the long run. An exceptional benefit of virtual commissioning is that troubleshooting in a virtual environment does not require the physical line to stop. According to the literature, hybrid models could be seen as the next step in the technical development of today's simulation models. Today, simulation and emulation models are used separately, in different stages of the development process, for different purposes. Simulation software is often used for highly flexible models, which often support 3D visualization. Emulation on the other hand, focuses on precision (simulating the system on a signal level), to emulate the real world and its functions. The signal emulation software we have used only provides 2D visualizations. The benefits a hybrid version of these models can bring is that the 3D visualization of the simulation model can be used when emulating the system, instead of the 2D environment currently used, increasing the understanding of the functions and material flow of the system. The improved visualization a hybrid model enables can significantly benefit the development process and minimize installation problems. For example, if errors occur in the hybrid model in the emulation mode, the model shows warnings in the virtual control system in the same way as shown in the physical control system during a physical implementation.

It takes time to build a simulation model of a production line and even more to create an emulation model. Today the same line needs to be constructed from scratch several times for the different software used in different stages of virtual commissioning. In a simulation model, the production line works similarly to the physical system. Therefore it is not possible to assume that physical implementation based on a simulation model would be implemented without errors. In the emulation model, the production line works precisely as the physical system, but it is often impossible to get a good visual view. If a graphical view of the emulated system can be retrieved in an already created 3D environment, creating 2D graphics in the emulation software is redundant. To investigate, we used the hybrid framework, both 3D graphics in the Plant Simulation model and 2D graphics in the SIMIT model control chart to compare the connection, and understand how it functioned, and what errors or improvements could be made. However, models made for a visual representation of a system during virtual commissioning could be reduced with this hybrid model, moving from 2D representation in an emulation software to the 3D representation in the simulation software.

The significant benefit of visualization in hybrid models can be carried out early in the design process. When building the case model, the simulation software gave a further understanding of the system we were building. We were never on-site, but with the help of the 3D environment we could understand the production line. We clearly saw the difference 2D and 3D visualization do for understanding a system for verification and validation purposes. This leads us to believe that having the possibility to run an emulation in a simulation software will help reduce errors before implementation, and to further optimize a system that only emulation or simulation can not do by themselves. With that said, the hybrid function adds a layer of complexity to the simulation model. The simulation logic stands for most of the model complexity, or rather the need to keep the different simulation and emulation logic separate within the hybrid model. Building a DES model without any simulation logic, only for emulation purposes is less complex, since the PLC or emulation software decides the logic and conditions within the system.

To mitigate the drawback of model complexity, the hybrid model framework was designed to be as user-friendly as possible. Structures can be used to handle the complexity. This is why we have aimed to work with name standards and general methods because this is a new way of working with simulation models, and we can see that many initial models are created for visualization. However, the complexity of the connection and the possibility of starting with simulation and with a click of a button change to emulation mode is a crucial part of the hybrid model concept. It opens up possibilities to use other modes for other connections to the DES model. A hybrid model connection allowing both manual and automatic data transfer paves the way for moving from manual data transfers and digital models, to creating Digital Shadows, Digital Twins, and interconnected systems.

From literature and case tests it was found that different mindsets are used when building simulation or emulation models, since engineers with simulation or emulation modeling experiences are building the models respectively. For hybrid modeling, both mindsets are needed and thus broader competence is required. For emulation there is a need for a mechanical mindset, seeing how objects collide, stop and act, which can be beneficial to investigate in 3D rather than 2D. From a simulation engineer's perspective, potential problems when building hybrid models were connected to the coding of the emulation logic. The simulation model is often the first step in virtual commissioning, and the model is thus built under simulation pretenses by simulation engineers with limited emulation knowledge. The limited knowledge of emulation could potentially lead to unnecessary complexity for the simulation engineers building the initial model. Thus, it is beneficial to have an idea beforehand whether the model should be hybrid or not and deciding early if it is worth increasing the base model's complexity or not.

Lastly, a drawback mentioned in the literature was the time delays found in hybrid model connections. Time delays were widely mentioned as a challenge with all types of hybrid models (Section *2.5 Hybrid Models*). Both in the earlier stages of the project and during the case tests, no direct time delays were registered while

running both models in real-time. The absence of time delays was perhaps related to the small size of the coupling and model. However, time delays could be registered if the simulation in Plant Simulation was not performed in real-time. Running the simulation faster than in real-time simulation mode means that the real-time emulation is not active anymore. Further investigation into time delays was not regarded in this thesis since it was outside the scope but is something that should be investigated in the future.

5.2.2 RQ3

How can a hybrid DES model with dual-functionality extend the life cycle of a simulation model?

Literature states that dual-functionality models can extend the life cycle of DES models and even reduce the number of DES models needed in the commissioning stage by increasing a model's functionality. Simulation models with increased functions like hybrid simulation-emulation models may even save money by shortening the commissioning time.

The significant benefits of using a hybrid model in the stages of development might not be clear. However, from a separate model perspective and with the dual-function of simulation and emulation mode, the benefits are apparent. The DES model's life cycle can, with the hybridization, be increased by using it to enhance emulation models to test control systems before physical implementation, enabling robust designs thus potentially shortening installation times. However, the hybrid model can still be used as an ordinary DES model. For example, the hybrid model could be used for both classical flow simulation and for final system tests before real world installation.

The framework with its function to enable or disable emulation mode can be achieved by following name standards and critical aspects of the framework when building initial simulation models in a project. It opens up the hybridization of simulation models, increasing the model's function and thus its usability in later design and implementation stages of virtual commissioning. The possibility to increase a simulation model's life cycle by increasing its functionality opens up for increased test and visualization possibilities, since using the same model for more advanced and real-life "alike" tests may largely reduce the time a production line must stop for the physical implementation.

After the project's development phase and test analyses, we agree with research that a dual-functioning hybrid model can potentially extend the life cycle of a simulation model. Even if the hybrid function can increase complexity and demand more knowledge of the simulation engineer, we believe that a framework and general structures can mitigate this drawback. Long-term gains can outweigh the short-term costs. As mentioned earlier in the discussion the hybrid connection opens up for possible digital twin development with the automatic data transfer between

models and the possibility to test real world systems using a virtual controller or even a real PLC.

5.3 Sustainability

The sustainability aspects connected to the project will be discussed in this section, using the TBL perspective described in Section *2.6 Sustainability*.

Social aspects
Building and managing complex hybrid simulation and emulation models require, as mentioned, knowledge in both areas of simulation and emulation. Unless the hybrid model engineer modeling the hybrid model has sufficient experience or training with both systems, mindsets, and programming languages, the hybrid model concept can create unnecessary stress on the individual. Being exposed to high levels of stress for long periods can cause severe diseases, resulting in sick leave.

Good documentation and structures ease the understanding of complex systems, reducing the stress of uncertainty and making it possible for someone else to adjust the hybrid model to a specific production, which is why we focused a lot on creating substantial documentation throughout the project. A structured framework and common way of working enable teamwork to a greater extent. Being a part of a team often feels rewarding, and the results are often of higher quality as several people gather their knowledge and work together. As the technical complexity increases the importance of highly educated workers or working together in teams also increases, to mitigate the stress.

Further, companies need to keep up with market changes and work fast and efficiently to stay competitive. A company at the forefront in technological development often attracts more customers and employees and can therefore grow and work towards sustainability on more fronts.

Economical aspects
In literature, this is the area under the TBL with the most direct and long-term sustainability gains, both from specialized areas of virtual commissioning but also as Industry 4.0 as a whole. Longer life cycles of a model used for virtual commissioning with more functions require fewer persons to build new models for different parts of the commissioning phase. Fewer models, fewer people, and less time spent on building models save money. It is often cost-effective to have digital models and test the system before implementation. Testing the system in advance may save time and money as the physical production line will not be stopped for as long as it would in a normal commissioning phase. Most of the implementation tests have already been done in a virtual environment. During the digital system development the developers go through the system, find errors, and make changes to finally verify the system with, for example, PLC controllers to make sure the system works as it should. This could be seen in the test phase, where the integration of PLC needed adjustment, much like when building the system for actual implementation.

Implementations that are not successful need adjustments which take time from the production. Significant errors could cause severe crashes in a physical system, for example, if the control panel sends the wrong signals and a car would fall off the rails and down on the floor and break, or if a machine gets overloaded and breaks. These severe mistakes can be reduced if the system has been tested virtually in advance. Since it is reasonable to assume that most companies do not have unlimited access to money or time, virtual tests can significantly improve the economic sustainability of a company.

Environmental aspects
Shorter commissioning time also reduces the required emissions from the implementation stages. It often requires more emissions when starting a cold machine than an already warm one. Suppose the machine does not need to stop for as long after a successful implementation is tested virtually. In that case, the machine will not be as cold, reducing the environmental impact. Another benefit with virtual models is that it is often possible to simulate the waste, emissions, and energy required for specific processes. Virtual modeling makes it possible for the production planners to early on in the design stages design for reduced environmental impact of a system. The engineers then have an excellent overview of the processes and emissions to make suitable adjustments to the physical system.

5.4 Future Research

Software improvement
During the development phase, adjustments in how the framework was developed had to be made concerning software limitations. Plant Simulation is designed to handle the connection with SIMIT, but the objects in Plant Simulation are designed for simulation purposes, not for building a hybrid model. Further development is needed to reduce the complexity of object library components and signal management to code for emulation in and hybrid model. One example is the need for more sensor possibilities on a wider variety of components in Plant Simulation.

Framework development
The hybrid model framework developed is still in the early development stages. Further research, development, and tests are needed to fully grasp the hybrid model's potential and find the best areas of use. As we see today, focus areas for future development are the sensor code and its signal transfer between the software, creating more emulation prepared objects to enable more extensive tests of the hybrid model concept. Since no real-time delays could be seen in our test models, a more extensive investigation into this specific area is needed. Time delays are also one of the primary concerns in the literature regarding hybrid models and an area in need of further investigation.

Hybrid development
Research should be made on stress testing a more extensive hybrid simulation-emulation system to picture the delays better and discover other possible complica-

tions. However, with hybrid models nominated as the future of simulation, the entire research area has potential for significant technological development with many different niches, such as testing other software capabilities and hybrid concepts.

6
Conclusion

This thesis has successfully created a working hybrid model with the dual functions of a DES and emulation. This model can be built using the framework developed and presented in this thesis, containing a name standard and an object library with emulation prepared simulation components. The purpose of the framework is to enable building emulation prepared DES models that can be run as either a DES model or using real-time emulation by establishing a connection with an emulation model or a virtual PLC. The framework covers the basis for a hybrid connection between software models in Plant Simulation and SIMIT, enabling an understanding of the needs and requirements that the different perspectives of simulation and emulation put on a hybrid model that could be applied to other software or hybrid concepts connected to simulation and emulation principles.

The hybrid model concept creates more visualization during the virtual commissioning. The model's increased functionality enables more integration tests before the installation of a system. The simulation model's life cycle can thus, with the hybridization, be increased and reduce the required time for virtual commissioning, making commissioning even shorter, simpler, and more cost-effective. Further, its automatic data transfer qualities do not only improve virtual commissioning, it also opens up for Digital Twin development. However, this dual functionality will increase the model complexity, but this complexity allows systems to be tested before implementation in a way that has not been done before.

Bibliography

[1] Björkdahl, J., Wallin, M. & Kronblad, C. (2018). *Digitalisering - mer än teknik, Kartläggning av svensk forskning och näringslivets behov*. VINOVA. https://www.vinnova.se/contentassets/24d0fc24d08b4d0d900a805384ffc51b/vr_18_06.pdf

[2] Davies, R., Coole, T., & Smith, A. (2017). Review of Sociotechnical Considerations to Ensure Successful Implementation of Industry 4.0. *Procedia Manufacturing, 11*, 1288-1295. https://doi.org/10.1016/j.promfg.2017.07.256

[3] Starner, C., & Chessin, M. (2010). Using emulation to enhance simulation. *In Proceedings of the 2010 Winter Simulation Conference*, 1711-1715. https://doi.org/10.1109/WSC.2010.5678900

[4] Meyer, T., Pöge, C. & Mayer, G. (2012). Integration of emulation functionality into an established simulation object library. *Proceedings of the 2012 Winter Simulation Conference (WSC)*, 1-11. https://doi.org/10.1109/WSC.2012.6465220

[5] de Paula Ferreira, W., Armellini, F. & De Santa-Eulalia, L. A. (2020). Simulation in industry 4.0: A state-of-the-art review. *Computers Industrial Engineering, 149*, Article 106868. https://doi.org/10.1016/j.cie.2020.106868

[6] Lee, C. G. & Park, S. C. (2014). Survey on the virtual commissioning of manufacturing systems. *Journal of Computational Design and Engineering, 1*(3), 213-222. https://doi.org/10.7315/JCDE.2014.021.

[7] Boerjesson, M., Hayes, P., Mirstam, E., Nordmark, E., Oppelt, M., Pohl, M., Schwaab, J., Andersson, K. P., Buhlin, A. & HaiderValue, G. (2020). *Value creation with plant modelling and simulation*. [White paper]. Siemens AFRY.

[8] Johansson, M. & Nilsson, J. (2015). *VIRTUAL COMMISSIONING: Verification of PLC logic in simulation programs*. [Bachelor thesis, University of Skövde]. University of Skövde.

[9] Ghobakhloo, M. (2018). The future of manufacturing industry: a strategic roadmap toward Industry 4.0. *Journal of Manufacturing Technology Management, 29*(6), 910-936. https://doi.org/10.1108/JMTM-02-2018-0057

[10] Kumar, K., Zindani, D., & Davim, J.P. (2019). Industry 4.0: Developments towards the Fourth Industrial Revolution. SpringerLink. https://doi-org.proxy.lib.chalmers.se/10.1007/978-981-13-8165-2

[11] Teknikföretagen. (2015). *Digitaliseringens betydelse för industrins förnyelse*. Teknikföretagen. https://www.teknikforetagen.se/globalassets/rapporter/digitalisering/digitaliseringens\protect\discretionary{\char\hyphenchar\font}{}{}betydelse-for-industrins-fornyelse.pdf

[12] Roser, C. (2015, December 29). *A Critical Look at Industry 4.0*. Allaboutlean.com. https://www.allaboutlean.com/industry-4-0/

[13] Lim, K. Y. H., Zheng, P. & Chen, C. H. (2019). A state-of-the-art survey of Digital Twin: techniques, engineering product lifecycle management and business innovation perspectives. *Journal of Intelligent Manufacturing, 31*, 1313–1337. https://doi.org/10.1007/s10845-019-01512-w

[14] Tao, F., Cheng, J., Qi, Q., Zhang, M., Zhang, H. & Sui, F. (2017). Digital twin-driven product design, manufacturing and service with big data. *The International Journal of Advanced Manufacturing Technology 10*(4), 3563–3576. https://doi.org/10.1007/s00170-017-0233-1

[15] Kritzinger, W., Karner, M., Traar, G., Henjes, J. & Sihn, W. (2018). Digital Twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine, 51*(11), 1016-1022. https://doi.org/10.1016/j.ifacol.2018.08.474

[16] Tao, F., Zhang, H., Liu, A. & Nee, A. Y. (2018). Digital twin in industry: State-of-the-art. *IEEE Transactions on Industrial Informatics, 15*(4), 2405-2415. https://doi.org/10.1109/TII.2018.2873186

[17] Boschert S. & Rosen R. (2016) Digital Twin—The Simulation Aspect. In P. Hehenberger D. Bradley (Eds.) *Mechatronic Futures*, 59-74. Springer, Cham. https://doi.org/10.1007/978-3-319-32156-1

[18] Hofmann, W., Langer, S., Lang, S. & Reggelin, T. (2017). Integrating virtual commissioning based on high level emulation into logistics education. *Procedia Engineering, 178*, 24-32. https://doi.org/10.1016/j.proeng.2017.01.055

[19] McGregor, I. (2002). The relationship between simulation and emulation. *In Proceedings of the Winter Simulation Conference, 2*, 683-1688. https://doi.org/10.1109/WSC.2002.1166451

[20] Ingalls, R.G. (2011). Introduction to simulation. *Proceedings of the 2011 Winter Simulation Conference (WSC)*, 1374-1388. https://doi.org/10.1109/WSC.2011.6147858

[21] Popovici, K. & Mosterman, P. J. (2017). (Ed.) *Real-time simulation technologies: principles, methodologies, and applications.* CRC Press. `https://books.google.se/books?id=GvlzMPbuZqAC&dq=real-time+simulation+in+discrete+event+model&lr=&hl=sv&source=gbs_navlinks_s`

[22] Fishman, S. G. (2001). *Discrete-Event Simulation Modeling, Programming, and Analysis*. Springer, New York. https://doi.org/10.1007/978-1-4757-3552-9

[23] Agalianos, K., Ponis, S. T., Aretoulaki, E., Plakas, G. & Efthymiou, O. (2020). Discrete Event Simulation and Digital Twins: Review and Challenges for Logistics. *Procedia Manufacturing, 51*, 1636-1641. https://doi.org/10.1016/j.promfg.2020.10.228.

[24] Rehman, M. & Pedersen, S. (2012). Validation of simulation models. *Journal of Experimental Theoretical Artificial Intelligence, 24*(3), 351–363. https://doi.org/10.1080/0952813X.2012.695459

[25] Sargent, R.D. (2017). Verification and validation of simulation models. *Journal of Simulation, 7*(1), 12-24. https://doi.org/10.1057/jos.2012.20

[26] Oreskes, N., Shrader-Frechette, K. & Belitz, K. (1994). Verification, validation, and confirmation of numerical models in the earth sciences. *Science, 263*(5147), 641-646.

[27] Beisbart, C. & Saam, N. J. (2019). *Computer simulation validation*. Springer, Cham. https://doi.org/10.1007/978-3-319-70766-2

[28] Brailsford, S. C, Eldabi, T., Kunc, M., Mustafee, N., & Osorio, A. F. (2019). Hybrid simulation modelling in operational research: A state-of-the-art review, *European Journal of Operational Research, 278*(3), 721-737. https://doi.org/10.1016/j.ejor.2018.10.025

[29] Scheidegger A.P.G., Pereira T.F., de Oliveira M.L.M., Banerjee A. & Montevechi J.A.B. (2018). An introductory guide for hybrid simulation modelers on the primary simulation methods in industrial engineering identified through a systematic review of the literature. *Computers Industrial Engineering, 124*, 474-492. https://doi.org/10.1016/j.cie.2018.07.046

[30] Hasan, B. K. (2005). *Methodology to develop hybrid simulation/emulation model*. [Doctoral thesis, Sheffield Hallam University]. Sheffield Hallam University `http://shura.shu.ac.uk/19768/`

[31] Lee, H.W., Thuente, D. & Sichitiu, M. (2014). Integrated simulation and emulation using adaptive time dilation. *Proceedings of the 2014 ACM Conference on SIGSIM Principles of Advanced Discrete Simulation*. https://doi.org/10.1145/2601381.2601384.

[32] Vera, E. A. M. (2020). *Virtual Commissioning of an industrial wood cutter machine*. [Bachelor thesis, Luleå University of Tehcnology]. Luleå University of Technology. `https://www.diva-portal.org/smash/get/diva2:1385515/FULLTEXT01.pdf`

[33] Johnstone, M., Creighton, D., & Nahavandi, S. (2007). Enabling industrial scale simulation/emulation models. *2007 Winter Simulation Conference*, 1028-1034. https://doi.org/10.1109/WSC.2007.4419701

[34] Ružarovský, R., Holubek, R., Sobrino, D. R. D., & Janíček, M. (2018). The simulation of conveyor control system using the virtual commissioning and virtual reality. *Advances in Science and Technology Research Journal, 12*(4), 164-171. https://doi.org/10.12913/22998624/100349

[35] Siemens. (2018). *Tecnomatix Plant Simulation Help*. Siemens Product Lifecycle Management Software Inc. `https://docs.plm.automation.siemens.com/content/plant_sim_help/15/plant_sim_all_in_one_html/en_US/tecnomatix_plant_simulation_help/tecnomatix_plant_simulation/tecnomatix_plant_simulation_help.html`

[36] Siemens. (2020). *SIMATIC SIMIT Simulation Platform (V10.2) - Operating Manual*. Siemens. `https://support.industry.siemens.com/cs/attachments/109780242/SIMIT_enUS_en-US.pdf`

[37] Siemens. (May 21, 2019). Virtual Commissioning for Tecnomatix Plant Simulation & PLCSIM Adv. [Video]. YouTube. `https://www.youtube.com/watch?v=NMaecAEuSak&ab_channel=Siemens`

[38] Andersson, M. & Zvantesson, A. (2018). *Discrete Event Simulation as a Tool for Virtual Commissioning*. [Master's thesis, Chalmers University of Technology]. Chalmers University of Technology. `https://hdl.handle.net/20.500.12380/255345`

[39] Engström, V., & Liao, Z. (2017). *PLC Integrated Discrete Event Simulation for Production Systems*. [Master's thesis, Chalmers University of Technol-

ogy]. Chalmers University of Technology. https://hdl.handle.net/20.500.12380/251696

[40] Hourneaux Jr, F., da Silva Gabriel, M. L., & Gallardo-Vázquez, D. A. (2018). Triple bottom line and sustainable performance measurement in industrial companies. textitRevista de Gestão 25(3). https://doi.org/10.1108/REGE-04-2018-0065

[41] Herrmann, C., Schmidt, C., Kurle, D., Blume, S., & Thiede, S. (2014). Sustainability in manufacturing and factories of the future. *International Journal of precision engineering and manufacturing-green technology, 1*(4), 283-292. https://doi.org/10.1007/s40684-014-0034-z

[42] Braccini, A. M., & Margherita, E. G. (2019). Exploring organizational sustainability of industry 4.0 under the triple bottom line: The case of a manufacturing company. *Sustainability, 11*(1), 36. https://doi.org/10.3390/su11010036

[43] Sony, M. (2020). Pros and cons of implementing Industry 4.0 for the organizations: a review and synthesis of evidence. *Production & Manufacturing Research, 8*(1). 244-272 https://doi.org/10.1080/21693277.2020.1781705

[44] Kamble, S. S., Gunasekaran, A., & Gawankar, S. A. (2018). Sustainable Industry 4.0 framework: A systematic literature review identifying the current trends and future perspectives. *Process Safety and Environmental Protection, 117*, 408-425. https://doi.org/10.1016/j.psep.2018.05.009

[45] Webster, D.G., Padgett, M.L., Hines, G.S., & Sirois, D.L. (1984). Determining the level of detail in a simulation model - a case study. Pergamon press Ltd. *Computer & Industrial Engineering, 8*(3-4), 215-255.

[46] Snyder, H. (2019). Literature review as a research methodology: An overview and guidelines. *Journal of Business Research, 104*, 333-339. https://doi.org/10.1016/j.jbusres.2019.07.039

[47] Shorten, A. & Joanna Smith, J. (2017). Mixed methods research: expanding the evidence base. *Evid Based Nurs, 20*(3). https://doi.org/10.1136/eb-2017-102699

[48] Schoonenboom, J. & Johnson, R.B. (2017). How to Construct a Mixed Methods Research Design. *Kolner Zeitschrift Fur Soziologie Und Sozialpsychologie, 69*(2), 107-131. https://doi.org/10.1007/s11577-017-0454-1

[49] Streefkerk, R. (2019). *Inductive vs. deductive reasoning.* Scribbr. https://www.scribbr.com/methodology/inductive-deductive-reasoning/

[50] *Snowballing Technique. A Dictionary of Sociology.* (2019) Encyclopedia. Retrieved Mars 18, 2021, from https://www.encyclopedia.com/social-sciences/dictionaries-thesauruses-pictures-and-press-releases/snowballing-technique

[51] Siemens. (2019). *SIMIT Simulation Platform.* Siemens. https://assets.new.siemens.com/siemens/assets/api/uuid:4ce72796-3757-493c-8afe-e1fefb988025/simitsimulationplatformdatasheet2019-1en-144.pdf

[52] Baxter, P. & Jack, S. (2008). Qualitative Case Study Methodology: Study Design and Implementation for Novice Researchers. *The Qualitative Report, 13*(4), 544-559. https://nsuworks.nova.edu/tqr/vol13/iss4/2

[53] Xie, I., & Matusiak, K. (2016). Interface design and evaluation. *Discover digital libraries: Theory and practice*, 205-230. https://doi.org/10.1016/B978-0-12-417112-1.00007-7

[54] Montevechi, J. A. B., Pereira, T. F., da Silva, C. E., Miranda, R. D. C., & Scheidegger, A. P. G. (2015). Identification of the main methods used in simulation projects. *2015 Winter Simulation Conference (WSC)*, 3469-3480. https://doi.org/10.1109/WSC.2015.7408507

[55] Banks, J. (1999). Introduction to simulation. *Proceedings of the 31st conference on Winter simulation: Simulation, 1*, 7-13. https://doi.org/10.1145/324138.324142

[56] Oscarsson, J., Urenda Moris, M. (2002). Best modeling methods: documentation of discrete event simulation models for manufacturing system life cycle simulation. *Proceedings of the 34th Winter Simulation, 2*, 1073-1078. https://doi.org/10.1109/WSC.2002.1166359

[57] Svensson, C., & Tehler, I. (2016). *Facilitating the working process and documentation of DES projects*. [Master's thesis, Lund University, LUP Student Papers] `http://lup.lub.lu.se/luur/download?func=downloadFile&recordOId=8887858&fileOId=8887859`

[58] Balci, O. (2010). Golden rules of verification, validation, testing and certification of modeling and simulation applications. *SCS M&S Magazine, 4*(4), 1-7. `http://scs.org/wp-content/uploads/2016/12/2010-10-Issue04-3.pdf`

[59] *Technical writing/Types of User Documentation: Process documentation.*(2020, October 20). Wikiversity. `https://en.wikiversity.org/wiki/Technical_writing/Types_of_User_Documentation`

[60] Mohajan, H. K. (2017). Two criteria for good measurements in research: Validity and reliability. *Annals of Spiru Haret University. Economic Series, 17*(4), https://doi.org/10.1177/160940690200100202

[61] Morse, J. M., Barrett, M., Mayan, M., Olson, K., & Spiers, J. (2002). Verification strategies for establishing reliability and validity in qualitative research. *International journal of qualitative methods, 1*(2), 13-22. https://doi.org/10.1177/160940690200100202

A

Simulation Methodologies

Bank's methodology covers the essential steps in a simulation project, from early planning and data collection to building, verifying, and validating the model in preparation for the experimental phase, where intended experimentation use of the model is carried out and lastly, to tie the process together the work should be documented. These steps can be divided into three main categories, preparation stage, model building, and analysis stage. This methodology is developed for classical simulation projects [55]. Hasan [30] has, in turn, created a similar methodology to that of DES modeling, but the methodology focuses on the work procedure for building hybrid emulation and simulation models (Figure A.1). The models consist of three main steps: developing a base simulation model, developing a detailed emulation model, and integrating the controller in the simulation model with the emulation model. Where the structure is again similar to Banks and models reviewed by Montevechi et al. [54]. This leaves the methodology developed in a simulation project open for adjustments so that the method and individual steps will suit the specific project and its limitations and intended goals.

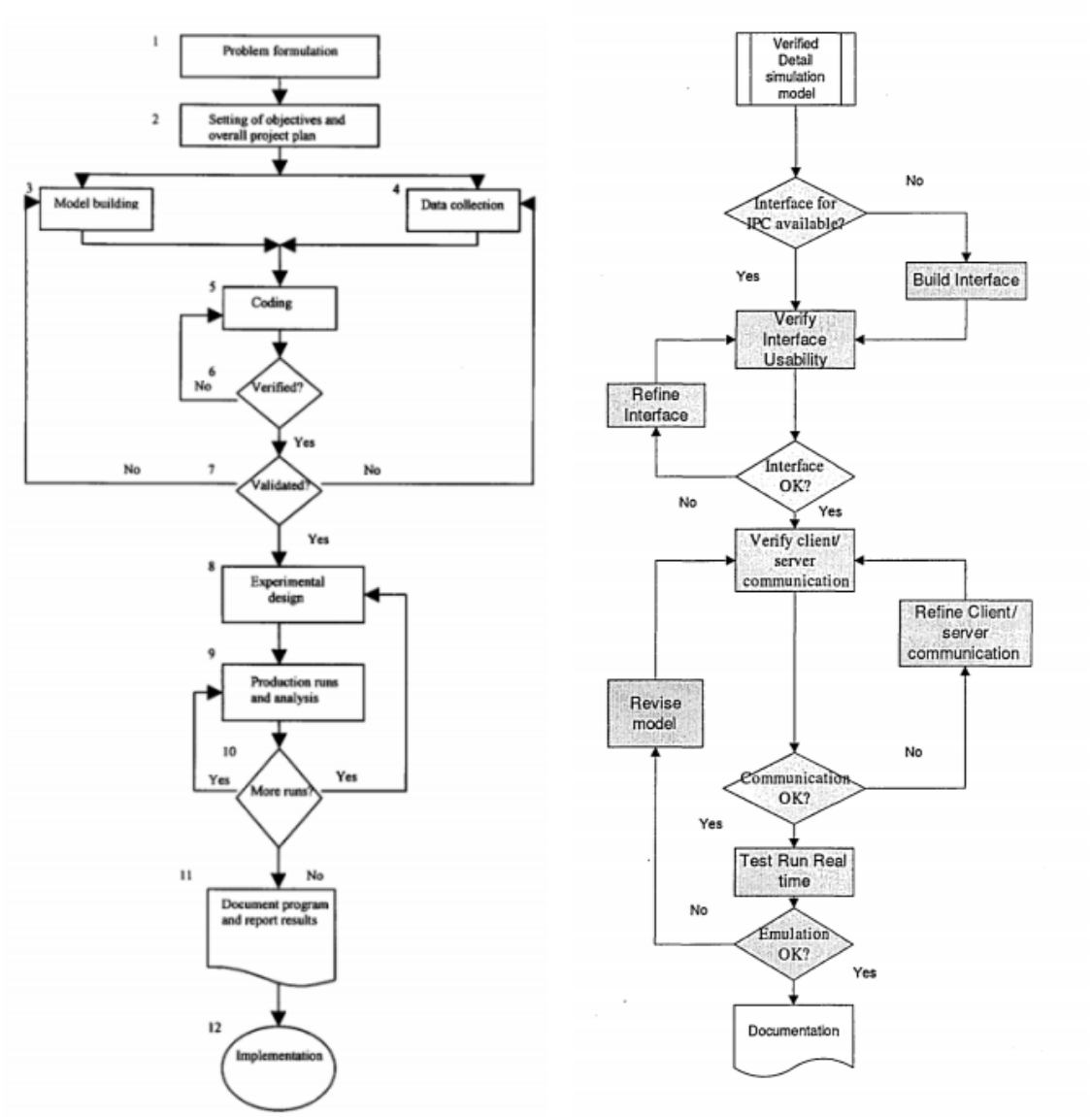


Figure A.1: Bank's model retrieved from Banks [55], on page 12 to the left, and the HSEM model retrieved from Hasan [30], on page 82 to the right.

B

Model Requirements

Translated version, from Swedish to English, of the initial requirement specification created in the development phase together with the team at AFRY. This document was used as a reference during the framework development to meet AFRYs requirements and wishes.

Object Library Goals

The goals are to make a new object library with associated methods that are sufficient to enable the SIMIT connection. The library should be so general that the user should be able to build the model according to the company's guidelines, but the user should not be limited in how the user usually structures his models. When entering an object in the Plant model, it is enough to give the object the correct name to be ready to receive SIMIT signals. If one wants an object in a Frame, this frame must be named the object name in the frame + "_Frame" (For several objects in a frame, there are options for this if you want more objects to be controlled by SIMIT in the same frame). Then the frame is ready to receive SIMIT signals. The SIMIT interface only needs to be updated by manually copying the "Item Name" column and pasting it into the "Alias" column. One also need to enter paths under "Changed-Value control," either via an init method that creates a list of the methods in the correct order, which you then paste, or if you have a method in the same frame that works for all signals, etc. (There are alternative solutions).

Prerequisites and Requirements

It is required to have a SIMIT model with a signal name with an associated value that matches the object names in the Plant model [name standards]. The signal names should have a similar basis for the same type of object (e.g., Conveyor1 or LiftTable2) and end with a descriptive word of the signal type (e.g., _Speed or _Acceleration). Signal names must start with a letter as a prefix because Plant Simulation requires that, e.g., Prefix_ObjectName. Spaces are not allowed either. The objects in the new library are pre-modeled so that they are just entered into the model and given the correct name so they can be controlled using SIMIT. The signals must be in a Coupling of the type Shared Memory - it is its contents that are read into the Plant Simulation model. One must manually enter this Shared Memory name in the SIMIT interface of the Plant model.

Signal description in the header must be checked under Shared Memory to be used (General tab under Shared Memory name and mutex). Currently, the SIMIT "Items" table in the Plant Simulation interface needs to be manually updated for the "Alias" and "Changed-Value control" columns. There are solutions on how to do this. SIMIT and Plant Simulation must be run simultaneously on the same computer. A Shared Memory list of signals is recommended to be completed before importing it to Plant Simulation (it can be a bit cumbersome in Plant Simulation when a signal is removed from SIMIT). If one wants to change the signals in SIMIT, one must select "Disable" on the SIMIT interface in Plant Simulation. When all changes have been made in SIMIT, one can again take "Enable" in Plant Simulation $\longrightarrow$ Apply $\longrightarrow$ Import Items and then update your "Items" table with Alias and Changed-Value control.

A limitation to both the software Plant Simulation and SIMIT is that if you open a project and make changes in a later, updated version of the software, it it not possible to open it in an earlier version again. Thus if several programmers are working on the same model, they must all have the same version of both programs. Further, the company can not choose to have the newest version for a while and then go back to an earlier version (assuming an earlier version may be less expensive). The models would require total re-building from scratch in that case. However, a project created in an early version can be opened and edited in a newer version. It is only stepping back that does not work.

Framework

If the SIMIT model exists and Plant Simulation model is created, the Plant Simulation model must be built with the help of the library of objects that contains finished building blocks to easily connect SIMIT, i.e., frames with finished methods laid out and objects that contain set attributes. A recommendation is to derive or duplicate these depending on how you want to build your model. The most important thing is to name all objects that are to be controlled by signals from SIMIT the same name as its corresponding signals according to the set standard. When the model is completed, the SIMIT interface is configured, where ItemNames is copied and pasted as Alias and Changed-Value control is created or generated. If Plant model exists and SIMIT is created: If the default name is used in the Plant model, signals that control an object must be named after the object. After that, all signals to be sent or received by Plant must be entered in one or more shared memories. The connection can be made by activating signal description in the header in shared memory and activating SIMIT interface in Plant to then copy ItemNames and paste under Alias and create Changed-Value control list or copy from excel generated. If both are built at the same time: Follow the instructions above from both directions.

C
Signal Management

Signal Types

Examples of the different signal or data types that can be used in the Shared Memory coupling are presented in Figure C.1 according to Siemens SIMATIC SIMIT Simulation Platform (V10.2) - Operating Manual on page 192 [36].

Table 3-4 Definition of data types

Data type	Variable	Notation	Value range
BOOL	1 bit	M<byte>.<bit>	True/False
BYTE	1 byte (8 bits)	MB<byte>	0 ... 255 or −128 ... 127
WORD	2 bytes	MW<byte>	0 ... 65,535
INT	2 bytes	MW<byte>	−32,768 ... 32,767
DWORD	4 bytes	MD<byte>	0 ... 4,294,967,295
DINT	4 bytes	MD<byte>	−2,147,483,648 ... 2,147,483,647
REAL	4 bytes	MD<byte>	$\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$

Figure C.1: The different data types that can be used in the SIMIT Shared Memory coupling, retrieved from the SIMIT handbook [36].

Changed Value Control Templates

Theoretically, the Item list in the SIMIT interface in Plant Simulation can be exported to Excel, edited, and imported into Plant Simulation, see Figure C.2. The list can also be Exported from Plant and Imported to SIMIT. This thesis made a small effort in testing these features, but it did not work straight away as desired. Because this thesis made a simulation model based on an emulation model, each time a signal in the interface was changed, the whole list had to be removed and re-imported. Therefore we chose to create all lists in Excel and copy-paste them into the correct locations of the models. Then we automatically replaced the old signals with the new ones. However, in a normal project, the simulation model would probably be built first and be finished before the SIMIT modeling would be initiated. Then these problems would not occur to the same extent as during our initial research of the topic.

	A	B	C	D	E	F	G
1	**Item Name**	**Type**	**Alias**	**Changed-Value Contrc**	**In/Out**	**Offset**	**Type**
2	CL_563M1UE1_Lowspeed	BOOL	CL_563M1UE1_Lowspeed		Output	-1	2
3	CL_563M1UE1_Pos	REAL	CL_563M1UE1_Pos	LiftMethod	Input	-1	13
4	CL_563M1UE1_Speed	REAL	CL_563M1UE1_Speed	ConveyorMethod	Input	-1	13
5	CL_563M1UE1_Stop	BOOL	CL_563M1UE1_Stop		Output	-1	3
6	CL_564M1UE1_EntrySpeed	REAL	CL_564M1UE1_EntrySpeed	ConveyorMethod	Input	-1	13
7	CL_564M1UE1_ExitSpeed	REAL	CL_564M1UE1_ExitSpeed	ConveyorMethod	Input	-1	13
8	CL_564M1UE1_Lowspeed	BOOL	CL_564M1UE1_Lowspeed		Output	-1	4
9	CL_564M1UE1_Pos	REAL	CL_564M1UE1_Pos	LiftMethod	Input	-1	13
10	CL_564M1UE1_Stop	BOOL	CL_564M1UE1_Stop		Output	-1	5

Figure C.2: Excel list of the exported Item list from the SIMIT interface in Plant Simulation.

It can be beneficial to have a parameter list with all the parameters collected. This could be made in a so-called Data Table in Plant Simulation. Then it is possible to categorize the data in the table after what type of data it is (for example, real, integer, string), and it is possible to label columns and rows easily. These Data Tables are compatible with Excel so that these lists can be Exported and Imported from Excel. Excel has more functions and possibilities of coding than a Data Table in Plant Simulation has. Therefore, Excel can be a suitable tool for storing, structuring, and managing data.

Excel as a tool for managing data can also be used to create changed-value control lists, for example, based on the Alias. We used Excel to retrieve a list of corresponding methods based on the signal name. All signals ending with "_Speed", "_EntrySpeed", or "_ExitSpeed" should be sent to the method "ConveyorMehtod". All signals ending with "_Pos" should be sent to the method "LiftMethod". The methods are described in section 4.1.3.2 Methods. The formula we used in excel to retrieve this list is shown in Equation C.1 (assume that the signal name lays in box A2). The same expression was used for all signals.

$$\begin{aligned} = OM(\text{ÄRTAL}(\text{SÖK}(\text{"}_{speed}\text{"}; A2)); \text{"}ConveyorMethod\text{"}; \\ OM(\text{ÄRTAL}(\text{SÖK}(\text{"}_{pos}\text{"}; A2)); \text{"}LiftMethod\text{"}; \text{""})) \end{aligned} \tag{C.1}$$

D

Technical Details

Wait until
Only watchable expressions can be coded in a "waituntil" statement. If an expression is watchable or not is shown in the attributes list of the specific object. An example of the waituntil statement:

```
waituntil a-watchable-statement
```

To stop the simulation until a non-watchable expression is fulfilled, a repeat-loop with a wait statement in it can be used. If the product should be stopped, do not forget to add @.*Stopped* := *true* before the wait loop, and @.*Stopped* := $false$ after the loop. The product can also be stopped by stopping the speed of the conveyor during this period by coding ?.*Speed* := 0 when the speed should be set to 0, and then ?.*Speed* := *originalSpeed*. An example of the repeat loop:

```
repeat
    wait 1
until a-non-watchable-statement
```

String search
Another coding example to keep in mind is the different string-search variations, i.e., regex and strRCopy. Regex_search searches for a specific string inside another string, while strRCopy copies a decided number of tokens in a string and creates a new string.

```
regex_search(SourceText, RegularExpression)

strRcopy(SourceText, NumberOfCharacters)
```

String to object
If the object name from a string is found using a string search and the aim is to set a new value to an object, the type of the variable needs to be changed from string to object. Simply referring to the object's name (which is a string) is not sufficient for Plant Simulation to find the correct object. The command str_to_obj can be used to transform a string into an object. Utilizing the outcome from this command, new values can be set to the correct object.

Reading data of type float
A real value from a SIMIT signal comes in the data datatype float. If the aim is to wait for a lift until it has a certain height, it is not sufficient to wait until the height equals an integer or real value. The wait until the condition has to be height is about equal, setting the allowed difference to epsilon. The float has 16 decimals;

thus, epsilon can be very small. For example, if the wait condition for the lift is set to reach 1.3 meters exactly, the signal from SIMIT stops when 1.2999995628 is reached because SIMIT thinks that the position is reached. Otherwise, the repeat condition will never be met. However, wait until the lift is 1.3 meters +/- 0.000001 meters works. This is an allowed difference of microscopical size and will only affect the transitioning between simulation and emulation mode.

When returning from the emulation mode to simulation mode, there is a need to reset the values of the lifts so that they are correct and not affected by the float data. Otherwise, a lift that will rise with 0.01 meters per step will rise above 1.3 meters by 0.01, which is more significant than epsilon. Thus the lift will keep going up forever. Using an init or reset method is suitable for resetting the values between the different modes.

Other Experimental Solutions for Angled Conveyor
Other solutions for angled conveyors were tested on a small scale at this stage, for example, taking two conveyors and placing them as a corner. Via methods, it can be possible to rotate the parts between the tables graphically. In that case, it is possible to use as many sensors as desired, wherever it is required. However, the front side of the part may not be the first side reaching a new position. The "front" may still be on the first front side. This extra code for rotating the part would also be needed on all upcoming conveyors where the part should not be rotated the same way as on the first conveyor. This solution would require more investigation. Another solution to the corners and rotation of the products is to use turntables or turn plates. Using a turntable or turn plate would mean that they would be added to the number of conveyors; thus, additional conveyors must be considered. Turntables and turn plates can not have sensors attached either in the way they are currently programmed.

E

Object Library

This appendix can be seen as a brief handbook describing the components of the object library for Plant Simulation. This appendix contains information on the emulation-prepared simulation objects and methods, best practices, and general information on the concept that can be used to develop the framework. This is an extension of the information found under the result section *4.1 Hybrid Model Framework*.

Simulation Object Library

The emulation prepared object library for Plant Simulation consists of methods, checkbox, data table, and objects enabling the user to build an emulation prepared simulation model. Figure E.1 displays the content of the object library, named "SIMITObjectLibrary". The general rule for building this hybrid simulation model is that simulation code and emulation code must be kept separate. This is done using an if-else condition that checks if a checkbox is true or false, i.e., if emulation or emulation mode is activated in all methods that want both simulation and emulation constraints and conditions.

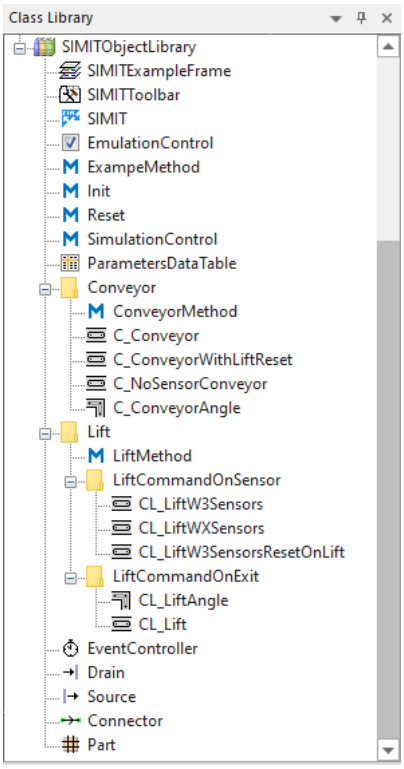


Figure E.1: The content of the emulation-prepared object library in Plant Simulation.

The "SIMITExampleFrame" is an example model built as a display for new library users to familiarize themselves with the concept and see what is needed for the hybrid function to work, see Figure E.2. It is built only with components from the library to simulate a lifting process, where the first and last conveyor are stationary, and the middle conveyor has the lift function. The model is prepared for the connection to a Shared Memory in SIMIT. The final step to enable the connection is to load or create a Shared Memory in SIMIT using the name

standard (Section *4.1.2 Name Standard*), activate the interface, and import the signals.

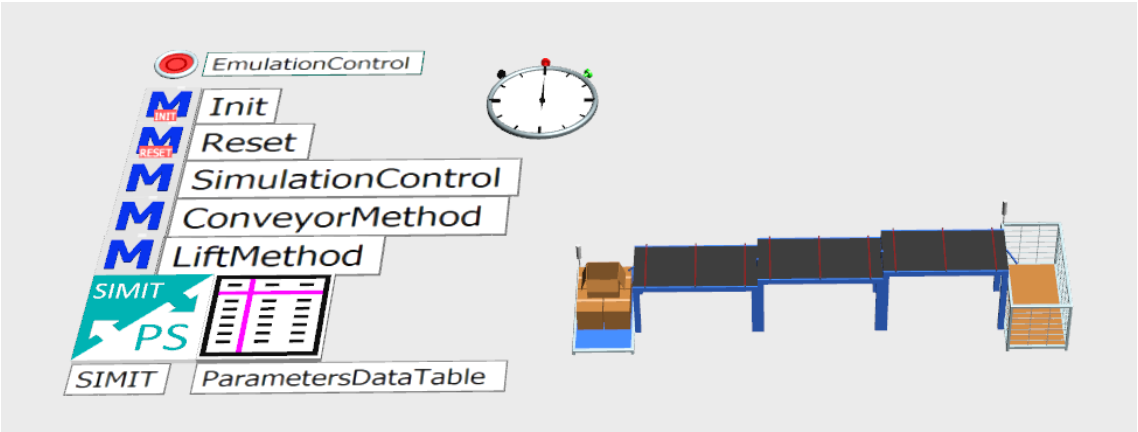


Figure E.2: Example frame of a system prepared for emulation control.

The following sections will describe the object library components and their function, divided into general methods and items, conveyors, and lifts.

General Methods

The checkbox in the library is named "EmulationControl" and is of the datatype Boolean, meaning either emulation control is false - emulation control is not active, regular simulation logic should control the model, or true - the simulation model should be controlled by emulation. The "ExampleMethod" was created as a template method, Figure E.3. Any new methods that the user wants to add are pre-coded with the if-else condition, checking if simulation or emulation is active. The same idea is used for the Reset method, Figure E.4, which runs on every model when reset initiates. The reset method controls the activation or deactivation of the SIMIT interface; if emulation signals can be sent or retrieved from SIMIT. A requirement for the activation is that the Shared Memory coupling name is written in the interface, and the SIMIT model is running simultaneously, Figure 4.6. Signals can now be imported, and the model can be emulated.

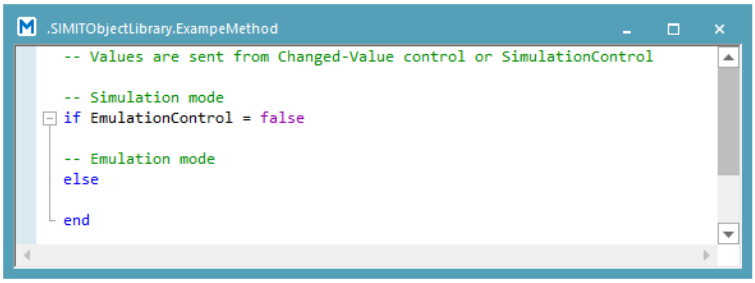


Figure E.3: Template method for model building, requires the checkbox "EmulationControl" in the same frame as the method.

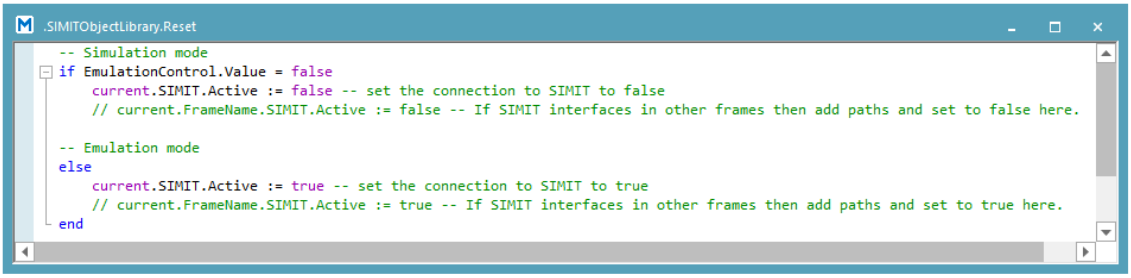


Figure E.4: The reset method used to activate or deactivate the SIMIT interface. Other reset conditions can be written under respective sections for simulation or emulation mode.

An Init method was also created that can be used to reset values when changing back and forth between simulation and emulation mode, Figure E.5. The Init method in Plant Simulation is triggered on start after a reset, not when the simulation only has been paused and restarted.

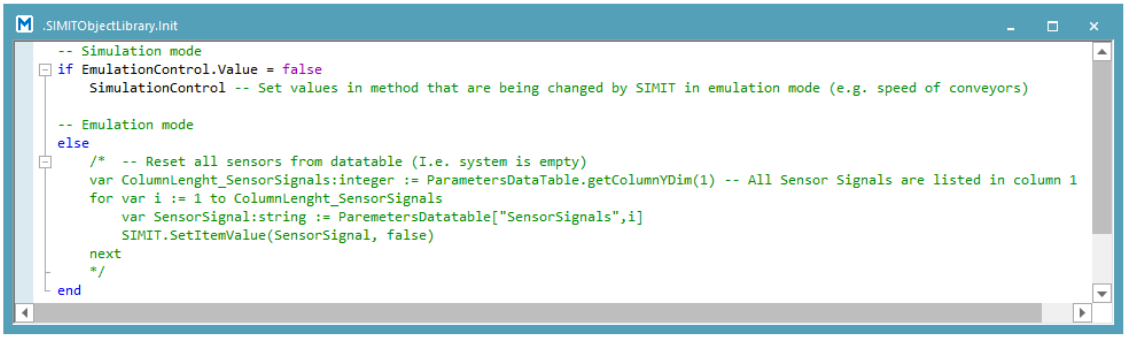


Figure E.5: Init method can be used for resetting values. Other init conditions can be written under respective sections for simulation or emulation mode.

There are two options for the reset method; option one is to write the code as presented under emulation control; here, the code resets all outgoing signals, in this case, sensors when the model is empty to ensure that no sensors are active in SIMIT when the Plant Simulation model is empty. Option two is to call on a method that contains code for setting the desired parameters. For the library, the "Simulation-Control" was created for that purpose, since when the model is run in simulation mode, more parameters are needed to be set in the model, Figure E.6 and E.7. This method can also be used for optimizing the simulation since the alternative code variants make parameters easily change on all objects.

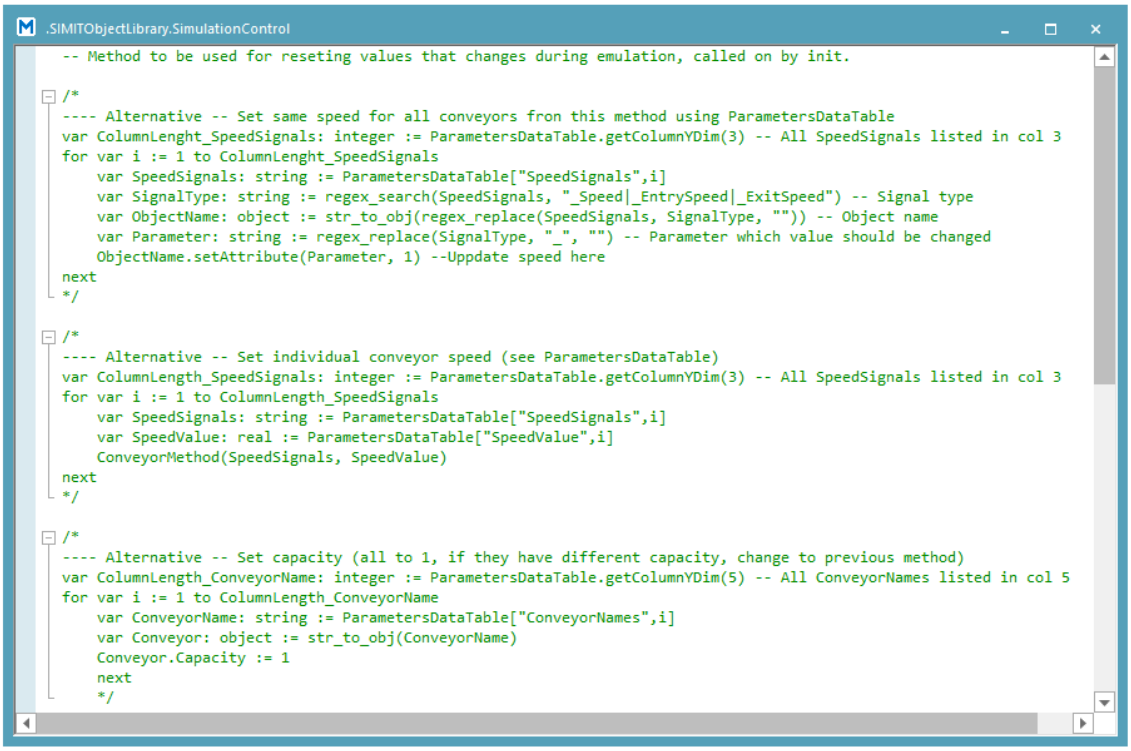


Figure E.6: Code alternatives for setting simulation parameters, part one.

Figure E.7: Code alternatives for setting simulation parameters, part two.

Both alternatives for setting and resetting values involves the use of a data table, Figure E.8. The idea is to gather all parameters and values in one place to have

easy access to them everywhere in the Plant Simulation model. In Plant simulation, values from this table can be retrieved and used in many ways. An example of how to use them can be seen in Figure E.6 and E.7. The data table can be manually created and edited for a small system and copied and pasted from other documents like Excel. Data tables in Plant Simulation also have an import function enabling tables to be created in Excel and then imported to the data table in Plant Simulation, the "ParametersDataTable". Important to note is that the code in the "SimulationControl" and "Reset" methods need to be adjusted if the order of the data table columns is changed so that the code in these methods is retrieving data from the right column.

It can be noted that it is possible to switch between the modes without the "Init" method resetting the values. The idea is then to perform a warm-up round where each station goes through by the programmer. This alternative takes a longer time than reading a reset method and was therefore neglected.

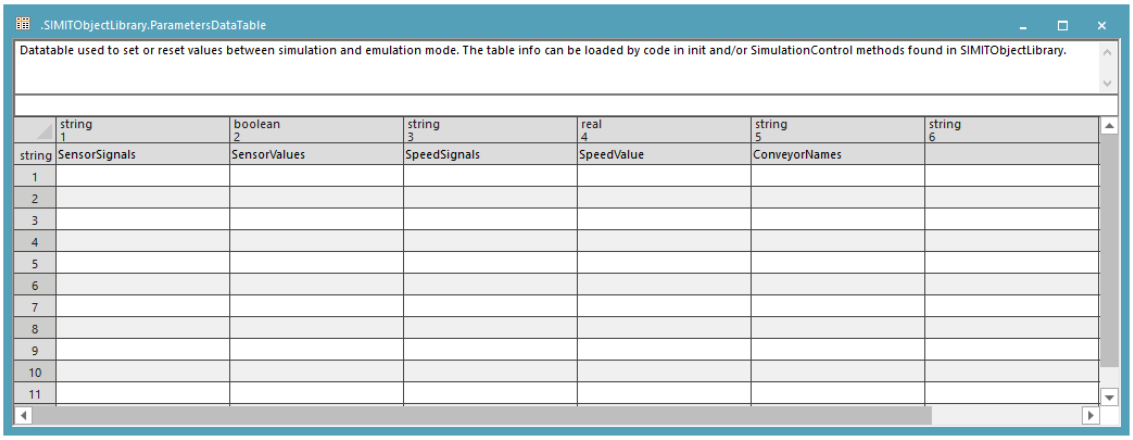


Figure E.8: Data table used in Plant Simulation to store model data.

Conveyors

With the Changed-Value control using methods, checkboxes, or global variables as control paths described in Section *4.1.1.3 Requirements on Software Connection*, to control conveyor parameters, a general conveyor method was created, Figure E.9. This method should be added under the Changed-Value control for all conveyor signals that should retrieve the signal's alias and value; an example of this see Figure E.10. As long as the name standard is followed when naming objects in the Plant Simulation model and the signals in the Shared Memory coupling SIMIT, the "ConveyorMethod" can find the object with a corresponding name with its code as the signal. The method can further set the parameters for those objects to the prefix of the signal, i.e., a signal named C_XXX_Speed with value 20 arrives in the method after being changed in SIMIT. Since the Changed-Value control calls on this method on signal value change, the ConveyorMethod code then splits the signal name into two variables, an object name C_XXX and a parameter Speed. These variables are then used to set the attribute of the object. This method can be used to change any changeable attribute for a conveyor. The theory is the same for any object in

Plant Simulation that can have a name as a designated path and an attribute that can be changed by SimTalk code.

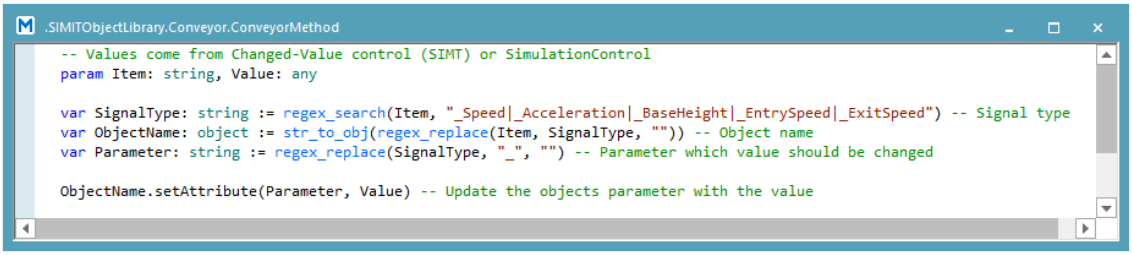


Figure E.9: The conveyor method that retrieves signal name and its value from SIMT and sets the value on the corresponding object parameter.

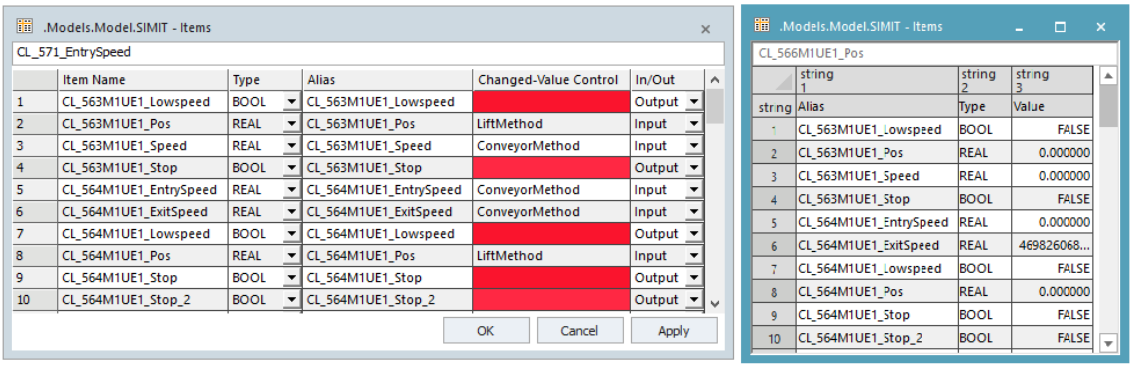


Figure E.10: The SIMIT interface in Plant Simulation, showing Items in the top figure and Show item values in bottom figure.

The basic conveyor library consists of three types of conveyors prepared for emulation, a basic conveyor with sensors (Figure E.13), a conveyor which in simulation mode have a lift reset on its sensors (Figure E.14), a conveyor without sensors (Figure E.15), and finally an angled conveyor (Figure E.16). The idea with the conveyor is that all methods should contain user-defined attributes that are pre-coded for the user to see where to add simulation logic and emulation commands, respectively. Most of the methods are the same for all conveyors; they have the same exit and entrance code, Figure E.11. The emulation mode has a mechanical constraint that stops a part from moving if the lift or conveyor after is higher, since the event in a real world would mean that the part crashed into the next object. The emulation part of the sensor code is also the same for all conveyors with sensors on them, Figure E.12. Their unique features can be found in each figure description. Model specific code and be added under all methods. Just be wary if the code is meant for simulation purposes or emulation.

.SIMITObjectLibrary.Conveyor.C_Conveyor.OnExit

```
-- Simulation mode
if EmulationControl.Value = false

--Emulation mode
else

end

-- Mechanical constraint that will stop the MU if successor can block its path
setEpsilon(0.0001) -- Tolerance for about equal
var Successor := regex_search(?.Succ.Name, "C_|CL_") -- Check if successor is a conveyor or lift
if Successor = "C_" or Successor = "CL_"
    if ?.BaseHeight < ?.Succ.BaseHeight -- If successor height is higher, MU stops
        @.Stopped := true
        repeat
            wait 0.01
        until ?.BaseHeight >~= ?.Succ.BaseHeight -- Height is equal or higher than successor MU can continue
        @.Stopped := false
    end
end

@.move
```

.SIMITObjectLibrary.Conveyor.C_Conveyor.OnEntrance

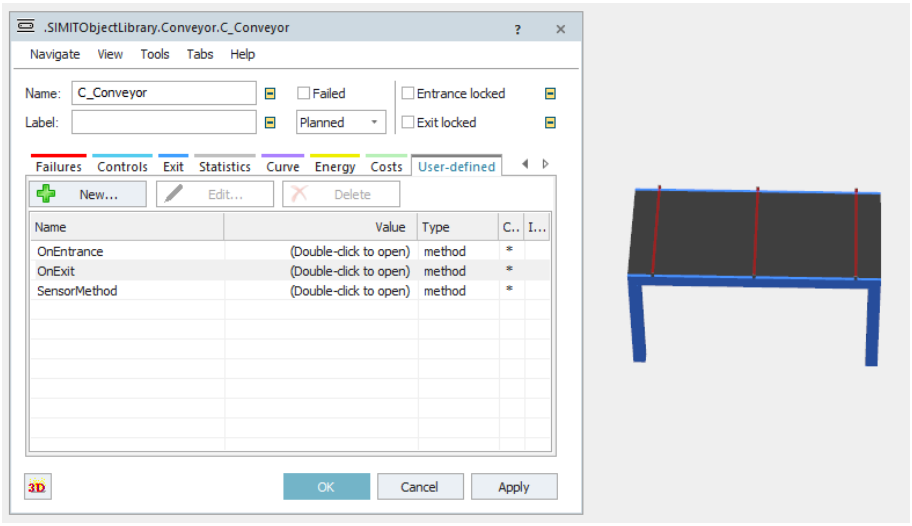
```
-- Simulation mode
if EmulationControl.Value = false

--Emulation mode
else

end
```

Figure E.11: "Entrance" and "Exit" methods for all conveyors and lifts, pre-coded with mechanical stop in emulation mode.

.SIMITObjectLibrary.Conveyor.C_Conveyor.SensorMethod

```
-- Emulation mode
else -- Send sensor value to SIMIT
    if SensorID = 1
        // Sensor code
    elseif SensorID = 2
        // Sensor code
        if Front = true
            SIMIT.SetItemValue(?.Name+"_Lowspeed", true)
        else
            SIMIT.SetItemValue(?.Name+"_Lowspeed", false)
        end
    elseif SensorID = 3
        // Sensor code
        if Front = true
            SIMIT.SetItemValue(?.Name+"_Stop", true)
        else
            SIMIT.SetItemValue(?.Name+"_Stop", false)
        end
    end
end
```

Figure E.12: Second part of the sensor code on all conveyors, the code sets Boolean expressions that tells SIMIT if there is a part at the sensor or not.



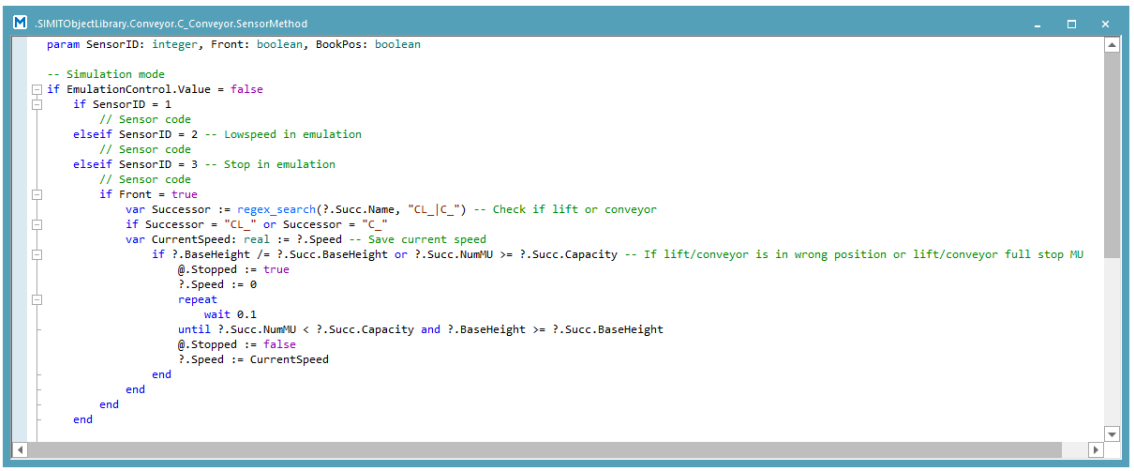


Figure E.13: Basic conveyor with three sensors which stops at sensor three if the object after is a conveyor or lift, and if the next object is higher or lower than itself, or if the conveyor after is full. The part continues when all of these statements are false. For other user-defined attributes see Figure E.11 and E.12. If it would have been desirable to have more than three sensors, it would be simple to add a new sensor, connect it to the existing sensor method, and add the extra logic in the method.

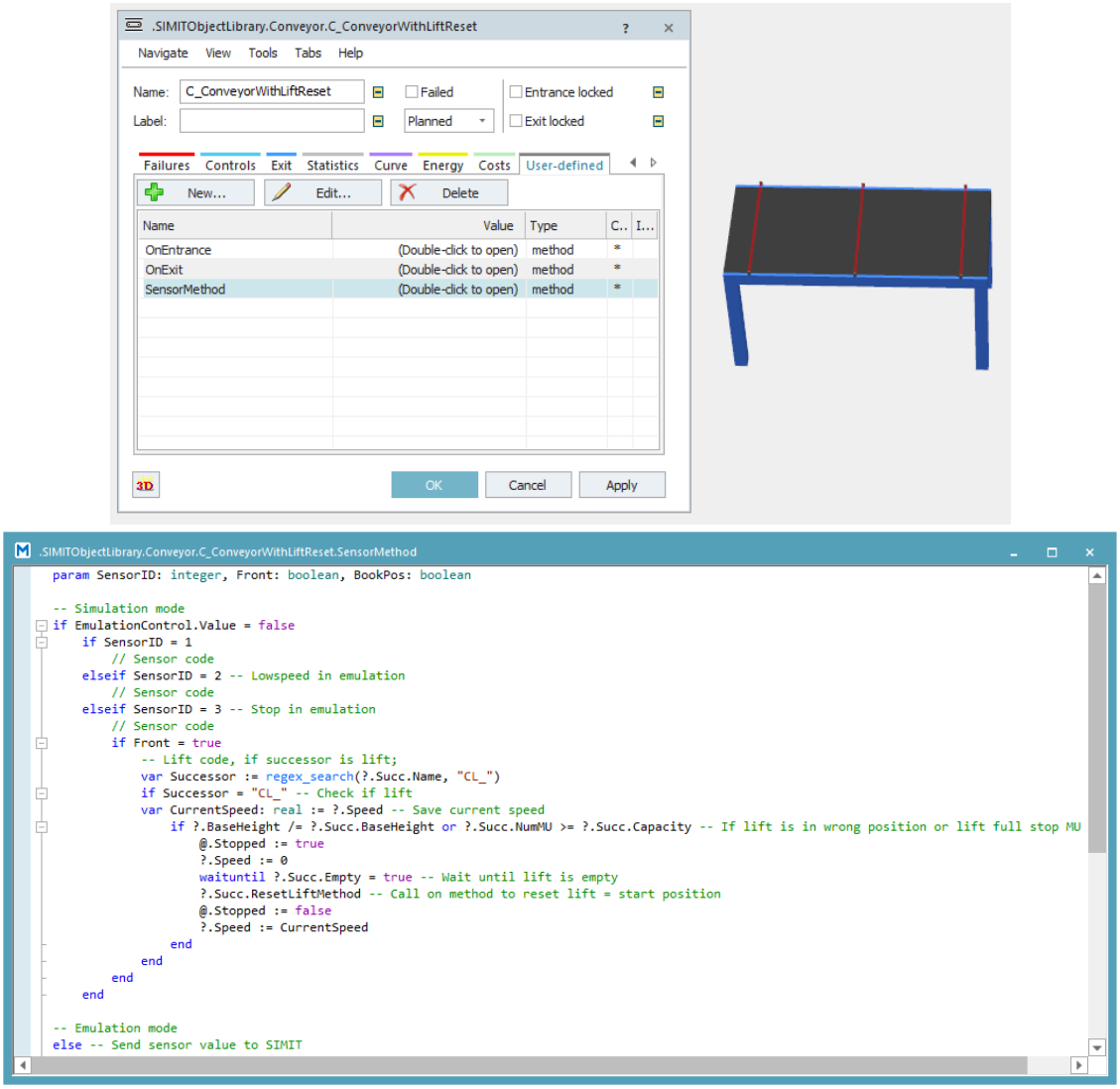


Figure E.14: Basic conveyor with three sensors which stops at sensor three if the object after is a lift, and this object is higher or lower than itself, or if the lift is full. It then calls on a lift method, located on the lift, to retrieve it. When the lift is in the same height as the conveyor and has free capacity, the product continues. For other user-defined attributes see Figure E.11 and E.12.

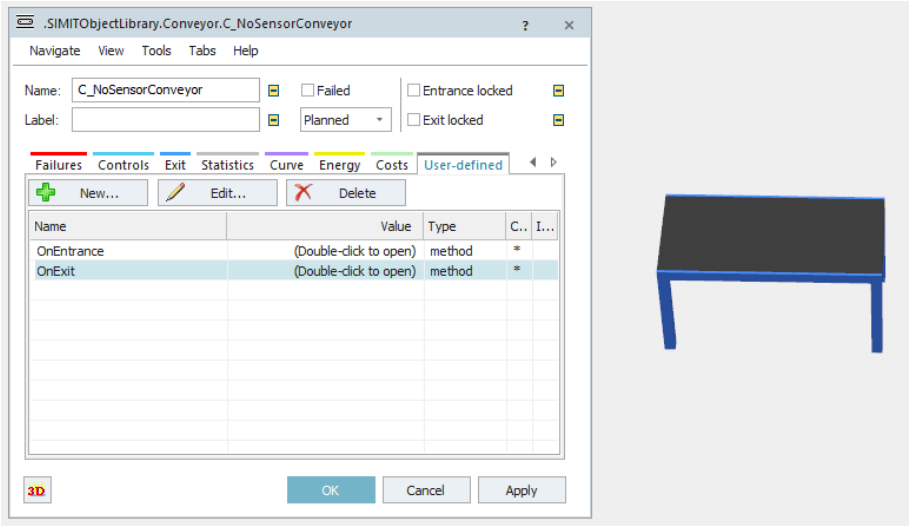


Figure E.15: Basic conveyor without any sensors. For other user-defined attributes see Figure E.11.

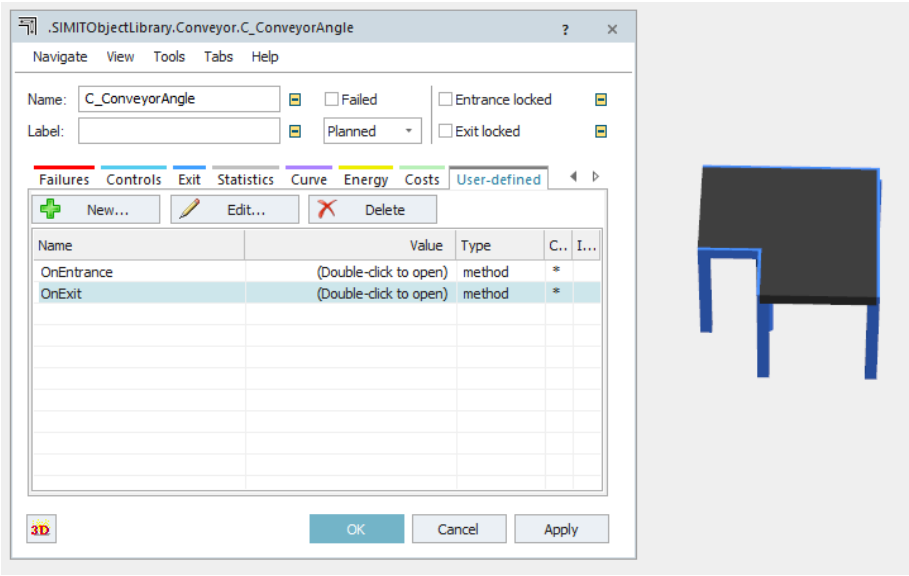


Figure E.16: Basic angled conveyor without any sensors. For other user-defined attributes see Figure E.11.

Lifts

To build the test model lifts wherever needed, lifts were developed using conveyors as the base and changing their base-height using simulation code or position values from an emulation model's lift component. The principle in simulation mode is to control the lift with user-defined attributes. Five different lifts were created, to have a base library of different lift components, divided into two categories of lifts with the lift command on a sensor (Figure E.17), or on an exit control (Figure E.18). Individually, a lift with three sensors (Figure E.19), a lift with X numbers of sensors (Figure, E.20), a lift with three sensor and a reset on the lift (Figure, E.21), an

angled lift (Figure E.22), and a basic lift without any sensors (Figure E.23).

.SIMITObjectLibrary.Lift.LiftCommandOnSensor.CL_LiftWXSensors.SensorMethod

```
param SensorID: integer, Front: boolean, BookPos: boolean

-- Simulation mode
if EmulationControl.Value = false
    if SensorID = 1
        // Sensor code
    elseif SensorID = 2 -- Lowspeed in emulation
        // Sensor code
        ?.LiftMethod -- Start lift
    elseif SensorID = 3 -- Stop in emulation
        // Sensor code
    //Cont...
    end

-- Emulation mode
else
    if SensorID = 1
        // Sensor code
    elseif SensorID = 2
        // Sensor code
    elseif SensorID = 3
        // Sensor code
    // Cont...
    end
end
```

Figure E.17: The sensor code for lifts with sensors, which in this code initiates the lift code on when the part reaches sensor three. More sensor code can be added under respective sections, as well as more sensors.

.SIMITObjectLibrary.Lift.LiftCommandOnExit.CL_LiftAngle.OnExit

```
-- Simulation mode
if EmulationControl.Value = false
    -- Start lift
    ?.LiftMethod

-- Emulation mode
else

-- Mechanical constraint that will stop the MU if successor can block its path
setEpsilon(0.0001) -- Tolerance for about equal
var Successor := regex_search(?.Succ.Name, "C_|CL_") -- Check if successor is a conveyor or lift
if Successor = "C_" or Successor = "CL_"
    if ?.BaseHeight < ?.Succ.BaseHeight -- If successor height is higher, MU stops
        @.Stopped := true
        repeat
            wait 0.01
        until ?.BaseHeight >~= ?.Succ.BaseHeight -- Height is equal or higher than successor MU can continue
        @.Stopped := false
    end
end

end

@.move
```

Figure E.18: Exit control code for lifts that initiated movement on-exit, "LiftMethod" is then call upon and initiated.

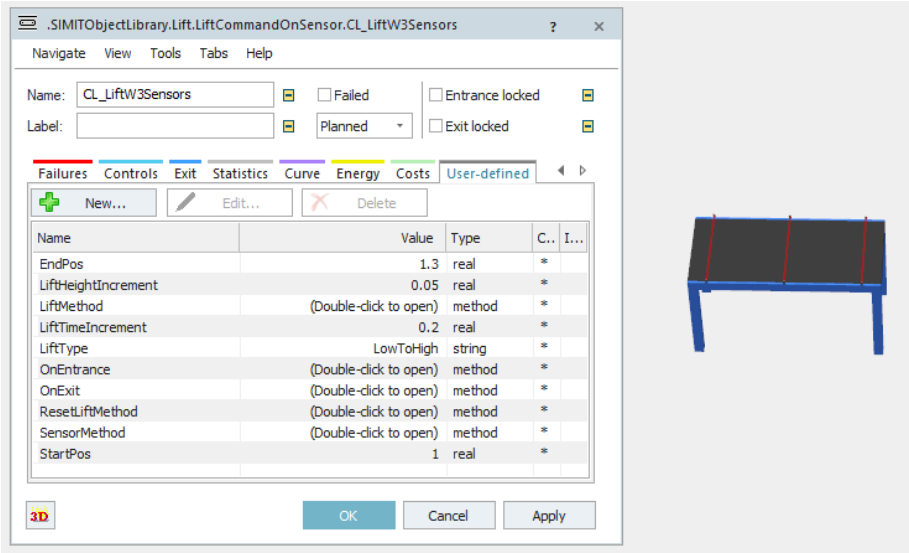


Figure E.19: Lift with three sensors, reset is done on the conveyor before. Sensor code as in Figure E.21 and entrance and exit code as in Figure E.11.

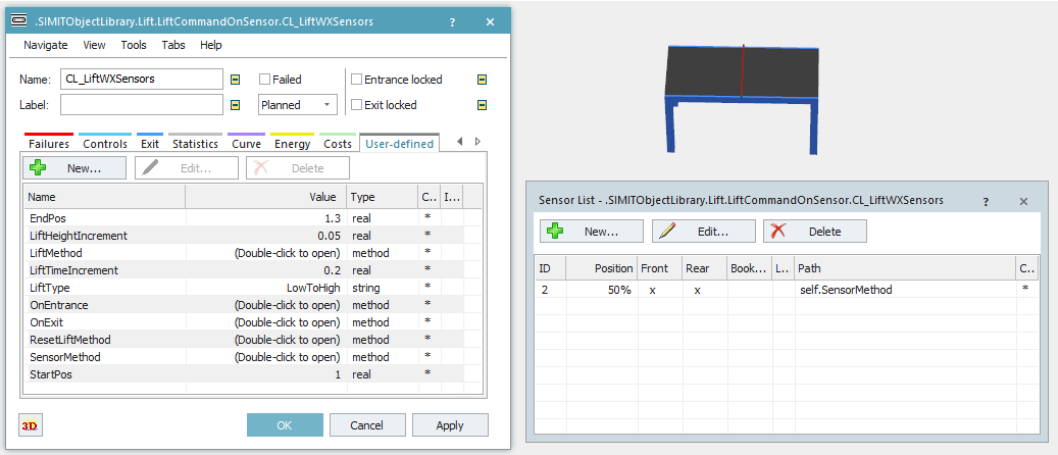
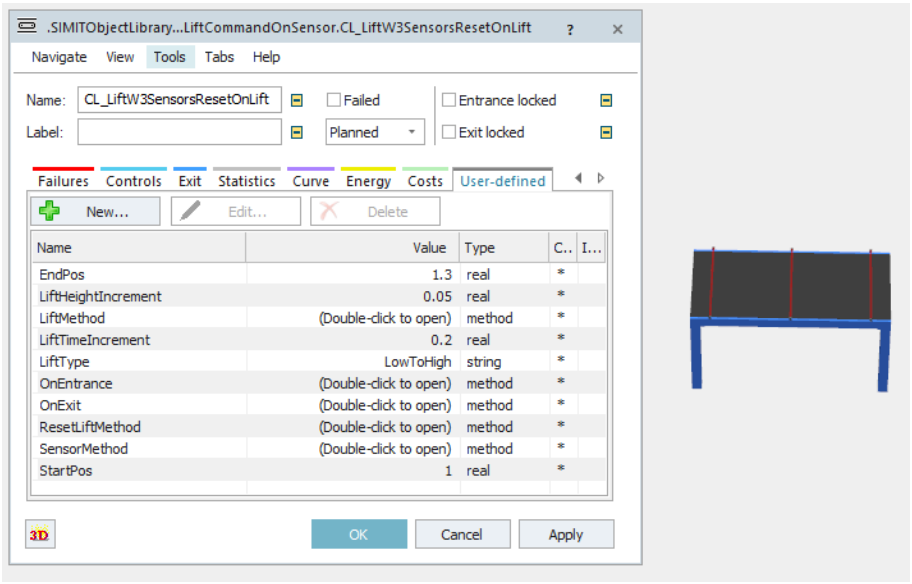


Figure E.20: Lift with an unlimited number of sensors. One sensor is added to begin with. The sensor code is similar to Figure E.21 and entrance and exit code similar to Figure E.11.



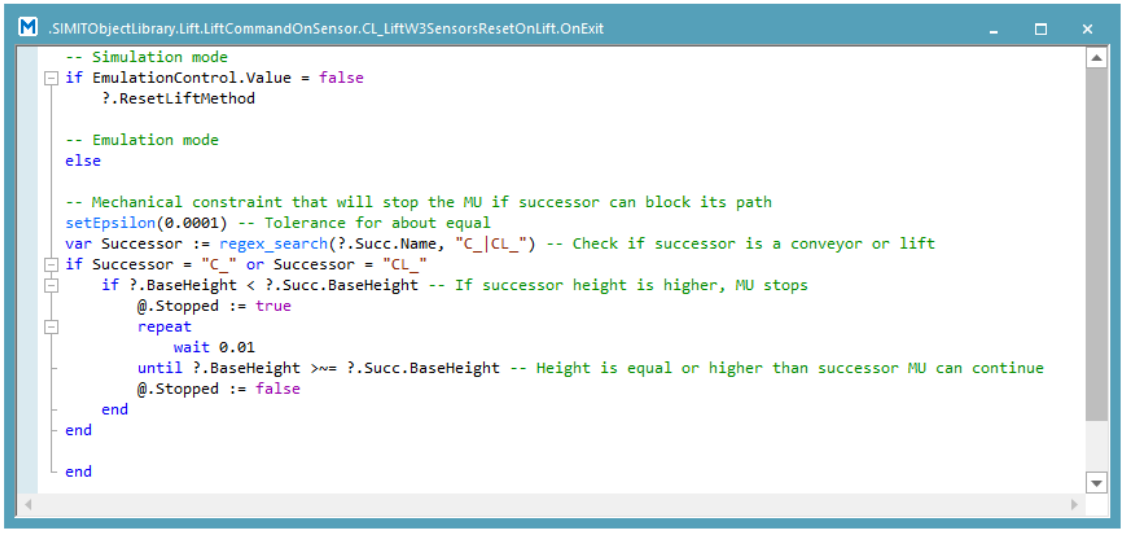


Figure E.21: Lift with three sensors, similar to Figure E.19, however a rear triggered exit control controls the reset, meaning that when the part has exited, the lift resets.

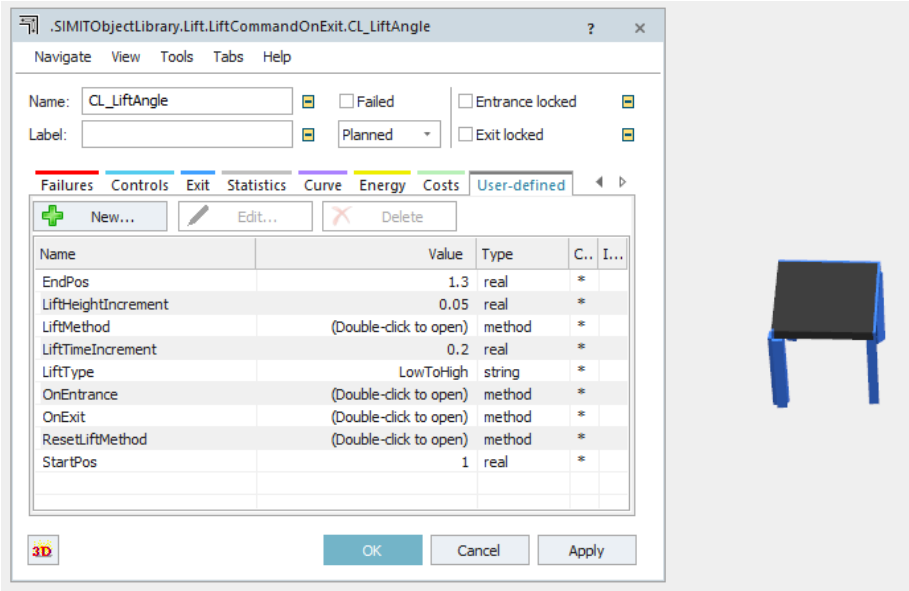


Figure E.22: Basic angle lift, with the lift control in the exit control, Figure E.18.

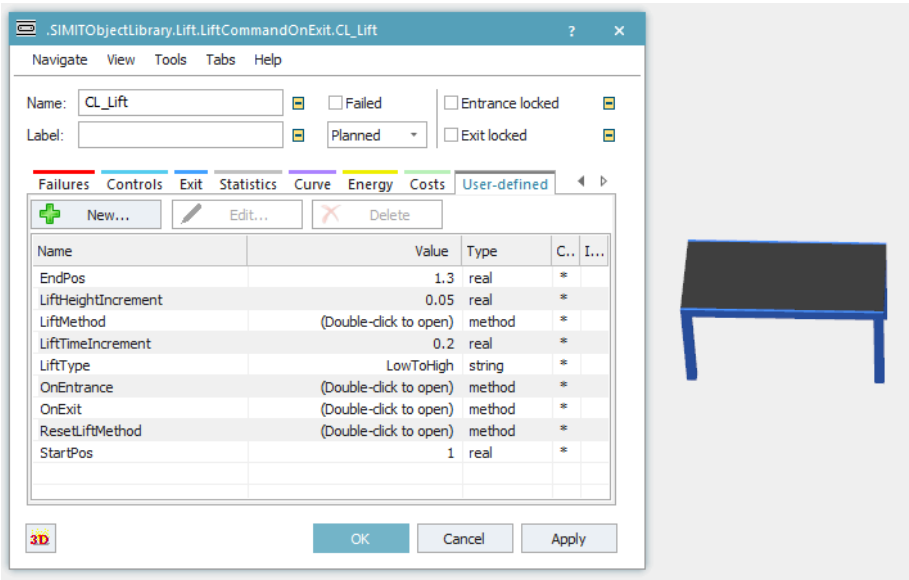


Figure E.23: Basic lift without sensors, lift control in the exit control, Figure E.18.

All lifts have the same lift method "LiftMethod" and reset lift method "ResetLift" code on them, Figure E.24. Having separate lift codes provides the possibility to initiate their function from different places within the simulation model. What varies from lift to lift is where these methods are called, described under the figure texts for each lift object. Model-specific code can be added under exit and entrance code. Just be wary if the code is meant for simulation or emulation purposes. All lifts with sensors share the same entrance and exit code as the conveyors, Figure E.11. There is only one lift that has a reset on the lift itself (Figure E.21). The other lifts

needs to be reset on the conveyor or object before, using the conveyor in Figure E.14.

.SIMITObjectLibrary.Lift.LiftCommandOnSensor.CL_LiftWXSensors.LiftMethod

```
if Self.~.BaseHeight < Self.~.Succ.BaseHeight or Self.~.BaseHeight > Self.~.Succ.BaseHeight -- MU stops if successor is higher
    var CurrentSpeed: real := ?.Speed -- Save current speed
    @.Stopped := true
    ?.Speed := 0
    waituntil Self.~.NumMU = Self.~.Capacity -- Wait until lift full

    -- Raise lift
    if Self.~.LiftType = "LowToHigh"
            -- Lift speed: LiftTimeIncrement*(?.Succ.BaseHeight-?.BaseHeight)/LiftHeightIncrement [s]
            repeat
                Self.~.BaseHeight := Self.~.BaseHeight+Self.~.LiftHeightIncrement -- Lift increment [m]
                wait Self.~.LiftTimeIncrement -- Lift increment [s]
            until Self.~.BaseHeight >= Self.~.Succ.BaseHeight
            Self.~.BaseHeight := Self.~.Succ.BaseHeight -- Matching height to successor

    -- Lower lift
    elseif Self.~.LiftType = "HighToLow"
        -- Lift speed: LiftTimeIncrement*(?.BaseHeight-?.Succ.BaseHeight)/LiftHeightIncrement [s]
        repeat
            Self.~.BaseHeight := Self.~.BaseHeight-Self.~.LiftHeightIncrement -- Lift increment [m]
            wait Self.~.LiftTimeIncrement -- Lift increment [s]
        until Self.~.BaseHeight <= Self.~.Succ.BaseHeight
        Self.~.BaseHeight := Self.~.Succ.BaseHeight -- Matching height to successor
    end

    @.Stopped := false
    ?.Speed := CurrentSpeed
end
```

.SIMITObjectLibrary.Lift.LiftCommandOnSensor.CL_LiftWXSensors.ResetLiftMethod

```
  -- Lower lift to starting position
if Self.~.LiftType = "LowToHigh"
    repeat
        -- Lift speed: LiftTimeIncrement*(?.BaseHeight-?.Pred.BaseHeight)/LiftHeightIncrement [s]
        Self.~.BaseHeight := Self.~.BaseHeight-Self.~.LiftHeightIncrement -- Lift increment [m]
        wait Self.~.LiftTimeIncrement -- Lift increment [s]
    until Self.~.BaseHeight <= Self.~.Pred.BaseHeight
    Self.~.BaseHeight := Self.~.Pred.BaseHeight -- Matching height to successor

  -- Raise lift to starting position
elseif  Self.~.LiftType = "HighToLow"
    repeat
        -- Lift speed: LiftTimeIncrement*(?.Pred.BaseHeight-?.BaseHeight)/LiftHeightIncrement [s]
        Self.~.BaseHeight := Self.~.BaseHeight+Self.~.LiftHeightIncrement -- Lift increment [m]
        wait Self.~.LiftTimeIncrement -- Lift increment [s]
    until Self.~.BaseHeight <= Self.~.Pred.BaseHeight
    Self.~.BaseHeight := Self.~.Pred.BaseHeight -- Matching height to successor
end
```

Figure E.24: The "LiftMethod" and "ResetLift" method, that comes as a user-defined attribute on each lift. The top figure is the "LiftMethod" and the lower is the "ResetLiftMethod".

In emulation mode, the lifts are controlled by the code in Figure E.25. This means that for each value change in SIMIT the position (height) of the lift will change. This requires the SIMIT component so send a height in meter as value since that is the unit on an object's base-height. This method needs to be added in the Changed-Value control for each lift or object.

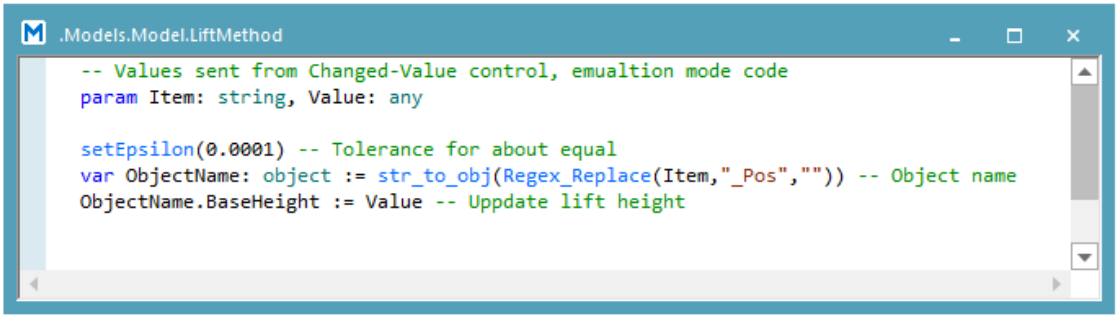


Figure E.25: Emulation code for controlling the lifts, it receives the alias name under Item and alias value under Value. Then modifies the alias into the corresponding object name to the signal, and sets the base-height for the object to the value.

DEPARTMENT OF INDUSTRY AND MATERIALS SCIENCE
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY