Digital Twin

PDEng THESIS REPORT

Digital Twin - Virtual Hardware Simulator for a Transmission Electron Microscope

Enkhdavaa Batlkhagva
October 2020
Department of Mathematics & Computer Science

PDEng SOFTWARE TECHNOLOGY

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Digital Twin - Virtual Hardware Simulator for a Transmission Electron Microscope

Enkhdavaa Batlkhagva

October 2020

Eindhoven University of Technology
Stan Ackermans Institute – Software Technology

PDEng Report: 2020/065

Confidentiality Status:
Confidential till January 2021.

Partners		
	Thermo Fisher Scientific	Eindhoven University of Technology
Steering Group	Arjen Klomp Pepijn Kramer Edwin Verschueren Ion Barosan	
Date	October 2020	

Composition of the Thesis Evaluation Committee:

Chair:	Harold Weffers
Members:	Pepijn Kramer
	Ion Barosan
	Adriana Hazelua
	Huub van de Wetering

The design that is described in this report has been carried out in accordance with the rules of the TU/e Code of Scientific Conduct.

Contact Address
Eindhoven University of Technology
Department of Mathematics and Computer Science
MF 5.080B, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
+31 402472759

Partnership
This project was supported by Eindhoven University of Technology and Thermo Fisher Scientific.

Published by
Eindhoven University of Technology
Stan Ackermans Institute

PDEng-report
2020/065

Preferred reference
Digital Twin - Virtual Hardware Simulator for a Transmission Electron Microscope. Eindhoven University of Technology, PDEng Report, 2020/065, October 2020

Abstract
Tests are carried out on the physical hardware to verify the product design. Due to a limited budget and time, the limited number of test systems of a product will be manufactured. As a result, the test systems limit the speed of the testing phase. Test systems are expensive due to manufacturing time and costs. Moreover, producing these systems (prototypes) takes a long time, which makes the testing and redesigning feedback loops slow. This report describes the design and implementation of the virtual hardware simulator that was introduced to eliminate the hardware dependency in the design verification tests. The virtual hardware simulator provides real-time motion visualization and sensor simulation.

Keywords
PDEng, TU Eindhoven, Virtual Hardware, Simulation, Virtual Prototype, Digital Twin

Foreword

At Thermo Fisher Scientific we are proud to develop leading edge electron microscopes consisting of many complex electro-mechanical and software components. The design process usually involves building hardware (+electronic) prototypes on which the microscope control software is further developed. Creating the first prototypes is usually a process with long lead times, and when some redesign is needed this lead time quickly grow. For development and testing of software, various strategies have been developed over the years: from hardware test benches, software simulators and simulation code in production code.
We started hearing about a new trend in the industry to create digital twins as virtual copies of the real hardware in which actuators, sensors and kinematic relations are fully simulated in a 3D environment. Since digital twins are new for Thermo Fisher we didn't know how to make them, where they would be most useful, what kind of process would be needed to create them and what the costs and benefits would be. For his PDEng project we asked Enkhdavaa to investigate these very open ended questions for us and to come up with a working prototype and recommendations on how to work with digital twins.

Enkhdavaa has enthusiastically, open-minded and pro-actively (including contact with Unit040, the supplier of Prespective, that we used as the technology of choice for this project) been working on finding the answers to those questions and it has been a joy working with him throughout the project. He created a software architecture, software design and working prototype that answered the questions of how to and where to integrate a digital twin in our software stack. What I personally really liked is that the digital twin connects at the same level as real hardware so that there is no need in the rest of the software stack to add specialized simulation control paths.
Enkhdavaa convincingly showed that digital twins have a role in the design phase of the hardware and mechatronics of a new microscope component. By taking 3D CAD data and turning it into interactive models. Allowing us to test and develop software even before the first physical prototype became available. He also uncovered there will be a need for specialized roles in the organization to develop and maintain the digital twins.

All in all it has been a pleasure working with Enkhdavaa and I wish him all the best in his career.

PROJECT MENTOR : Pepijn Kramer Msc.
Date: September 10^{th,} 2020

Preface

The project was carried out for Thermo Fisher Scientific, a company, which aims to enable its customers to make the world healthier, cleaner, and safer. The company designs and develops various types of microscopes, and one of them is Transmission Electron Microscopes (TEM).

This project was conducted by Enkhdavaa Batlkhagva from the Stan Ackerman's Institute, Professional Doctorate in Engineering (PDEng) Software Technology program of the Eindhoven University of Technology. This project is the ten-month final assignment for the two-year PDEng program, known by its Dutch name as Ontwerpers Opleiding Technische Informatica (OOTI).

This report is intended for people who have a technical background in motion engineering, digital twinning, and three-dimensional (3D) modeling. However, no specialized knowledge of these disciplines is needed.

Readers who are interested in non-technical parts of the report should read chapters 1-6. Those who are interested in technical details should read chapters 7-10. Finally, the result of the project is presented in Chapter 11.

Enkhdavaa Batlkhagva
October 2020

Acknowledgments

This thesis would not have been a success without inspirational and motivational people who have provided continuous support and valuable feedback. Thanks to them for being a part of this journey and assisting me to grow professionally and personally.

Firstly, I am deeply grateful to the people from Thermo Fisher Scientific. I am especially thankful to Pepijn Kramer, Edwin Verschueren, and Iryna Stepaniak, who have given me practical advice and guided me throughout this project. Without your tremendous support and guidance, I would never have reached this level of the project. You taught me a ton of things, and I have learned a lot from you. I also would like to thank all Thermo Fisher Scientific engineers who have supported and collaborated with me. I would like to acknowledge the help provided by Arjen Klomp, who has helped me to integrate with the company and to obtain the necessary software products for the project. Mike Musterd, Ferry Hoogedoorn, and the Manage Sample group have been the source of the domain knowledge. Bhupal Reddy Bommineni has helped me to obtain necessary files from the mechanical team. I also send my regards to the SETI and IT teams.

I would especially like to express my gratitude to Ion Barosan, who has been my supervisor at the university. He has given me plenty of valuable advice enthusiastically. His advice and suggestions helped me to determine the next steps when I was blocked during the project. He also has given me support in writing the project report.

My acknowledgment goes to everybody who has been involved in the PDEng program. I would like to thank Yanja Dajsuren and Desiree van de Oorschot for managing the PDEng program and giving me the opportunities to have valuable workshops and lectures.

Finally, I would like to thank my parents and my sisters for always giving me inspirational advice. I also would like to appreciate Ganduulga Gankhuyag and Giovanni de Almeida Calheiros for their support during difficult times.

Enkhdavaa Batlkhagva
October 2020

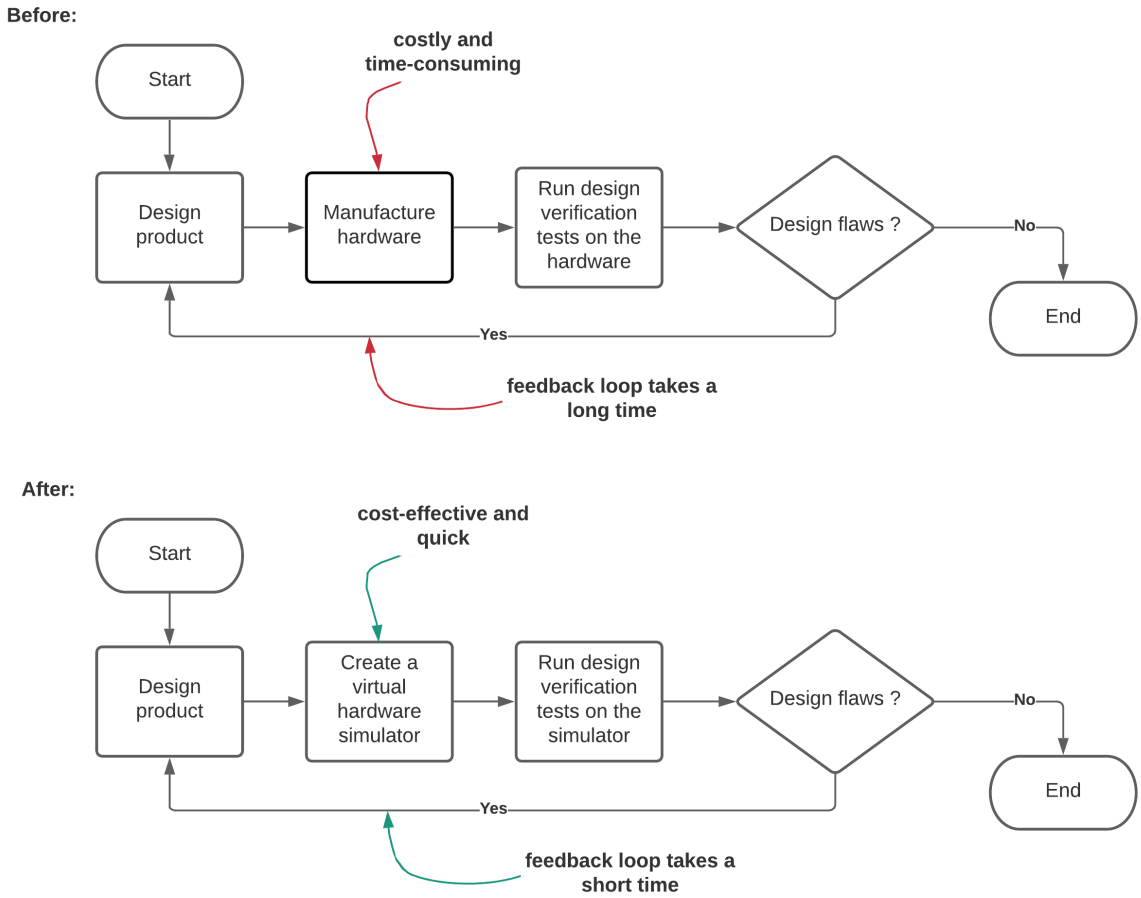
Executive Summary

This document summarizes the “Virtual Hardware Simulator for a Transmission Electron Microscope” project. It addresses the challenges of creating a simulator that is able to simulate real-time hardware behavior, visualize it, and perform real-time sensor simulation using models from various disciplines. The primary purpose of the simulator is to perform tests without relying on hardware.

Thermo Fisher Scientific continually improves the functionality of the TEM with new features. During the development of these new features, design verification tests must take place as soon as possible to detect design flaws. Early error detection can save additional costs, such as manufacturing time and effort, and can accelerate product delivery time.

Designing and developing these features is not straightforward because of the complex management and involvement of a multidisciplinary team. Especially during the verification phase, the involvement of the multidisciplinary team is mandatory. Therefore, the project was conducted to explore how the concept of digital twinning can facilitate the design verification phase and speed up the product development lifecycle.

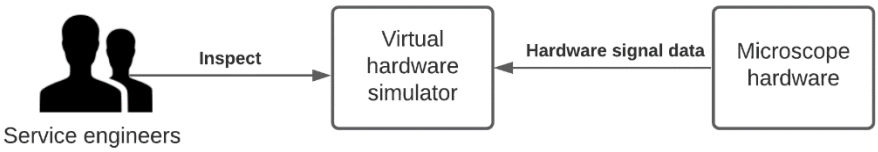
The first conclusion is that the concept of digital twinning can be the most beneficial to bring the multidisciplinary team together to verify their design during product development. Currently, the verification is dependent on physical hardware (test systems) that takes around two months to be manufactured and assembled. Design incompatibilities and flaws are usually found while design verification tests take place on physical hardware. To speed up the verification process, we introduced a virtual hardware simulator that is a virtual prototype of the product.



The simulation can be created from various design models before hardware is manufactured. Also, the simulation visualizes the real-time motion behavior of hardware using three-dimensional (3D) models in a 3D environment.

In that environment, virtual sensors are placed to simulate 3D movements, which allows us to verify the design and possibly detect design flaws.

The second conclusion is that the concept of the digital twin is beneficial for product maintenance by visualizing the sensor data of the real system. From the visualization, engineers are able to inspect the real-time motion behavior of physical hardware to diagnose problems.



In this project, we designed and implemented a virtual hardware simulator. It brings a multidisciplinary team together in the early phase of TEM development. Failures in the design, such as collisions, can be detected. Also, the software execution on the hardware can be monitored in real time. Moreover, the simulator can draw sensor data from a real microscope to visualize the hardware behavior in a 3D environment. From the virtual environment, the service team can inspect problems when maintenance is required. As a result, problem diagnosis is faster when TEM misbehaves.

Table of Contents

Foreword ..... **i**

Preface ..... **iii**

Acknowledgments ..... **v**

Executive Summary ..... **vii**

Table of Contents ..... **ix**

List of Figures ..... **xiii**

List of Tables ..... **xv**

1. Introduction **1**

1.1 Context *1*

1.2 Goal *1*

1.3 Methodology *1*

1.4 Outline *2*

2. Domain Analysis **3**

2.1 Transmission Electron Microscope *3*

2.2 Microscope Development *4*

2.2.1. Design phase 5

2.2.2. Implementation phase 9

2.2.3. Design verification phase 9

2.3 Conclusion *9*

3. Problem Analysis **11**

3.1 Problem statement *11*

3.1.1. Challenges in firmware development 11

3.1.2. Software development dependencies 11

3.1.3. Limited hardware resources 11

3.2 Solution *11*

3.3 Conclusion *12*

4. Technology Exploration **13**

4.1 Commercial software products *13*

4.2 Criteria for selecting software products *13*

4.2.1. PRESPECTIVE 14

4.2.2. Visual Components 15

4.2.3. Solidworks 15

4.2.4. Siemens NX 16

4.3 Selecting a simulation software product *16*

4.4 Conclusion *17*

5. Stakeholder Analysis **19**

5.1 Thermo Fisher Scientific *19*

5.2 TU/e *20*

5.3 Conclusion *20*

6. Analysis – Requirements, System Context, and Use Cases **21**

6.1 Introduction *21*

6.2 System requirements *21*
6.2.1. REQ1.1-Provide real-time sensor simulation 21
6.2.2. REQ1.2-Provide a visual representation for real-time hardware motion 22

6.3 User requirements *22*

6.4 Non- functional requirements *23*
6.4.1. Compatibility 23
6.4.2. Extensibility 23
6.4.3. Availability 23

6.5 System Context *23*
6.5.1. System context 23
6.5.2. Information Flow 24

6.6 Use cases *27*

6.7 Conclusion *28*

7. System Design **29**

7.1 Introduction *29*

7.2 Step 1 ..... *29*
7.2.1. Creating virtual hardware 29

7.3 Step 2 ..... *29*
7.3.1. Design Choice 1 29
7.3.2. Design Choice 2 30
7.3.3. Design Decision 32

7.4 Conclusion *32*

8. System Architecture **33**

8.1 Architecture design *33*

8.2 Unity game engine *33*
8.2.1. Visualizing motion behavior 34
8.2.2. Sensor simulation 34

8.3 StagePlugin *34*

8.4 StageAdapter *34*
8.4.1. Configuration File 34
8.4.2. Communication Technology 34

8.5 Conclusion *35*

9. Implementation **37**

9.1 Development environment *37*
9.2 Simulation on Unity game engine *37*
9.3 StagePlugin *39*
9.4 StageAdapter *39*
9.4.1. Configuration file 40
9.5 System behavior *40*
9.6 Conclusion *41*

10. Verification **43**
10.1 Preparing a test environment *43*
10.2 Verification *43*
10.2.1. Use case: Visualize the hardware motion 43
10.2.2. Use case: Testing desired behavior 44
10.2.3. Use case: Testing desired behavior (fault scenario) 45
10.2.4. Visualizing the physical hardware behavior 46
10.3 Conclusion *46*

11. Conclusions & Recommendations **47**
11.1 Summary *47*
11.2 Recommendations *48*
11.2.1. Short-term 48
11.2.2. Long-term 48

12. Project Management **49**
12.1 Introduction *49*
12.2 Project Planning and Scheduling *49*
12.3 Project execution *49*
12.4 Risk analysis *49*
12.5 Challenges *50*

13. Project Retrospective **51**
13.1 Reflection *51*

Glossary **53**

Bibliography **55**

About the Authors **57**

List of Figures

Figure 1: Transmission Electron Microscope 3
Figure 2: The internal software structure of the TEM server 4
Figure 3: V-Model 5
Figure 4: TEM design levels 5
Figure 5: TEM software and its interfaces 6
Figure 6: Motion controller 6
Figure 7: A mathematical model that was created for an actuator 7
Figure 8: Motion graph of an actuator 7
Figure 9: Diagram showing that the same firmware runs on both model and hardware 8
Figure 10. Modeled mechanical design that is imported in Unity game engine 9
Figure 11: System requirements 21
Figure 12: System context diagram 24
Figure 13: CAD models 25
Figure 14: Actuator signal data 25
Figure 15. Sensor signal data 25
Figure 16: User input 26
Figure 17: Sensor command 26
Figure 18: A use case diagram for a mechatronics engineer 27
Figure 19. A use case diagram for a test engineer 28
Figure 20: Process of importing 3D models into PRESPECTIVE environment 29
Figure 21: Simulation diagram when using *IMdlMotion* 30
Figure 22: Simulation diagram when using *IHalMotion* 30
Figure 23. Motion controller modes 31
Figure 24: An example diagram for ball-valve component 31
Figure 25: Visualizing physical hardware in a virtual environment 32
Figure 26: Virtual Hardware Simulator (VHS) 33
Figure 27: StartScene 37
Figure 28: Simulation scene for the ball valve component 38
Figure 29: Class diagram for visualizing motion behavior and simulating sensors 38
Figure 30: Class diagram of StagePlugin 39
Figure 31: Class diagram of StageAdapter 40
Figure 32: The short version of the configuration file 40
Figure 33. Sequence diagram for the VHS and external systems 41
Figure 34. Sequence diagram for simulating the ball-valve component 44
Figure 35: Real-time sensor simulation for the ball-valve component 45
Figure 36: The result after collision 45
Figure 37. Sequence diagram for visualizing the physical hardware on the VHS 46
Figure 38. Development dependency for the creation of the virtual hardware simulator 47
Figure 39: Project plan 49

List of Tables

Table 1: Commercial software products 13
Table 2: Criteria for selecting a commercial software product 13
Table 3: PRESPECTIVE investigation 14
Table 4: Visual Components investigation 15
Table 5: Solidworks investigation 15
Table 6: Siemens NX investigation 16
Table 7: Company stakeholder 1 19
Table 8: Company stakeholder 2 19
Table 9: Company stakeholder 3 19
Table 10: Company stakeholder 4 20
Table 11: Company stakeholder 5 20
Table 12: TU/e stakeholder 1 20
Table 13: TU/e stakeholder 2 20
Table 14: Functional requirements to perform real-time sensor simulation 21
Table 15: Functional requirements to visualize real-time motion behavior 22
Table 16: User requirements 22
Table 17: Risk analysis 50

1. Introduction

This chapter firstly introduces the project context and project goals. The following are the methodologies that are used during the project lifetime and the outline section that gives an overview structure of this report.

1.1 Context

Thermo Fisher Scientific (formerly known as FEI) is a world-leading company that aims to enable its customers to make the world healthier, cleaner, and safer. The company designs and develops different types of TEM that take ultra-high-resolution images at the nanometer level.

Various customers use TEM for their daily work. Based on their feedback, new features are added to the TEM. To deliver these new features, engineers at the research and development department of Thermo Fisher Scientific continuously develop, test, and release new features to improve the functionality of the microscopes.

Introducing a new feature to TEM is not straightforward because the hardware consists of mechanical, optical, and electronic parts, which are controlled by real-time and operational software algorithms. Moreover, the needing for a multidisciplinary team contribution is essential for the design, development, and test of the new features. The new features must be implemented within a time-constrained project planning with proven reliability. At the same time, engineers must find bugs and flaws as fast as possible to speed up the product development lifecycle.

To detect the faults from the design of the new feature, tests must be conducted intensively. These tests ensure that the functionality of the microscope performs correctly. After successful tests, the new feature is integrated with the existing microscope system. Notably, the integration process is also challenging because most of the bugs and flaws are usually found during this process.

Nowadays, the involvement of physical hardware is crucial for design verification because software and firmware need to be integrated with it. Unfortunately, preparing the hardware is not an easy task. Hardware needs to be manufactured, assembled, and prepared for the testing. This lengthy process takes a couple of months and human and machine effort.

Since hardware is the central part of the design verification, several pieces of hardware are prepared. As a result, the number of tests is limited by the number of available hardware. Sometimes, the hardware is not functioning at optimal performance. It also can be a case that the hardware is occupied, not allowing other engineers to perform tests because multiple disciplines need to validate their design and put a claim on available machine time. The time for the tests is limited. Therefore, the availability of the hardware is essential to finish carrying out the tests within a dedicated timeframe.

This project aims to eliminate the hardware dependency in testing by creating virtual hardware. More specifically, we focused on the part of TEM hardware that manipulates a specimen. This part is called the *stage*, and we intended to create a virtual version of it.

1.2 Goal

The concept of digital twinning is thriving these days [5][6][7][9][16]. The goal of this project was to explore the concept of digital twinning and use the concept to accelerate product development. By definition, a digital twin is a dynamic model that represents a physical twin throughout its entire lifecycle. Any data that can be obtained from a physical twin can be obtained from the digital twin.

However, we used the concept partially. We only implemented the digital twin without the involvement of the physical twin because all of the identified problems in this project context occur before the physical system exists. In that case, the digital twin act as a digital prototype to mitigate technical risks and uncover issues in upfront engineering. And we aimed to create a digital prototype that is a virtual hardware simulator (VHS).

1.3 Methodology

The Unified Modelling Language (UML) from the Object Management Group (OMG) is used to analyze domain knowledge, use cases, and software architecture. Additionally, a Model-Based System Engineering (MBSE)

methodology was considerably helpful in analyzing interactions and defining information flows between external systems and the VHS.

1.4 Outline

The next chapter, Chapter 2, describes the domain analysis. It is followed by Chapter 3 that explains the problems that we wanted to solve and the goals that we wanted to achieve during this project. The commercial tool exploration is presented in Chapter 4, while Chapter 5 presents stakeholder analysis. In Chapter 6, system requirements, functional and non-functional requirements, and use cases are presented. The following is Chapter 7 that describes the system design. Chapter 8 presents the system architecture. Chapter 9 focuses on the implementation, while Chapter 10 explains the system verification. The following is the conclusion of the project with recommendations. The project management process is explained in Chapter 12. The last chapter of this report (Chapter 13) describes the retrospective and reflection on the project from the author's perspective.

2. Domain Analysis

In this chapter, first, descriptions of TEM is explained. The description covers a brief internal structure and the composition of the microscope. Then it is followed by the TEM development phases, namely design, implementation, and verification.

2.1 Transmission Electron Microscope

The Transmission Electron Microscope (TEM) fires electrons into a specimen, and then the electrons are projected on the Charged Coupled Device (CCD) camera to create an image. The image is a grayscale image that shows the result of the specimen magnification. TEM is able to magnify a specimen up to ten million times. Even a single atom of the specimen can be inspected. Figure 1 shows the appearance of the TEM on the left side and the internal magnification process on the right side.

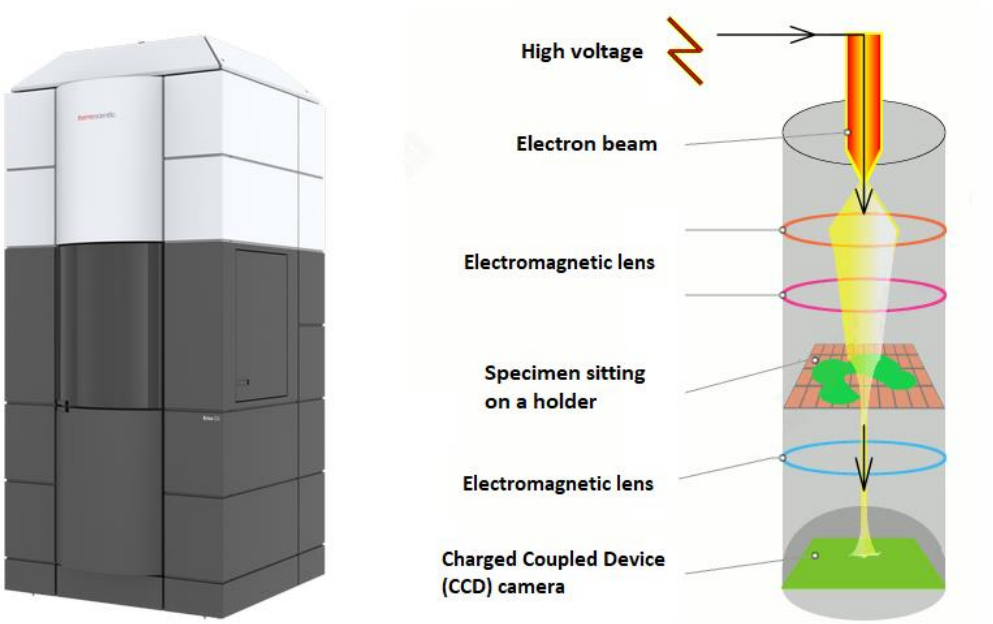


Figure 1: Transmission Electron Microscope

In a complex system like TEM, with the advancement of hardware and software features, testing is intricate. Therefore, TEM is decomposed into microscope components, and each component is designed and developed independently. This independent development gives an advantage in testing components individually.

Figure 2 shows the internal structure of the microscope software system. Several components form a single module. Several modules structure a subsystem. There are four subsystems inside the TEM, namely, *motion*, (electron) *source*, *acquisition*, and *vacuum*.

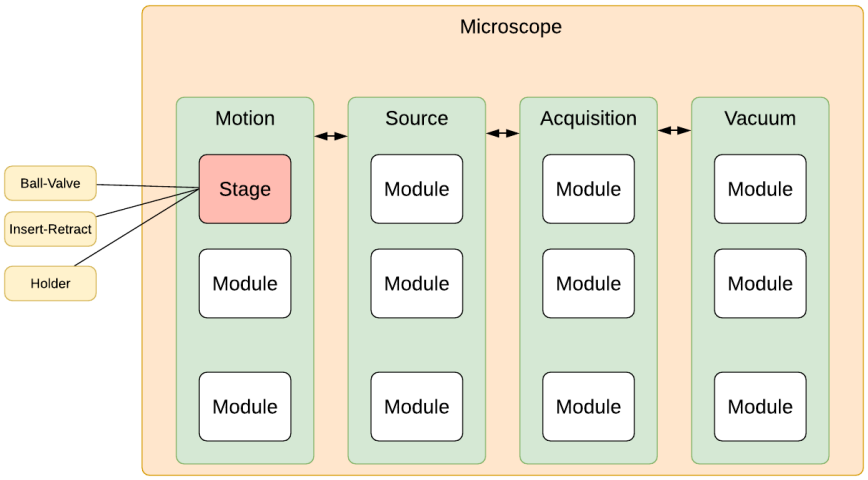


Figure 2: The internal software structure of the TEM server

The *motion* subsystem covers all the modules and components that use actuators and motion sensors. As it is mentioned in the project context, the microscope *stage* is a module, and it is a part of the motion subsystem. The module, *stage*, manipulates a specimen to specify the target for magnification. The *source* subsystem is responsible for accelerating the electron beam. As for the *vacuum* system, it keeps the internal microscope environment in a vacuum state. In this state, electron beams travel without any interference. Otherwise, the electron beams collide with air atoms, resulting in poor performance. Finally, the *acquisition* subsystem is responsible for capturing and transmitting the magnified image. These four subsystems are integrated to form a TEM.

The *stage* has several components. As shown in Figure 2, the *ball-valve*, *insert-retract*, and *holder* are parts of the *stage*. The *ball-valve* is responsible for opening or closing the valve through which the *holder* can be inserted. The *holder* component is responsible for holding a specimen (sample). The *insert-retract* component is responsible for inserting and retracting the *holder*.

The multidisciplinary team regularly adds new features to TEM because clients require new features. The team analyzes requirements from the clients and introduces a new design for additional features to TEM. Each feature might consist of one or more microscope components, and the integration between these components needs to be seamless. Therefore, the design of each microscope component needs to be tested and verified to ensure seamless integration.

2.2 Microscope Development

Figure 3 shows the V-model that is used for microscope development at Thermo Fisher Scientific. On the left side of Figure 3, the system design is decomposed into module designs. Furthermore, each module design is also divided into different component designs. The main output of this phase is to have a complete specification of the microscope design.

Once the design phase is complete, the implementation phase starts. In this phase, the operational software, called *TEM software*, and the hardware controller, called *motion controller*, are being developed. The TEM software is responsible for managing hardware behavior by sending requests to the Motion controller. Then, the motion controller handles hardware motion using actuators and sensors in real time.

The right side of Figure 3 shows integration tests for components, modules, and systems. During the tests, the multidisciplinary team comes together to verify the design. The manufactured hardware from the implementation phase plays a significant role in testing activities. The embedded control firmware and TEM software start to run on the manufactured hardware to check whether the design conforms to the requirements.

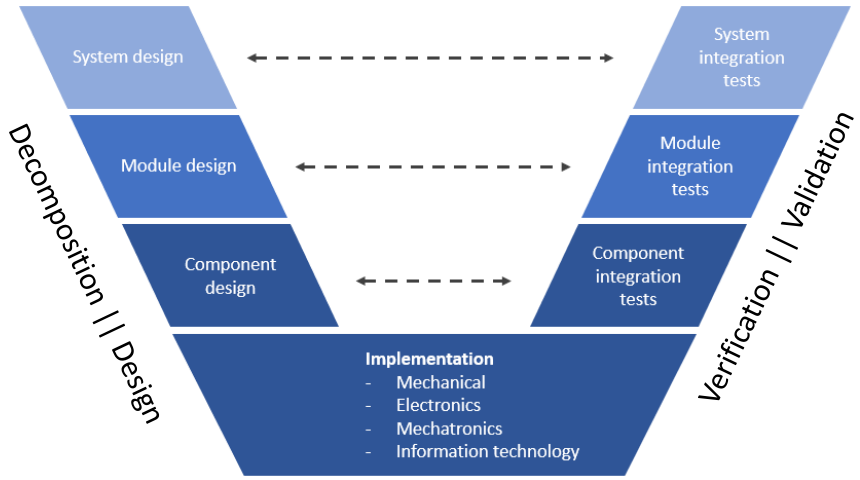


Figure 3: V-Model

Design, implementation, and design verification phases, as shown in Figure 3, are sequential phases. One needs to finish before another phase starts. These phases repeat to improve the design in each iteration. In other words, after the verification phase points out the design defects from the implementation, the design phase starts again to make necessary changes for design improvement.

2.2.1. Design phase

As depicted in Figure 4, the TEM design has three hierarchical levels, namely hardware level, control level, and operational level. In the hardware level, actuator and sensor, which are responsible for detecting and moving microscope components, belong to this level. The next level is the control level. In this level, the motion controller plays a major role in carrying out certain tasks, such as opening a valve and inserting a holder. As for the operational level, it contains the TEM software and its user application. These two are responsible for integrating the system and managing the TEM behavior.

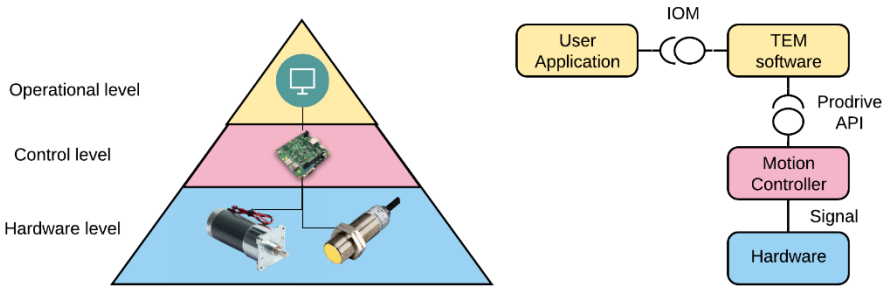


Figure 4: TEM design levels

The user application provides a user-friendly Graphical User Interface (GUI) to users. From the GUI, TEM software receives commands to control the behavior of TEM. The commands are transmitted from the user application over Instrument Object Model (IOM) interface, as shown in Figure 4. The motion controller receives commands from TEM software using device-specific libraries. One of the libraries is Prodrive, and it provides a Prodrive Application Programming Interface (API), which is used to receive *stage* motion commands from TEM software.

2.2.1.1. TEM software

In Figure 5, the structure of the TEM software is depicted. The TEM software is developed in programming language C++, and it has three layers, namely *BeHaVior* (BHV), *MoDuLe* (MDL), and *Hardware Abstraction Layer* (HAL). The BHV layer exposes the Instrument Object Model Interface (IOM) that can be used for user application development. Moreover, the Component Object Model (COM) technology is used to provide compatible interfaces between these layers. For example, the MDL and HAL layers provide two COM interfaces, namely *IMdlMotion* and *IHalMotion*.

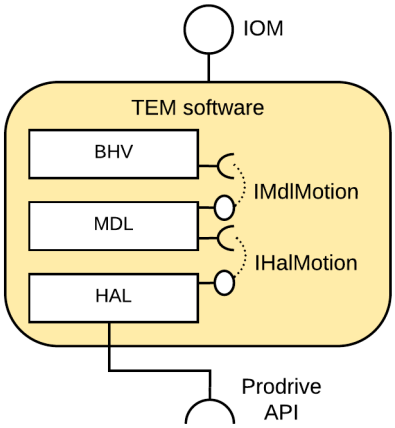


Figure 5: TEM software and its interfaces

2.2.1.2. Motion controller

The motion controller is a closed-loop system, and it manages the motion behavior of hardware. Through the Prodrive API provided by the device-specific library, the motion controller receives a command from TEM software to manage the hardware behavior using peripheral devices such as actuators and sensors. To manage these devices reliably and accurately, the firmware plays a major role.

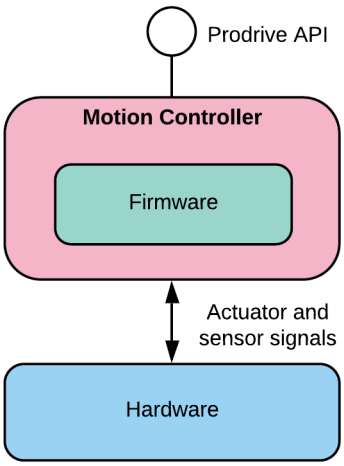


Figure 6: Motion controller

2.2.1.2.1. Device-specific library

As mentioned in the previous section, the library provides API that gives a client application control over hardware. For instance, a client application that uses Prodrive API is able to know the hardware sensor status. Moreover, clients can send commands to the firmware to change the hardware behavior. Currently, Prodrive API is used by TEM software.

2.2.1.2.2. Firmware

Firmware is a machine-specific and real-time software. The firmware uses the Proportional Integral Derivative (PID) control to perform hardware motion accurately with peripheral devices. This PID control has its parameters that are strongly dependent on inertia and torques generated by actuators. Therefore, correct tuning of PID parameters plays a significant role in performing accurate motion tracking. Incorrect parameters may cause abrupt and unexpected motion, resulting in poor performance in sample (specimen) positioning.

To tune the PID parameters, a mathematical modeling approach is used. Mathematical models represent the different parts of the system behavior. For example, the *stage* has three components, namely *ball-valve*, *insert-*

retract, and *holder*. Mathematical models represent each of their behavior. To control their behavior, actuators play a significant role.

At first, a mathematical model that represents the behaviors of the actuator is created. Secondly, the PID parameters are tuned using the model. As an example, Figure 7 shows a mathematical model for Direct Current (DC) motor. In the model, *Vmotor* is an input for voltage supply, and *dAngle/dt* is an angular velocity. The angular velocity implies the speed of the motor rotation. The *angle* refers to the actual position of the motor.

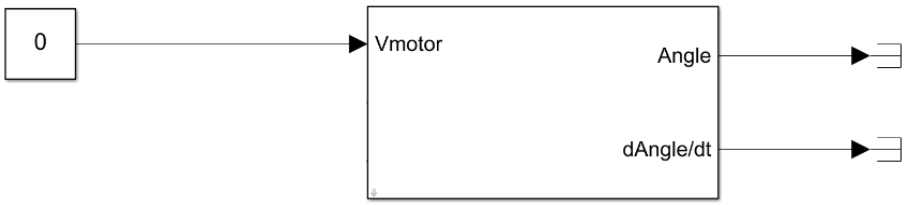


Figure 7: A mathematical model that was created for an actuator

When the PID parameters are set, firmware supplies the value of the voltage to the input of the mathematical model. Then, the output of the model is measured from the model, and this measurement is depicted as a motion graph. Figure 8 shows the measured output of the actuator model. The vertical axis represents the output, *Angle*, in degree, and the horizontal axis represents *time* in seconds.

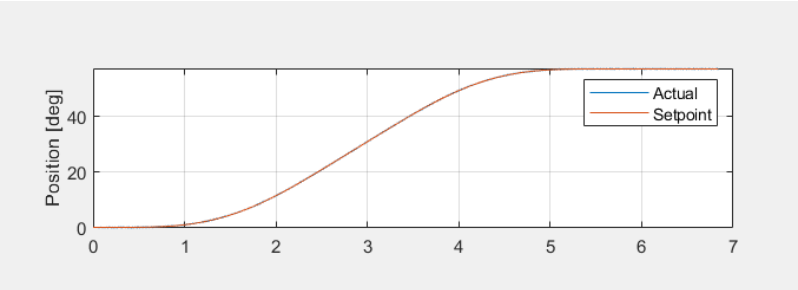


Figure 8: Motion graph of an actuator

Figure 8 shows that the firmware instructed the actuator smoothly, without any abrupt actions. This graph also indicates the PID parameters are configured correctly. Moreover, it is important to note that the firmware that instructs the mathematical model is also used in controlling actual hardware, as shown in Figure 9.

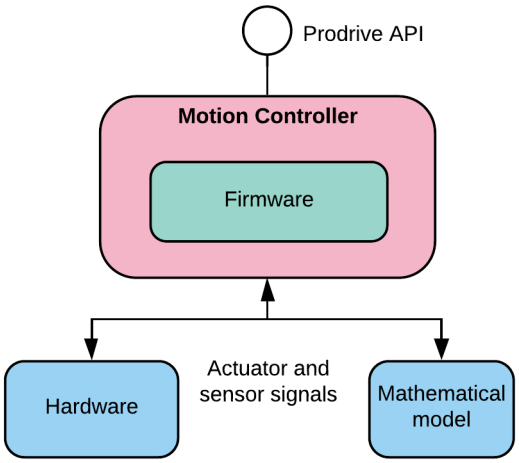


Figure 9: Diagram showing that the same firmware runs on both model and hardware

To summarize, the motion controller is mainly responsible for managing the motion behavior of hardware components. The device-specific library provides an API that establishes a connection to the TEM software. The library is also can be used by other applications, allowing them to monitors signals of the peripheral devices and send data to the firmware. The firmware is the main logic for managing hardware devices such as actuators and sensors. In firmware, PID parameters need to be set correctly to control hardware motion reliably and accurately using actuators and sensors.

2.2.1.3. Mechanical design

Mechanical design means the design of components and systems of a mechanical nature. For example, designs of various machine elements such as shafts, gears, and bearings are a part of the scope of mechanical design.

Computer-Aided Design (CAD) software is used to analyze the design requirements. The mechanical design can be changed multiple times, and each design version represented by a three-dimensional (3D) model.

In the 3D model, geometrics, materials, and names of the mechanical parts are defined. Moreover, the hierarchies of the mechanical parts are also part of the model. To share this model, the software encapsulates them in an assembly file, which is also called a CAD file. There are a few types of the assembly file, and the most used assembly file at the company is the Jupiter Tessellation (JT) file.

Figure 10 shows a modeled mechanical design that was imported into a Unity game engine. On the left side of Figure 10, part hierarchies represent hierarchies structure of modeled parts. Each modeled part has its name and its geometrics. The geometrics and materials are illustrated as 3D models on the right side of Figure 10.

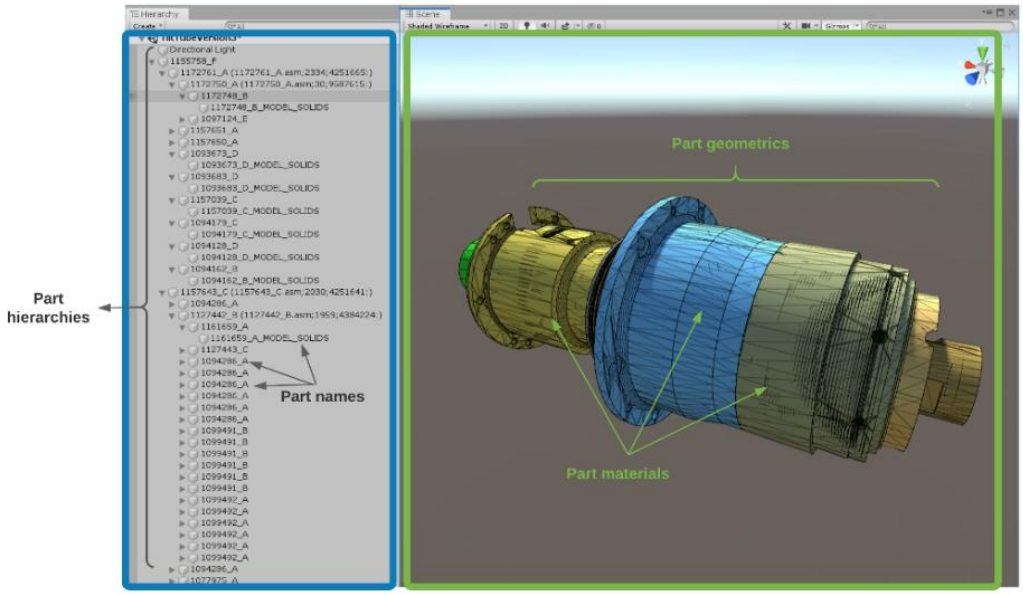


Figure 10. Modeled mechanical design that is imported in Unity game engine

2.2.2. Implementation phase

The implementation phase begins after the design phase. The implementation phase takes place to bring product design into reality. In this phase, complete specifications of hardware are sent to the factory for manufacturing. Moreover, firmware and TEM software expand with new implementations.

Currently, the firmware and TEM software developments are dependent on hardware. When the hardware is manufactured, the firmware and TEM software developments start. Due to the hardware dependency, the firmware and TEM software developments delay until the hardware is manufactured.

2.2.3. Design verification phase

The design verification phase begins after the implementation phase. In this phase, the multidisciplinary team comes together to verify the design. The hardware components are being assembled and integrated. On the motion controller, the firmware is installed, becoming ready to receive commands from TEM software.

During the verification phase, various tests are carried out to check whether the design conforms to the requirements. These tests help detect incompatibilities between hardware and software. Due to the hardware dependency in testing, design flaws and incompatibilities are found lately. As a result, product design needs modification. From the design phase until the verification phase, much time and effort are needed primarily because of hardware manufacturing.

2.3 Conclusion

This chapter provides the necessary domain knowledge about TEM development phases. TEM software structure and the role of the motion controller are explained in the description of the design phase. The second phase is about the implementation. The explanation of this phase gives a brief overview of the manufacturing activities and the dependency on the firmware and TEM software development. The firmware development is depending on the hardware. The implementation of the TEM software begins when both the firmware and hardware development finishes. As a result, the throughput time of the new feature development increases. The last phase is the design verification phase. In this phase, hardware and software are integrated to perform design verification tests. The next chapter explains the problems in detail. It further presents solutions and project goals.

3. Problem Analysis

This chapter firstly presents a problem statement. Secondly, the chapter describes solutions to these problems. The following is the conclusion of this chapter.

3.1 Problem statement

Manufacturing hardware is a time-consuming process. The whole manufacturing process takes around two months. Once the hardware is manufactured, mechanical engineers firstly take the hardware to verify the mechanical design. This verification process takes a couple of iterations. In each iteration, the mechanical design needs changes and retakes some time for manufacturing/adaptation. The verification process lasts until the mechanical design fulfills its requirements. After these lengthy processes, the hardware finally becomes ready, and firmware and software development teams can use the hardware for their development.

3.1.1. Challenges in firmware development

The firmware has its motion control algorithms that accurately manage hardware position and speed. However, implementing these algorithms is an error-prone task because multiple parameters need to be tuned manually. During this process, some error conditions may be triggered, like collisions between mechanical parts. Without hardware, it is hard to know whether collisions take place or not because the detailed interaction between the mechanical parts is mentally hard to imagine. As a consequence, these aspects of the motion control algorithms are not detected until the tests are conducted on the manufactured hardware.

3.1.2. Software development dependencies

Sending requests to the firmware is one of the responsibilities of the TEM software. For instance, the TEM software sends a request to move the *stage* to a specific coordinate. And the firmware takes the request and starts controlling the hardware components. Therefore, testing and development of the TEM software also require hardware. In other words, TEM software development and testing cannot start until the hardware is manufactured. This hardware dependency increases overall software development time.

3.1.3. Limited hardware resources

Due to the limited number of test systems (hardware), firmware and software tests are also delayed. The budget for hardware manufacturing is limited. Thus, a limited number of test systems are produced, resulting in a lack of hardware resources. Test systems are shared within multiple disciplines. Sometimes, test systems are fully occupied by other teams, not allowing firmware and TEM software development teams to proceed.

3.2 Solution

To solve these problems, we introduced a virtual hardware simulator (VHS) that replaces the test systems (hardware). The VHS utilizes the initial 3D CAD models, and it can be created earlier before hardware is manufactured. Using the VHS, the teams can test and develop firmware algorithms and TEM software without the hardware systems.

The VHS uses 3D models of hardware components and simulates them in a 3D environment. In the VHS, the behaviors of 3D models are similar to the behaviors of physical hardware. Physical hardware also has sensors such as proximity and touch sensors. These sensors are simulated and configured on the 3D models. From the 3D environment of the simulator, the interaction between 3D models is visible and testable. Collisions caused by firmware flaws and software flaws are detectable using virtual sensors and collision detection algorithms.

To summarize, the VHS has the following features. The first one is that the VHS provides a visual representation of the physical hardware. With this feature, the teams can have deep insight into how firmware and software instruct the hardware components. Moreover, the interaction between hardware components is visible and testable. The second feature is to simulate sensors. Sensors were placed on the simulator's 3D models to detect collisions caused by software and firmware bugs.

3.3 Conclusion

The problems and their solutions are presented in this chapter. These problems increase the throughput time for feature development. After analyzing these problems, the solution is to make a VHS that can speed up the development phase by simulating microscope components in a 3D environment. To bring the solutions into reality, we explored into commercial software products in the next chapter.

4. Technology Exploration

This chapter introduces different commercial software products and selection criteria. The following is the assessment for each one of them. Finally, the conclusion is present at the end of this chapter.

4.1 Commercial software products

The first step was to investigate commercial software products for creating a virtual/digital version of the hardware. There are several software products on the market, as mentioned in Table 1.

Table 1: Commercial software products

	Software products	**Investigation**
1	PRESPECTIVE [55]	Evaluated and selected
2	Visual Components [55]	Evaluated
3	Solidworks [55]	Evaluated
4	Siemens NX [55]	Evaluated

We defined a set of criteria to choose the one that fits the most in the project in the next section.

4.2 Criteria for selecting software products

This section explains a set of criteria for selecting the software for the project. The criteria are defined in the table below. All of the software products that are mentioned above mainly use CAD models and simulate them in their 3D environment.

Table 2: Criteria for selecting a commercial software product

	Criteria	**Description**
1	Programming language support	The software product should support the following languages C# or C++
2	Performance	The software product should be able to simulate the motion sub-system of the microscope.
3	Computer-Aided Design (CAD) support	The software product should be able to import geometrical models with its color from CAD files
		The software product should import not only geometrical models but also constraints from CAD files.
		The software product should import not only geometrical models but also hierarchies from CAD files.
4	Defining motion behavior	The software product should be able to define motion behavior on CAD models.
5	Defining sensor behavior	The software product should be able to define custom sensor behavior on CAD models
6	Third-party support	The software product should support third-party applications.
7	Building an executable	The software product should build an executable artifact that can be easily deployable.
8	Training	The software product should have available training and documentation.
9	Software support	The company, which develops the software product, should provide support during the project.
10	Cost	The software product price should be reasonable.

The first criterion is that a chosen software should support C# or C++. These two programming languages are mainly used in the company.

Although we created virtual hardware for the stage module of the microscope, the current implementation needs to expand with other parts of TEM. Finally, the whole microscope needs to be simulated. TEM is big and complex.

Simulating TEM might require a heavy calculation and reliable performance. Therefore, performance is the next criterion.

The visualization is one of the main requirements. Therefore, a chosen software must be able to use CAD models in its 3D environment. The models are static, and CAD files do not include any information about motion constraints. Thus the software must be able to define motion behavior in its environment.

Apart from the previous criterion, the behavior of sensors needs to be defined by a chosen software. The *stage* has proximity sensors and touch sensors, and they have their unique behaviors. Therefore, the behavior of sensors must be defined to fulfill the requirement.

Integration with the TEM software is crucial to perform real-time simulation. Therefore, the chosen software needs to connect to TEM software. Over the connection, simulated motion and sensor data must be transmitted in real time. Therefore, another criterion is third-party support.

The seventh criterion is about building an executable artifact. Availability is one of the non-functional requirements. After creating a VHS using the chosen software product, the simulator should be easily deployable to the team. Therefore, the simulator should be delivered as an executable artifact.

Learning a new specialized software requires effort. Therefore, training materials and courses can accelerate the learning and implementation process. Moreover, close support and feedback from the company, which develops the software product, can speed up the project development process.

The criteria that are explained above are discussed for each software individually. The result of the discussion is present in the section below.

4.2.1. PRESPECTIVE

PRESPECTIVE is a software platform for creating a digital twin. It is developed on top of the Unity game engine that uses the Mono development platform. Table 3 shows the evaluation result according to the criteria.

Table 3: PRESPECTIVE investigation

	Criteria	Description	Satisfaction
1	Programming language support	The software supports C# language.	Yes
2	Performance	-	-
3	Computer-Aided Design (CAD) support	PRESPECTIVE import CAD files using additional software called PIXYZ plugin.	Yes
4	Defining motion behavior	PRESPECTIVE has features that are used to define motion behavior using kinematics relation.	Yes
5	Defining sensor behavior	Proximity and touch sensor behaviors are created using Unity C# scripting.	Yes
6	Third-party support	The software is able to be integrated with .Net applications.	Yes
7	Building an artifact	The unity game engine builds an executable artifact. Therefore, no licenses are needed to deploy the artifacts.	Yes
8	Training	The company conducts regular training. Documentation and video tutorials are also provided online.	Yes
9	Software support	The company provides close support and fast feedback. Moreover, face-to-face meetings can be organized to get support from the company.	Yes
10	Cost	Reasonable (1000-5000 euros per year for a single user)	-

Based on the defined criteria, PRESPECTIVE scores well on all requirements.

4.2.2. Visual Components

The software is specially designed for simulating factory. The software provides .Net and Python API to third-party applications. Table 4 shows the investigation results in respect of the criteria.

Table 4: Visual Components investigation

	Criteria	Description	Satisfaction
1	Programming language support	The software supports C# and Python languages.	Yes
2	Performance	-	-
3	Computer-Aided Design (CAD) support	The software has a built-in feature to import CAD files with its hierarchies.	Yes
4	Defining motion behavior	Motion behavior can be defined in the CAD model using Python scripting.	Yes
5	Defining sensor behavior	Sensor behavior can be defined using Python scripting.	Yes
6	Third-party support	The software provides .Net and Python API.	Yes
7	Building an artifact	The software does not generate an artifact. Therefore, licenses are needed to run the VHS on developer machines.	No
8	Training	The company has an online forum and online courses.	Yes
9	Software support	-	-
10	Cost	-	-

4.2.3. Solidworks

Solidworks is one of the popular simulation software products. Therefore, Solidworks is also investigated. Figure 5 shows the result of the investigation.

Table 5: Solidworks investigation

	Criteria	Description	Satisfaction
1	Programming language support	The software supports C#, C++, and Visual Basic languages.	Yes
2	Performance	-	-
3	Computer-Aided Design (CAD) support	The software has built-in features to import CAD files with its hierarchies.	Yes
4	Defining motion behavior	The software defines motion behavior in its environment.	Yes
5	Defining sensor behavior	The software defines sensor behavior in its environment.	Yes
6	Third-party support	It provides a Solidworks API.	Yes
7	Generating artifact	The software does not generate a standalone executable file. Therefore, licenses are needed to run the VHS on developer machines.	No
8	Training	The company does not provide enough available information about the product API for real-time simulation.	No
9	Software support	Support is provided through online courses and forums.	-
10	Cost	Moderate (5000-10000 euros per year for a single user), according to the cati.com.	-

4.2.4. Siemens NX

Siemens NX is used as the main mechanical drafting tool within Thermo Fisher. The software also can be used for motion simulation.

Table 6: Siemens NX investigation

	Criteria	Description	Satisfaction
1	Programming language support	The software supports Java, C#, C++, Python, and Visual Basic languages.	Yes
2	Performance	-	-
3	Matlab/Simulink support	The software has direct support for Matlab/Simulink.	Yes
4	Computer-Aided Design (CAD) support	The software has built-in features to import CAD files with its hierarchies.	Yes
5	Defining motion behavior	The software defines motion behavior in its environment.	Yes
6	Defining sensor behavior	The software defines sensor behavior in its environment.	Yes
7	Generating artifact	The software does not generate a standalone executable file. Therefore, licenses are needed to run the VHS on developer machines.	No
8	Training	The company does not provide enough available information about the product API for real-time simulation.	No
9	Software support	The company provides close support because engineers from Siemens works at Thermo Fisher once a week.	Yes
10	Cost	Expensive (more than 10000 euros for a one-time purchase for a single user).	-

It provides API that is used to communicate with custom applications. Defining motion and sensor behaviors on the 3D models were successful. However, getting the sensor value into third-party applications has not been tested and investigated due to time limitations.

4.3 Selecting a simulation software product

Based on the criteria, PRESPECTIVE was chosen as the software for the simulation. The primary reasons to choose PRESPECTIVE are the following:

- The software product builds a standalone executable artifact while other software products do not. This building activity can be integrated with the company version control system. In other words, any engineer can obtain the executable artifact and use it locally for testing purposes.

- The software product can be easily integrated with the TEM software because the Unity game engine can use native COM components.

- The software product allows customizing its GUI that can be modified for multiple uses. For instance, categorized strip menus or buttons can be created for automating the testing activities. And a pop-up notification can be shown as a simulation result.

- The price of the software product is reasonable because of two reasons. The first one is that Unity builds an executable artifact that can be deployed for multiple users. The second reason is that a single user license costs less than other software products listed in Table 1 (excluding Visual components).

- The company that develops PRESPECTIVE provides close support and provide documentation and tutorials.

The Unity game engine is not able to import CAD files into its environment. Therefore, the following options were investigated. There are two plugins for the Unity game engine that can fill the gap.

- **PIXYZ plugin**: The plugin can import most of the CAD files into the game engine environment. The most used CAD files in the company are JT, STP, and X_T.

- **CADlink:** This plugin supports the 3MF file format, which is not commonly used in the company.

We chose PIXYZ as the CAD files importer tool because it supports the necessary CAD files, while CADlink only supports 3MF format.

4.4 Conclusion

This chapter explained the various commercial software packages. After evaluating and comparing the software products based on the defined criteria, PRESPECTIVE software was chosen to create a VHS. The chosen software uses 3D models from CAD files. The next chapter presents the stakeholder and their concerns.

5. Stakeholder Analysis

This chapter presents the concerns and the involvement of the main stakeholders. In this project, two significant organizations are involved, namely Thermo Fisher Scientific and Eindhoven University of Technology (TU/e).

5.1 Thermo Fisher Scientific

During this project, the collaboration with the motion division of Thermo Fisher Scientific was crucial to gather the necessary knowledge and project requirements. The main stakeholders of Thermo Fisher Scientific are listed below.

Table 7: Company stakeholder 1

Name	Pepijn Kramer
Role	Software Architect/ Project supervisor
Goals	- Convey the gathered knowledge during this project to follow up project(s) and engineers - Evaluate the implementation and the design of the project - Identify the benefits of the digital twinning concept in software development
Tasks	- Provide relevant information about the software stack and the organization - Provide technical advice on the project - Provide advice on the system design - Help evaluate the project progress - Discuss implementation ideas with the trainee - Provide advice on software design/architecture in general - Review the project's final report
Involvement	- Weekly meeting - Monthly project steering group meeting - Ad-hoc meeting

Table 8: Company stakeholder 2

Name	Iryna Stepaniak/ Project supervisor
Role	Test and quality assurance
Goals	- Convey the gathered knowledge during this project to another project
Tasks	- Provide relevant information about the testing and the organization - Provide organizational support to the trainee - Evaluate the project progress
Involvement	- Weekly meeting - Monthly project steering group meeting - Ad-hoc meeting

Table 9: Company stakeholder 3

Name	Edwin Verschueren
Role	Mechatronics specialist
Goals	- Define virtual hardware simulator (VHS) capabilities for future uses - Create a VHS that can replace physical hardware - Find an appropriate development process to create a VHS
Tasks	- Provide relevant information about TEM development - Evaluate the project progress - Review the project's final report
Involvement	- Weekly meeting - Monthly project steering group meeting - Ad-hoc meeting

Table 10: Company stakeholder 4

Name	Paul de Brabander
Role	Test engineer
Goals	- Test the microscope system without relying on hardware
Tasks	- Provide relevant information about the current testing methods
Involvement	- Ad-hoc meeting

Table 11: Company stakeholder 5

Name	Arjen Klomp
Role	Senior R&D manager
Goals	- Find the business benefits out of this project - Provide trainee organizational support
Tasks	- Support on ordering the necessary commercial tools - Provide information about the company organization
Involvement	- Monthly project steering group meeting - Ad-hoc meeting

5.2 TU/e

The key stakeholders from TU/e are listed below.

Table 12: TU/e stakeholder 1

Name	Ion Barosan
Role	Assistant professor at the Faculty of Mathematics and Computing Science of TU/e / project supervisor
Goals	- Evaluate the project based on PDEng's project standard - Evaluate the project progress - Guide the trainee through the project
Tasks	- Provide relevant knowledge - Monitor project's process - Evaluate the progress regularly and provide advice and feedback to the trainee - Review the project's final report
Involvement	- Monthly project steering group meeting - Ad-hoc meeting - Weekly meeting

Table 13: TU/e stakeholder 2

Name	Yanja Dajsuren
Role	Software technology PDEng program director
Goals	- Project success
Tasks	- Provide advice to the trainee
Involvement	- Ad-hoc meeting

5.3 Conclusion

This chapter presents the stakeholders and their roles, goals, tasks, and involvement in this project. Concerning their goals, we identified system requirements, functional requirements, non-functional requirements, and use cases in the next chapter.

6. Analysis – Requirements, System Context, and Use Cases

This chapter firstly presents requirements. The followings are system context and use cases. Finally, the conclusion is present at the end of this chapter.

6.1 Introduction

The first endeavor of this project was to gather detailed requirements from the users: a mechatronics engineer and a software test engineer. During the requirements gathering process, a bottom-up approach was helpful. A small prototype was periodically made to show simulator capability. This approach helped to communicate with the stakeholders and triggered ideas for further requirements.

6.2 System requirements

We defined two high-level features for the virtual hardware simulator (VHS). These two are presented in Figure 11.

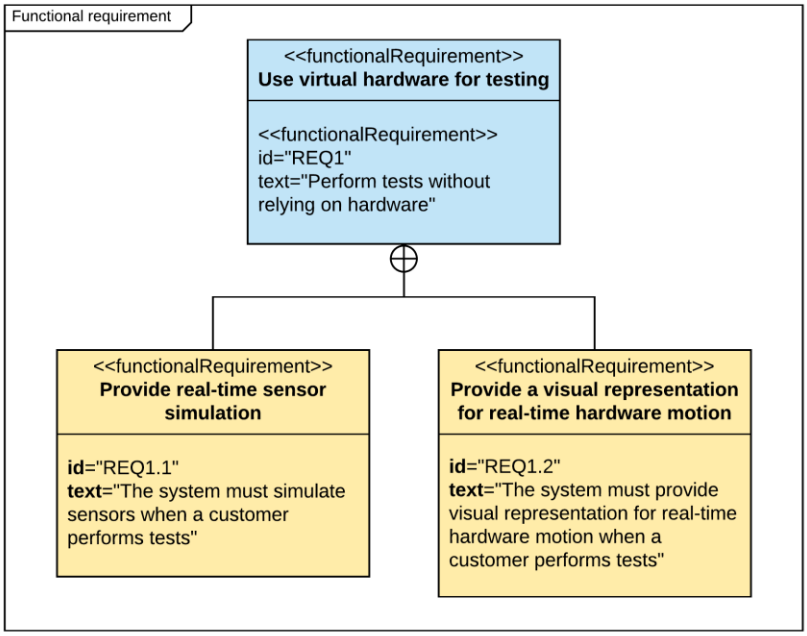


Figure 11: System requirements

As stated in the project context, we aimed to create a VHS for a microscope module, *stage*, that manipulates a specimen. And the module consists of several microscope components, and we used three of them, namely *ball-valve*, *insert-retract*, and *holder*. Therefore, detailed requirements and use cases were identified based on the functionality of these components.

6.2.1. REQ1.1-Provide real-time sensor simulation

One of the requirements that the system must have is that the system must simulate sensors in real time when a customer performs tests. Firstly, different types of sensors of the *stage* needed to be specified to fulfill this requirement. The *stage* has two kinds of sensors: proximity sensors and touch sensors, which are the essential parts of the *stage*. These two sensors must be virtually placed in the system and simulated in real time. Table 14 shows the defined requirements for the stage.

Table 14: Functional requirements to perform real-time sensor simulation

ID	Topic	Priority

1	The system must provide real-time simulation for the proximity sensors of the ball-valve component	Must have
2	The system must provide real-time simulation for the touch sensors of the ball-valve component	Must have
3	The system must provide real-time simulation for the touch sensors of the insert-retract component	Must have
4	The system must provide real-time simulation for the touch sensor of the holder component	Must have

The responsibility of the proximity sensors is to know the status of the hardware. For example, the proximity sensors of the ball-valve component are designed to detect whether the valve is open or not while the touch sensors are to detect collisions between hardware parts.

6.2.2. REQ1.2-Provide a visual representation for real-time hardware motion

Each microscope component (hardware) has a distinct motion behavior. Behaviors of these microscope components must be visible to the users in real-time when they run tests on the VHS. Therefore, the following requirements are needed. Table 15 shows the defined requirements.

Table 15: Functional requirements to visualize real-time motion behavior

ID	Topic	Priority
1	The system must visualize the motion behavior of the ball-valve component in real time when a customer performs tests	Must have
2	The system must visualize the motion behavior of the insert-retract component in real time when a customer performs tests	Must have
3	The system must visualize the motion behavior of the holder component in real time when a customer performs tests	Must have

6.3 User requirements

During the project, the users were a mechatronics engineer and a test engineer. The requirements from the users are presented in Table 16.

Table 16: User requirements

ID	Topic	Priority
1	The system must provide a flexible viewpoint in order to see the virtual hardware from a different side	Must have
2	The system must have a configuration file in order to configure the simulation	Must have
3	The system should disable/enable the proximity sensors in order to test fault scenarios scenarios	Should have
4	The system should trigger the proximity sensors in order to test the software behavior	Should have
5	The system should show the part name of the hardware in order to inform the part name	Should have

The first requirement came from a mechatronics engineer. The mechatronics engineer wants to inspect the *stage* components from different sides while the virtual hardware (3D models) are moving in the simulator. To fulfill this requirement, the system must provide users an interactive GUI that allows the users to change the viewpoint. For example, zooming in/out, rotating, and panning the viewpoint are the basic functionalities that the system must provide.

The mechatronics engineer wants to use a configuration file to set up the simulation. With the file, the engineer is able to choose microscope components they would like to simulate. For example, the file configures the system to simulate only a ball-valve component and its sensors and actuators. Moreover, the file is also used for other purposes, such as setting up the simulation speed. For these reasons, the second requirement was needed.

The third requirement came from the software test engineer to test fault scenarios. Disabling and enabling sensors means that the user virtually breaks or fixes them. Once the sensors are disabled, they no longer give feedback. In

that case, the system performance with the broken sensors can be inspected. When they are enabled, the sensors perform as expected.

Triggering sensors from GUI is another requirement that came from the software test engineer. This requirement is to test whether the software handles the sensor data correctly or not.

The fifth requirement is about the visibility of the stage's names, which have to be visible when the mouse hovers over the 3D design. This feature is useful when flaws from the mechanical design are inspected. The name of a part that causes incompatibility should be visible.

6.4 Non- functional requirements

This section describes the non-functional requirements that are defined during project development time.

6.4.1. Compatibility

The non-functional requirement of compatibility is that the system must be compatible with the existing software system to perform tests. Secondly, the system must be delivered as a software component. To deliver the VHS as a software component, the VHS must follow the build server rules.

6.4.2. Extensibility

In this context, extensibility means the extension of the functionality and improvement in the services. Modifying the VHS and understanding its source code must be comfortable. In order to meet this requirement, the simulation should have a modular software architecture. The implementation of the simulator should use SOLID principles.

6.4.3. Availability

Availability means the extent to which a software product is easily distributable and deployable. In other words, the VHS needs to be delivered as an executable artifact. The installation process of the simulator should not be dependent on additional software products.

6.5 System Context

After defining the requirements, we analyzed the external systems that need to interact with the virtual hardware simulator (VHS) to fulfill the requirements defined above. This section explains information flows between external systems and the VHS.

6.5.1. System context

The external systems that interact with the VHS are shown in the system context diagram, Figure 12. The diagram illustrates two actors and three external systems that directly and indirectly communicate with the VHS.

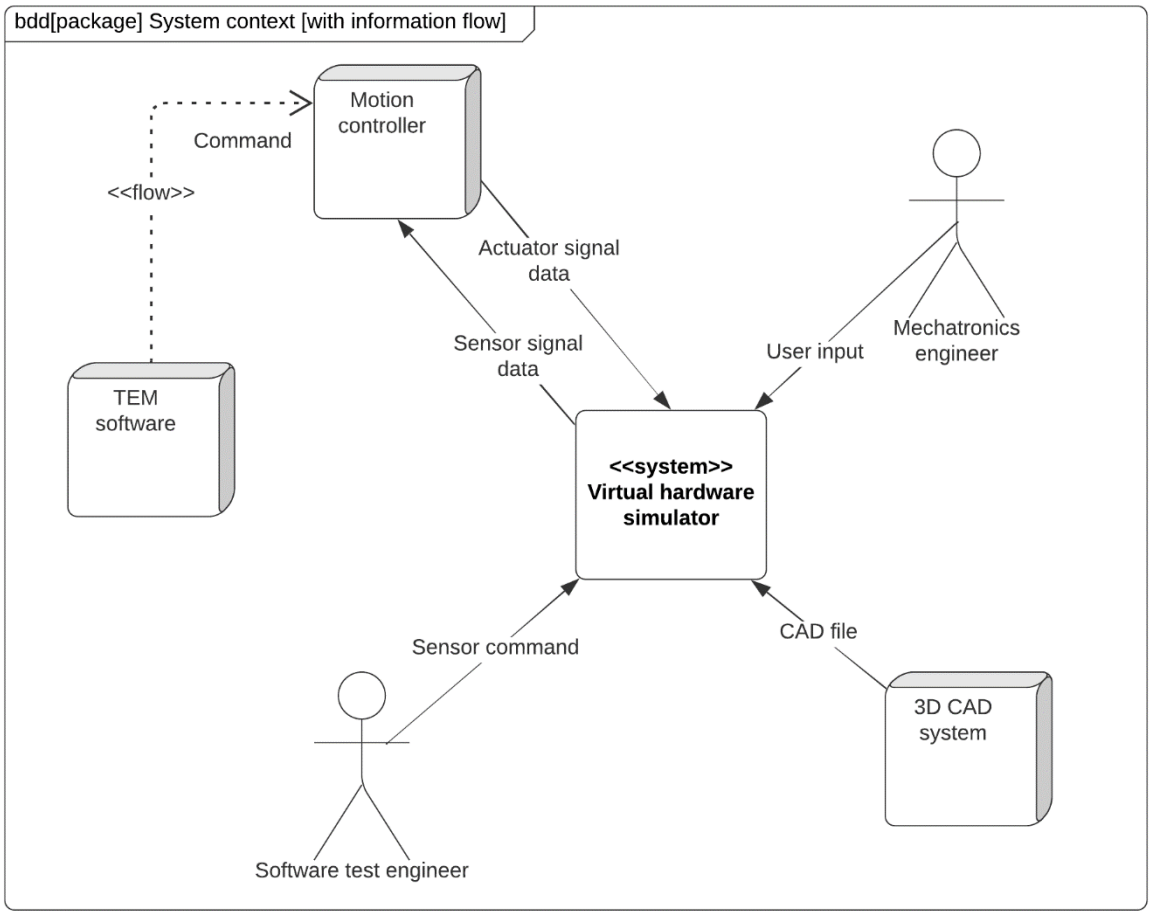


Figure 12: System context diagram

6.5.2. Information Flow

6.5.2.1. Command

A command is an indirect information flow for the VHS. TEM software sends commands to the motion controller to control the behavior of the 3D model in the VHS.

6.5.2.2. Computer-Aided-Design (CAD) file

The VHS uses 3D models to create virtual hardware. These 3D models are encapsulated in CAD files that come from the 3D CAD system, as shown in Figure 12. The simulator imports these files into its environment to obtain the 3D models.

We aimed to create a virtual hardware simulator for three hardware components, namely *ball-valve*, *insert-retract*, and *holder*. Therefore, we need three models as inputs from the 3D CAD system, as illustrated in Figure 13.

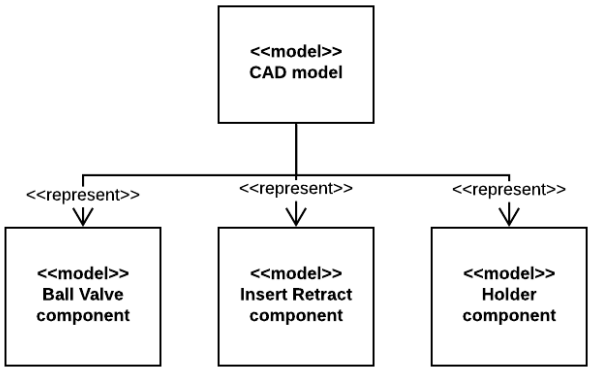


Figure 13: CAD models

6.5.2.3. Actuator and sensor signal data

In Figure 14, the motion controller holds all the signal data of the hardware. Using Prodrive API, the VHS can read and write these signal data to the motion controller. As shown in Figure 14, actuator signal data represents data of *ball-valve* and *insert-retract* components. These data are obtained from the motion controller and applied onto 3D models in the VHS.

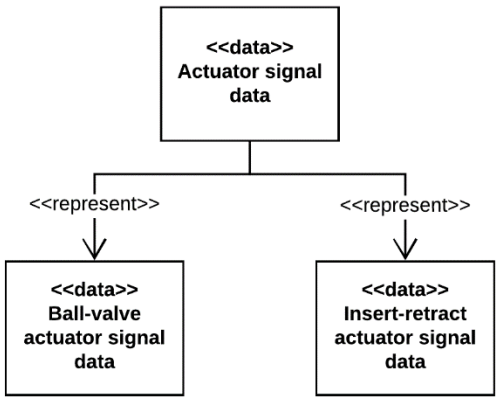


Figure 14: Actuator signal data

Virtual sensors are placed on the 3D models. When these sensors are triggered, sensor signal data are also transmitted back to the motion controller as feedback. Currently, the VHS has two types of sensors: touch sensor and proximity sensor.

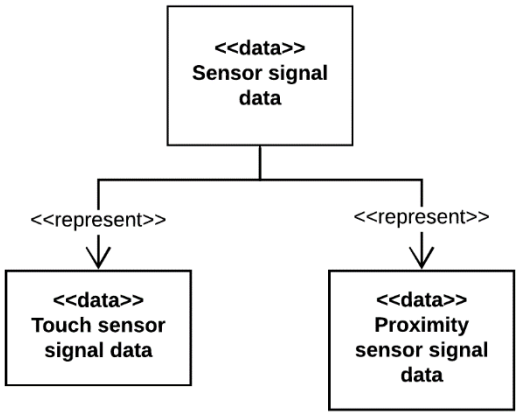


Figure 15. Sensor signal data

6.5.2.4. User input

User input is used for two purposes. The first one is to establish a connection to the motion controller from the VHS. To establish the connection, the name and Internet Protocol (IP) address of the motion controller need to be provided to the simulator. Over the connection, sensor and actuator signal data can be exchanged with the VHS. The second purpose is to choose sensors and actuators that need to be involved in the simulation. Therefore, their specification needs to be provided to the simulator.

The sensor specification contains the list of virtual sensors that need to be involved in the simulation. As for actuator specification, it contains a list of actuators list that also need to be involved in the simulation. Figure 16 shows the user input diagram.

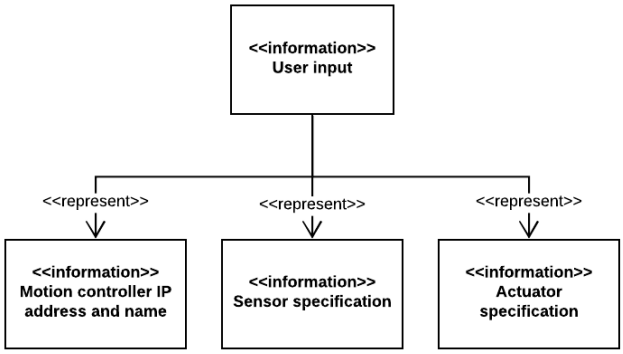


Figure 16: User input

6.5.2.5. Sensor command

A software test engineer sends sensor commands to the VHS to test fault scenarios. For instance, breaking a sensor virtually in the simulator is one of the tests. Therefore, the system needs to receive sensor commands to disable or enable virtual sensors. Figure 17 shows two types of commands for sensor manipulation.

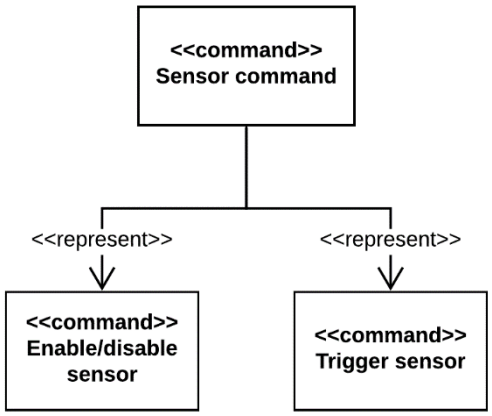


Figure 17: Sensor command

This section introduced the information flow between the external systems and the VHS. As described above, a mechatronics engineer provides the motion controller address to the VHS. With this input, the VHS can establish a connection to the motion controller. Through the connection, sensor and actuator signal data are exchanged. The mechatronics engineer also provides sensor and actuator specifications to the VHS to set up the simulation. Moreover, a software test engineer provides the VHS sensor commands to manipulate virtual sensors. The 3D CAD system provides 3D models to the VHS. Using the 3D models, the VHS can visualize and simulate them in its 3D environment.

6.6 Use cases

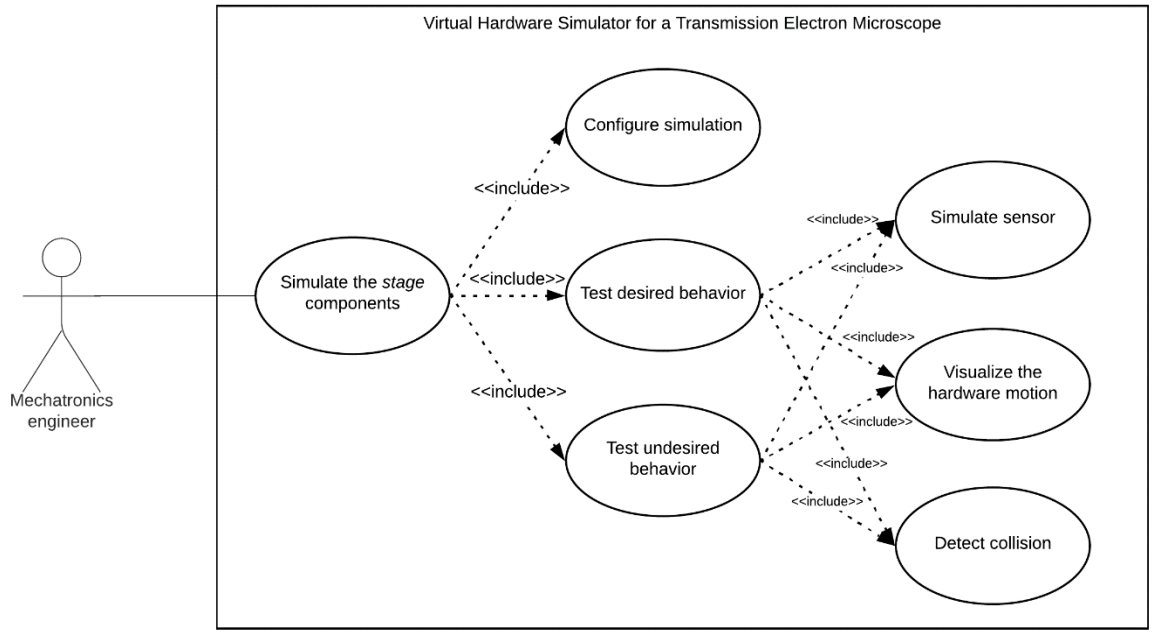


Figure 18: A use case diagram for a mechatronics engineer

The primary user of the VHS is a mechatronics engineer. The primary use case of the VHS is *simulate stage components*. This use case includes three other use cases, namely *test desired behavior, test undesired behavior*, and *configure simulation*, as shown in Figure 18.

The use case, *configure simulation* is about setting up the VHS to establish a connection to the motion controller before the simulation starts. In this use case, the VHS utilizes user inputs, as defined in Section 6.5.2.4.

The use case, *test desired behavior* is about testing the defined behaviors of the *stage* components. Each component has its distinct behavior, and the firmware must control it correctly. The purpose of this use case is to test how the firmware controls the desired behavior of the hardware.

Another use case is called *test undesired behavior.* It is about testing undesired scenarios virtually instead of carrying out these scenarios physically. The undesired scenario usually causes a collision between hardware components. The purpose of this use case is to test how the firmware reacts when undesired scenarios take place.

The use case, *simulate sensors*, is to test the firmware. While hardware parts are moving, sensors on the hardware are triggered, and the firmware must handle the triggered sensor data correctly.

The use case, *visualize the hardware motion*, is about visualizing the hardware motion using 3D models. These models are provided by the 3D CAD system. When the mechatronics engineer executes the firmware algorithms to control the hardware components, the system must visualize how the firmware instructs the motion behavior of the hardware. With this visualization, the mechatronics engineer can inspect the interaction between hardware components.

Another use case is called *detect collision* is presented in Figure 18. For this use case, the touch sensors of the component play a major role.

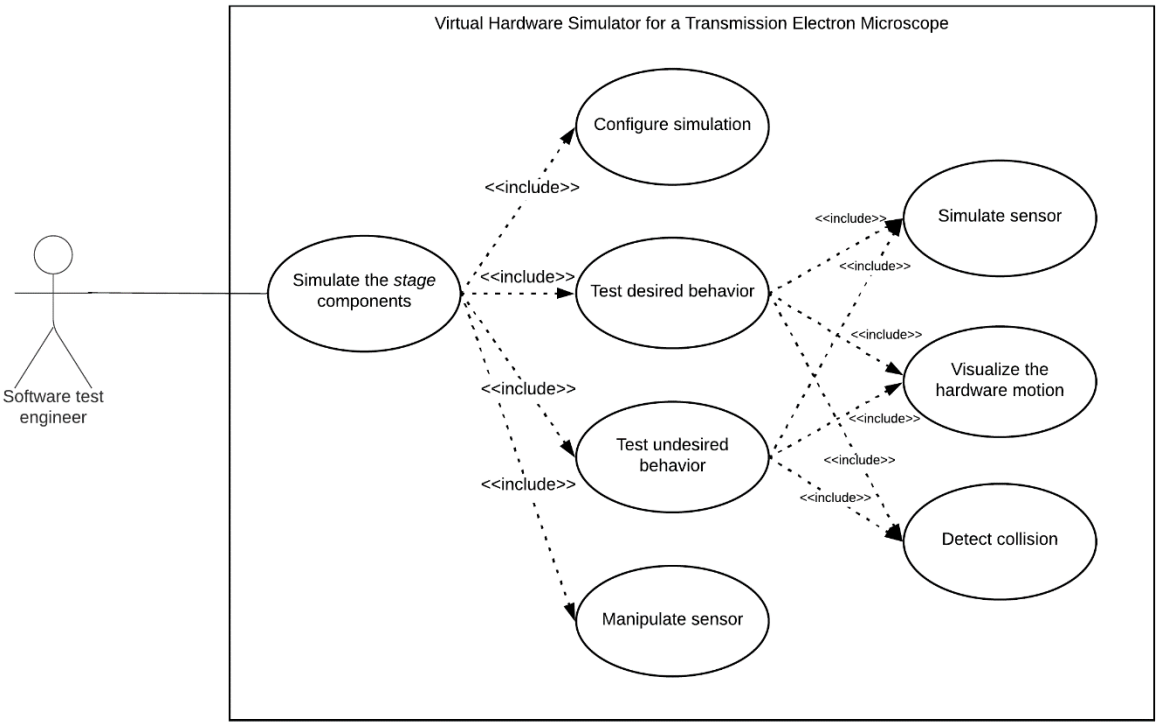


Figure 19. A use case diagram for a test engineer

Another user of the VHS is a software test engineer, as depicted in Figure 19. There is only a different use case compared to the use cases from mechatronics engineer. This use case is *manipulate sensors*. It is about disabling, enabling, and triggering the stage's sensors from the VHS using sensor commands, as defined in Section 6.5.2.5. The purpose of this use case is to utilize virtual sensors to test system behavior.

6.7 Conclusion

In this chapter, system requirements, user requirements, non-functional requirements, and system context are explained. In addition to that, the use cases that came from the users are also described in this chapter. The next chapter is about system design.

7. System Design

In the previous chapters, we chose the technology to develop a virtual hardware simulator (VHS) and defined the use cases and requirements. In this chapter, the design of the VHS is explained. Firstly, the design choices are introduced. The followings are the design decision and conclusion.

7.1 Introduction

Two steps were defined in designing a VHS. The first step was to identify what are the mandatory inputs in creating virtual hardware in the Unity environment. From the environment, the motion behavior of the hardware must be visible according to the requirements that are mentioned in Section 6.2.2. The second step was to integrate the virtual hardware with the existing software stack. By integrating them, TEM software is able to control the motion behavior of the virtual hardware. At the same time, virtual sensors must be simulated according to the requirements that are mentioned in Section 6.2.1.

7.2 Step 1

7.2.1. Creating virtual hardware

The simulator must visualize the motion behavior of hardware. Therefore, we needed 3D models of hardware components. These models are imported into a simulation environment of the Unity game engine. On the imported 3D models, motion behaviors are defined, and virtual sensors are also placed.

To obtain 3D models, CAD files are used. Figure 20 shows the process of importing 3D models from CAD files into the Unity game engine using the PIXYZ plugin. On the imported models, PRESPECTIVE defines kinematics relation and its constraints.

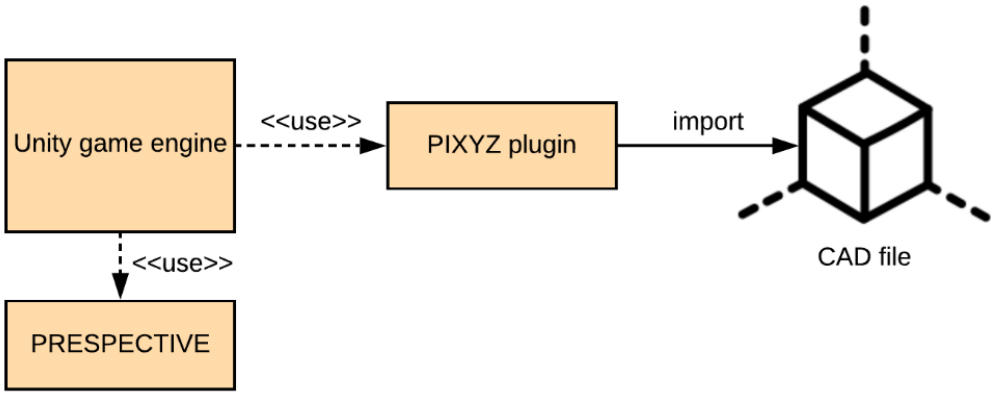


Figure 20: Process of importing 3D models into PRESPECTIVE environment

The main output of this step is to have virtual hardware that has the same behavior as a physical one.

7.3 Step 2

7.3.1. Design Choice 1

As described in Section 2.2.1.1. , two COM interfaces, namely *IHalMotion* and *IMdlMotion*, are provided by MDL and HAL layers, respectively. These interfaces can be utilized to instruct the 3D models of the VHS. Both interfaces support the C# programming language.

Firstly, we assessed the design when using the *IMdlMotion* interface. A plugin is needed to transfer commands from TEM software to virtual hardware, as shown in Figure 21. It implements the interface and creates a new COM object. Then the object is called by the TEM software to control the behavior of 3D models of the VHS.

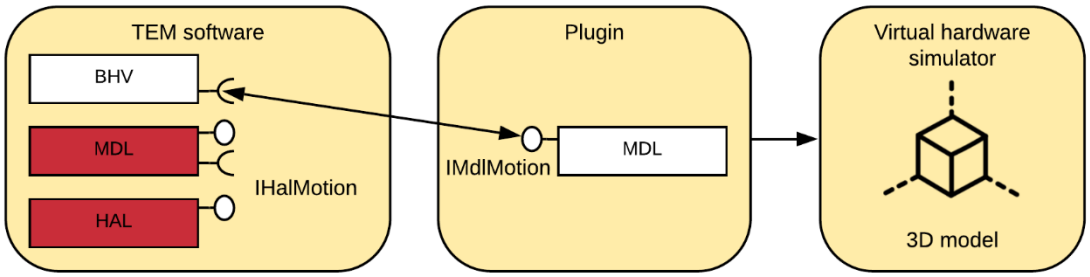


Figure 21: Simulation diagram when using *IMdlMotion*

However, this design has disadvantages. The existing implementations of the MDL and HAL layers in the TEM software are not involved in the simulation. Moreover, real-time hardware motion and sensor simulation are not possible because the interface is abstract and does not involve data about sensors and actuators signals.

Secondly, we assessed the design when using the *IHalMotion* interface. A plugin is also needed to transfer commands from TEM software to virtual hardware. It implements the interface and creates a new COM object. The object is called by the MDL layer of TEM software, as shown in Figure 22.

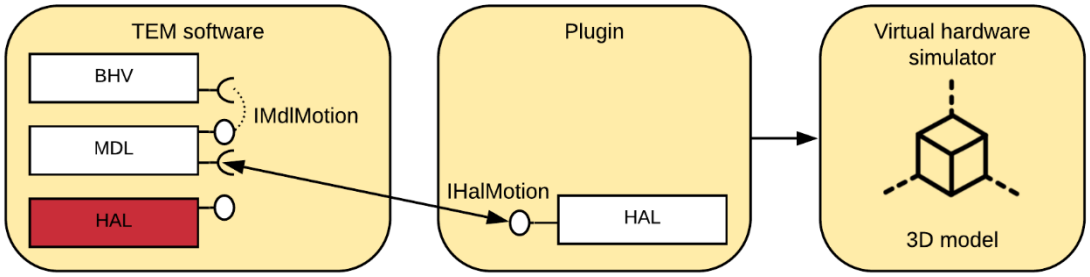


Figure 22: Simulation diagram when using *IHalMotion*

In that case, the existing implementation of the HAL layer in TEM software is not involved in the simulation. The virtual hardware can be used only for testing MDL and BHV layers. Moreover, the interface provided by the HAL layer does not allow the VHS to control the behaviors of 3D models in real time because the *IHalMotion* does not involve real-time data. Moreover, the HAL layer in TEM software was in the process of development during this project. Therefore, *IHalMotion* was not ready to be used.

7.3.2. Design Choice 2

The next design choice was to utilize the motion controller. The motion controller has two modes: non-simulated and simulated. In the non-simulated mode, the motion controller interacts with the physical hardware through signal interfaces. Through these signal interfaces, actuator and sensor signal data is exchanged. In the simulated mode, the motion controller interacts with the mathematical model that represents the actuator and sensor behaviors. These two modes are shown in Figure 23.

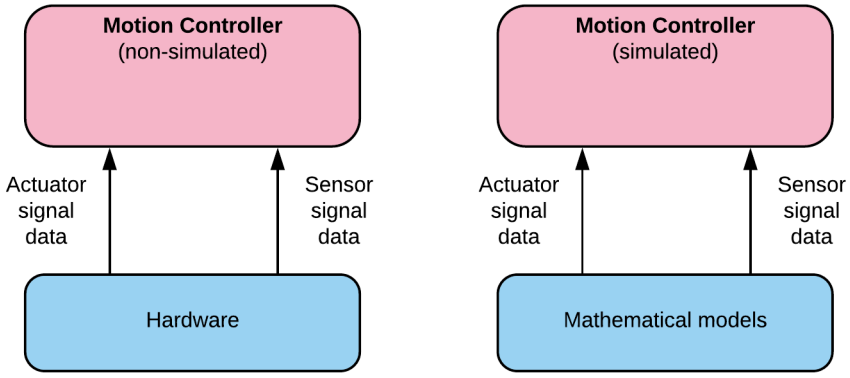


Figure 23. Motion controller modes

At first, we assessed using the simulated motion controller. By combining mathematical models with 3D models of the VHS, the simulator can have real-time motion visualization because the mathematical models are instructed in real time by the motion controller. In Figure 24, the ball-valve actuator is represented by a mathematical model. While the motion controller instructs the mathematical model, the VHS takes the real-time signal data of the actuator model and applies it onto the 3D model of ball-valve. While 3D models move using the signal data, virtual sensors are triggered. The sensor signal data is also sent to the motion controller as feedback in real time.

Mathematical models are created during the design phase. These models for the *stage* components were ready to be used in this project.

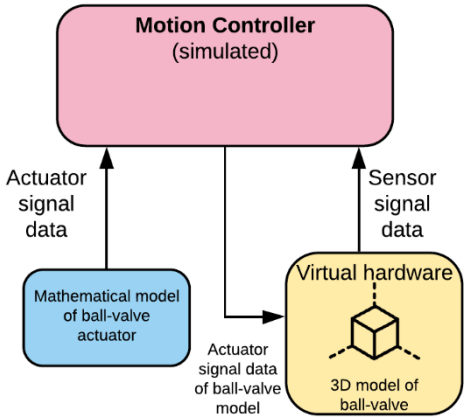


Figure 24: An example diagram for ball-valve component

The design also gives other possibilities. Physical hardware can be visualized using 3D models. By utilizing real-time actuator signal data of physical hardware and applying the data onto 3D models, the VHS can visualize the hardware status, as shown in Figure 25.

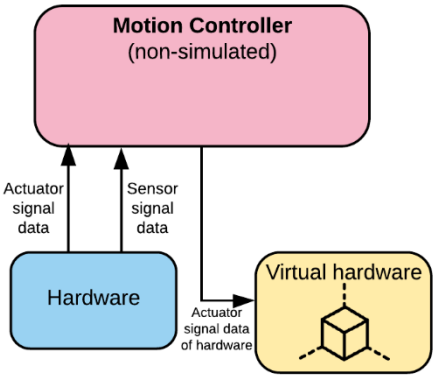


Figure 25: Visualizing physical hardware in a virtual environment

7.3.3. Design Decision

Using the interfaces provided by MDL and HAL layers, the VHS could be developed. However, there are the following disadvantages:

- As VHS is developed using the interfaces from MDL and HAL, new COM objects are created. These objects are only used for simulation purposes, and they replace existing objects.

- As VHS is developed using the interface from the MDL layer, only software implementation in the upper layer (BHV) can be involved in the hardware simulation. In other words, implementations in the MDL, HAL, and firmware layers cannot be involved in the hardware simulation.

- As VHS is developed using the interface from the HAL layer, only upper layers can be tested. In other words, HAL and firmware cannot be involved in the hardware simulation. Moreover, HAL was in the process of development. Therefore, some interfaces were not usable at that time.

- As VHS is developed using the interfaces from the MDL and HAL layers, the simulation cannot be conducted in real-time because of the layer abstraction.

To have maximum advantages, we developed the VHS that interrelates with the motion controller and uses signal data from mathematical models and hardware. The design gives the following advantages:

- As VHS is developed at the motion controller level, the VHS manipulates the 3D models and simulates virtual sensors in real-time. It means that the VHS can provide real-time motion visualization, and the interaction between the hardware components is visible. Moreover, virtual sensors can transfer its signal data to the motion controller in real time.

- As VHS is developed at the motion controller level, both the firmware and TEM software can be tested using the VHS. In other words, there is no need to create different simulations for the firmware and the TEM software because VHS is developed at the lowest level.

- As VHS is developed on the motion controller level, VHS can visualize the real hardware by applying actuator signals onto 3D models.

7.4 Conclusion

This chapter firstly explained how virtual hardware is created using the chosen technology and 3D models. Secondly, the chapter presented design choices and their pros and cons. By evaluating these design choices, we decided to use the motion controller and mathematical models. This design gives the VHS to obtain real-time motion visualization and sensor simulation. Most importantly, the VHS can be used to test not only the firmware but also TEM software. Moreover, the VHS can visualize the physical hardware in its 3D environment using 3D models.

8.System Architecture

This chapter focuses on the system architecture of the virtual hardware simulator (VHS). The architecture consists of three main components, namely the Unity game engine, StagePlugin, and StageAdapter.

8.1 Architecture design

Figure 26 shows the overall architecture of the VHS. In order to fulfill the non-functional requirement of extensibility, components: StagePlugin and StageAdapter were introduced to the system architecture. And each component and its responsibility are explained in the sections below.

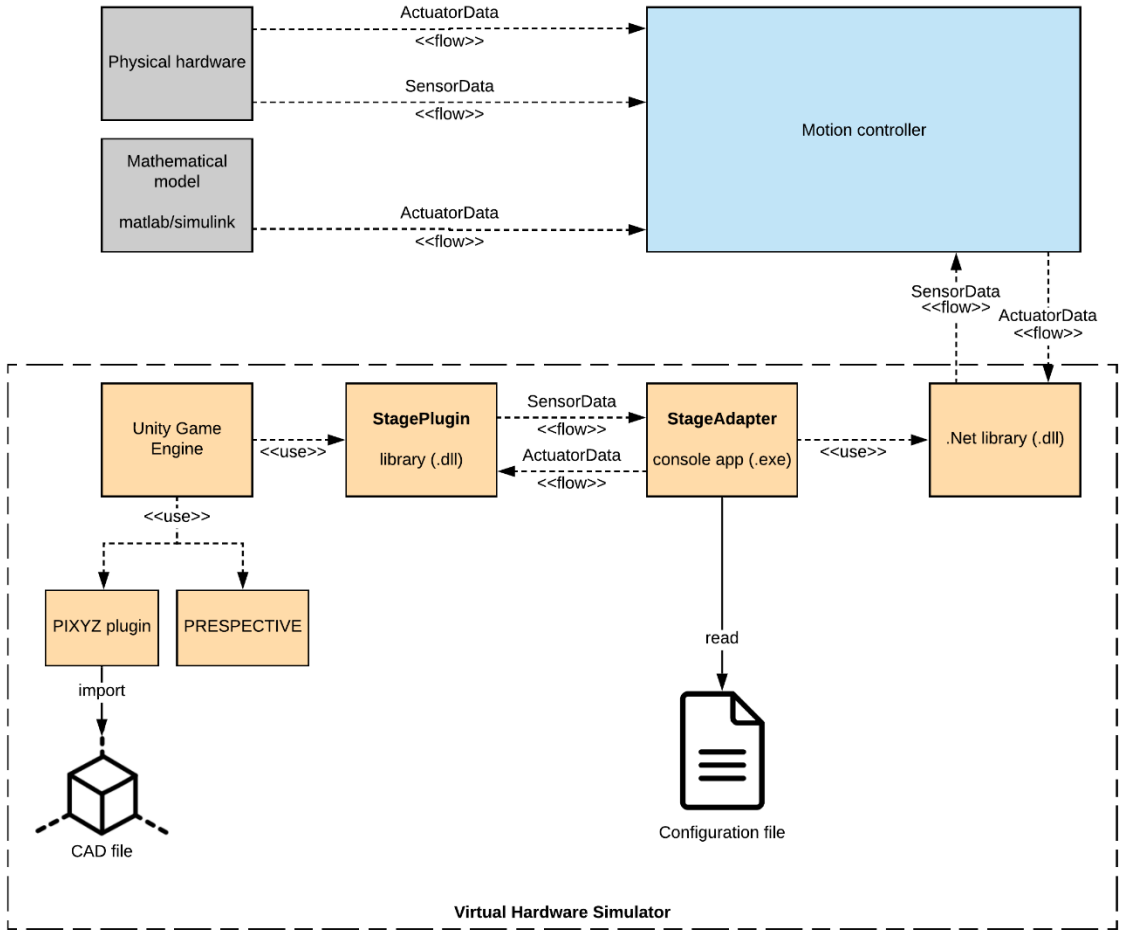


Figure 26: Virtual Hardware Simulator (VHS)

8.2 Unity game engine

As shown in Figure 26, The unity game engine uses two software components: the PIXYZ plugin and PRESPECTIVE. Using the PIXYZ plugin, 3D geometrical models are imported into the environment of the Unity game engine. In this environment, PRESPECTIVE defines motion behavior for the 3D models using kinematics. Virtual sensors are placed on the 3D models, and the sensor behaviors are created using C# scripting. These three software products are utilized to fulfill the system requirements by visualizing motion behavior as well as simulating the virtual sensors.

8.2.1. Visualizing motion behavior

The motion behaviors need to be defined using the 3D models that are the primary entities for the visualization. To visualize motion behavior in real time, Unity asks for actuator signal data periodically from the physical hardware or mathematical model using StagePlugin.

The frame rate of the Unity game engine defines the frequency of asking actuator data. In each frame, Unity updates the position of 3D models with the new data from the motion controller.

8.2.2. Sensor simulation

Virtual sensors are created using the 3D models of the *stage* components. When the 3D models are moving in real time, the sensors are triggered. The triggered sensor data is transmitted to the motion controller using StagePlugin.

There are two types of sensors that are created in the Unity environment:

- **Touch sensor:** A sensor is triggered when the sensor contacts another 3D model.

- **Proximity sensor:** A sensor is triggered when the distance between the sensor and another part exceeds the threshold distance.

In each frame, Unity checks whether sensors are triggered or not. It means the timing accuracy of the check depends on the framerate of the simulator.

8.3 StagePlugin

StagePlugin is a plugin that is used by the Unity Game Engine. It establishes a connection to the StageAdapter. The plugin is a client for the StageAdapter and is used to retrieve the actuator data as well as send the simulated sensor data to the motion controller. Currently, the Windows Communication Foundation (WCF) is the chief technology for exchanging data between the StagePlugin and the StageAdapter.

The primary responsibility of the StagePlguin is to establish a connection to StageAdapter for data exchange. The plugin is in the form of a Dynamic-link library (DLL) file.

8.4 StageAdapter

StageApdater is an executable running on windows using .Net platform, and it is responsible for the following:

- The adapter gathers the actuator signal data of actuators from the motion controller and sends the data to the StagePlugin when requests come.
- The adapter firstly registers the actuators that are specified in the configuration file. The adapter updates the signal data of the registered actuators at a frequency that is defined in the configuration file. These actuator signal data are taken from the actuator mathematical models.
- The adapter sends the simulated sensor data to the motion controller when it receives the sensor data request from the StagePlugin.

8.4.1. Configuration File

The configuration file is used to configure the VHS for the following reasons:

- The address of the motion controller should be specified in this file.
- Hardware components that need to be involved in the simulation should be specified in this file.
- The StageAdapter should update registered actuator data at a defined frequency. The frequency should be defined in this file.

The configuration file is in the form of an Extensible Markup Language (XML) file.

8.4.2. Communication Technology

Windows Communication Foundation (WCF) is used to exchange data between StagePlugin and StageAdapter. StagePlugin is a WCF client, while StageAdapter is a WCF server.

8.5 Conclusion

This section presents the system architecture. In order to have a modular architecture for the future extensibility, StagePlugin and StageAdapter were introduced. StagePlugin is responsible for transmitting the signal data between the Unity Game Engine and StageAdapter, while StageAdapter is responsible for communicating with the motion controller to read the actuator signal data and write simulated sensor data. The next chapter explains the implementation of the architecture.

9.Implementation

This chapter describes the development environment and implementation of the virtual hardware simulator (VHS) for the following system design explained in the previous chapter.

9.1 Development environment

PRESPECTIVE software is the digital twining software platform that was used to define hardware behavior on 3D models. The software is run on the top of the Unity Game Engine.

Version: PRESPECTIVE deployment version 1.2.86.1706

PIXYZ is a plugin for Unity. The plugin is used to import 3D models into PRESPECTIVE environment. On the imported 3D models, PRESPECTIVE software defines hardware behavior.

Version: 2019.2.0.59

Unity is a real-time development 3D platform for a software application. PRESPECTIVE software and PIXYZ plugin are installed on Unity. Unity supports C# scripting language.

Version: 2019.2.19f1

Visual Studio 2019 and .Net framework 4.7.2 were used for the implementation of the StagePlugin and StageAdapter that is shown in Figure 14.

Git was a version control system for this project.

9.2 Simulation on Unity game engine

Two unity scenes are created in the Unity game engine. The first scene, called StartScene, is loaded when the simulator starts to establish a connection to the motion controller. Figure 27 shows the StartScene.

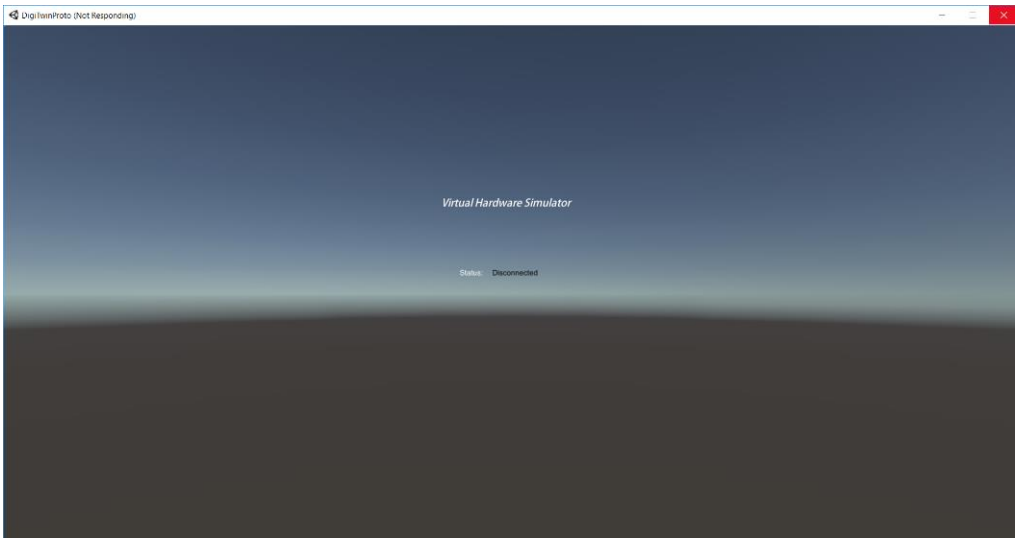


Figure 27: StartScene

When the connection is established successfully, the second scene, called SimulationScene, is loaded to start the simulation and show the virtual hardware (dynamic 3D models). Figure 28 shows the SimulationScene.

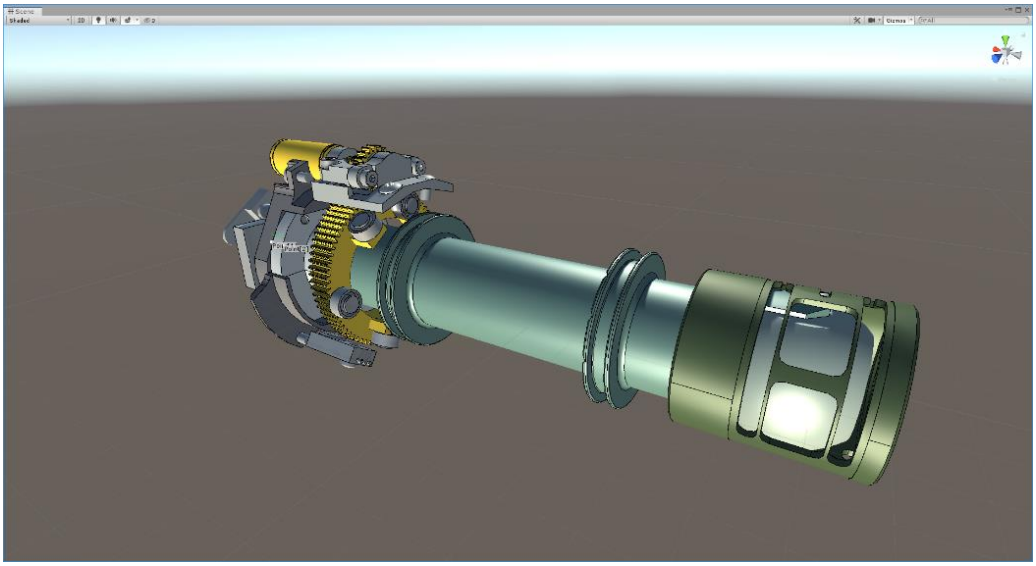


Figure 28: Simulation scene for the ball valve component

There are two types of sensors on the *stage*. Each sensor has a distinct behavior. The behaviors of the sensors are defined using the scripting language.

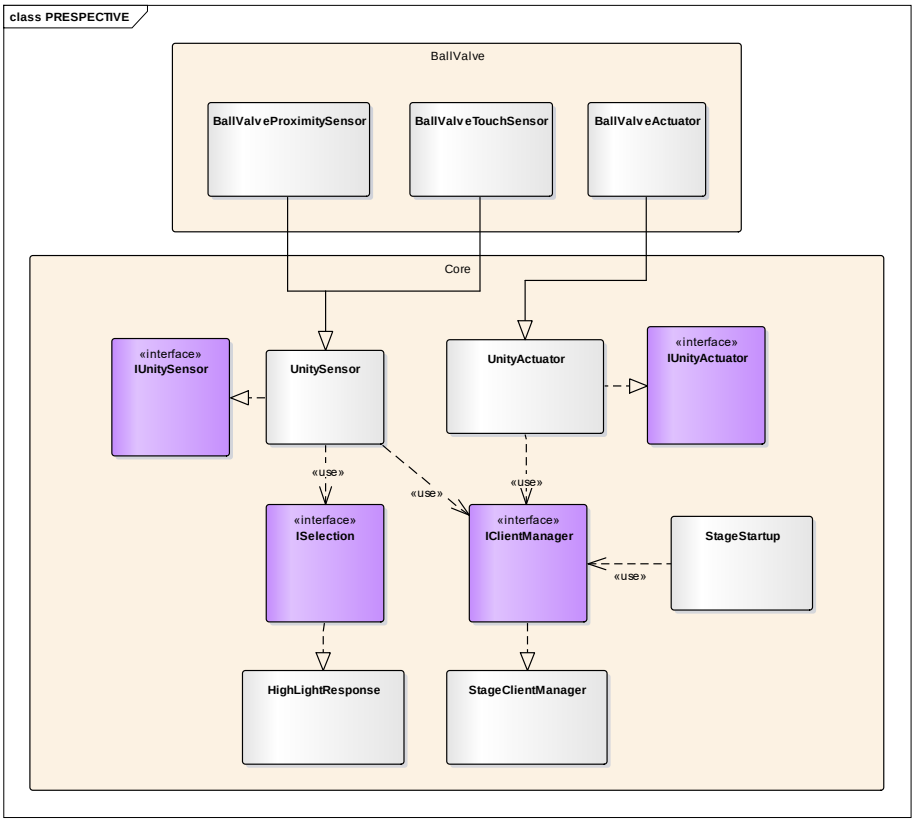


Figure 29: Class diagram for visualizing motion behavior and simulating sensors

In Figure 29, the class diagram is shown. It has two namespaces, and each of them has its own responsibility.

Core: Core namespace is responsible for the following:

- Providing base class, UnitySensor, for sensors
- Providing base class, UnityActuator, for actuators

- Highlighting the triggered sensor using HighLightResponse class
- Establishing a connection to the motion controller using StageClientManager class
- Initializing the system for simulation using StageStartup class

BallValve: Ball valve is a microscope component, and it has its sensors and actuators. Each actuator and sensor implements the base classes that are in the Core namespaces.

9.3 StagePlugin

The plugin is a C# (Mono) assembly in the form of a DLL file. It provides the WCF client and its data contracts. It is utilized by the Unity to exchange actuator data and simulated sensor data. In Figure 30, ActuatorData and SensorData are the contacts with the WCF server. Once the client of the WCF is created, data is exchanged using IStageService.

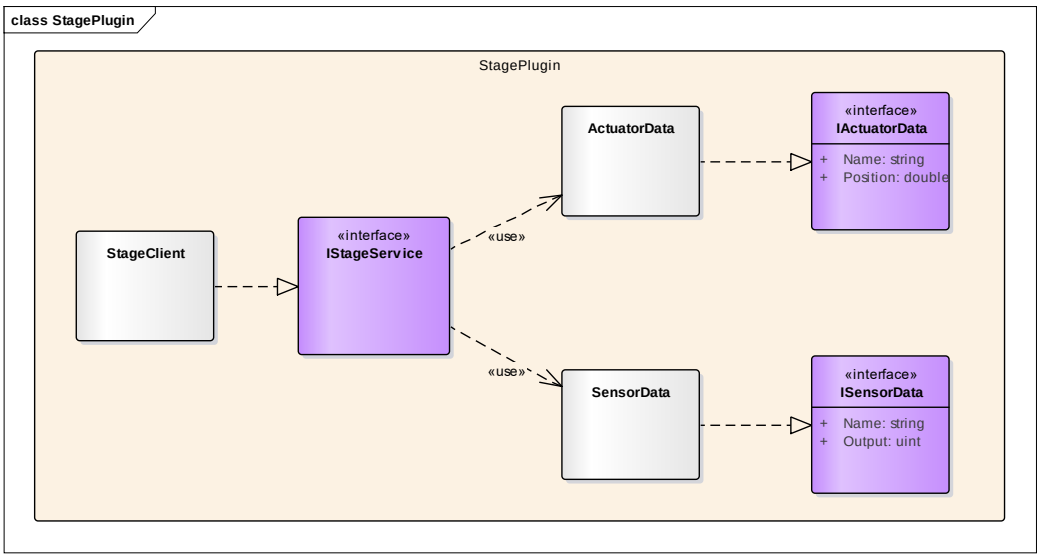


Figure 30: Class diagram of StagePlugin

9.4 StageAdapter

The StageAdapter is a C# (.Net) executable, and it has three namespaces, namely **CompProdirive**, **Config**, and **Core**, as shown in Figure 31.

Config: This namespace contains classes that represent the structure of the configuration file. StageConfig class represents a root element of the configuration file, which is in the form of an XML file.

CompProdrive: This namespace contains classes that access the motion controller to send sensor signals and read actuator signals. ProdriveSensor and ProdriveActuator represent a physical or mathematically modeled sensor and actuator, respectively. To access the motion controller, the ProdriveManager class is used.

Core: The core namespace holds the main logic of the adapter. Firstly, StageSerializer class de-serializes the configuration file. All the actuators and sensors are registered in the ComponentRegistration class. This class is updates signal data of the registered actuators regularly using the StageTimer class. StageServiceManager starts the WCF service using StageService class. Lastly, StageManager is a bridge to convey data between ComponenetRegistration and the WCF client, which is StagePlugin.

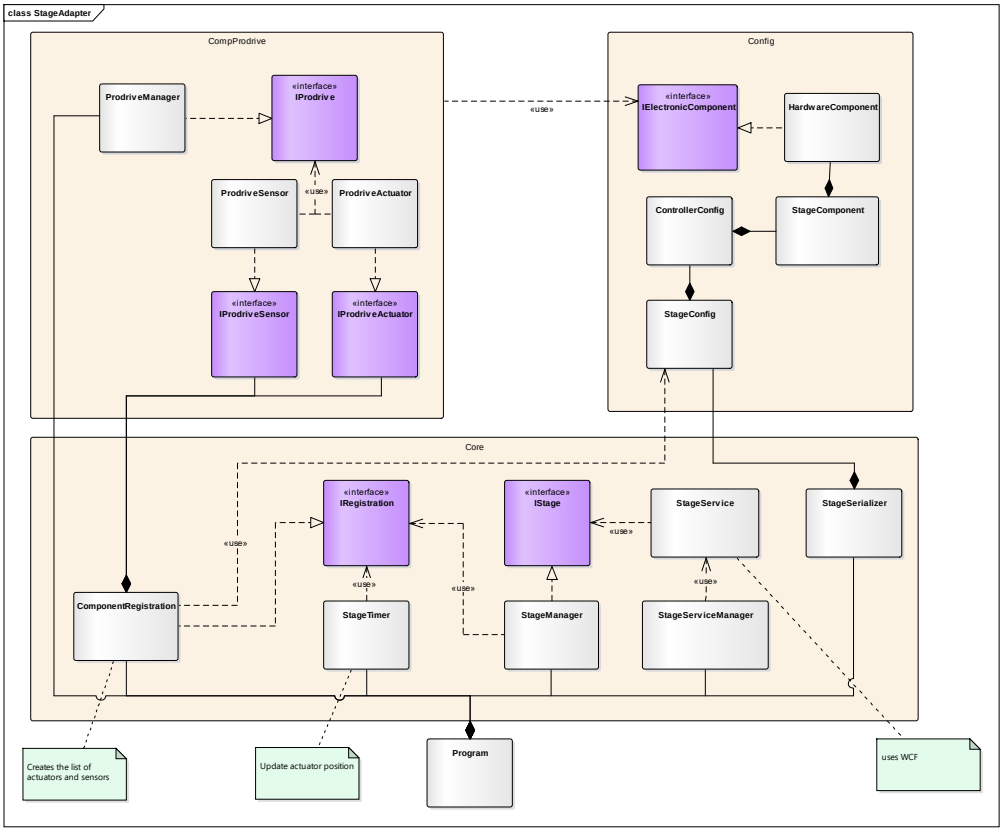


Figure 31: Class diagram of StageAdapter

9.4.1. Configuration file

The file contains the following configurations:

- The controller element in the file contains three configurations. The first one is the motion controller name. The second is the Internet Protocol (IP) address of the motion controller. The third is the frequency that StageAdapter uses.

- The component list contains the microscope components that are involved in the simulation. Each component has an actuator list and a sensor list. Each actuator and sensor has its signal and name and address in the motion controller. The signal address is used by the StageAdapter to exchange signal data with the motion controller, while the Unity game engine uses the name to exchange data with StageAdapter.

```
Configuration.xml
1  <?xml version="1.0" encoding="utf-8"?>
2
3  <SmartStage>
4
5    <Controller ControllerName="SSC-Master" IPaddress="127.0.0.1"  Frequency="50">
6      <Components>
7        <!--Ball Valve component-->
8          <Component Name='BallValve'>
9            <Actuators>
10             <Actuator Name='BallValveActuator' Signal='Baseboard/AxisBV/ControlNetwork/ActualPosition'> </Actuator>
11           </Actuators>
12           <Sensors>
13             <Sensor Name='ProximitySensorForClosing' Signal='SSIOB/SIM_EPD_BV/Enable'> </Sensor>
14           </Sensors>
15         </Component>
16     </Components>
17
18   </Controller>
19 </SmartStage>
20
21
```

Figure 32: The short version of the configuration file

9.5 System behavior

The system behavior diagram is shown below. In the diagram, five components are presented, namely motion controller, mathematical model, StageAdapter, StagePlugin, and Unity. Three of them: StageAdapter,

StagePlugin, and Unity, form the VHS. The other two are external systems that interact with the VHS. The main actor in this diagram is the mechatronics engineer, as shown in Figure 33.

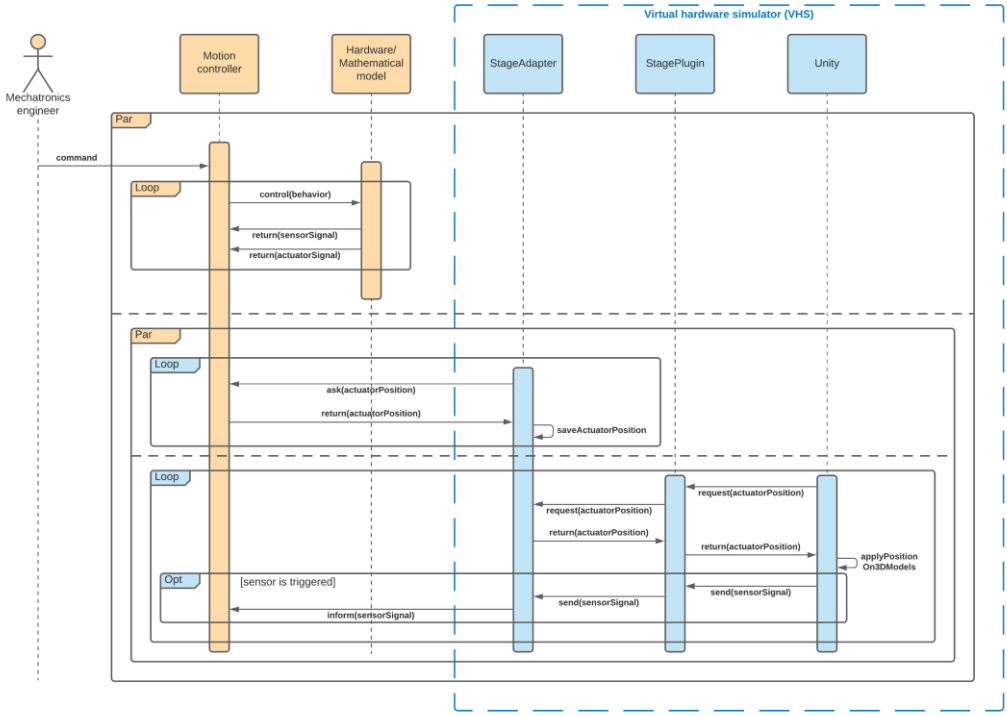


Figure 33. Sequence diagram for the VHS and external systems

The actor starts the initial action to move the *stage.* The motion controller receives a command from the actor and starts controlling the hardware behavior in real time. While the hardware is moving its components, actuator and sensor signal data are also received in real time by the motion controller.

During this real-time control, the VHS role comes into play. The StageAdapter polls the motion controller at the configured frequency to get actuator positioning data. The StageAdapter holds these data and sends them to the StagePlugin when requested. This data request is initiated by Unity, which manipulates StagePlugin to get the actuator data. Unity applies the data onto 3D models and simulates them. When any virtual sensor is triggered on the 3D models, Unity utilizes the StagePlugin to send the sensor signal data to StageAdapter. Then the StageAdapter conveys the data to the motion controller for further processing.

9.6 Conclusion

In order to meet the non-functional requirement (extensibility), the system responsibilities are delegated into different components, namely PRESPECTIVE, StagePlugin, and StageAdapter. Moreover, each component is implemented using the SOLID principle.

10. Verification

This chapter explains the process of verification of the virtual hardware simulator (VHS). The process is used to check whether the developed software meets the requirements of the project and satisfies the customer need. The testing process is conducted manually.

10.1 Preparing a test environment

To verify and validate the implementation, the testing environment was set up at the beginning of the project. This testing environment consists of three pieces of software that are described below:

- The motion controller needs to be set, and it has two modes. The first is simulation mode that can run a mathematical model instead of the real hardware, whereas another mode directly interacts with the real hardware. Since the real hardware is not usually available, the motion controller simulation is set.

- The test software needs to be set. This software represents the TEM software that invokes firmware algorithms by sending commands to the motion controller.

- The VHS is the software that we developed. Before the test starts, the simulator must be configured using the configuration file. In the file, microscope components that are intended to be simulated and the motion controller network address must be defined.

10.2 Verification

The virtual hardware must conform to the high-level requirements that are defined in Section 6.2 The processes of verification are explained using the use cases in the below sections.

10.2.1. Use case: Visualize the hardware motion

The firmware controls the behavior of the hardware system. By visualizing them on the 3D models, the interaction between the hardware components can be inspected. The firmware is a real-time software. Therefore, the visualization also must be in real time.

Test scenario:

1. The user starts an open ball-valve command from the test software.
2. The test software sends the command to initiate a firmware algorithm in the motion controller.
3. The motion controller instructs the behavior of the mathematical model that represents the ball-valve actuator.
4. The system polls the actuator data of the mathematical model.
5. The system applies the actuator data on the 3D models in real time.

The VHS successfully visualized the hardware motion in real time. Figure 34 shows the sequence diagram for simulating the ball-valve components. In the diagram, the steps for the test scenario are illustrated.

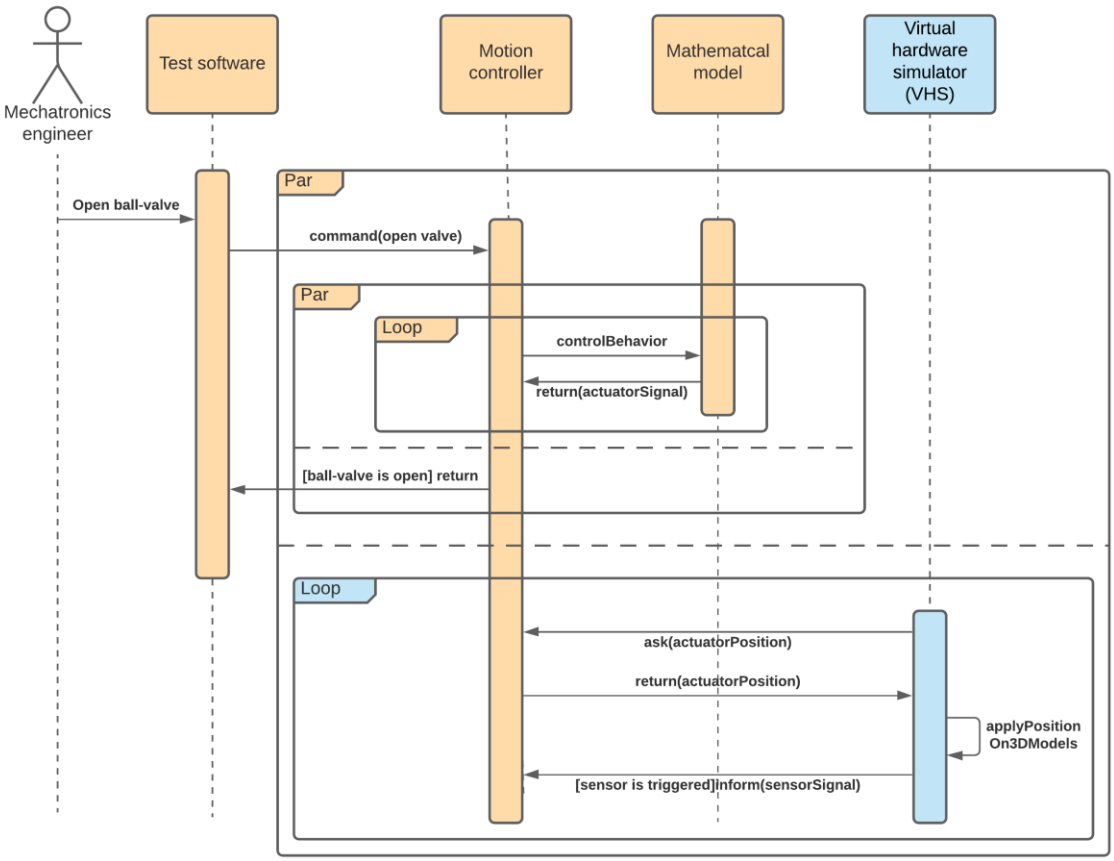


Figure 34. Sequence diagram for simulating the ball-valve component

10.2.2. Use case: Testing desired behavior

The ball-valve has two proximity sensors. These sensors are designed to detect whether the ball-valve is open or not. As part of the verification process, a proximity sensor is placed to detect whether the ball-valve component is open or not.

Test scenario:

1. The user initiates an open ball-valve command from the test software.
2. The test software sends the command to initiate a firmware algorithm in the motion controller.
3. The firmware controls the behavior of the mathematical model that represents the ball-valve actuator.
4. The system polls the actuator data of the mathematical model.
5. The system applies the actuator data on the 3D models in real time.
6. The system sends the triggered sensor data to the motion controller.

The test scenario can be inspected in Figure 34.

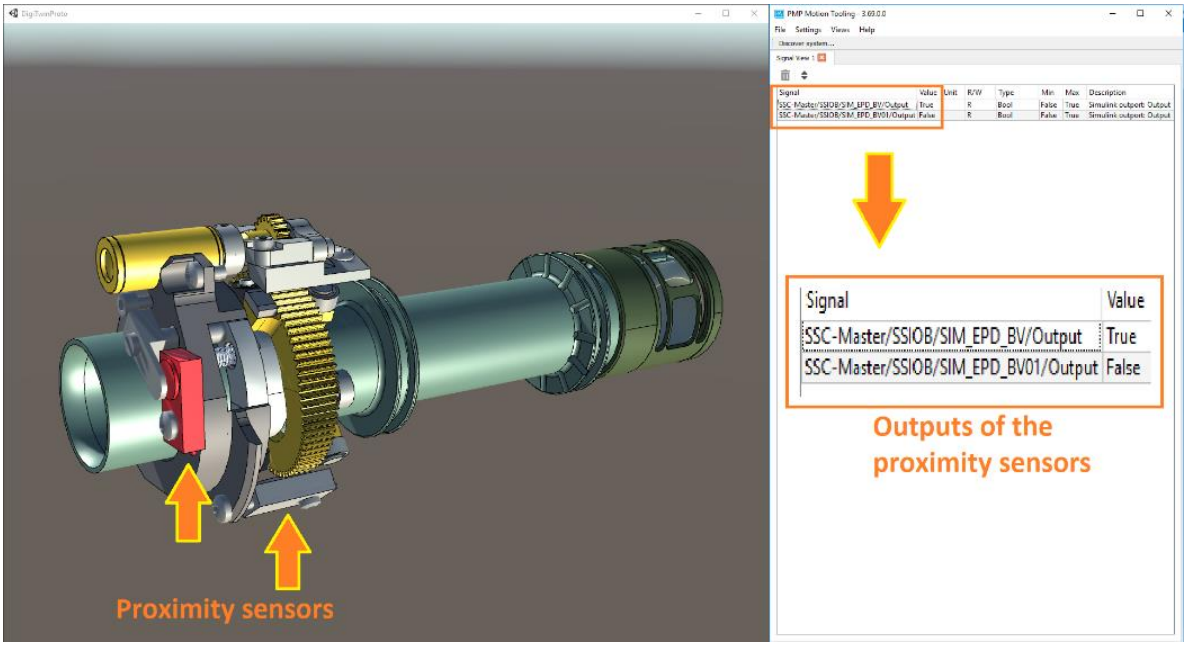


Figure 35: Real-time sensor simulation for the ball-valve component

Figure 35 shows the VHS on the left side and the motion controller tool on the right side. One of the proximity sensors is triggered, and the system highlights the sensor by changing its color to red. From the motion platform tool, the sensor status is shown in real time.

10.2.3. Use case: Testing desired behavior (fault scenario)

A fault scenario was carried out on the VHS. In this test, the firmware has a bug, which does not take sensor feedback. In this scenario, a collision takes place. When a collision happens on the virtual hardware, it highlights the collided part by changing its color to red and sends the collision information to the motion controller. Figure 36 shows a screenshot that shows the result of the collision.

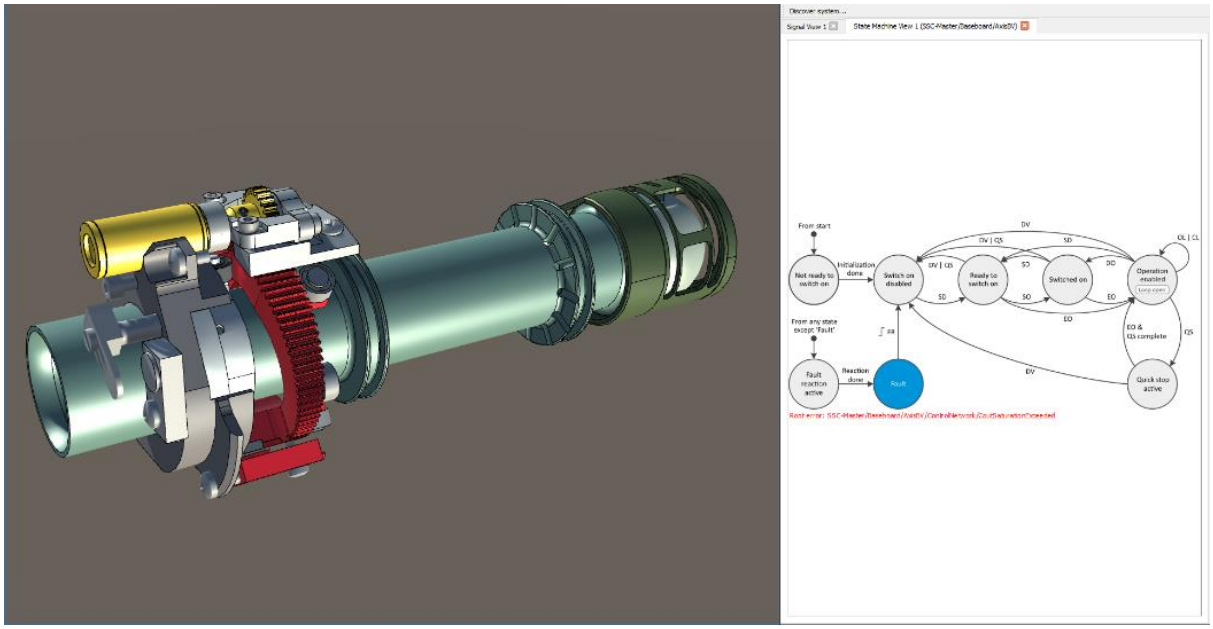


Figure 36: The result after collision

On the right side of Figure 36, the state machine diagram of the ball-valve component is shown. After receiving a signal from the VHS, the hardware state turns to a fault state.

10.2.4. Visualizing the physical hardware behavior

The VHS is able to visualize the behavior of the physical hardware. This feature is also verified by conducting a test in the motion lab.

Test scenario:

1. The user initiates an open ball-valve command from the TEM software.
2. The TEM software sends the command to initiate a firmware algorithm in the motion controller.
3. The firmware controls the behavior of the hardware.
4. The system polls the actuator data of the mathematical model.
5. The system applies the actuator data on the 3D models in real time.

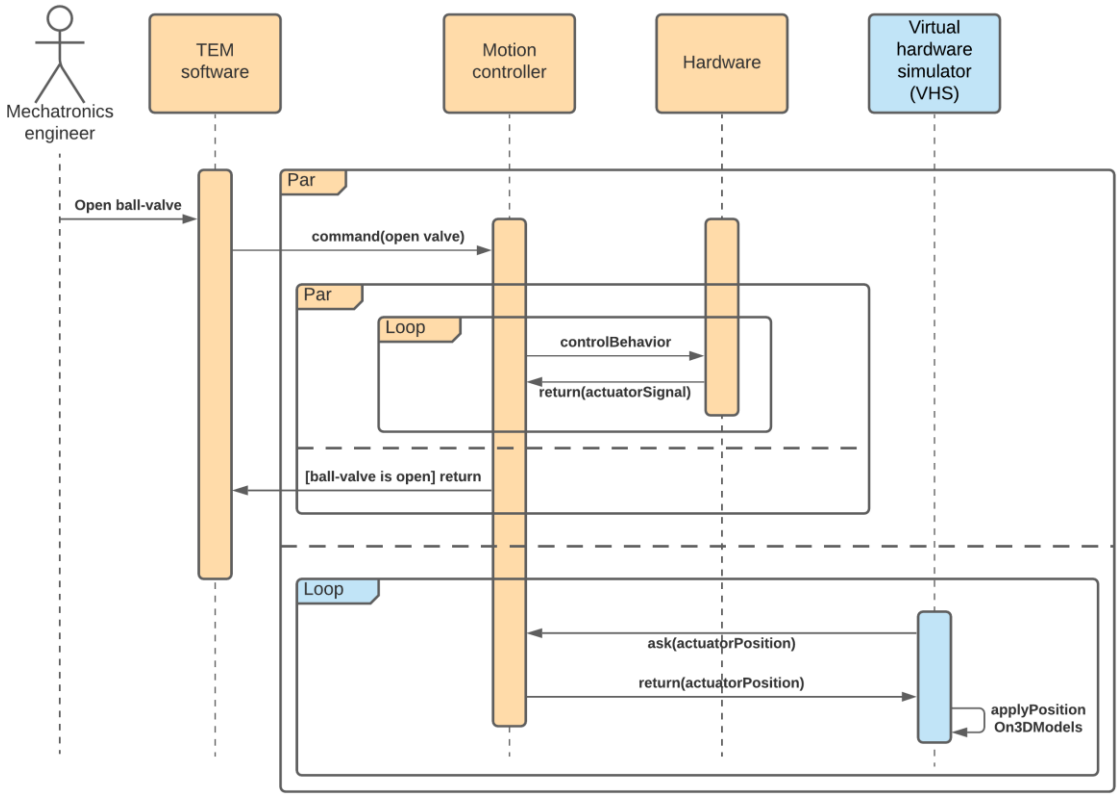


Figure 37. Sequence diagram for visualizing the physical hardware on the VHS

10.3 Conclusion

This chapter explains the verification process to check whether the implementation satisfies the requirements. In order to carry out the verification process, the testing environment was set. The next chapter is a conclusion of the project.

11. Conclusions & Recommendations

This chapter focuses on the conclusions of the project and elaborates on the achieved results. It also presents future work.

11.1 Summary

The project was conducted to investigate and explore how the concept of digital twinning can benefit the product development lifecycle in Thermo Fisher Scientific. Investigating the possibilities and implementing a proof of concept is the focus of this project.

Currently, the design verification process is highly dependent on the hardware that takes a long time to be manufactured, assembled, and prepared. Due to the dependency, the product development lifecycle takes a long time. By investigating possibilities to speed up the development time, we have developed a software tool that is called virtual hardware simulator (VHS). It is created during the design phase, allowing engineers to test the firmware and TEM software designs before hardware is manufactured.

We chose PRESPECTIVE as the main technology to create a VHS. PRESPECTIVE uses 3D CAD models for motion visualization and simulation. The motion behavior of the hardware is defined using PRESPECTIVE feature called kinematics relation. In these models, virtual sensors are placed. And the behaviors of virtual sensors are defined using C# scripting.

Compared to other technologies, PRESPECTIVE is a software development platform, while others are application software. Therefore, PRESPECTIVE that is developed on top of the Unity game engine is more customizable in creating a custom GUI and open for client applications. It also capability to build artifacts that can be locally used by any engineer.

The VHS simulator communicates with the motion controller to simulate hardware components. By communicating with the motion controller, the VHS fulfills the requirements of real-time simulation. With this design, the VHS allows mechatronics engineers to test firmware and software test engineers to test the TEM software. Moreover, the VHS allows TEM software developers to start developing software before actual hardware is available.

The creation of the VHS is dependent on the two disciplines, as shown in Figure 38. Firstly, the VHS uses 3D models from mechanical engineers. Secondly, the mechatronics engineer supplies the motion controller library that allows the VHS to connect to the mathematical models. Once the VHS is created, firmware and software tests can be carried out.

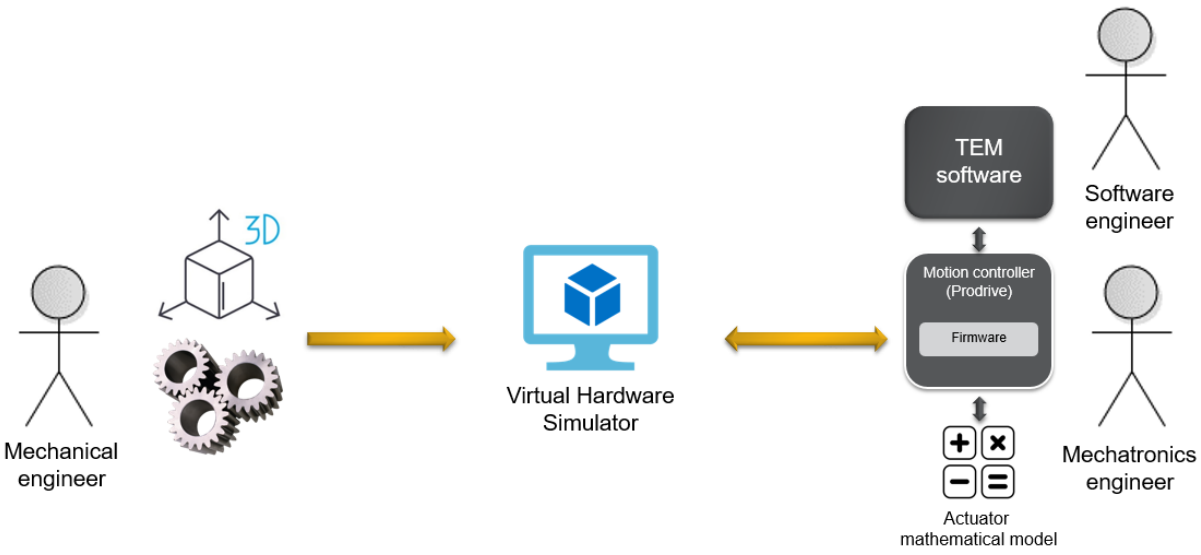


Figure 38. Development dependency for the creation of the virtual hardware simulator

During the firmware and software test, design flaws that cause collisions can be detected. As a result, necessary implementation changes can be carried out earlier before the hardware manufacturing process takes place. Additionally, the VHS enables software engineers to develop their software without relying on physical hardware.

With the defined system design, the VHS can be used for remote problem diagnosis. The simulator receives the real-time sensor data from physical hardware and visualizes the data onto 3D models, enabling service engineers to get a better insight into the hardware motion.

11.2 Recommendations

11.2.1. Short-term

Currently, the VHS simulates the non-deformable materials of the stage. The *stage* also has flexible materials such as springs and belt. In order to simulate these materials, a Unity asset called MegaFierce can be used. I recommend investigating the use of the asset for future use.

PRESPECTIVE is used to define kinematics relations on the 3D models. The defined kinematics is lost if the 3D model is updated. We recommend investigating a way to keep constraints and kinematics when the model is updated. The company, called Unit040, develops PRESPECTIVE, and they are working on this feature. Contacting them is the first step for the future development of VHS.

The unity project is built manually to generate an executable artifact because the building process of the Unity game engine requires its editor. To automate the build process, two approaches were investigated. The first one is to use the Unity Cloud Build. The second one is to set up the docker container in which the Unity editor is installed. Therefore, choosing one of the approaches and integrating the build process with the company build server are the next short-term goals.

During the project, use cases from a software test engineer have not been implemented due to the project time limitation. Implementing the use case and delivering the VHS to the test engineer is another next step for further development.

11.2.2. Long-term

The current implementation can now draw real-time actuator data from the physical hardware onto virtual hardware in the simulator. By extending the current VHS, the whole microscope can be visualized in the 3D environment. This feature can be used for remote problem diagnosis when physical hardware misbehaves in a remote location.

Moreover, actuator and sensor data can be collected in the VHS. On the collected data, advanced machine learning algorithms can run to predict possible failures. Furthermore, the VHS can be used for service engineering training and support. By visualizing the hardware components in the VHS, the service engineers can learn about hardware structures and their interactions in detail.

The development process of the VHS needs to be integrated seamlessly into the product development process of the company. To achieve it, I recommend having a separate role in developing and maintaining the VHS within the company because this role would be closely interacting with multiple disciplines: software, mechatronics, and mechanics. The VHS can then be delivered to the mechatronics and software engineers to start developing and testing the firmware and TEM software.

12. Project Management

This chapter presents the project management process that was conducted during the project.

12.1 Introduction

We followed an agile methodology. We held one meeting per week to discuss the outcome of the investigation. Based on the discussion, we planned the activities for the following weeks. We aimed to have a prototype that shows the outcome of the investigation and implementation. The prototype gave a good insight into where we were heading and helped gather more specific feedback.

12.2 Project Planning and Scheduling

The project was executed in ten months, starting in January 2020 until October 2020. For these ten months, high-level planning has been depicted in Figure 39, which shows the roadmap of the project plan. The roadmap shows different activities that took place during the project lifetime.

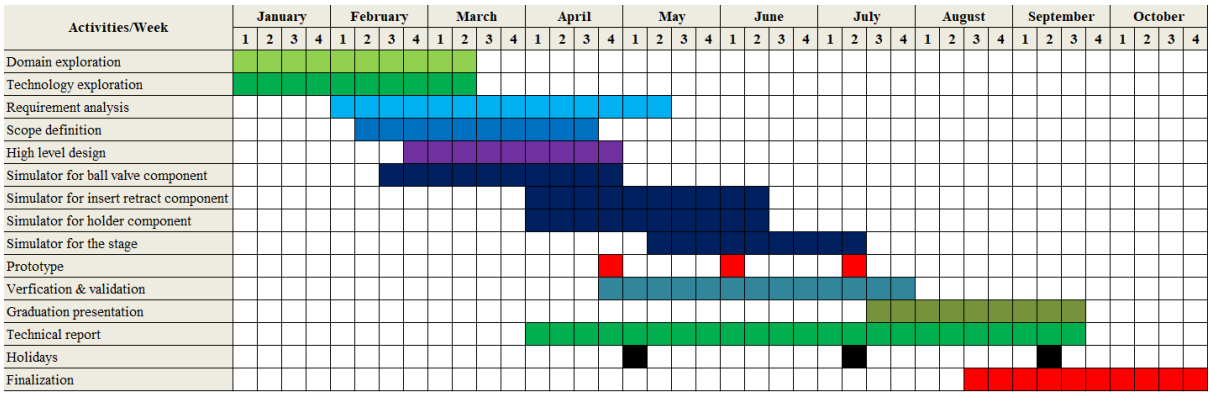


Figure 39: Project plan

12.3 Project execution

The project description used the term “digital twin”. Therefore, the first activity at the beginning of the project was to understand the digital twin concept. After the exploration of the concept, the second step was to identify problems that we can solve with the concept. To identify problems, domain analysis had been executed. At the same time, commercial software products related to digital twinning had been investigated.

With the technology investigation, small prototypes were made. These prototypes showed the capability of commercial software products as well as the compatibility with the software stack of Thermo Fisher Scientific. The primary purpose of these prototypes was to make design decisions as well as to gather stakeholders’ user requirements.

Requirement gathering and implementation had been conducted simultaneously during the project. We refined the prototype and added more functionalities over time. This prototype was also used in a regular meeting to get feedback during the project progress. Until the end of the project, the prototype was evolved.

12.4 Risk analysis

The section explains the risks that are identified during the project. The identified risks are related to project scope, procurement, and communication. Table 17 shows the risks and their description, consequences, and mitigation.

Table 17: Risk analysis

Step 1: Risk Identification		Step 2: Risk Assessment		Step 3: Risk Response	
		Rank	Impact		
Risk Name	Detailed Description	Probability (1-5)	Severity (1-5) and Impact	Response Catergory	Response
Unclear project scope	Project scope could not be defined if business values are not identified. Identifying the values might take an unexpectedly long time.	2	5. Having unclear project scope causes project failure because project planning and tasks cannot be explicitly defined.	Avoid	Regularly update stakeholders on the project progress Arrange various meetings with people to identify problems that can be mitigated with this project
Difficulty in stakeholder management	If the project scope is not clear, meetings need to take place to define the scope. During these meetings, many stakeholders might be involved.	2	3. Managing various stakeholders is challenging. The project time might not be enough to fulfill all of the requirements. As a result, some stakeholders might be unsatisfactory with the outcome of the project.	Avoid	Prioritize the stakeholders and negotiate with them.
Slow procurement process	Two commercial pieces of software are used in this project. Therefore, their licenses need to be ordered. The company procurement process is complex and might take a long time to obtain them.	3	4. The project progress slows down, causing delays in the planning. As a result, stakeholders might be unsatisfied.	Avoid	Order the licenses early Contact with the companies to obtain trial licenses. Track license ordering process
Backward incompatibility	The current implementation uses third-party motion controller software. The software is updated regularly and it might cause backward incompatibility.	N/A	2. The effort is needed to maintain the project implementation with the updated controller.	Accept	Create a modular architecture
Third-party dependency	The current implementation uses third-party motion controller software. The software might be changed in the future.	N/A	3. The effort is needed to maintain the project implementation with the new motion controller.	Mitigate	Create a modular architecture for future change
Inefficient communication	Due to the COVID-19 outbreak, meetings take place online. The speed of approaching people became slow. It might cause inefficient communication and slack engagement with the stakeholders.	4	4. The project progress slows down, causing delays in the planning. As a result, stakeholders might be unsatisfied.	Mitigate	Use presentation and other visual aids during meetings Create a virtual board for stakeholder to inform the project progress and results
Health problem	There is a possibility that I become ill.	1	4. The project progress slows down, causing delays in the planning. As a result, stakeholders might be unsatisfied.	Avoid	Follow the advices to stay hygienic

12.5 Challenges

The project was exploratory at the beginning. Learning the domain knowledge and finding an appropriate use of this project in product development was quite challenging. To find beneficial uses of this project, we needed to prepare prototypes, present the capabilities of the prototypes, and discuss the progress with various people.

The project period was ten-month. Within this dedicated timeframe, stakeholder concerns needed to be prioritized and divided into small tasks. To identify concrete tasks, the domain knowledge needed to be perceived at a detailed level. With the domain knowledge, the project implementation needed to be finished and delivered within the dedicated timeframe.

Planning ahead was challenging at the beginning of the project because the usefulness of the virtual hardware simulator for the company was not apparent. In this generic situation, planning for the future was not possible. By discussing the progress with the main stakeholders frequently, the project goal became apparent, and the short-term and long-term plans were identified.

13. Project Retrospective

This chapter finalizes the technical report by reflecting on the project based on the author's point of view.

13.1 Reflection

The project that was conducted at Thermo Fisher Scientific brought me many challenges and yet an exciting experience. These challenges gave me many opportunities to improve myself on both personal and professional skills.

The goal of the project was to find beneficial uses of the digital twinning in the product development process of the company and implement a proof of concept for these uses. At the beginning of the project, I had to gather domain knowledge of the TEM structure and its development processes.

After understanding the capabilities of digital twinning, I gathered domain knowledge and identified the main problems. The problem identification took a very long time and was challenging. To identify them, meeting with various people was helpful. Step by step, I demonstrated presentations and prototypes to these people in the company, and their feedback was valuable. These meetings were also beneficial to make correct design decisions for the VHS.

The implementation is a proof of concept. Therefore, a modular design for the VHS was essential for future development. During the implementation, I was continually aiming to use the SOLID principle to make the simulator extensible, easy-to-understand, and maintainable.

To summarize, I have had a challenging but eye-opening experience with this project. It was an excellent opportunity not only to improve my technical skills related to software development but also to enhance my organizational skills. Managing stakeholders and cooperating with people from a multidisciplinary team have improved both my professional and personal skills. The knowledge and experience that I gained during this project have opened a promising future for me.

Glossary

TFS	Thermo Fisher Scientific
VHS	Virtual Hardware Simulator
CCD	Charged Coupled Device
BHV	Behavior Layer
MDL	Module Layer
HAL	Hardware Abstraction Layer
MBE	Model-Based Engineering
DC	Direct Current
API	Application Programming Interface
CI	Continuous Integration
COM	Component Object Model
TEM	Transmission Electron Microscope
PDEng	Professional Doctorate in Engineering
MBPD	Model-Based Product Development
OMG	Object Management Group
GUI	Graphical User Interface
UML	Unified Modelling Language
SYSMOD	System Modelling
3D	Three Dimensional
CAD	Computer-Aided Design
MC	Motion Controller
FPS	Frame Per Second
IP	Internet Protocol
DLL	Dynamic-Link Library
SDK	Software Development Kit
XML	Extensible Markup Language

Bibliography

1. Len Bass, (3rd edition). Software Architecture in Practice.
2. Tim Weilkiens, System Engineering with SysML/UML, 2008.
3. A Deloitte series on Industry 4.0, digital manufacturing enterprises, and digital supply networks https://www2.deloitte.com/content/dam/Deloitte/cn/Documents/cip/deloitte-cn-cip-industry-4-0-digital-twin-technology-en-171215.pdf
4. Tretmans, J. (Editor). / Tangram : model-based integration and testing of complex high-tech systems. Eindhoven : Technische Universiteit Eindhoven, Embedded Systems Institute, 2007. 231 p. https://research.tue.nl/en/publications/tangram-model-based-integration-and-testing-of-complex-high-tech-
5. Fei Tao, Meng Zhang, A.Y.C. Nee. Digital Twin Driven Smart Manufacturing. 2019 https://books.google.nl/books?hl=en&lr=&id=PvKGDwAAQBAJ&oi=fnd&pg=PP1&dq=digital+twin+driven+smart+manufacturing&ots=hk0mJOMjkj&sig=LsPnGTOLXRvU02aXEHv-jBkv7pQ#v=onepage&q&f=false
6. Michael Grieves, John Vickers. Origins of the Digital Twin Concept. 2016
7. Franz-Josef Kahlen, Shannon Flumerfelt, Anabela Alves. Transdisciplinary Perspectives on Complex Systems. 2017
8. Zhuming Bi. Finite Element Analysis Applications. 2018
9. Azad M. Madni, Carla C.Madni, Scott D.Lucero. Leveraging Digital Twin Technology in Model-Based Systems Engineering. 2019
10. [Online]. PRESPECTIVE software user documentation, 2020. https://unit040.atlassian.net/wiki/spaces/PUD/overview?homepageId=6979754
11. [Online]. Unity user documentation, 2020: https://docs.unity3d.com/
12. [Online]. Windows Communication Foundation for Unity, 2016 http://gyanendushekhar.com/2016/04/21/how-to-call-wcf-service-in-unity/
13. [Online]. Windows Communication Foundation, 2020 https://docs.microsoft.com/en-us/dotnet/framework/wcf/whats-wcf
14. [Online]. C# Design Patterns. 2020 https://refactoring.guru/design-patterns/
15. [Online]. Sample Use Case Example http://tynerblain.com/blog/2007/04/09/sample-use-case-example/
16. [Online]. Industry 4.0 and the digital twin https://www2.deloitte.com/us/en/insights/focus/industry-4-0/digital-twin-technology-smart-factory.html
17. [Online] PRESPECTIVE software: https://prespective-software.com/
18. [Online] Unity Game Engine: https://unity.com/
19. [Online] Visual Components: https://www.visualcomponents.com/
20. [Online] SolidWorks: https://www.solidworks.com/
21. [Online] Siemens NX: https://www.plm.automation.siemens.com/global/en/products/nx/

22. [Online] PIXYZ plugin for Unity: https://www.pixyz-software.com/plugin/

About the Authors

Enkhdavaa Batlkhagva received his Bachelor's degree in Electronics Engineering Technology from the Mongolian University of Science and Technology in 2015. After his graduation, he worked as a developer in a start-up company, which develops home automation systems. During his work, he specialized in the schema design of sensors and the implementation of wireless networks. In 2017, he was an intern at Innovations for High-Performance Microelectronics Institute in Frankfurt, Germany. The internship focused on the topic of a wireless sensor network, as well as field power measurement. Subsequently, he worked on a payment system of a bank as an embedded software engineer for almost one year. He is interested in Embedded Systems, Machine Learning, Internet of Things, and Software Architecture.

PO Box 513
5600 MB Eindhoven
The Netherlands
tue.nl

PDEng SOFTWARE TECHNOLOGY

EINDHOVEN
UNIVERSITY OF
TECHNOLOGY