Aalto-yliopisto
Insinööritieteiden
korkeakoulu

Joel Isometsä

Component Classification and Best Practices in Production Simulation Modeling

Master's Thesis, which has been given in for a degree of Master of Science in Technology.

Espoo 4 March 2018
Supervisor: Professor Esko Niemi
Instructor: Samuli Ahonen

Aalto-yliopisto
Insinööritieteiden korkeakoulu

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi

Diplomityön tiivistelmä

Tekijä Joel Isometsä

Työn nimi Komponenttien luokittelu ja parhaat käytännöt tuotantosimulaation mallinnuksessa

Koulutusohjelma Konetekniikan koulutusohjelma

Pääaine Koneensuunnittelu **Koodi** K3001

Työn valvoja Prof. Esko Niemi

Työn ohjaaja DI Samuli Ahonen

Päivämäärä 4 maaliskuu 2018 **Sivumäärä** 60+26 **Kieli** englanti

Tiivistelmä

Tuotantosimulaatio on tärkeässä osassa tuotantojärjestelmien validoinnissa, optimoinnissa ja visualisoinnissa. Tuotantosimulaation toiminta perustuu yleisesti komponentteihin ja niiden väliseen vuorovaikutukseen. Komponentit esittävät tyypillisesti tehtaasta löytyviä laitteita ja esineitä, mutta komponentteja voidaan käyttää myös visualisointiin, statistiikan keräämiseen, järjestelmän ohjaukseen tai muuhun tarpeeseen simuloinnissa.

Tämän diplomityön tavoitteita oli kehittää komponenttiluokkia teollisuudesta valittujen laitteiden perusteella, mikä mahdollistaa mallinnusratkaisujen standardoinnin. Sen lisäksi tavoitteena oli kehittää parhaat käytännöt komponenttimallinnukseen. Muita tavoitteita oli tunnistaa ja analysoida tulevaisuuden näkymiä tuotantosimulaatiolle. Tämä keskittyi pääosin digitaaliseen kaksoseen, jota voidaan kuvata reaaliaikaisesti peilautuvaksi simulaatiomalliksi todellisesta järjestelmästä. Tämän lisäksi työssä keskityttiin formaaleihin mallinnuskieliin.

Diplomityön lopputulos esittää kehitetyt komponenttiluokat ja parhaat käytännöt komponenttimallinnuksessa. Komponenttien luokittelussa keskityttiin kehittämään geneerisiä komponentteja, joita voidaan ohjata signaalipohjaisilla komennoilla. Tämä mahdollistaa komponentin ohjaamisen myös simulointiohjelman ulkopuolelta. Tämän lisäksi automaattista komponenttien luomistyökalua käytettiin luokiteltujen komponenttien luomisessa. Parhaat käytännöt komponenttimallinnuksessa pohjautuivat mallinnuksen oleellisimpiin osa-alueisiin tavanomaisissa mallinnustilanteissa. Parhaiden käytäntöjen kehityksessä haastateltiin simulointiammattilaisia, joiden mielipiteistä muodostettiin perusta käytäntöjen kehitykselle.

Avainsanat Simulointi, Diskreetti Tapahtumapohjainen Simulointi, Komponenttimallinnus, Digitaalinen Kaksonen, Komponenttiluokittelu, Parhaat Käytännöt, Mallinnuskielet

Author Joel Isometsä

Title of thesis Component Classification and Best Practices in Production Simulation Modeling

Degree programme Programme of machine design

Major Machine design **Code** K3001

Thesis supervisor Prof. Esko Niemi

Thesis advisor M.Sc. Samuli Ahonen

Date 4 March 2018 **Number of pages** 60+26 **Language** English

Abstract

Production simulation software plays a major role in validation, optimization and illustration of production systems. Operation of production simulation is generally based on components and their interaction. Components typically represent factory floor devices, but in addition, there can be components to provide visualization, statistics, control or other input to simulation. The demand for having high-quality, easy-to-use and compatible components emphasizes the importance of component modelling.

The objectives of this thesis were to develop component classes based on industrial devices, to standardize component modelling solutions and best practices in component modelling. Other objectives were to identify and analyse future prospects of production simulation. This focuses on the concept of digital twin, which could be described as reflective real-time simulation model from the physical system. In addition, focus is also set on formal modelling languages.

The outcome of this thesis presents component classes and best practices in component modelling. In component classification, the focus was set to development of generic components, which can be controlled with signal-based logic. This enables components from the software to be externally controlled. In addition, automatic model creation tool wizard, is implemented to instantly generate components based on the defined component classes. Best practices were based on the selected modelling fields that are most relevant for general use. In the development of best practices, interviewing method was utilized to receive input from simulation experts.

Keywords Simulation, Discrete-Event Simulation, Component Modelling, Digital Twin, Component Classification, Best Practices, Formal Modelling Languages

Preface

This master's thesis was carried out in co-operation with Visual Components Ltd. The thesis has been partially funded under the European Commission's H2020 framework program in the project Factory2Fit "Empowering and Participatory Adaptation of Factory Automation to Fit for Workers". The academic examination of this thesis was conducted by the Department of Mechanical Engineering at Aalto University of Technology.

I would like to thank my supervisor, Professor D.Sc. Esko Niemi, and my advisor, M.Sc. Samuli Ahonen for guidance and constructive feedback to my thesis. I would also like to thank M.Sc Fernando Ubis for general assistance and valuable feedback.

In addition, I would like to sincerely thank Visual Components for providing this opportunity, and my colleagues who have supported me to thorough this project. I would like to particularly thank my colleague Tomi Syväjärvi for detailed proofreading of this thesis. Special thanks belong to my family for proofreading and general support during the thesis.

Espoo, Finland, 4 March 2018

Joel Isometsä

Table of contents

List of Figures

Abbreviations

AutomationML	Automation Markup Language
FBD	Function Block Diagram
CAD	Computer Aided Design
CAE	Computer Aided Engineering
CAEX	Computer Aided Engineering Exchange
COLLADA	COLLAborative Design Activity
CPS	Cyber-Physical System
CPPS	Cyber-Physical Production System
DES	Discrete-Event Simulation
DTI	Digital Twin Instance
DTP	Digital Twin Prototype
DTE	Digital Twin Environment
EDT	Experimentable Digital Twin
ERP	Enterprise Resource Planning
FFS	Form-Fill-Seal
ICT	Information and Communication Technologies
IL	Instruction List
IML	Intermediate Modeling Layer
INCOSE	The International Council on Systems Engineering
ISO	International Organization for Standardization
LD	Ladder Diagram
MBSE	Model-Based Systems Engineering
MES	Manufacturing Execution System
OMG	Object Management Group
PLC	Programmable Logic Controller
PLM	Product Lifecycle Management
P&F	Power and Free
SFC	Sequential Function Chart
ST	Structured Text
STEP	Standard for the Exchange of Product Model Data
SysML	Systems Modeling Language
UML	Unified Modeling Language
UR	Unified Repository
RFID	Radio-Frequency Identification
VC	Visual Components
XML	Extensible Markup Language

1 Introduction

1.1 Background

Computer simulation can be used to reproduce physical phenomena and behaviors with a mathematical model. This model represents key characteristics of an object or a system it is reflecting. The model is executed on a computer, that exceeds the human capabilities on numerical computing and data processing.

Simulation can be applied to a large number of use cases, which leads to the development of several types of simulation software (Mourtzis, et al., 2014). Simulation has become important tool in the production domain for the design, analysis and visualization of manufacturing systems. Discrete-event based simulation tools, have established their position as a standard method of validating and optimizing manufacturing systems. The operation of these tools is based on simulation components and their interaction. Typically, these components resemble the physical devices on the factory floor. (Modrak and Semanco, 2014, p. 89-115)

The process of creating a simulation of a manufacturing system, starts with the creation of a virtual model of the system. This virtual model is the digital representation of the real model in the simulation environment. Component modeling plays a key role in the use of production simulation. Accurate and appropriate modeling of the components determines the validity of the results obtained during the simulation. Modern simulation tools allow to reuse components from the component libraries, which facilitates the creation of new simulations.

Productivity has been dramatically increased by the different industrial revolutions. From the introduction of steam powered machines to the electrification of factories, resulting to the emergence of automation. The introduction of information and communication technologies (ICT), pushed the development of automation technologies to a new level.

Digitalization and simulation technologies have been developing in parallel during the evolution of the ICT technologies. Currently, we are in the midst of a fourth wave of industrial revolution. Generally accepted name for the fourth industrial revolution is Industry 4.0. It focuses on flexible and autonomous manufacturing systems, that intelligently exploit the relevant information from massive amounts of data. All information systems and data will be stored in a cloud, where it can be accessed from anywhere. (Lasi, 2014) (Rüssmann, 2015)

Simulation has become one of the nine pillars for building the Industry 4.0, supporting and enabling the digital twin in the virtual space. The digital twin reflects a physical object or a system in real-time, and it is capable to give feedback, such as optimized values and control signals to the physical counterpart. The twin integrates all the relevant data and simulation models of a physical counterpart into one entity, and the twin becomes the center point of engineering processes. Engineering processes will change from documentation-based to model-centric approach, where the importance of simulation increases. (Grieves, 2014) (Rüssmann, 2015)

1.2 Objectives

This research focuses on component modeling for Visual Components (VC) production simulation software. To achieve this wide target, it has been divided into four objectives. First objective is to identify and analyze future prospects in production simulation domain.

Second objective targets the classification of simulation components. Classification is a method to standardize modeling solutions. To achieve this, selected devices from pre-defined industrial fields have been used as a basis for the classification. Component classification has not been previously performed with VC software, which leads to the development of a classification method as well.

Third objective is to develop best practices in component modeling. Best practices are guidelines for the end-users in the most relevant fields related to modeling. Best practices provide proven to be good modeling solutions and standardization of modeling practices to achieve standardized and compatible components that are easy to adopt.

Fourth objective is to implement a case example, which utilizes and combines the results from component classification and best practices. Selected implementation type is a wizard tool that instantly generates predefined and possibly customizable set of attributes to applied component, which enables fast and standardized creation of components.

1.3 Scope

The examination of the future prospects focuses on the concept of digital twin and on formal modeling languages. The examination of digital twin is based on the selected modeling approaches available in the scientific literature.

The classification of components is based on the industrial devices from the automotive and the food & beverage industries, which were pre-defined by Visual Components. In automotive industry, focus is set to all manufacturing areas from chassis manufacturing to finalized product. In food & beverage industry, focus is emphasized to packaging area, which is typical target for simulation. In both industries, the selection of the devices is limited to most relevant and generally utilized devices.

The best practices are presented as general guidelines, and no further modeling instructions are presented to implement best practices. Best practices are based on the selected topics, which are selected based on the most relevant fields from the general modeling point of view.

The development of wizard functions as a proof of concept and prototype for the developed classes and best practices. This leads to the exclusion of comprehensive testing and finalization of the wizard.

1.4 Structure of Thesis

Chapter 2 first presents the concept of production simulation software. Thereafter, the concept and modeling approaches of the digital twin are presented. At the end of this chapter, the existing formal modeling languages are presented. Chapter 3 first presents automotive industry and the selected industrial devices. Thereafter, the chapter presents food & beverage industry and the selected industrial devices. Chapter 4 presents the component modeling in

the Visual Components software. After that, the development opportunities and approaches are presented. Chapter 5 presents and analyzes the result of the implementations. Chapter 6 presents the outcome of the thesis including conclusions and suggestions for further work.

2 Digital Modeling and Manufacturing

2.1 Production Simulation

Production simulation software are based on discrete-event simulation (DES). DES utilizes discrete time sequenced events to model the operation of a system. Events occur at specific time instant, and no change in the system between consecutive events is assumed to occur. Because DES is based on the discretely occurring events and not continuous sampling over time, it is generally faster to run than continuous simulations.

Production simulation cannot be purely defined as DES software, because some functionalities require continuous simulation as well. One example of continuous simulation is visualization of the events, which is based on sampling over simulation time. Some functionalities are based on this continuous visualization, such as collision detection and certain types of sensors. Other example is the possibility to use physics simulation, which is integrated to the production simulation. This enables the use of forces and their interaction between the components during the simulation (see Figure 2.1).

Layouts in production simulation are constructed with components, which typically represents factory floor devices. These components can be imported from component libraries or they can be created from the beginning. These components and their operation and interaction define the factory floor operations.

Production simulation software can be used for several purposes. As an example, it can be used to find bottlenecks from the production system, validate and optimize production lines, validate programmable logic controller (PLC) logics and visualize manufacturing solutions as a proof of concept. Figure 2.2 presents a scenario in which simulation is utilized to simulate complex welding process in automotive industry.

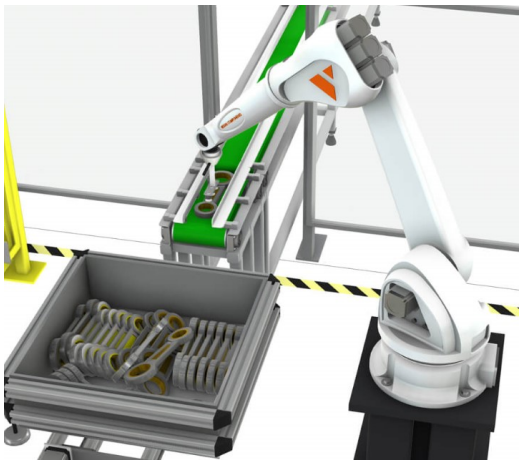

Figure 2.1. In addition to discrete-event simulation, continuous simulation with physics engine can be utilized, for example, in bin picking process

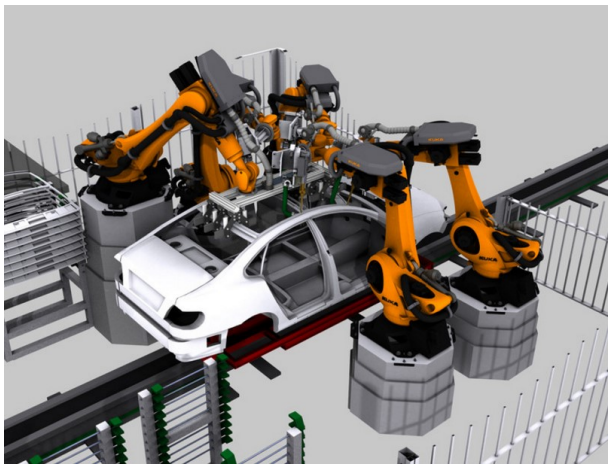

Figure 2.2. Simulation of a welding process in automotive industry.

Virtual Commissioning in Production Simulation

Commissioning is the process of testing and verifying system functions, and comparing them to designed requirements and specifications. Commissioning of the system can be accomplished without the use of simulation, but it can be effectively started only when the system to be commissioned is implemented. In this case, every undiscovered problem may delay the start of production. (Syväjärvi, 2016, p.4-8)

Virtual commissioning utilizes the use of simulation in the validation of the real control systems, such as programmable logic controllers (PLC) (Syväjärvi, 2016, p.4-5). This enables performing significant amount of commissioning activities before the actual system exist. With the use of virtual commissioning, even 75% of the time used for real commissioning, can be reduced due to enhanced quality of the manufacturing system (Koo, et al., 2011).

In system development, commissioning is the final part that results in a fully operational and tested system, ready to be used and delivered to customer. In practice, the commissioning of automation systems includes various procedures to check, inspect and test every operational component of the system. This includes everything from physical fit of components, connections of electrical wiring to correct operation of work cells and the system as a whole. (Liu, et al., 2012)

2.2 Digital Twin

The first section introduces the digital twin and cyber-physical systems. Second section presents modeling approaches for the digital twin. In third section, the use of digital twins in simulations is examined. Last section discusses the current challenges related to digital twins and their applications.

2.2.1 Introduction to Digital Twins

The Concept of Digital Twin

The concept of digital twin was first introduced by Grieves at one of his presentations about product lifecycle management in 2003 at University of Michigan. According to Grieves, digital twin contains three parts: physical product in real space, virtual product in virtual space and the connections of data and information, which ties the virtual and real product together (see Figure 2.3). Virtual product is presented as a rich representation of a product that is virtually indistinguishable from its physical counterpart. (Grieves, 2014)

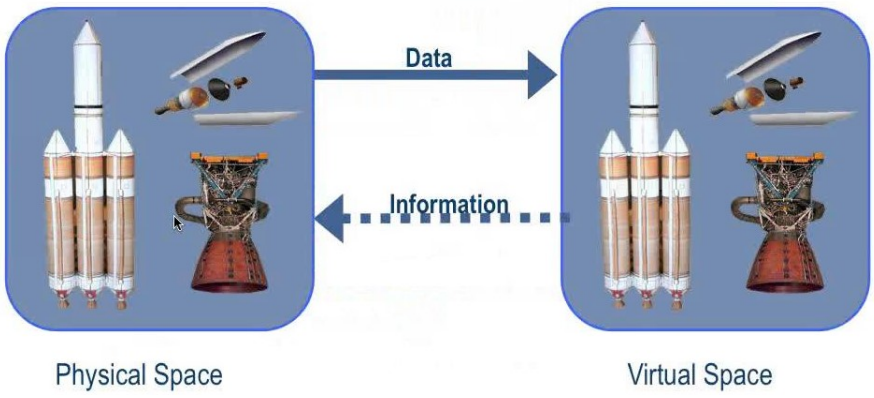

Figure 2.3. The concept of digital twin. (modified from Grieves, 2014)

Virtual and physical model can be simultaneously viewed and compared providing several benefits, especially in the manufacturing point of view. Instead of simulating what should happen on the factory floor, digital twin replicates what has actually happened, at every step in manufacturing process. This communication would happen real-time or near real-time, and it would transfer information, such as raw sensor data from physical object to virtual object. Information flow is not only one directional, but it can also be transferred from virtual object to physical object. This information can be, for example, data used for device control.

The digital twin capacity supports three powerful tools in the human knowledge kit. These tools are conceptualization, comparison and collaboration. Humans prefer to visually conceptualize situations, rather than look at a table of numbers, reports or other symbolic information. The digital twin enables information to be visually reviewed. Comparison is efficient tool of reviewing products. Physical product information can be visually and effectively compared to ideal characteristics of virtual product information. Collaboration provides more expertise, more variability of perspectives and improved problem solving capabilities. Digital twin allows sharing ideas and conceptual designs, which can be easily distributed between the different shareholders to be visualized, analyzed and improved. This allows global comparison between factories, which results in capabilities to improve manufacturing solutions immediately across the globe. (Grieves, 2014)

Generally Accepted Definition of Digital Twin

Based on the concept by Grieves, there are several variating definitions defining the digital twin. A general definition of digital twin, which has been recognized and used by most people, was given by Glaessegen and Stargel in 2012 (Tao, et al., 2017): digital twin is an integrated multi-physics, multi-scale, probabilistic simulation of a physical object that uses the best available physical models, sensor updates, etc., to mirror the life of its corresponding twin.

The definition emphasizes the integration of different simulation types. Multi-physics simulation is definition for coupling different physical phenomena, such as mechanical, electrical, fluid, chemical into one simulation. Multi-scale simulation combines simulations from different abstraction levels, which enables the interaction between these levels. Probabilistic simulation denotes in this context more advanced probabilistic simulation methods than what is currently widely available. Current probabilistic methods are based on assumed similitude of conditions, which makes them inadequate. Statistical assessments must also be individually tailored to fit the needs of each device. (Glaessegen and Stargel, 2012)

Cyber-Physical Systems

Cyber-physical systems (CPS) are frequently presented in scientific literature with digital twins. This is because the digital twin is one of the key enablers in the concept of cyber-physical systems, which focus more on the opportunities on the factory floor and the system level, than on the detailed model level. Cyber-physical systems could be described as a group of physical devices, equipment and other objects, which interact with a virtual cyberspace through a communication network. (Schroeder G, et al., 2016) (Lee, 2008) (Baheti and Gill, 2011)

2.2.2 Digital Twin Modeling Approaches

This section presents several modeling approaches for digital twins. Some of the approaches are more conceptual, while the others are more focused on specific subjects. These approaches are generally based on specific approaches published in scientific publications, and the most advanced and appropriate information for this thesis has been gathered and presented in this chapter.

Digital Twin Types, Lightweight Model and Unified Repository

First approach is mainly based on the work from Michael Grieves, who is research professor at Florida Institute of Technology. Grieves introduced the term digital twin in 2003 at one of his presentations about product lifecycle management. (Grieves, 2014)

Digital Twin Types

Digital Twin could be divided into two different types: digital twin prototype (DTP) and digital twin instance (DTI). There could also be digital twin environment (DTE), which provides operation environment for the digital twins. DTP would describe the physical object at a prototype level. It would contain all the information required to describe and produce the physical object. DTP would include information, such as product requirements, annotated 3D model, bill of materials, bill of processes, bill of services and bill of disposal. DTI would individually mirror the physical product throughout the life of the product. In addition to DTP, DTI could contain data such as sensor data, service record, operational states, dimension data and other data depending on the use cases. (Grieves and Vickers, 2016)

DTE is an integrated, multi-domain physics application space for operating on digital twins. It could be used for variety of purposes such as predictive and interrogative use of digital twins. The goal of the predictive mode is to predict future behavior and performance of the product. The prediction with DTP would focus on the behavior of the designed products with components that, for example, vary between their tolerance values. The prediction with DTI would focus on providing a range of possible future states of the product, based on the information from the actual products and their history records. Interrogative mode of the DTE applies to DTI, and it can be used to interrogate the past history of the instances, such as fuel amount, geographic location, structural stress and other product specific instances. (Grieves and Vickers, 2016)

Lightweight Model

Lightweight model can be created from the virtual model of the digital twin. Purpose of lightweight model is to only select required characteristics, and attributes without anything unnecessary. This enables visualization and simulation of complex systems and systems of systems within real-time requirements and acceptable computing costs. (Grieves, 2014)

Unified Repository

Unified Repository (UR) is a two-way connection between physical and virtual products. The UR would be populated with virtual development tools and physical collection tools. The virtual tool would have knowledge of the identified characteristics, such as dimensions, tolerances and torque requirements. These characteristics would have a unique tag in the virtual model that provides a placeholder for the data from the physical product. After the design is released for production, these tags would be collected from the virtual model and used to create the UR. These tags would be included in the manufacturing execution system (MES) that is a control system for managing and monitoring processes on a factory floor.

MES would output the captured characteristics to the UR according to completed processes. The UR data is incorporated to factory simulation, where it is used to synchronize the simulation with the actual factory floor. (Grieves, 2014)

Digital Twin Driven Product Design, Manufacturing and Service

Second approach is based on the work from Fei Tao et al. (2017). Tao is professor at Beihang University and he has specialized in manufacturing systems, intelligent manufacturing and digital twins. In this approach, the digital twin plays a central role in the whole lifecycle of a product. This approach focuses on three fields: digital twin driven product design, manufacturing and service.

First, the concept of product lifecycle is briefly introduced. Product lifecycle starts from engineering design and continues through manufacturing and service to disposal of the product. Product lifecycle has been generally managed with product lifecycle management (PLM) software, which integrates people, data, processes and business systems (see Figure 2.4).

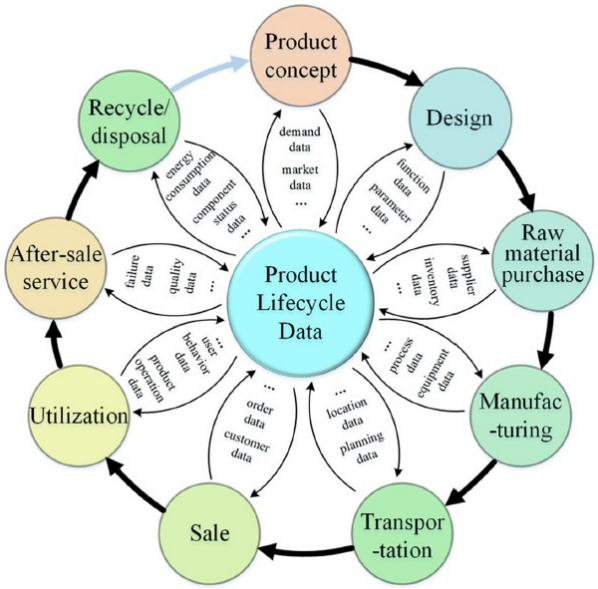

Figure 2.4. An example of product lifecycle data. (Tao, et al., 2017)

Product Design

Digital twin-driven product design can be divided to three phases. First phase is the conceptual design, second phase is detailed design and the third phase is virtual verification. Illustrative example of digital twin in design phases can be seen in Figure 2.5.

Conceptual design is the first and also the most important phase of the whole product design process. Designer defines the concept, esthetics and the main functions of the product while dealing with various types of information, such as customer satisfaction, product sales, investment plans and other. The amount of required data is large, and it is in scattered form. With a digital twin, scattered data can be integrated, and the twin also enables more transparent communication between clients and designers. (Tao, et al., 2017)

In detailed design, the design and construction of a prototype should be completed. In addition, tools and equipment utilized in production should be developed. The detailed design stage also includes simulation tests to verify the fulfillment of performance requirements. Due to lack of real-time and environmental-impacted data, simulation results may not be accurate. Digital twin plays a major role to solve this problem. It can coevolve with the physical object from the start of the lifecycle, as well as it collects product and environment related data. (Tao, et al., 2017)

In a traditional verification process, the validity and feasibility of the design cannot be evaluated until a small batch of the product has been produced. This delays the start of the actual production, which leads to increased financial costs. Digital twin could be used to effectively predict the quality of the product before starting the production. Digital twin-driven virtual verification allows full utilization of the data from equipment, environment, material, customers, physical characteristics, and history data. With this method, possible design defects and root causes can be effectively searched. In addition to this, digital twin can be used to propose solutions and optimization to the real systems. (Tao, et al., 2017)

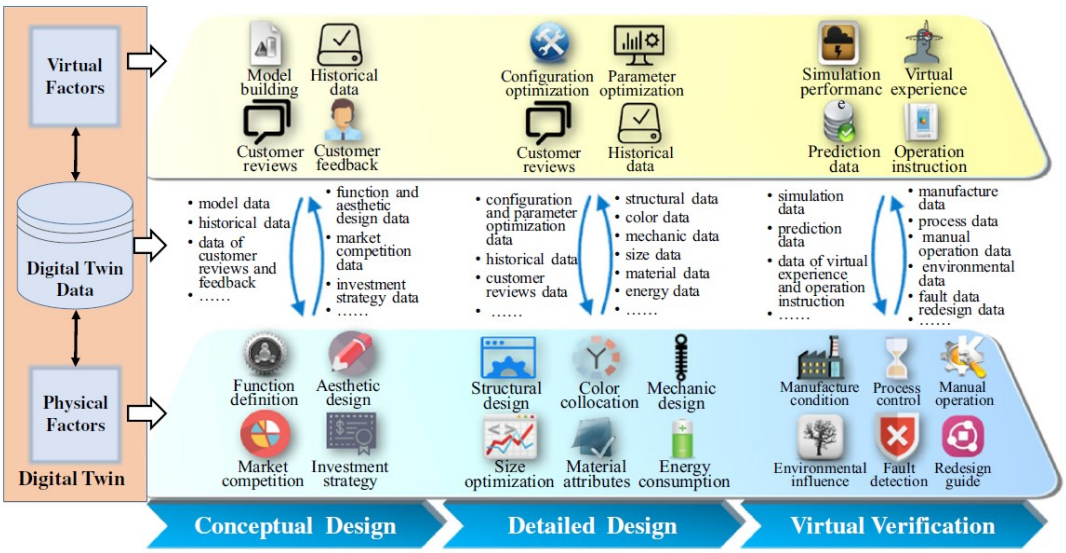

Figure 2.5. A digital twin in every phase of design processes. (Tao, et al., 2017)

Product Manufacturing

Product manufacturing is the entire manufacturing process from the raw materials to finished goods. This includes mainly three aspects: resource management, production plan and process control. First, resources such as materials, equipment, tools, operators, and others should be prepared and allocated. Second, production plan should be devised to predefine the manufacturing process. This includes machining, assembly, logistics, and other to achieve objectives like cost reduction, shorter manufacturing time and improved quality. Last, to ensure the accuracy, stability, and high efficiency of the process, following things are required to be monitored and controlled in the execution stage: real-time states, such as the production schedule, material storage and product quality. (Tao, et al., 2017)

Tao et al. proposed a new paradigm for product manufacturing: Digital Twin Shop Floor (DTS) which is composed of the following components:

- Physical Shop Floor (PS)
  PS is objective entities set that is responsible for receiving production tasks and predefined orders, and executing the orders to manufacture products.

- Virtual Shop Floor (VS)
  VS is an accurate virtual model, which can simulate and forecast production plans and processes. It also provides optimization strategies to Shop Floor Service System and monitors and regulates the manufacturing process in real-time.

- Shop Floor Service System (SSS)
  SSS is the set of service systems, providing support and services for the product manufacturing.

- Shop Floor Digital Twin Data (SDTD)
  SDTD refers to all data related to PS, VS, and SSS.

Figure 2.6 shows how PS, VS and SSS interact with each other through SDTD to accomplish the iterative optimization for resource management, production plan, and process control.

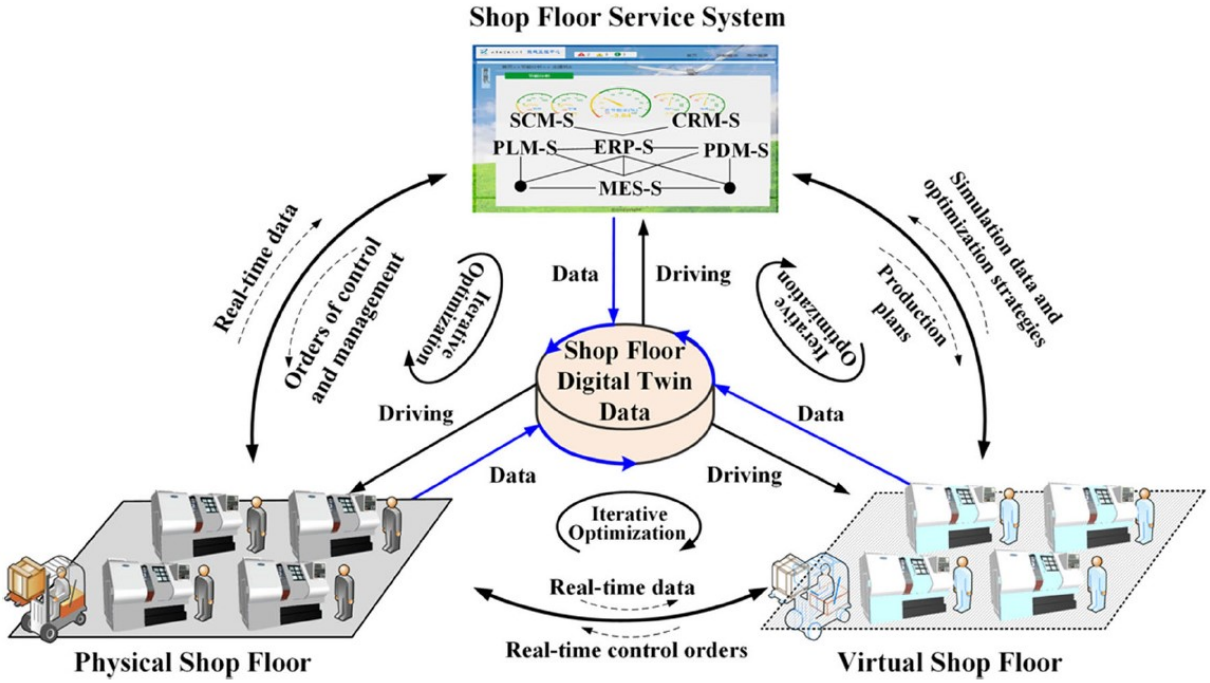

Figure 2.6. The concept of digital twin shop floor. (Tao, et al., 2017)

Product Service

Defects in complex products, such as automobiles and aircrafts can lead to malfunctions or even serious safety accidents. For this purpose, it is important to carry out recommended maintenance for these products. Current maintenance methodologies for complex products are inadequately based on similitude and a heuristic understanding instead of the specific materials, structural configuration, and usage of an individual product. With a digital twin methodology, degradation and anomalous events can be predicted in advance. This provides

possibility to individually customize the most relevant services for complex products. (Tao, et al., 2017)

Experimentable Digital Twins

This approach is based on the work from Michael Schluse et al. at University of Aachen. This approach presents the concept of experimental digital twins, which introduces eRobotics as the platform and Virtual Testbeds as the testing framework for digital twins. Also, model-based systems engineering is introduced as a tool to model more and more complex systems.

Background

The current use of simulation tools could be described as tool-centric approach where most simulation tools are able to solve exactly one specific application. This results in a discontinuous, time consuming, expensive and error-prone use of simulation systems throughout the development. To overcome these limitations, it was necessary to develop new concepts that can flexibly combine and exchange different simulation aspects. (Schluse, et al., 2016)

First part of the solution was not to have separate individual simulations but to have digital twin in the middle of the development process. This results in situation, where the focus concentrates on the digital twin objects, which become increasingly more elaborated over time. As an example: the lifecycle of a digital twin may start with a simple product related data. After that, a discrete event simulation could be added for analysis on a system level. At the end of design phase, a combined rigid body and FEM simulation could be utilized for analyzation purposes. Different digital twins could also be combined to set up multi-scale simulations with different level of details. These digital twins could represent systems, systems in their environments or system of systems. The same digital twin should be usable in various scenarios or even completely different applications as well. (Schluse, et al., 2016)

Second part of the solution addresses the technological challenges. New comprehensive interdisciplinary approach is required to combine the existing realizations of the various aspects of simulation technology, integrating various systems and systems of systems to be simulated, and providing development methods supporting different development processes. (Schluse, et al., 2016)

eRobotics and Virtual Testbeds

The proposed solution is based on the use of eRobotics and Virtual Testbeds. The aim of eRobotics methodology is to provide a platform and a comprehensive software environment for the development of complex technical systems. For example, it can be used in user requirements analysis, system design, support for the development and selection of hardware, programming, system and process simulation and control design. This allows the use of simulations right from the beginning of the development process to enable system testing in the concept phase. (Schluse, et al., 2016)

Virtual Testbeds in eRobotics are used to design, program, control and optimize complex systems and their interaction with prospective environment in simulation. Testbed allows engineers to simultaneously examine the entire digital twin in its environment. This allows efficient development, test, and verification on a component and system level at any time. Virtual Testbed combines data processing system with simulated environment. Example of the structure of Virtual Testbed can be seen in Figure 2.7. Combining Virtual Testbeds and

digital twins on eRobotics platform, leads to new kind of experimentable digital twins (EDT). (Schluse, et al., 2016)

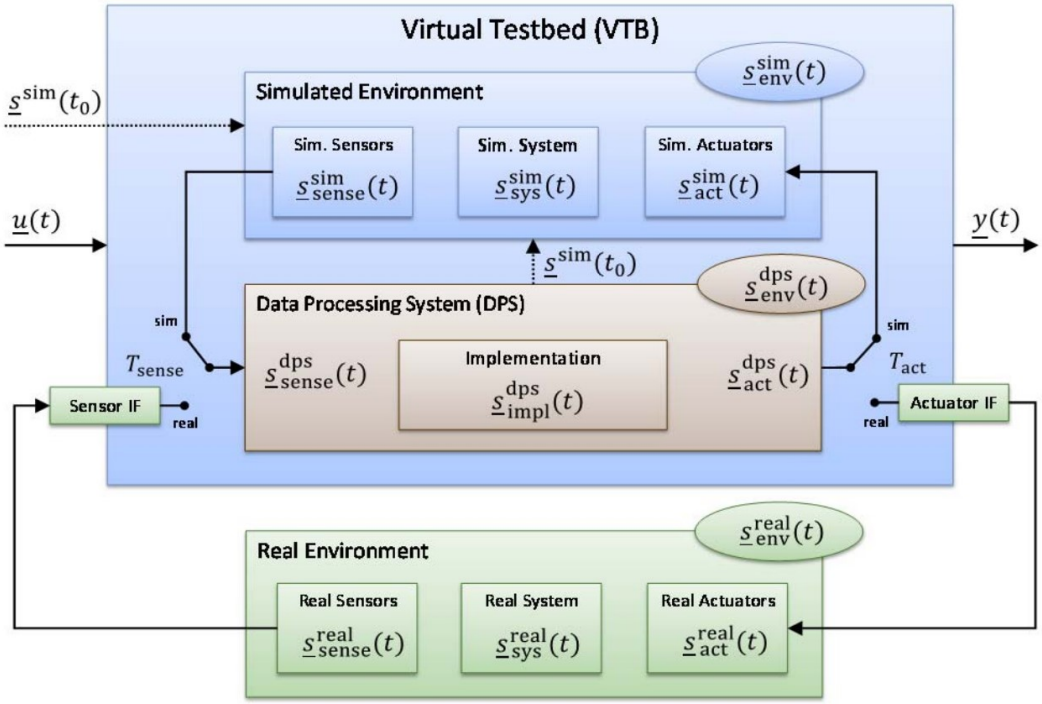

Figure 2.7. The structure of virtual testbed. (Schluse, et al., 2016)

Versatile Simulation Database

To be able to model and to concurrently simulate digital twins with the presented eRobotics concept, new simulator requirements emerges. Simulator can be used to build a model reproducing a system's dynamic behavior and processes and to make the model executable. To overcome the current limitations in simulation technology, a new micro kernel architecture for simulation systems was developed. This micro kernel is called as Versatile Simulation Database (VSD). VSD is an object-oriented real-time database that provides central building blocks for data management, meta information, communication, persistence, and user interaction. VSD is not only a static data container, but it also contains the algorithms and interfaces to manipulate data. One of the most important features of this architecture is that it is capable to integrate various data sources, simulation systems, visualization, interaction and feedback devices as well as the real-world counterparts of the digital twins. (Schluse, et al., 2016)

Model-based Systems Engineering with Experimentable Digital Twin

Model-based Systems Engineering (MBSE) focuses on domain models as the primary means of information exchange. This allows visualization of the technical system in more abstract level without losing necessary information. MBSE is presented in more detail at section 2.4.1.

MBSE and simulation technology play a key role in coping with the more increasing complex technical systems. The first steps of using the MBSE, such as iterative modeling of requirements, designs, behaviors and tests, have become standard procedures, but the transition of MBSE models to simulation is often restricted to simple scenarios. One of the main reasons for this is that it would require state-of-the-art simulation technology and simulation framework to enable multidisciplinary simulation. This problem could be solved with Versatile Simulation Database. Figure 2.8 presents the Experimentable Digital Twin as the node between systems engineering and simulation technology. (Schluse, et al., 2017).

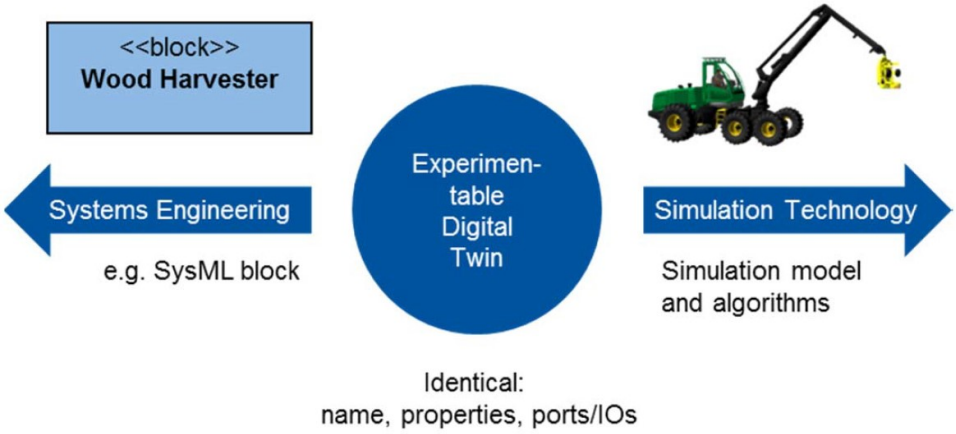

Figure 2.8. The relation between model-based system engineering and experimentable digital twin. (Schluse, et al., 2017)

Digital Twin Data Modeling with AutomationML

Schroeder et al. (2016) presents a concept for digital twin data modeling with AutomationML and a communication methodology for data exchange. AutomationML is briefly introduced before presenting the concept.

AutomationML

Automation Markup Language (AutomationML) is a neutral and open standard data exchange format, which is based on Extensible Markup Language (XML). Goal of AutomationML is to interconnect engineering tools from different engineering disciplines, such as mechanical, electrical, systems and control engineering. AutomationML is standardized in IEC 62714. (Schmidt and Lüder, 2015)

AutomationML utilizes four already existing standards. CAEX (Computer Aided Engineering Exchange, IEC 62424) implements the topology of the model including properties and relations of objects in hierarchical structure. COLLADA (COLLAborative Design Activity) implements geometry and kinematics. PLCopen XML implements logics. Structure of AutomationML can be seen in Figure 2.9. (Schmidt and Lüder, 2015)

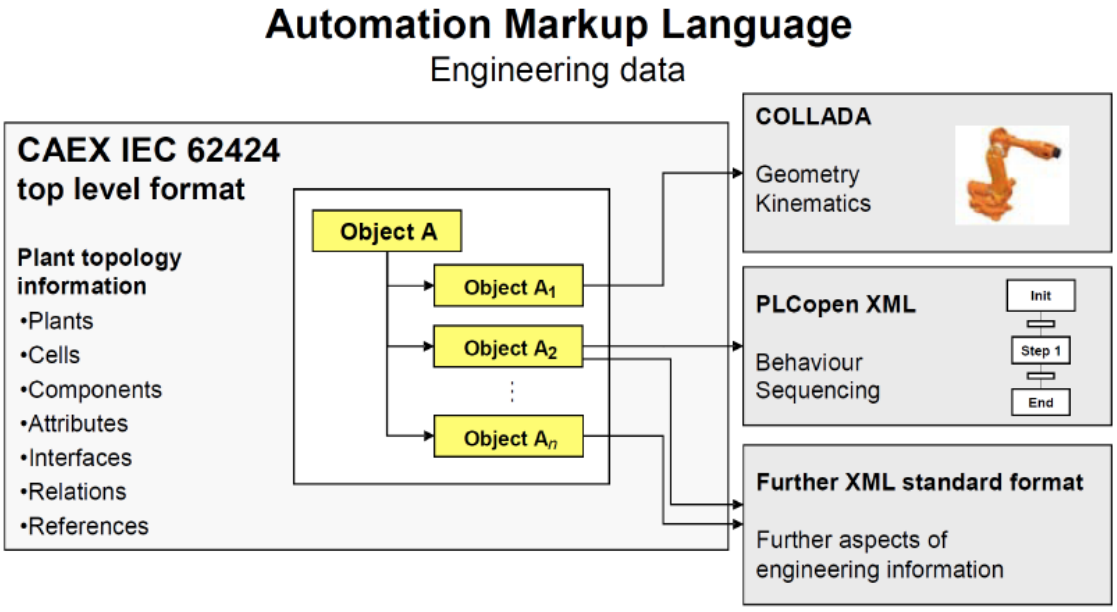

Figure 2.9. The structure of AutomationML elements (AutomationML, 2014).

AutomationML supports integration framework philosophy, in which centralized data broker is utilized to transfer data between engineering tools (see Figure 2.10). AutomationML as standardized data exchange format can be utilized as central data broker between the engineering tools. (Schmidt and Lüder, 2015)

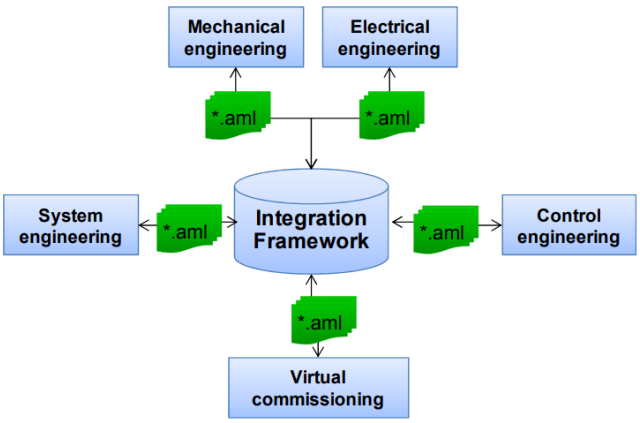

Figure 2.10. *Integration framework as a central data broker for AutomationML format. (Schmidt and Lüder, 2015)*

AutomationML with Digital Twin

Digital twin contains different types of models such as systems models, 3D models, multi-physics models, manufacturing models and other models, which are created during different phases of product lifecycle. AutomationML is a proposed solution to be used to create these models from physical devices. The aim of the research was to contribute a methodology that allows data to be available for data exchange using AutomationML models. *(Schroeder, et al., 2016)*

The methodology is based on the steps seen in Figure 2.11. First stage is creating a model of a physical device using a modeling tool such as AutomationML. Second stage is to enable the exchange of information of the modeled attributes in the model. These models should also be available in an open format such as text file model. Providing information to other systems can be implemented, for example, with the use of middlewares. The last stage is composed by the consumers of information systems such as monitoring applications and augmented reality applications. *(Schroeder, et al., 2016)*

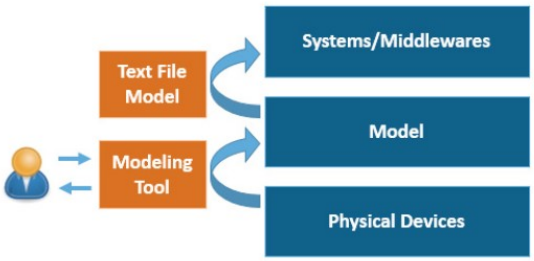

Figure 2.11. The methodology steps for data exchange with AutomationML. (Schroeder, et al., 2016)

2.3 Digital Twin in Simulation

Digital twin provides comprehensive real-time information of the current state of the system. This extends the possibilities for the use of simulation. Instead of using simulation only in the design and validation phase, simulation could be used during the system operation in real-time. Real-time simulation operations introduce new requirements for simulation software to provide automatic operation handling, because operations executed by human operators do not meet the real-time requirements.

2.3.1 Real-Time Operations

Digital twin is a key enabler of the real-time or near real-time operations, such as system optimization, production planning and validation, and other operations, in which the knowledge of the current state of the system is required.

Based on the concept of digital twin, the twin should always reflect the current state of the physical system in real-time or near real-time. However, there could simultaneously exists additional digital twins for different purposes. The creation of these additional twins would be based on the original digital twin.

Uhlemann et al. (2017) presents a concept, where the original digital twin generates a database from the data it contains. This database is used to generate additional digital twins, which reflect the current states of the system. The purpose of the original digital twin is only to reflect the physical system and update the database, while the additional digital twins are available for other operations. One of these additional digital twins should act as a reference of the current system during a comparison, while the other additional twins, would be modified. Modification operation could be, for example, calculating set of optimized parameters for the digital twin. Twin with the optimized set of parameters would be compared to the

reference digital twin to acquire knowledge of the achieved outcome. When suitable optimization values have been found, those could be automatically transferred to production control system.

Virtual testbed environment presented by Schluse et al. in section Experimentable Digital Twins, could provide an effective tool to perform real-time operations as well. The testbed has integrated tools that can be used in operations such as optimization. However, optimization capabilities and methods of virtual testbeds was no further described by Schluse et al.

In both of these approaches, digital twin plays a major role as the enabler of these real-time operations. However, most of the current methods to perform optimization require the definition of parameters and the evaluation of the solutions by the human operators. Coupling between simulation and optimization is still under research (Uhlemann, et al. 2017).

2.3.2 Autonomous Systems

Requirements for product complexity and customizability are increasing, while the ramp-up time of the production should decrease. Autonomous systems play a key role to solve this problem. Autonomous systems are capable to execute high-level tasks independently without detailed human control. These tasks can be production planning after the configuration change in the production system or optimization of the production. (Rosen, et al. 2015)

In order to implement autonomous systems, the system will require realistic models describing the current state of the process and the interaction of their own actions. This can be achieved with digital twin. (Rosen, et al., 2015)

One of the key principles in the operation of autonomous systems is to decentralize decision making. In centralized decision making, there are clear hierarchy levels, where upper level controls the lower levels. When production configuration is changed in centralized decision making, reprogramming of the control system is generally required. (Rosen, et al., 2015)

Decentralized decision making is based on less hierarchical and more networked structure (see Figure 2.12). One generally accepted approach of decentralized system is multi-agent systems, which are composed of multiple individually functioning and interactive agents. There are different types of agents, for example, some can represent factory floor devices, and other can represent manufactured products. These agents have sets of skills that define the capabilities of the agents. The knowledge of capabilities and interactive communication of agents leads to highly modular systems. (Rosen, et al., 2015) (Wang, et al., 2016)

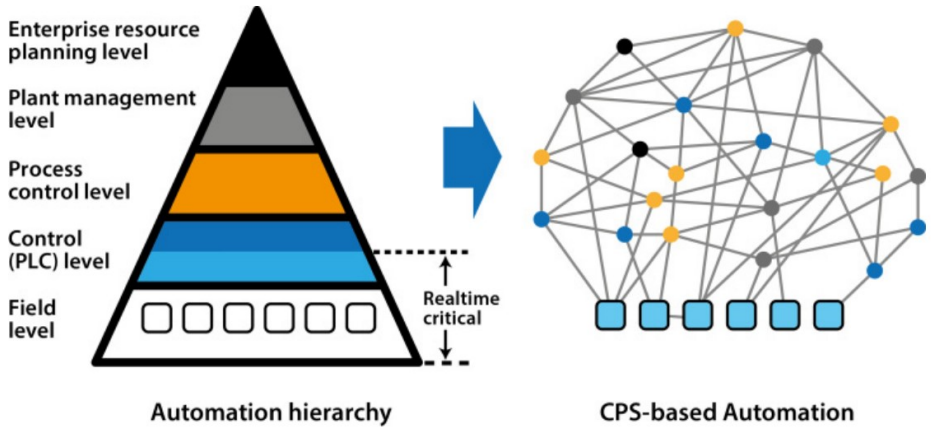

Figure 2.12. Illustrative examples of centralized system hierarchy and decentralized system network. (Monostori, 2014)

2.4 Formal Modeling Languages

When systems become more and more complex, modeling languages can be utilized to achieve higher level of abstraction to the system model. Formal modeling languages also provide widely applied and standardized modeling format, which enables improved possibilities of sharing the information of the model, as well as independency from certain individual syntaxes or software. Formal representations also have the major advantage that they can be machine readable (Gianni, et al., 2017 p. 150).

This chapter first introduces model-based systems engineering, and thereafter two modeling languages: Unified Modeling Language (UML) and Systems Modeling Language (SysML). The last section introduces intermediate modeling layer.

2.4.1 Model-Based Systems Engineering

The Model-based systems engineering (MBSE) approach was popularized by The International Council on Systems Engineering (INCOSE) in January 2007. MBSE is the formalized application of modeling to support system requirements, design, analysis, verification and validation activities from the beginning of the conceptual design phase throughout the development phase to later lifecycle phases. MBSE methodology focuses on creating and utilizing domain models as the primary means of information exchange between engineers to support analysis, specification, design and verification of the system being developed. Domain model is a conceptual model that incorporates behavior and data. (Hart, 2015) (Friedenthal et al. 2015, p. 15-21)

MBSE is an enabling technology for innovative, interdisciplinary product design. One of the main goals of the MBSE is to transfer from document-centric engineering to model-centric engineering (Gianni, et al., 2017 p. 158). In addition to being machine readable, the biggest advantages of MBSE is the automatic generation of various artifacts for design and analysis operations, such as automatic checks, dependency analysis, performance analysis and report generation. (Gianni, et al., 2017 p. 178). These operations can save significant amount of time from repetitive work as well as reduce human errors.

Modeling language used in MBSE is generally SysML that is widely used general purpose language based on the UML. SysML is intented to facilitate the applications of an MBSE approach. System models, created with SysML, can be used to specify the hardware and software components of the system. Software component models can be expressed with languages such as Unified Modeling Language (UML), while the hardware components can be expressed with Computer Aided Design and Computer Aided Engineering (CAD/CAE) models (see Figure 2.13). (Friedenthal et al. 2015, p. 15-21)

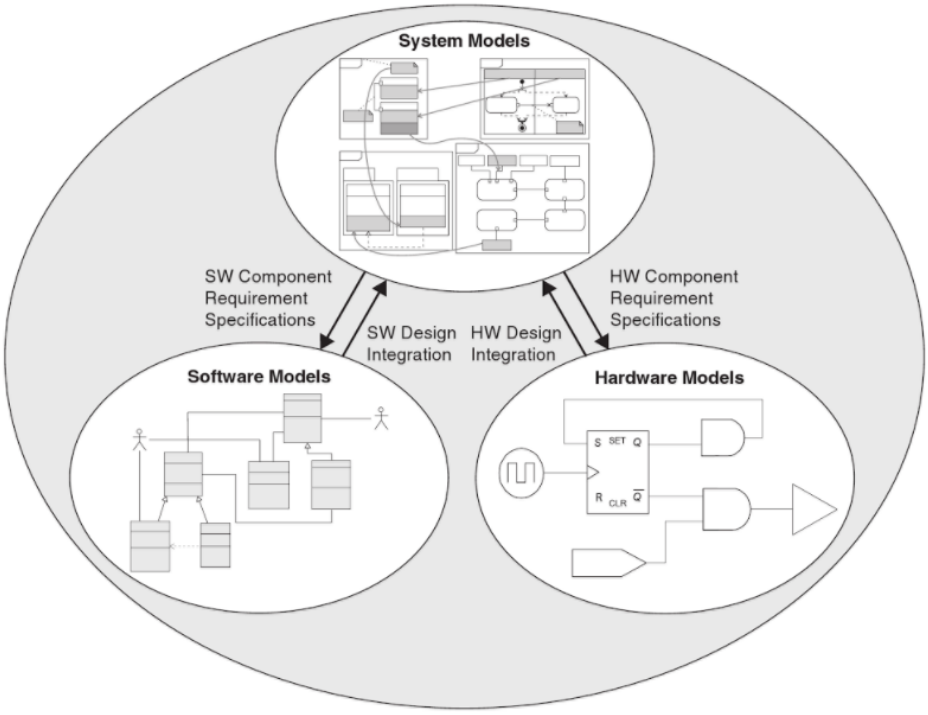

Figure 2.13. MBSE approach with system models connected to software and hardware models. (Friedenthal et al. 2015, p. 18)

2.4.2 Unified Modeling Language

The Unified Modeling Language (UML) is general purpose modeling language in software engineering field. Goal of the UML is to provide standardized graphical way based on diagrams to present the design of a system. UML was adopted as a standard in 1997 by the Object Management Group (OMG). In 2005 UML was published as an approved ISO standard by the International Organization for Standardization (ISO) that is periodically revised to cover the current revision of UML. (Weilkiens, 2007, p. 16-17)

Current UML versions since the version 2.0, include thirteen types of diagrams, which can be divided into three categories: structure diagrams, behavior diagrams and interaction diagrams. Structural diagrams are used to model the organization of the system or the structure of the data that is processed by the system. Behavior diagrams are used to model the dynamic behavior of the system and how it responds to events. Interaction diagrams are used to model the interactions between a system and its environment, or between the components of a system. (Weilkiens, 2007, p. 146-147)

2.4.3 Systems Modeling Language

The Systems Modeling Language (SysML) is a general-purpose modeling language in systems engineering field. It supports the specification, analysis, design, verification and validation of complex systems. One system may include several elements such as hardware, equipment, software, data, personnel and other. SysML can be used to specify and architect systems, as well as to specify components, that can be designed with other domain-specific languages, such as UML for software design and CAD modeling for mechanical designs. (Friedenthal et al. 2015, p. 16-19, 31-32) (OMG, 2018) (SysML, 2015)

The SysML was introduced in 2007. OMG and INCOSE developed the UML for system engineering request for proposal in 2003, which specified the requirements for extending UML to support the needs of system engineering community. In response to these requirements, the SysML was developed by diverse group of vendors, end users, academia, and government representatives. The SysML was then adopted by the OMG. (OMG, 2018) (SysML, 2015)

SysML offers improvements over UML for systems engineers. SysML provides more flexible and expressive semantics and the software-centric restrictions of UML are reduced. SysML is also smaller language than UML, which makes it easier to adopt. Illustrative comparison of UML and SysML can be seen in Figure 2.14. (Friedenthal et al. 2015, p. 33-36) (OMG, 2018)

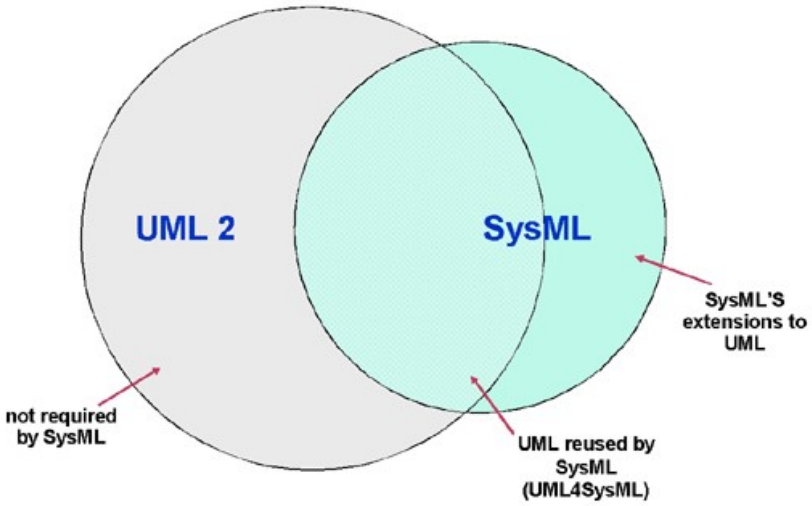

Figure 2.14. The relation between UML 2 and SysML. (OMG, 2018)

Example model of SysML can be seen in Figure 2.15. This example model consists of structures, behaviors, requirements and parametrics. The structure contains the blocks to determine the structure and hierarchy of the system. The behaviors contain an activity diagram that specifies the external and indirect interactions. The requirements contain a set of requirements that would generally be found in a system specification. The parametrics blocks contain the system properties that are bound to the parameters. (Friedenthal et al. 2015, p. 33-36)

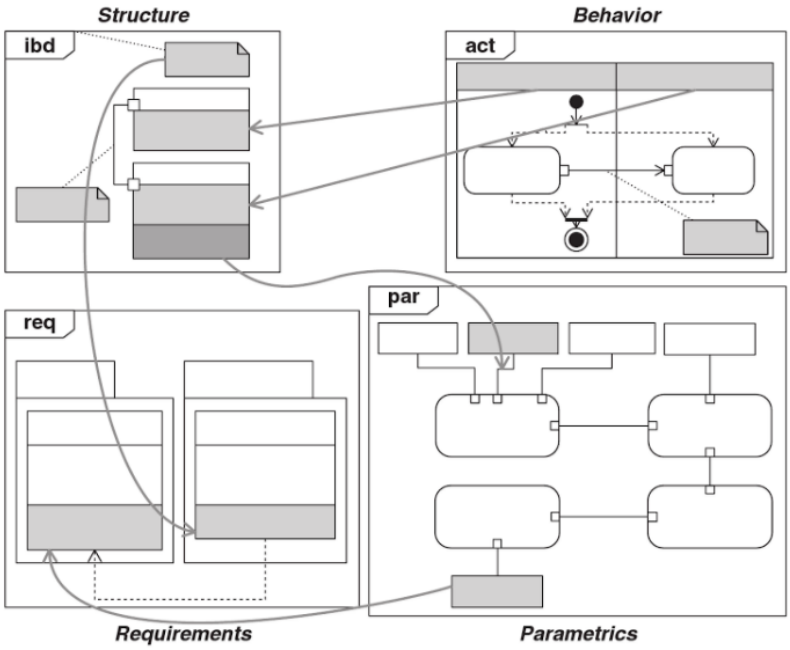

Figure 2.15. An example of a structure of a system model. (Friedenthal et al. 2015, p. 17)

2.4.4 Intermediate Modeling Layer

The engineering of production systems is collaborative work of many different engineering disciplines, which typically utilize specialized software tools that are used to define the behavior of manufacturing system. Engineering also includes different development stages from rough overall description of production system to more detailed system, and results in fully specified and ready to use system. (Mayerhofer, 2016)

The factory behavior is typically defined with representation data formats, such as Gantt charts, impulse diagrams, and sequential function charts. Example of Gantt chart can be seen in Figure 2.16. However, the compatibility of the software tools utilized in engineering processes is not generally satisfactory, which leads to manually performed data exchange. (Mayerhofer, 2016)

AutomationML introduces an intermediate format named as Intermediate Modeling Layer (IML), which functions as an adapter to transform plant behaviors to target format of PLCopen XML. PLCopen XML is based on the standard IEC 61131-3, which is noted one of the most successful global standards for industrial control software (Vyatkin, 2013) (PLCopen, p. 6-9).

PLCopen XML can define control logics with the following PLC programming languages defined by the IEC 61131-3: sequential function charts (SFC), function block diagrams (FBD), ladder diagrams (LD), structured texts (ST) and instruction lists (IL). Example of SFC language can be seen in Figure 2.16. (PLCopen, p. 6-9)

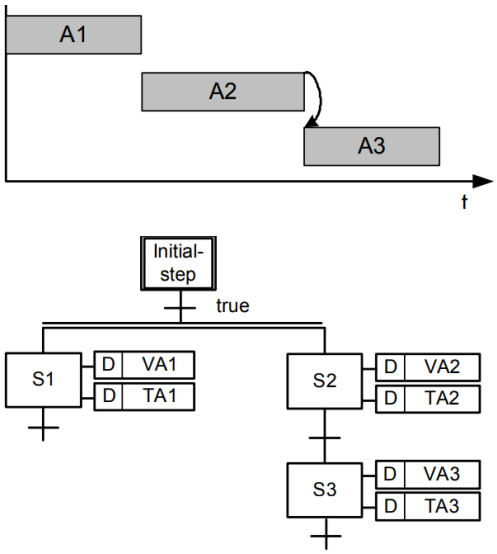

Figure 2.16. The upper chart represents a Gantt chart and the lower chart represents sequential function chart. Both charts represent identical behavior. (Drath, et al., 2008)

To transform plant behavior data formats to IML, transformation rules require to be defined. These transformation rules also enable the creation of data formats from IML, which enables possibility to transform from one data format into another. One major benefit of IML is that complex transformation rules between IML and PLCopen XML require to be defined only once for IML. Illustrative example from the IML operations can be seen in Figure 2.17. (Mayerhofer, 2016)

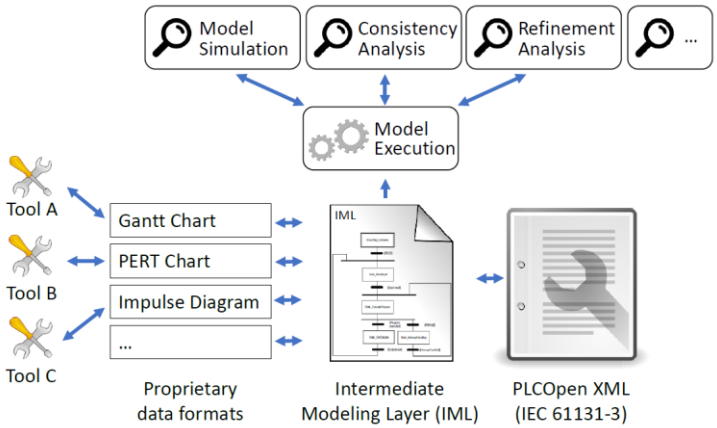

Figure 2.17. Intermediate Modeling Layer between data formats and PLCopen XML. (Mayerhofer, 2016)

The IML, and the mappings between the IML and the languages of PLCopen XML are only semi-formally described. To constitute first step towards formalization and validation of IML data exchange, Mayerhofer (2016) proposed a metamodel and operational semantics for the IML.

3 Selection of Industrial Devices

One of the main goals of this thesis is the development of component classification for simulation components. To develop component classes, potential industrial devices must to be selected, which constitute the basis of the classification. The industrial domains are limited to the automotive industry and the food & beverage industry with focus in the packaging area.

In the first section of this chapter, automotive industry is introduced and the selected devices from automotive industry are introduced. In the second section, food & beverage industry is introduced, and the selected devices from the industry are presented.

3.1 Use Case: Automotive Industry

3.1.1 Automotive Industry

Automotive industry includes manufacturing and assembly of complex products, high production rate and dangerous work environment. These factors drive automotive industry to utilize increasingly more automation. Weyer et al. (2016) described automotive as one of the most competitive, advanced and complex industrial sector.

One challenge in the automotive factory is to convey the chassis of the car through every process step. This results in several types of conveyor systems, from simple belt conveyors to sophisticated fully programmable and automated conveyors systems that are capable to communicate with each other. In addition, automotive factories require large amounts of factory floor and space, which increase the need for space saving solutions. Because of this, the automotive industry has exploited the factory space vertically. Several conveyor solutions are built as overhead conveyors, that can significantly reduce the required space and material flow distances.

Automotive factory typically consists of the stamping shop, body shop, paint shop, assembly shop and powertrain shop as seen in Figure 3.1. In the stamping shop, chassis is manufactured from pressed sheet metal parts. These parts will be used as assembly parts for the main chassis, doors, hoods and other chassis related objects. Manufacturing starts from blank sheet metal coils which are straightened and cut. These sheet metal parts are transported to presses where they obtain their final form.

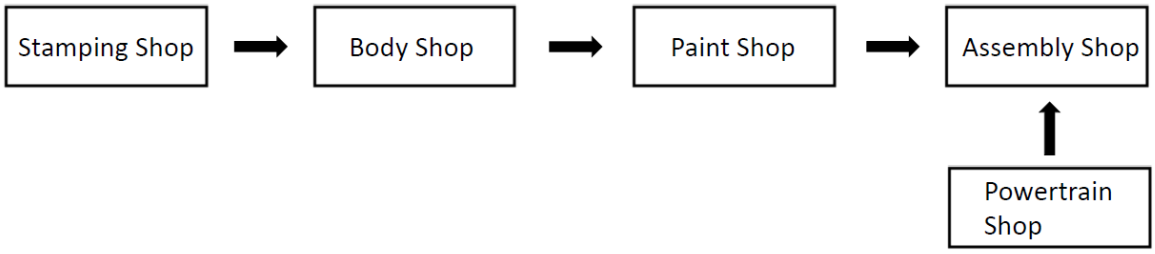

Figure 3.1. Typical domains in the automotive factory.

In body shop, these pressed parts are assembled to form the chassis of a car (see Figure 3.2). This assembly is typically performed by robots and the parts are attached with spot welding techniques. Some of the assembly work can be performed in separate robot cells, but the main chassis is moving on a conveyor, which is surrounded by robots performing assembly work. (BMW Body Shop, 2018)

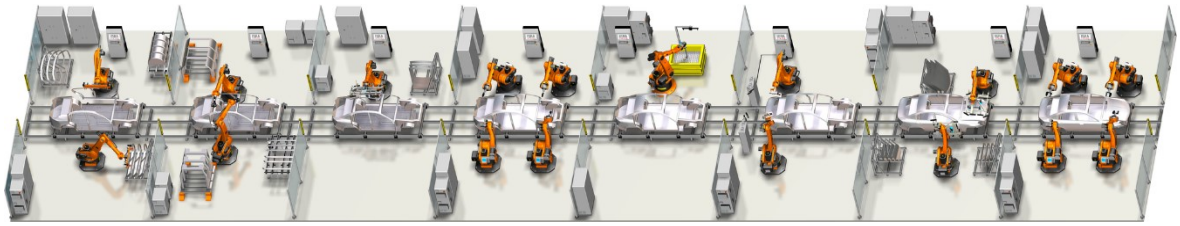

Figure 3.2. *An illustrative and simplified example of a production line in the body shop. (ISRA vision, 2018)*

After body shop the chassis is transferred to the paint shop. At first, chemical treatments are given to the chassis by typically dipping the chassis to a pool of chemicals. These treatments provide protective layer on the surface to protect against environmental conditions. Next step is to seal the chassis before painting. The chassis is coated with multiple layers of paint. Between the painting and dipping processes, the chassis may be warmed and cooled to accelerate painting processes. After the chassis is painted, the paint quality is inspected. (Dürr Paint, 2018)

After the chassis is painted, it goes to the assembly shop, where the cars are assembled. One process step is to remove painted doors from the chassis to ease the assembly processes. The exactly same doors will be attached to the chassis later during the final assembly to ensure the color compatibility. Early process steps in assembly shop is the marriage process between powertrain and the chassis. In this process, powertrain is inserted and attached to the chassis. During the assembly process interior of the car is assembled to the chassis. In the final assembly phase, seats and doors will be inserted to the car. (BMW Assembly, 2018)

3.1.2 Selected Devices from Automotive Industry

Progressive Press Tooling

Chassis of the car is assembled from pressed sheet metal parts. The final form of these pressed parts is typically complex and it cannot be achieved with single press cycle. Having one press for each process step is not always feasible. Presses capable of operating with progressive tooling are used in automotive industry (see Figure 3.3). These presses contain multiple stage forming tools that are used simultaneously during one process cycle. After the process, parts are transported to the next process location and the process is executed. Progressive press takes sheet metal plates as an input without first cutting it to product units. After every press cycle, the sheet metal is moved one step to the next process position. At the end of the process steps, the products are cut from the sheet metal strip. (Hyundai, 2017)

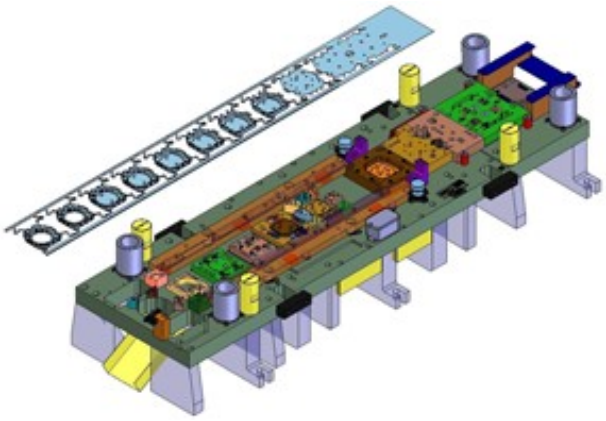

Figure 3.3. An assembly picture of progressive press tools and product preforms. (STM, 2018)

Tandem Press

For larger products, progressive press tooling approach is not as feasible as with smaller products. Tandem presses are used for pressing larger products. These presses are located near each other and the operation is automated with robots (see Figure 3.4). Robots can be integrated to the presses or they can be mounted separately. (Hyundai, 2017)

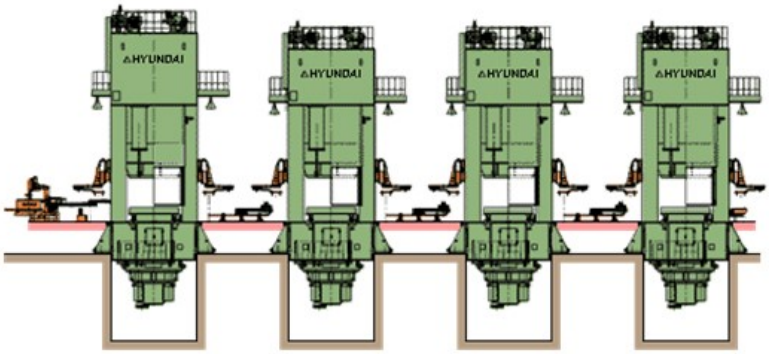

Figure 3.4. A side picture of a tandem press line with robots. (GR IAS, 2016)

Robots

Robots are effective at executing repetitive tasks, such as material flow handling, assembly, welding and painting, which are ergonomically difficult and dangerous for humans. Robots are capable to maintain uniform execution of tasks, which is a significant advantage in an industry, where the pace of production should be predictable. In automotive industry, robots are typically located in robot cells or next to the conveyor line.

End Effectors - Robots

End effectors are mounted to the end of the kinematic chain of a robot. End effectors are used to perform actions that interact with products. In automotive industry, typical actions are welding, painting and clamping. Typical end effectors for welding purposes are spot welding and arc welding guns.

Workpiece Positioners
Workpiece positioner positions the workpiece for robots (see Figure 3.5). The joints of the workpiece positioner can be controlled by a robot controller to improve the accessibility of the workpiece for the robot. These positioners can be applied for several purposes, for example, welding of large and complex products.

Figure 3.5. A r*obot workpiece positioner. (KUKA, 2018)*

Robot Positioners
Robot positioners positions the robot to improves the reachability of the robot (see Figure 3.6). The joints of the robot positioner can be controlled by a robot controller.

Figure 3.6. A robot mounted onto a robot positioner. (Direct Industry Positioner, 2018)

Chain Conveyor
Chain conveyor is a conveyor, which consists of a track, trolleys and carriers. The trolleys are moving on the track and they are mechanically attached to the chain of the track. When the chain is powered, the trolleys will move according to the movement of the chain. Carriers are typically integrated to the trolleys. Carriers are holding the products when the conveyor is moving.

Power and Free Conveyor

Power and Free (P&F) conveyor is a chain driven conveyor that consist of individual trolleys moving on a track. These trolleys are not mechanically fixed to the chain but they can be mechanically connected and disconnected with pusher dog components on the powered chain (see Figure 3.7). This feature enables asynchronous behavior which provides several benefits, such as collision avoidance, individual stop of specific trolley and trolley buffering. (McGuire, 2009, p. 135-140)

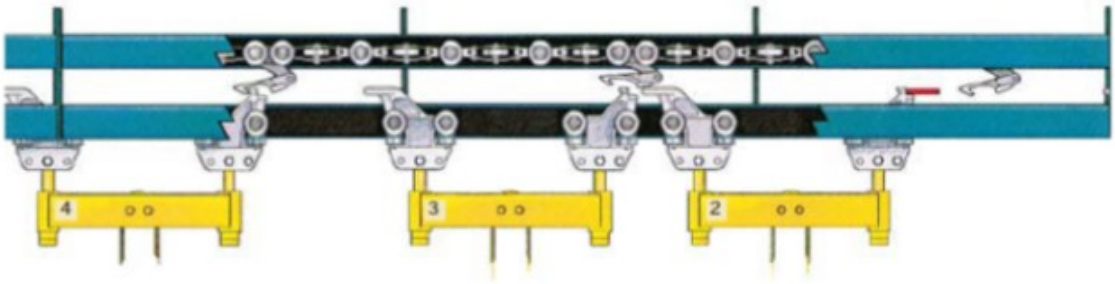

Figure 3.7. Trolleys on the conveyor track illustrates the mechanical working principle of P&F conveyors. (modified from P&F Mechanics)

The mechanical connection between the track and trolley is accomplished with an interaction of a pusher dog and the trolley. These pusher dogs are evenly located in the driving chain. When the pusher dog reaches the counterpart in the trolley, it starts pushing the conveyor. This connection can be disconnected when mechanically interacting with other trolleys or with the specific stop and go switch. After the stop, trolley can reconnect with the pusher dog when the next trolley moves or when the stop and go switch is turned on. (McGuire, 2009, p. 135-140)

The conveyor is not restricted to form a single loop with one powered chain configuration, but several chains can be used in one system. This allows trolleys to move from one chain to other chains while the trolley is still on the same track. In addition to several chain systems, there are merge, divert and lift components for the trolleys. This enables complex conveyor systems to be built for different purposes (see Figure 3.8). A typical carrier for the P&F conveyor in the automotive industry can be seen in Figure 3.9. (McGuire, 2009, p. 135-140)

Figure 3.8. Track sections of a complex power and free system. (IMH, 2017)

Figure 3.9. Carriers can hold the chassis of a car in overhead conveyor systems. (DS, 2018)

Mechanical operation principle in P&F conveyor provides several benefits. This conveyor can be used in heat treatment without the risk of damaging electronic components. P&F conveyors are also rather simple, durable and robust, which is one major factor explaining their wide usage in automotive industry. These conveyors are used, for example, in paint and assembly shops. (McGuire, 2009, p. 135-140) (DMW P&F, 2018)

Electrified Monorail System

Electrified Monorail System contains individually driven vehicles that move independently on the rail system. These vehicles include sensor systems that are used to control the vehicle and to avoid crashes. The control of the vehicles is fully programmable, which enables sophisticated material handling solutions. (ASI, 2018)

Skillet Conveyor

Skillet conveyor consists of skillets moving in a straight and one directional line. One skillet consists of a ground level pad and a vertically moving holder, that can hold the chassis (see Figure 3.10). The pad allows workers to step on the conveyor and move with it, which eases assembly operations. Skillet conveyors have been typically utilized in the assembly shop before the cars are capable to stand on their tires. (DMW, 2018)

Figure 3.10. A skillet conveyor holding a chassis. (ASAS, 2018)

Skid Conveyor

A skid conveyor consists of skids and roller beds. Skids are moving on the roller beds and they are capable to carry products. The skid is not mechanically fixed to any part in the conveyor system, which enables modular systems to be built (see Figure 3.11). In addition to roller beds, skid conveyor system can include turntables, cross transfers, lifts and other solutions that enable complex conveyor systems. Skid conveyors can be used in all manufacturing areas from body shop to final assembly. (Dürr, 2018)

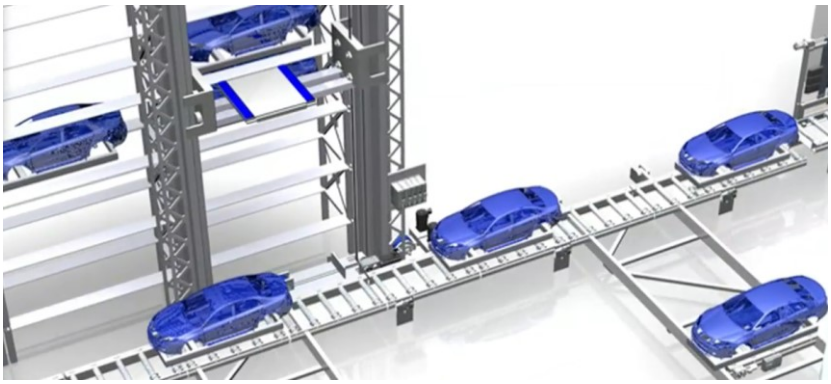

Figure 3.11. A skid conveyor system with a cross transfer and chassis storage system. (Lenze, 2018)

Inspection and Conditioning

Inspection process is typically applied to verify the painting quality after paint shop. Painted chassis is conveyed through an inspection room, where the reflection of light is used as inspection criterion. Inspection can be performed by a robot as well with an inspection tool as an end effector (Micro-Epsilon, 2018). Conditioning is typically heating or cooling of the chassis to accelerate paint shop processes. Heat treatment is performed to accelerate the curing of the paint, and cooling is performed to accelerate the cooling of the chassis after the heat treatment. (Dürr Paint, 2018)

Lift Assist

Despite the large utilization of automation, human operators perform some assembly operations. These operations may include lifting of heavy objects, which requires the utilization of lift tools. Lift tools can be used to pick, hold and place heavy objects. It can also include tools, such as pneumatic screwdrivers (see Figure 3.12). (Knight, 2015)

Figure 3.12. A human operated lift assist. (Knight, 2015)

3.2 Use Case: Food & Beverage Industry

3.2.1 Food & Beverage Industry

Food & beverage industry focuses on handling and processing massive amounts of products, which need to be effectively transported between processes. Packaging plays a major role in the food & beverage industry, providing a physical protection for the products, enabling efficient and reliable material handling as well as transmitting product and batch information. In addition to these factors, packaging can have additional targets such as portion size control and the extension of the product life time.

There are several noticeable phases in the production of food & beverage products. Firstly, products are generally transported on conveyor systems, in which the material flow can be influenced with devices such as robots, line routers and line pushers. The material flow is typically directed through several in-line processes. After the products are produced, they are packaged inside cases, which are palletized on top of pallets. Thereafter, pallets are wrapped with plastic film.

3.2.2 Selected Devices from Food & Beverage Industry

Line Mergers and Diverters

Line merger takes products from multiple line inputs and merges the products to one output. One type of merging solution can be seen in Figure 3.13. This device merges products by mechanically forcing them to the output. Line diverters function in the opposite direction compared to merging devices. Diverters require additional diverter part that mechanically directs products to targeted line output.

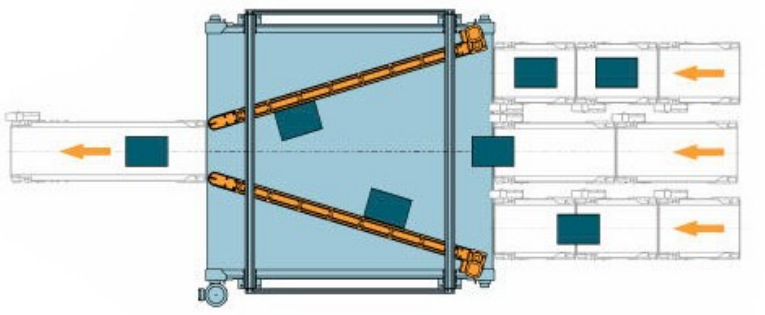

Figure 3.13. A mechanical line merger with three inputs and one output. (Direct Industry Merge)

Indexing Conveyor

Indexing conveyor consist of holder units that can contain products (see Figure 3.14). Indexing conveyors can be driven with constant speed or they can be driven with step motion. Triggering signal for step motion can be produced with external control, which utilizes sensors to inform product locations. Step motion can also be achieved with mechanical solutions. These mechanical solutions drive conveyor belt and peripheral devices synchronously according to motor speed.

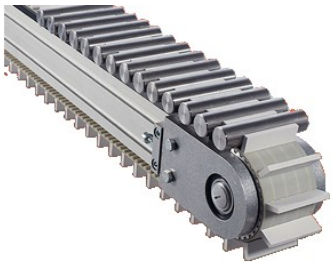

Figure 3.14. An indexing conveyor. (MK, 2018)

Package Wrapping, Closing and Sealing Machines

One typical method to package food & beverage products is to wrap the product with plastic film, close and seal the package. There are numerous different machine types and configurations to perform these operations. Form-fill-seal machines (FFS) are able to perform all of these actions from wrapping to sealing of the product. Two typical configurations of FFS machines can be seen in Figure 3.15 and Figure 3.16. Figure 3.15 presents vertical FFS machine that also measures the portions to be filled into the packages. Figure 3.16 presents horizontal FFS machine that packages products that are conveyed to the machine. (ULMA, 2018)

Figure 3.15. The principle of vertical form-fill-seal process. (ULMA Vertical, 2018)

Figure 3.16. The principle of horizontal form-fill-seal machine. (ULMA Horizontal, 2018)

Cartoning Machines

Cartoning machine erects, closes and seals cartons as well as the machine inserts products and product accessories inside the carton. Example of cartoning machine process steps can be seen in Figure 3.17.

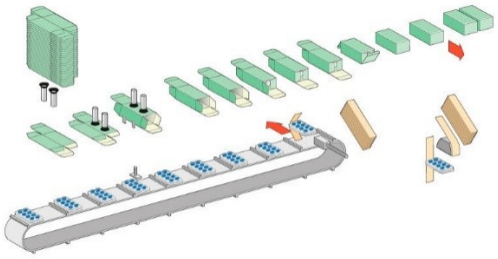

Figure 3.17. The principle of a cartoning process. (Direct Industry Cartoning, 2018)

Case Handling Machines

Case packaging denotes operations such as case erecting, closing, sealing and labeling. Example of a case erecting machine can be seen in Figure 3.18. Stack of case preforms arrives to this machine, and these cases are mechanically erected one by one. Thereafter, cases continue movement to line output, which can be connected to a production line. Other case handling machines are typically integrated to a production line.

Figure 3.18. A case erecting machine. (Radpak, 2014)

Palletizing and Wrapping Machines

Palletizing is a process in which cases are placed layer by layer in a specific formation on a pallet. Palletizing is typically performed at the end of production line. The formation of cases improves the stability of the loaded pallet but it is not an adequate measure. Wrapping of the loaded pallet is used to ensure, that the cases will hold the formation throughout logistical processes.

Palletizing operations can be divided to mechanical palletizing and robotic palletizing. In mechanical palletizing, cases are buffered at the end of line. One configuration of a mechanical palletizer can be seen in Figure 3.19. This palletizer waits when right amount of cases is buffered at the end of the line and then, the row of cases will be simultaneously pushed to a pad. This operation is repeated until the pad is full, which represents one pallet layer. This layer is conveyed to a lift that lifts the layer on top of a pallet. Robot palletizing utilize robot in the palletizing operations. Cases are buffered in a line and robot picks and places them on a pallet (see Figure 3.20). Robot enables more configurable palletizing patterns as well as

capabilities to handle varying size of cases. Wrapping machines wrap plastic film around the loaded pallet which holds the cases in formation and prevents them from falling. Wrapping machines are typically integrated with a conveyor line.

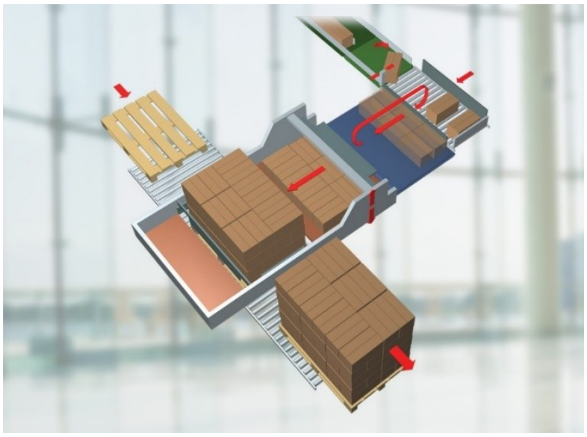

Figure 3.20. Mechanical palletizing process. (Gebo Cermex, 2017)

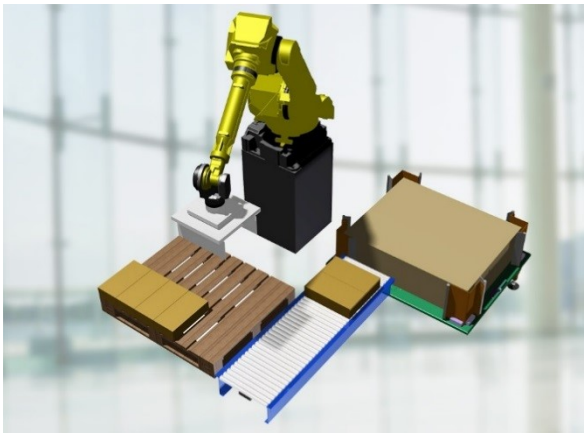

Figure 3.19. Robot palletizing process. (Gebo Cermex, 2017)

4 Component Modeling

This chapter introduces basic building blocks and principles of component modeling in VC software in section 4.1. In addition, development opportunities in the field of component modeling are presented in section 4.2.

4.1 Component Modeling in Visual Components

This section firstly presents basic information of what component model comprises. Secondly, creation process of the components is presented. At the end of this section, an automatic component creation tool wizard is presented.

4.1.1 Static and Dynamic Components

Simulation consist of static and dynamic components. Static components exist before the start of the simulation, and they typically represent factory floor devices, such as robots, conveyors, machines, and others. Static components are stored in component libraries from where they can be imported to the simulation environment. These components can also be created by end-users.

Dynamic components represent the products that are produced and processed in the layout during a simulation. These components are created by static components during the simulation, and they will be removed when the simulation ends. Static and dynamic components are equal in terms of component characteristics. However, dynamic components should be used as passive components without containing any executable elements, such as Python scripts. Figure 4.1 illustrates a static feeder component, which is creating dynamic rim components transferred to a static conveyor component.

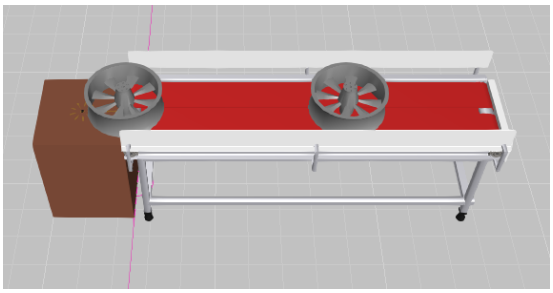

Figure 4.1. Dynamic components moving on a static conveyor component.

4.1.2 Structure of a Component

This section introduces basic and common attributes that comprise a component model. Component model consist of VC software specific attributes such as *properties*, *behaviors*, *features* within a node structure. Structure of a component model can be seen in Figure 4.2. Component consist of nodes that are formed by joints and links. Each node contains its own *features*, such as geometry and frames. The *behaviors* that describe the behavior of a component are typically located in the root node of a component. Parameters are used to parametrize the component.

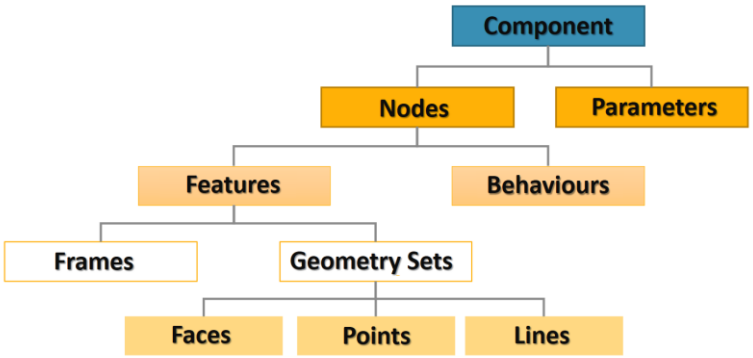

Figure 4.2. Structure of components.

Properties

A component *property* is a parameter that can be linked to *features* and *behaviors*. *Properties* can be read with Python script as well, which enables the possibility to utilize *property* values in component logics. There are several types of *properties*, such as real, integer, Boolean and distribution values, as well as string of characters. These types can be used when defining component dimensions, processing times, product filtering, failure times and probabilities, and other attributes of the component (see Figure 4.3).

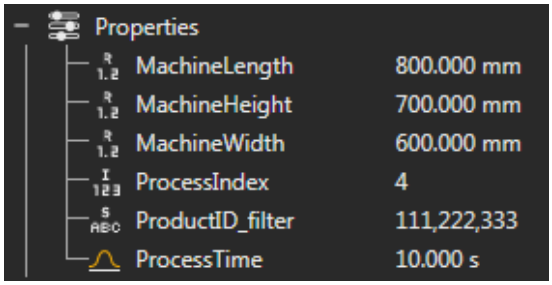

Figure 4.3. An example set of properties.

Features

Frames

Components are capable to include frames in their nodes. Frames can be used to define accurate locations and orientations of component *behaviors*, as well as to coordinate material flow.

Geometry

Geometries can be either imported or created with VC software. Geometries are converted into tessellated form, and they are constructed with faces, points and lines. When geometries are created with tools provided by VC software, the geometry can be assembled with primitive geometries, which are simple, configurable geometries with pre-defined shapes. There are also more advanced tools for the creation of the geometries, as well as there are tools to provide mirroring and cloning of the geometries.

Other Features

Transform *feature* modifies location and orientation of all *features* under the hierarchy of the transform *feature.* The location and orientation change is based on the values in the expression field of the transform *feature*. Switch *feature* enables to visualize only selected geometry from the group of geometries under the switch *feature*. The geometry selection is defined by the index number of the switch *feature*. One use case for this *feature* is to visualize the progression of the products during processes.

Behaviors

Containers and Paths

Dynamic components always require to be held by a container or a path *behavior* during the simulation, or they will disappear from the simulation. Container *behavior* is linked to a selected frame in a selected node. The part held by the container will move according to the frame movement. Path *behavior* functions as a container, but in addition to that, the part is also moving on a path. Pat*h behavior* is constituted by set of frames, such as start-frame, end-frame, and others depending on the use-case. Path *behaviors* are capable to transmit and receive parts with other paths by input and output ports. Python scripts can also be used to grab parts to a selected container or path.

Signals

Signals are typically used to trigger actions, visualize component states, and transfer information. Signals can be used as internal signals within the component or as external signals between components. Signals can be connected to Python scripts, where the signal value can trigger the execution of the script.

There are several different types of signals such as integer, real, Boolean, string and component signals. Boolean signals are frequently used signals, and they are typically used to control and indicate the status of the component. Component signal is a special type of signal, which transfers the component object as a signal. Component objects have information about the current location of the component in the simulation environment, which can be further used in part picking and placing operations as well.

Sensors

Path sensor *behavior* is connected to a selected frame of a path. The sensor can be connected to Boolean and component signals, which are triggered when a dynamic component reaches the sensor frame.

Interfaces

Interface *behaviors* enable connections between components. Interface defines how components transfer information to each other without exposing their internal details. There are several types of attributes that can be transferred via interface, such as signals, material flow, node hierarchy and joint values. To enable interface connection between components, the interfaces are required to be compatible and have matching set of interface attributes with each other. Interface matching could be illustrated with an electric plug, as seen in Figure 4.4, where the plug is able to connect only with matching interface. The purpose of restricting interface connections is to ensure that appropriate components are connected to each other.

Figure 4.4. An illustrative example of the connection criteria of interfaces.

Servos
Servo controller *behavior* is used to manipulate joints between nodes. Joints can be configured by the user to provide desired motion trajectory characteristics relative to links of the component. These joints are then connected to a servo controller. Thereafter, servo controller can be driven with a Python script.

Kinematics and Controller
Kinematics *behavior* is able calculate forward and inverse kinematic solutions for the kinematic chain of the component. This is used, for example, in robots to calculate complex motion trajectories. Kinematics *behavior* is connected to a robot controller *behavior*, which resembles servo controller *behavior*, but the controller is extended to utilize the kinematic *behavior* in trajectory calculations. Robot controller can be driven with a Python script, executor *behavior*, or with both.

Executor
Robots have a program editor tool for motion statement and logic statement planning. In this tool, tool center point of a robot can be dragged or snapped to a target location, and the current kinematic chain values can be stored to the statements. Robot can be effectively programmed using a set of these statements with logics statements. Executor *behavior* is the link between program editor tool and the component model. Executor can be connected and utilized with robot controllers and servo controllers. Executor statements can also be accessed and executed from Python script.

Physics
Physics *behaviors* enables the creation of colliders from geometries. These colliders can react to other colliders, which enables the interaction of forces between components. Physics *behaviors* are based on PhysX engine and it has three accuracies for recognizing the form of colliders, as seen in Figure 4.5. The utilization of force calculation in simulation requires continuous sampling techniques. However, the main simulation model is still based on discrete-event simulation. The use of physics *behaviors* may require significant amount of additional computing power.

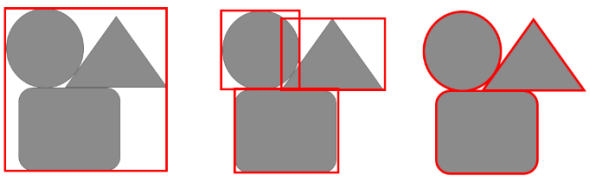

Figure 4.5. Different accuracies for physics colliders.

Statistics
The statistics *behavior* is used to collect and visualize data. The statistics can collect data from *behaviors*, such as containers. In addition, the statistics enables the utilization and tracking of component states, such as busy, idle and broken. These states can be recorded and visualized with the statistics as well.

4.1.3 Python Scripting

Python script acts as a glue between *properties*, *behaviors* and *features*, and almost anything in the components can be manipulated with a Python script. Python scripts include function that starts executing when the simulation starts running. This provides a method to run a script in a loop while the simulation is running. In addition, it is possible to execute a function every time an event-trigger is received by the script. However, scripts require to be edited and compiled when the simulation is not running.

VC software use Python version 2.7.1. Python is an interpreted, object-oriented and high-level programming language with dynamic semantics. Python syntax is rather simple and easy, as well as it emphasizes readability. One of the well-known characteristics of the Python language is the usage of whitespace indentation to delimit code blocks rather than brackets. (Python, 2018)

4.1.4 Creation of Components

The purpose of this section is to present creation process of a basic component. However, it should be noted that the modeling approach is case-specific, and there is not a single modeling approach for every component.

Import or Create Geometries
Geometries can be imported from Standard for the Exchange of Product Model Data (STEP) -format or created with VC software. If the geometry is imported, the tessellation quality of the geometry should be adjusted between quality and performance factor. Node structure is not importable, so if the imported geometry should have a node structure, it must be reassigned. Geometries can be also separated by specific tools, such as geometry split tool. Geometry separation is especially useful when working with assembly models, where geometries are merged. If geometries are created with the tools provided by VC software, the creation is typically performed simultaneously with the creation of *features* and node structure.

Origin, Features and Node Structure
The origin of the component should be placed to an appropriate component-specific position in the component. Thereafter, nodes are constructed from geometries that are extracted as links. The nodes can be organized hierarchically to form a hierarchical node structure. Each node can have an individual set of *features*, such as frames.

Behaviors and Properties
When the node structure is implemented, *behaviors* can be created based on the operational requirements of the component. Typically, there are several modeling approaches to achieve the same result. For this reason, selection of *behaviors* should be thoroughly considered. *Properties* are typically created during the whole modeling process, depending where they are used.

Write Scripts and Connect
When the structure of the components is defined, the last part is to write component logics to a script. Selected *properties* and *behaviors* are initialized in the Python script and they can be accessed and manipulated in the script. When a component is ready for use, it can be connected to other components with appropriate interfaces. Complex components might also require some debugging during the validation of the component.

4.1.5 Wizards

A wizard is a tool that instantly generates pre-defined and possibly configurable set of *properties*, *features* and *behaviors* to a component. The wizard is basically a Python script that is located in the application context, and not in a specific component. The script can provide a user-interface for the wizard, which can include objects such as check-boxes and lists. These objects can be dynamically hided and created according to interaction with the user.

The main purpose of wizards is to make component modeling faster, more standardized, and to reduce human errors and demand for repetitive work. There are different types of wizards for different purposes. Some wizards create only a few attributes, while other wizards can generate complete components.

4.2 Development Opportunities and Approaches

This section firstly presents the development opportunities in the field of component modeling which lead to development of component classification and best practices. This section secondly presents the development approaches for component classes and best practices for component modeling.

4.2.1 Development Opportunity: Component Classes

The classification of components is based on the selected industrial devices from the automotive and food & beverage industries. The selected devices are presented in sections 3.1.2 and 3.2.2. In some cases, the component class can be directly based on the real industrial device. However, in many cases, more creative approaches are required. Purpose of the classification is not to create classes supporting only specific devices in a certain industry, but to provide more widely applicable components. Components could be classified according to the operational principles and characteristics of the component, while the attributes related to the outfit, such as geometry and dimensions should not have significant impact to the classification.

Component classes provide standardized modeling solutions, which is a tool to improve consistency and reliability of the components as well as standardized solutions are easier to adopt. Classification has also an important role to define which components will be compatible with each other. In addition to compatible interfaces, components should be able to operate with each other. This may require specific type of operation principles from the components, such as a certain process workflow. In addition, hard-coded naming practices for *features*, *behaviors* and *properties*, which components are expecting to find from connected components, may be required.

These classes are also an effective tool to influence to the future of component modelling. One challenge in current component modelling is that most of the currently available components in the component library do not operate like real industrial devices from the control

point of view. One reason for the lack of this modeling approach is the unnecessary amount of scripting, which may enable some shortcuts to the logic implementations.

Real industrial devices typically operate with signal-based logic, where they utilize use of signals, such as sensor signals and communication signals between devices. These signals generally represent simple data types, such as Boolean values. The use of signal-based logic improves the capabilities of the components to be connected to external systems, which is a requirement in further digital twin and virtual commissioning applications.

4.2.2 Development Opportunity: Best Practices

Component modeling process includes several phases, and there are numerous things that the end-user should consider during the modeling process. Some of these things are software-specific, which typically requires some level of earlier use experience with the software.
To solve this challenge, best practices provide an effective method to assist end-users to utilize the most feasible modeling solutions.

Best practices gather the most relevant guidelines related to modeling into one compact entity. These guidelines include preferred proven to be good modeling solutions from the most relevant topics in the modeling field. In addition, best practices can be used to improve standardization of the modelling solutions.

4.2.3 Other Development Opportunities

Wizards
As automatic component creation tools, wizards have significant potential in component modeling. Currently, only a few wizards are provided with a standard installation of the VC software. End-users are capable to create and customize wizards for their own specific requirements. However, wizards could be implemented to a more generic level as well. Generic wizards could provide a customizable component framework, in which required details can be supplemented to already functioning component preforms. Generic wizards could also utilize information from the component classes.

Digital Twin
The approaches to develop digital twins are constantly evolving and no unanimous modeling solutions or general standardization are currently available. If comprehensively compatible digital twin solutions are targeted, then further standardization of digital twins is required. There are still development opportunities, that could be implemented for the current component modelling field.

To achieve real-time digital twins of production systems, simulation layouts should be capable to be driven by the physical system. This requires simulation components to be synchronized with the devices in the physical systems. VC software provides a connectivity feature that can be utilized to form signal connections between VC software and external systems. In addition to connectivity feature, simulation components must correspond the behaviors of the physical from the control point of view. Material flow in the physical system must be synchronized to simulation as well. This includes challenges in the real-time position monitoring of the material flow. If the real system is unable to know exact location of the product, neither can the simulation model.

A method to synchronize material flow with the simulation model could be developed. This method could utilize synchronization data based on the available sensors in the physical system. Sensor data could include, for example, position sensors, radio-frequency identification (RFID) readers, machine vision, ultrasound tracking and other techniques. If sensor data is not available, material flow can be estimated with the simulation model.

Independently functioning layouts without synchronization to physical systems could be still used for analyzation purposes. These layouts could provide near real-time feedback and optimized control commands to physical systems. Data framework for the components could also be developed. This would enable, as an example, manufacturing data to be stored to components in real-time. Due to scope of the thesis, further digital twin approaches are not implemented.

Formal Modeling Languages
Formal modeling languages, such as SysML provide abstract level user-interface for system design and control. These languages could be used as integrated user-interfaces for digital twins. In current component modeling, these languages could be technically applied to create complex systems. However, formal modeling languages may form a significant learning threshold for an average end-user. Due to the scope of the thesis, formal modeling language approaches are not implemented in this thesis.

4.2.4 Development Approach: Component Classes

This section firstly introduces component class approach for robots and end effectors, which are already widely found components in the current component library. Secondly, approach for generic machine class is introduced. Thirdly, approach for synchronous and asynchronous carrier conveyor classes are introduced.

Robots and End Effectors
Robots end effectors are classes derived directly from the selected industrial devices. These components are common components in the current component library of VC software. These components have already some unofficially classified attributes, which can be utilized in the classification.

Generic Machines
There is numerous amount of different types of machines. It is not feasible to attempt to create class for every type of machine. Chosen approach is to create generic machines, which are capable to correspond to multiple types of machines. This raises a challenge how to create generic machines that are practical, as well as capable to cover multiple types.

Classifying machines based on their process types, is challenging due to the amount and variation in different types of processes. A more feasible approach is to classify machines based on their material flow. Material flow types can be simplified to flow-based or resource-based material flow. Flow-based material flow transfers parts to other components with a flow interface. For example, a line of conveyors transfers parts with flow interface. Resource-based material flow transfers parts to other components with the aid of resources, such as humans or robots. These two material flow types provide the basis for the classification of generic machines. Generic machines should also utilize signal-based logics.

Synchronous and Asynchronous Carrier Conveyors
A carrier conveyor consists of a track and carriers. These carriers are attached to the track and they are capable to convey products. Carrier conveyors are especially common in the automotive industry. There are several types of these conveyors, and they may have significant differences in their operating principles. The operation of these conveyors could be classified as synchronous and asynchronous. In a synchronous conveyor, all carriers move synchronously, while in an asynchronous conveyor, the carrier movement is individual according to the type of the conveyor application.

4.2.5 Development Approach: Best Practices

Some of the best practices are generally applicable practices in the field of simulation, but many of the best practices are software specific. Best input for the development of best practices is the knowledge gathered from simulation experts, and the input from the simulation experts should play a major role in the development of best practices.

The simulation experts may have different type of core knowledge from different fields, such as robotics, manufacturing systems, software development and virtual commissioning. All of these fields include valuable information to the development of best practices.

Because there are experts with different backgrounds, they may not have similar approach and opinions in the same topics. For this reason, suitable approach for gathering input should be developed. First, the best practices should be divided into the most relevant topics in the field of component modeling. These topics provide a basis where the best practices will be supplemented. Thereafter, all ideas and best practice proposals are gathered. Based on this collected information, the most relevant best practice proposals will be chosen for further discussion. In addition, a discussion session with a group of experts should be arranged, to achieve the benefits of a peer review. The purpose of the discussion session is to achieve general consensus on the best practices based on the gathered proposals.

Best Practices Interview Form and Group Discussion
Development of best practices was done in two phases. First, a best practices interview form was shared with the group of simulation experts. Purpose of this form was to gather all best practice proposals that the simulation experts came up with. To ensure higher response rate, this form was stripped from weighting factors and other special formats. Second phase in the development of best practices was a group discussion with simulation experts, in which ideas and solutions are further developed and peer-reviewed.

5 Implementation

This chapter first presents the results of component classification and development of best practices. Thereafter, these results are analyzed. At the end of this chapter, a wizard for the generic machine is presented.

5.1 Results

5.1.1 Component Classification

Classification Form

Classification form defines the component classes, and it also plays a major role to share the information of the component classes. The form can be seen in Appendices A-F. The form is divided to the abstract and the detailed section.

The abstract section includes description, functional details and compatibility information of the class. The description field briefly describes the purpose of the component. Functional details describe the most important functional characteristics of the component. The compatibility field describes where the component can be connected. The model details describe information that is required to model the component such as component *properties*, *behaviors*, *features* and a node structure.

Robot and End Effector

The class of a robot and end effector were derived based on components already found in the component library. The most essential attributes from group of these components were selected to form these classes. Classification form for the robot and effector can be found from the Appendix A and B.

Generic Machines

Generic machines can be classified to a start-of-line, end-of-line, stand-alone and in-line machines (see Figure 5.1) (see Appendices C, D, E and F). The end-of-line machine receives part from the previous component via a flow interface and the resource picks the part from the machine. The stand-alone machine receives part from the resource and the resource picks the part from the machine. The start-of-line machine receives part from the resource and it transmit parts to the next component via a flow interface. The in-line machine receives and transmit part via a flow interface.

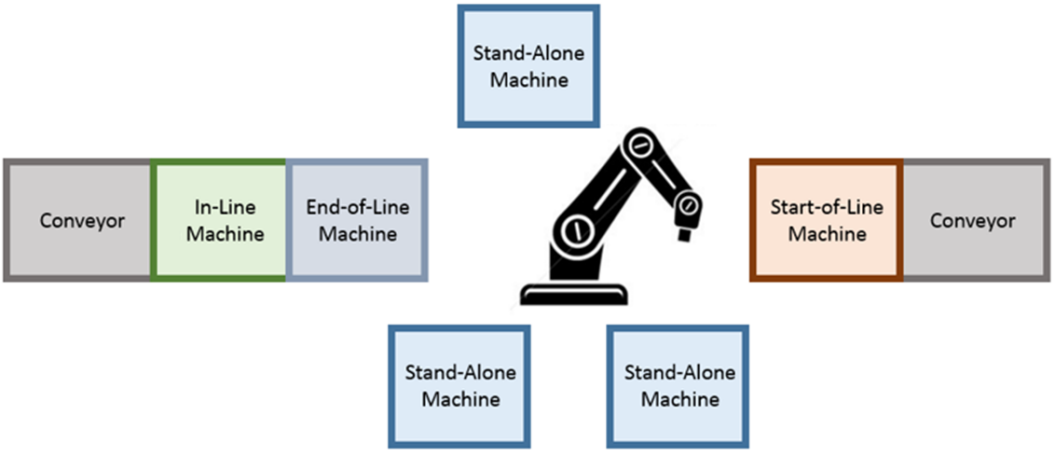

Figure 5.1. The concept of generic machine with material flow from left to right.

Material flow is based on transport operations executed by resources, as well as flow interfaces. A transport operation requires two Boolean signals from transmitter and receiver components. These signals are divided to two one directional signals. Request-signals request transport operation from the resource manager, and status-signals indicate when the transport operation is completed. This concept can be seen in Figure 5.2.

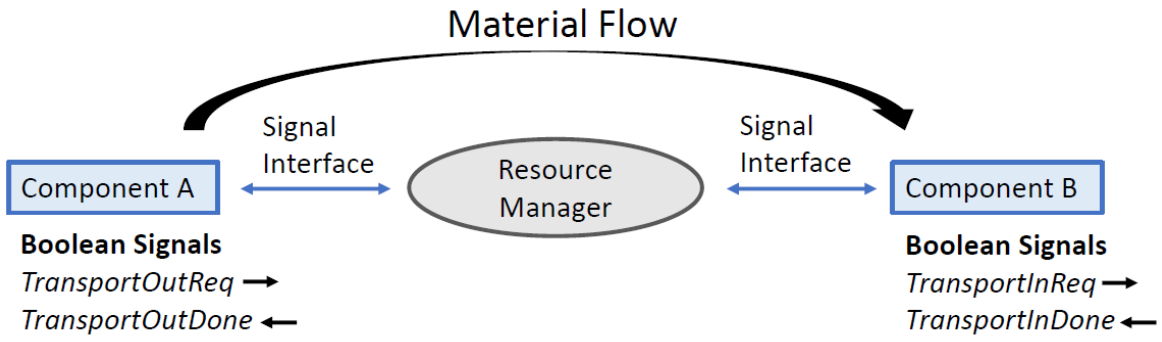

Figure 5.2. The principle of Boolean signal based transport operations. Black arrows after the signals indicate the direction of the signals.

The first phase of a transport operation is to have true values on both *TransportOutReq* and *TransportInReq* signals, seen in Figure 5.3. The second phase is to check, if the transport operation is possible, which is evaluated by the resource manager. The evaluation criteria are user-defined. If the evaluation result is approved, the transport operation can be executed. When the part is taken from the component, the value of *TransportOutDone* is set as true, and when the part is transported to the next component, the value of *TransportInDone* is set as true.

To achieve continuous execution of these transport operations, some additional logics are required. After a component sets *TransportOutDone* or *TransportInDone* signal to true, it immediately sets *TransportOutReq* or *TransportInReq* signal to false. When the component sets *TransportOutReq* or *TransportInReq* signal as true, the resource manager sets *TransportOutDone* or *TransportInDone* signal as false.

Machines that utilize resources such as robots or humans for the material flow, have container *behaviors* to contain components. These containers are also used to coordinate material flow. Containers are used to find the pick and place locations for the transport operations. The containers have hard-coded names such as *Process1Container* and *Process2Container,* which can be accessed from other components while executing transport operations.

Additional functionalities for the machines can utilize signal-based logics as well. Stand-alone machines may include doors that are operated during process cycles. The door logics are implemented with one directional Boolean control and status signals. There can be control signals for opening and closing the doors, and there can be status signals indicating if the doors are at open or closed state. Example set of stand-alone machine signals with door logics can be seen in Figure 5.3.

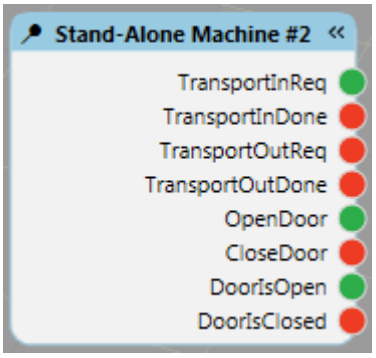

Figure 5.3. Example set of stand-alone machine signals, including door logics.

Generic machine types provide a basis for the material flow characteristics of the machine. However, real machines typically perform processes for products in addition to only routing and holding parts for certain amount of time. These processes could be standardized as well. Five examples of processes are presented below:

- Servo Process

The servo process moves component joints that are configured in selected servo controller *behavior*. Movement is executed according to motion data stored in a note *behavior*.

- Inspection Process

The inspection process evaluates products, and gives them a Boolean *property* value according to pre-defined probability, which is defined in a distribution *property* of the inspector component.

- Product Packaging Process

First, a pre-defined amount of product components is collected. Thereafter a package component is created, in which the product components are packaged. Product data such as component-specific ID and *properties* are stored to a note *behavior* in the package component as a string value. After the data is stored, product components can be deleted, and the package component is ready to be handled. Because all relevant data is stored from the products, the packaging process can be reversed.

- Change Geometry Process

Geometry of the product is changed with a switch *feature*. The switch *feature* requires alternative geometries to already exist under the hierarchy of the switch *feature*. The geometry change is triggered by an integer *property*.

- Assembly Process

The Assembly process assembles products to each other. This process requires products to have information of their parent components, and the location and orientation relative to their parent components.

Synchronous Carrier Conveyor

The synchronous carrier conveyor system can be seen in Figure 5.4. Due to time limits of this thesis, no detailed classification form, or simulation validation were made. The synchronous carrier conveyor is divided to following classes:

- Line

  The line component forms a path for the carriers. Line components are connected to each other with flow interfaces. Line components inherit line parameters such as path speed from previous line components.

- Carrier Feeder – Synchronous

  The carrier feeder component is connected to a selected line component as a child component via an interface. The feeder creates carriers to the conveyor line when a simulation starts. In the synchronous conveyor carriers can be created instantly or one by one.

- Controller – Synchronous

  The controller component is connected to a selected line component as a child component. The controller is connected to the sensors of the conveyor system via a remote interface. The controller manages conveyor operations, such as stopping and starting the motion of the conveyor line according to the sensor signals. There should be only one controller per synchronous conveyor system.

  In synchronous conveyor systems, multiple carrier operations such as pick and place of products can be executed during one stop cycle. When multiple simultaneous operations are executed, carriers have to be stopped simultaneously. However, multiple location sensors do not trigger an event at the exact same time instant. Sensors could be grouped in the controller component to provide a general condition, in which all sensors in the group are required to have true value to continue execution.

- Carriers

  Carriers are dynamic components that can be divided to active and passive carriers. Both carriers include component containers to carry products. Active carriers are capable to execute carrier-specific actions, such as pick and place of products, unlike passive carriers. However, active carriers are not able to execute actions independently via script, which is not recommended for any dynamic components. Actions are performed with utilizing external components, such as sensors.

- Location Sensor
  The location sensor is attached to a line, and it indicates if a carrier is at the sensor position with a Boolean value.

- Stop & Go Sensor – Synchronous
  Stop & go sensor is capable to switch the line motion of a conveyor to on/off state. The line motion is controlled with stop and go Boolean signals of the sensor, which are connected to the controller via a remote interface.

- Carrier Action Sensors – Synchronous
  Carrier action sensors execute carrier actions and they are compatible only with active carriers. The pick sensor controls a carrier to pick a product, while the place sensor is utilized to place a product. The process sensor controls a carrier to execute process specific actions. The action sensors are connected to the controller via a remote interface.

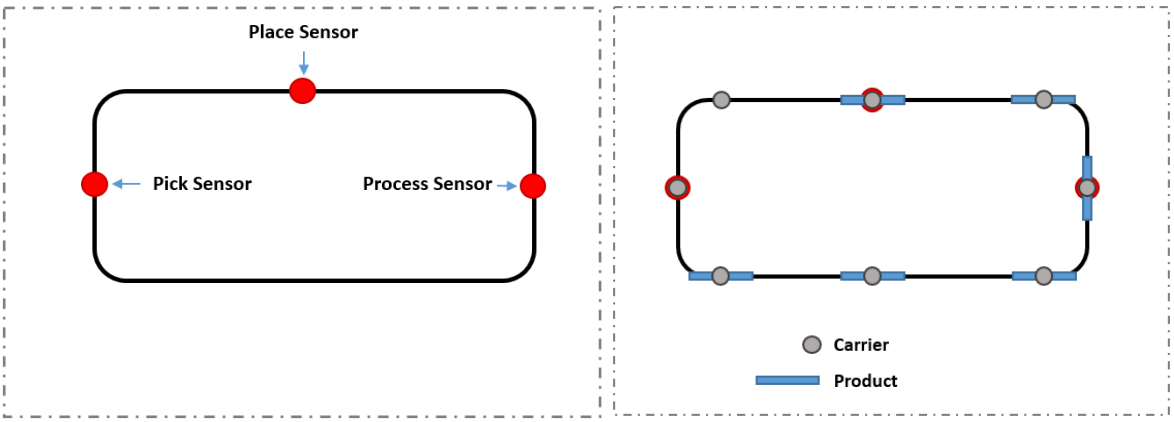

Figure 5.4. An illustrative example of synchronous conveyor system without and with carriers.

Asynchronous Carrier Conveyor
The concept of synchronous carrier conveyor system can be seen in Figure 5.5 and Figure 5.6. Due to time limits of this thesis, no detailed classification form, or simulation validation were made.

In asynchronous carrier conveyors, carriers are capable of moving individually. Two methods are utilized to handle carrier flow. In the first method, carrier routes are pre-defined. Carrier route information is composed of track nodes, which are components capable to influence the carrier flow such as diverters, routers and lifts. Individual carrier route information is stored in each of the carriers. In the second method, list of processes is pre-defined for each carrier, and the route is defined dynamically at each track node. The process list and process index are stored in each of the carriers. This method also requires the use of the controller component.

The asynchronous carrier conveyor is divided to following classes:

- Line
  This component is identical to the line component in synchronous carrier conveyors.

- Carrier Feeder – Asynchronous
  The carrier feeder component is connected to a selected line component as a child component. The feeder creates carriers to the conveyor line one by one. Material flow information of the carriers is configured in this component. Carrier routes are defined and configured in a note *behavior* of this component and carrier-specific route information is given to each carrier as a string *property*. Routes are defined with track nodes of the conveyor track. If the process list is used for material flow instead, the process list is given to carriers as a string *property*.

- Line Merger and Diverter – Asynchronous
  The line merger merges two input lines into one output line, and line diverter diverts one input line into two line outputs. Both merger and diverter components have an integer *property* to define the currently active port.

- Turn Router – Asynchronous
  The turn router is capable to switch the current route configuration. The router reads the carrier route information, thereafter the router configuration is switched according to the information.

- Lift – Asynchronous
  The lift is capable to transports carriers between tracks. The line lift reads the carrier route information, which defines the lift action.

- Carriers
  Carriers are identical to carrier components in synchronous carrier conveyor, except the additional route or process information, which is added by asynchronous carrier feeder component.

- Location Sensor
  This component is identical to the location sensor component in synchronous carrier conveyors.

- Stop & Go Sensor – Asynchronous
  The stop & go sensor is capable to switch the motion of a single carrier to on/off. The motion is controlled with a stop and go Boolean signals of the sensor.

- Carrier Action Sensors – Asynchronous
  These components are based on the carrier action sensor components in synchronous carrier conveyors. In asynchronous conveyor system, action sensors are capable to operate independently without an interface connection to controller.

- Controller – Asynchronous
  The controller is optional component for asynchronous conveyor systems. The controller can be used when the process list is utilized for dynamic material flow instead of fixed carrier routes. The controller is capable to read the status of the sensors and control the track nodes to influence the material flow. The controller logic is required to be further defined by the user, and one conveyor system could include multiple controllers.

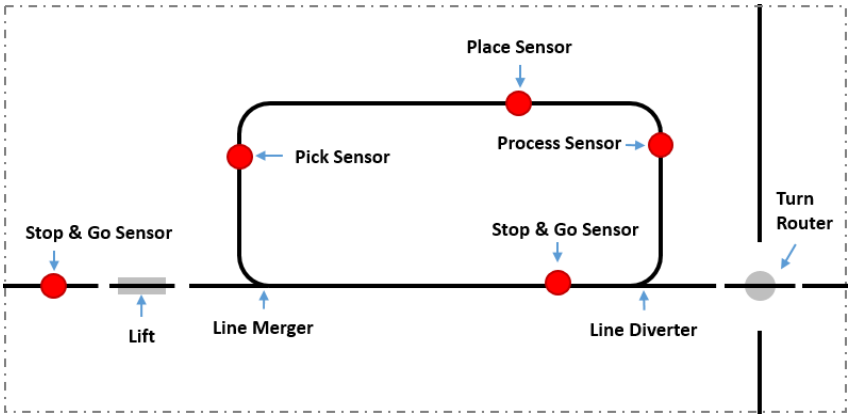

Figure 5.5. An illustrative example of an asynchronous conveyor system without carriers.

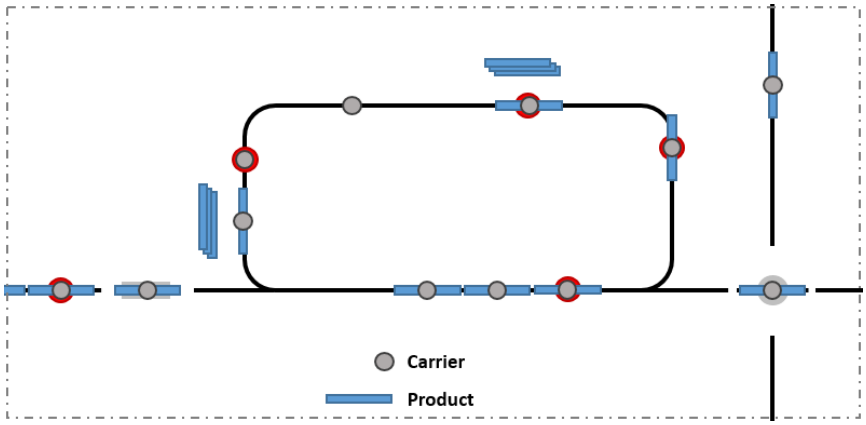

Figure 5.6. An illustrative example of an asynchronous conveyor system with carriers.

5.1.2 Best Practices

The development of best practices started by creating topics for the best practices. These topics are presented below:

- Features, Behaviors, Properties
  These topics discuss modeling practices related to general building blocks of the components.

- Naming Conventions
  Naming conventions play an important role at expressing how the components function.

- Python Scripting
  This topic discusses general scripting practices.

- Robustness, Reusability
  These topics discuss the reliability and reusability of components.

- Performance of the Component
  The performance topic discusses methods to reduce the amount of required computing resources during a simulation.

- Component Connectivity
  The component connectivity topic discusses the capabilities of components to be connected to external systems, such as logic controllers.

After the topics were selected, an interview form was created based on these topics. The interview form and the collected results can be seen in the Appendix G. Based on these results, two best practices group discussions were organized. The collected results from the interview forms were discussed in the group discussions. These discussions led to the development of best practices. The best practices can be seen in the Appendix H.

5.2 Analysis of the Results

5.2.1 Analysis of the Classification

Classification Form

The classification form containing classification criterion can be seen in Appendixes A-F. The form provides comprehensive and quickly adoptable method of classifying components. However, the form is laborious to fill and because of the amount of manual work, it is error-prone. In addition, multi-purpose and highly configurable components may be challenging to define with the form.

The improved solution could be to utilize a component itself to describe the component details. This would require the component itself to be well-documented, including descriptions of signals, interfaces, frames and other important attributes. This would enable faster, more coherent and error-proof creation of the component classes. However, the major disadvantage is that the component classes could not be presented in a static 2D format.

Classes - Robots and Accessories

The classes for robots and accessories corresponds currently available components in the component libraries. For this reason, no further analysis is performed.

Classes - Generic Machines

The classification of generic machines provides a basis for modular and simple components that follow coherent and transparent control methods based on signals. Utilization of signals in material flow enables the it to be comprehensively configured without special knowledge of complex scripts. In addition, the signal based control methods enable components to be externally connected form the software.

However, generic machines introduce a new concept that must be learned. Also, in several cases the generic machines provide only a basis for the components. This results in the requirement to create scripts and other logic attributes, such as signals, to achieve a fully functioning component. In some complex and multi-purpose components, signal-based logics can suffocate the component with massive amounts of signals, which makes the component more challenging to adopt. The generic machines serve certain purposes, and it may not be

the most feasible approach for every case. Nevertheless, in many cases it may provide significant value for the end-users.

Classes – Synchronous & Asynchronous Carrier Conveyors
The purpose of both synchronous and asynchronous carrier conveyors is to provide environment for a wide range of conveyor applications. These conveyors should also be comprehensively extensible for additional modifications. The operation principle of asynchronous carrier conveyors resembles power and free conveyors, but is not limited to these solutions.

The generic concept of synchronous and asynchronous carrier conveyors retains similar challenges as generic machines. These conveyors may require to be further supplemented to specific use-cases. Other current challenge with asynchronous conveyor systems is the recognition of carrier component collisions that exceed line component boundaries. One possible solution is to develop complex script logic to prevent the collisions.

5.2.2 Analysis of Best Practices

The topics of the best practices provide an effective method to filter less important best practices according to the current use-case. The best practices are easily adoptable, and they facilitate end-users to guide their focus and time with issues that can significantly improve the quality of the component. However, the best practices are unable to provide comprehensive solutions in several cases. This is because, the solutions are typically highly case-specific and inappropriate modeling solutions could be misleading. One possible approach would be to link a comprehensive set of examples with the best practices.

5.3 Case Example: Generic Machine Wizard

The wizard is implemented to generate generic machines. The wizard consists of a set of Python scripts (see Figure 5.7). *MachineWizard.py* script provides a user-interface to configure the wizard functions. When the wizard is executed, *MachineWizard.py* script generates component attributes according to selected user-interface values. Machine scripts (*StandAloneMachine.py, StartOfLineMachine.py, EndOfLineMachine.py, InLineMachine.py*) provide a basis for the component logics (see Appendix I). These scripts can be further customized by the *MachineWizard.py* script. Thereafter, the machine script is inserted to a Python script *behavior* of the generated machine component.

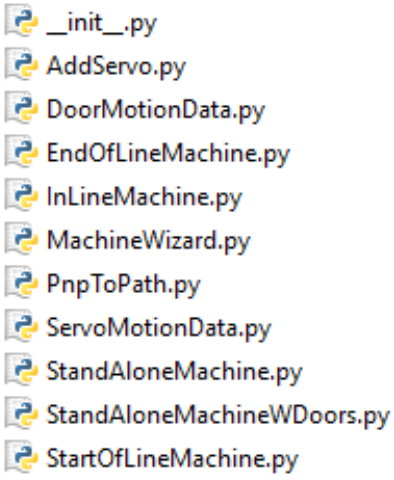

Figure 5.7. The structure of the generic machine wizard

Wizard in use

The wizard should be applied to an empty component including only geometry *features*. An exception is made if joint movement is required. In that specific case, joints must be defined and the controller *behavior* must be configured before the execution of the wizard. A component with a one joint for door movement and two joints for process motion can be seen in Figure 5.8.

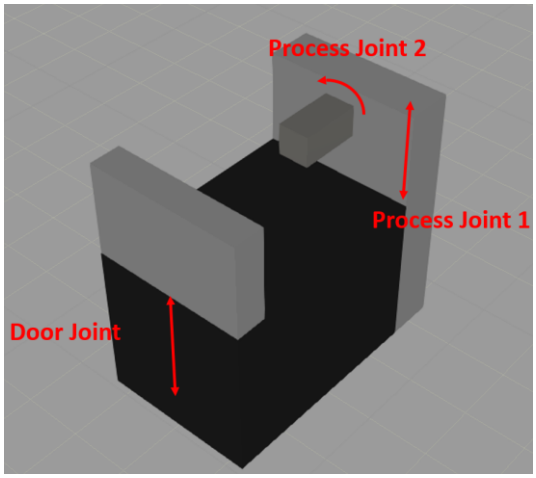

Figure 5.8. A component with three joints ready to be further generated by the wizard.

User-interface of the wizard can be seen in Figure 5.9. The first option is to define the machine type, which has influence to further configurability options of the wizard. For a stand-alone machine, the wizard currently provides options to select if doors are included and if a process is included. Selecting these options will provide more detailed configurable options, such as selecting the appropriate controller *behavior* or selecting the appropriate node for material flow input. Without any additional configuration, the machine wizard will only create a machine based on the defined machine classes.

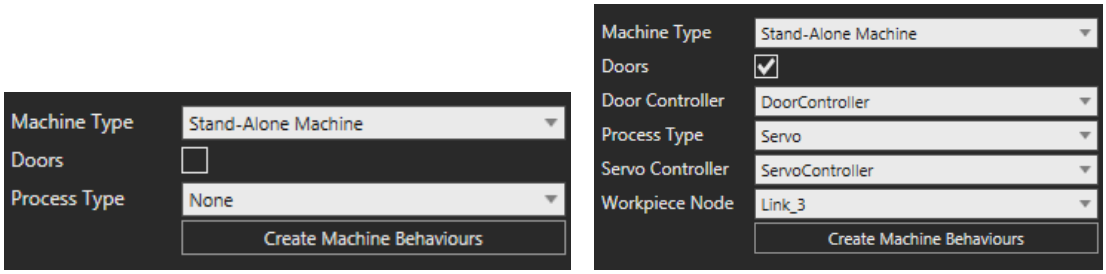

Figure 5.9. On the left is the wizard user-interface for the stand-alone machine with no additional options chosen. On the right is the same user-interface with additional options chosen.

The layout with the machines created by the wizard can be seen in Figure 5.10. Signal representation is set visible to indicate the values of the signals in the system. The robot seen in the figure is controlled by a robot controller component, which operates as a resource manager. The controller receives request signals, transmits material flow status signals, and executes transport actions if the criteria for the transport operations are fulfilled.

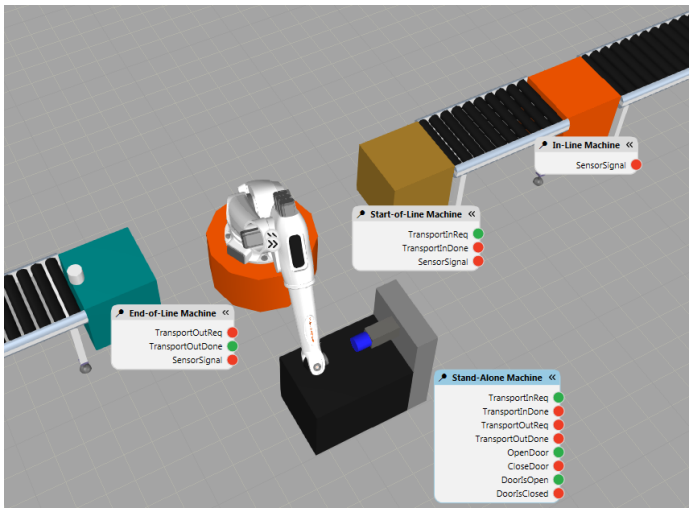

Figure 5.10. A layout in which end-of-line, start-of-line, and in-line machines are connected to conveyor paths. Signal status of the components is set to visible.

The start-of-line, end-of-line, in-line machines have an extra option to be connected to a path *behavior* of a component. The user-interface for start-of-line machine can be seen in Figure 5.11. The layout with machines connected to a component path can be seen in Figure 5.12.

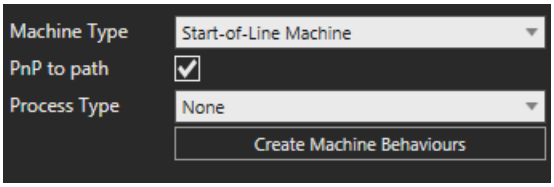

Figure 5.11. The wizard configurability for the start-of-line machine. "PnP" is abbreviated from plug and play, and the term is typically used when connecting components.

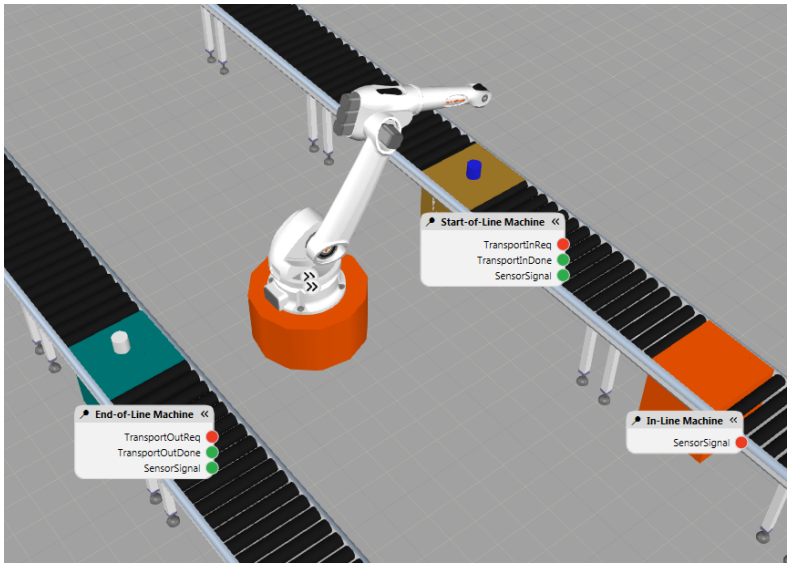

Figure 5.12. A layout in which end-of-line, start-of-line, and in-line machines are connected to the conveyor paths.

6 Conclusions and recommendations

6.1 Conclusions

This thesis examines potential future prospects of production simulation software. The digital twin modeling approaches presented in the literature generally indicate that the digital twin will become increasingly important tool in engineering processes, and it will integrate all relevant information in the whole lifecycle of a product. This results in the current document-centric engineering to develop into model-based engineering.

However, there are challenges concerning the digital twin implementations. There is generally no consensus about the detailed technical solutions in the development of digital twins. Due to the lack of standardization, comprehensive digital twin approaches may require a lot of resources with no guarantee of compatibility to future systems.

In addition to digital twins, this thesis presents formal modeling languages. These languages are effective tools to provide abstract and standardized representations of complex models. The use of these modeling languages could be integrated to digital twins as well. These languages could also play a major role in the creation of formal interfaces between different systems.

The component classification is developed based on selected devices from the automotive and food & beverage industries. The target of the developed component classes is to create interdisciplinary components that are not limited to specific industrial fields. This leads to creation of the generic machine and carrier conveyor concepts. These concepts provide highly configurable and extensible components. The generic machine is further developed to be generated with a wizard. This wizard acts as a proof of concept, and it demonstrates the capabilities of configurable and instant component creation solutions.

The development of best practices for component modeling is based on the selected topics in the field of component modeling. Input for the best practices is collected from the simulation experts in the case company with an interview form. Based on the input, group discussion sessions were organized with these experts to review and select the most appropriate best practices. Best practices are an effective method to guide end-users to focus their time on the most relevant modeling tasks.

The development of component classes and best practices should be continued. The generic machine concept could be extended to provide more functionalities, and synchronous and asynchronous carrier conveyors could be further tested and validated. More classes could be developed based on the concept of generic machines. In addition, the model-based classification method as mentioned in section 5.2.1 could be further implemented. The best practices could be extended. In some practices, more detailed solutions and example proposals could be given, as mentioned in section 5.2.2. Best practices for specific classes could also be developed. These practices would contain more detailed class-specific practices.

6.2 Suggestions for Further Work

In the future, the development of digital twin modelling approaches should be pursued. The particularly interesting subject is to follow if any digital twin related technical solution rises above others. The general trend of industry 4.0 must also be followed. One important topic of industry 4.0 is the autonomy, in which the digital twin is expected to play a major role (Rosen, et al. 2015). Steps towards the digital twin concept could also be taken in the Visual Components software. As presented in section 4.2.3, a data framework *behavior* to gather production data and real-time mirroring capabilities to represent factory floor actions could be developed.

Bibliography

ASAS, (2018). ASAS Systems. [online]. Available at: http://www.asas.es/assets/uploads/proyectos/3c497-clip_image001.jpg [Accessed 2 Jan. 2018]

ASI, (2018). *EMS – Electrified Monorail Systems.* Automatic Systems, Inc. [online]. Available at: http://www.asi.com/auto-industrial/overhead-conveyors/ems.php [Accessed 2 Jan. 2018]

AutomationML. (2014). *The Glue for Seamless Automation Engineering*. [online]. Available at: https://www.automationml.org/o.red/uploads/dateien/1417686950-AutomationML%20Whitepaper%20Part%201%20-%20AutomationML%20Architecture%20v2_Oct2014.pdf [Accessed 2 Jan. 2018]

Baheti, R. Gill, H. (2011). *Cyber-Physical Systems*, in: The Impact of Control Technology, 12, 161–166. Available at: http://ieeecss.org/sites/ieeecss.org/files/documents/IoCT-Part3-02CyberphysicalSystems.pdf [Accessed 10 Dec. 2017]

BMW Body Shop, (2018). BMW. [online]. Available at: https://www.bmwusfactory.com/manufacturing/production-process/body-shop/ [Accessed 2 Jan. 2018]

BMW Assembly, (2018). BMW. [online]. Available at: https://www.bmwusfactory.com/manufacturing/production-process/assembly/ [Accessed 2 Jan. 2018]

Friedenthal, S. Moore, A. Steiner, R. (2015). *A Practical Guide to SysML: The Systems Modeling Language*. 3rd ed. The MK/OMG Press. [eBook]. ISBN 978-0-12-800202-5

Direct Industry Cartoning, (2018). Direct Industry. [online]. http://img.directindustry.com/images_di/photo-g/123957-7195183.jpg [Accessed 2 Jan. 2018]

Direct Industry Merge, (2018). Direct Industry. [online]. Available at: http://www.directindustry.com/prod/transnorm-system-gmbh/product-58734-536377.html [Accessed 2 Jan. 2018]

Direct Industry Positioner, (2018). Direct Industry. [online]. Available at: http://img.directindustry.com/images_di/photo-m2/17587-5294509.jpg [Accessed 2 Jan. 2018]

Drath, R. Lüder, A. Perschke, J. and Hundt, L. (2008). *AutomationML – The Glue for Seamless automation engineering*, in: Emerging Technologies and Factory Automation (ETFA). IEEE International Conference. DOI: 10.1109/ETFA.2008.4638461

DMW, (2018). *Skillet Systems*. Dearborn Mid-West Company. [online]. Available at: http://www.dmwcc.com/skillet-systems [Accessed 2 Jan. 2018]

DMW P&F, (2018). *Overhead / Inverted Power and Free Systems*. Dearborn Mid-West Company. [online]. Available at: http://www.dmwcc.com/inverted-power-free-systems [Accessed 2 Jan. 2018]

DS, (2018). *Design Systems*. Design Systems, Inc. [online]. Available at: http://www.dsidsc.com/images/ph-me-complete-carrier-design-overhead.jpg [Accessed 2 Jan. 2018]

Dürr, (2018). *Skid Conveyor*. Dürr. [online]. Available at: http://www.durr-paint.com/paint-systems-products/final-assembly-and-conveyor-technology-products/conveyors/skid/ [Accessed 2 Jan. 2018

Dürr Paint, (2018). *Complete Paint Shops for the Automotive Industry and its Suppliers*. Dürr. [online]. Available at: http://www.durr.com/fileadmin/user_upload/duerr/en/pdf/pas/Paint-Brochure-english.pdf [Accessed 2 Jan. 2018]

Gebo Cermex, (2018). Gebo Cermex. [online]. Available at: http://fr.gebocermex.com/equipement/palettisation-d%C3%A9palettisation/palettisation/cellule-de-palettisation-compacte-robotis%C3%A9e [Accessed 2 Jan. 2018]

Gianni, D. D'Ambrogio, A. and Tolk, A. (2017). *Modeling and Simulation-Based Systems Engineering Handbook*. CRC Press. [eBook]. ISBN 9781138748941

Glaessegen, E. and Stargel, D. (2012). *The Digital Twin Paradigm for Future NASA and U.S. Air Force Vehicles*, in: Paper for the 53rd Structures, Structural Dynamics, and Materials Conference: Special Session on the Digital Twin

Grieves, M. (2014). *Digital Twin: Manufacturing Excellence Through Virtual Factory Replication*. [online]. Available: http://innovate.fit.edu/plm/documents/doc_mgr/912/1411.0_Digital_Twin_White_Paper_Dr_Grieves.pdf [Accessed 1 Oct. 2017]

Grieves, M. and Vickers, J. (2016). *Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems.* [online]. Available: http://research3.fit.edu/camid/documents/doc_mgr/1221/Origin%20and%20Types%20of%20the%20Digital%20Twin.pdf [Accessed 1 Oct. 2017]

GR IAS, (2016). GR Industrial Automation. [online]. Available at: http://www.grindustrialautomation.com/images/gr/press1.jpg [Accessed 2 Jan. 2018]

Hart, L. (2015). *Introduction to Model-Based System Engineering (MBSE) and SysML*. [Online]. Available at URL: http://www.incose.org/docs/default-source/delaware-valley/mbse-overview-incose-30-july-2015.pdf

Hyundai, (2017). Hyundai WIA [online]. Available at: http://en.hyundai-wia.com/business/machinery_press01.asp [Accessed 31 Oct. 2017]

IMH, (2017). *WP-content*. International Material Handling, Inc. [online]. Available at: http://internationalmaterialhandling.com/wp-content/uploads/2017/05/1.png [Accessed 2 Jan. 2018]

ISRA vision, (2018). *Body Shop*. ISRA Vision[online]. Available at: http://www.isravision.com/javascript/automotive/produktionsstrasse/assets/images/2_body_shop_small.png [Accessed 2 Jan. 2018]

Knight, (2015). *Lift Assists*. Knight. [online]. Available at: http://www.knight-ind.com/design_gallery.html [Accessed 8 Nov. 2017]

Koo LJ, Park CM, Lee CH, Park SC, Wang GN, (2011). *Simulation framework for the verification of PLC programs in automobile industries,* in: International Journal of Production Research. 2011; 49(16): 4925-4943

KUKA, (2018). *Standard Positioners*. KUKA [online]. Available at: https://www.kuka.com/en-de/products/robot-systems/robot-periphery/positioners/standard-positioners [Accessed 2 Jan. 2018]

Lasi, H., Fettke, P., Feld, T., and Hoffman, M. (2014). *Industry 4.0, in: Wirtschaftsinformatik*. doi: 10.1007/s11576-014-0424-4*,* in: Business & Information Systems Engineering, August 2014, Volume 6, Issue 4, pp 239–242

Lee, E. (2008). *Cyber-Physical Systems: Design challenges. In Object Oriented Real-Time Distributed Computing*, in: (ISORC), 200811th, IEEE International Symposiumon,363–369.IEEE.

Lenze, (2018). *Skillet-line conveyors: assembly platforms in continuous operation.* Lenze [online]. Available at: http://www.lenze.com/fi-fi/asiantuntijoiden-lausunnot/automotive/final-assembly/skillet-line-conveyor/ [Accessed 2 Jan. 2018]

Liu, Z. Diedrich, C. and Suchold, N. (2012). *Virtual Commissioning of Automated Systems*, in: INTECH, [online]. Available at: http://cdn.intechopen.com/pdfs/37992/InTech-Virtual_commissioning_of_auto-mated_systems.pdf. DOI: 10.5772/45730 [Accessed 27 Oct. 2017]

Mayerhofer, T., Wimmer, M., Berardinelli, L., Maetzler, E., Schmidt, N., (2016). *Towards Semantic Integration of Plant Behavior Models with AutomationML's Intermediate Modeling Layer,* in: Proceedings of the 4th International Workshop on The Globalization of Modeling Languages (GEMOC 2016), CEUR Workshop Proceedings, 1731 (2016), S. 28 – 37

McGuire, P. (2009). *Conveyors: Application, Selection, and Integration.* CRC Press. [eBook]. ISBN 978-1-4398-0390-5

Micro-Epsilon, (2018). *Fully automatic surface inspection of painted car bodies*. Micro-Epsilon [online]. Available at: https://www.micro-epsilon.com/measurement-systems/Paint-Inspection/karosserie/ [Accessed 2 Jan. 2018]

MK, (2018). MK North America, Inc. [online]. Available at: https://www.mknorthamerica.com/Customer-Content/www/Products/Photos/Full/ZRF-P2040.02-2.png [Accessed 2 Jan. 2018]

Modrak, V., and Semanco, P., (2014). *Handbook of Research on Design and Management of Lean Production Systems*. IGI Global. [eBook]. ISBN 978-1-4666-5040-4

Monostori, L., (2014). *Cyber-Physical Production Systems: Roots, Expectations and R&D Challenges*, in: Procedia CIRP Volume 17, 2014, Pages 9-13. DOI: 10.1016/j.procir.2014.03.115

Mourtzis, D., Doukas, M., and Bernidaki, D. (2014). *Simulation in Manufacturing: Review and Challenges*., in: Procedia CIRP Volume 17, 2014, Pages 9-13. DOI: 10.1016/j.procir.2014.10.032

OMG, (2018). *What is SysML*. Object Management Group, [online]. Available at: http://www.omgsysml.org/what-is-sysml.htm [Accessed 2 Jan. 2018]

P&F Mechanics. [online]. Available at: https://fx8ds4d92ag4dbov01l0m0x1-wpengine.netdna-ssl.com/wp-content/uploads/2015/06/Power-and-free-conveyor-function.png [Accessed 2 Jan. 2018]

PLCopen, (2009). *XML Formats for IEC 61131-3*. PLCopen Technical Committee 6, [online]. Available at: http://www.plcopen.org/pages/tc6_xml/downloads/tc6_xml_v201_technical_doc.pdf [Accessed 27 Oct. 2017]

Python, (2018). *What is Python? Executive Summary*. Python. [online]. Available at: https://www.python.org/doc/essays/blurb/ [Accessed 2 Jan. 2018]

Radpak, (2014). Radpak. [online]. Available at: http://www.radpak.net/machines,end-of-line-systems,case-erector.html#prettyPhoto [Accessed 2 Jan. 2018]

Rosen, R., Wichert, G., and Lo, G. (2015). *About The Importance of Autonomy and Digital Twins for the Future of Manufacturing*, in: IFAC-PapersOnLine Volume 48, Issue 3, 2015, Pages 567-572.

Rüssman, M., Lorenz, M., Gerbert, P., et al. (2015). *Industry 4.0: The Future of Productivity and Growth in Manufacturing Industries. Boston Consulting Group 9*. [Online]. Available at: http://www.inovasyon.org/pdf/bcg.perspectives_Industry.4.0_2015.pdf [Accessed 10 Nov. 2017]

Schluse, M., Rossmann, J., (2016). *From simulation to experimentable digital twins: Simulation-based development and operation of complex technical systems*, in: 2016 IEEE International Symposium on Systems Engineering (ISSE). pp. 1–6. DOI:10.1109/SysEng.2016.7753162

Schluse, M., Linus Atorf, et al., (2017). *Experimentable Digital Twins for Model-Based System Engineering and Simulation-Based Development*, in: System Conference (SysCon), 2017 Annual IEEE International. DOI: 10.1109/SYSCON.2017.7934796

Schmidt, N. Lüder, A. (2017). *The Glue for Seamless Automation Engineering. AutomationML*, [Online]. Available at: https://www.automationml.org/o.red/uploads/dateien/1447420977-AutomationML%20in%20a%20Nutshell_151104.pdf [Accessed 3 Nov. 2017]

Schroeder, G. et al., (2016). *Digital Twin Data Modeling with AutomationML and a Communication Methodology for Data Exchange*. in: IFAC-PapersOnLine Volume 49, Issue 30, 2016, Pages 12-17. DOI: 10.1016/j.ifacol.2016.11.115

STM, (2018). *Progressive Dies*. STM Tooling. [online]. Available at: http://www.stmtooling.com/images/progressive-dies1.png [Accessed 2 Jan. 2018]

SysML, (2015). *OMG Systems Modeling Language*. SysML. [online]. Available at: http://sysml.org/docs/specs/OMGSysML-v1.4-15-06-03.pdf [Accessed 31 Oct. 2017]

Syväjärvi T, (2016). *Requirements of Simulation Software for Virtual Commissioning of Discrete Manufacturing Systems.* Master's Thesis. Tampere University of Technology.

Tao, F. Cheng, J., Qi, Q., Zhang, M., Zhang, H., and Sui, F. (2017). *Digital twin-driven product design, manufacturing and service with big data*, in: The International Journal of Advanced Manufacturing Technology. DOI: 10.1007/s00170-017-0233-1

Uhlemann, T., Lehmann, C., and Steinhilper, R. (2017). *The Digital Twin: Realizing the Cyber-Physical Production System for Industry 4.0,* in: The 24th CIRP Conference on Life Cycle Engineering. DOI: 10.1016/j.procir.2016.11.152

ULMA, (2018). *Packaging Machines*. ULMA. [online]. Available at: http://www.ulmapackaging.com/packaging-machines [Accessed 2 Jan. 2018]

ULMA Vertical, (2018). *Packaging Machines*. ULMA. [online]. Available at: http://www.ulmapackaging.com/packaging-machines/vertical-vffs [Accessed 2 Jan. 2018]

ULMA Horizontal, (2018). *Packaging Machines*. ULMA. [online]. Available at: http://www.ulmapackaging.com/packaging-machines/flow-pack-hffs [Accessed 8 Nov. 2017]

Vyatkin, V. (2013). *Software Engineering in Industrial Automation: State-of-the-Art Review,* in: IEEEE Transactions on Industrial Informatics, Volume: 9, Issue: 3, Aug. 2013. DOI: 10.1109/TII.2013.2258165

Wang, S. et al, (2016). *Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination*, in: Computer Networks Volume 101, 4 June 2016, Pages 158-168. DOI: 10.1016/j.comnet.2015.12.017

Weilkiens, T. (2007). S*ystem Engineering with SysML/UML: Modeling, Analysis, Design. Elsevier*. MK/OMG Press. [eBook]. ISBN 978-0-12-374274-2

Weyer, S., Meyer, T., Ohmer, M., Gorecky, D., and Zühlke, D. (2016). *Future Modeling and Simulation of CPS-based Factories: An Example from the Automotive Industry*, in: IFAC-PapersOnLine Volume 49, Issue 31, 2016, Pages 97-102. DOI: 10.1016/j.ifacol.2016.12.168

List of Appendixes

Appendix A. Component Class – Robot

Robot – Class

Description

Robot is a machine that is able to perform repetitive tasks automatically, such as material flow handling, grasp/release part(s) and mount/unmount tool. Robot can wield tools, it can be mounted to a robot positioner, and it can work with a workpiece positioner. Robot positioner and workpiece positioner are connected to a robot as an external joint to the kinematic chain of the robot. Robot is also able to transmit and receive signals to interact with external devices, such as doors of the machine.

Functional Details

- Robot is programmed with Program Editor, Python Script or both
- Robot kinematics is defined by joint definitions and link offsets. Robot calculates forward and inverse kinematic solutions for the kinematic chain of the robot.
- Robot I/Os are Boolean signal map signals for inputs and outputs. 1-81 of output signals are mapped to grasp/release, tracing, mount and dismount of tool component actions

Compatibility

- Non-abstract interface to robot tool
- Non-abstract interface to robot positioner
- Abstract interface to workpiece positioner
- Non-abstract interface to accessories

Model Details

Properties

Property	Type & Unit	Description
J1, ..., J"n"	Joint (mm/degree)	Robot joint values
SignalActions::Configure	Button	Created by ActionScript
SignalActions::UpdateWorldIOnRelease	Boolean	Created by ActionScript
SignalActions::DisplayMessages	Boolean	Created by ActionScript
SignalActions::ReleaseToWorld	Boolean	Created by ActionScript
SignalActions::ReleaseToPhysics	String	Created by ActionScript
SignalActions::MultiGrasp	Boolean	Created by ActionScript
SignalActions::TraceWidth	Integer	Created by ActionScript
SignalActions::TraceZOffset	Real	Created by ActionScript

NOTE: ActionScript is Python Script Behavior. Script generates required Properties and Action Container Behavior to enable the use of actions.

Node Structure

Robot joints form a hierarchical node structure with MountPlate as the last node of the hierarchy chain. Robot can also have parallel kinematic structures (e.g. delta robot). Robot can also have assistive link structure.

Example:

Root Node (Base)

L1 (Hierarchical Node under Root Node)

...

L"n'

MountPlate (Flange Node. Tool is attached to this node)

Behaviors

Name	Type	Group	Node	Details	Description
Inputs	Boolean Signal Map	Signal	Root Node	Listeners: Executor	N/A
Outputs	Robot-Specific	Signal	Root Node	Listeners: Executor	Outputs: 1-16 Grasp/Release 17-32 Trace Tool On/Off 33-48 Mount/Unmount Tool 49-80 Trace External Tool On/Off 81 Start/Stop Swept Volume
Kinematics	Robot Controller	Kinematics	Root Node	N/A	Kinematics calculates values for kinematic chain
Controller	Robot Controller	Robotics	Root Node	Joints: J1, ..., J"n" Kinematics: Kinematics	Controller calculates kinematic chain values for a robot
Executor	Executor	Robotics	Root Node	isEnabled: True Controller: Controller	Executor allows you to read, write and execute a robot program
ActionScript	Python Script	Misc	Root Node	N/A	ActionScript generates required Properties and Action Container Behavior to enable the use of actions
Actions	Action Container	Misc	Root Node	Connections: None	An Action Container allows the sending, receiving and storing of actions
Statistics	Statistics	Misc	Root Node	N/A	A statistics collects and report statistics
GraspContainer	Container	Material Flow	Flange Node	TransitionSignal: Null Location: Null	When part(s) are grasped by a robot, they are contained here
JogInfo	Jog Info	Misc	Nodes in robot kinematic chain	N/A	Joints in the kinematic chain of the robot can be interactively moved by mouse drag

Optional

Name	Type	Group	Node	Details	Description
PythonScript	Python Script	Misc	Root Node	N/A	Can be used for value updating etc.

NOTE

Character "*" indicates that the current name should not be modified

Interfaces

Name	Type	Connection	Section	Field		Node	Description
Tool	One to One	Non-Abstract	Hierachy_Tools_Joints	Type: Hierarchy Name: Hierarchy	Node: mountplate Frame: FlangeFrame Parent: True	End Node	Interface to Robot Tools
				Type: Tools Name: Tools	ToolList: Controller Export: True		
				Type: Joints Name: Joints	Controller: Controller Export: False		

			Hierarchy_Tools	Type: Hierarchy Name: Hierarchy	Node: mountplate Frame: FlangeFrame Parent: True		
				Type: Tools Name: Tools	ToolList: Controller Export: True		
			Hierarchy_Joints	Type: Hierarchy Name: Hierarchy	Node: mountplate Frame: FlangeFrame Parent: True		
				Type: Joints Name: Joints	Controller: Controller Export: False		
			Hierarchy	Type: Hierarchy Name: Hierarchy	Node: mountplate Frame: FlangeFrame Parent: True		
RobotPositioner	One to One	Non-Abstract	Hierachy_Joints_Bases	Type: Hierarchy Name: Hierarchy	Node: GenericRobot Frame: RootFrame Parent: False	Root Node	Interface to Robot Positioner
				Type: Joints Name: Joints	Controller: Controller Export: False		
				Type: Bases Name: Bases	BaseList: Controller Export: False		
			Hierarchy_Joints	Type: Hierarchy Name: Hierarchy	Node: GenericRobot Frame: RootFrame Parent: False		
				Type: Joints Name: Joints	Controller: Controller Export: False		
			Hierarchy	Type: Hierarchy Name: Hierarchy	Node: GenericRobot Frame: RootFrame Parent: False		
WorkpiecePositionerJoints	One to Many	Abstract	Joints	Type: Joints Name: Joints	Controller: Controller Export: False	Root Node	Interface to Robot Workpiece Positioner
Accessories	One to Many	Abstract	Accessories Section Frame: Root Frame	N/A	N/A	Root Node	Interface to Robot Accessories

Features

Name	Type	Node	Description
RootFrame	Frame	Root Node	Root location
(user-defined)	Geometry	(user-defined)	Base and link geometries

Appendix B. Component Class – End Effector

End Effector (I/O control) – Class

Description

End effector is a tool for a robot and it is connected to the tool interface of a robot.

Functional Details

- End effector is attached to robot as child component
- End effector can be controlled with I/O signals
- Tool frames of the end effector are exported to a robot (optional)

Compatibility

- Connected to the tool interface of a robot

Model Details

Properties

Property	Type & Unit	Description
None	N/A	N/A

Node Structure

Node structure is user-defined

Behaviors

Controller is required when end effector has joint motion

Name	Type	Group	Node	Details	Description
Controller	Servo Controller	Robotics	Root Node	Joints assigned to controller are user-defined	Controller controls joint values

Interfaces

Without Tool Export

MountInterface	One to One	Non-Abstract	Name: Hierarchy	Type: Hierarchy Name: Hierarchy	Frame: MountFrame Parent: False	Root Node	Hierarchy interface to a robot

With Tool Export

MountInterface	One to One	Non-Abstract	Name: Hierarchy	Type: Hierarchy Name: Hierarchy	Frame: MountFrame Parent: False	Root Node	Hierarchy interface to a robot
			Name: Tool	Type: ToolExport Name: ToolExport	Export: True		Tool export to a robot

Features

Name	Type	Node	Description
RootFrame	Frame	Root Node	Root location
(user-defined)	Geometry	(user-defined)	Base and link geometries

End Effector (joint control) – Class

Description

End effector is a tool for a robot and it is connected to the tool interface of a robot.

Functional Details

- End effector is attached to robot as child component
- Joints assigned to servo controller are integrated to the kinematic chain of a robot as external joints
- Tool frames of the end effector are exported to a robot (optional)

Compatibility

- Connected to the tool interface of a robot

Model Details

Properties

Property	Type & Unit	Description
JointValue	Real (mm/degree)	Expresses joint value

Node Structure

Node structure is user-defined

Behaviors

Name	Type	Group	Node	Details	Description
Controller	Servo Controller	Robotics	Root Node	Joints assigned to controller are user-defined	Controller controls joint values

Interfaces

Without Tool Export

MountInterface	One to One	Non-Abstract	Name: Hierarchy	Type: Hierarchy Name: Hierarchy	Frame: MountFrame Parent: False	Root Node	Hierarchy interface to a robot
			Name: JointExport	Type: JointExport Name: JointExport	Controller: Controller Export: True		Joint export to a robot

With Tool Export

MountInterface	One to One	Non-Abstract	Name: Hierarchy	Type: Hierarchy Name: Hierarchy	Frame: MountFrame Parent: False	Root Node	Hierarchy interface to a robot
			Name: JointExport	Type: JointExport Name: JointExport	Controller: Controller Export: True		Joint export to a robot
			Name: Tool	Type: ToolExport Name: ToolExport	Export: True		Tool export to a robot

Features

Name	Type	Node	Description
RootFrame	Frame	Root Node	Root location
(user-defined)	Geometry	(user-defined)	Base and link geometries

Appendix C. Component Class – Stand-Alone Machine

Stand-Alone Machine – Class

Description

Stand-Alone Machine receives part(s) from resource (e.g. robot) and processes it. Then, part(s) are transferred from the machine by a resource

Functional Details

- Machine requests part(s) with "TransportInReq" Boolean signal to a resource manager that responds with "TransportInDone" Boolean signal when the part(s) are transmitted
- Machine processes part(s) when they are at process locations
- Machine requests part(s) to be taken with "TransportOutReq" Boolean signal to a resource manager that responds with "TransportOutDone" Boolean signal when the part(s) are taken

Compatibility

- The component has abstract interface to resource manager

Model Details

Properties

Property	Type & Unit	Description
TransportType	String	Defines the material flow transport type
ProcessTime	Distribution (s)	Duration of the process
PartsInProcess	Integer	Number of parts in a process
ProcessID	String	Determines process flow
BrokenStatus	Boolean	Is the machine at broken state
MTBF	Distribution (s)	Mean time between failures in the machine
MTTR	Distribution (s)	Mean time to repair the machine
PartYield	Distribution	Probability of a failure in a part
Optional		
DoorOpenTime	Distribution (s)	Duration of door closing time
DoorCloseTime	Distribution (s)	Duration of door opening time

Node Structure

Node structure is device-specific

Behaviors

Name	Type	Group	Node	Details	Description
TransportInReq*	Boolean Signal (out)	Signal	Root Node	Connections: None	Request for part(s) to be transported to the machine
TransportInDone*	Boolean Signal (in)	Signal	Root Node	Connections: MachineScript	Part(s) are transported to the machine
TransportOutReq*	Boolean Signal (out)	Signal	Root Node	Connections: None	Request for part(s) to be transported out of the machine
TransportOutDone*	Boolean Signal (in)	Signal	Root Node	Connections: MachineScript	Part(s) are transported from the machine
TriggerSignal	Boolean Signal (in)	Signal	Root Node	Connections: MachineScript	Triggering signal
MachineScript	Python Script	Misc	Root Node	N/A	MachineScript contains the logics of the machine
Process1Container*	Container	Material Flow	Root Node	Location: Process1Location TransitionSignal: Sensor	Contains part(s) during the process

Statistics	Statistics	Misc	Root Node	N/A	A statistics enables to collect and report statistics

Optional

Name	Type	Group	Node	Details	Description
OpenDoors*	Boolean Signal (in)	Signal	Root Node	Connections: MachineScript	Doors are opened with True value and closed with False value
DoorIsOpen*	Boolean Signal (out)	Signal	Root Node	Connections: None	Signal is True when doors are open
DoorIsClosed*	Boolean Signal (out)	Signal	Root Node	Connections: None	Signal is True when doors are closed
ReservedBy*	String Signal (in)	Signal	Root Node	Connections: MachineScript	Component can be reserved for transport operations. This signal can be created by a resource manager
DoorController	Servo Controller	Robotics	Root Node	Joints: user-defined	Moves selected joints according to ServoMotionData
DoorMotionData*	Note	Misc	Root Node	N/A	Contains motion data for servo movements
ServoController	Servo Controller	Robotics	Root Node	Joints: user-defined	Moves selected joints according to ServoMotionData
ServoMotionData*	Note	Misc	Root Node	N/A	Contains motion data for servo movements

NOTE

Character "*" indicates that the current name should not be modified

Interfaces

Name	Type	Connection	Section	Field	Description	Node	Description
ResourceIface	One to One	Abstract	Name: Section	Type: Signal Name: Trigger	Signal: TriggerSignal Connection: Normal	Root Node	Interface to Robot
				Type: IntegerCompatibility Name: Compatibility	Value: 5001		

Features

Name	Type	Node	Description
Process1Location*	Frame	Root Node	Location of the process
(user-defined)	Geometry	(user-defined)	Base and link geometries

NOTE

Character "*" indicates that the current name should not be modified

Appendix D. Component Class – Start-of-Line Machine

Start-of-Line Machine – Class

Description

Start-of-Line Machine receives part(s) from resource (e.g. robot). Part(s) can be processed by the machine. Then, part(s) are transferred to a line, starting from the machine.

Functional Details

- Machine requests part(s) with "TransportInReq" Boolean signal to a resource manager that responds with "TransportInDone" Boolean signal when the part(s) are transmitted
- Machine can process part(s) when they are at process locations. Process is not required, if the machine is used only for material flow tasks
- Part(s) are transferred from the line output of the machine to the next line component

Compatibility

- The component has abstract interface to resource manager
- The component has non-abstract interface to a line component (e.g. conveyor)

Model Details

Properties

Property	Type & Unit	Description
TransportType	String	Defines the material flow transport type
ProcessTime	Distribution (s)	Duration of the process
PartsInProcess	Integer	Number of parts in a process
ProcessID	String	Determines process flow
BrokenStatus	Boolean	Is the machine at broken state
MTBF	Distribution (s)	Mean time between failures in the machine
MTTR	Distribution (s)	Mean time to repair the machine
PartYield	Distribution	Probability of a failure in a part
Optional		
DoorOpenTime	Distribution (s)	Duration of door closing time
DoorCloseTime	Distribution (s)	Duration of door opening time

Node Structure

Node structure is device-specific

Behaviors

Name	Type	Group	Node	Details	Description
TransportInReq*	Boolean Signal (out)	Signal	Root Node	Connections: None	Request for part(s) to be transported to the machine
TransportInDone*	Boolean Signal (in)	Signal	Root Node	Connections: MachineScript	Part(s) are transported to the machine
TriggerSignal	Boolean Signal (in)	Signal	Root Node	Connections: MachineScript	Triggering signal
SensorSignal	Boolean Signal	Signal	Root Node	Connections: MachineScript	Indicates when part is in the Process1Container
MachineScript	Python Script	Misc	Root Node	N/A	MachineScript contains the logics of the machine
Path	One Way Path	Material Flow	Root Node	Path: InFrame, Process1Location, OutFrame	Transfer parts(s) from the line input to the process location and to the line output

PathSensor	Sensors	Sensors	Root Node	Frame: Process1Location BoolSignal: SensorSignal BoolSignalValue: True	Indicates when part(s) are in the Process1Container
Process1Container*	Container	Material Flow	Root Node	Location: Process1Location TransitionSignal: Sensor	Contains part(s) during the process
Statistics	Statistics	Misc	Root Node	N/A	A statistics enables to collect and report statistics

Optional

Name	Type	Group	Node	Details	Description
OpenDoors*	Boolean Signal (in)	Signal	Root Node	Connections: MachineScript	Doors are opened with True value and closed with False value
DoorIsOpen*	Boolean Signal (out)	Signal	Root Node	Connections: None	Signal is True when doors are open
DoorIsClosed*	Boolean Signal (out)	Signal	Root Node	Connections: None	Signal is True when doors are closed
ReservedBy*	String Signal (in)	Signal	Root Node	Connections: MachineScript	Component can be reserved for transport operations. This signal can be created by a resource manager
DoorController	Servo Controller	Robotics	Root Node	Joints: user-defined	Moves selected joints according to ServoMotionData
DoorMotionData*	Note	Misc	Root Node	N/A	Contains motion data for servo movements
ServoController	Servo Controller	Robotics	Root Node	Joints: user-defined	Moves selected joints according to ServoMotionData
ServoMotionData*	Note	Misc	Root Node	N/A	Contains motion data for servo movements

NOTE

Character "*" indicates that the current name should not be modified

Interfaces

Name	Type	Connection	Section	Field	Description	Node	Description
ResourceIface	One to One	Abstract	Name: Section	Type: Signal Name: Trigger	Signal: TriggerSignal Connection: Normal	Root Node	Interface to Robot
				Type: IntegerCompatibility Name: Compatibility	Value: 5001		
OutIface	One to One	Non-Abstract	Name: OutSection	Type: Flow Name: Out	Container: PathOut PortName: Output	Root Node	Interface to Robot

Features

Name	Type	Node	Description
OutFrame*	Frame	Root Node	Location of line output
Process1Location*	Frame	Root Node	Location of the process
(user-defined)	Geometry	(user-defined)	Base and link geometries

NOTE

Character "*" indicates that the current name should not be modified

Appendix E. Component Class – End-of-Line Machine

End-of-Line Machine – Class

Description

End-of-Line Machine receives part(s) from the line input and processes it. Then, part(s) are transferred from the machine by a resource (e.g. robot).

Functional Details

- Part(s) are transferred from the previous line components to the line input of the machine
- Machine can process part(s) when they are at process locations. Process is not required, if the machine is used only for material flow tasks
- Machine requests part(s) to be taken with "TransportOutReq" Boolean signal to a resource manager that responds with "TransportOutDone" Boolean signal when the part(s) are taken
- "TransportOutDone" Boolean signal when the part(s) are taken

Compatibility

- The component has abstract interface to resource manager
- The component has on-abstract interface to line input component (e.g. conveyor)

Model Details

Properties

Property	Type & Unit	Description
TransportType	String	Defines the material flow transport type
ProcessTime	Distribution (s)	Duration of the process
PartsInProcess	Integer	Number of parts in a process
ProcessID	String	Determines process flow
BrokenStatus	Boolean	Is the machine at broken state
MTBF	Distribution (s)	Mean time between failures in the machine
MTTR	Distribution (s)	Mean time to repair the machine
PartYield	Distribution	Probability of a failure in a part
Optional		
DoorOpenTime	Distribution (s)	Duration of door closing time
DoorCloseTime	Distribution (s)	Duration of door opening time

Node Structure

Node structure is device-specific

Behaviors

Name	Type	Group	Node	Details	Description
TransportOutReq*	Boolean Signal (out)	Signal	Root Node	Connections: None	Request for part(s) to be transported out of the machine
TransportOutDone*	Boolean Signal (in)	Signal	Root Node	Connections: MachineScript	Part(s) are transported from the machine
TriggerSignal	Boolean Signal (in)	Signal	Root Node	Connections: MachineScript	Triggering signal
SensorSignal	Boolean Signal	Signal	Root Node	Connections: MachineScript	Indicates when part is in the Process1Container
MachineScript	Python Script	Misc	Root Node	N/A	MachineScript contains the logics of the machine
Path	One Way Path	Material Flow	Root Node	Path: InFrame, Process1Location, OutFrame	Transfer parts(s) from the line input to the process location and to the line output

PathSensor	Sensors	Sensors	Root Node	Frame: Process1Location BoolSignal: SensorSignal BoolSignalValue: True	Indicates when part(s) are in the Process1Container
Process1Container*	Container	Material Flow	Root Node	Location: Process1Location TransitionSignal: Sensor	Contains part(s) during the process
Statistics	Statistics	Misc	Root Node	N/A	A statistics enables to collect and report statistics

Optional

Name	Type	Group	Node	Details	Description
OpenDoors*	Boolean Signal (in)	Signal	Root Node	Connections: MachineScript	Doors are opened with True value and closed with False value
DoorIsOpen*	Boolean Signal (out)	Signal	Root Node	Connections: None	Signal is True when doors are open
DoorIsClosed*	Boolean Signal (out)	Signal	Root Node	Connections: None	Signal is True when doors are closed
ReservedBy*	String Signal (in)	Signal	Root Node	Connections: MachineScript	Component can be reserved for transport operations. This signal can be created by a resource manager
DoorController	Servo Controller	Robotics	Root Node	Joints: user-defined	Moves selected joints according to ServoMotionData
DoorMotionData*	Note	Misc	Root Node	N/A	Contains motion data for servo movements
ServoController	Servo Controller	Robotics	Root Node	Joints: user-defined	Moves selected joints according to ServoMotionData
ServoMotionData*	Note	Misc	Root Node	N/A	Contains motion data for servo movements

NOTE

Character "*" indicates that the current name should not be modified

Interfaces

Name	Type	Connection	Section	Field	Description	Node	Description
ResourceIface	One to One	Abstract	Name: Section	Type: Signal Name: Trigger	Signal: TriggerSignal Connection: Normal	Root Node	Interface to Robot
				Type: IntegerCompatibility Name: Compatibility	Value: 5001		
InIface	One to One	Non-Abstract	Name: InSection	Type: Flow Name: In	Container: PathIn PortName: Input	Root Node	Interface to Robot

Features

Name	Type	Node	Description
InFrame*	Frame	Root Node	Location of line input
Process1Location*	Frame	Root Node	Location of the process
(user-defined)	Geometry	(user-defined)	Base and link geometries

NOTE

Character "*" indicates that the current name should not be modified

Appendix F. Component Class – In-Line Machine

In-Line Machine – Class

Description

In-Line Machine receives part(s) from the line input and processes it. Then, part(s) are transferred to the line output.

Functional Details

- Part(s) are transferred from the previous line components to the line input of the machine
- Machine processes part(s) when they are at process locations
- Part(s) are transferred from the line output of the machine to the next line component

Compatibility

- The component has non-abstract input interface to a line component (e.g. conveyor)
- The component has non-abstract output interface to a line component (e.g. conveyor)

Model Details

Properties

Property	Type & Unit	Description
ProcessTime	Distribution (s)	Duration of the process
PartsInProcess	Integer	Number of parts in a process
ProcessID	String	Determines process flow
BrokenStatus	Boolean	Is the machine at broken state
MTBF	Distribution (s)	Mean time between failures in the machine
MTTR	Distribution (s)	Mean time to repair the machine
PartYield	Distribution	Probability of a failure in a part

Node Structure

Node structure is device-specific

Behaviors

Name	Type	Group	Node	Details	Description
SensorSignal	Boolean Signal	Signal	Root Node	Connections: MachineScript	Indicates when part is in the Process1Container
SensorComp	Component Signal	Signal	Root Node	Connections: MachineScript	Contains the part, that triggered the sensor
MachineScript	Python Script	Misc	Root Node	N/A	MachineScript contains the logics of the machine
Path	One Way Path	Material Flow	Root Node	Path: InFrame, Process1Location, OutFrame	Transfer parts(s) from the line input to the process location and to the line output
PathSensor	Sensors	Sensors	Root Node	Frame: Process1Location BoolSignal: SensorSignal BoolSignalValue: True	Indicates when part(s) are in the Process1Container
Statistics	Statistics	Misc	Root Node	N/A	A statistics enables to collect and report statistics

Optional

Name	Type	Group	Node	Details	Description
ServoController	Servo Controller	Robotics	Root Node	Joints: user-defined	Moves selected joints according to ServoMotionData
ServoMotionData*	Note	Misc	Root Node	N/A	Contains motion data for servo movements

NOTE

Character "*" indicates that the current name should not be modified

Interfaces

Name	Type	Connection	Section	Field	Description	Node	Description
InIface	One to One	Non-Abstract	Name: InSection	Type: Flow Name: In	Container: PathIn PortName: Input	Root Node	Interface to Robot
OutIface	One to One	Non-Abstract	Name: OutSection	Type: Flow Name: Out	Container: PathOut PortName: Output	Root Node	Interface to Robot

Features

Name	**Type**	**Node**	**Description**
InFrame*	Frame	Root Node	Location of line input
OutFrame*	Frame	Root Node	Location of line output
Process1Location*	Frame	Root Node	Location of the process
(user-defined)	Geometry	(user-defined)	Base and link geometries

NOTE

Character "*" indicates that the current name should not be modified

Appendix G. Best Practices Interview For – Collected Results

This Appendix includes collected results from best practices interview forms. Five engineers responded to the interview form and the engineers are named as Eng. A, B, C, D and E. Some of the answers are rephrased to make the answers shorter or more understandable.

Best Practices Interview Form – Collected Results

Modeling - General

- **Features** (e.g. use of Frames, Geometries, Transforms, Clones, place of origin …)

1. What practices should be avoided?

- Use primitive geometry when possible. It is lighter and prevents VCID errors. [Eng. A]
- Editing an operation feature before applying it. [Eng. B]
- Using default names. [Eng. B]
- Cloning on the fly during simulation. [Eng. B]

2. What practices should be used?

- Naming of primitives and operators to help future authors. [Eng. A]
- Organization of feature tree. [Eng. A]
- Good Origins. Avoid Rot/Trans outside of Transform operator (confusing as to where transformation comes from). [Eng. A]
- Pick a good origin early and work from it. [Eng. A & D]
- Use template to build robots. [Eng. B]
- Create note behavior to contain data sheet, important measurements, and links to documentation and other important files. [Eng. B]
- If possible, clean up geometry in native CAD editor before import. Then simplify and remove geometry that has no purpose. [Eng. B]
- Simple and clear expressions: prefer changing parent location, use properties as dimension parameters. [Eng. D]

- **Behaviors** (e.g. use of Signals, Paths, Containers …)

3. What practices should be avoided?

- Using default names. [Eng. B]
- Routing rules 5 or more levels deep. [Eng. B]
- A component creator that does output to a container. [Eng. B]

4. What practices should be used?

- Descriptive naming. [Eng. A, D]
- Clear description of interface fields. [Eng. B]
- Note behavior to describe signal processes. [Eng. B]

- **Properties** (e.g. String, Boolean, Integer, Real …)

5. What practices should be avoided?

- Leaving unused behaviors. [Eng. A] [Eng. B]
- Using default names. [Eng. A]

6. What practices should be used?

- Hide where needed. [Eng. A]
- Putting behaviors in root node. [Eng. A]
- Good naming. [Eng. A & B]
- Logical grouping. [Eng. A]
- Use of tabs and button. [Eng. A]
- Always assign quantity, if possible [Eng. B]

Naming Conventions (e.g. name of Features/Behaviors/Properties/Script variables/Parameter names…)

7. What practices should be used?

- No underscores for spaces. [Eng. A]
- Descriptive naming. [Eng. A & C & D]
- Check spelling. [Eng. A]
- AmountOf vs NumberOf. [Eng. A]
- Consistency. [Eng. A & B]
- Diameter not diagonal. [Eng. A]

Python Scripting (e.g. structure of the script, loops, variables, scripting methods …)

8. What practices should be avoided?

- Complexity and obscure programming structures. [Eng. A]
- No commenting. [Eng. B]
- Lack of whitespace. [Eng. B]
- Nested loops. [Eng. B]
- The use of delays [Eng. D]

9. What practices should be used?

- Keep it simple. [Eng. A]
- Clean-up dead code. [Eng. A]
- Write modular code [Eng. A & D]
- Maintainable. [Eng. A]
- Only use if cannot be done with Expressions (Avoid on Event). [Eng. A]
- Put GUI control in separate script from simulation code and give proper name to it. [Eng. A]
- Clearly sectioned code. [Eng. B]
- Comment when profitable. [Eng. D]

Robustness, Reusability and Error Handling of the Model

10. What reduces the robustness of the model?

- Complexity and scripting. [Eng. A]
- The use of delays [Eng. C]

11. What improves the robustness of the model?

- Simplicity, elegance of design. [Eng. A]

12. What reduces the reusability of the model?

- Complexity. [Eng. A]
- Deprecated attributes, such as behaviors and outdated scripts. [Eng. B]

13. What improves the reusability of the model?

- Simplicity. [Eng. A]
- Modular design when appropriate. E.g. machine built with multiple components. [Eng. B]
- Contains the latest, most stable parts, e.g. ActionScript. [Eng. B]
- Appropriate level of quality of the geometry for intended use-case. [Eng. B]
- Take model extension into account in the design phase [Eng. C]

14. What practices should be used in error handling?

- Print error messages. [Eng. A]
- Always check output panel. [Eng. B]
- Print component name in error messages. Print also other information, such as node name when relevant. [Eng. D]

Performance of the Model (e.g. CAD model import characteristics, Python loops, Behavior types, communication …)

15. What practices should be avoided?

- Avoid delays, and use conditions when needed. [Eng. A]
- Avoid triggerCondition. Use condition instead. [Eng. A]
- ComponentFlowProxie, RoutingRule, TwoWayPaths. [Eng. A]
- Avoid unnecessary geometries. [Eng. B]
- Exclude unnecessary details [Eng. C]
- Avoid unnecessary rebuilding in properties when it is not required. [Eng. D]
- Avoid generating unnecessary property / signal change events. [Eng. E]

16. What practices should be used?

- Scripting only where needed. [Eng. A]

Component Connectivity

17. What practices should be avoided?

- Generic interfaces. [Eng. A]
- Avoid relying on delays, use sensors or sensor-like logic instead to know when action has completed. [Eng. E]
- Don’t mix input and output. One signal should be either input or output. [Eng. E]

18. What practices should be used?

- Good naming to interfaces. [Eng. A]
- Integer compatibility in interfaces. [Eng. A]
- Begin/End in signals to reduce potential partners. [Eng. A]
- Check if the PLC logics works before implementation. [Eng. C]
- Try to think what inputs and outputs the real machine would need to have and expose those as signals for external control. [Eng. E]
- Material handling components should also work with physics for emulation purposes. [Eng. E]
- Use simple data types for IO signals and properties. [Eng. E]
- Include state signals that can be used to verify and time actions. [Eng. E]
- Separate IO variables from the rest with naming conventions (when suitable). [Eng. E]
- Treat all IOs as asynchronous. [Eng. E]

<u>Connectivity definition:</u>

Connectivity and communication with external systems / controllers (e.g. PLC)

Appendix H. Best Practices

1. Modeling – General

1.1 Features

Best Practices

1.1.1 Component origin

Selecting the origin for the component should be the starting point of modeling process. When components are imported to the simulation model, the orientation and the location on the z-axis is defined by the origin. Good component origin is component-specific and for this reason origins should be individually determined.

When determining the origin, consider following, if no reason to do otherwise:

- Origin should be on the ground level of the component
- The component should be easily rotated around the origin
- Choose location of the origin based on the component. In conveyors, the origin should be at the bottom-middle of the start of the conveyor. In machines, the origin should be at the center bottom of the machine.
- Choose orientation of the origin based on the component. In conveyors, the x-axis should have the same direction than the length of the conveyor. In machines, the y-axis should have the same direction than the approach direction to the machine.
- Use symmetry when possible. Symmetry enables origin to stay on appropriate position, even if the geometry changes. Also, symmetry enables easier clone geometry operations.

1.1.2 Rename features

Created features should be descriptively named according to the purpose of the feature. Naming practices are further discussed in the section 2. Naming Conventions.

1.1.3 Primitive geometries

Pre-defined primitive geometry features should be used when suitable. They provide parametric lightweight performance models and prevents VCID errors.

1.1.4 Property values as parameters

In parametric models, property values should be used as feature parameters. This enables end-user to easily configure the model without the need of further examination of the feature tree.

1.1.5 Transform – simple expressions

Expression of the transform feature should be simple and understandable. Expressions describes the translation and rotation of the features under the transform feature. One solution is to use feature properties with parent coordinates to modify position and orientation. Other solution is to separate complex transforms to multiple and more simple transform features. For example, one transform with x-, y-, z-directional translation operations, should be separated to three transform features with each of them performing one translation operation.

1.1.6 Locate frames and geometries under separated transforms

To improve simplicity and easier change of geometry, frames and geometries should be located under separate transform features.

1.1.7 Adding label

If a label is required to the component, it should be added as a decal.

1.1.8 Remove unused features

Unused features should be removed to improve simplicity.

1.2 Behaviors

Best Practices

1.2.1 Behaviors in root node

Behaviors should be located under the root node, if no reason to do otherwise.

1.2.2 Rename behaviors

Created behaviors should be descriptively named according to the purpose of the behavior. Naming practices are further discussed in section 2. Naming Conventions.

1.2.3 Remove unused behaviors

Unused behaviors should be removed to improve simplicity.

1.2.4 Avoid complex interfaces

Interfaces between components should include only necessary information

1.2.5 Interface connection only between matching components

Interface between components should be possible only with matching components. This can be achieved with the use of compatibility integer field in the Interface sections. Compatibility integer requires the same integer value in both sides of the Interface to enable the connection.

Development proposal: Compatibility String

Compatibility string could provide more practical method to limit and control interface connections. String provides more descriptive naming instead of an integer.

1.2.6 Avoid complex routing rules

Complex routing rules should be avoided. Python script should be considered instead of complex routine rule logics.

1.2.7 Note to describe signals and interfaces

Examination of signals and interfaces is effective method to understand the logics of the component. However, in some cases the purpose and logic of signals and interfaces is not sufficiently self-explaining, which leads to the examination of the script. In these cases, note behavior should include description of signals and interfaces.

Development proposal: Description Field

Most suitable location for descriptions is the behavior itself. Signal and interface behaviors could include description field, where the description could be added.

1.2.8 Note to store component related data and links

Important information related to the component can be stored in a note behavior. This information can include links to reference material, dimension values and other relevant information. When including links or other time sensitive material, accessed date should be marked.

1.3 Properties

Best Practices

1.3.1 Group and hide properties

Grouping and hiding is effective method to improve the simplicity of the properties when the amount of properties increases. Grouping should be performed to separate properties to specific groups by their purpose. For example, properties defining the dimensions could form one group, and properties defining process parameters could form second group. Properties that are irrelevant for the actual use of the component, should be hided.

1.3.2 Rename properties

Created properties should be descriptively named, according to the purpose of the property. Naming practices are further discussed in section “2. Naming Conventions”.

1.3.3 Remove unused properties

Unused properties should be removed to improve simplicity.

1.3.4 Order of properties

The order of some specific set of properties should be standardized to improve readability and consistency. For example, properties including x, y, z or length, width, height.

1.3.5 Use appropriate property types

Appropriate property types should be used. For example, real property should not be used as Integer.

1.3.6 Assign units

Property should have unit assigned when possible

2. Naming Conventions

Good naming conventions should not be overlooked in modeling process. Naming conventions plays an important role at expressing how the model functions.

Best Practices

2.1 Descriptive and extensible naming

Naming should describe the purpose and function of the named attribute. Ideal goal of descriptive naming is to achieve self-explaining names, without the need of additional descriptions. For example, ConveyorLength*, ProcessTime*.

Extensible naming enables the extension of the current naming practice. Extensibility can be achieved, for example, with indexing number after the name. For example, *ProcessPoint1, ProcessPoint2*.

2.2 Abbreviations

Abbreviations should be avoided when possible. However, sometimes the use of abbreviations is profitable, especially if it enables to significantly reduce the length of the naming. Abbreviations should be consistent to avoid conflicts between similar abbreviation terms.

Development proposal: List of standardized terms and abbreviations

The list of standardized terms and abbreviations would act as a link between terms and abbreviations.

2.3 Capital letters instead of underscores

Terms should be separated with capital letters instead of underscores.

2.4 Consistency

Same naming convention in different use cases should always have the same purpose.

2.5 Check spelling

Spellings should be checked for errors.

2.6 Signal naming

Signals are typically separated to control and state signals. Control signals should be named describing the action, and state signals should be named describing the state. For example, Boolean signal controlling the door could be names as *OpenDoor* and *CloseDoor*, as well as Boolean signals indicating the status of the door could be named as *DoorIsOpen* and *DoorIsClosed*. Status signals could also be named based on the sensors, for example, *SensorLineIn* and *SensorLineOut*.

In addition, sometimes triggering signal is required in a component to request execution of the component. For example, component signal functioning as a trigger, should be named as *TriggerSignal*.

3. Python Scripting

Scripting is an effective tool, but with incoherent and messy coding practices, it can form a significant threshold in the adoption of the model. In addition, end-users may not be very familiar with programming. For these reasons script should be kept as simple and readable as possible.

Best Practices

3.1 Keep it simple

Python script should only have necessary amount of code. Complicated structures, such as nested loops should be avoided when possible.

3.2 Descriptive code

Descriptive naming of variables and functions should ideally self-explain how the script functions. In addition, descriptive commentary is recommended. Commenting is used to describe the script operations in detailed and abstract level when profitable.

3.3 Clearly sectioned code

Script should be clearly sectioned to improve readability. Most important event handlers, such as *OnRun* and *OnSignal* should be near the top of the script. Functions should be located in one place, after the most important event handlers. Variables should be initialized at the bottom of the script. Global declarations should be done at the beginning of the functions.

3.4 Modular code

Modular coding practices, such as modular functions should be used when possible. However, case-specific naming practices should still be applied.

3.5 Remove unused code

Unused code, such as variables and functions should be removed to improve simplicity.

3.6 Avoid unnecessary delays

Unnecessary delays should be avoided. These delays can, for example, distort cycle times.

3.7 Snippets for repetitive code

Snippets are pre-defined sets of code that can be generated instantly in the script. Snippets could be used to create repetitive sets of code, such as variable initialization and functions.

3.8 Event-based logics – *condition(), triggerCondition() and OnSignal()*

In event-based logics, events are waited in the script to trigger execution of the script. There are two methods to wait events, *condition()* and *triggerCondition()*. They both require user-defined condition criteria to be true to continue execution of the script. The criteria in *condition()* is evaluated at the initial execution of the method, and every time trigger is received by the script. The criteria in *triggerCondition()* is evaluated only when the specific and pre-defined trigger is received. This method contains a risk that the trigger occurs before the *triggerCondition()* method is initially executed, which can lead to a deadlock. If there is no specific reason to apply *triggerCondition()* method, then *condition()* method should be used instead. In addition to these methods, there is *OnSignal()* function, that is triggered and executed every time trigger is received by the script.

3.9 Traceable and descriptive errors print

Error prints should be printed when the script is not working desirably. Prints are visible in the Output panel, and in large layouts, the origin of the print might be challenging to solve. Error prints should include the name of the component, as well as to be descriptive, to possibly avoid the need of actually opening and analyzing the script. Example of error print: “Error in component: X, part not found in node Y“.

3.10 Try - Except

Try and except can be used to avoid errors in the script. End-users should be cautious with the use of try and error, because it can hide the sources of errors.

4. Robustness, Reusability

Best Practices

4.1.1 Simple and coherent model

Simplicity and coherency plays major roles in the robustness and reusability of the model. When the model is simple and coherent, it is easier to adopt, extend and debug.

4.1.2 Avoid execution of logics inside dynamic components

Dynamic components should be handled as passive components without executing logics, such as Python script.

4.1.3 Design extensible model

Reusability of components can be improved with extensible and parametric design. Extensible design should include, for example, extensible naming practices. Parametric design should include, for example, comprehensive use of parameters in the model configuration.

4.1.4 Naming of joints

Name of the component joint should start with "*J*" or "*Joint*" characters. Main kinematic chain of the component should have either following naming practice *J1, J2, J3, …*, or node-based naming practice, such as *JointLeftLeg*, *JointCables*, … Other kinematic chains in the component should utilize node-based naming practice, if no reason to do otherwise.

5. Performance of the Model

Performance of the model focuses on the amount of computing resources required during the simulation.

Best Practices

5.1 Simplify geometry

Geometry is always compromise between performance and quality. General performance indicator of the geometry is polygon count, which is the number of triangles required to construct the geometry. This polygon count can be influenced with tessellation quality and with geometry filters. Tessellation quality defines the overall quality of the geometry. Geometry filters are used to remove unnecessarily detailed geometry based on the filtering criteria. Polygon count should be adjusted to visually sufficient level, but without consuming unnecessary amount of resources.

5.2 Avoid unnecessary polling and sampling

Polling is definition of reading values continuously with certain frequency. For example, script using polling method can consume significant amount of resources. Unnecessary polling should be avoided and event-based methods used instead.

Sampling is definition of performing some operation continuously with certain frequency. Behaviors, such as raycast and volume sensor operates using sampling. Functionalities in robot program, such as collision detection and limit checking operates with sampling. Physics models is based on sampling as well. Unnecessary sampling should be avoided to save computing resources.

5.3 Avoid unnecessary property rebuild

Properties have option to continuously rebuild the component. This is useful if property is used to define features or expressions in a node. Otherwise, it is unnecessary and should be avoided to consuming resources.

5.4 Physics modeling – Avoid unnecessary colliders

In physics modeling, colliders are created from component geometries. These colliders can react to physical objects that enables the interaction of forces. Unnecessary colliders that does not have purpose in the simulation should be avoided.

6. Component Connectivity

The purpose of component connectivity is to enable connection between simulation components and external systems.

Best Practices

6.1 Check if external control logics can function with the system

It is recommended to consider before implementation that is the current system capable to be externally controlled.

6.2 Example from real machines

Components should use sensors or sensor-like logic and unnecessary delays should be avoided. Also, minimal logic in external control mode should be used when possible.

6.3 Inputs and Outputs

Signals used as inputs and outputs in component connectivity should be one directional. This denotes that one signal should be either input or output. Signals are typically divided to control and state signals. Control signals are inputs signals to the component while the state signals are outputs. Separation of the signals should be made with naming practices (see Section 2.6 Signal naming).

In addition to separation of inputs and outputs, simple data types should be used for IO signals and properties. For example, types as Int/Bool/Real/String. In addition, all IOs should be treated as asynchronous. Nothing happens immediately and IO signal should not be relied to come in specific order.

Development Proposal: optional Begin/End signal attributes

Reliable method to ensure one directional use of signals would be Begin/End options in signal behavior. Therefore, signals with begin attribute could be only connected to signals with end attribute and the other way around.

Appendix I. Machine Scripts

Machine scripts are created by the wizard presented at section 5.3. The scripts are located in the Python Script behaviors in the generated machine components. These scripts provide the basis for the component logics, and they are further customized with modular code sections managed by the wizard.

StartOfLineMachine.py

```
from vcScript import *

def OnRun():
  global sensorSignal,processTime,sensorComp
  processTime = comp.getProperty('ProcessTime')

  while True:
    transportInReq.signal(True)
    triggerSignal.signal(comp)
    stats.State = 'Idle'
    triggerCondition(lambda: getTrigger() == transportInDone and transportInDone.Value == True)
    #condition(lambda: transportInDone.Value == True)
    transportInReq.signal(False)
    part = cont.Components[0]
    path.grab(part)
    #condition(lambda: sensorSignal.Value)
    condition(lambda: sensorComp.Value)
    # Stop part
    part = sensorComp.Value
    part.stopMovement()
    # Process starts
    stats.State = 'Busy'
    delay(processTime.Value)
    part.startMovement()
    # Request part

# INITIALIZATION
comp = getComponent()
stats = comp.findBehaviour('Statistics')
transportInReq = comp.findBehaviour('TransportInReq')
transportInDone = comp.findBehaviour('TransportInDone')
triggerSignal = comp.findBehaviour('TriggerSignal')
sensorSignal = comp.findBehaviour('SensorSignal')
sensorComp = comp.findBehaviour('SensorComp')
cont = comp.findBehaviour('Process1Container')
path = comp.findBehaviour('Path')
```

StandAloneMachine.py

```
from vcScript import *

def OnRun():
  global processTime
  processTime = comp.getProperty('ProcessTime')

  while True:
    # Next part to the machine
    transportInReq.signal(True)
    stats.State = 'Idle'
    triggerCondition(lambda: getTrigger() == transportInDone and transportInDone.Value == True)
    transportInReq.signal(False)
    stats.State = 'Busy'
    # Process starts
    delay(processTime.Value)
    transportOutReq.signal(True)
    triggerSignal.signal(comp)
    stats.State = 'Idle'
    triggerCondition(lambda: getTrigger() == transportOutDone and transportOutDone.Value == True)
    transportOutReq.signal(False)

# INITIALIZATION
comp = getComponent()
stats = comp.findBehaviour('Statistics')
transportInReq = comp.findBehaviour('TransportInReq')
transportInDone = comp.findBehaviour('TransportInDone')
transportOutReq = comp.findBehaviour('TransportOutReq')
transportOutDone = comp.findBehaviour('TransportOutDone')
triggerSignal = comp.findBehaviour('TriggerSignal')
```

EndOfLineMachine.py

```
from vcScript import *

def OnRun():
  global sensorSignal,processTime,sensorComp
  processTime = comp.getProperty('ProcessTime')

  while True:
    stats.State = 'Idle'
    triggerCondition(lambda: getTrigger() == sensorComp and sensorComp.Value)
    part = sensorComp.Value
    part.stopMovement()
    stats.State = 'Busy'
    # Process starts
    delay(processTime.Value)
    # Request part to be picked
    cont.grab(part)
    transportOutReq.signal(True)
    triggerSignal.signal(comp)
    triggerCondition(lambda: getTrigger() == transportOutDone and transportOutDone.Value == True)
    transportOutReq.signal(False)

# INITIALIZATION
comp = getComponent()
stats = comp.findBehaviour('Statistics')
transportOutReq = comp.findBehaviour('TransportOutReq')
transportOutDone = comp.findBehaviour('TransportOutDone')
triggerSignal = comp.findBehaviour('TriggerSignal')
sensorSignal = comp.findBehaviour('SensorSignal')
sensorComp = comp.findBehaviour('SensorComp')
cont = comp.findBehaviour('Process1Container')
path = comp.findBehaviour('Path')
```

InLineMachine.py

```
from vcScript import *

def OnSignal( signal ):
  pass

def OnRun():
  global sensorSignal,processTime,sensorComp
  processTime = comp.getProperty('ProcessTime')

  while True:
    stats.State = 'Idle'
    triggerCondition(lambda: getTrigger() == sensorComp and sensorComp.Value)
    part = sensorComp.Value
    part.stopMovement()
    stats.State = 'Busy'
    # Process starts
    delay(processTime.Value)
    part.startMovement()

# INITIALIZATION
comp = getComponent()
sensorSignal = comp.findBehaviour('SensorSignal')
sensorComp = comp.findBehaviour('SensorComp')
stats = comp.findBehaviour('Statistics')
path = comp.findBehaviour('Path')
```