Technische Universiteit
Eindhoven
University of Technology

/ Department of Mathematics and Computer Science / PDEng Software Technology

Model-Based System Engineering Design of Functional Modules for Configurable Topology

Ganduulga Gankhuyag

Model-Based System Engineering Design of Functional Modules for Configurable Topology

Ganduulga Gankhuyag

Date (October 2019)

Model-Based Systems Engineering Design of Functional Modules for Configurable Topology

Ganduulga Gankhuyag

Eindhoven University of Technology

Partners

Vanderlande	Eindhoven University of Technology

Steering Group Ganduulga Gankhuyag
Lennart Swartjes
Tanir Ozcelebi
Yanja Dajsuren
Bart van Dartel

Date October 2019

Document Status Public

SAI report no. 2019/082

Contact Address

Eindhoven University of Technology
Department of Mathematics and Computer Science
MF 5.080A
P.O. Box 513
NL-5600 MB
Eindhoven, The Netherlands
+31 402474334

Published by

Eindhoven University of Technology

Printed by

Eindhoven University of Technology
UniversiteitsDrukkerij

SAI report no.

2019/082

Abstract

Vanderlande is a global market leader in logistics automation systems. The company provides the systems in airport, warehouse, and parcel markets. Recently, the company has encountered a problem related to efficiency in development due to the diversity of the systems within the markets. To improve efficiency, the company produced a modular system architecture.

This project specifies the system architecture further by addressing the most fundamental software modules that are directly responsible for moving physical items throughout the systems. As a result of the problem, domain and requirement analysis, a reference architecture is proposed using Model-Based System Engineering methodology. The proposed architecture and design models were developed by emphasizing modularity and (re)configurability for the reusability purpose of the modules. The software development efficiency increases thanks to the reusability aspect of the proposed architecture.

Keywords

MBSE, Logistics automation, Modular system, Vanderlande, TU/e.

Preferred reference

Ganduulga Gankhuyag, Model-Based System Engineering Design of Functional Modules for Configurable Topology Eindhoven University of Technology, October 2019. (2019/082)

This PDEng thesis is approved by the supervisors and the Thesis Evaluation Committee is composed of the following members:

Chair:	Yanja Dajsuren
Supervisors:	Lennart Swartjes
	Tanir Ozcelebi
Members:	Bart van Dartel
	Joost van Eekelen
	Kees Huizing

Foreword

Vanderlande is leading in process automation of warehouse solutions and is a global market leader for value-added logistic process automation of airport and parcel solutions. To stay ahead in these markets, Vanderlande must remain an innovative company and must be able to deliver a solution with less time-to-market. As a result, some of our processes need to be updated to allow for this innovation and decreased time-to-market. In the past, each product could be designed for a specific customer. Nowadays, the need for standardization in the design processes is a must while still being able to tailor these systems to the customers' needs. Ganduulga's project is focused on the software architecture aspect of these shifts in requirements: what is necessary to standardize our current software solutions for the control of material handling systems while remaining configurable, and how would the architecture of this solution look like?

This project posed several challenges for Ganduulga. Firstly, it is of importance to know the playing field: what is the current way of working, what is lacking, and what is needed to improve. This entailed to obtain a lot of domain knowledge from the different markets and bringing that all together in a uniform manner. Secondly, using this gained knowledge, a software architecture must be devised that is uniform to all applications areas which entails finding the correct use cases and carefully assigning the requirements for the proposed design. Finally, it must be shown that the final solution is indeed a match with the requirements, i.e., show that the proposed solution is what is desired.

Ganduulga started on these challenges with a lot of enthusiasm. He quickly found out how broad the application area of Vanderlande is and how many different solutions there currently are. This introduced quite a challenging job for him to derive the requirements, use cases, and accompanied architecture. Nevertheless, Ganduulga did not give up, remained enthusiastic, and was able to explain the "why" in his report in a concise manner. In the end, he was also able to give a proper "how" and "what".

The final report provides one specific architecture that abides by the vision of Vanderlande, with accompanied rationale. This result provides insights in our current way of working and provides feedback on our vision. An even greater contribution lies in the classification and clarification of the problem: Ganduulga provided insights on how the functional requirements lead to a particular model-driven design with accompanied design patterns. These new insights will provide very valuable in the future to (re)classify our current vision and to derive the next steps and focus areas.

Lennart Swartjes
R&D, Vanderlande
29th of October 2019

Preface

This report was written in the scope of the project titled “Model-Based System Engineering Design of Functional Modules for Configurable Topology”. The report summarizes project results with corresponding in-depth problem analysis, procedures and rationales to obtain these results.

The project was carried out by Ganduulga Gankhuyag as his graduation and design project of the Professional Doctorate in Engineering (PDEng) program in Software Technology (ST), provided by the Eindhoven University of Technology, Stan Ackermans Institute. The project continued for 10 months in Research and Development (R&D) department at Vanderlande, Veghel.

Since the report is structured in a way that derives the technical solution from customer perspective problems, the content is appropriate for both technical and non-technical audiences. The report is more worthwhile for system and software architects who want to know more about modularity and (re)configurability aspects in the context of Vanderlande’s new platform architecture.

It is recommended for readers who want to implement the proposed design to refer to chapters 4, 5, and 6. Audiences who are interested more in problem investigation and derivation can read chapters 1-3. Project managers might be interested in reading chapters 8-9. Basic SysML/UML knowledge is required to understand architectural concepts.

Ganduulga Gankhuyag
October 2019

Acknowledgements

I would like to express my very great appreciation to all the people who supported me throughout the project in many different ways.

Firstly, I am deeply grateful to the people from Vanderlande. Special thanks to Lennart Swartjes and Bart van Dartel for giving me the opportunity to conduct my PDEng design project at Vanderlande. I am particularly grateful for the assistance given by Lennart Swartjes, my company supervisor. Pieces of Advice given by him have been of great help in formulating the design and report of the project. Without his continuous guidance, collaboration and critical feedback, I would not be able to make it a successful project.

From the Eindhoven University of Technology, I would like to express my deep gratitude to my university supervisor, Tanir Ozcelebi, for his patient guidance, enthusiastic encouragement, and useful critiques. His structured and detailed reviews of the project report helped me to realize the better flow of the documentation.

I would also like to extend my appreciation to everybody who is involved in the PDEng program, especially Yanja Dajsuren, Ad Aerts, and Desiree van Oorschot for giving me the opportunity to be part of the program. Furthermore, I am thankful to my colleagues from the PDEng program for sharing a great and wonderful experience, knowledge, culture, and many other everlasting memories. My acknowledgment also goes to Mohamed Abdel-Alim Alosta, a PDEng MSD trainee, who also carried out his design project at Vanderlande, for his patient and detailed reviews of my thesis.

Finally, I would like to thank my wife and daughter for their continuous encouragement and precious love through my ups and downs.

Ganduulga Gankhuyag
October 2019

Executive Summary

Technological development is shaping the world and the future. The impact can be realized not only from high-tech sectors but also the automation industry. Due to the acceleration of internet-driven technology, industrial automation systems are becoming more complex. The most vivid example is the Industrial Internet of Things (IIoT) which is the industry's adjusted form of the Internet of Things (IoT). IIoT is complex because everything is connected to everything. Another reason for complexity in industrial systems is continuously changing customers' demand, in terms of customization. Therefore, the industrial automation systems are forced to become more flexible and more efficient to resolve the complexity. The state-of-the-art solution is to reorganize system architecture as modular and as customizable as possible. As a result, instead of a sturdy and rigid system, the system will be composed of flexibly reconfigurable modules with well-defined responsibility.

Vanderlande is a global leader in industrial automation systems. To provide more flexible products efficiently thereby to add more value to their customers' business, the company started adapting the trend of modularity and customization in their product-systems. The company's recent change in the system architecture focuses on the client's business characteristics. Every system with a distinct characteristic is a Market Leading Concept (MLC). MLC consists of a predefined, fixed set of Functional Modules (FM). In this project, we aimed to specify the most fundamental FMs that are directly responsible for moving physical items throughout the system.

We proposed a master-slave design pattern to organize the architecture of FMs that perform essential tasks in the automation system. For the flexibility of (re)configuration in terms of topology, a parameterized FM mechanism is specified with support of the master component of the proposed design. Since modern industrial automation standards, namely IEC 61131-3 and IEC 61499, are compliant with the object-oriented concept, architectural specifications and designs are illustrated with the use of SysML diagrams.

Simulations of sequence diagrams and Anylogic simulation together show the conceptual validity and feasibility of proposed design and architecture. Based on the result, we conclude that the Model-Based System Engineering approach is efficient to show the feasibility of FM design.

Table of Contents

List of Figures

List of Tables

1. Introduction

This chapter provides introductory information about the project context, project goal, preliminaries, and outline of the report. Section 1.1 briefly introduces company domain and project problems with simple examples. Section 1.2 describes the project goal. Section 1.3 aims at giving the background information of approach, and architectural framework. Section 1.4 gives an overview of the structure of the report.

1.1 Context

1.1.1. Automated material handling

Automated Material Handling domain is one of the most popular industrial fields in the modern world. Before explaining the domain and its systems, the historical origin of the field is discussed.

In its simplest definition, material handling is a field of item processing that involves loading, moving, controlling, and unloading substances in any form [1]. In order to make the process efficient and safe, people started to automate the process by using various kinds of equipment and gadgets. Starting from such a simple idea, it became a whole industrial sector in the modern world.

Due to their high throughput, efficiency, and excellent quality of the process, Automated Material Handling Systems (AMHS) are extensively utilized in various industrial sectors. A clear example can be observed in e-commerce. In the ever-increasing demand of consumers, AMHS helps e-commerce to achieve productivity and thereby increases profitability [2]. Apart from this special sector, AMHS plays a big role in many other sectors such as manufacturing, mining, construction, the ship as well as aircraft building industries.

Another major reason, which makes this domain the largest and fastest-growing sector in the industry, is that an AMHS (i.e., a logistic system) is the only industrial system that touches every material as it moves through the whole supply chain. The world's most successful brands are aware of the strategic value of AMHS and use it in their supply chains, where they employ state-of-art technology and gadgets to meet their customers' needs.

One of the leading competitors in supplying AMHS is Vanderlande. The ultimate goal of the company is to help the customers to improve their competency within their markets. The company provides three different extremely advanced AMHS products to the customers. These systems are Airport, Warehouse, and Parcel and Postal systems as shown in Figure 1. These systems have common functionalities such as transporting, storing, identifying and tracking items. However, there are market-specific system challenges that make the systems different.

The Airport system is an end-to-end AMHS that facilitates smooth transit of both luggage and travelers from check-in and security check to baggage retrieval. Due to the rise of passenger volume, the system is required to be scalable and flexible.

The Warehouse system is an AMHS that delivers a wide range of products from suppliers to end-users. The customers' challenges are the accuracy of deliveries, dealing with a wide range of products and with seasonal increase and decrease of the merchandise.

Parcel and Postal systems handle parcels, mails, and posts efficiently. The customers' challenges are achieving shorter order lead times, later cut-off times, as well as dealing with smaller orders and greater product diversity.

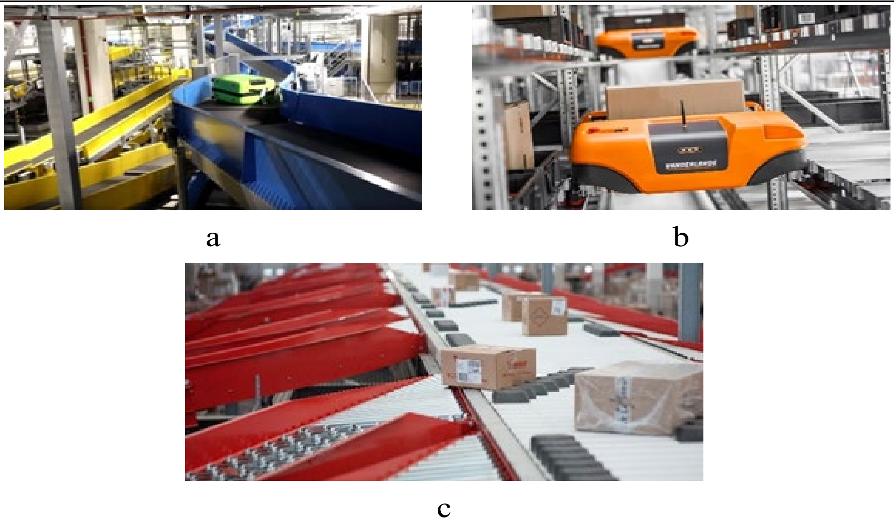


Figure 1 – a. Airport system b. Warehouse system c. Parcel and Postal system

1.1.2. The complexity of software development

Vanderlande systems handle a wide variety of items such as baggage in Airport, products in Warehouse, and packages in Parcel and Postal systems. A typical Warehouse system is shown in Figure 2. From the figure, it can be clearly seen that the system can be enormous, needing to control a huge number of devices. Moreover, nowadays company customers demand their market-specific systems. In this case, a large amount of various equipment has to be arranged in different forms of setup. These characteristics make the system extremely complex. Such a complex system has complex software and hardware. In the context of this project, a complex system is a system that has multiple functionalities that are carried out by various equipment included.

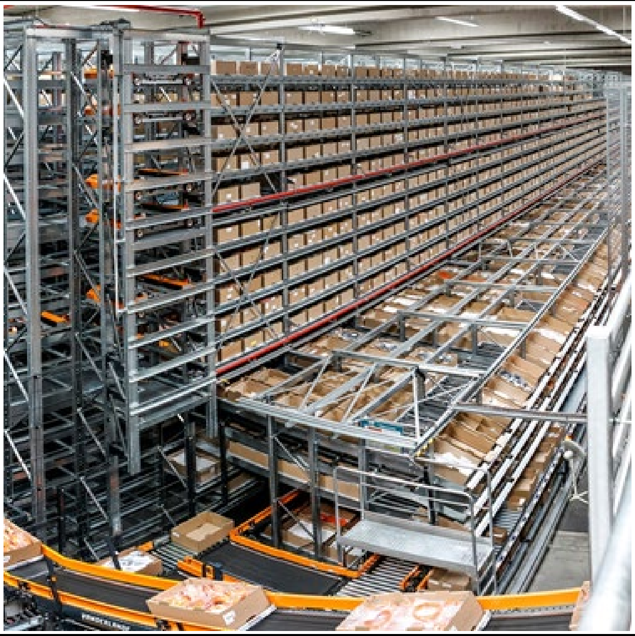


Figure 2 – Typical warehouse system

Even a relatively small sorting system, as illustrated in Figure 3a, has complex software to control all the hardware components in a synchronized and scheduled manner. In the well-scheduled and synchronized system, software controls integrated actions of multiple hardware components in a timely aligned routine. To be specific, the software of this sorting system has functionalities of transporting items by conveyors, diverting item flow by mobile sorting components, and distinguishing various kinds of items by different kinds of sensors within the system. While delivering these functionalities in the desired behavior and in a synchronized manner, the software also processes all the data related to each item. These characteristics make the system more complex. Thus, Vanderlande seeks appropriate systems and software architectures to cope with such complexity.

In the fast-growing technological world, industrial control systems are forced to be developed quickly even though they are deployed on complex systems similar to the ones previously mentioned. The typical solution for coping with complexity has always been modularity. In general, modularity enables the system or software to be composed of small building blocks that can be reused and customized in product-family systems. Hence, achieving a proper degree of modularity brings not only a solution to the complex system problem but also an additional benefit of generalizing product-family systems.

In the Vanderlande's product-family systems, namely Parcel and Postal, Warehouse, and Airport systems, numerous pieces of equipment operate roughly the same way. The company wants to achieve a proper degree of modularity by enabling one software module to control functionally equivalent components or equipment. With added software modularity, Vanderlande can generalize existing product-family systems into a universal platform.

Apart from the software complexity, the current way of developing software whose behaviors are variations of the existing ones is not efficient within the company. For instance, suppose that a customer has a system with one sorting output as seen in Figure 3a. When the customer wants to add more hardware, such as one more sorting output as depicted in Figure 3b, the current solution for making sure the resulting system has the correct behavior is that a team of Vanderlande engineers develop new software for the new infrastructure from scratch.

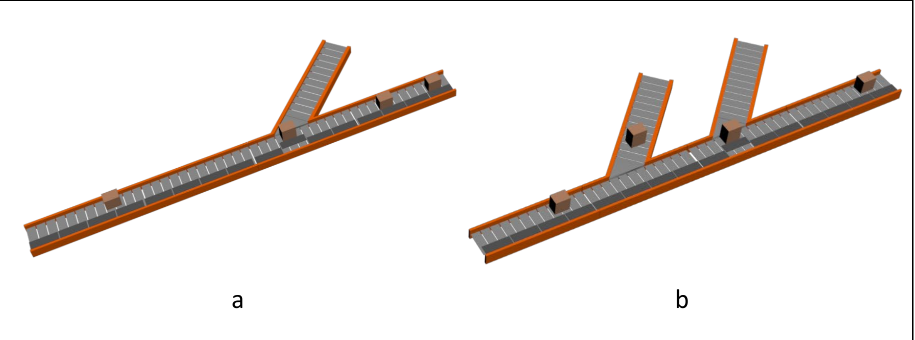


Figure 3 – a. Simple sorting system b. Sorting system with two outputs

Vanderlande's current system software is not (re)configurable, which means it cannot easily accommodate changes made to the hardware infrastructure of a system. Vanderlande engineers spend a significant amount of development time by duplicating a majority of the steps in building a variation of the previous software version, which is very costly.

1.2 *Project Goal*

The company wants to design and develop a new **modular** and (re)**configurable** software architecture that can be used as a reference for AMHS software development within Vanderlande. The developed software architecture shall not only save development time and cost for its material handling domain, but also allow an AMHS to change from its current configuration to another configuration without being taken offline. This makes it possible to maintain system effectiveness when (sudden) changes in customer demands or unpredictable events such as failures and disruptions occur [3].

1.3 Preliminaries

This section gives general information on methodologies that helps to achieve project goals systematically and efficiently. Since the project intends to answer system-level questions, the Model-Based System Engineering (MBSE) approach was used for the purpose of efficiency. An introduction to the MBSE approach and its advantages over the conventional development approach, the document-based approach, are explained in Section 1.3.1. The systematical reasoning roadmap to reach the project goals given in Section 1.2 is defined in Section 1.3.2.

1.3.1. Model-Based Systems Engineering

Any system can be viewed as a set of system components. The system concerns interactions not only internally between the components but also externally to its environment as a whole. The process of designing and developing these components and interactions is defined in the scope of Systems Engineering. Systems Engineering is a multidisciplinary approach to present system solutions to satisfy the diverse requirements of the customers and stakeholders.

A system engineering conventional approach to design and develop systems is based entirely on documents. These documents are intended to capture the results of every single development phase of the project life-cycle from customer requirement analysis to the final product. In those life-cycle activities, system engineers produce all kinds of disjoint documents and artifacts manually. Examples of such artifacts are the requirement specification document, the architectural description document, the system design description document, and the interface specification.

These artifacts are key communication utensils of agreement between users, customers, developers, and testers. Besides, the artifacts are characterized by different forms of documents, such as textual documents, spreadsheets, presentation files, and diagrams. Unfortunately, there are several fundamental limitations in the document-based approach. The most important issue is that this approach is expensive to maintain. Quite often we can see the requirements in a project change after the project has already started. Thus, the relevant artifacts are to change accordingly. However, the information is spread over multiple documents. The changes then have to be reflected in every single one of the “relevant” documents, but typically the workflow is tedious and document traceability is poor. Therefore, it is an inefficient way to work for engineers.

As a result of advancements in computer processing, storage, and network technology, MBSE has received attention in a wide range of industrial domains. The MBSE approach not only mitigates the drawbacks of the traditional document-based systems engineering approach but also brings additional advantages to the systems engineering field [4]. Examples of those advantages are traceability and integration of the system development processes. One of the most important features that MBSE brought to the field is simulation capability.

Systems engineers adapted the Model-Based approach from other engineering disciplines, such as electrical and mechanical engineering. Mechanical engineers started using two-dimensional and three-dimensional computer-aided design tools instead of boards. Meanwhile, it was efficient for electrical engineers to utilize computer software to capture and analyze circuit schematics instead of manually building electrical circuits. In a similar way, the Model-Based approach is becoming standard practice for systems engineers because it improves productivity and quality of the work. In the MBSE approach, the project development team has a common repository of all kinds of joint analysis and models in one generic standard. It makes project progress more efficient because models are highly synchronized to each other, consistent because of standards, and traceable not only to the high-level requirement analysis but also to the low-level hardware and software designs.

Moreover, other features that MBSE brings to the field are enhanced knowledge transfer or communication between stakeholders, reduced development risk, verification, and validation. MBSE approach is becoming more popular also because it increases the level of abstraction and the level of automation. Depending on tools that system engineers utilize, automation features such as code and documentation generation and visual simulation are possible nowadays.

There are three pillars in MBSE [5]: Method, Language, and Tool. In the next section of the chapter, the method is discussed in detail. Language and Tool are analyzed in Chapter 4.

1.3.2. Architectural reasoning methodology

According to a recent study, an average person thinks between 60000 to 80000 thoughts in a day. It would be great to achieve one's goal if these thoughts are interconnected systematically or focused on one point. Unfortunately, the majority of these thoughts are absolutely irrelevant or the same repetitive worries. A systematically structured way of thinking helps enormously to plan steadily and proceed accordingly.

In the rapidly changing, advanced technology-driven industrial era, an architectural reasoning methodology supports architects, designers, and engineers to realize their ideas into a product in a highly structured manner. There are numerous such well-thought methodologies for different purposes and perspectives. A classification of architectural reasoning methodologies is depicted in Figure 4 [6].

In this project, CAFCR was chosen because the framework is commonly facilitated for multidisciplinary system-level projects. The other multi-view reasoning approaches, especially Kruchten's 4+1 software architecting method, were considered. However, because the nature of the project context is multidisciplinary and relevant to system-level architecting including hardware and software, CAFCR is preferred over Kruchten's 4+1 method.

CACFR is a reasoning methodology to find a problem and solve it with concrete rationales and a straightforward procedure [7]. An original idea of the methodology aims to decrease a conceptual gap between the industrial and academic worlds. Rationales are solid because an architect has to put herself into different project stakeholders' shoes, enabling designers to view problems from different perspectives.

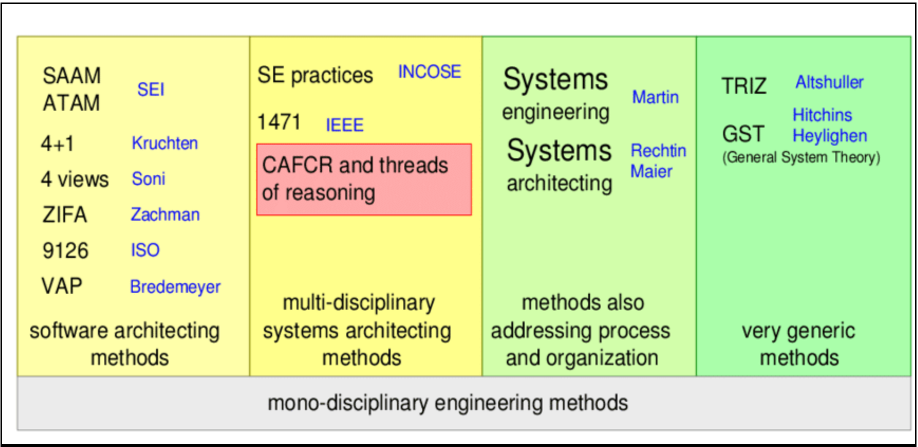


Figure 4 – Classification of architectural reasoning methodologies [6]

The CAFCR methodology is composed of five different views as depicted in Figure 5: Customer objective, Application, Functional, Conceptual, and Realization. The Customer objective view explains what problem customer wants to be solved. The Application view describes the context or environment of the problem. The Functional view defines functional and non-functional requirements from stakeholders. The Conceptual and Realization views define the design of the solution at high and low levels respectively. CAF views are considered separately from CR views because CAF views are intended for analyzing the problem from different levels and different perspectives and CR views are meant for design.

By putting the views in this order, a solution to the problem becomes clear to stakeholders even if the stakeholders do not have any technical knowledge. The reasoning transition and flow are clear as the views are iterated. Besides, a view drives the adjacent next view and in the opposite direction, the view enables or supports the previous view.

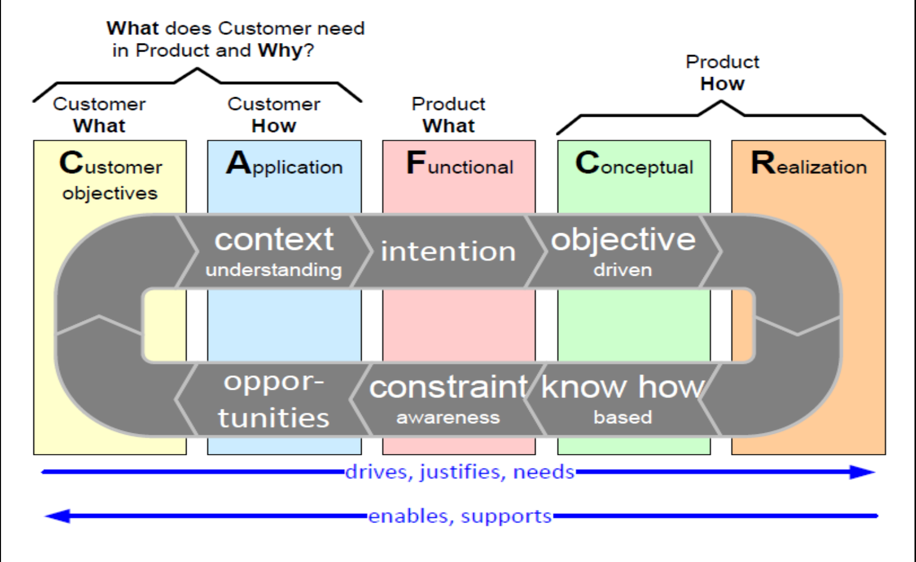


Figure 5 – CAFCR [7]

1.4 Outline

The entire project report is structured based on the CAFCR architectural reasoning methodology. CAFCR views are mapped into Problem analysis, Domain analysis sections, Requirement analysis, System Architecture, and System Design chapters respectively.
Chapter 2 explores project problem insights with reasons why this project was initiated in the first place from a customer's perspective. Furthermore, the context of the problem and the environment where this problem is allocated in the system are also defined in Chapter 2. Detailed functional and non-functional requirements are listed as the system requirements in Chapter 3. Based on the previous chapters, the design of the architecture solution is introduced in chapters 4 and 5. Chapter 6 describes our evaluation of the design. The conclusions are given in Chapter 7. Finally, Chapters 8 and 9 of the report, provide the details of project management related activities.■

2. Project problems and conceptual solution

This chapter consists of two main sections. Section 2.1 highlights the problems in the scope of the project. These problems are analyzed from the main stakeholders' point of view, which depicts the Customer objectives view of the CAFCR.
Section 2.2 provides the conceptual guideline of the solution to the highlighted problems. This guideline was formulated by analyzing a similar problem solution in the industrial automation domain. Moreover, we learn that the company has started applying this conceptual guideline by producing a new system architecture. Furthermore, the scope of the project is defined within the new system architecture. This section is the Application view of the CAFCR, which defines the project scope within its environment or context. Chapter overview is depicted in Figure 6.

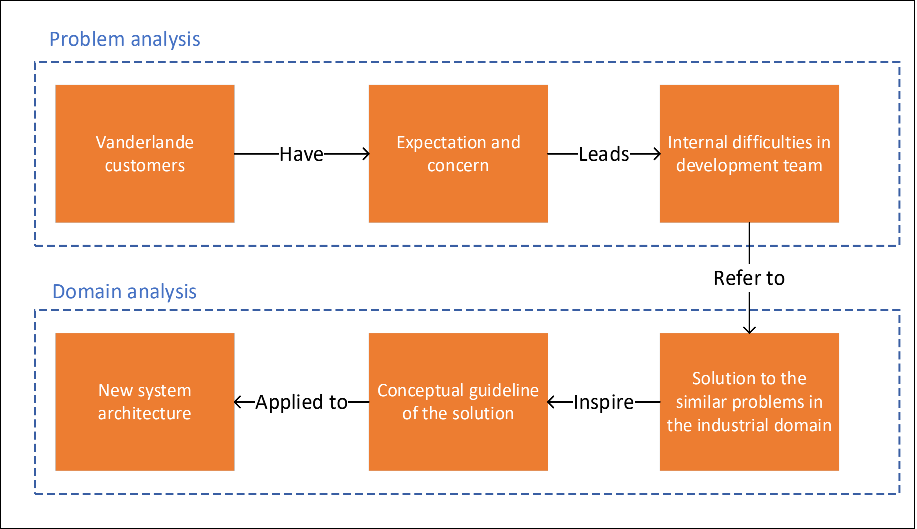

Problem analysis
Vanderlande customers
Have
Expectation and concern
Leads
Internal difficulties in development team
Refer to
Domain analysis
New system architecture
Applied to
Conceptual guideline of the solution
Inspire
Solution to the similar problems in the industrial domain

Figure 6 – Chapter 2 overview

2.1 *Problem analysis*

This section is the Customer objective view in the CAFCR framework, which is illustrated in Figure 5. This view captures the stakeholders' concerns without mentioning any technical details. In the project, company customers and company development teams are considered as the main stakeholders.

Vanderlande customers are companies, organizations, or people who need Airport, Warehouse or Parcel systems. These systems are typically complex. There are two main reasons as to why Vanderlande systems are complex.

The first reason is the diversity of the customer's market. Even though Vanderlande provides logistic process automation systems in airports, warehouses, and postal sectors, customers demand different types of systems depending on their business needs within these three sectors. Such diversity can be explained with an example of Albert Heijn and Zalando's solution in the warehouse sector. In spite of the fact that these companies use Vanderlande's the same Warehouse system, their requirements are different depending on what business segment they operate in. On one hand, Albert Heijn must cope with the growing volumes of diversified food stock and multiple store formats. On the other hand, as an e-commerce company, Zalando must cope with seasonal and fast-changing trends efficiently and handle product returns flexibly. In a few words, there has been an increasing demand for a generic logistic solution that can be extended with customer-specific requirements.

Secondly, typical systems that company supplies are relatively large in terms of size. The larger the system is, the more equipment the system should manage as well as the interactions between them. Moreover, there is a wide variety of equipment that is used as building blocks of the company systems. As an example of Vanderlande systems, a typical Warehouse system is depicted in Figure 2. This system consists of a wide range of equipment, such as conveyors, robots and sensors. Even single equipment varies depending on their purpose. For instance, varieties of sensors are barcode sensors, weight and dimension sensors. In addition, these diverse equipment is utilized in hundreds and thousands of places within this Warehouse system.

In addition to the systems being complex, the company customers have numerous expectations and concerns from Vanderlande. One of the most important concerns is that they demand these complex systems to be delivered with excellent quality and outstanding performance in a short time. Satisfying the company customers' concerns leads to internal difficulties within the development team of the company. The development team eventually satisfies the quality and performance of the customers' requirements for their systems. However, it has always been a challenge to deliver complex systems in a limited amount of time because the current process of system development is time-consuming and inefficient to cope with the complexity of the systems. Within the scope of the project, we analyzed the problem of the inefficiency and identified problem causes.

We identified the major causes of the inefficient and time-consuming software development process. The reasons are the low level of software reusability and the conventional way of developing software within the company.

The current software development process is inefficient and time-consuming because a major part of the existing system software is not reusable. To develop a complex system quickly, the company should have a reusable template software. Since this template is non-existent, the developers frequently encounter the problem of duplicating previous work over and over again. A piece of evidence that the company does not have the template software is that Airport, Warehouse and Parcel systems are not entirely generalized. In terms of hardware, these systems share common equipment but in terms of software, these systems have distinct software, even though the majority of the software is meant for delivering fundamental logistic automation functions.

Another reason for the inefficiency is that developers within the company follow the conventional way of software development, also known as manual coding, which is error-prone. It is common to encounter various bugs in the software when developers utilized manual coding. When the bugs are present, debugging usually requires a lot of time and effort. Furthermore, code is not understandable to other people unless the developer wrote detailed documentation about his developed part of the software. Understanding other developers' software logic and concept takes time. Therefore, the conventional way of developing software, the paper-based software development approach, is inefficient.

In order to solve these problems, we looked into the trend of the Industrial Internet of Things (IIoT) within the industrial automation systems. We learn that IIoT facilitates software development by making use of reusable modules to cope with the complexity of the systems. Furthermore, we also explain how the company is transitioning from the conventional way of developing software to software modeling approach in the next section.

2.2 Domain analysis

This section aims to define the conceptual guideline of the solution to the above-mentioned problems by referencing to the trend of the Industrial Internet of Things (IIoT). Section 2.2.1 explains a trend of IIoT and Cyber-Physical System (CPS) that elaborates on how reusable modules can be used as a solution to inefficiency in the development of a complex system. Considering the complexity, the company has already started facilitating such reusable modules in the system architecture, which will be discussed in Section 2.2.3.

2.2.1. New trends in the industrial automation domain

Nowadays, most industrial automation companies, including Vanderlande, follow the classic automation pyramid. The pyramid is an integrated technological hierarchy of different levels of automation in the industry. It is composed of five levels, namely Field level, Control level, Supervisory level, Planning level, and Management level. The automation pyramid hierarchy is captured in Figure 7a [8].

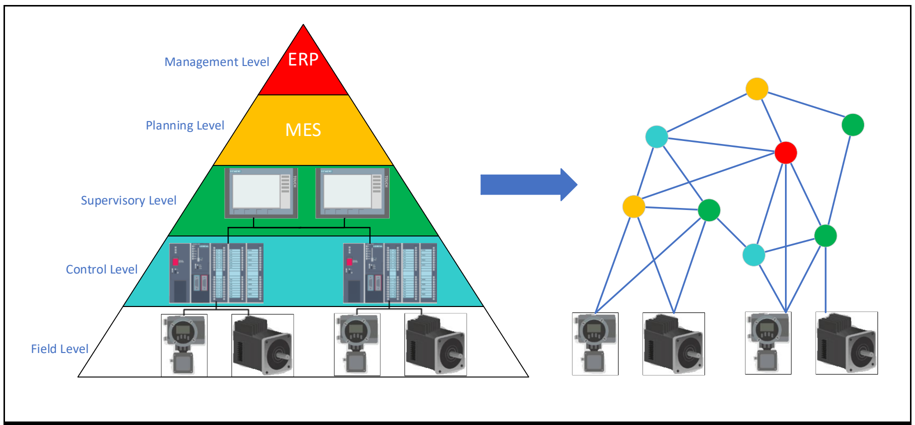


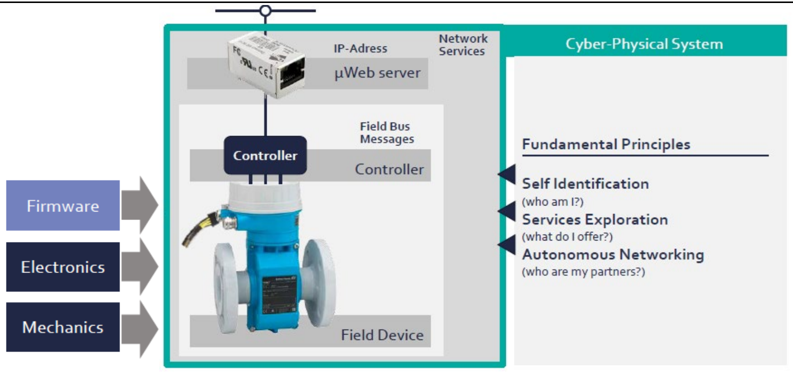
Figure 7 – Transition from classic automation to IIoT

Industrial physical components that detect and move varieties of materials in factories or industrial applications belong to the Field level automation. Sensors and actuators, which are responsible for detecting and moving items receptively, play a major role in the Field level. The Control level manages or runs the Field level. At the control level, the Programmable Logic Controller (PLC) is the main actor. It takes sensor data from the Field level and instructs corresponding actuators to react to it. The supervisory level utilizes Supervisory Control And Data Acquisition (SCADA), which accesses data and control multiple systems from a single location using a graphical user interface or Human Machine Interface (HMI). On top of the Supervisory level, there is the Planning level that utilizes a computer management system. At this level, the complete manufacturing process from raw material to the final product is monitored by the Manufacturing Execution System (MES). The topmost level of the pyramid is the Management level. This level is dedicated to monitoring and controlling the company's management, which includes manufacturing, sales, finance, and others. Enterprise Resource Planning (ERP) is well known for this level.

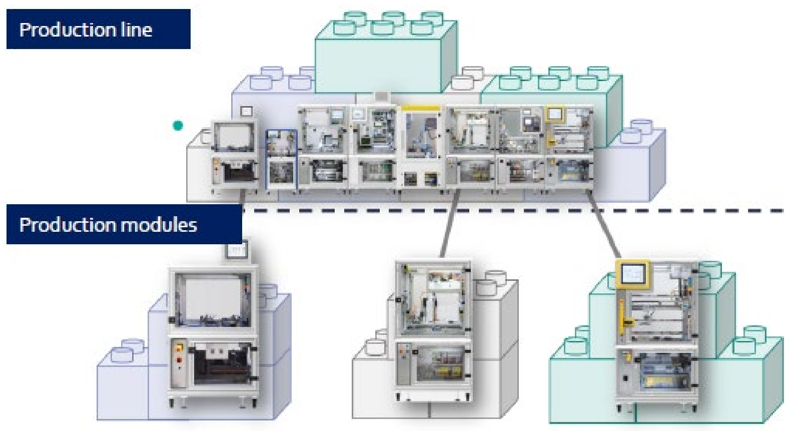
Due to the internet-driven, rapidly emerging new technologies, the industrial automation domain is forced to change the classic automation pyramid by adopting these new technologies in order to enhance the manufacturing and industrial process. One example of the new trends is the Internet of Things (IoT) in which all system components are interconnected. Industry customized the concept for its own context, creating the concept of IIoT. The future of the industry is greatly influenced by the IoT, which results in IIoT to turn the classic automation pyramid hierarchy into a more flexible network. Figure 7b shows how the IIoT network is formed in the future. In this network, all system components are interconnected. Thus, typical IIoT systems are complex just like Vanderlande's systems. However, IIoT embraces modularity to efficiently build such complex systems.

Cyber-Physical System (CPS) is a technology that is used in IIoT. A smart module, as shown in Figure 8a, is an example of a CPS. In the smart module, all small field devices are equipped with electronic, and mechanical solutions as it was in the past. On top of that, it has its own micro web server, which enables the module to be connected to a network. Improvement to the old system of the module is the software in the smart module. In the past, the system achieved modularity in electronic and mechanical components but not in software. Therefore, instead of a monolithic complete software for the whole system, engineers started to develop modules with small firmware. The rest of the software is filled with Internet applications that can be easily acquired. In the same manner that a USB printer can be connected to any personal computer, this type of smart module can be attached to any CPS. Fundamental principles that make the "plug-and-play" concept factual are the smart module's features, namely Self Identification, Service Exploration, and Autonomous Networking [9].

Furthermore, the smart module concept can be expanded in terms of size to bigger industrial line equipment or production modules. The production modules resemble Lego bricks. As children play around with the Lego bricks, the production modules also can be rearranged without any complication because both Lego bricks and production modules have standardized connections or standardized (network) interfaces between them. Therefore, it is easy to construct the production line composed of these modules. The concept is visualized in Figure 8b. As a consequence, the industry has been transforming rapidly from a facility with a fixed location to changing location; from monolithic to modular; from rigid hierarchical to distributed; and from wired to wireless.



a



b

Figure 8 – a. Smart module b. Modular factory [9]

From these examples, it can be concluded that the system would be more flexible and can be efficiently developed when it is designed in a modular way. Furthermore, these examples inspired to identify a conceptual guideline of the solution to the problem of inefficiency in complex systems' software development.

2.2.2. Conceptual solution

This section explains two main aspects, namely reusability and modeling, of a conceptual solution to the problem of inefficiency in systems' software development. These aspects allow the company to develop complex systems efficiently. In Section I, a conceptual guideline of realizing the full potential of reusability is defined. The guideline helps to formulate the architectural solution of the complex systems' software. In Section II, how modeling helps to increase development efficiency is explained.

I. Reusability

Reusability is one of the most discussed topics in software engineering. It promises a reduction of both cost and development time for software systems, as well as better software quality [10]. In order to solve the problem of inefficiency in software development, the company should adopt reusability in the systems' software architecture. We created a guideline that encourages to apply two levels of reusability to the system software architecture in order to realize the full potential of reusability. In the first level, complex and monolithic systems should be modularized thereby system software's small building blocks, also known as modules, can be reused in the different system software applications. In the second level, these applications should be built configurable. Configurable software applications can be reused in similar applications with different configurations of modules. The guideline of realizing the full potential of reusability is explained with examples of Vanderlande's systems below.

i. Modularity

The first level of reusability is modularity, which is inspired by examples of IIoT and CPS. These examples encourage to adopt modularity to cope with system complexity efficiently. To develop a complex system efficiently, the company should have a template system that is composed of small building blocks, also known as modules. In the case of customization, these building blocks can be arranged in a different setup depending on the customer's demand. Vanderlande achieved a proper degree of modularity in hardware but modular software has always been an issue because the software has to be changed relatively faster and more frequent than hardware. Thus, achieving a proper degree of modularity in software would be a solution to the complexity.

How to modularize the system software would be the next question to be addressed. The company has been supplying their solutions to customers with the most fundamental logistic functionalities. These functionalities are the main reference for how to modularize system software. By modularizing the system software, the company will gain extensively in product development time because once small building blocks of the software are defined, Vanderlande will gain universal system software, which can be customized to Airport, Warehouse or Parcel systems.

Currently, all Vanderlande systems have their own distinct software even though most of the functionalities are the same. Common functionalities are the most fundamental operations of the software to process items such as transporting, identifying, and sorting items. These functionalities are a key point to split system software into modules. In the future, the company aims to achieve that the desired system consists of Functional Modules, which are building blocks with their own distinct responsibility. As a result, Vanderlande systems can be generalized into one logistic automation system that consists of predefined, developed and tested modules. It can be explained with the reusability concept. The company lacks software reusability, which leads to a waste of resources such as system production lead time, and manpower.

Vanderlande will gain significantly by transforming its current distinct systems to the modular systems. The advantage of the modularity and reusability in the Vanderlande systems is depicted in Figure 9 using simple puzzle blocks. In the figure, Vanderlande's current situation of distinct systems is illustrated at the top. Airport, Warehouse and Parcel systems share common hardware components but the software is different for each system. This is why the company needs to modularize the systems software.

When the company achieves a proper degree of modularity in system software, all the systems are generalized into one universal modular platform that has shared functionalities. This universal system is considered as a template system. It will be straightforward for the developers to build Vanderlande systems from the template because all the functionalities are predefined, developed, and tested.

Furthermore, customer market-specific solution becomes easy to develop from the modular systems. Figure 9 further shows how the reuse of modules helps to cope with customer market-specific solutions with examples of Albert Heijn and Zalando systems. These solutions require different functionalities due to market-specific characteristics. However, there will always be shared software modules among the solutions because fundamental functionalities are meant for the Warehouse system.

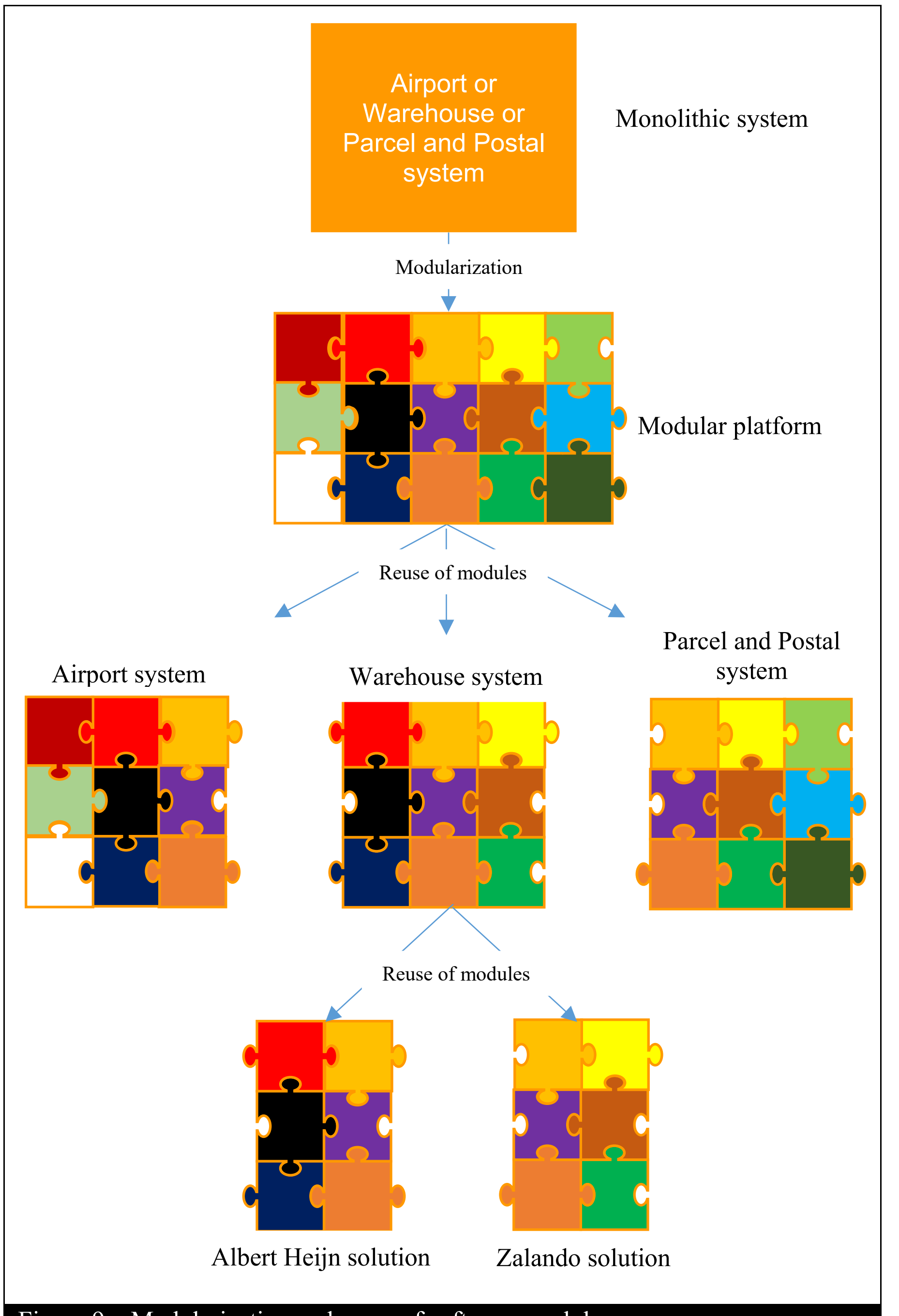


Figure 9 – Modularization and reuse of software modules.

ii. Configurability

The second level of reusability is configurability. Configurable software is another form of reusability. This is where developers reuse the entire software application. As opposed to the modularity, the scope of the configurability is bigger. On one hand, modularity allows developers to reuse software entities again in different applications. On the other hand, configurability enables developers to reuse the entire software application for similar projects. Suppose the application of Alber Heijn's software in Figure 9 was built configurable. In the case of hardware change in the system, there is no need to develop new software. Instead, the software can be reconfigured for the new setup of hardware. As an example, two different configurations of the same Albert Heijn system are depicted in Figure 10. As shown in the figure, the structure of the software is not modified but the size is configured. Configurability is further clarified with a practical example.

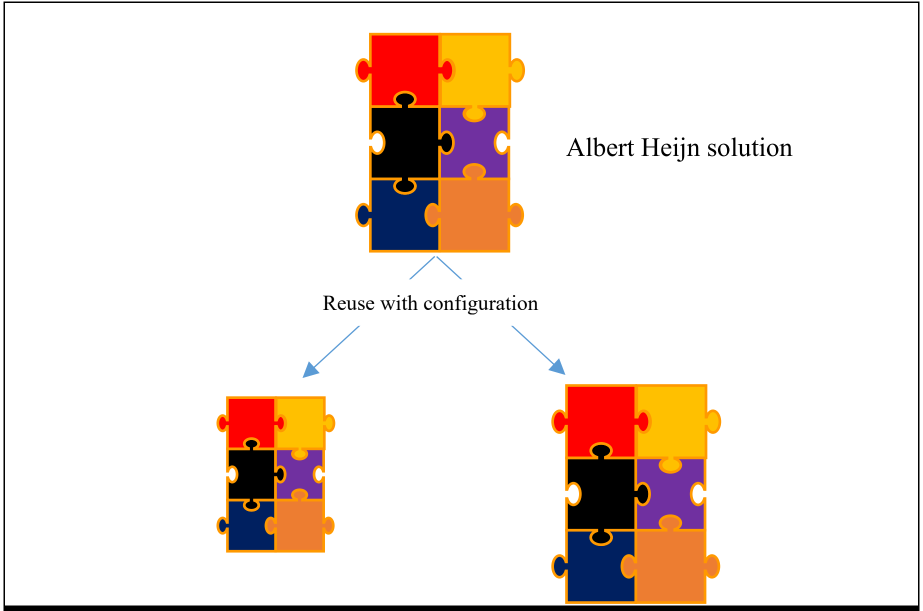


Figure 10 – Reuse with configurability

Conceptual configurability, Figure 10, is clarified with a more concrete example of a small system in Figure 11. Existing customers tend to extend their automation system with a few more hardware components. In this case, engineers must discard previous software and deliver another software regarding the system infrastructure change.

The current software architecture of the systems is not adaptable to this hardware infrastructure change. A software configurability is one of the main interests in the project for making the system software reusable with regard to hardware update. A simple example is given in Figure 11 to show an advantage of the configurability concept in the project. The figure is a more detailed illustration of Figure 3 including all equipment that is involved in the task of sorting items by conveyor belt.

Suppose a customer had a system with only one sorting path in it. Figure 11a shows the infrastructure of the system. Because the customer's business was successful, he decided to enlarge his system with another sorting path in the system, which is shown in Figure 11b.

From Figure 3, it seemed like adding one more conveyor is the only change to the initial system layout. However, from the detailed figure below, it can be seen that adding one more sorting path needs other associate devices such as barcode scanner to distinguish items, Photo Electric Cell (PEC) to check item presence and extra diverting shoes to direct items to the desired path. Functionalities and interactions of additional hardware should be captured by the new software version.

The current situation is that Vanderlande engineers have to discard initial software entirely and develop new software dedicated to the new hardware layout. This is not a flexible and efficient way to deal with this customer's need. Through this project, the company wants to investigate a method that existing software in the original system layout still works in the customized layout in Figure 11b.

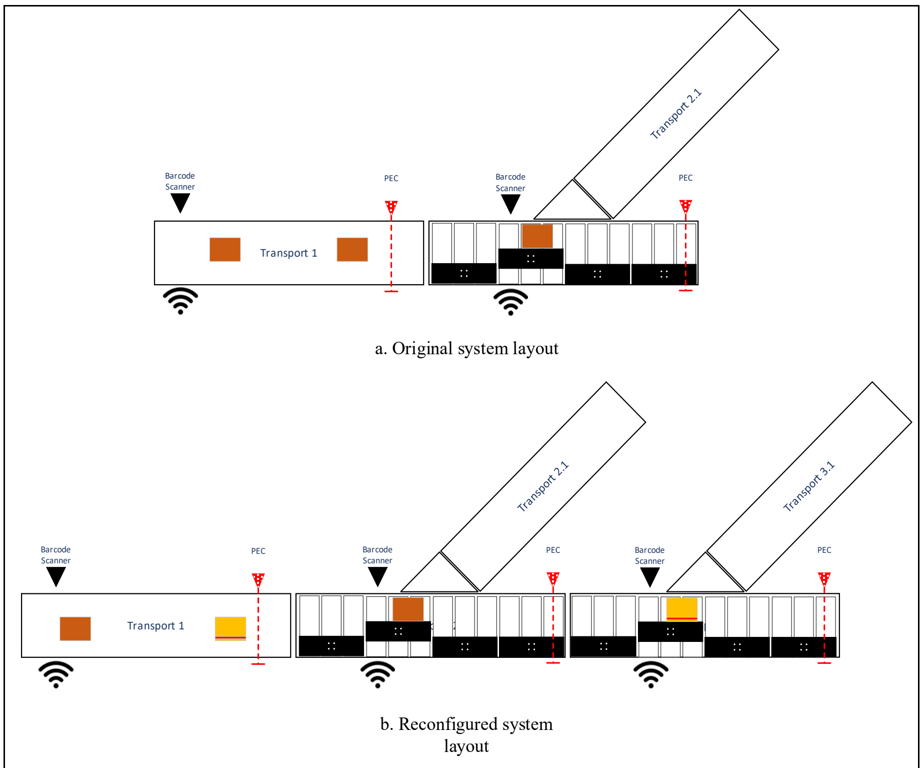


Figure 11 – Example system infrastructure for configurability

For instance, in terms of modularity, software for Figure 11a is composed of several entities each of which has distinct responsibility. These entities can be reused in a totally different system layout. Let's suppose there is a software entity that controls only a barcode scanner. Since a barcode scanner can be used everywhere in any system, the corresponding entity can be reused wherever the device is needed.

In terms of configurability, the same software controls both layouts in Figure 11a and Figure 11b with a different configuration of entities. In other words, the entire software application of the original layout is reused with a slight configuration change in the new layout. However, this system change works as long as the change is within a predefined range. The predefined range is the scope of original system software functionality. Put differently, if the company customer requests a totally different functionality that was not in the original system, configurable software is not applicable in this case.

To conclude the conceptual guideline, systems' software should be modular and further configurable to realize the full potential of reusability. The reusability helps to address the problem of inefficiency in software development within the company.

II. Modeling

Apart from the above-mentioned system software relevant guidelines, the company desires to increase development efficiency with modeling techniques. Nowadays, the modeling is considered more productive than conventional software development activities such as manual coding because code is error-prone when the system software becomes larger. Besides, code is difficult to understand for other people unless the software developer wrote clean code with clearly explained documentation. Therefore, Vanderlande is adapting modeling techniques because models are easier to understand

and error can be detected in the early stage of development thanks to Model-Based simulation and verification with stakeholders.

Modeling is a requirement from the company to conduct the project for the following reasons. There are different documentation about systems and software in the company dedicated only to specific professionals while people from other departments cannot understand the documentation. To communicate with the same language, the modeling is proposed because diagrams and models are more likely to be understood by people from different backgrounds.

Modeling also has advantages of analyzing and verifying the concept with stakeholders in the early stage of development. With this advantage, a developer can immediately prove his concept of the system with stakeholders' imagination of the system. In other words, it complies "fail fast" principle.

Another significant benefit that engineers profit from the model is a code-generation feature from models. Nowadays, it is becoming more practical to generate code from the models if the models are defined in great detail. However, the code-generation feature has several limitations. It is almost impossible to generate whole system code from the models. Moreover, the generated code is difficult to understand for developers for the purpose of modification. In this project, we did not go towards this feature because the aim of the project was not implementation.

Although implementation was not a primary purpose in the project, we analyzed several standards of approved implementation technology in the industrial automation control domain. These standards are crucial because this project's main deliverable, reference architecture, should allow engineers to implement systems complying with these standards. Additionally, our project design models should also comply with the standards for the ease of implementation.

In this field, most sectors utilize PLCs. A PLC is an industrial digital computer that can be programmed for controlling manufacturing processes such as assembly lines, robotic device and heavy machinery in the system. Industrial engineers' first choice for the automation controller has always been PLC because it is robust in the harsh industrial environment and thereby it is also highly reliable. Another advantage is that it supports the modularity concept. In the industrial equipment, engineers achieved building physically modular PLC or modular hardware. Modular hardware required modular software. A (re)configuration mechanism was missing in the software of the system to form it more flexible. To solve this problem, standards for the PLC programming language have been developed over the last few years.

The International Electrotechnical Commission (IEC) has been publishing the standards for modularity and (re)configurability. The first influential one was IEC 1131, especially part 3 (IEC 1131-3), which deals with the software model and programming languages for Industrial Process Measurement and Control Systems (IPMCS) [11]. In the standard, five programming languages were defined, emphasizing software reuse.

Due to the numbering system in the IEC, the standard was renamed as IEC 61131 in the later version of the standard. In a similar way that object-oriented programming (OOP) languages use class, the standard IEC 61131-3 defined Functional Block (FB) as an entity that has encapsulated data structure and algorithm working with this data. In application, these FBs can be connected to each other in a data-driven approach [3].

The IEC 61131-3 standard is extended in IEC 61499. IEC 61499 promoted event-driven FBs, which are also modular and reusable. The main advantage of IEC 61499 over 61131-3 was the dynamic reconfiguration mechanism. The reconfiguration mechanism was enabled thanks to getting rid of global data, indirect data access, and the event-driven approach.

These standards are enablers of modularity because the standards support the OOP paradigm with their FBs. The FBs in these standards can be considered as analogous to a class in the OOP paradigm. Modularity is a feature in the OOP paradigm because OOP allows developers to construct software by manipulating objects that are instances of software building block: a class. For this reason, the standards are enablers of the modularity and configurability.

2.2.3. Vanderlande's approach to reusability

Vanderlande has been innovative and visionary about new technologies to remain as a leader in the domain. Therefore, the company started applying the guideline to the reusability. The first step is the systems' modularization. The company already commenced modularizing the systems with new high-level architecture. As a second step of the reusability, the company initiated Next Level Automation (NLA) that aims to configurability of the systems' software. Our project is auxiliary to the NLA project and new system architecture. Thus, the scope of the project is defined in this section.

Recently, Vanderlande identified customer-specific market diversity within the same Vanderlande system. The market diversity within the same Warehouse system was explained with examples of Albert Heijn and Zalando in Section 2.2.2. Subsequently, the company has created a new system architecture to figure out the needs of diversity and issues that are mentioned in Section 2.1. Key new concepts in the new architecture are are i) Market Leading Concept and ii) Functional Modules.

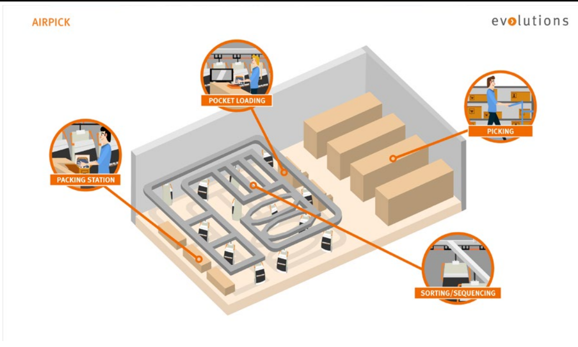
Market Leading Concept (MLC) is a standardized distinct solution for every business segment, namely food, fashion, parcel, general merchandise as it is specified by the company. To capture the opportunities in the growing markets for warehousing, parcels, and airports, Vanderlande has chosen to focus on specific market segments and to tune product offerings accordingly. For each business segment, these offerings center around market-leading concepts: system concepts that answer to specific requirements for a business segment. The requirements are determined by the commonalities in the value drivers of customers in that business segment.

As examples of MLC in a Warehouse system, AIRPICK and FASTPICK for fashion and general merchandise, and STOREPICK for the food market segment are shown in Figure 12. Each of these MLCs has distinct traits to offer for customers but they have some common functionalities.

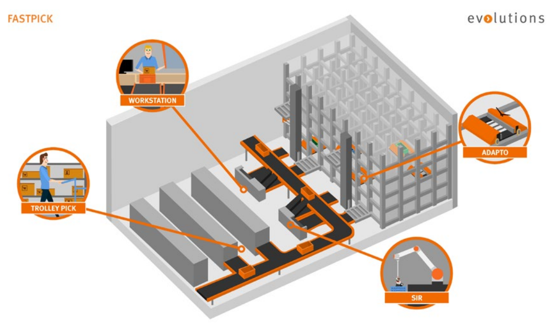
AIRPICK [12] combines efficient picking with flawless automated sortation to individual orders in the pocket sorter, AIRTRAX Pocket. AIRPICK can sort an extremely wide range of products at a low investment level. FASTPICK [13] is a goods-to-person order fulfillment system that uses the advanced ADAPTO shuttles for product storage and retrieval for day-to-day operations. To handle short-term peaks in a cost-efficient way, highly efficient trolley picking is used to complement the goods-to-person system. STOREPICK [14] is a robotized, end-to-end automated case picking (ACP) warehouse solution that allows customers to optimize the processes of the entire value chain. It effectively handles both incoming and outgoing goods and guarantees store-friendly deliveries across multiple store formats.

A MLC consists of a fixed set of Functional Modules (FMs). As it is defined in [15], The FMs are the building blocks that can be used in one or more MLCs or custom system solutions. Each FM provides a piece of integrated functionality that directly adds value to the customer. In addition, FMs have clear responsibilities with maximum decoupling between them, so that they can be easily reused in different system solutions.

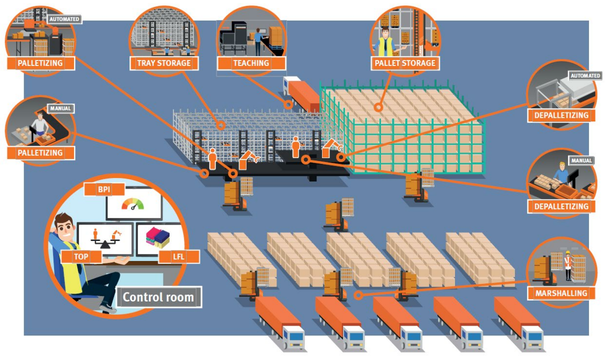
In the scope of this project, a FM is a small building block of the system software that controls functionally equivalent physical components in the system. It can be configured by removing some unnecessary FMs or modifying parameters of the FM based on a customer's need. Besides, FMs can be found in the form of hardware-and-software or only software.



a. AIRPICK



b. FASTPICK



c. STOREPICK

Figure 12 – Market Leading Concepts [12] [13] [14]

Figure 13 outlines the new architecture of the Warehouse system. The same architecture holds for the other systems. The system consists of three layers. The Enterprise domain determines what the Warehouse needs to do in terms of customer order fulfillment and inbound goods receipts. The Process domain fulfills the inbound and outbound orders from the Enterprise domain by controlling the process flow of stock and packages through the warehouse. The Material Handling Domain (MHD) is then responsible for the actual physical movements of the items or a Material Handling Unit (MHU)in the warehouse. A MHU is an item such as baggage in Airport, products in Warehouse, and packages in Parcel and Postal systems.

Each layer has multiple FMs, each with a distinct responsibility. The FMs are the building blocks that can be used in one or more Market Leading Concepts or custom system solutions. Each FM provides a piece of integrated functionality that directly adds value to the customer. In addition, FMs have clear responsibilities with maximum decoupling between them, so that they can be easily reused in different system solutions.

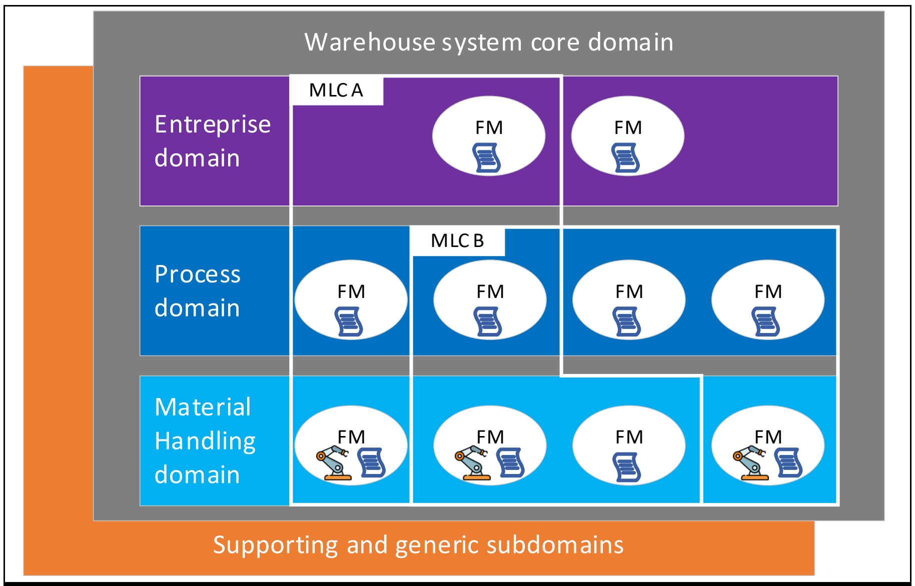


Figure 13 – New architecture of Warehouse system

System architects of the company identified several FMs. The defined FMs in MHD are MHU Tracker, Clearing, Upstream Executing MHD FM, Downstream Executing MHU FM, Empty Carrier Broker, and Sort & Transport FM.

The scope of the project was determined to specify the Sort & Transport FM within the MHD. The reason why this FM is chosen is that it is the most fundamental and core FM in the whole system. Put it differently, all MLCs have this FM to transport and sort MHU within the system. Once this FM is specified more in detail, architecture and design can be developed further. As seen in Figure 13, the FMs of the MHD are in the form of hardware and software but the other two domains have FMs in the form of only software. Therefore, the project is multidisciplinary despite the fact that the project aims to provide more about the logic of the software part in the system architecture. In the future, the project can be extended with other multidisciplinary FMs. ■

3. Project Requirements

In this chapter, the requirements of the project are identified and analyzed under the categories of Functional and non-Functional requirements. The Functional view of the CAFCR framework describes what the system is expected to achieve. It includes use-case and requirements analysis.

3.1 Use-case analysis

In this section, use case analysis is performed by examining the example of the most basic logistics automation system setup, illustrated in Figure 11. User goals are defined as a result of inspecting the capability and functionality of the example system. The overall user goal is depicted in Figure 14. In addition to the inspection of the running example, we looked into the company's system functions. A system user has several goals that can be achieved through system functions that are essential for all systems to handle materials. These functions are Transport, Accumulate, Output, Induct, Merge, Divert, and Identify. The system user can be a customer of the company or an operator who controls the system. Another user goal is the reconfiguration of the system by a company engineer. The reconfiguration takes place when customers of the system want to modify their system by either adding and/or removing existing system components.

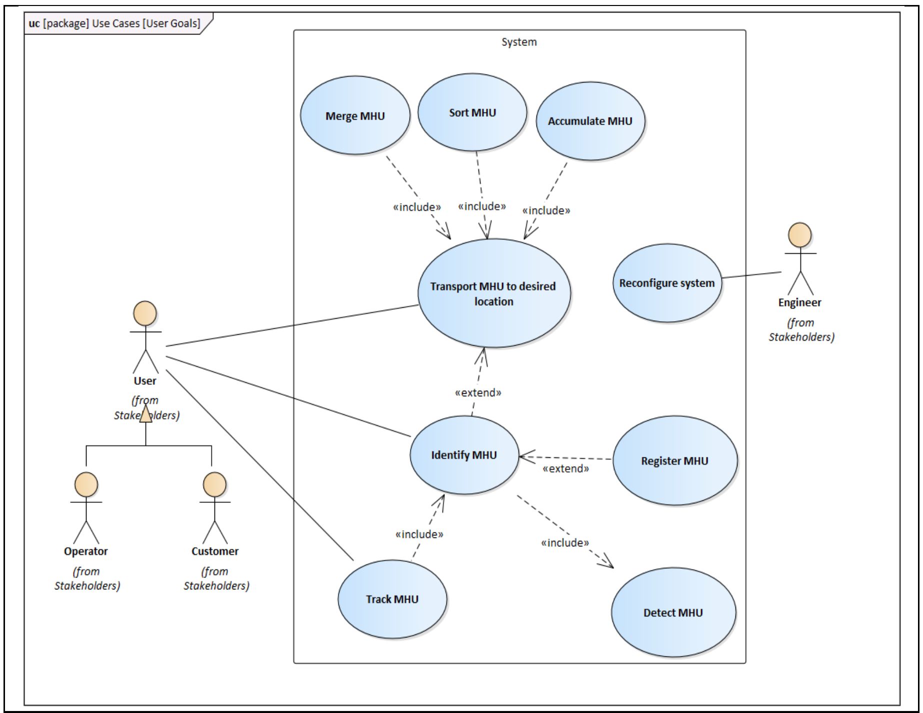


Figure 14 – User goals

The most fundamental user goal is *Transport MHU to the desired location* using the system. This user goal can be extended with identification to show additional functionality; thereby the user can benefit more efficiency and accuracy to reach the goal. By identifying the MHU, the user can physically allocate where this MHU is in the system.

This is the user goal *Detect MHU*. *Track MHU* is a user goal to allocate the MHU virtually within the system. When the identity of the MHU is not found in the system, the MHU has to be registered afterward. This user goal is *Register MHU*.

The other user goals, namely *Merge MHU*, *Sort MHU*, and *Accumulate MHU*, are not possible without the transport use case. Therefore, the relationship between the user goals to the goal of "*Transport MHU to the desired location*" is <<include>> [16].

In Table 1, the *Transport MHU* use case scenario is broken into the scenario steps. The assumption that the conveyor system was turned on is easily satisfied in the real world.

Table 1 – Transport MHU to desired location scenario

Precondition	Conveyor is on.
Actor	**Action**
User	User loads a MHU to the system.
System	Conveyor transports the MHU to the designated location.
User	The user unloads the MHU.

This plane transportation scenario can be extended with identification to save energy. In this case, when a MHU is detected by the system conveyor, which then starts transporting the MHU to the desired location. The scenario is explained in Table 2.

Table 2 – Transport MHU to the desired location with identification

Precondition	Conveyor is off.
Actor	**Action**
User	User loads a MHU to the system.
System	Sensor identifies the MHU.
System	Conveyor transports MHU to the designated location.
System	Sensor at the desired location identifies MHU to check.
User	The user unloads the MHU.

From the use case analysis, the majority of the functional system requirements are derived. These requirements are discussed in the following section.

3.2 Requirement analysis

The Functional view of the CAFCR framework, which is illustrated in Figure 5 defines project requirements. The Functional view explains what the system is expected to accomplish in the scope of the project. In this view, it is often suggested to consider the system as a black box. What features the black box offers are stated in the view. Even though the name of the view is Functional, non-Functional requirements are also part of it. In other words, requirement analysis states not only functional features of the system but also how well this system delivers these functional features to the stakeholders in terms of performance, efficiency, and many other qualities. Thus, project requirements are concrete features, conditions, and tasks that have to be completed to give assurance of a successful project.

The main aim of the project was to help Vanderlande with designing a part of their new architecture of the systems using models. The requirement gathering process was conducted based on analyzing the problems and realizing the feasibilities. All these activities were performed by extensively reviewing the new architectural documentation of the systems, meeting domain experts and stakeholders. These processes helped to produce more concrete requirements. Throughout the project, system requirements have been modified iteratively to make the requirements more concrete.

All of the requirements are derived from the use case analysis, reusability guideline, and modeling aspects of the conceptual solution. The guideline and the modeling were explained in Section 3.1, 2.2.2- I and II respectively. Overall derivation of the requirements and dependency can be seen from Figure 15. Traces between user goals and requirements are shown in the Appendix chapter.

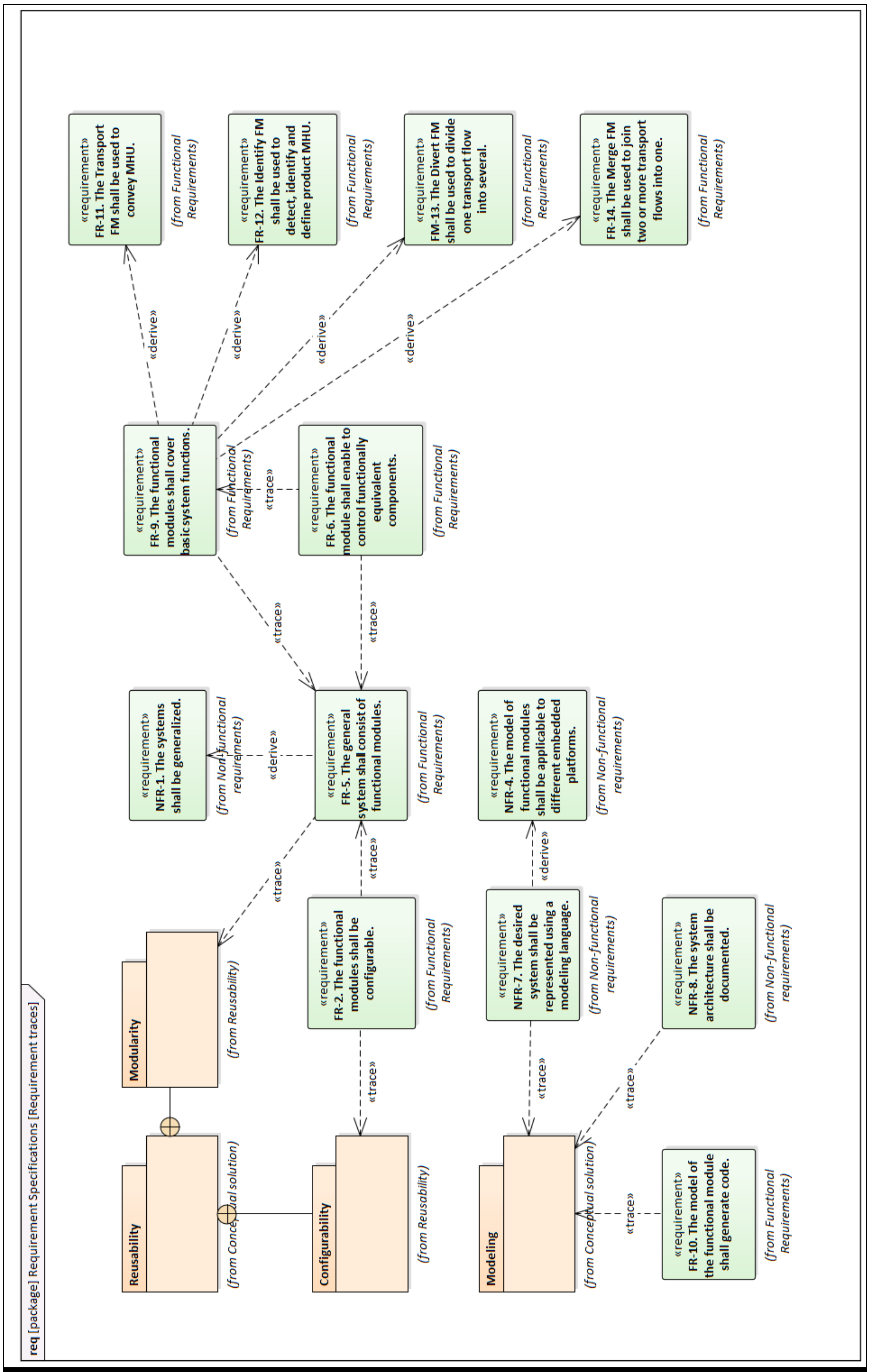

req [package] Requirement Specifications [Requirement traces]
Reusability
(from Conceptual solution)
Modularity
(from Reusability)
Configurability
(from Reusability)
Modeling
(from Conceptual solution)
«requirement» NFR-1. The systems shall be generalized.
(from Non-functional requirements)
«requirement» FR-9. The functional modules shall cover basic system functions.
(from Functional Requirements)
«requirement» FR-6. The functional module shall enable to control functionally equivalent components.
(from Functional Requirements)
«requirement» FR-5. The general system shall consist of functional modules.
(from Functional Requirements)
«requirement» FR-2. The functional modules shall be configurable.
(from Functional Requirements)
«requirement» NFR-4. The model of functional modules shall be applicable to different embedded platforms.
(from Non-functional requirements)
«requirement» NFR-7. The desired system shall be represented using a modeling language.
(from Non-functional requirements)
«requirement» NFR-8. The system architecture shall be documented.
(from Non-functional requirements)
«requirement» FR-10. The model of the functional module shall generate code.
(from Functional Requirements)
«requirement» FR-11. The Transport FM shall be used to convey MHU.
(from Functional Requirements)
«requirement» FR-12. The Identify FM shall be used to detect, identify and define product MHU.
(from Functional Requirements)
«requirement» FM-13. The Divert FM shall be used to divide one transport flow into several.
(from Functional Requirements)
«requirement» FR-14. The Merge FM shall be used to join two or more transport flows into one.
(from Functional Requirements)
«derive»
«derive»
«derive»
«derive»
«trace»
«trace»
«trace»
«derive»
«trace»
«trace»
«derive»
«trace»
«trace»
«trace»
«trace»

Figure 15 – Requirement traces and derivation

The gathered requirements are listed in Table 3. In the table, functional requirements are abbreviated as FR and non-functional requirements as NFR. These categories are integrated with requirement numbers to generate unique identification of the requirements. We defined the list of requirements with prioritization indicator keys: Must, Should, and Optional.
Must: This key indicates a requirement with the highest priority. The requirements that fall in this category have to be satisfied to deliver the core concept of the project. All requirements that are derived from the system functions and the most fundamental user goals fall in this priority category.
Should: The requirements that fall in this category must partially be satisfied or explained because the main concept of the project partially depends on these requirements. If the requirement is not satisfied fully, the remainder can be left for future work.
Optional: This key indicates a requirement with the lowest priority. The requirements that belong to this category add minor value to the project. Thus, these requirements are considered but may not be satisfied because of time constraints.

Table 3 – Project system requirements

ID	Description	Priority
NFR-1	The systems shall be generalized.	Should
FR-2	The Functional Modules shall be configurable.	Must
NFR-3	The design of Functional Modules shall be implemented to show that the recommended architecture is correct.	Optional
NFR-4	The model of the Functional Modules shall be applicable to different embedded platforms.	Should
FR-5	The general system shall consist of Functional Modules.	Must
FR-6	The Functional Module shall enable to control functionally equivalent components.	Must
NFR-7	The desired system shall be represented using a modeling language.	Should
NFR-8	The system architecture shall be documented.	Must
FR-9	The Functional Modules should cover basic system functions.	Must
FR-10	The model of the Functional Module shall generate code.	Optional
FR-11	The Transport FM shall be used to move MHU.	Must
FR-12	The Sense FM shall be used to detect, identify and define product MHU.	Must
FR-13	The Divert FM shall be used to divide one transport flow into several.	Must
FR-14	The Merge FM shall be used to join two or more transport flows into one.	Must
FR-15	The suggested SW architecture should cover the following system functional concepts: Energy-save, die-back, gap control, handover.	Should

As a result of modularizing the system, Vanderlande would benefit from the generalization of the systems. This requirement is stated in NFR-1 and is derived from FR-5. How the systems should be modularized is linked to the requirement FR-9.
As it was described in the MLC and FM based architecture of the company, all other requirements except FR-9 and its derived requirements were explained briefly in Section 2.2.3.

During the requirement elicitation activity, we identified FR-9 that is derived from the use case analysis in which user goals are achieved through the system functions. Another requirement we had to integrate FR-9 with was FR-6. The modules that we design should control functionally equivalent components. These functions, namely

Transport, Accumulate, Output, and Induct, are appeared to be controlled by the same components delivering the functionality of transport with different time configurations. Hence, we derived FR-11 for these similar system functions. All other derived requirements were originated from the other system functions. Regarding FR-2, the configurability was explained in Section 2.2.2-I-ii.

All other requirements that are relevant to high-level requirement or aspect of modeling are derived from Section 2.2.2-II. For the reason of traceability, the self-documentation and code generation, the aspect is preferred over a document-based approach. Traceability means that all the decisions and derivations of system design should be clear and documented. Every design, model, and diagrams express its idea without extensive explanation in the documentation. Besides, code generation for the design is an advantage for validation.■

4. Architectural decisions

In this chapter, system requirements are realized by system architectural decisions. In the CAFCR framework, the Conceptual view describes how the product is depicted at a high-level design. In addition, modeling approach selection and design decisions are explained in this chapter.

System Architecture

In the previous chapter, we defined the list of system requirements considering the system as a black box in the Functional view. In the Conceptual and Realization views, we open the black box to specify the system more in detail. Thus, the Conceptual and Realization views are often considered as white box and these views show how functionality requirements are realized with what components in the system. To be more specific, the Conceptual view captures the concepts behind the design and this view is relatively more stable than the Realization view. In this view, reusable and modular building blocks of the desired architecture are defined. The requirements that were derived from reusability guidelines are the main input to the architecture. The architecture provides direction to the design and to its implementation.

4.1 Modeling approach selection

It was a requirement from the company to conduct the project by using a modeling technique in order to improve the efficiency of the development. This requirement is captured in NFR-7. We analyzed modeling methodologies that are the most suitable for the nature of the project. There are several model-related methodologies that we considered in this project. These methodologies are Model-Based Design (MBD), Model-Driven Engineering (MDE), and Model-Based System Engineering (MBSE). Engineers from different fields apply these methodologies for different purposes but there is common ground in which these approaches converge. This common ground is bounded in the scope of the MBSE approach. The explanation of the model-related approaches is summarized in Figure 16, where a methodology explanation is written in blue and popular tools are highlighted in orange.

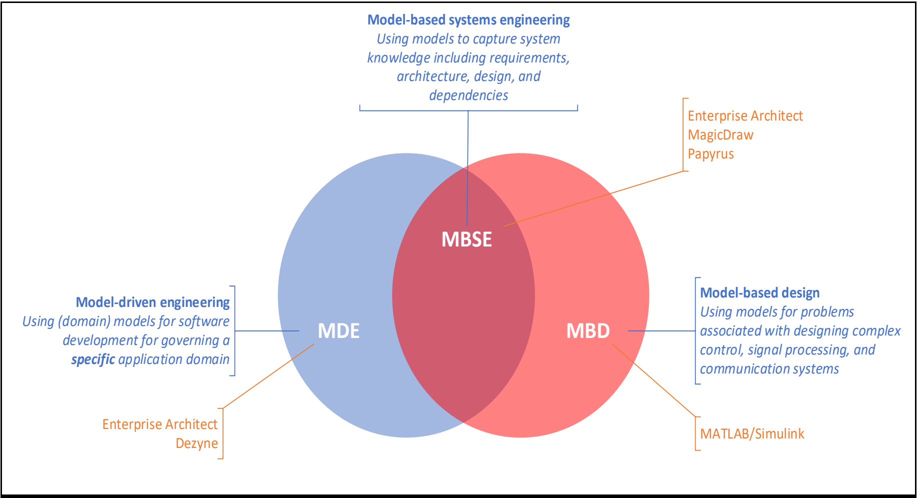

Model-based systems engineering
Using models to capture system knowledge including requirements, architecture, design, and dependencies
Enterprise Architect
MagicDraw
Papyrus
MBSE
Model-driven engineering
Using (domain) models for software development for governing a specific application domain
MDE
MBD
Model-based design
Using models for problems associated with designing complex control, signal processing, and communication systems
Enterprise Architect
Dezyne
MATLAB/Simulink

Figure 16 – Tools and explanation of model-related methodologies

In general, MDE methodology is utilized to provide domain models for a particular software application. For this methodology, software engineers commonly use Unified Modeling Language (UML) to define the design and architecture of the software applications. In the company, these software applications are discussed mostly in higher levels of the Control pyramid in the industrial automation domain.

On the other hand, MBD is utilized more commonly for the purpose of designing control systems. The control systems refer to the lower levels of the Control pyramid, which were explained in Section 2.2.1. In the company, engineers model a plant as well as a controller for the plant using Matlab Simulink in most cases. Simulink simulation helps to verify and validate plant controller behavior before it is deployed on the hardware.

MBD and MDE converge in the MBSE scope. MBSE defines conceptual architecture, design, and specification with rational requirements for the system rather than only software. It also emphasizes specification rather than implementation. The output of the MBSE life cycle can be used as the implementation guideline for MBD and MDE.

As a result of comparing methodologies' benefits for this project trait, MBSE is selected. The most important reason was that architecture, which is based on Market Leading Concept and FM is relatively new in the company and it was not concrete enough to proceed either with MBD or MDE. Because the project scope, which was defined in Section 2.2.3, can be mapped to the lower level of the Control pyramid, MBD was seriously considered as a modeling methodology to conduct the project. However, MBSE is preferred over MBD because MBD lacks the capability to define architecture. In addition, MBD is mostly facilitated for application-specific solutions, meaning that one MBD solution may not be reused in other solutions. This was contradicting the project's one of the main goals to generalize family-products and components that control functionally equivalent equipment.

Furthermore, we considered modeling language for the project. The most common modeling language in MBSE is SysML. SysML fits the project for several reasons.

Firstly, we defined that the scope of the project is Material Handling Domain, which is a set of multidisciplinary FMs that are responsible for moving physical items in the system. Since the FMs of MHD are multidisciplinary, the FMs are a combination of software components, electro-mechanical components, and mechanical components. SysML is suitable for the project because the language is expressive to show the multidisciplinary design. Even though our aim was modeling mostly software logics for FMs, we considered the future work extension of the project.

Secondly, the architecture, which is based on Market Leading Concept and FM, is relatively new in the company. Hence, we analyzed that Vanderlande is deficient in the architecture specification. SysML has the potential to provide clear architecture specifications in a multidisciplinary setup.

SysML is a general-purpose, graphical modeling language for system engineering applications [17]. It is extended on the basis of UML. On the one hand, UML was designed to be standard for the software engineering domain. On the other hand, SysML is designed to support the analysis, specification, design, verification, and validation of complex systems that may include hardware, software, data components. Figure 17 shows an overview of what diagrams SysML offers and what diagrams are common in both UML and SysML to provide a detailed specification of the system architecture.

From the SysML diagrams, requirement analysis and use case analysis are performed using requirement and use case diagrams respectively. In the design and validation, block definition diagram (bdd), internal block diagram (ibd), activity diagram, and sequence diagrams are utilized to give a more detailed specification of the suggested architecture.

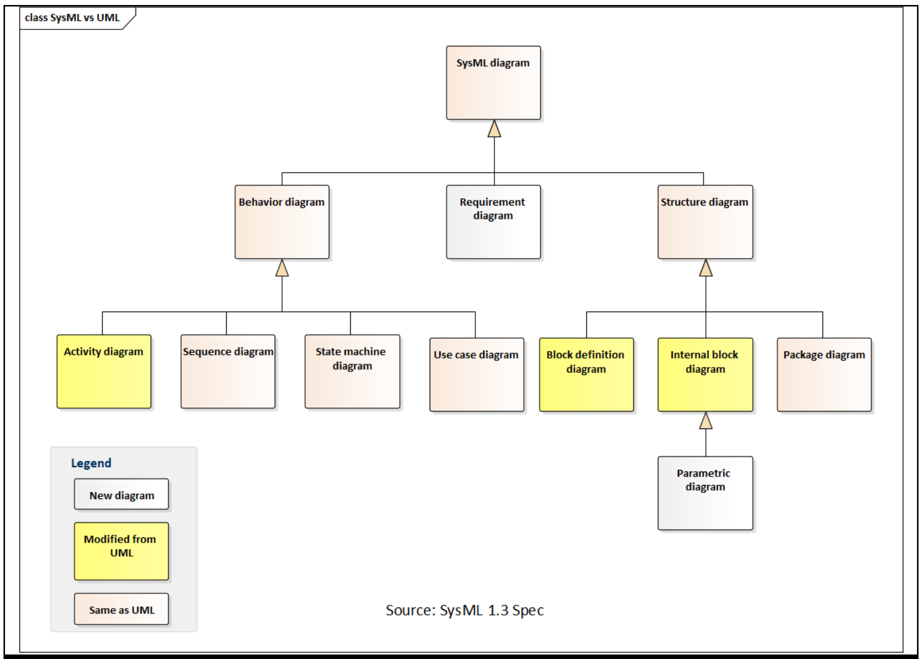


Figure 17 – SysML diagrams

Thirdly, another important trait that the SysML fits the project is the fact that industrial standards support the object-oriented concept. These standards and how they support the object-oriented programming (OOP) concept were explained in Section 2.2.2-II. SysML is an essential modeling language for the OOP paradigm where modularity is a feature.

4.2 Pattern selection

The aim of the system architecture is to give a conceptual guideline for system design. As it is mentioned in the CAFCR framework, the Conceptual view of the system drives the Realization view. In the opposite direction, the Realization view supports the Conceptual view.

Building a modular system was the most important requirement in the project. To be specific, software that controls the same system components in terms of system function had to be modularized as one building block in the system. Furthermore, large system tasks had to be divided into small subtasks so the corresponding components take the subtasks and execute their responsible system function. In this sense, we decided to build an architecture of a distributed computing software system.

Another factor why we suggest the distributed computing system architecture was an analogy of the human body. Every part of the body has a corresponding responsibility to perform subtasks. When a human is given to carry out a task, the brain subconsciously divides the task into small subtasks that should be performed by the corresponding part of the body. For instance, a person is given a task to take an item from a table and put it on another table. The brain decomposes the task into small subtasks. Firstly, eyes are dedicated to locating the item where it is on the table. When the person locates the item, he approaches the item by walking. The foot is dedicated to approaching to the item. A hand is utilized to carry the item and to place it on another table. In this example, all the body parts have their own responsibility to execute these distributed subtasks. In a similar way, a system can carry out the large task by decomposing it into a sequence of subtasks and distributing the subtasks to the software modules. In our case, these modules are Functional Modules.

A software expert's conventional way of solving problems is to apply or reuse already proven solutions to similar problems. This is practical and useful because the experts know the solution worked in similar problems from experience. In software engineering, this solution is known as a pattern. The solution can be a pattern or collection of patterns. Therefore, we also considered software architectural and design patterns in the project. The selection of the appropriate pattern procedure includes the following six steps and the procedure [10] and the procedure is also shown in Figure 18.

Step 1. Specify the problem:
Before finding the appropriate patterns for our particular problem, we needed to specify the problem precisely. This step refers to Section 2.1: Problem Analysis. As software complexity increases, we need more modular software architecture. This architecture is used as a reference template to build different software applications or customer-specific complex software with less effort. Therefore, we seek a solution with modularity and (re)configurability in terms of functionality and topology, respectively.

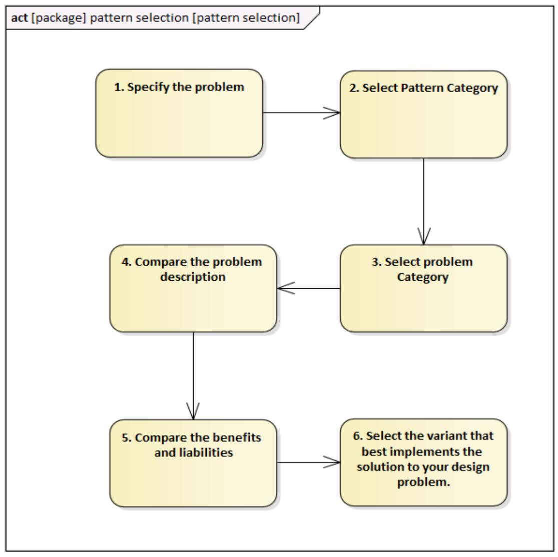

act [package] pattern selection [pattern selection]
1. Specify the problem
2. Select Pattern Category
3. Select problem Category
4. Compare the problem description
5. Compare the benefits and liabilities
6. Select the variant that best implements the solution to your design problem.

Figure 18 – Pattern selection procedure

Step 2. Select pattern category:
As it is stated in [10], there are three pattern categories: architectural patterns, design patterns, and idioms. An architectural pattern provides a high level, fundamental organization schema for software design. It specifies a set of subsystems, responsibilities of the subsystems, and the relationship between them. A design pattern expresses a scheme for specifying further insight of the subsystems, components of the subsystems, and the relationship between the components. An idiom defines how to implement a specific aspect of components or relationships between them using a particular feature of a programming language.

In the scope of the project, we aimed to contribute more on design rather than architecture and implementation because the system architecture was already defined by the system architect team of Vanderlande. The architecture, which is based on MLC and FMs, is explained in Section 2.2.3. Therefore, design pattern is the pattern category that we are looking for.

Step 3. Select the problem category:
The problem category outlines types of problems in general, not considering minor details. Each problem category has a pattern that addresses the corresponding problem differently. Well-known problem categories are listed in the first column of Table 4.

For the modularity aspect, which was specified in FR-5, FR-6, and FR-9, we considered the problem category of Distributed system, Structural decomposition and Organization of work. We did not consider the other problem categories because they emphasize to solve problems for different aspects than **modularity** and thereby, reusability. For instance, the interactive systems category highlights solving problems, which occur in the interaction between users and systems.

Besides, the Adaptable system problem category was also considered for the (re)configurability aspect. However, our main concern was the design pattern of modular systems rather than the architectural pattern as we mentioned in step 2. The high-level system architecture is outlined in Figure 13. This project's purpose is to specify this architecture in more detail with the design proposal of MHD Functional Modules. We focused more on how to modularize the Vanderlande system software with regards to system functions. Therefore, we decided to examine 'structural decomposition' and 'organization of work' problem categories further. These categories give design patterns that we are looking for.

Table 4 – Pattern classification schema

Problem category	Architectural patterns	Design patterns	Idioms
From Mud to Structure	Layers Pipes and Filters Blackboard		
Distributed systems	Broker Pipes and Filters Microkernel		
Interactive systems	MVC PAC		
Adaptable systems	Microkernel Reflection		
Structural decomposition		Whole-Part	
Organization of work		Master-Slave	
Access control		Proxy	
Management		Command Processor View Handler	
Communication		Publisher-Subscriber Forwarder-Receiver Client-Dispatcher-Server	
Resource Handling			Counted Pointer

Step 4. Compare the problem description:
In the selected problem category, each pattern addresses a specific part of our problem. In this step, we try to find the appropriate pattern that addresses the modularity aspect of our problem. Our main goal is to split a task into subtasks so FMs execute corresponding subtasks. The design patterns of 'structural decomposition' and 'organization of work' problem categories are Whole-Part and Master-Slave patterns, respectively.

The Whole-Part design pattern, Figure 19a, helps to execute a task by dividing a component of the system into multiple dividend components that perform the task uniformly. In other words, the aggregate component, the Whole, allows a user to interact with smaller components, Parts, and composition of components uniformly. Direct interaction between a user and a smaller component is not possible. For instance, when a user calls service1 from the Whole, corresponding services, namely serviceA1 and serviceN1, will be called and executed by dividend components PartA and PartN respectively. ServiceA1 and serviceN1 are versions of service1 of PartA and PartN respectively.

On the other hand, the Master-slave design pattern, Figure 19b, divides work into identical subtasks that are further processed by individual slaves. The master splits a task into subtasks and delegates them to the corresponding slaves. The whole service is calculated using the results from each slave.

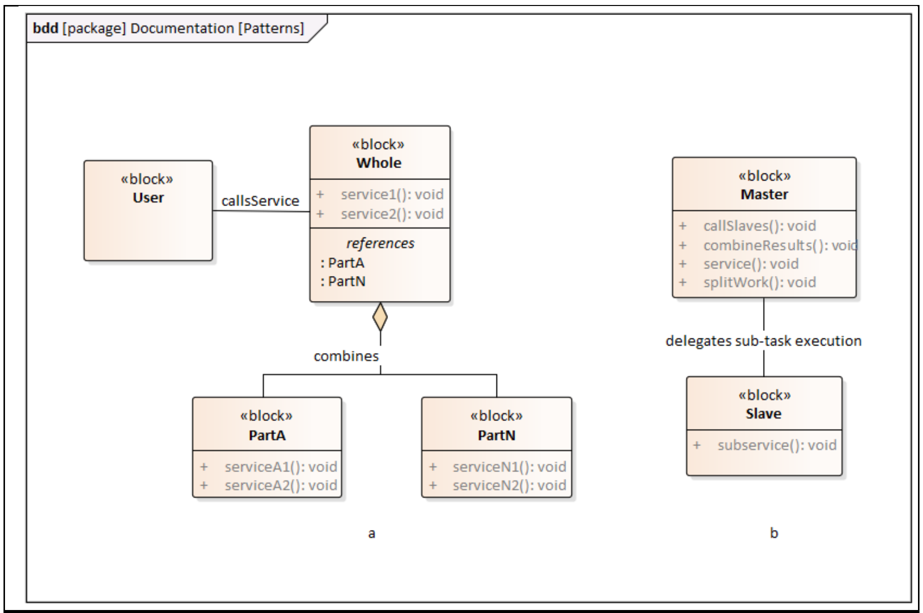


Figure 19 – a. Whole-part b. Master-slave design pattern

From the design pattern descriptions above, we can conclude that Master-slave is more satisfactory than the Whole-part design pattern for our problem description. The main reason why Master-slave is preferred over Whole-part pattern is that Whole-part emphasizes component decomposition rather than service decomposition. However, in the Master-slave pattern, service can be decomposed into sub-services and Master coordinates slaves to execute their own distinct sub-services. These sub-services are performed by different FMs, which are explained with FR-9 and its derived requirements FR-11, FR-12, FR-13, and FM-14.

Step 5 Compare the benefits and liabilities:
In this step, selected patterns are compared with their advantages. Patterns that have more advantages to solve the problem and whose liabilities are of least concern are selected in this step. The benefits and liabilities of considered two patterns are outlined in Table 5.

Table 5 – Benefits and liabilities of Master-slave and Whole-part patterns

Design patterns	Benefit	Liability
Master-slave	*Extensibility and exchangeability* If we introduce abstract Slave, it will be easy to extend the software with new slaves without changing Master.	*Machine dependency* The master-slave pattern strongly depends on the architecture of the machine on which the software is deployed.
	Separation of concern Master splits big tasks into subtask so slaves perform distinct subtask individually.	*Hard to implement* It is often hard to implement the Master-slave pattern because of parallel computing.
	Efficiency The master-slave pattern supports parallel computation which makes the software more efficient.	*Portability* Because of machine dependency, it is difficult to transfer software that was running in one machine to another machine.

	Reusability This pattern allows the reusability of slaves in different applications of software.	
	Distinct subtasks by slaves Master treats slaves differently depending on distinct subtasks. Also master chooses which slave should be executed in what order.	
Whole-part	*Changeability of parts* Parts are modifiable without influencing other Parts and a client in the implementation because Whole encapsulates Parts thereby it conceals Parts from the client.	*The complexity of decomposition of parts* The appropriate composition of Whole from Parts is difficult especially when the bottom-up approach is chosen.
	Separation of concern It is easy to implement complex tasks with this pattern because the complex task can be divided into simple task thereby Parts implement these subtasks. Each part has its own concern of subtask.	*Lower efficiency through indirection* Since the client cannot access Parts directly, it introduces inefficiency. This may cause additional runtime compared to monolithic software.
	Reusability The pattern supports two kinds of reusability aspects. The first is that Parts can be reused in other aggregate objects. The second is that since Whole encapsulates Parts, it restricts the client to create Parts all over the software. Thus, the reusability of Whole is considered in this case.	
	Uniform execution by Parts When a client calls a service, Whole calls corresponding subservices from Parts uniformly which is efficient when Whole is complex.	

Step 6 Select the variant that best implements the solution to your design problem. The most important aspect we considered in this project is modularity. Modularity always comes along with aspects of separation of concern and reusability. Both of the considered patterns have these benefits. Both of the patterns are also appropriate for (re)configuration of the software. It would be efficient if the (re)configuration is performed from the central component of the software. In some researches, [18] and [19], it is also recommended to use Master-Slave Pattern or centralized control pattern for the (re)configuration.

Although both of the patterns have common traits that fit the reusability of modules and reusability with configuration, Master-Slave pattern fits more to the design because there is a special characteristic that the system should deliver. The characteristic is that systems functions are different. Thus, the components that are performing these functions should be treated differently in the system. In the whole-part pattern, these functionalities are supposed to be executed uniformly meaning that these functionalities are different versions of one service. On the other hand, the Master-slave pattern can make this distinction of functionality with different slaves. Master treats slaves differently. Considering these characteristics, Master-slave is preferred over Whole-part. ■

5. System Design

In this chapter, the selected design pattern, Master-slave, is realized using SysML diagrams to show the behavior and the structure of the proposed design.

System design

The (low-level) system design is depicted in the Realization view of the CAFCR framework. The architectural concepts that are described in the Conceptual view (considered the high-level design) give abstract guidelines to the design process and change very slowly across generations of products. For each generation these concepts are detailed and realized, possibly in different ways and using more recent technology, in the Realization view (considered the low-level design), addressing the particular problem of the project. Hence, as opposed to the Conceptual view, the Realization view can evolve very fast via frequent changes made to the design, for example, in order to catch up with new requirements of customers and with new technologies.

5.1 Structural model

The block definition diagram and the internal block diagram of SysML are utilized to show the hierarchy of the proposed design. We identified Transport, Sense, Merge and Divert FMs that control functionally equivalent corresponding equipment of the system. These FMs are entities that execute a system function and are derived from the most fundamental operations in the system. From the Master-slave design pattern perspective, the FMs are slaves. The way the FMs interact with each other should be orchestrated by another entity that has the master role in the pattern. Therefore, we identified another entity, Material Handling Domain Manager, as the master. The overall responsibilities of the FMs and other MHD entities are outlined in Table 6.

Table 6 – Responsibilities of MHD entities

Entity	Responsibility explanation
MHD Manager	MHD Manager has the following responsibilities as a Master. • To orchestrate FMs. This is the most important responsibility of the MHD Manager. • To obtain MHU information from Sense FM through Report interface. Based on this information, the MHD Manager decides which FM to execute next. • To command executing FMs, namely Transport FM, Divert FM, and Merge FM, to perform their tasks through the Command interface. • To retrieve MHU destination or routing information from the Process domain and to incorporate it with the MHU database. • To create, read, update, and delete the MHU database. • To map layout information to FMs. When hardware infrastructure reconfiguration takes

	place, new hardware infrastructure information is retrieved through the Configuration interface. This information is a map of system layout including each equipment's start position, stop position, upstream and downstream device.
Transport FM	The Transport FM controls all kinds of transporting equipment in the system. The equipment includes carrier transport and various conveyors, namely roller conveyor, belt conveyor, and wheel conveyor. The AGV transport is out of scope in this project but in the future, architectural and design solutions should be suggested as an extension of Transport FM. From the system perspective, the above-mentioned various equipment performs the same in terms of functionality.
Sense FM	The Sense FM controls all kinds of sensor equipment. These various pieces of equipment include barcode scanners, photoelectric cells (PEC), weighing scales, and dimensional measurement sensors. From the system perspective, these devices have the same functionality to determine MHU in many different ways by defining the location, weight, dimension, and orientation of MHU.
Divert FM	The Divert FM controls all the kinds of equipment that divide one transport flow into several.
Merge FM	The Merge FM controls all kinds of equipment that join two or more transport flows into one.
MHU	This is a database that keeps track of all attributes of every physical MHU in the system.

The main idea of the design that conforms to the Master-slave pattern is that a big task of transporting a physical MHU throughout the whole system infrastructure can be achieved by splitting the task into small subtasks. The small subtasks are executed by corresponding FMs. The overall hierarchical system design that combines the above-mentioned responsibilities of FMs with the Master-slave design pattern is depicted in Figure 20.

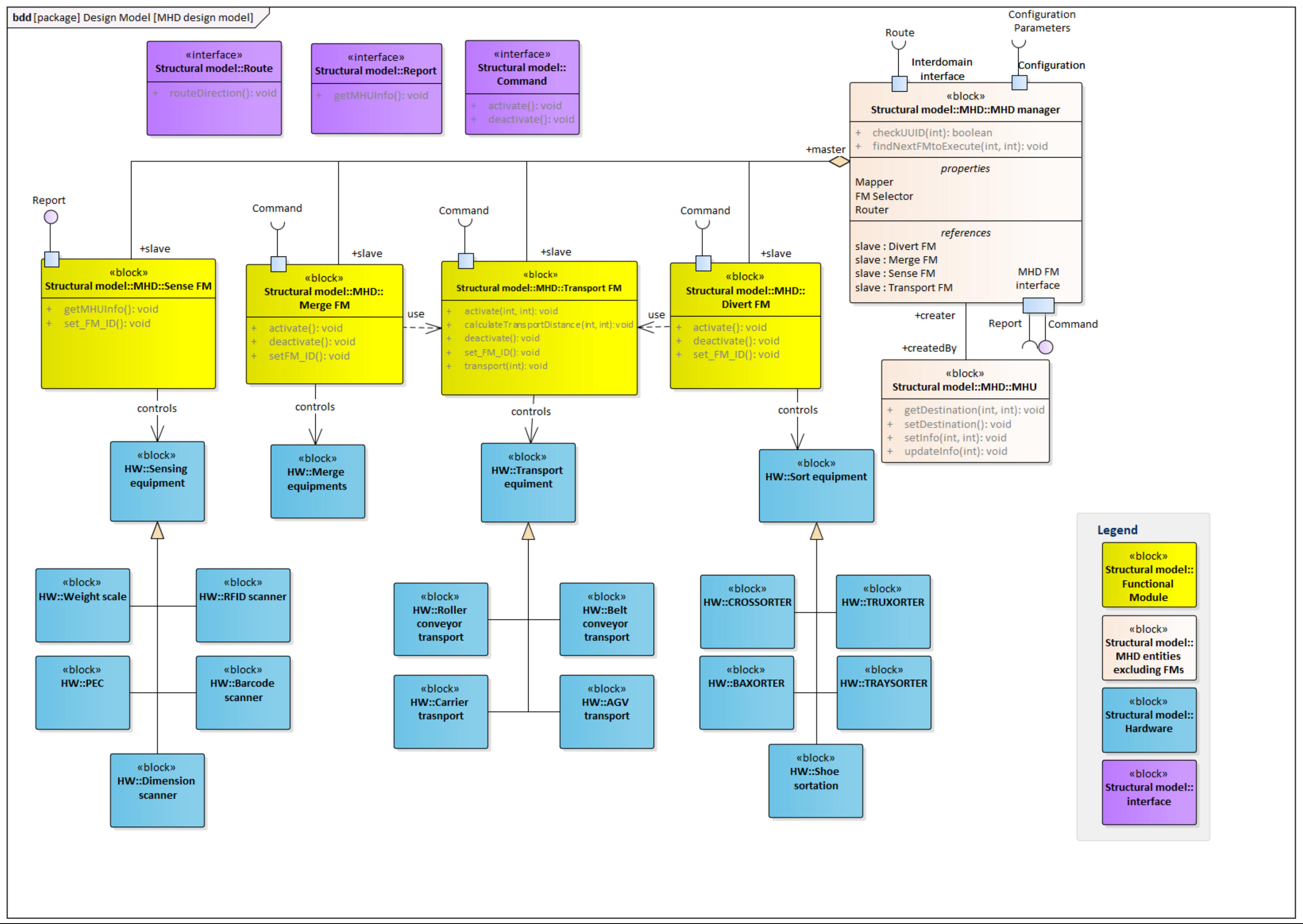

bdd [package] Design Model [MHD design model]
«interface» Structural model::Route
+ routeDirection(): void
«interface» Structural model::Report
+ getMHUInfo(): void
«interface» Structural model:: Command
+ activate(): void
+ deactivate(): void
Route
Interdomain interface
Configuration Parameters
Configuration
«block» Structural model::MHD::MHD manager
+ checkUUID(int): boolean
+ findNextFMtoExecute(int, int): void
properties
Mapper
FM Selector
Router
references
slave : Divert FM
slave : Merge FM
slave : Sense FM
slave : Transport FM
MHD FM interface
+master
+slave
Report
Command
«block» Structural model::MHD::Sense FM
+ getMHUInfo(): void
+ set_FM_ID(): void
«block» Structural model::MHD:: Merge FM
+ activate(): void
+ deactivate(): void
+ setFM_ID(): void
use
«block» Structural model::MHD::Transport FM
+ activate(int, int): void
+ calculateTransportDistance(int, int): void
+ deactivate(): void
+ set_FM_ID(): void
+ transport(int): void
«block» Structural model::MHD:: Divert FM
+ activate(): void
+ deactivate(): void
+ set_FM_ID(): void
+creater
+createdBy
«block» Structural model::MHD::MHU
+ getDestination(int, int): void
+ setDestination(): void
+ setInfo(int, int): void
+ updateInfo(int): void
controls
«block» HW::Sensing equipment
«block» HW::Merge equipments
«block» HW::Transport equiment
«block» HW::Sort equipment
«block» HW::Weight scale
«block» HW::RFID scanner
«block» HW::PEC
«block» HW::Barcode scanner
«block» HW::Dimension scanner
«block» HW::Roller conveyor transport
«block» HW::Belt conveyor transport
«block» HW::Carrier trasnport
«block» HW::AGV transport
«block» HW::CROSSORTER
«block» HW::TRUXORTER
«block» HW::BAXORTER
«block» HW::TRAYSORTER
«block» HW::Shoe sortation
Legend
«block» Structural model:: Functional Module
«block» Structural model:: MHD entities excluding FMs
«block» Structural model:: Hardware
«block» Structural model:: interface

Figure 20 – Structure of the proposed design

To specify more details of the MHD Manager responsibility, an internal block diagram is given in Figure 21 where the property blocks and interfaces are shown. Here, a property block is a part of the MHD Manager that executes corresponding calculations and operations. Each property block executes a task, which refers to the different responsibilities of the MHD Manager.

One of the high priority project requirements is to specify the reconfiguration mechanism of the system. The Mapper property block of the MHU Manager plays an important role for reconfiguration. Layout information of the reconfiguration process is retrieved by Mapper through the Configuration interface. This information includes a number of each FM as a parameter, their starting positions, end positions, and IDs of upstream and downstream devices. In short, this information is a map of system layout. Mapper allocates these parameters to each instance of FMs. All FMs have set_FM_ID operation, which is used to configure unique identifiers to all FMs. When the reconfiguration is performed, the MHU Manager knows all FMs with their unique identification. This identification defines the execution order of coinciding physical components of FMs.

Another responsibility of the MHD Manager is to provide a routing destination to corresponding FMs. This responsibility is crucial for sorting MHUs. The routing path or destination of a MHU is determined by the Process domain. This information is requested and retrieved by the MHD Manager through the Interdomain interface. Property block Router processes directional information and passes the result to the FM Selector. The Mapper property block maps whole system components to their corresponding FMs. By cooperating with Mapper, FM Selector then assigns which FM should be executed in order to guide the MHU to its destination.

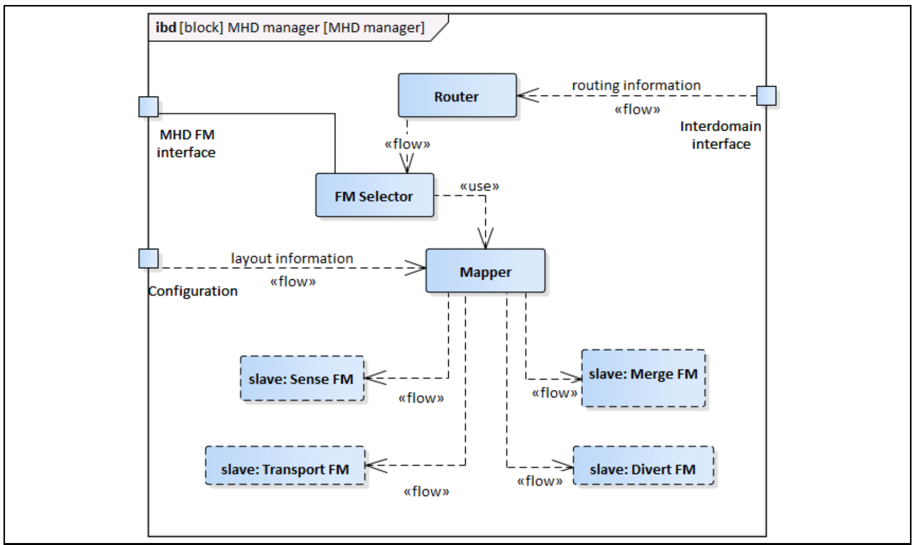


Figure 21 – Internal block diagram of MHD Manager

5.2 *Behavioral model*

The sequence diagrams and activity diagrams are utilized to show the behavior of the proposed design of the system. We chose a typical system layout to show interactions between MHD FMs and the other MHD entities.

Scenario 1 - Transporting one MHU on one conveyor belt:
This scenario is depicted in Figure 22. Scanner1 is an instance of a Sense FM and it controls the Barcode Scanner component in the layout shown in Figure 22a. Similarly, a Transport FM called ’TFM1’, a MHD Manager called ‘Manager’, and a MHU database entity called ‘MHU1’ are also instantiated in the sequence diagram given in Figure 22b, showing the interactions between these entities. TFM1 controls the ‘Transport

1’ conveyor. MHU1 stores data of the physical MHUs. Note that in this case there is only one MHU. The MHD Manager instance ‘Manager’ orchestrates the two FMs. Step by step explanations of the interactions, Figure 22b, are listed below.

- The task of conveying the physical MHU from the beginning of the conveyor to the end is triggered by an interrupt of physical MHU. This interrupt is firstly received and handled by Scanner1.
- Scanner1 acquires the unique identification (UUID) and the location of the physical MHU and passes it to the MHD Manager.
- The MHD Manager retrieves the MHU destination from the MHU1 database block. It is assumed that MHU1 was already created by the MHD Manager when the physical MHU enters the system the first time. In other words, the MHD Manager already registered the MHU by creating MHU1 and set its destination afterward.
- Another assumption is that MHU updates its current location virtually with the assistance of Track FM. The Track FM is responsible to virtually estimate the location of physical MHU within the system. Track FM is not specified in the report because it is out of scope in this project. However, this current location estimation is corrected by the updateInfo() operation.

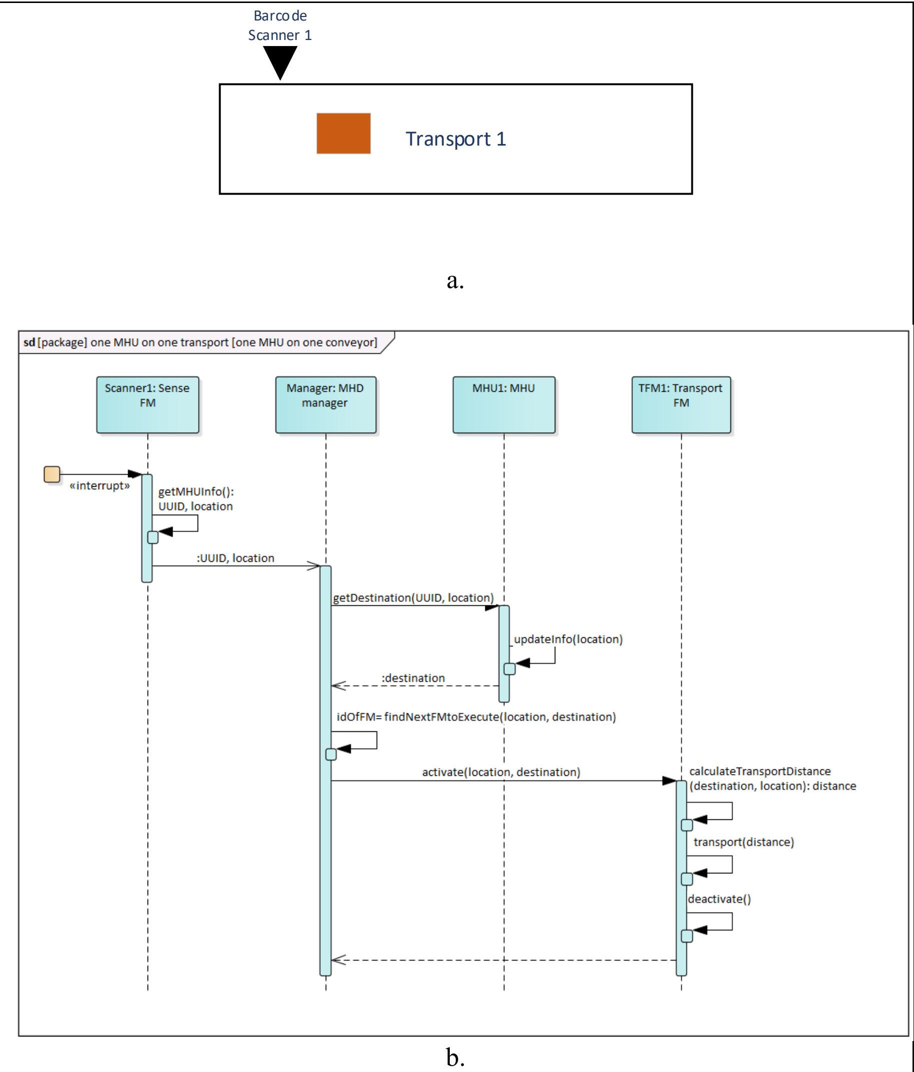

Barcode Scanner 1
Transport 1
a.
sd [package] one MHU on one transport [one MHU on one conveyor]
Scanner1: Sense FM
Manager: MHD manager
MHU1: MHU
TFM1: Transport FM
«interrupt»
getMHUInfo(): UUID, location
:UUID, location
getDestination(UUID, location)
updateInfo(location)
:destination
idOfFM= findNextFMtoExecute(location, destination)
activate(location, destination)
calculateTransportDistance (destination, location): distance
transport(distance)
deactivate()
b.

Figure 22 – a. System layout b. Corresponding sequence diagram of one MHU being transported on one conveyor

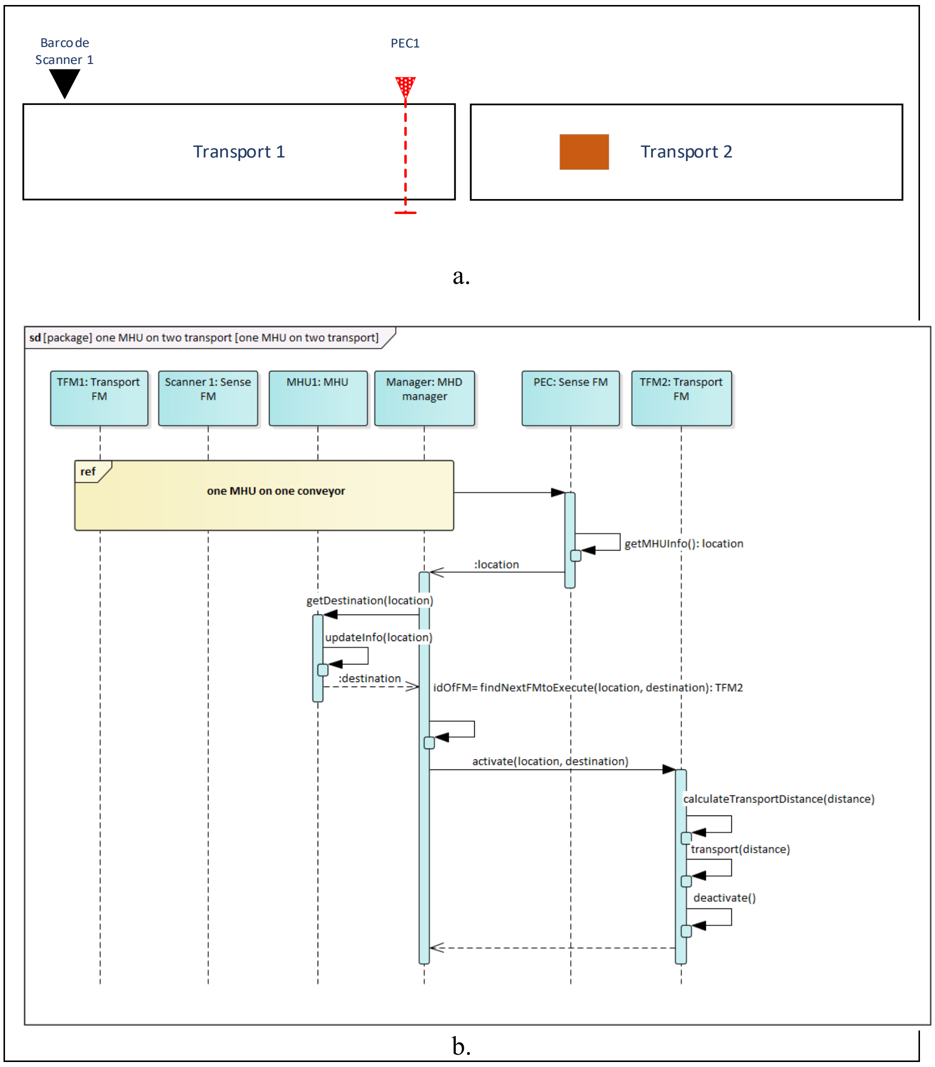
- findNextFMtoExecute() operation is dedicated to finding which FM instance to execute next. The decision is based on the current location and destination of MHU. Thus, the operation takes these parameters as inputs.

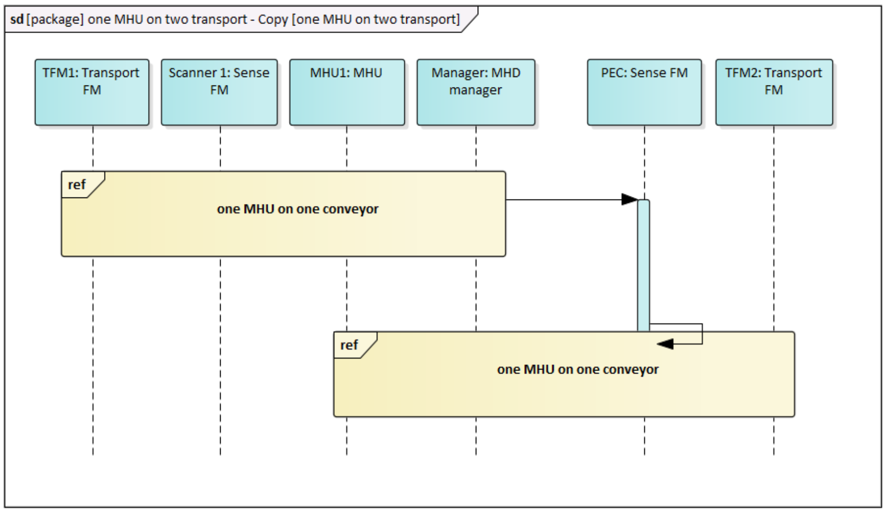
- When the MHD manager knows which FM to execute, it activates the corresponding FM instance. In our scenario, this is TFM1.
- TFM1 calculates how many meters of distance it should run. Afterward, it transports MHU for the calculated distance. In the end, TFM1 deactivates itself.

Scenario 2 - Transporting one MHU on two conveyor belts:
Figure 23a shows an additional layout where one more conveyor belt 'Transport 2' and one Photo Electric Cell (PEC) called PEC1 (an instance of the Sense FM) are added to the layout of Figure 22a. A PEC is a light reflective sensor that is used for detecting MHUs on the conveyor belt.

In the corresponding sequence diagram, Figure 23b, the same interactions between MHD FMs and other MHD entities, as it was illustrated in Figure 22b, take place for the first conveyor belt part. In order to avoid repetition, these interactions are shown as a yellow reference diagram block in the diagram, Figure 23b, while the additional interactions of the MHD Manager are shown explicitly.

In terms of modularity and reusability of FMs, interactions between instances of FMs and the MHD Manager are identical in Figure 22b and Figure 23b, meaning that the same FMs are reusable in both layouts. In other words, the same reference interaction diagram can be reused twice as depicted in Figure 23c.



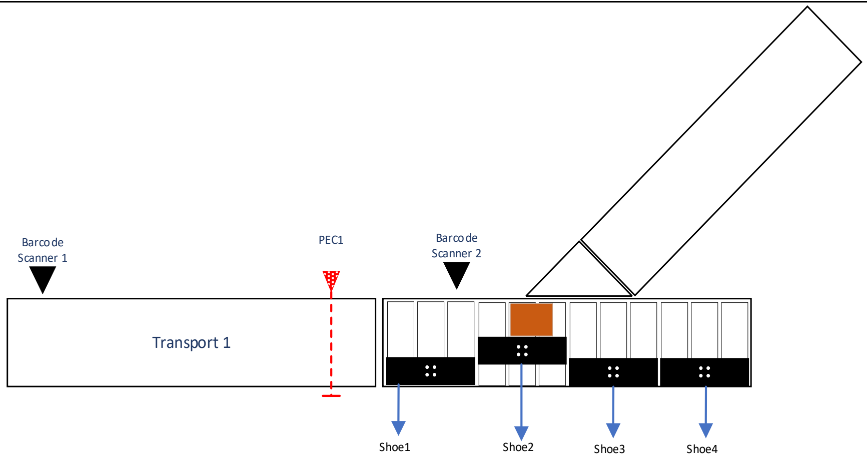


c.

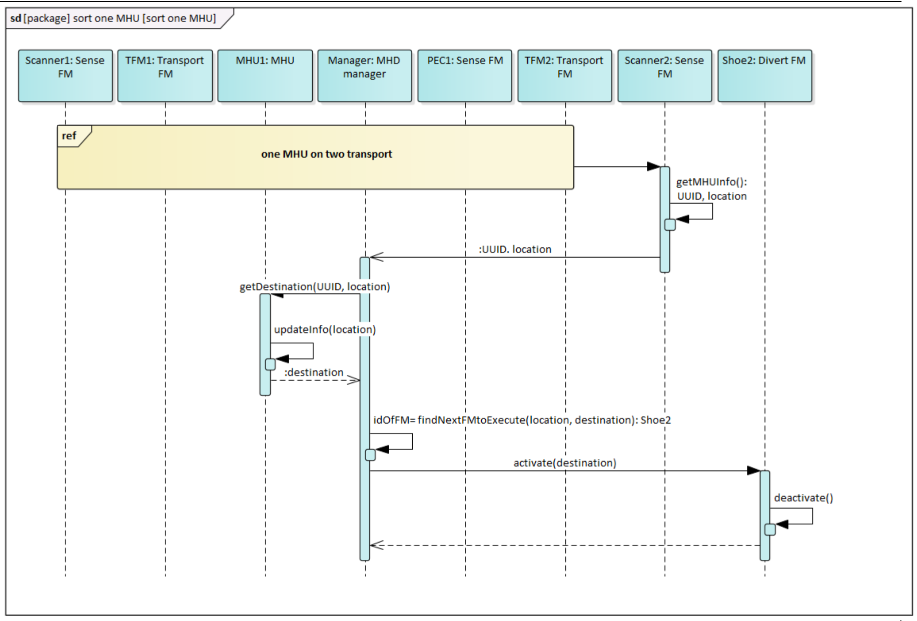
Figure 23 – a. System layout b. Corresponding sequence diagram of one MHU being transported on two conveyor belts c. Representation of interactions reusability

Scenario 3 - Sorting:
The sorting scenario is given with system layout and interactions between FMs and other MHD entities in Figure 24. Figure 24a shows additional system components where four diverting devices 'Shoe 1-4' and one barcode scanner called 'Barcode Scanner 2' are added to the layout of Figure 23aFigure 22.

The corresponding sequence diagram is based on the interactions of Scenario 2. Therefore, we also reused the sequence diagram of Scenario 2 as a reference diagram block in Figure 24b. Since components, namely Barcode Scanner 2 and Shoe 1-4, are added to the system layout, the corresponding FMs and interactions between FMs and other MHD entities are shown in Figure 24the diagram. The scanner is controlled by Scanner2 (instance of Sense FM) and shoes are controlled by Shoe1-4 (instances of Divert FM). However, only the interaction of Shoe2 is depicted in the diagram because the scenario shows the sorting of only one physical MHU. This is realized by the operation of findNextFMtoExecute in which the Manager delegates the diverting task to the Shoe2. This operation takes location and destination of MHU as inputs. At the end of the scenario, the Manager activates Shoe2 to divert the physical MHU.



a.



b.

Figure 24 – Sorting scenario layout and interactions

In this chapter, we showed how the Master-slave pattern can be utilized in the modular system software. The design concept is explained with the structural and behavioral models. On one hand, the structural model captures FMs and other MHD entities with their relations. On the other hand, behavioral models showed the interaction between them by taking three fundamental scenarios of the system. Because the scenarios present typical system functionalities of transporting, sensing, and sorting, any modular system with these functionalities can reference to the proposed Master-slave design in order to implement it.

6. Verification and Validation

This chapter presents how verification and validation processes were carried out for the design. Several approaches, namely model reviewing, sequence diagram simulation, and conceptual simulation, were applied for this stage of the project. As a validation, we facilitated a conceptual simulation that shows how the system should behave with regards to the modularity and configurability, and analyzed its behavior.

6.1 Verification

Verification ensures the proposed design models are correctly built. The correctness of the design was verified by inspecting and reviewing structural and behavioral models of the proposed design with the support of domain experts, supervisors, and Enterprise Architect simulation execution.

Both the requirements and system design were verified together with project stakeholders on a weekly basis. In doing this, behavioral diagrams namely, sequence and activity diagrams, were reviewed and inspected. During the inspection, the correctness of the behavioral logic was examined. Besides, the consistency of operations and corresponding parameters in MHD entities and FMs is kept both in the behavioral diagrams and structural diagrams.

Throughout the project, we used Enterprise Architect (EA) to design the modular system software for its advantage of availability to the trainee and traceability from requirements to the design. In addition to the availability and traceability, EA provides sequence and activity diagram simulation functionality. The simulation helps to examine use-case scenarios by going through a series of message exchanges between instances of MHD entities and FMs. This examination is executed step by step in both sequence and activity diagrams.

6.2 Validation

Validation ensures the built models satisfy the project requirement. Our validation of the design was carried out through the implementation of animated conceptual simulation, analyzing its behavior and checking the correctness of the behavioral design models by comparing them with the functional requirements of the customer.

6.2.1. Simulation Environment

As a simulation tool, Anylogic was utilized for its advantage of the short learning curve and built-in libraries, especially the Material Handling library. The tool helped to visualize the running systems that are modeled using proposed design concepts. We constructed the simulation setup, seen in Figure 25, using the built-in blocks of Anylogic [20]. For validation, it is necessary to construct the simulation with the conceptual FMs of the proposed design. Therefore, we facilitated Anylogic blocks as conceptual FMs of the proposed design. In the simulation, there are two system constructional panes:

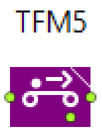
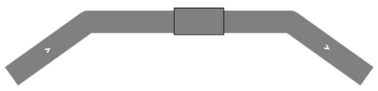
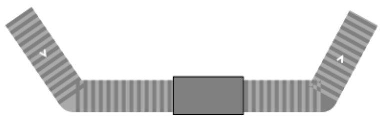
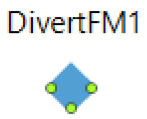
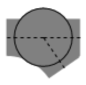
- One is a representation of the system topology in which we built whole hardware infrastructure using Anylogic's space markup elements including Conveyor, Turntable, and Position on conveyor.

- The other constructional section of the simulation is system logic. System logic is built using Anylogic's Material Handling library blocks such as Convey, Select Output, Source, and Sink.

In the simulation, every Material Handling library block controls the corresponding equipment in the system infrastructure. In the system logic pane, the purple-colored blocks are the Convey blocks that control the corresponding conveyor in the system infrastructure pane. These Convey blocks are conceptual representations of the Transport FM, named TFMx. With the Transport FM, we can control various transporting devices. As an example of these various transporting devices, roller conveyors and belt conveyors are shown in the simulation. For instance, the TFM5 block controls the purple-colored conveyor path in the system infrastructure. Similarly, the diamond-shaped blocks are the Select Output blocks that control the corresponding Turntable to sort physical MHUs. These blocks are conceptual representations of Divert FM of our proposed design. As a conceptual representation of the Sense FM, a space markup element 'Position on conveyor' is shown in the system infrastructure pane, which is integrated as a piece of code in the Select Output block.

In the simulation, it is assumed that the MHD manager puts all these FMs in order and gives commands to the FMs. Thus, MHD manager behavior is integrated with other FMs behavior in the simulation. Table 7 shows the overall set of logic blocks and the corresponding infrastructure elements as representations of Functional Modules in the simulation environment.

Table 7 – Explanation of FM representations in the simulation

Functional Modules	Logic representation in the simulation	Infrastructure representation in the simulation
Transport Functional Module	Convey block:  Convey block is a logical representation of the TFM and it controls different kinds of transporting devices such as belt conveyor and roller conveyor in the system infrastructure of the simulation.	Belt conveyor:  Roller conveyor:  These conveyors are simulation representations of physical conveyors.
Divert Functional Module	Select Output block:  Select Output block is a logical representation of the Divert FM and it controls various sorting devices. In the simulation it controls turntable.	Turntable:  Turntable is a simulation representation of physical sorting devices in the system.
Sense Functional Module	Integrated with the Select Output block as a code	Position on conveyor:

The simulation complies with the proposed design. We can see the possibility of simulating modularized template building blocks can be utilized for controlling corresponding equipment in this example simulation. Anylogic also provides 3D illustration of the simulation, which is depicted in Figure 26.

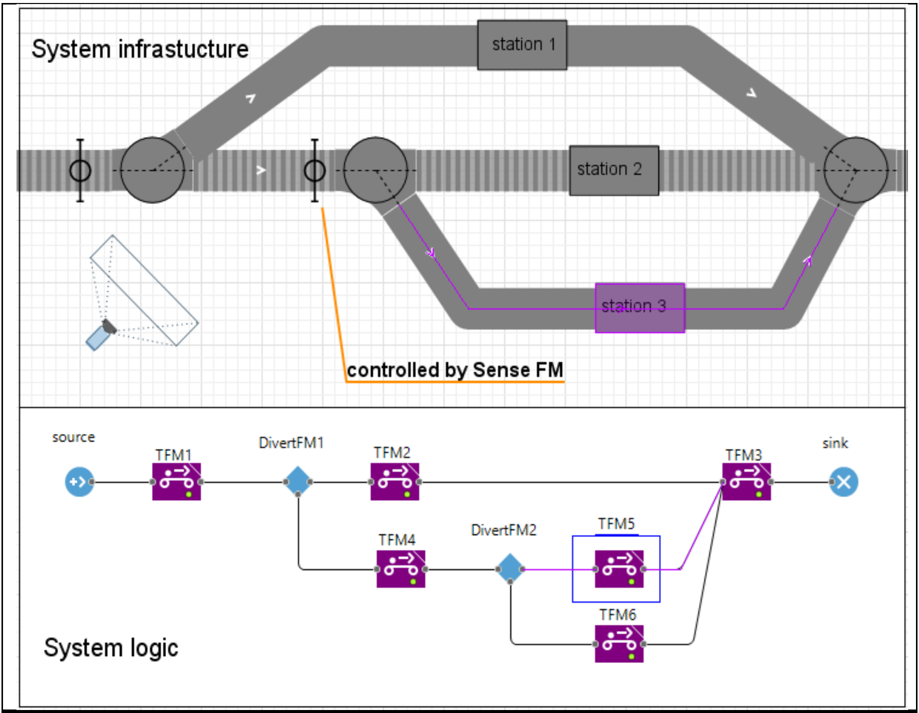


Figure 25 – Example simulation a. System infrastructure b. System logic

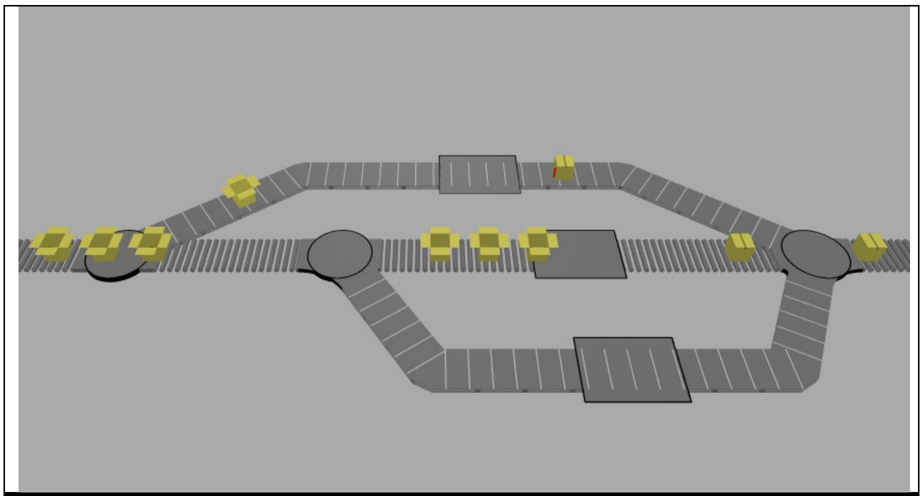


Figure 26 – 3D illustration of the example simulation shows the expected behavior of the system.

The above example simulation of the system is built from modular building blocks. With the simulation, we were able to illustrate the behaviors of Transport FM, Sense FM and Divert FM on the corresponding simulation blocks of *Convey*, *Select Output*, and *Position on conveyor,* respectively. In the example simulation, the expected behavior of the *Convey* block is to transport a MHU to its desired destination. This behavior is noted as A in the corresponding sequence diagram. The behavior of the *Position on conveyor* is to identify MHU – behavior B. With the use of the *Position on conveyor*, the *Select Output* block directs a MHU to the next conveyor on the desired source-to-destination route. For instance, DivertFM1, in Figure 25b, is expected to direct a MHU to a secondary transporting route when it is activated – behavior C. Here, the main route and the secondary route of the conveyor network are represented by the roller conveyor and the belt conveyor, respectively. Additionally, TFM2 controls the main route and TFM4 controls the secondary route.

6.2.2. Validation Process

From the simulation, we could visualize expected system behavior. The expected behavior is validated with actual behavior models of the proposed design, which is presented with a sequence diagram in Figure 27. In the diagram, the above-mentioned behaviors are highlighted in the red (A), orange (B), and green(C) boxes. The behaviors A, B, C are mapped to the FR-11, 12, 13, respectively. The reasoning of the MHU flow division depends on the destination of the MHUs. If the destination of the MHU is station1, DivertFM1 is activated. Afterward, TFM2 controls the secondary route to take the MHU to station 1. On the other hand, if the destination of the MHU is station 2, the system continues transporting the MHU to station 2 via TFM4.

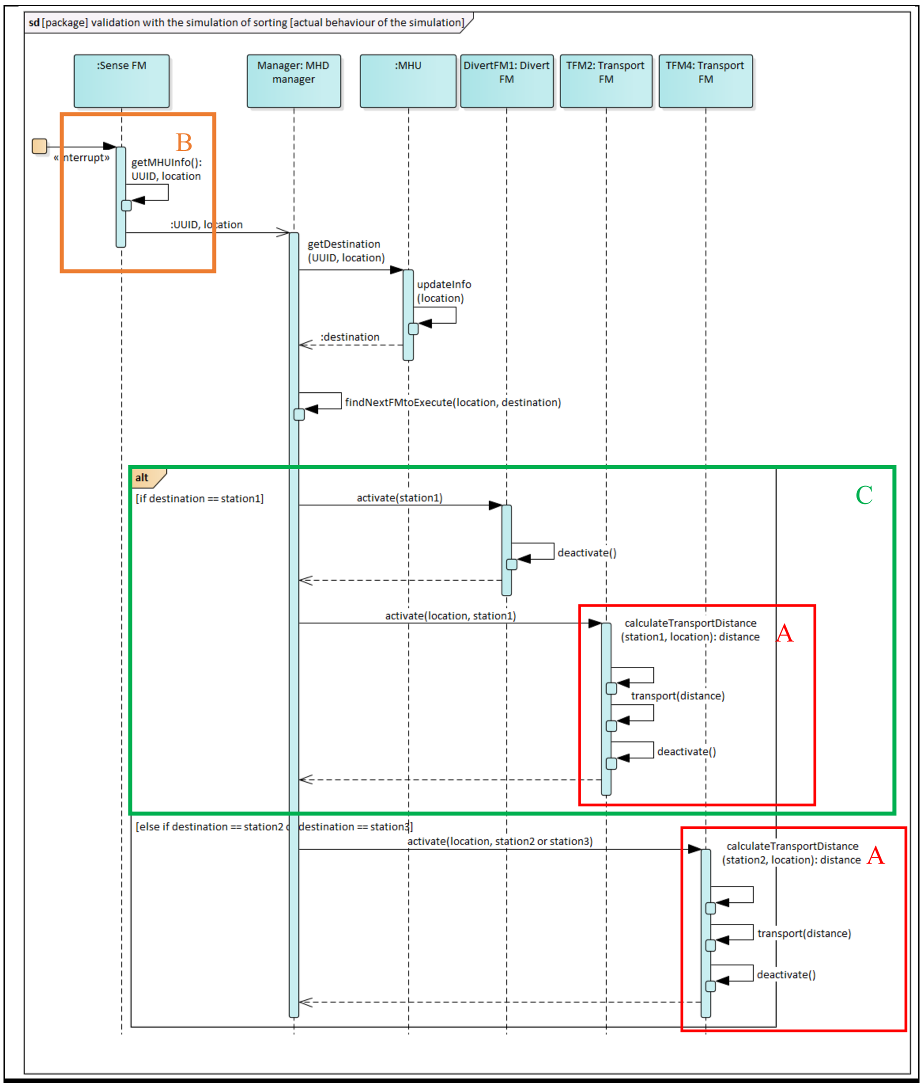
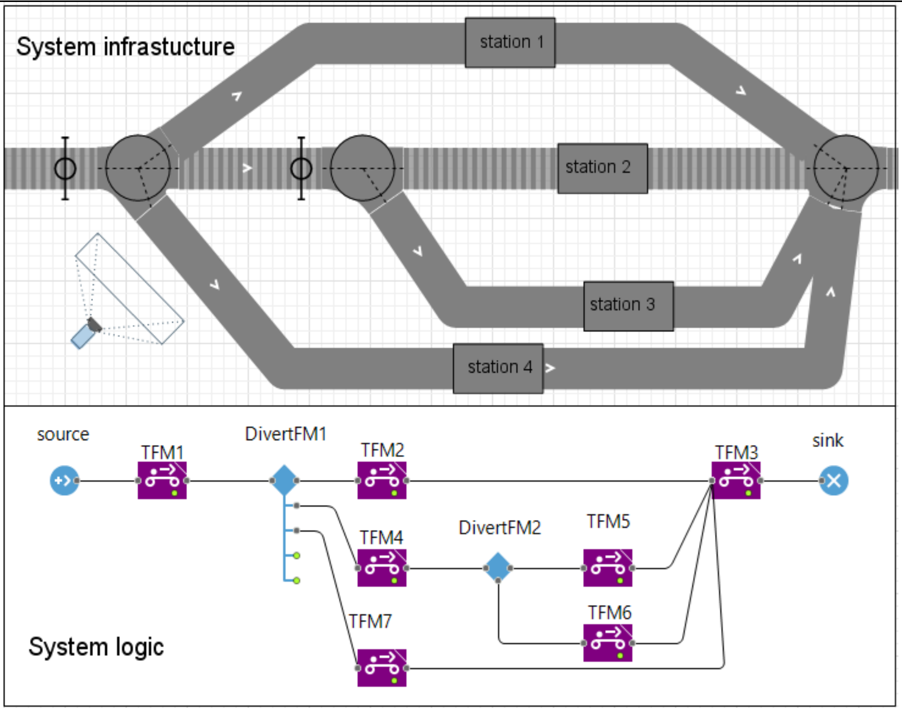

sd [package] validation with the simulation of sorting [actual behaviour of the simulation]
:Sense FM
Manager: MHD manager
:MHU
DivertFM1: Divert FM
TFM2: Transport FM
TFM4: Transport FM
B
«interrupt»
getMHUInfo(): UUID, location
:UUID, location
getDestination (UUID, location)
updateInfo (location)
:destination
findNextFMtoExecute(location, destination)
alt
[if destination == station1]
C
activate(station1)
deactivate()
activate(location, station1)
calculateTransportDistance (station1, location): distance
A
transport(distance)
deactivate()
[else if destination == station2 or destination == station3]
activate(location, station2 or station3)
calculateTransportDistance (station2, location): distance
A
transport(distance)
deactivate()

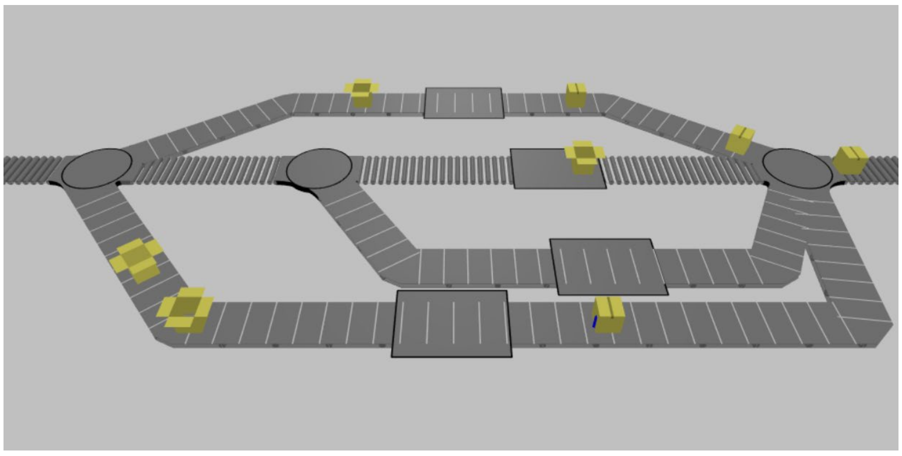
Figure 27 – Actual system behavior of the example simulation

From the simulation, we see the expected behavior of the system. From the sequence diagram, we see the actual behavior of the system. The resemblance of the two behaviors shows the validity of the project design.

For the configurability requirement, FR-2, we also used Anylogic simulation. The simulation for configurable topology is depicted in Figure 28. In order to validate the configurability concept, we need to show that the same logical component of the modular system works for different system layouts. The previous simulation layout, Figure 25a, is extended with an additional sorting path to the first turntable. This turntable is controlled by DivertFM1 in both simulations but in Figure 28, it is reconfigured for the additional sorting path. In short, the same DivertFM1 is used both in Figure 25 and Figure 28. 3D view of the reconfigured simulation is captured in Figure 28b. The more advanced simulation for configurability is shown in Section 10.4.



a.



b.

Figure 28 – Conceptual simulation of reconfigured DivertFM1

The corresponding behavior diagram of the reconfigured DivertFM1 simulation is illustrated in Figure 29. The sequence diagram complies with the proposed design's behavior models. From the figure, we can see that divert functionality operations are identical in two cases of sorting a MHU to station 1 and station 4. This validates the idea that the Divert FM can be reconfigurable with parameters of how many sorting paths its hardware is connected to. In short, Divert FM can be reused based on its configuration.

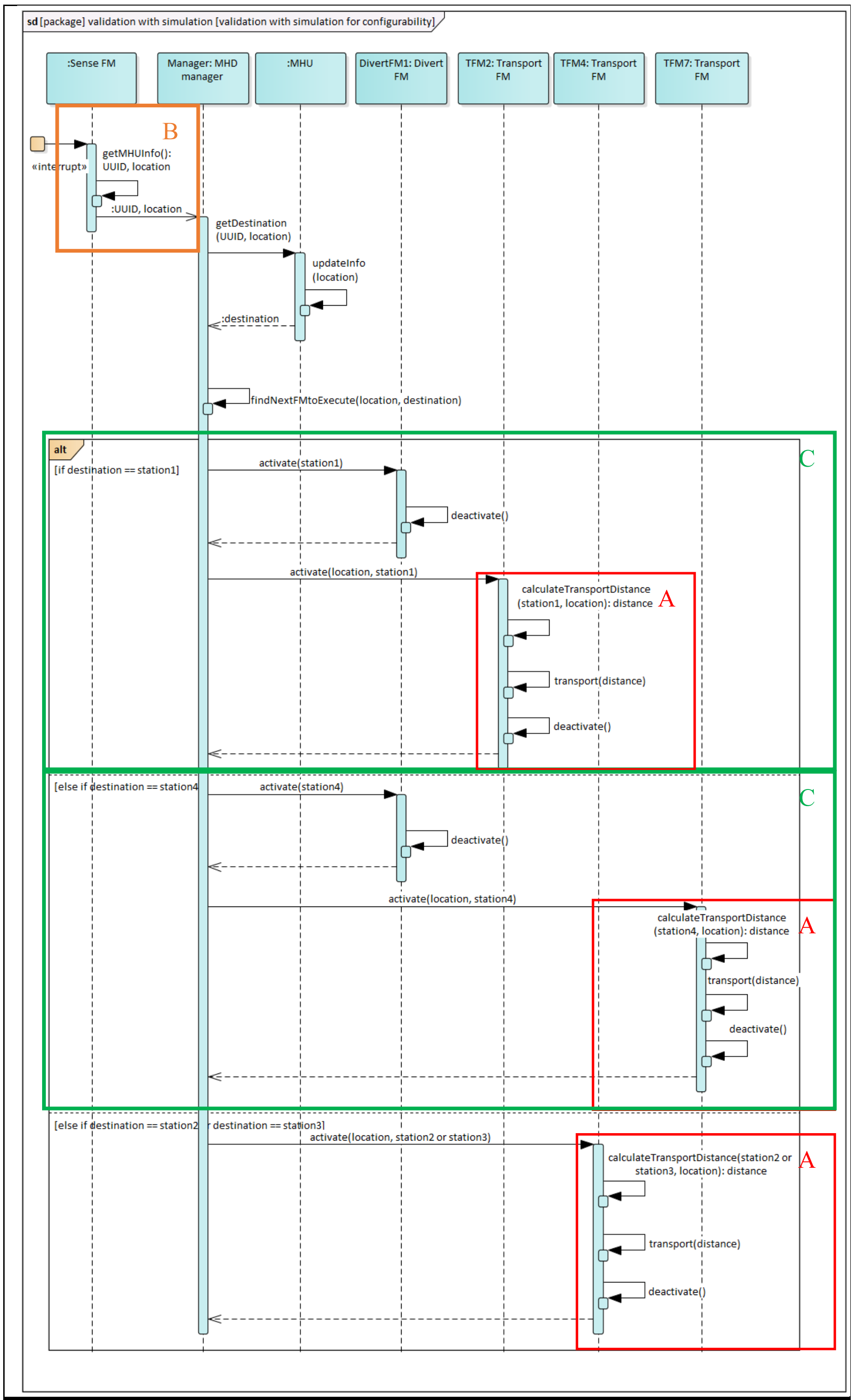

sd [package] validation with simulation [validation with simulation for configurability]
:Sense FM
Manager: MHD manager
:MHU
DivertFM1: Divert FM
TFM2: Transport FM
TFM4: Transport FM
TFM7: Transport FM
B
«interrupt»
getMHUInfo(): UUID, location
:UUID, location
getDestination (UUID, location)
updateInfo (location)
:destination
findNextFMtoExecute(location, destination)
alt
[if destination == station1]
activate(station1)
deactivate()
activate(location, station1)
calculateTransportDistance (station1, location): distance
transport(distance)
deactivate()
A
C
[else if destination == station4]
activate(station4)
deactivate()
activate(location, station4)
calculateTransportDistance (station4, location): distance
transport(distance)
deactivate()
A
C
[else if destination == station2 or destination == station3]
activate(location, station2 or station3)
calculateTransportDistance(station2 or station3, location): distance
transport(distance)
deactivate()
A

Figure 29 – Configurable Divert FM behavior

Considering the simulation, the result of validation in accordance with the requirement is shown in Table 8.

Table 8 – Project requirement validation

Requirement ID	Priority	Status	Explanation
FR-11, FR-12, FR-13	Must	Designed and FRs are validated with conceptual simulation.	These FRs are derived from the FR-9.

FR-2	Must	Designed and validated the concept with simulation.	
FR-5, FR-6, NFR-8, FR-9	Must	Accomplished	FR-9 is partially accomplished because one of the derived requirement, FR-14, is not accomplished due to the time constraint of the project.
NFR-1, NFR-4, NFR-7	Should	Designed	
FR-14, FR-15	Should	Not accomplished	These FRs can be achieved by specifying the proposed design further in the future.
NFR-3, FR-10	Optional	Not accomplished	Optional requirements can be accomplished extending on the project design in the future work.

7. Conclusions

This chapter summarizes the output of the project by overviewing what has been done in order to approach the main goals. The recommendation for the company is followed at the end of the chapter.

7.1 Summary

The project was conducted to investigate modular and configurable system software architecture to address inefficiency in the development of complex system software within the company. The main output of the project is to propose modular and configurable reference architecture that complies with the company's new system architecture. To obtain the desired architecture rationally, we facilitated the CAFCR framework, which gives systematic architectural decision guidelines to the project. The CAFCR views are mapped to the problem analysis, domain analysis, project requirements, architectural decisions, and system design, respectively.

In the problem analysis, we observed frequently emerging above mentioned problems from a customer's point of view. Troubles that the company's customers and software developers encounter regularly is discussed in great detail in the context of the project. In the industrial automation domain, the example of how modularity helps to address above mentioned problems was reviewed with the modern technology of IIoT and CPS. These trend examples inspired conceptual guidelines to realize the full potential of reusable software. In addition to the reusability of the software, modeling increases efficiency. Hence, modeling related domain standards are explained afterward. Furthermore, Chapter 2 explains that the company started realizing the reusability guideline by producing the new system architecture. The scope of the project is defined within this new architecture. In short, crucial questions of the project, namely why, how, what, are analyzed and answered which then enable us to realize the project requirements.

All the analysis and scope of the project provides a direction of requirement elicitation. The elicitation started with analyzing the most fundamental user goals and system functions within the project scope. Moreover, non-functional and functional requirements are defined in Chapter 3. We assigned the higher priority to the requirements which are derived from the system functions because the most fundamental user goals of the system are achieved with these functions. Furthermore, these requirements became references for how to model the systems modularly.

Considering all the requirements, we investigated patterns that satisfy modular and configurable software architecture. All the rationales why we chose the Master-slave pattern is stated in Chapter 4. The pattern clarifies the work organization of the modules. In addition, one of the most important and difficult decisions we made was the modeling approach selection in which we chose MBSE over MBD. We realized that the MBSE approach benefits the company in the long term because it helps to generalize the company systems in the multidisciplinary view. Moreover, we extended the explanation of modeling approach decisions with modeling language that complies with the industrial standards, which were described in the domain analysis.

In Chapter 5, the selected pattern is facilitated in the design process of the MHD, which is the scope of the project. The project's most crucial contribution to the company, MHD design, is given in forms of structural and behavioral models. The structural models give an overview of the software entities and interfaces between them. The behavioral models give detailed interaction between the entities through the interfaces. Due to the time limitations and the lack of domain knowledge, some requirements were not captured in the design. However, the proposed design is definitely extensible because of its genericity.

Furthermore, we explained the verification and conceptual validation process. The validation is performed by comparing the proposed design sequence diagrams with corresponding Anylogic simulations. The validation led to draw the conclusion of how many requirements are met with the proposed design. As has been noted, we proposed a modular reference architecture that we produced based on whole lifecycle processes of analysis, requirements elicitation, architectural decisions, design and validation activities. All activities contributed to the project with fruitful insights.

7.2 Recommendations

- I recommend the company to apply Model-Based System Engineering methodology before implementing a concept with either manual coding or other modeling approaches. MBSE enables system and software developers to capture the concept with high-level architecture. Thanks to its expressive and multidisciplinary modeling language, SysML, it will be easier for developers to implement the architecture and design. This approach also helps to link architects' works to the developers' work.
- I also recommend the company to continue this project because of its value of connection between architect's and developers' work. Future work can include followings:
    - Investigate MBD realization of the proposed design.
    - Continue the proposed design to satisfy unaccomplished requirements.■

8. Project Management

To carry out the project successfully, project management was considered another main factor besides the lifecycle of the model development process in the project. This chapter defines project management relevant activities. The activities include managerial tasks including stakeholder management, project planning for time management, and risk management for mitigation planning of project uncertainty.

8.1 Stakeholder Analysis and Management

This section identifies project stakeholders with their detailed concerns and involvements in the project. Since the project was initiated based on the cooperation of Vanderlande and the Eindhoven University of Technology, the stakeholders belong to the two parties with their distinct concerns.
At the end of the section, the communication plan of the project is explained as a result of the stakeholder matrix.

8.1.1. Stakeholder analysis

In the analysis section, all stakeholders are examined within the criteria of their role, interest, acceptance criteria and involvement to give more insight into their concerns in the project.

To draw an overview of the involvement of the company side, stakeholders influenced the project in terms of requirements, knowledge, and expectation.
This project is a realization of the development process in the system architecting department to adapt to Industry 4.0. Regarding this development process, Vanderlande commenced several projects such as Next Level Automation (NLA). This PDEng design project is a sub-project of the development process. Therefore, stakeholders from the NLA project played an important role in the project to give domain experts' knowledge.

Table 9 – Stakeholder analysis from company side

Stakeholder	Supervisor from Vanderlande
Responsibility	1. Monitoring the project progress 2. Giving feedback on the design 3. Referring the trainee to the domain experts 4. Evaluating the trainee 5. Reviewing the report
Representative	Lennart Swartjes
Interests	1. Model-Based design in systems architecting 2. Promoting Model-Based approach
Acceptance criteria	1. Well thought design 2. Well written report with detailed rationales
Involvement	1. Continuous participation throughout the project 2. Weekly meeting 3. Ad-hoc meeting 4. PSG meeting

Stakeholders	Domain experts in the company
Responsibility	1. Transferring domain knowledge to the trainee 2. Giving updates of subproject progress
Representatives	Marc van Kerkhof, Bart Vorstemans
Interest	Reflection of NLA in Model-Based approach
Acceptance criteria	Well thought design
Involvement	1. Weekly meeting with van Kerkhof while the company supervisor was on his holiday 2. Ad-hoc meeting 3. Skype consulting

From the university's point of view, stakeholders' concerns are related to the project processes that must meet certain academic standards. The processes include project management, design, implementation, verification, and validation.

Table 10 – Stakeholder analysis from the university side

Stakeholder	A supervisor from TU/e
Responsibility	1. Monitoring the project progress 2. Giving feedback on project management and design 3. Guiding the trainee for successful project completion 4. Evaluating the trainee 5. Reviewing the report
Representative	Tanir Ozcelebi
Interests	1. The technical report 2. The smooth progress of the project
Acceptance criteria	1. Report monthly project progress 2. Well-written report that meets the program standards
Involvement	1. PSG meeting 2. Monthly meeting

Stakeholder	The program director from TU/e
Responsibility	1. Provide information and guidance about carrying out the project 2. Evaluate the trainee
Representative	Yanja Dajsuren
Interests	1. Good cooperation between the company and the university 2. Successful completion of the project, thereby successful graduation of the trainee
Acceptance criteria	1. Well-written report that meets the program standards 2. Personal and professional development of the trainee based on evaluation
Involvement	Occasional PSG meetings

Stakeholder	PDEng trainee
Responsibility	1. Managing the project 2. Designing a solution that satisfies requirements 3. Verifying and validating the design 4. Writing the report
Representative	Ganduulga Gankhuyag
Interests	1. Successful completion of the project 2. Learning MBSE 3. Learning the domain 4. Graduation
Acceptance criteria	1. A timely report of the deliverables 2. Sufficient quality of the deliverables
Involvement	Continuous participation throughout the project

8.1.2. Stakeholder management

In order to manage the stakeholders of the project, it is crucial to carry out a stakeholder analysis. In the previous section, the analysis of every stakeholder is shown separately but in detail. In this section, the stakeholders are allocated in the stakeholder analysis map based on their interests and power. The map is depicted in Figure 4.

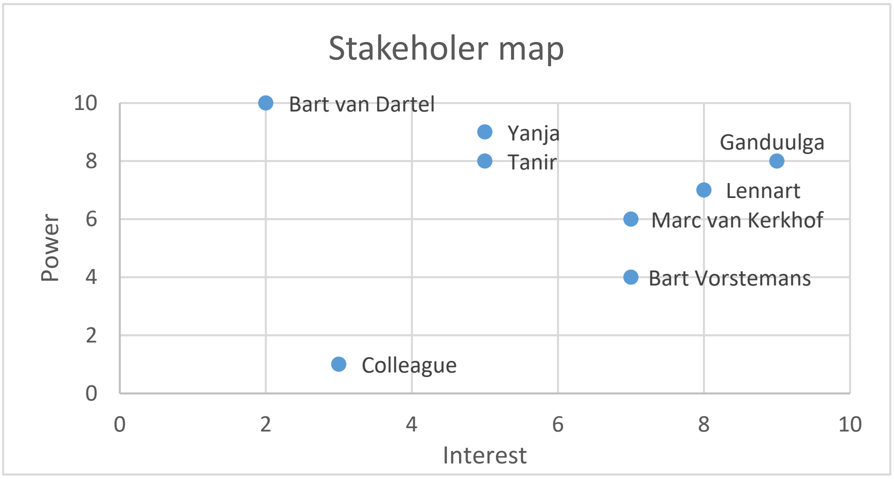


Figure 30 – Stakeholder map

The map helps the trainee to prioritize the stakeholders further to figure out what strategy the trainee should use for communication with every stakeholder and how much attention every stakeholder needs. Besides, the map is also beneficial for identifying potential risks and misunderstanding. The trainee can adjust his or her influence on the stakeholders based on the map. There are four communication strategies that are overviewed in Figure 5 [21].

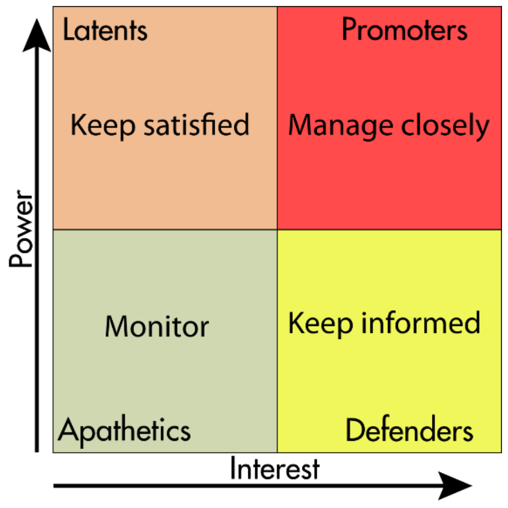


Figure 31 – Stakeholder management strategies on the map

8.2 Work-Breakdown Structure (WBS)

This section describes how the whole project work was structured. The project work was divided into four categories of tasks: Project management, Analysis, Design and Verification, and Documentation. Detailed activities of the work-breakdown are illustrated in Figure 32.

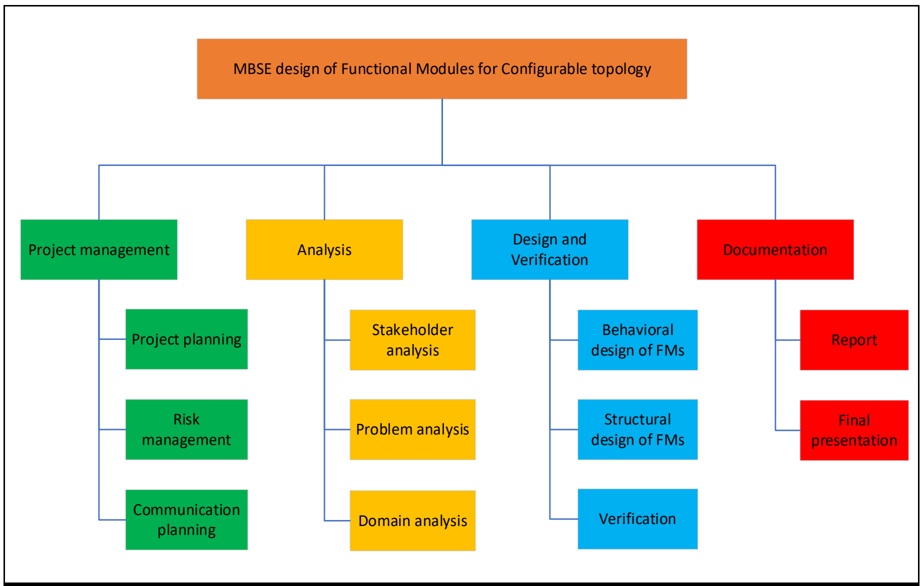
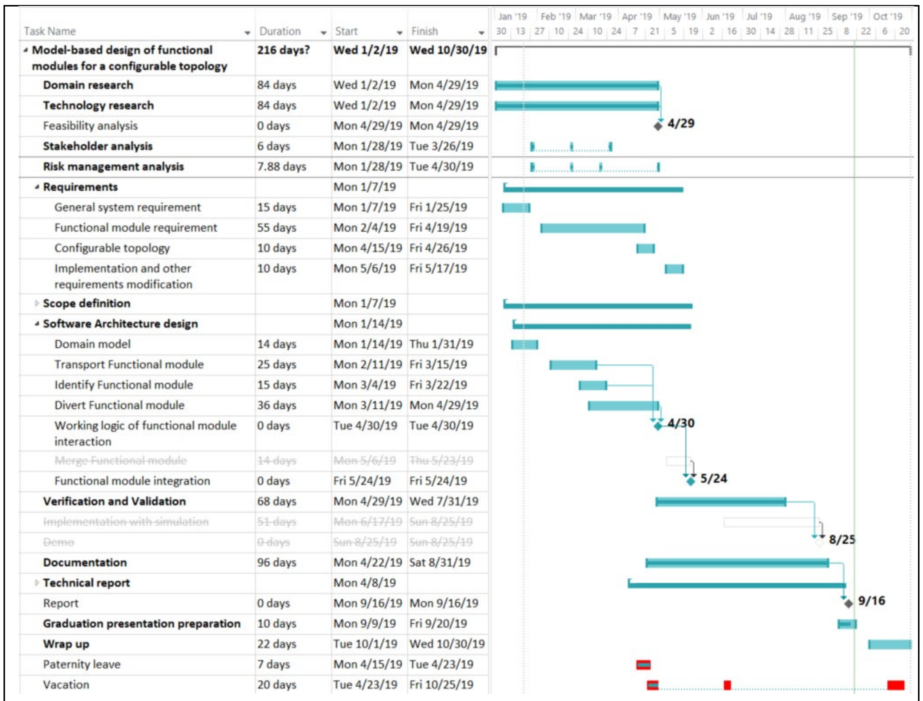


Figure 32 – Work-Breakdown structure of the project

8.3 *Project Planning and Scheduling*

The project planning was performed in accordance with the work-breakdown and its detailed activities. We used a Gantt chart to show project planning in Figure 33. The initial plan was adapted flexibly during the different periods of time in the project. The planning modification was conducted with regard to risk management analysis and risk mitigation activities.

A few tasks are inactivated as a modification to the initial planning in the figure below. These tasks refer to system requirements with a priority of Optional label. Optional requirements provide extra value to the project but these requirements are not mandatory to carry out the project successfully.

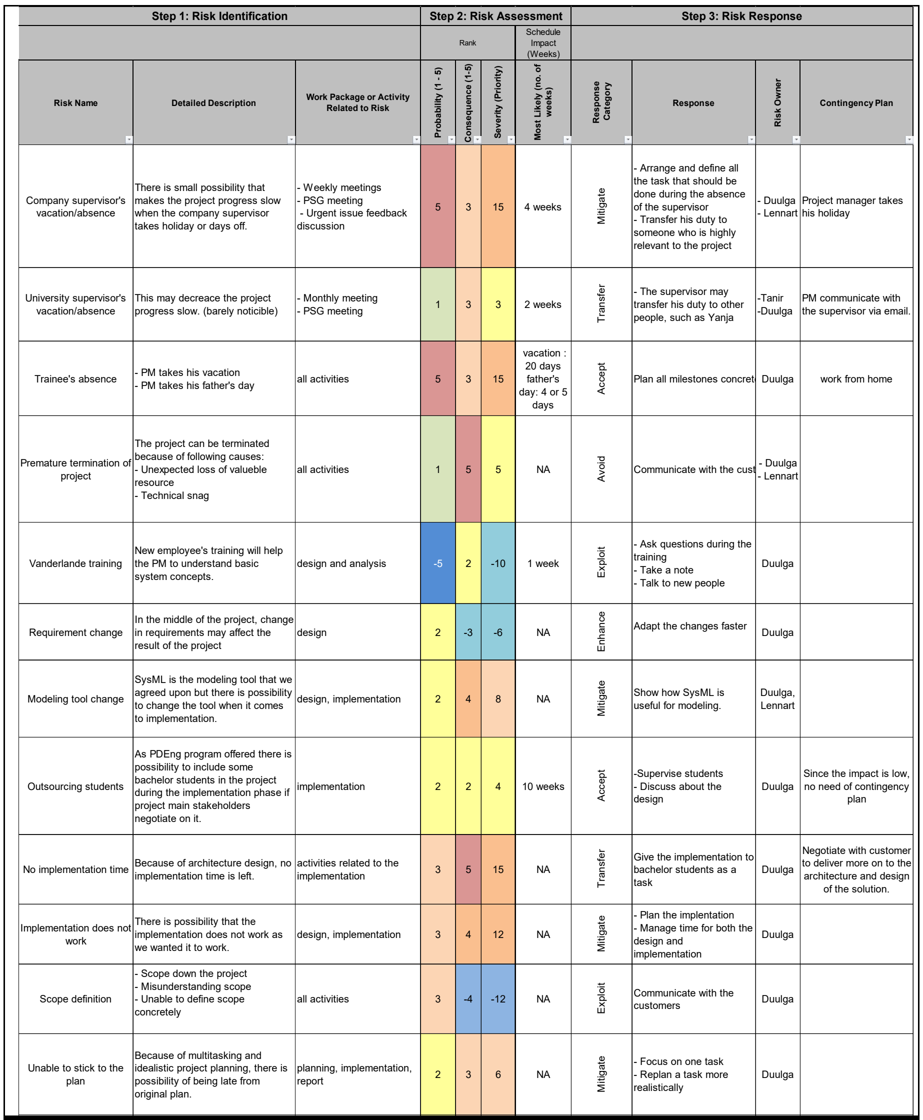


Task Name	Duration	Start	Finish
Model-based design of functional modules for a configurable topology	**216 days?**	**Wed 1/2/19**	**Wed 10/30/19**
Domain research	84 days	Wed 1/2/19	Mon 4/29/19
Technology research	84 days	Wed 1/2/19	Mon 4/29/19
Feasibility analysis	0 days	Mon 4/29/19	Mon 4/29/19
Stakeholder analysis	6 days	Mon 1/28/19	Tue 3/26/19
Risk management analysis	7.88 days	Mon 1/28/19	Tue 4/30/19
Requirements		Mon 1/7/19	
General system requirement	15 days	Mon 1/7/19	Fri 1/25/19
Functional module requirement	55 days	Mon 2/4/19	Fri 4/19/19
Configurable topology	10 days	Mon 4/15/19	Fri 4/26/19
Implementation and other requirements modification	10 days	Mon 5/6/19	Fri 5/17/19
Scope definition		Mon 1/7/19	
Software Architecture design		Mon 1/14/19	
Domain model	14 days	Mon 1/14/19	Thu 1/31/19
Transport Functional module	25 days	Mon 2/11/19	Fri 3/15/19
Identify Functional module	15 days	Mon 3/4/19	Fri 3/22/19
Divert Functional module	36 days	Mon 3/11/19	Mon 4/29/19
Working logic of functional module interaction	0 days	Tue 4/30/19	Tue 4/30/19
~~Merge Functional module~~	~~14 days~~	~~Mon 5/6/19~~	~~Thu 5/23/19~~
Functional module integration	0 days	Fri 5/24/19	Fri 5/24/19
Verification and Validation	68 days	Mon 4/29/19	Wed 7/31/19
~~Implementation with simulation~~	~~51 days~~	~~Mon 6/17/19~~	~~Sun 8/25/19~~
~~Demo~~	~~0 days~~	~~Sun 8/25/19~~	~~Sun 8/25/19~~
Documentation	96 days	Mon 4/22/19	Sat 8/31/19
Technical report		Mon 4/8/19	
Report	0 days	Mon 9/16/19	Mon 9/16/19
Graduation presentation preparation	10 days	Mon 9/9/19	Fri 9/20/19
Wrap up	22 days	Tue 10/1/19	Wed 10/30/19
Paternity leave	7 days	Mon 4/15/19	Tue 4/23/19
Vacation	20 days	Tue 4/23/19	Fri 10/25/19

Figure 33 – Project planning

8.4 *Risk analysis*

This section indicates project uncertainties that may decelerate or accelerate the project progress. Overall risk analysis is depicted in Figure 34. The analysis was conducted in three steps: Risk identification, Risk assessment, and Risk response.



Step 1: Risk Identification			Step 2: Risk Assessment				Step 3: Risk Response			
			Rank			Schedule Impact (Weeks)				
Risk Name	Detailed Description	Work Package or Activity Related to Risk	Probability (1 - 5)	Consequence (1-5)	Severity (Priority)	Most Likely (no. of weeks)	Response Category	Response	Risk Owner	Contingency Plan
Company supervisor's vacation/absence	There is small possibility that makes the project progress slow when the company supervisor takes holiday or days off.	- Weekly meetings - PSG meeting - Urgent issue feedback discussion	5	3	15	4 weeks	Mitigate	- Arrange and define all the task that should be done during the absence of the supervisor - Transfer his duty to someone who is highly relevant to the project	- Duulga - Lennart	Project manager takes his holiday
University supervisor's vacation/absence	This may decreace the project progress slow. (barely noticible)	- Monthly meeting - PSG meeting	1	3	3	2 weeks	Transfer	- The supervisor may transfer his duty to other people, such as Yanja	-Tanir -Duulga	PM communicate with the supervisor via email.
Trainee's absence	- PM takes his vacation - PM takes his father's day	all activities	5	3	15	vacation : 20 days father's day: 4 or 5 days	Accept	Plan all milestones concret	Duulga	work from home
Premature termination of project	The project can be terminated because of following causes: - Unexpected loss of valueble resource - Technical snag	all activities	1	5	5	NA	Avoid	Communicate with the cust	- Duulga - Lennart	
Vanderlande training	New employee's training will help the PM to understand basic system concepts.	design and analysis	-5	2	-10	1 week	Exploit	- Ask questions during the training - Take a note - Talk to new people	Duulga	
Requirement change	In the middle of the project, change in requirements may affect the result of the project	design	2	-3	-6	NA	Enhance	Adapt the changes faster	Duulga	
Modeling tool change	SysML is the modeling tool that we agreed upon but there is possibility to change the tool when it comes to implementation.	design, implementation	2	4	8	NA	Mitigate	Show how SysML is useful for modeling.	Duulga, Lennart	
Outsourcing students	As PDEng program offered there is possibility to include some bachelor students in the project during the implementation phase if project main stakeholders negotiate on it.	implementation	2	2	4	10 weeks	Accept	-Supervise students - Discuss about the design	Duulga	Since the impact is low, no need of contingency plan
No implementation time	Because of architecture design, no implementation time is left.	activities related to the implementation	3	5	15	NA	Transfer	Give the implementation to bachelor students as a task	Duulga	Negotiate with customer to deliver more on to the architecture and design of the solution.
Implementation does not work	There is possibility that the implementation does not work as we wanted it to work.	design, implementation	3	4	12	NA	Mitigate	- Plan the implentation - Manage time for both the design and implementation	Duulga	
Scope definition	- Scope down the project - Misunderstanding scope - Unable to define scope concretely	all activities	3	-4	-12	NA	Exploit	Communicate with the customers	Duulga	
Unable to stick to the plan	Because of multitasking and idealistic project planning, there is possibility of being late from original plan.	planning, implementation, report	2	3	6	NA	Mitigate	- Focus on one task - Replan a task more realistically	Duulga	

Figure 34 – Risk management plan

In the first step, we identified the risks and defined each risk with detailed descriptions. This step required the ability to realize and discover probable uncertainty during the project. In addition, every risk was defined with several activities that can be influenced significantly by the corresponding risk.

In the second step, risks are assessed with two criteria: probability and consequence. Probability indicates how certain the risk occurs. On the other hand, consequence describes the impact of the risk. Each criterion has rank points from1 to 5 in order to assess the risk. Multiplication of the two criteria is severity, which is the overall risk priority that the trainee considered. Besides, there are two types of risks:

- Threat which slows down the project progress. The rank point ranges from 1 to 5.
- Opportunity which accelerates the project progress. The rank point ranges from -1 to -5.

In the last step, we described response activities of the risk when the defined risk occurs. ■

9. Project Retrospective

This chapter finalizes the report by providing reflection including challenges from the author's point of view. Furthermore, lessons learned are explained in two categories of technical and organizational insights.

9.1 *Reflection*

The project conducted during the last ten months at Vanderlande brought me quite a challenging and yet interesting experience. Through the project, I improved my personal and professional skills by facing both technical and managerial challenges.

As the project goal was quite vague and general at the beginning, a significant amount of time has been devoted to analyzing the project problems in detail. For this purpose, numerous interviews were conducted with stakeholders in the first quarter of the project. Meanwhile, I had to define project scope clearly considering the limited time to satisfy the company's requirements as well as university standards.

Moreover, understanding the automated material handling domain was necessary to be able to create a project solution. Thus, I enrolled in the company's internal trainings to gain domain knowledge. Additionally, a considerable period of time was dedicated to learning internal documents about systems and systems' architecture for the purpose of domain knowledge.

Since the company desired to carry out the project using a model-based approach, several approaches are compared by analyzing their traits. Choosing the appropriate modeling approach of either MBD or MBSE was such a dilemma. Thanks to the company's new system architecture and nature of the project, MBSE was selected.

In addition to the technical challenges, I have encountered managerial challenges because I was in charge of the entire process during the project. This gave me the possibility to experience different roles at the same time.

Moreover, I could grow further as a Software/System Architect and Designer by means of the lessons that this process taught me. The lessons are listed below.

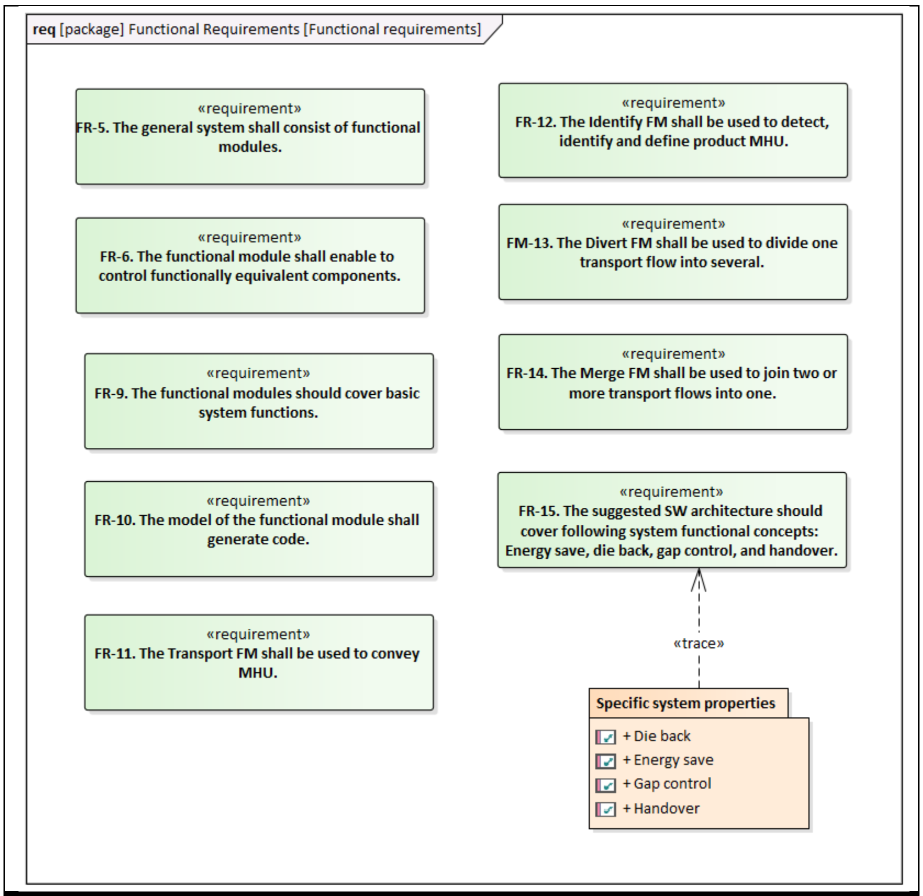
9.2 *Lessons learned*

- Technical insights
    - Modeling approaches. During the project, I learned to distinguish which modeling approach fits for what problem.
    - SysML. Even though I had SysML training during the first year of the PDEng program, I could not apply the knowledge I gained to the broader context than the training. Through the project, I improved my SysML understanding by applying it to the domain.
    - Patterns. For the purpose of pattern selection, I learned multiple design patterns and their traits.
- Organizational insights
    - I realized that I should be more proactive and initiative to make things clear.
    - I understood the feedback culture in general. A lesson I learned is "Receive feedbacks on the project work as early as possible". This also leads to a communication plan in which I should consider that stakeholders are also busy doing their work.
    - Start immediately and fail fast. This concept helps to decrease the project risk and verify requirements in the early stage of the project.

10. Appendix

This chapter gives additional information on the corresponding processes of the project.

10.1 System requirements


req [package] Functional Requirements [Functional requirements]

«requirement»
FR-5. The general system shall consist of functional modules.

«requirement»
FR-6. The functional module shall enable to control functionally equivalent components.

«requirement»
FR-9. The functional modules should cover basic system functions.

«requirement»
FR-10. The model of the functional module shall generate code.

«requirement»
FR-11. The Transport FM shall be used to convey MHU.

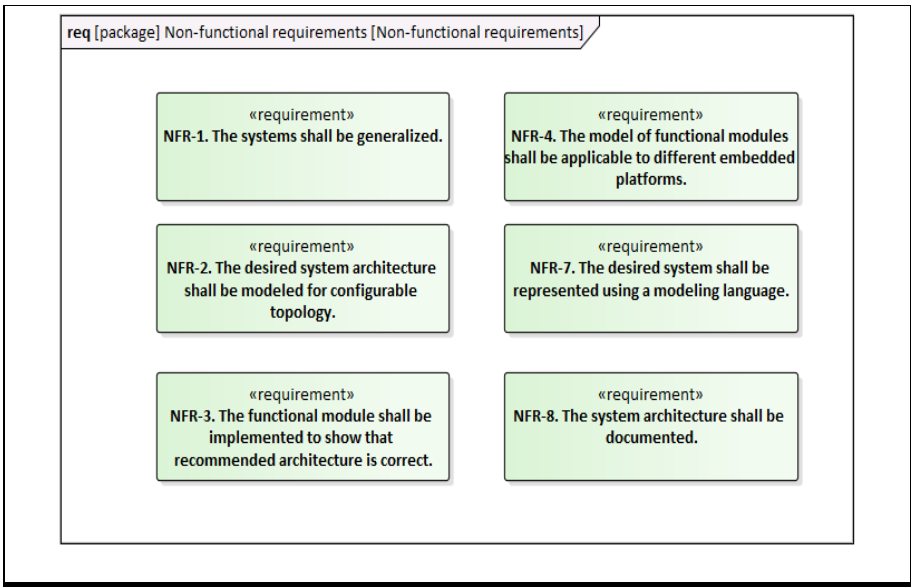
«requirement»
FR-12. The Identify FM shall be used to detect, identify and define product MHU.

«requirement»
FM-13. The Divert FM shall be used to divide one transport flow into several.

«requirement»
FR-14. The Merge FM shall be used to join two or more transport flows into one.

«requirement»
FR-15. The suggested SW architecture should cover following system functional concepts: Energy save, die back, gap control, and handover.

Figure 35 – Functional requirements


req [package] Non-functional requirements [Non-functional requirements]

«requirement»
NFR-1. The systems shall be generalized.

«requirement»
NFR-4. The model of functional modules shall be applicable to different embedded platforms.

«requirement»
NFR-2. The desired system architecture shall be modeled for configurable topology.

«requirement»
NFR-7. The desired system shall be represented using a modeling language.

«requirement»
NFR-3. The functional module shall be implemented to show that recommended architecture is correct.

«requirement»
NFR-8. The system architecture shall be documented.

Figure 36 – Non-functional requirements

10.2 Use case analysis

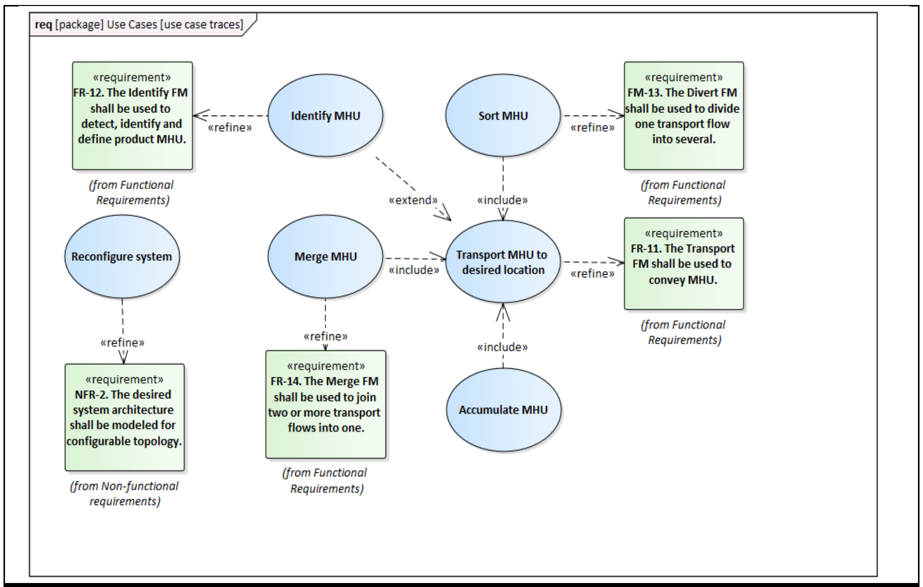

req [package] Use Cases [use case traces]
«requirement» FR-12. The Identify FM shall be used to detect, identify and define product MHU.
(from Functional Requirements)
«refine»
Identify MHU
Sort MHU
«refine»
«requirement» FM-13. The Divert FM shall be used to divide one transport flow into several.
(from Functional Requirements)
«extend»
«include»
Reconfigure system
Merge MHU
«include»
Transport MHU to desired location
«refine»
«requirement» FR-11. The Transport FM shall be used to convey MHU.
(from Functional Requirements)
«refine»
«refine»
«include»
«requirement» NFR-2. The desired system architecture shall be modeled for configurable topology.
(from Non-functional requirements)
«requirement» FR-14. The Merge FM shall be used to join two or more transport flows into one.
(from Functional Requirements)
Accumulate MHU

Figure 37 – Traces between user goal and system requirement

10.3 System design

Activities in the general scenario of sorting are illustrated in Figure 38. Suppose the system is dedicated to sorting out fragile MHU in the system. This scenario activities belong to Sense FM, Transport FM, and Divert FM.
When a MHU enters the system, Sense FM retrieves MHU information by controlling sensors. If the detected unique ID (UUID) of the MHU does not exist in the system, the system registers MHU by creating MHU data with corresponding attribute values. If the ID exists, the system simply updates corresponding attributes. These activities are specified in the sequence diagram by illustrating interactions between MHD Manager and Sense FM instances in Figure 39.

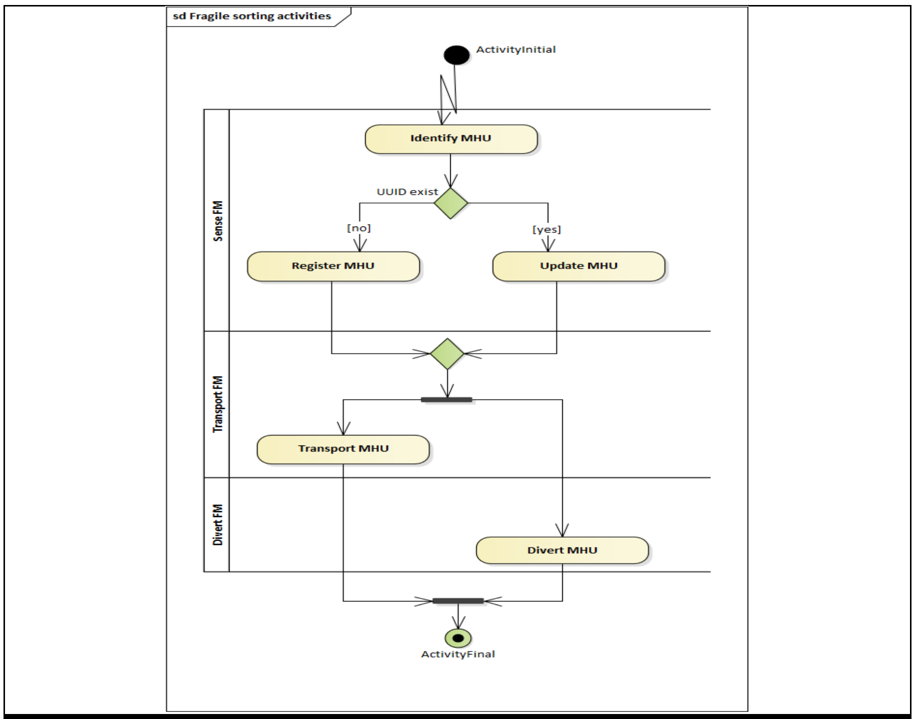


Figure 38 – Activity diagram of sorting scenario

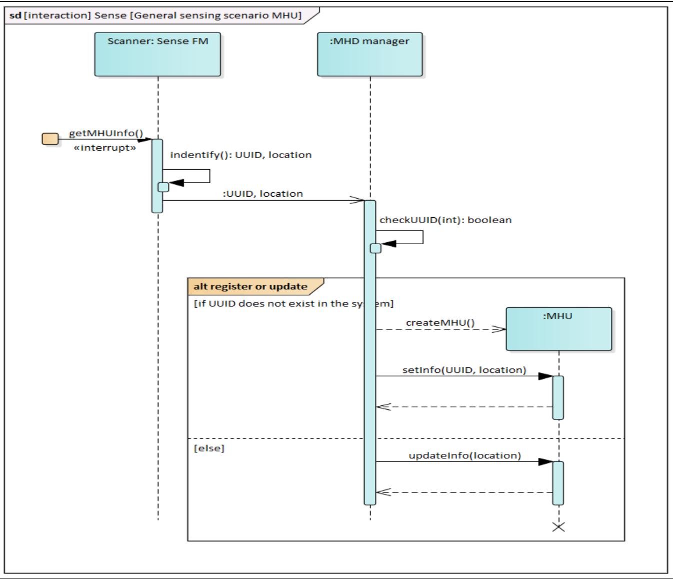


Figure 39 – Interactions between SenseFM and MHD Manager

10.4 Conceptual simulation for (re)configurability

For the configurability requirement, we also used the simulation to visualize the concept. The simulation for configurable topology is depicted in Figure 40, showing two system layouts. In this simulation, the system is built in the same way as we developed in the previous simulations. In order to validate the design from the configurability perspective, we need to show that the same logical components of the modular system work for different system layouts.

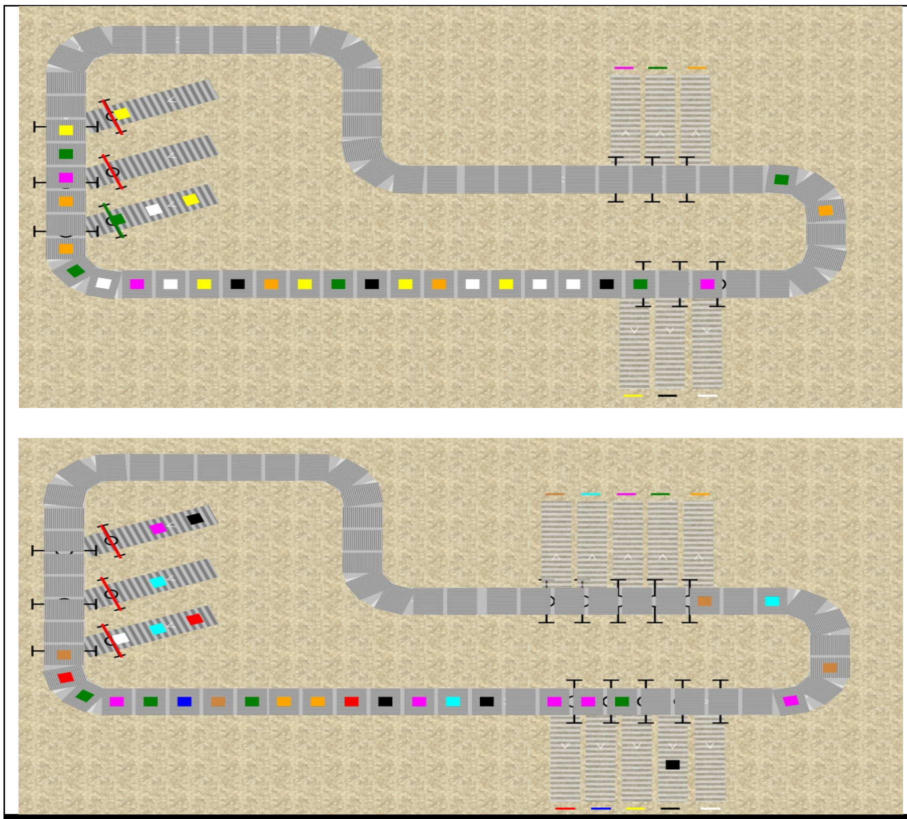


Figure 40 – System layout for configurable topology simulation

The first system layout has six sorting outputs and the second system layout has ten sorting outputs. Even though the layouts are different, the same logic runs on the two systems. To do so, we added four more sorting paths on the layout and assigned the same behaviors to these paths.

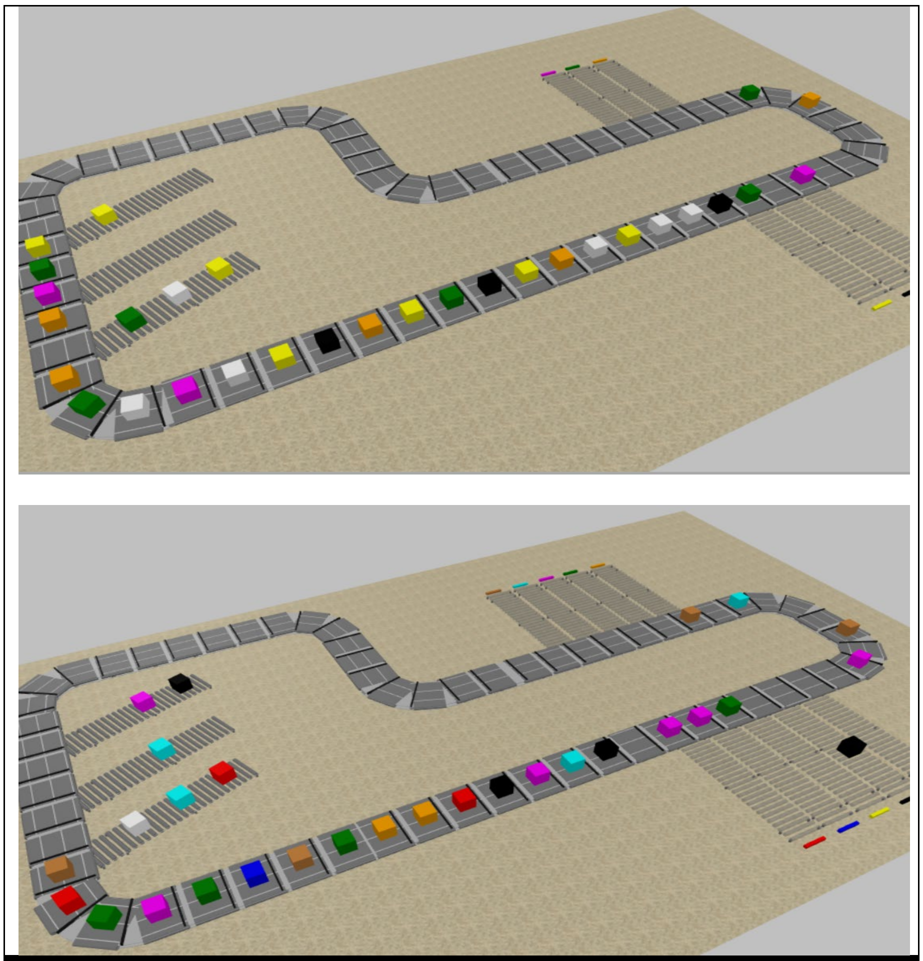


Figure 41 – 3D illustration of configurable topology simulation

Glossary

Terminologies of the project are defined in the glossary. Some common technical terminologies are referenced from Wikipedia.

Functional Module	Functional Module is a small building block of the system software that controls functionally equivalent physical components in the system.
Market Leading Concept	Market Leading Concept (MLC) is a standardized template solution for every business segment, namely food, fashion, parcel, general merchandise as it is specified by the company. It has a predefined set of Functional Modules from which some FMs can be discarded on the customer's requirement but cannot be added to the set.
Material Handling Unit	A MHU is any physical object in a system that can be conveyed by any transport equipment or component in the system. For example, baggage in Airport, parcel in Postal, product in Warehouse systems.
Material Handling Domain	The Material Handling domain specifies how the items should be moved to fulfill the process flow. It handles the actual physical movements of the items but is not aware of the reasons why these items need to be handled.
Model-based design	Model-Based Design is a mathematical and visual method of addressing problems associated with designing complex control, signal processing, and communication systems.
Model-Based Systems Engineering	Model-Based Systems Engineering is a systems engineering methodology that focuses on creating and exploiting domain models as the primary means of information exchange between engineers, rather than on document-based information exchange.
Programmable Logic Control	A programmable logic controller is an industrial digital computer that has been ruggedized and adapted for the control of manufacturing processes, such as assembly lines, or robotic devices, or any activity that requires high-reliability control and ease of programming and process fault diagnosis.
Unified Modeling Language	The Unified Modeling Language is a general-purpose, developmental, modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system.
Systems Modeling Language	The Systems Modeling Language is a general-purpose modeling language for systems engineering applications.
Topology	Hardware infrastructure of the system.

Bibliography

[1] S. Ray, Introduction to Materials Handling, New Age International (P) Limited, 2007.

[2] M. Spindler, T. Aicher, D. Sch_tz, B. Vogel-Heuser and W. A. G_nthner, "Efficient Control Software Design for Automated Material Handling Systems Based on a Two-Layer Architecture," in *5th IEEE International Conference on Advanced Logistics and Transport (ICALT)*, 2016.

[3] W. Lepuschitz, A. Zoitl, M. Vall_e and M. Merdan, "Toward Self-Reconfiguration of Manufacturing Systems Using Automation Agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews),* vol. 41, pp. 52-69, 1 2011.

[4] S. Friedenthal, A. Moore and R. Steiner, A Practical Guide to SysML: The Systems Modeling Language, Elsevier Science, 2011.

[5] L. Delligatti, SysML Distilled: A Brief Guide to the Systems Modeling Language, Addison-Wesley, 2014.

[6] G. Muller, "Positioning the CAFCR Method in the World," 2018.

[7] G. Muller, "CAFCR: A multi-view method for embedded systems architecting".

[8] [Online]. Available: https://realpars.com/automation-pyramid/.

[9] D. Gorecky, S. Weyer, A. Hennecke and D. Zühlke, "Design and instantiation of a modular system architecture for smart factories," *IFAC-PapersOnLine,* vol. 49, pp. 79-84, 2016.

[10] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, "A System of Patterns," in *Pattern-Oriented Software Architecture*, Wiley, August 1996.

[11] E. Estevez, M. Marcos, N. Iriondo and D. Orive, "Graphical modeling of plc-based industrial control applications," in *2007 American Control Conference*, 2007.

[12] [Online]. Available: https://www.vanderlande.com/warehousing/evolutions/airpick/.

[13] [Online]. Available: https://www.vanderlande.com/warehousing/evolutions/fastpick/.

[14] [Online]. Available: https://www.vanderlande.com/warehousing/evolutions/storepick/.

[15] Vanderlande System Architecture Team, "SYSTEM/SUBSYSTEM DESIGN DESCRIPTION W-PLATFORM SSDD," Veghel, 2019.

[16] [Online]. Available: https://www.uml-diagrams.org/use-case-include.html.

[17] "OMG Systems Modeling Language," Object Management Group, [Online]. Available: https://www.omg.org/intro/SysML.pdf.

[18] M. H. Hassan Gomaa, "Software Reconfiguration Patterns for Dynamic Evolution of Software," in *Proceedings of the Fourth Working IEEE/IFIP Conference on Software Architecture*, 2004.

[19] H. C. Robert Szepesi, "An Overview on Software Reconfiguration," *Theory and Applications of Mathematics & Computer Science,* vol. 1, pp. 74-79, 2011.

[20] A. Zhiliaeva, "Anylogic blog," [Online]. Available: https://www.anylogic.com/blog/conveyors-using-the-material-handling-library-part-1/.

[21] [Online]. Available: https://en.wikipedia.org/wiki/Stakeholder_analysis.

About the Author

Ganduulga Gankhuyag received both his BSc and MSc degree in Electronics Engineering from the Department of Electronics and Information Engineering of Chonbuk National University, Jeonju city, South Korea in 2012 and 2015, respectively.

His master thesis concerned the idea that heading and position accuracy of the navigation system can be enhanced using various Kalman filters integrating Global Positioning System and Inertial Navigation System.

After his graduation, he did his internship at Daewoo Electronic Components Co., Ltd and worked as a Hardware and Software R&D engineer at Ametros Solutions LLC.

From November 2017 until October 2019, he worked at the Eindhoven University of Technology, as a PDEng trainee in the Software Technology program from the 4TU.Stan Ackermans Institute. He is interested in topics related to Embedded systems, the Internet of things, Artificial Intelligence and Software architecture.

Where innovation starts