Digital twins configurator for HIL simulations

Hoogstrate, Erick

Award date:
2022

Department of Mechanical Engineering
Control Systems Technology Group

Digital twins configurator for HIL simulations

Bachelor End Project Report

Erick Hoogstrate
1455176

Supervisors

dr. ir. J.M. van de Mortel-Fronczak
ir. L. Moormann
prof. dr. ir. J.E. Rooda

July 29, 2022

Abstract

Rijkswaterstaat is responsible for managing many of the road and waterway infrastructural systems in the Netherlands, including numerous locks. Since a few years, in cooperation with Control Systems Technology group of Eindhoven University of Technology, Rijkswaterstaat is investigating possible ways to renovate these systems in an efficient and cost-effective manner. The two research lines investigated in the Control Systems Technology group are related to the development of supervisory controllers using supervisory control theory and the design of lock product families.
Making use of supervisory control theory during the development of supervisory controllers results in a reduction in time to create and validate a supervisory controller. In addition, it also helps to reduce the number of errors encountered in the end, because supervisory control theory allows for intermediate testing of the controller through simulations. There are two simulation methods, one being software-in-the-loop and the other hardware-in-the-loop. With hardware-in-the-loop simulations, the controller is implemented on the actual control hardware, a programmable logic controller, and is connected to a virtual, visualized, model of the system, called the plant. When using software-in-the-loop simulations, the supervisory controller and the plant run on the same PC. Currently, the plant is visualized in 2D during simulation. A visualization gives the user many benefits, it makes it easier to understand the plant and the control system and validating supervisory controllers becomes easier as well, due to errors being easier to spot.
The visualization aspect can be improved by making use of a digital twin, a digital twin is a 3D simulation and visualization model that resembles the real system. In recent work, a digital twin of the Prins Bernhardsluis is created that is used for this project. The use of a digital twin for the visualization provides more benefits compared to the 2D visualization. As an example, it allows the visualization to be more detailed and it can be used for operator training, to name a few. Like the benefits show, it is beneficial to substitute the 2D visualization by a digital twin. This is already done for software-in-the-loop simulations and this project focuses on doing this for hardware-in-the-loop simulations.
To accomplish use a digital twin for the visualization in a hardware-in-the-loop simulation, a connection between Unity, the program used to create the digital twin, and the programmable logic controller needs to be established. Due to differences in interfaces, use is made of a tool, Prespective, to help bridge this gap. Because of a bug within the software of Prespective, the credentials used to secure the server, to which the programmable logic controller is connected, had to be removed and anonymous access had to be turned on. The only things left to do before a connection can be established is creating and trusting certificates. Because Prespective does not support certificates yet another tool is used for this, UaExpert. Once the certificates are created and trusted, it is possible to establish a connection between Unity and the server, and the digital twin can be used in hardware-in-the-loop simulations.

As mentioned before, digital twins have several advantages over 2D visualizations, but creating a digital twin takes a lot of time and effort. That is why the other research line, related to the design of lock product families, examines whether it is possible to simplify this process. In recent research all locks in the Netherlands are divided into seven clusters. These clusters can be used as templates in a lock configurator. By using a configurator it is possible to create a lock digital twin quickly and easily. For example, if the supervisory controller of a lock is going to be updated, a digital twin of the lock can be made first, after which the supervisory controller is validated in a hardware-in-the-loop simulation using the created digital twin.
The configurator is based on the properties of the seven clusters and uses the components of the Prins Bernhardsluis digital twin. At the moment, only the interface of the configurator exists and only the components of the Prins Bernhardsluis digital twin can be used. But once the configurator is finished, it is possible to make a digital twin for hardware-in-the-loop simulations. For hardware-in-the-loop simulations a graphical user interface is needed to provide the digital twin with signals. In this project a graphical user interface is created for an example digital twin, the Prinses Marijke lock complex, that could be created with the configurator. This graphical user interface is made modular, making it easier to adapt when a different lock configuration is used.

Contents

1 Introduction

In this chapter it is explained how models can help design control systems and what role digital twins play here. Next, the problem definition is given. After that the considered infrastructure is described. This chapter continues by giving the research objectives. Finally, the report structure is elaborated upon.

1.1 Project context

Some (components of) infrastructure, objects; and processes need some sort of control system to behave and operate in the desired way. One way to get to this goal is by making use of models. So called Model-Based Design (MBD) can help facilitate the ease of design by leveraging advanced processor functionality. Through this method functionalities can be optimized while overall design time can be minimized. It also provides the possibility to reuse models, this way simulation models can be used for or quickly adapted to its final application [1]. The general framework used for MBD, consists of the following steps:

1. Modeling a plant
2. Synthesizing and analyzing a controller for the plant
3. Simulating the plant and controller

As one can see, MBD only consists out of three steps. Making use of MBD results in rapid prototyping, software testing and validation. The disadvantages of MBD are for instance the additional time it takes to create the models and while models can be reused it often has to be adjusted to its specific application, a model of a tunnel cannot be used directly for a water lock as an example. Some advantages of MBD for design reuse are more apparent when the model applications are very similar. So for instance when one wants to create a model of a (different) water lock one can take an already existing model of a water lock and only make a few relatively minor adjustments to make the model work for that specific water lock. Since MBD provides a common design environment, it provides a general basis for communication between various (development) groups. It can also help find errors early in the design phase of a system to minimize the time and financial impact. In addition to this, it can provide a basis for so called hardware-in-the-loop (HIL) testing. With a HIL setup one can test the synthesized controller on the physical hardware it will run on eventually. In most cases this is a Programmable Logic Controller (PLC) [2]. Performing tests with a HIL setup could be useful to examine the dynamic behaviour and different cycle times (the time it takes for a signal to return back to the sender), which can vary due to the additional time required for data to travel to and from the physical hardware [3] [4]. In a HIL setup the synthesized controller is implemented on a PLC and executes code. Where the I/O of the PLC is connected to the I/O of the plant model. This plant model is currently visualized in 2D during simulations, this allows for better analysis and understanding of the model. The next step in visualization is to go to 3D, this provides an even better visualization manner. Besides testing on a HIL setup also a software-in-the-loop setup is used, where use is made of a soft PLC [5]. Soft PLC is a software technology designed to turn an embedded computer into a fully functional PLC. This permits the PLC to run on the same PC as the model, allowing for faster testing of controllers due to the minimal setup time.
Digital twin (DT), a digital counterpart of a physical object or process. The first idea of using a DT came from Michael Grieves at the University of Michigan at the beginning of this century in 2002 [6]. However, it did not get much traction until NASA started to use it in an attempt to improve physical model simulation of spacecraft in 2010 [7]. The following years DTs started to gather more and more traction and in the most recent years it started to expand to multiple different fields like healthcare [8] and infrastructure, to which this report applies. DTs have been linked to and are part of the so called Smart Industry or Industry 4.0, the fourth industrial revolution [9]. The fourth industrial revolution refers to the technological developments in the industry and the increasing manner companies become digital while they communicate, analyze and use information to increase its actions in the physical world [9] [10].

As mentioned before, a DT can be seen as a virtual representation that can serve as the real-time digital counterpart of a physical object or process. In [11] a nice example is given that compares a DT to scanning and archiving of a physical document. This is an overly simplified example as a document cannot be tested or simulated, this example is only used to clarify the general idea behind a DT. So instead of throwing away the physical version of a document one has two versions of the same 'thing'. This can be useful since one can use the digital version to test, simulate and analyze different scenarios instead of on the physical copy. This will save time and money, since by detecting mechanical failures before they can stop production, it is possible to perform easier and cheaper repair [12]. A DT can also help during the designing phase, as studies have shown that investing 5-10 percent more in up-front cost can reduce costs later on during the lifetime of the product by 50-100 percent. Since fewer issues are encountered [13].
It is also possible to create a DT of an already existing construction. Some advantage of using a DT on an already existing infrastructure are for instance the option to provide operator training [14]. By using a DT to train operators unnecessary downtime and cost can be avoided, since (part of) the plant does not have to be shut down. Another benefit is that the operator does not have to be on-site to get the training. Besides operator training a DT can also provide valuable information by performing tests and experiments on it. This can be useful when it is necessary to upgrade the object or process. For example, it can be used to test different supervisory controllers to increase the throughput, to test a new design that can be realised after renovations or the feasibility of a certain upgrade, to name a few.

1.2 Problem definition and research questions

Rijkswaterstaat is part of the Dutch Ministry of Infrastructure and Water Management and responsible for the design, construction, management and maintenance of the main infrastructure facilities in the Netherlands [15]. The need for this report arose through the ongoing collaboration between the Technical University Eindhoven Control Systems Technology Group and Rijkswaterstaat, abbreviated as RWS for the remainder of this report. RWS has a vast portfolio of different water locks. In the coming years these locks will eventually reach their end of life span and will have to be replaced or renovated. Since a few years, in cooperation with TU/e, RWS is investigating possible ways to renovate these systems in an efficient and cost-effective manner. One of the research lines is related to the development of supervisory controllers using supervisory control theory (SCT). Another research line is related to the design of lock product families.
All locks are designed for their local needs which has a negative impact on lock reliability and availability and the life-cycle-costs; primarily due to the need for local specialized knowledge to operate and maintain these locks and the need for many expensive and one-of-a-kind spare parts. By creating a so called family of locks which consists out of subfamilies, having common components, the negative impact can be reduced. Within these subfamilies locks can vary to meet the requirements of local stakeholders and to satisfy constraints of the local environment. A configurator is used to give the user the possibility to create a configuration based on a subfamily. Therefore, it is interesting to know in what way the current digital twin should be adapted to make it configurable for families of locks that can be used in HIL simulations. This is a fascinating topic, as there is no literature available about a lock configurator, indicating that this is a fairly unexplored topic. There is, however, literature available about a configurator for supervisory controllers of roadside systems [16].
This project is a continuation of the work in [17]. In [17], the DT of the Prins Bernhardsluis is created and the DT is validated using a synthesized supervisory controller for the Prins Bernhardsluis. This synthesized supervisory controller is based upon the files used for creating a synthesized supervisory controller for the Prinses Marijke lock complex [18], only a small adjustment is needed to make it work for the Prins Bernhardsluis. The simulation model with the synthesized supervisory controller of the Prins Bernhardsluis is shown in Appendix C. It forms the basis for the work described in this report. The already existing DT, together with the implemented controller, is used to set up and test the connection between the PLC and the simulation in the HIL setup. As it is previously only been used for software-in-the-loop simulations in a soft PLC setup. By connecting the DT to the PLC in the HIL setup hopefully more information can be gathered about the performance of the

controller. Besides this project there is also another relevant project. In this project digital twins have been developed for controller validation of workstations of a Festo production line in the course 4TC00 [19]. The main thing that is used from this work is the documentation.
The main goal of this project is to connect a digital twin to a PLC in the HIL setup. This part of the project runs parallel to [20]. In [20] a DT of the Swalmentunnel is used. Because it does not matter which DT is used for connecting the DT to a PLC in the HIL setup, it is decided to use the tunnel DT. The first research question is defined as:

1. What is needed to connect a DT to a PLC in a HIL setup?

Besides the first research objective there is also a second one, namely designing and creating a lock DT-configurator. This configurator should be able to create different configurations of locks of the same family. The purpose of this configurator is to create different configurations of locks to test the supervisory controller on. This defines the second research question:

2. What should be done to make the current DT of the Prins Bernhardsluis a suitable configurator basis for a family of water locks?

1.3 Report structure

The structure of the report is as follows. Chapter 2 is devoted to the methods and tools used for this project. In this chapter the used software with its application is described. The next chapter Chapter 3, talks about the workings of water locks and some more background information about locks in general, like what constitutes a family of locks. Chapter 4 contains information about the HIL setup. The second research objective is worked out in Chapter 5. Finally, Chapter 6 describes the conclusions from this research and gives recommendations for any potential follow up research.

2 Methods and tools

This chapter describes the methods and tools used in this project. First, the concept of a supervisory controller is discussed, after which the traditional engineering method, to create supervisory controllers, is compared to the new method, that is based on supervisory control theory. Next, a brief description will be given on the approach for the lock configurator. Lastly, the software and tools required in this project are detailed.

2.1 Supervisory control

In Section 1.2, it is mentioned that RWS is investigating two research lines to renovate their systems in a cost-effective and efficient manner. One of these research lines is about the development of supervisory controllers using supervisory control theory (SCT). A supervisory controller is used to control the different components and subsystems of the entire system, called the plant, such that it behaves as intended [21]. In Figure 2.1 a schematic overview of an infrastructural control system is given. Here, an operator issues commands through the Graphical User Interface (GUI), which are checked by the supervising controller before being sent to the plant. The steps a signal goes through when an operator gives an input are as follows:

- The operator interacts with the GUI.
- The GUI sends signals to the supervisory controller.
- The supervisory controller checks if the sent signal is allowed based on the state of the plant.
- If the signal is allowed by the supervisory controller, it gets sent to the actuators of the plant.
- In the plant the state of the mechanical components is changed by the actuators.
- When the state of the mechanical components in the plant changes, this is detected by the sensors.
- The sensors feed signals back to the supervisory controller.
- The supervisory controller sends a signal to the GUI to update the visualization of the plant, it can also send another signal to the actuators.

The supervisory controller forms the layer between the plant and the GUI. A supervisory controller is able to block signals to the actuator to prevent unwanted, unsafe behavior, like opening both lock doors at the same time. Signals sent by the sensors are always able to occur as these are uncontrollable, compared to the controllable signals of the actuators.

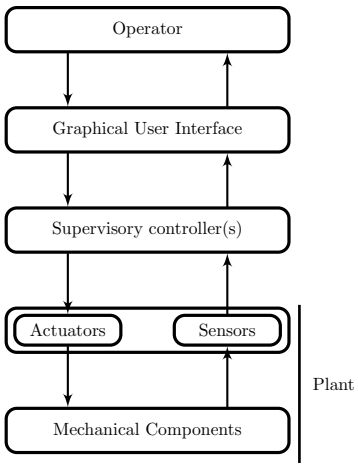

Figure 2.1: Schematic view of a control system structure of an infrastructural system (adapted from [21]).

2.2 Supervisory control theory

Now that it is clear what role a supervisory controller plays in an infrastructure system, the next step is to explain how a supervisory controller is made. In the past, supervisory controllers are made by hand, based on control requirements. This process is schematically depicted in Figure 2.2. A document icon indicates a documented product and a square indicates a realized product [22].

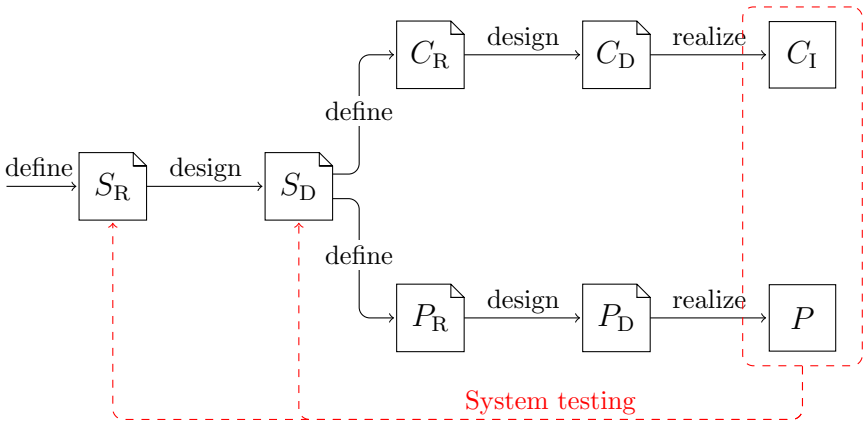

Figure 2.2: Schematic overview of the traditional engineering method [22].

The first step of this traditional engineering method is to define system requirements (S_R), after which a document-based design of the system is made (S_D). The controller and plant requirements (C_R and P_R, respectively) are defined based on the document-based design of the system. This step is followed by the creation of the document-based design of the controller and plant (C_D and P_D, respectively). Finally, the controller and the plant are realized (C_I and P, respectively). Once the last step has been completed, it is possible to test the system. If errors are then discovered, the system design has to be adjusted or even the system requirements. If this is the case, one is back to square one. That is why the traditional engineering method works quite slowly, also because it is not possible to test in the meantime [22].
To solve the problems of the traditional engineering method, a new method is developed, Synthesis-based engineering (SBE). This new method has a modified overview, which is depicted in Figure 2.3.

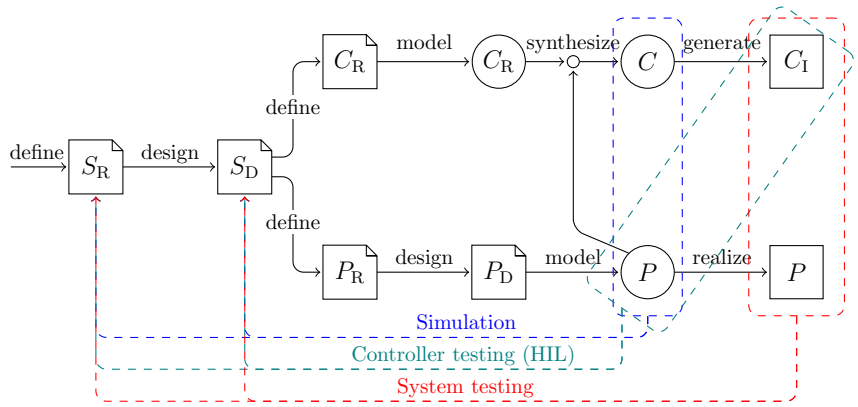

Figure 2.3: Schematic overview of the synthesis-based engineering method [22]

Circles are added to this overview, these circles represent models [22]. These models are used for testing before the plant is realized. To use these models, a design of the plant (P_D) is first made and the requirements of the controller (C_R) are modelled. Once these steps are completed it is possible to synthesize a controller (C). This controller (C) can be used with the model of the plant (P) to

perform simulations, these simulations can be used as intermediate test. It is possible to use a DT to represent the plant in these simulations. If problems are encountered, it is possible to move back to an earlier stage to solve them, this is already an improvement over the traditional engineering method. Another improvement over the traditional engineering method, is the option to validate the supervisory controller in a HIL setup. This options provided an additional stage of validation before the supervisory controller is implemented on the realized plant.
In a HIL setup, the supervisory controller is generated and put on a PLC, also for these simulations it is possible to use a DT to visualize the plant. The advantage of testing supervisory controllers in a HIL setup is that the performance of the controller can be tested on (final) hardware. This is important, because minor changes can occur once hardware is used for controller testing. For example, a difference in cycle time can affect the performance of the controller. If there are no problems found, it is possible to realize the plant and to test and validate the system.

2.3 Approach configurator

The intention for the configurator is to use as much of the old DT as possible. Everything should be made as modular as possible and the configurator should be easy to use, i.e. set up as much as possible.

2.4 Software

For everything to run smoothly the correct software is needed. In [23] more information can be found about the main software that is used to create a digital twin. The main suite of software that is used for this project consist out of the following programs:

Unity

Unity 3D[1] is a game engine that was used to develop the Prins Bernhardsluis DT. 3D-models from an online warehouse[2] or created in Siemens NX[3] are loaded into this game engine. The interaction and behaviour of different components are modeled using various tools, among which are Prespective[4] and built-in tools from Unity. The Unity version used during the development of the DT and that is used in this project is Unity version 2019.3.10f1. Which can be downloaded from the Unity download archive[5]. The main purpose of Unity is the visualization aspect of the DT.

Prespective

Unity itself is a game engine, which is not capable of controlling the DT on its own. To do that a plugin for Unity called Prespective[4] is used to implement essential tools. Prespective adds realistic standard physical components like motors and indicator lights. These are, for example, used to move the lock gates. The most useful addition that Prespective provides is an interface between Unity and a (soft) PLC. Version 2020.1.1500.2 of Prespective is used for this project.

TwinCAT 3

Twincat[6] is used to run the soft PLC. The soft PLC is used to test the functionality of the supervisory controller, the existing DT and configurations made with the configurator.

[1] https://unity.com/
[2] https://3dwarehouse.sketchup.com/
[3] https://www.plm.automation.siemens.com/global/en/products/nx/
[4] https://prespective-software.com/
[5] https://unity3d.com/get-unity/download/archive
[6] https://www.beckhoff.com/en-us/products/automation/twincat/

CIF 3

CIF 3, the Compositional Interchange Format for hybrid systems. CIF is an automata-based modeling language for the specification of discrete event, timed, and hybrid systems. The CIF 3 tooling supports the entire development process of controllers, including among others specification, supervisory controller synthesis, simulation-based validation and visualization, verification, real-time testing, and code generation. CIF 3 was created and is currently developed by the Systems Engineering group of the Mechanical Engineering department, at the Eindhoven University of Technology (TU/e). The CIF 3 tooling is free, and is available under the MIT open source license[7]. The CIF 3 language is used to create the PLC code from the supervisory controller.

IDE

Since CIF 3 is a modeling language, an editor is required. For this the use is made of the Eclipse IDE[8]. IDE stands for Integrated Development Environment. This is a computer program that supports the developer during development. It can be seen as an editor in which code can be created, edited or removed. Besides Eclipse, also use is made of Visual Studio or Visual Studio Code and sometimes even Notepad++, while the latter is considered a text or source code editor. Eclipse is used to run the CIF 3 code and to extract the variable names of the components from the model. The other editors are used to create or edit scripts for the configurator or various other tasks.

ABB Compact Control Builder

The ABB Compact Control Builder is a program installed on the Engineering PC that communicates with the PLCs of the HIL setup[9]. It is used to upload the PLC code to the PLC hardware.

Ignition OPC server

The OPC server runs within the Ignition server and allows for data sharing between the Ignition server and the PLC. OPC, or *'Open Platform Communications'*, is an industry-wide standard that can be used for the secure and reliable exchange of data in the industrial automation space or in other industries[10]. The Ignition OPC server is hosted on the Engineering PC.

ABB OPC DA server

The ABB OPC DA server enables a connection from the outside to the variables present on the ABB PLC. This server is already setup correctly, and thus is largely left alone for this project. However, this is a vital part of the resulting setup.

Ignition OPC UA server

The Ignition OPC UA server enables connectivity between Ignition clients and the Ignition OPC server.

Ignition Designer

The Ignition Designer is used to design and run the GUI. The Ignition Designer is able to connect to the Ignition server to add, change or delete data. This property is used for the connection between Unity and the PLC[11].

[7] https://cstweb.wtb.tue.nl/cif/trunk-r9682/
[8] https://www.eclipse.org/ide/
[9] https://new.abb.com/control-systems/essential-automation/compact-product-suite/essential-controller-suite/compact-control-builder
[10] https://opcfoundation.org/about/what-is-opc/
[11] https://inductiveautomation.com/ignition/designer

Wireshark

Wireshark is a network protocol analyzer, that allows the user to see what is happening on their network[12]. This is used to detect if data is sent over an Ethernet cable in the HIL setup during remote testing.

UaExpert

UaExpert is a full-featured OPC UA client, that is used to test the connection to an OPC server[13]. It is also used for creating and accepting certificates that are needed for the connection between Unity and the Ignition OPC server.

OPC UA ANSI C Demo Server

The OPC UA ANSI C Demo Server, abbreviated as demo server, is a simplified third-party OPC UA server that is used to test certain connections[14]. For instance, to check if it is possible to connect to Unity from another PC hosting the demo server.

[12] https://www.wireshark.org/
[13] https://www.unified-automation.com/products/development-tools/uaexpert.html
[14] https://documentation.unified-automation.com/uasdkc/1.4.0/html/demoserver.html

3 Water locks

A water lock is used by vessels to navigate between different water levels. A conventional lock works by increasing or lowering the water level inside a watertight chamber by means of gravitation; larger and or modern locks may use pumps. However, there are also other types of locks used throughout the world to move vessels from one level to the other. They mainly operate by the rise and fall of the chamber itself, in that case usually called a caisson. A fairly well known example is the Falkirk wheel in Scotland [24]. This chapter explains how a pound lock works in general followed by the main building blocks of the considered locks in this report. After that, the configuration of the Prins Bernhardsluis is discussed. Furthermore, the families of locks are elaborated upon.

3.1 Functioning of a lock

When a vessel would like to move from a lower to a higher water level it starts its journey by entering the chamber from the lower water level, see the top illustration in Figure 3.1. Once the vessel has entered the chamber the 'downstream' (the lower water level) gate can be closed. Next, small doors inside the larger gate, called (levelling) paddles [25], open up to let water in from 'upstream' (the higher water level), see the middle illustration in Figure 3.1. Once the water level inside the chamber has equalized with the water level upstream, the upstream gates can be opened and the boat can continue its journey, see the bottom illustration in the figure below. This process is reversed when a ship wants to travel from a higher to a lower water level.

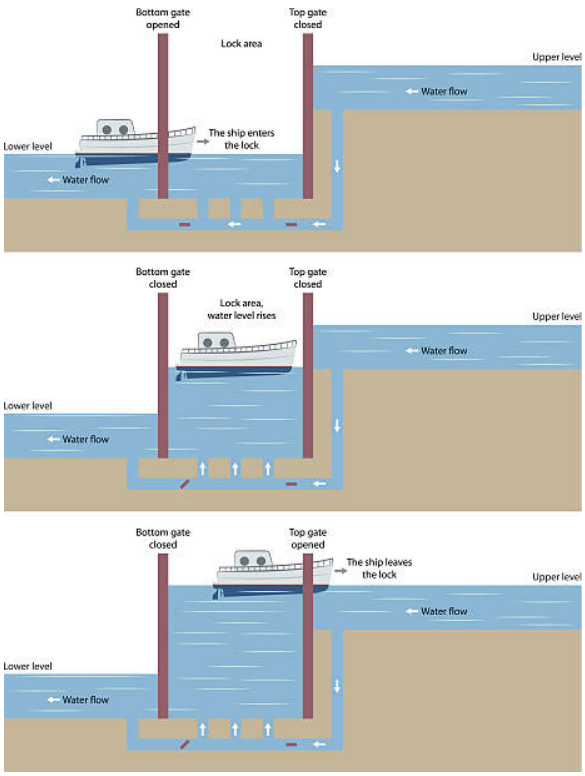

Figure 3.1: Functioning of a water lock [26].

3.2 Building blocks

Larger installations or systems can often be divided into smaller subsystems, which form the building blocks of that system. This is also the case for water locks. Water locks consist out of different variants of the following main subsystems: traffic lights, doors and levelling system. Of course there

are more options for the various subsystems than only the ones mentioned below but only the ones used in the Prins Bernhardsluis and the configurator are discussed in this chapter.

Traffic lights

Traffic lights are used to communicate the state of the lock to the vessels that want to make use of it. The meaning of the different states can be found in Figure 3.2.

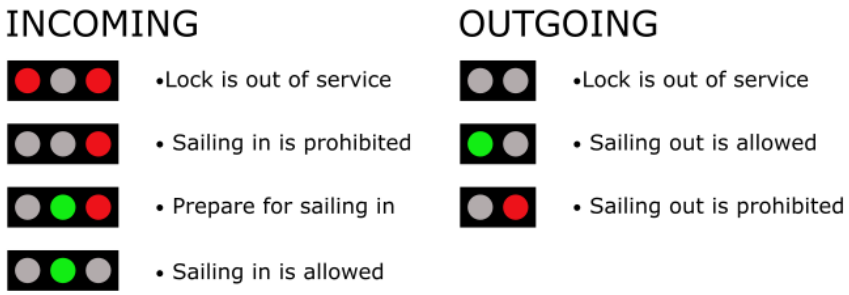

Figure 3.2: Different states of the traffic lights and their definition

For the Prins Bernhardsluis, when the first chamber is not being used, the outgoing traffic lights involved in operating the first chamber, indicated by the red ellipse in Figure 3.3, are turned off. Indicating that the first chamber is not used.

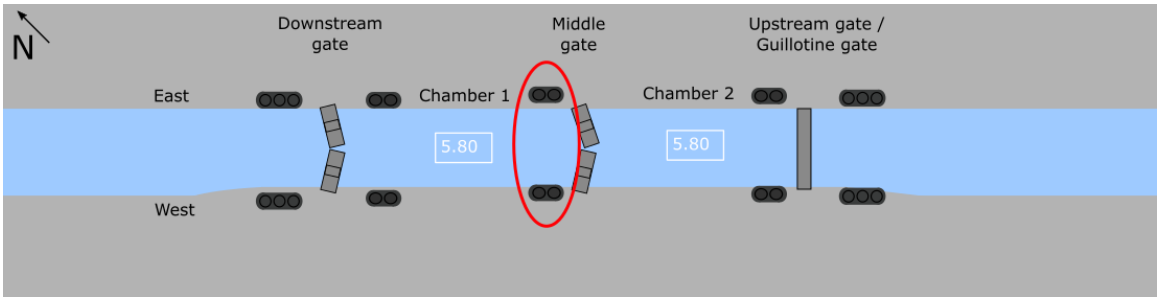

Figure 3.3: Traffic lights of the middle gate

Mitre gates

Mitre gates, invented by Leonardo da Vinci, are a set of doors that combined have a length that is longer than the width of the lock. As a result the doors close under an angle, with the point facing upstream. The doors cannot be used until the water level is equal on both sides of the door. Because the point of the door is directed towards the highest water level, upstream, the pressure from the higher water level will cause the doors to be pushed shut until the pressure on both sides of the doors is equal. When this is the case, the doors of large locks can be opened by means of hydraulic, mechanical, or electrical power. So the hydraulic, mechanical, or electrical power is used to move the gate when the water level on each side is equal and the pressure difference between the two water levels will keep the doors shut when there is a difference in water level.
For the Prins Bernhardsluis only a single set of doors is used on each side of a chamber, this is depicted in Figure 3.3. It is also possible to have double mitre gates, shown in Figure 3.4. Double mitres gates are used in scenarios where the tides play a large role. When the water level in the downstream direction is higher than that of the upstream direction due to tides, a single set of mitre gates will no longer function. Because the doors are no longer pushed closed, but rather open. This is the reason why mitre gates are considered mono-directional gates. They only work if the pressure difference is coming from one direction. A solution to this is a double set of mitre gates where the

mitre gates both point in a different direction, seen in Figure 3.4. By using a double mitre gate it does not matter which direction has the highest water level. In this project only a single set of mitre gates will be considered.

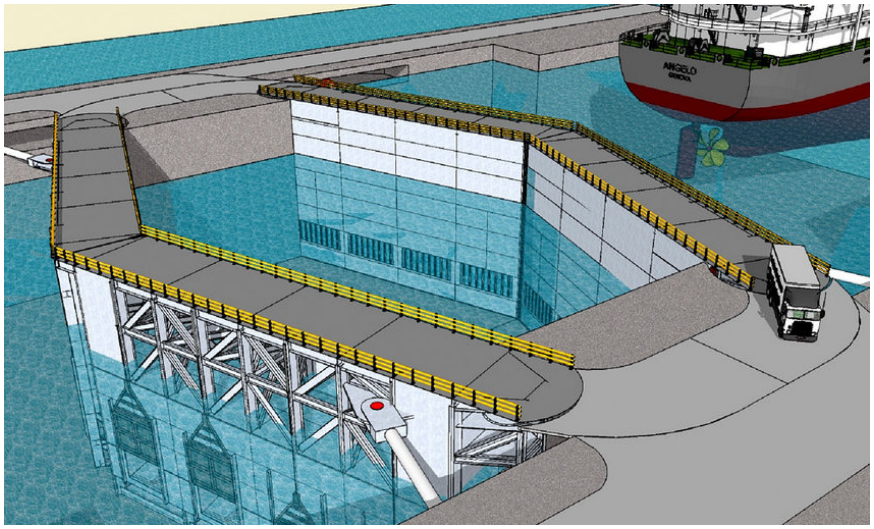

Figure 3.4: Mitre gates [27]

The doors used for this project are adjusted by hydraulic actuators. The hydraulic cylinder can extend and retract at two different speed values, fast and slow, together with a rest position where the velocity is equal to zero. A slow velocity is used for the first and final degrees of movement and prevents the gates from hitting each other or the wall at full speed. The closing and opening of the gate happens according to the following velocity profile. In the first five seconds the gate moves at a rate of 5 mm/s, then the gate accelerates to 46 mm/s before it slows down again to 5 mm/s.

Guillotine gate

Guillotine gates, as the name might suggest, have a strong resemblances with the French guillotine. This type of door simply works by moving a door vertically downwards to close off one side of the chamber. Since the shape of the gate is the same on each side it does not matter which side of the door a higher water level or pressure has. This is useful when the side with the highest water level varies due to tide. These type of doors are called bi-directional gates as they can operate in both directions.

Figure 3.5: Guillotine gate [28]

Leveling paddles

To let water get in or out the chamber use is made of so called leveling paddles. These can be seen in Figure 3.4 and Figure 3.5. The leveling paddles cover holes in the gate, by moving each paddle individually up or down the amount of water that flows in or out the chamber can be regulated.

3.3 Prins Bernhardsluis

The Prins Bernhardsluis which was used to create the DT in [17] is located in Tiel, the Netherlands, can be seen in Figure 3.6. The lock connects the river the Waal to the Amsterdam-Rhine channel, which are both at different water levels. The lock actually consists of two separate water locks. The water lock on the left in Figure 3.6 is built in 1952. This lock is much older compared to the right lock, which is built in 1974 and is therefore referred to as the old lock. The newer lock is mainly used for facilitating the passage of push barges, which are vessels used for transporting cargo. For this report the old lock is considered, as this is the first lock that RWS will have to renovate.

Figure 3.6: Prins Bernhardsluis

3.4 Prins Bernhardsluis

As mentioned before in this report only the oldest lock is considered, sometimes also referred to as the western lock. The old lock consists out of the following components from Section 3.2, which can be seen in Figure 3.7.

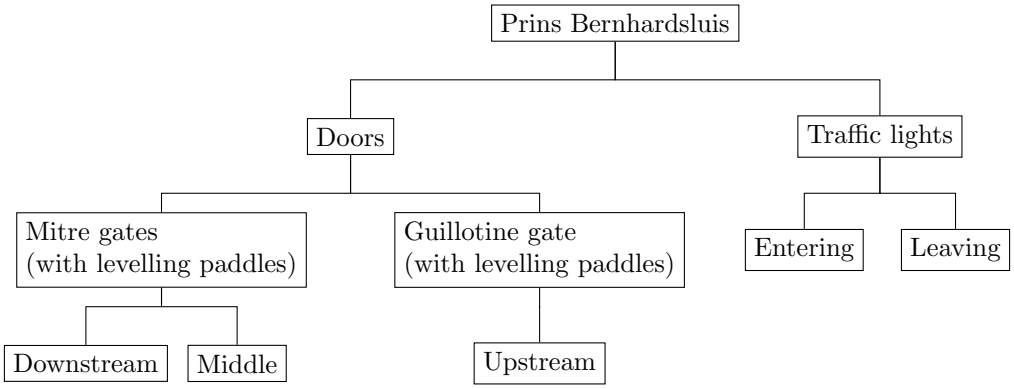

Figure 3.7: Components of the Prins Bernhardsluis

The old lock contains three gates, while only two are required for a lock to function properly. The reason for adding an additional gate is to create a second lock chamber. By constructing an additional lock chamber the effective length of the lock is reduced. This has as benefit that the amount of water that has to be leveled is reduced and hence also the locking or waiting time. This smaller lock is mainly used when only a few (smaller) vessels need to make use of the lock. This can be seen in Figure 3.3.

3.5 Families of locks

RWS has a vast portfolio of locks. Currently all locks are specially designed for the location they are located at. This has a negative impact on lock reliability and availability and the life-cycle-costs. However, from Section 3.2 follows that most locks consist of certain building blocks. With this information a distinction can be made between different groups of locks which have certain

components in common, which than form a so called (sub)family. To know which lock falls under which group or family the information from a report which performed a similarity, modularity and commonality analysis of navigation locks in the Netherlands [29] is taken. They made use of so called Dependency Structure Matrices (DSM), which is shown in Figure 3.8, to find commonalities among the various locks. By making use of these DSM's seven groups or clusters are created. The Prins Bernhardsluis can be found in the report in Cluster 4; the old lock has entry 65 and the new lock entry 60. However, when discussing the families of locks or the configurator in the remainder of this report, information from Cluster 5 of Figure 3.8 is considered. The reason for this is that the Prinses Marijkesluis lock complex, row 75 and 78 in Figure 3.8, belongs to this cluster. This lock complex is chosen because there are already models and code present at the TU/e, this includes a simulation model with CIF 3 code and PLC code.
As the Prinses Marijkesluis lock complex is part of Cluster 5 it must also adhere to the properties of Cluster 5. Locks in Cluster 5 have the following properties:

- Mono-directional water retention
- Single head high water retention
- Lock chamber width between 10 and 20 meter
- Mitre gates (primarily electromechanical)
- Gate openings (primarily electromechanical)

A configuration based upon Cluster 5 will only have mitre gates as doors with paddles, the paddles are used for equalizing the water levels. These "building blocks" are described in Section 3.2. Single head high water retention indicates that only a single set of doors is used for each side of a chamber, similar to Figure 3.3 [30]. To satisfy the mono-directional water retention property, all mitre gates should point in the same direction. If this is not the case, the doors would break open due to the force of the water, as described in Section 3.2. Finally, the lock chamber width van vary between 10 and 20 meter.

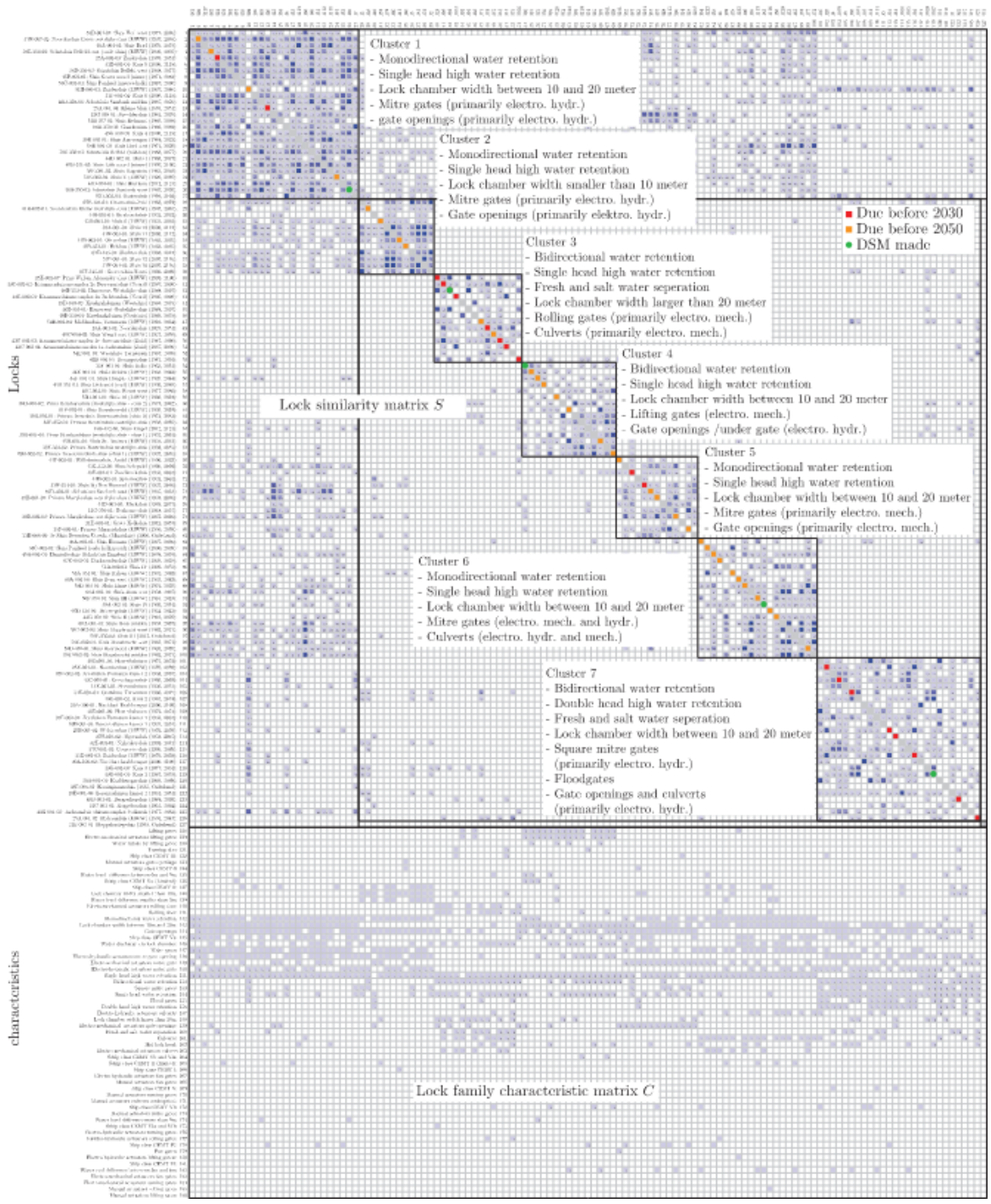

Figure 3.8: Dependency Structure Matrices of navigation locks in the Netherlands [29]

3.6 Conclusion

This chapter explained how a pound lock works and it detailed which "building blocks" are considered in this report. Next, general information and details about the Prins Bernhardsluis are given, which includes a break down of the components of the Prins Bernhardsluis. Finally, the families of locks are discussed, here it is mentioned that Cluster 5 from Figure 3.8 will be used for this project.

4 DT connection to HIL setup

In this chapter, the technical details of the hardware-in-the-loop (HIL) are discussed. This is done by first explaining the topology of the elements in the network of the HIL setup, followed by the steps taken during this project for setting up a connection between Unity and the PLC. The encountered problems and a solution for connecting the variables between Unity and the PLC are explained in detail. A detailed step-by-step guide for setting up a connection between Unity and the PLC and the explanation how to connect the tags for data exchange, are provided in Appendix A. For this part of the report the tunnel DT is used, as mentioned before.

4.1 Network and data topology

The HIL setup that is used in this project consists of multiple elements: three PLCs, three laptops and one network switch. In Figure 4.1 the network topology of the setup that is present at the TU/e is depicted. This setup is used for HIL simulations with 2D models.

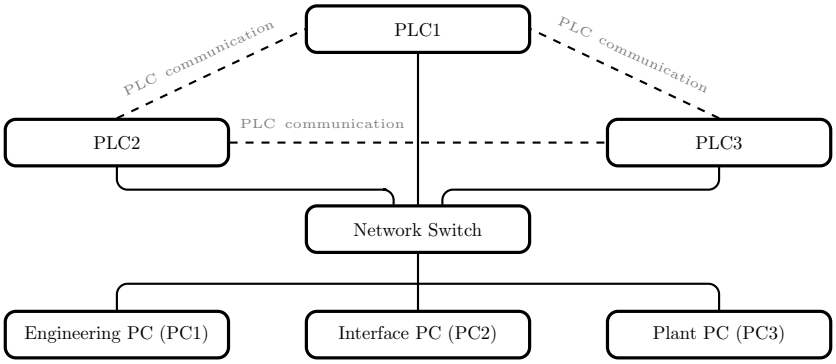

Figure 4.1: Network topology of the HIL setup present at the TU/e [31]

Network topology refers to the arrangement of the elements in a (communication) network. The HIL setup present at the TU/e has three PLCs which are connected with a network switch. This setup is used in a previous project to measure the effect of signal delays, between multiple PLCs, on the performance of the controller. The HIL setup that is used in this project makes use of an adapted network topology, which is visualized in Figure 4.2. This setup is only physically connected to one PLC, compared to the three physical connections to the PLCs previously, as this is already sufficient for the tasks performed in this project. There are three PCs included with this setup: the Engineering PC (PC1), Interface PC (PC2) and the Plant PC (PC3). The Engineering PC is used to upload the PLC code to the PLC and to host the OPC server, the Interface PC hosts the GUI and the Plant PC hosts the DT. It is also possible to run the interface on the Engineering PC rendering the Interface PC unnecessary, which allows for a setup with two PCs. There is also a possibility to only use one PC but this would have to be the Engineering PC, as this one hosts the OPC server. The minimum connection for the setup to operate is indicated by the bold line in Figure 4.2.

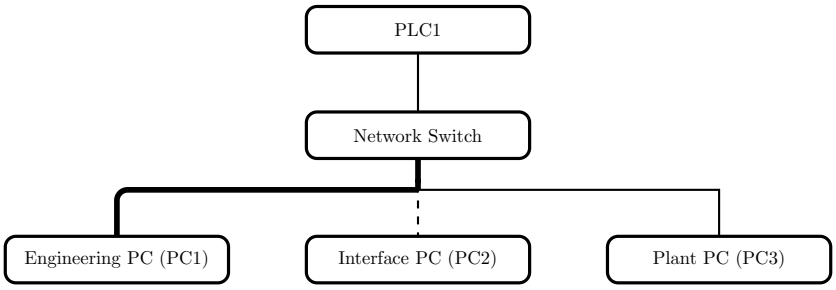

Figure 4.2: Adapted network topology

Besides a network topology there is also a data flow schema. This shows the paths in which data moves. The original data network that is used for 2D HIL simulations is schematically visualized in Figure 4.3. The Engineering PC hosts the ABB OPC DA server and the Ignition OPC UA server. The ABB OPC DA server interacts with the ABB PLC network as well as with the Ignition OPC UA server. The Ignition OPC UA server in turn communicates with the Ignition clients on the Interface and Plant PC.

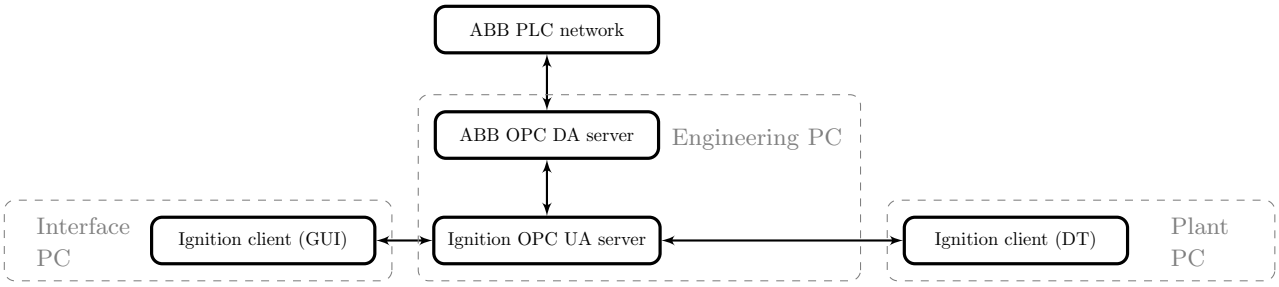

Figure 4.3: Data flow schema

4.2 Setting up a connection

A connection between Unity and the PLC is required for the HIL setup. Otherwise it would not be possible for the controller on the PLC to send or receive commands, likewise it would also not be possible for the DT to function. There are some differences compared to the previously used setup. The first difference is that Unity is a third party service compared to the native Ignition service that is used in the previous setup. In the previous setup, the Ignition Designer Launcher software is used on the Interface and Plant PC for the operator interface and visualization of the plant, respectively. Ignition clients on these PCs are used to communicate with the Ignition OPC UA server on the Engineering PC which stores all tags, which can range from memory tags that store a certain, possibly self assigned, value to OPC tags that link directly to a PLC variable on the ABB OPC DA server. The data flow representing this is visualized in Figure 4.3. Since most of the software in this setup is from the same company, Ignition, the communication between the different PCs is easily established. Because Unity is a third party service this process is less streamlined.
Unity itself is not able to connect to an OPC server. To accomplish this, use is made of the Prespective plug-in. Ideally, one wants to directly connect to the ABB OPC DA server, indicated by the black arrow in Figure 4.4, considering that all the PLC variables are stored there. However, the Prespective plug-in only provides an OPC_UA adapter, which means only the connection indicated by the red line in Figure 4.4 is possible.

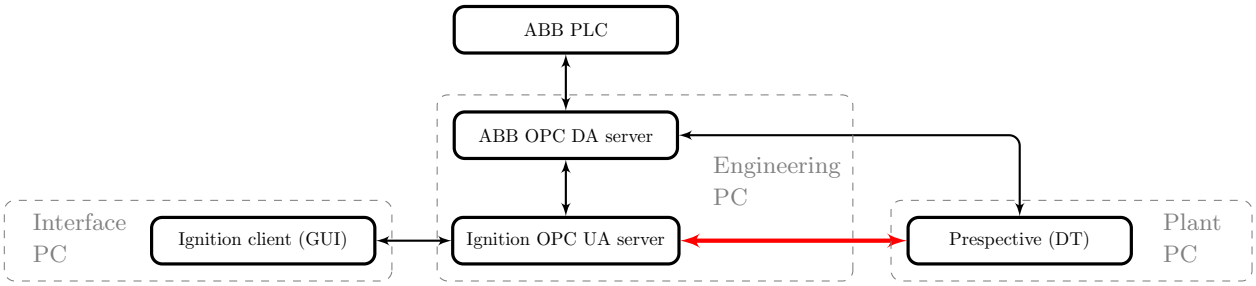

Figure 4.4: Data network schema directly to ABB OPC DA server

Unity

The Prespective plug-in gives the user the ability to add a script to the DT to connect it to the OPC UA server by means of an adapter. To do this the following steps are taken:

- First a new GameObject is created in the scene, `Ctrl + Shift + N` or `GameObject > Create Empty`.
- Next, the component: *Pre Logic Simulator* needs to be added to the GameObject.
- In the Pre Logic Simulator the adapter settings can be changed. The first setting that needs to be changed is the *Adapter Target*, this needs to be set to "OPC_UA", see Figure 4.5.
- The next setting, *Endpoint URL* requires the use of the Engineering PC and the Ignition Gateway. On the Engineering PC open a browser and go to: *localhost:8088*.
- Once on the webpage, go to `Config > OPC client > OPC connections > more > endpoint` and copy the shown Endpoint URL, which should look like something along the lines of *opc.tcp://localhost:62541/discovery*. This needs to be pasted into the corresponding setting in Unity.
- The *configurationFilePath* setting is not changed as this should be set automatically correct. When this is not the case, it should be set to the path of the configuration file that contains the configuration XML.
- The *namespaceIndex* and *IdentifierTypeOpc* are not changed for the moment and kept on their initial values of "0" and "Numeric", respectively.
- For the *UserName* and *PassWord* the credentials of the setup are used, found in [31].
- Finally, these settings are exported by pressing the `Export Policy` button below the adapter settings.

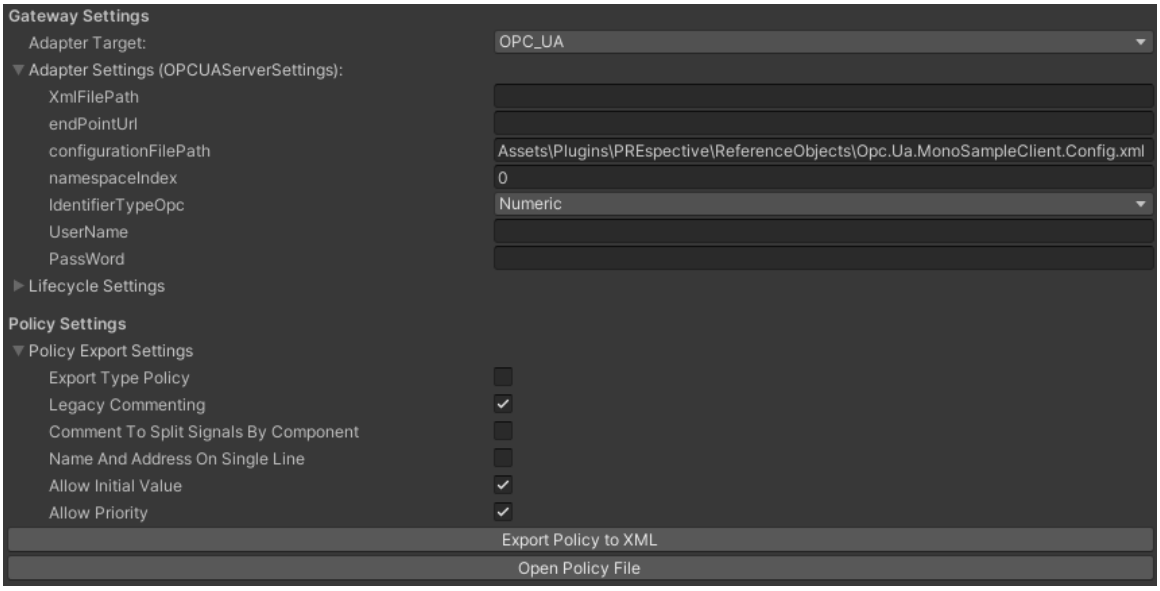

Figure 4.5: Gateway adapter settings in Unity

Wireshark

Subsequently the connection can be tested by going into *Play Mode* in Unity. This results in a BadNotConnected error. To find the cause for the unestablished connection, a program called Wireshark is used. This program can visualize what data is sent over an Ethernet cable. Since there are very few settings and the assumption is made that the username and password are correct, only the Endpoint URL is examined. Changing the Endpoint URL caused some changes in the data that is sent over the Ethernet cable via the TCP protocol. Some changes that are tried: removing the "/discovery" and changing the port. For the port, different values are selected which can be found in the Ignition Gateway. Besides the standard port, 62541 also port numbers 8088 and 8043 are tested. These ports represent the HTTP and HTTPS ports respectively. These can be found on the following page, `Config > Network > Web Sever`. The explanations next to the ports in the Gateway tell that Ignition will listen on these ports for incoming HTTP or HTTPS signals. Using these ports caused some changes in the data sent. The meaning of these messages is not explained by Wireshark which resulted in speculation of what the correct Endpoint URL is. To avoid guessing it is assumed that the Endpoint URL given by the Gateway, `opc.tcp://localhost:62541/discovery`, is correct. Eventually, the following conclusions are made:

- **BadNotConnected**

  Request to connect to a remote server was unsuccessful, and no response was received. This can be caused by using the wrong Endpoint URL.

- **BadSecureChannelClosed**

  Request to connect to a remote server was unsuccessful, a response from the remote server was received, but the connection was disconnected abruptly.

Policy file

Since all possible combinations are tried with the Endpoint URL, without much result, the next possible cause, the credentials, are examined. Upon further investigation it is discovered that something peculiar happens within the XML file. The username and password cannot be found inside this file, as can be seen in Listing 1. One theory is that the username and password are stored somewhere else or are encrypted in some way for security reasons.

```
<ControllerSpecification>
  <ActiveController>
    <OPC_UA>
      <Endpoint>opc.tcp://localhost:62541/discovery</Endpoint>
      <NamespaceIndex>0</NamespaceIndex>
      <IdentifierType>i=</IdentifierType>
    </OPC_UA>
  </ActiveController>
</ControllerSpecification>
```

Listing 1: Policy.xml

Contact with Prespective

Due to the bug found in the XML file concerning the credentials and not having a connection established, it is decided to contact Prespective, the supplier of the software used, as it seems that the reason for not being able to create a connection has something to do with the software of Prespective. It is tried to make contact with Prespective on several occasions. Unfortunately, if any help was provided at all, only a short tip which showed that they failed to understand the problem faced was given. Eventually the communication from their side stopped entirely. Which is remarkable, since this company has worked before with the TU/e on DTs. In particular the creation of a DT of a Festo machine, for a course taught at the TU/e [19].

UaExpert

The documentation from Prespective is quite mediocre, having said that the documentation got updated some time after our attempt at contacting Prespective. One of the changes in the documentation pointed out that Prespective and Unity are not (yet) capable of accepting certificates and another tool has to be used for this, called UaExpert. UaExpert is designed as a general purpose OPC UA (test) Client. Which means that is can also be used for more applications than only accepting certificates. It can also be used for setting up and testing connections with the OPC UA server. A connection is tried to be established between UaExpert and the OPC UA server by using the Endpoint URL provided by the Gateway. That being `opc.tcp://localhost:62541/discovery` with localhost changed to the ip-adress of the Engineering PC for the connection from the Plant PC to the Engineering PC. Resulting in `opc.tcp://172.16.0.10:62541/discovery`. Regardless of how, no connection is possible between UaExpert on the Plant PC and the OPC UA server on the Engineering PC. As it already not possible to connect to the OPC UA server with UaExpert, a (basic) general purpose tool, it was determined that a connection with Prespective is out of the question. Being that the Gateway provides an Endpoint URL which contains localhost, meaning it can only be accessed locally on that PC, with that PC being the Engineering PC, it is tested if UaExpert on the Engineering PC is capable of connecting to the OPC UA server. This turned out to be the case. After setting up this initial connection the certificates are trusted within UaExpert. See the step-by-step guide in Appendix A.3, item 4 that explains the steps required for setting up a connection with the OPC UA server and accepting the certificates within UaExpert. As it is possible for UaExpert to connect to the OPC UA server on the Engineering PC it is decided to hosts Unity, and hence the DT, "locally" on the Engineering PC. Which results in the final data flow schema as seen in Figure 4.6.

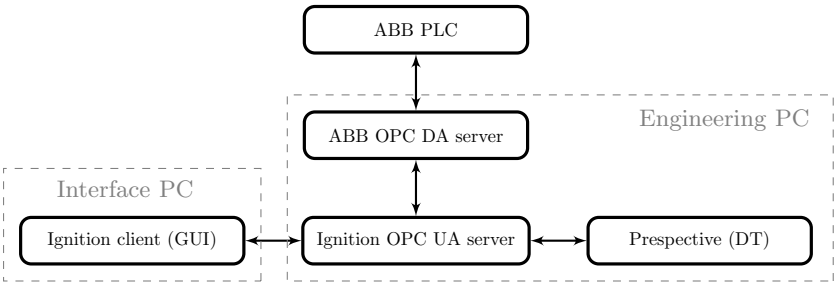

Figure 4.6: Final data flow schema

Credentials

Testing the connection between Unity and the OPC UA server again did not result in a success or a new error, which means that the problem still needs to be found elsewhere. To narrow the problem down to either Unity (client side) or Ignition (server side), use is made of the *OPC UA ANSI C Demo Server*. The results from some connection tests can be seen below in Table 4.1. It is also noted which server makes use of credentials and which does not. It should however be noted that when a connection to the OPC UA server from UaExpert is established a popup menu appears that asks for the credentials. This indicates that there is a request from the server for the credentials and it can be concluded that the credentials are sent to the server, since the connection is established upon filling in the correct credentials. With the information from Table 4.1 it is observed that Unity is capable of connecting to an OPC UA demo server which has no credentials. However, it does have trouble establishing a connection with the OPC UA server which has credentials, while UaExpert is capable of connecting to the OPC UA server perfectly fine. This leads to the conclusion that Prespective does not send the credentials properly with the connection request to the OPC UA server.

Table 4.1: Connection table

From:	To:	Credentials?	Connection?
UaExpert	Demo server	No	✓
UaExpert	OPC UA server	Yes	✓
Unity	Demo server	No	✓
Unity	OPC UA server	Yes	✗

Removing credentials from OPC UA server

The conclusion that Prespective does not send the credentials properly with the connection request to the OPC UA server, lines up with the earlier observation of the missing credentials in the XML file. A solution was proposed by manually adding the credentials to the XML file. As there is no template of how the XML file should look like with credentials, this inevitably ended up not working. Therefore another solution is proposed, which is, turning off the credentials of the OPC UA server. This can be done in the Ignition Gateway under `Config > OPC client > OPC connections > Ignition OPC UA Server > edit`. Leave the username and password empty, and save the changes. Besides removing the credentials, also anonymous access is turned on to compensate for the missing credentials. Allowing for anonymous access can be done via `Config > OPC UA > Server Settings > Anonymous Access Allowed` and toggling it to true. While in the Gateway it is also discovered that the certificate needs to be accepted on the server. This can be done by going to `Config > OPC UA > Security > Server`. Trusting the certificates of UaExpert and Unity by clicking the green "Trust" button next to the certificates.

Proper connection

After applying these changes the Ignition Gateway needs to be restarted. This can be done in multiple ways. For instance by restarting the Engineering PC or by opening Command Prompt as an administrator and typing `net stop ignition` in the Command Prompt window, waiting until the messages say that Ignition is stopped, followed by `net start ignition`. Now trying to connect to the OPC UA server from Unity with Prespective results in a proper connection. In the console some information messages are printed, which can be seen in Figure 4.7. The fifth message indicates that the certificates are accepted. Besides some general information messages about the connection, there are also information messages about tags that are not properly connected and one error message about a FormatException. Tags are elaborated upon in the next section. The error: *FormatException: Input string was not in a correct format.* is solved by changing the *IdentifierTypeOpc* setting in the Gateway settings of the Pre Logic Simulator in Figure 4.5 to "String".

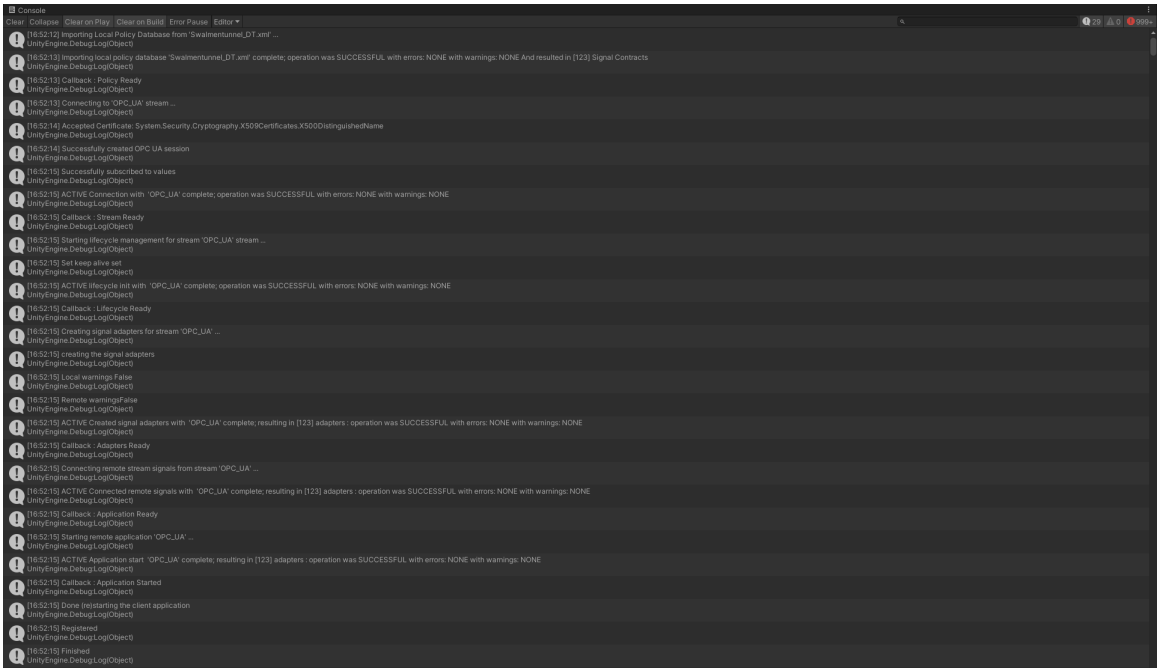

Figure 4.7: Information messages after successful connection

Connection problem with Ignition Designer Launcher

After finally getting a connection working between Unity and the OPC UA server on the Engineering PC it is noticed that the Ignition Designer Launcher on the Interface PC or Plant PC is not making a connection anymore with the OPC UA server, preventing it from opening. After an investigation, the cause of the problem is found to be the firewall of the Engineering PC blocking the outgoing signals. This problem is solved by going to the firewall of the Engineering PC, by typing "Firewall" in the search box of the Taskbar and going to the outbound rules `Advanced settings > Outbound Rules`. Adding port 8088: `New Rule > Port > TCP > Specific local ports > 8088`. Selecting *Allow the connection* and apply to all networks. This process is repeated for the Inbound Rules as well, to be sure. This solved the problem and Ignition Designer Launcher is accessible on the 'remote' PCs.

4.3 Connecting tags

After successfully establishing a connection between Unity and the OPC UA server, information messages are displayed in the console letting the user know that the actuators are not linked properly anymore. The reason for this is that the DT is set-up for TwinCAT. TwinCAT makes use of slightly

different settings and path names, the different path names result in Prespective being unable to find the PLC variable connected to a specific variable in Unity that is connected to an actuator or sensor. The next step is to connect the variables from Unity to the PLC variables. Currently a connection is established with the OPC UA server. However, the PLC variables are stored on the OPC DA server. Before it is also mentioned that it is not possible for Prespective to directly connect to the ABB OPC DA server, with this connection being indicated by the black line from Prespective to the ABB OPC DA server in Figure 4.4, also "locally" this connection is not possible as Prespective does not provide the correct adapter for this connection.

OPC-COM Tunneler Module

After contacting the company that created, and set up, the HIL setup for the TU/e, AT-Automation, a suggestion is proposed for connecting Unity to the PLC variables. The suggestion is to use a module, the OPC-COM Tunneller module, this module is installed on the Ignition Gateway to expose the PLC variables from the OPC DA server directly to the outside by means of an OPC UA interface via the OPC UA server. Figure 4.8 visualizes how this would look like in the data flow schema of Figure 4.6. Figure 4.9 shows the path a signal will travel in a setup using the Tunneller module.

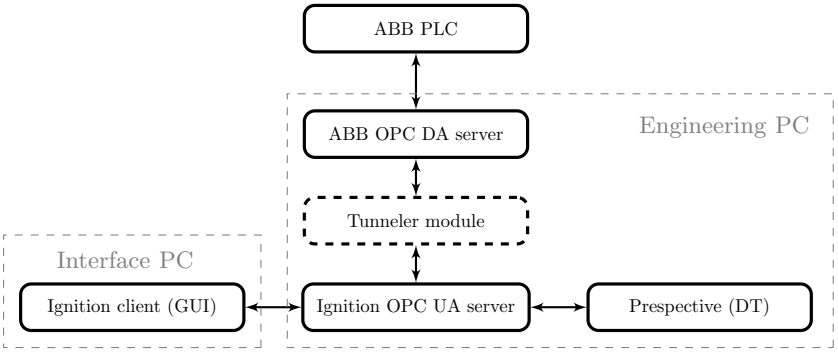

Figure 4.8: Data flow schema with tunneler module

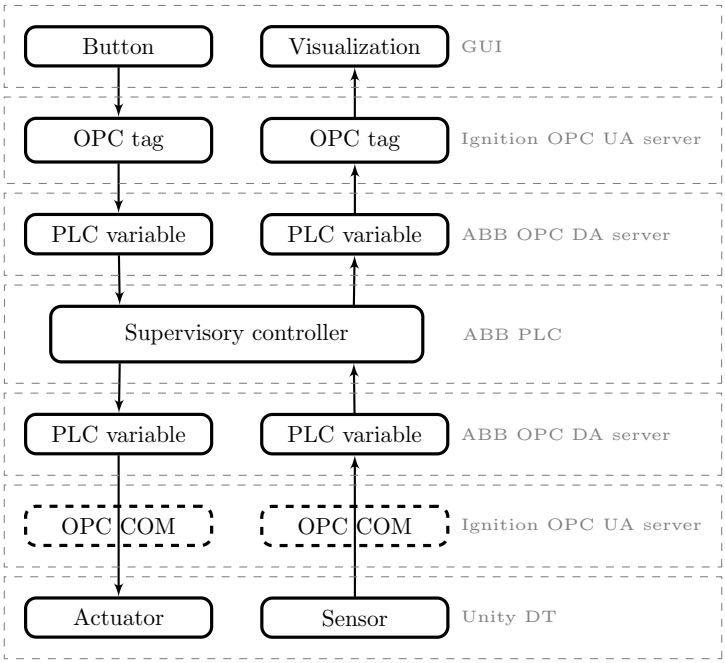

Figure 4.9: Variable linkage with the OPC-COM Tunneller module

The OPC-COM Tunneller module is downloaded from the Inductive Automation website[15]. Scrolling down until *OpcCom Tunneller Module* is found and download it by clicking the hyperlink. Next, the module is added to the Ignition Gateway, this is done by going to `Config > System > Modules`. Scroll down to the bottom and click "Install or Upgrade a Module...". On the next screen select the download file, with the file selector, and press the blue "Install" button. Currently the module is installed but not yet set up, to do this an OPC-COM connection, in the Gateway, from the module to the OPC DA server [32] needs to be established. This is done by setting up an OPC-COM Tunnel Driver connection in `Config > OPC > OPC connections`. On this page click "Create new OPC Connection...". Select "OPC-DA COM Connection", "Local", "OPC DA Server for Control Network" on the following pages. Finally, the settings of the connection are checked and if everything is filled in correctly the connection is created by pressing the blue "Create New OPC Connection" button. To apply the change, the Ignition Gateway is restarted. Restarting the Ignition Gateway is done by opening Command Prompt as an administrator and typing `net stop ignition` in the Command Prompt window, waiting until the messages say that Ignition is stopped, followed by typing `net start ignition`. Now the Tunneller module is visible in the `OPC Quick Client`.

With the Tunneller module visible, it is possible to connect to the OPC DA server with UaExpert, and to change the values for variables. When trying to do the same with Unity, an error is returned indicating a wrong path is used to access the variables. No solutions for this error are found and another suggestion is investigated.

Exposing Tag Providers

After more contact with AT-Automation, and some experimenting, it is decided that exposing the tag providers inside the Ignition Gateway could give some options. This is done in `Config > Security > Show advanced properties > Expose Tag Providers`. By exposing the tag provider it is possible to access the tags from the Ignition Designer. In the Ignition Designer it is possible to create various kinds of tags: memory, expression, OPC and a few more. The OPC tags are directly linked to the PLC variables on the OPC DA server. So by connecting to these tags via the OPC UA server, it is

[15]https://inductiveautomation.com/downloads/ignition/8.1.17

possible for Unity to make a connection to the PLC variables. However, this method does require that all variables that are used in Unity need to be linked to an Ignition (OPC) tag.
Giving an input, via the GUI, changes an Ignition tag value inside the Ignition tag database. The Ignition tag database is located on the Ignition OPC UA server and stores all Ignition tags. When a value in this database changes it is detected by the PLC. Because, the tags are linked to PLC variables. This signal gets sent as an input to the supervisory controller, on the PLC, which returns an output, which can consist out of multiple signals, back to the Ignition tag database. The output signal of the PLC is used to update the value of the Ignition tag in the database. This flow of data is visualized in Figure 4.10.
It is also possible to connect to tags stored on the Ignition tag database that are not linked to any PLC variables, skipping the ABB OPC DA server and PLC entirely, such as memory tags. This flow of data is not used in the DTs of this project and will therefore not be considered. At the same time, this possibility can be useful for other research where this behavior is desired. All steps required for setting up Ignition tags and connecting them in Unity can be found in Appendix A.4.

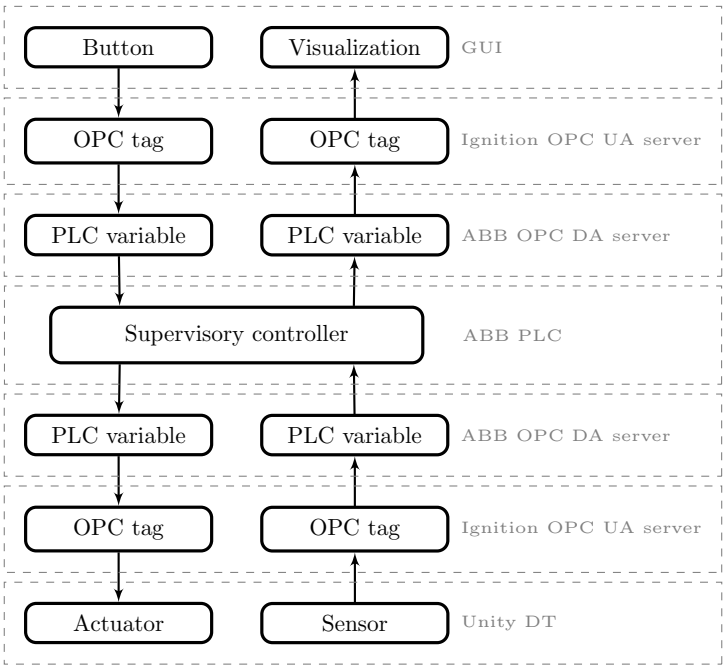

Figure 4.10: Variable linkage

Linking variables

When the variables are set up in Unity, use is made of so called input and output logic scripts, in combination with integer or boolean toggle scripts. An example of an output boolean logic script can be seen in Appendix A.6, as well as a boolean toggle. These scripts read or write the value from or to a tag on the Ignition tag database. In Appendix A.4, items 3 and 4 explain how the scripts are used for setting up the variables in Unity. The logic and toggle scripts are custom made for the Swalmen tunnel DT, more information about them can be read in [20]. The logic and toggle scripts are validated in Section 4.4 of [20].

4.4 Remote connection

At first it was determined that it is not possible to use a remote connection for third party software. This is strange because a connection exists between the Ignition client on a remote PC and the Engineering PC. This can be demonstrated by opening the Ignition Designer Launcher on a remote PC and opening a project stored on the OPC server located on the Engineering PC. A possible

explanation for this could be the use of Ignitions software on both PCs which allows for mutual communication across multiple PCs. Another explanation could be that the connection for the Ignition Designer Launcher takes place over a HTTP connection compared to the TCP connection required for this project. The reason for abandoning the goal of using a remote connection is the fact that UaExpert is not able to make a connection between the remote PC and the Engineering PC. To solve this problem multiple solutions were tried. Because in the end it was not possible to set up a remote connection this idea is shelved and the project continued with a local connection. Eventually a forum entry is discovered that has not been found before during research concerning remote connection. The forum entry, "Connecting to Ignition's OPC UA server" [33], talks about connecting to an Ignition OPC UA server locally, which is already established. The thing however, that is useful, is a response from a lead software engineer from Inductive Automation, the company behind Ignition. In his response he mentions that the default configuration only allows clients on the same machine to connect to the OPC UA server. With this discovery in mind it is tried to change the default configuration so it can be used on remote PCs as well. Before any tests are performed it is noticed that the Ignition OPC UA server on the Plant PC returns a faulted connection. It is still possible to open the Ignition Designer Launcher on the Plant PC successfully. Indicating that a connection is present anyway. Many small experiments are performed leading to the following step-by-step plan for setting up a remote connection:

- It is discovered that the firewall on the Engineering PC interferes with the remote connection. This problem is solved by going to the firewall of the Engineering PC, by typing "Firewall" in the search box of the Taskbar and going to the inbound rules `Advanced settings > Inbound Rules`. Adding port 62541: `New Rule > Port > TCP > Specific local ports > 62541`. Selecting *Allow the connection* and apply to all networks. This is quite strange as one might expect the firewall to block an outgoing signal over that port. A possible explanation is that when trying to set up a connection with UaExpert, on a remote PC, a request is sent to this port on the Engineering PC which is considered as an inbound signal for the Engineering PC.
- On the Engineering PC go to `Config > OPC UA > Server Settings`. Change the *Bind Addresses* to `172.16.0.10`, the ip-adres of the Engineering PC which hosts the OPC server. Change the *Endpoint Addresses* to `172.16.0.10, <localhost>`.
- A new Ignition OPC UA server can be created with a new Endpoint URL or the old Ignition OPC UA server can be adjusted. Go to `Config > OPC Client > OPC Connections`, click the blue "edit" button behind the Ignition OPC UA server. Change the Endpoint URL from `opc.tcp://localhost:62541/discovery` to `opc.tcp://172.16.0.10:62541/discovery`. Continue clicking next to get to the overview page where the settings are saved. The OPC UA server now has a new Endpoint URL which is accessible remotely. Yet, the Gateway first needs to be restarted for the changes to take effect. This can be done by either restarting the Engineering PC or by opening Command Prompt as an administrator and typing `net stop ignition` in the Command Prompt window, waiting until the messages say that Ignition is stopped, followed by `net start ignition`.
- The final step has to do with certificates. This process is very similar to the steps described in Appendix A about certificates. UaExpert is used on the remote PC to create and trust certificates. On the remote PC, a connection with the OPC UA server is established, using the new Endpoint URL, after which the certificates on the client side are trusted. Finally, the certificates of the remote PC need to be trusted on the server side, on the Engineering PC, this is done by going to `Config > opc client > security > server` and trusting the untrusted certificates of UaExpert and Unity.

After applying the settings mentioned above, it is possible to set up a connection from a remote PC, either the Interface PC or the Plant PC, to the Engineering PC with the new Endpoint URL. With this the initial desired data network configuration, seen in Figure 4.3 is obtained. The connection is tested and validated both with UaExpert and Unity. It should be mentioned that the Endpoint URL of the Gateway settings in Unity needs to be changed to the new Endpoint URL to establish a remote connection. Listing 2 shows the updated policy file with the new Endpoint URL.

```
<ControllerSpecification>
  <ActiveController>
    <OPC_UA>
      <Endpoint>opc.tcp://172.16.0.10:62541/discovery</Endpoint>
      <NamespaceIndex>2</NamespaceIndex>
      <IdentifierType>s=</IdentifierType>
    </OPC_UA>
  </ActiveController>
</ControllerSpecification>
```

Listing 2: *Policy.xml*

Arbitrary PC

Now that a remote connection with Unity is proven, the next step is to look at connecting an arbitrary PC, that is not included with the setup present at the TU/e. A benefit of using an arbitrary PC is that the hardware is much better. To connect an arbitrary PC, the same steps as for the RWS PCs apply, using the new Endpoint URL and creating and trusting certificates, plus an additional step. The additional step has to do with the IP address of the arbitrary PC not matching the IP address format of the setup. This problem is easily solved by changing the IP address of the arbitrary PC to the format used by the setup. Only the IP address of the Ethernet adapter needs to be changed, as the communication from the arbitrary PC to the OPC UA server goes over Ethernet. The IP address of the Ethernet adapter is changed by going to `Network & Internet settings > Change adapter options > Ethernet > Properties > Internet Protocol Version 4 (TCP/IPv4)`. In the settings window that is opened, select "Use the following IP address" and fill in "172.16.0.X", where X represents a value in the range 1 to 255, for *IP address*. The value of X cannot be a number already used by the system, at the moment the system occupies most numbers up to 12, e.g. selecting 200 for X will work. After a IP address is selected, the *Subnet mask* is filled in, here the same subnet mask as that of the system is used. Finally, these settings are saved by clicking "OK". The arbitrary PC is now visible for the Engineering PC and vice versa.
Using UaExpert, it is possible to connect to the OPC UA server and to trust certificates, once this is done, it is possible to change the values of Ignition tags stored on the Ignition tag provider. When trying to connect Unity to the OPC UA server the following error is obtained: `ArgumentException: IP Address is invalid`, the full error is shown in Appendix C, Listing 23. This issue is investigated and it is concluded that it has something to do with certificates. This conclusion is backed up by evidence of the OPC UA log file, shown in Listing 3. This log file shows Unity trying to view a certificate but failing, on the RWS PCs Unity is able to find and open this certificate. It is peculiar that UaExpert is able to connect to the Ignition OPC UA server, while Unity is not. It is therefore difficult to pinpoint the origin of the problem. Since the connection between Unity and the Ignition OPC UA server is present for the RWS PCs, the decision is made to let this problem be. A remote connection in which any PC can be used is desirable, but should be further investigated.

```
************************ Logging started at 06/16/2022 13:57:11
6/16/2022 13:57:11.336 Checking application instance certificate.
6/16/2022 13:57:11.340 Creating application instance certificate.
6/16/2022 13:57:11.341 Deleting application instance certificate.
```

Listing 3: Remote HIL test using an arbitrary PC

4.5 Alternative OPC UA adapter

Because Prespective causes many problems, an alternative has been looked for. One such alternative is the OPCUA4UNITY tool by in2sight [34]. This asset works similar to Prespective, it allows a Unity client to connect to the OPC server. One advantage of the OPCUA4UNITY tool is the fact that it allows for certificate accepting within the Unity client, contrary to Prespective. It might also be possible to use credentials with this tool. The OPCUA4UNITY tool has not been tested in

this project. It should be mentioned that switching to an alternative of Prespective might causes problems with the performance of the already realized DTs, as Prespective is not only used as an OPC UA adapter but also for the behavior of the components of the DT, e.g. the movement of the lock doors. This must be taken into account when an alternative to Prespective is considered.

4.6 Conclusion

An overview of the used network and data topology is given, where it is explained how the physical components of the setup are connected and how the data flows between the different components. Then it is explained in steps how a connection between Unity and OPC UA server is established, here it is also explained how the Ignition tags, which are used to send data to and from the DT, are linked. A detailed step-by-step plan is given in Appendix A. With these steps a local connection is established, in Section 4.4 the steps required for establishing a remote connection are detailed. Finally, a recommendation for an alternative to Prespective is given.

5 Configurator

This chapter discusses the need for and the required functionality of the configurator. During the construction of the configurator problems with the Prins Bernhardsluis, referred to as the old DT, are observed. Because the configurator is based upon the old DT, these problems have to be resolved first. This is described in detail in this chapter. Next, the description is provided of what is required to create a configurator in Unity. This is followed up by the presentation of the interface of an example configuration, for this configuration the Prinses Marijkesluis is selected.

5.1 Background

There are many locks in the Netherlands which have great variations between them, but most components from a lock can be generalized and standardized as discussed in Section 3.2. Although there is variation between the locks, there is a finite number of clusters in which the ones are grouped that are similar. The similarity can vary from the water flowing in a singular direction through the lock (mono-directional water retention) to having the same building blocks. In this report, Cluster 5 from [29] is examined as mentioned in 3.5. The locks in Cluster 5 all adhere to the following properties:

- Mono-directional water retention
- Single head high water retention
- Lock chamber width between 10 and 20 meter
- Mitre gates (primarily electromechanical)
- Gate openings (primarily electromechanical)

A description of each property is given in Section 3.5. As mentioned before, DTs can provide many benefits to (existing) infrastructure. However, hardly any lock in the Netherlands has a DT, as it is a time consuming task to create a DT from scratch every time. One possible solution for this is to use a configurator that aids with (quickly) building a configuration of a lock from the general building blocks as described in Section 3.2. This configurator can be modelled and specified for the clusters in [29] which helps in reducing complexity in the creation of a DT. By taking the clusters as a basis for the configurator, a large subset of all locks in the Netherlands can be considered. Besides aiding in the creation of DTs of already existing locks it can also be used to test different configurations of already existing DTs or locks. This can be useful when a new configuration is desired for a lock because the old lock has to be replaced or optimized. Employing a configurator also permits somebody with less (technical) knowledge to create a lock configuration. So the goal of this configurator is to create or adapt DTs of future or existing locks in a shorter time span compared to doing it manually. The configurator is detailed for Cluster 5 only in this report but the same method and techniques can be applied to the other Clusters to extend the configurator to include more or all Clusters from [29].
The idea of using a configurator for this kind of application is rather novel, due to this there is very limited literature available on this subject. In spite of that, this configurator can (to a certain degree) be compared to a car configurator. Just like in a car configurator one can select from a predefined, mostly limited, set of options. However, compared to the car configurator one can, additionally, decide upon the dimensions and placement of certain parts like the gates, as well as the number desired.

5.2 OLD DT

Since the DT created in the project reported on in [17] forms the basis of the configurator it is important that it functions properly. To this end, the functionality of the DT from [17] is evaluated with TwinCAT. First, inputs are given to the operator interface in the control room showing that

only the guillotine gate is working properly. To test the other components, TwinCAT is used to manually send signals to the actuators to rule out a problem with the operator interface. The following problems are encountered, where the naming schema of the gates is detailed in Figure 3.3:

- Unity is extremely slow, also without running Unity in Play Mode.
- When running Unity more than 600 warnings are generated mentioning missing scripts.
- The middle gates of the lock do not work, this also includes the paddles.
- The Eastern downstream gate does not move when an input is given while the Western downstream gate moves according to the given input.

To solve these problems the following steps are taken.

Slow performance

The cause for the slow performance in Unity is most likely the large number of cameras used for the Closed-circuit television (CCTV) system. In addition, there are also three audio listeners (components used by Unity to detect audio in the scene) while Unity prefers to have exactly one audio listener in the scene at a time. This problem is solved by turning off all the cameras in the scene except the one used as the "MainCamera". In addition, all the audio listeners except one are removed from the DT. To do this, the following steps are performed, where use is made of the built-in search function in the hierarchy:

- Enter "camera" in the search box indicated by ① in Figure 5.1.
- Start scrolling through the results until a GameObject with an active Camera component, indicated by ② in Figure 5.1 is found.
- Turning this component off by unticking the tick box, the Camera component of the MainCamera GameObject can be kept turned on.

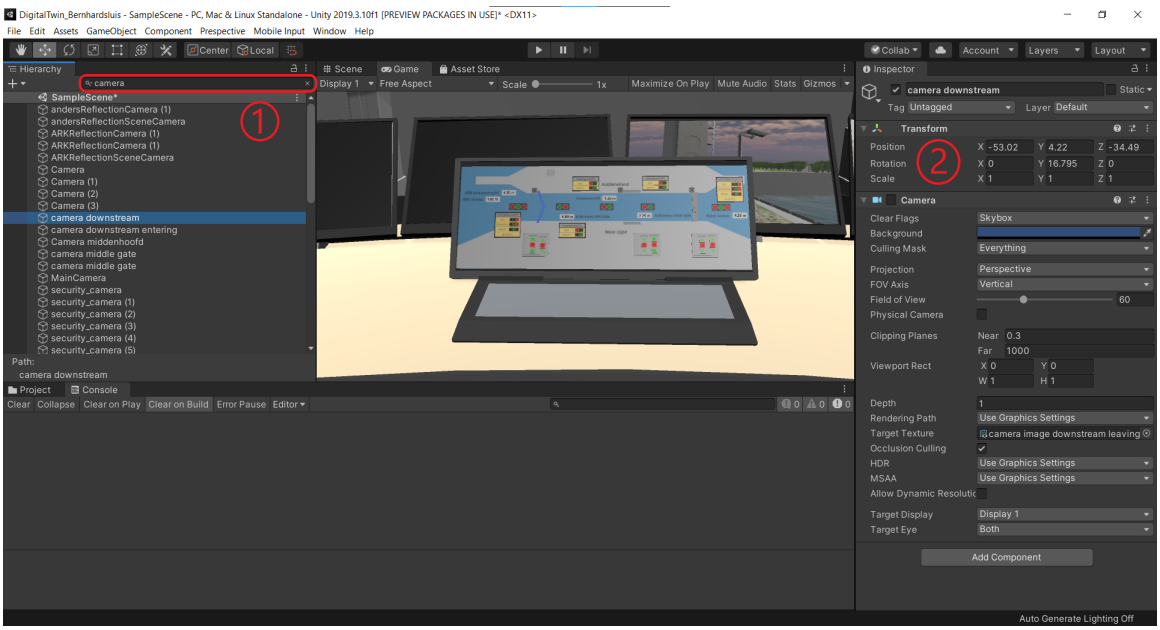

Figure 5.1: Unity editor

This significantly increases the speed of Unity which helps in navigating the editor for solving the remaining problems. The only downside to this solution is that the CCTV functionality is lost. A solution to this problem is provided at the end of this section.

Warnings

When starting the DT in Play Mode exactly 632 warnings are created. There are some slight variations between the warnings but they are all along the same line as the following warning: "The referenced script (Unknown) on this Behaviour is missing!". The reason for this is that the plug-in used for importing the CAD models into Unity, called Pixyz, adds a script to folders and sketches but this script is not included in the files. However, after the CAD models are imported into Unity these scripts are not required anymore. In addition, the CAD models are imported together with their sketches, although these are not visible in Unity. This results in additional empty GameObjects with missing scripts. The warnings are largely solved by removing the GameObjects from the sketches, the remaining warnings are caused by the missing scripts attached to the folders. It is possible to remove these manually, out there is a quicker way of establishing the same result. The GameObjects of the sketches can be removed via the same method that was used for turning off the cameras. In Figure 5.1, use the search query "sketc" at (1), next select all the filtered GameObjects in the hierarchy and delete them. The next step is to remove the remaining missing references. Since these are attached to GameObjects that should not be removed, use is made of a script to find and remove the reference component from the GameObjects. The code is a combination of [35] and [36] and can be seen in Appendix C, Listing 22. This script has to be added to Unity after which it can be executed by going to `Window > Missing Script Window`, see Figure C.1. By pressing the `Search scene` button one gets to see all the GameObjects with missing script references. This tool can be used to see if there are still GameObjects remaining with missing script references. To automatically remove the missing reference components select `GameObject > Remove Missing Scripts`. This can be checked by using the tool mentioned before. When running Unity in Play Mode now, no warnings about missing script references should be generated.

Middle set of gates

After inspection of the middle set of gates it is noticed that the gates can be moved manually inside Unity through the `Control Panel Interface UIE` script attached to the Bar Linkage Solver GameObject. This leads to the conclusion that the gates behave as expected. Yet, when using the interface which can be seen in the center of Figure 5.1, nothing happens. To rule out a wrong connection within Unity, the variables connected to the actuators of the gates are set manually in TwinCAT. These variables cannot be found within TwinCAT as they are missing from the Main program. Listing 21 shows the original list of actuators used by TwinCAT, here the actuators for the middle set of doors are missing, also the ones of the paddles. Since this code was added manually in TwinCAT it is the easiest to also manually add the actuators for the middle set of gates. The result can be seen in Listing 5 which replaces code from Listing 21 in the `MAIN (PRG)` file within TwinCAT.

```
1  GVLS.Actuators[0].oToggle  :=MAIN.state0.dvar_M_M_HW_Downstream_Gate_east_Q_Open;
2  GVLS.Actuators[1].oToggle  :=MAIN.state0.dvar_M_M_HW_Downstream_Gate_east_Q_Close;
3  GVLS.Actuators[2].oToggle  :=MAIN.state0.dvar_M_M_HW_Downstream_Gate_West_Q_Open;
4  GVLS.Actuators[3].oToggle  :=MAIN.state0.dvar_M_M_HW_Downstream_Gate_West_Q_Close;
5  GVLS.Actuators[4].oToggle  :=MAIN.state0.dvar_M_M_HW_Downstream_Paddle_East_Q_Open;
6  GVLS.Actuators[5].oToggle  :=MAIN.state0.dvar_M_M_HW_Downstream_Paddle_East_Q_Close;
7  GVLS.Actuators[6].oToggle  :=MAIN.state0.dvar_M_M_HW_Downstream_Paddle_West_Q_Open;
8  GVLS.Actuators[7].oToggle  :=MAIN.state0.dvar_M_M_HW_Downstream_Paddle_West_Q_Close;
9  GVLS.Actuators[8].oToggle  :=MAIN.state0.dvar_M_M_HW_Upstream_Gate_Q_Open;
10 GVLS.Actuators[9].oToggle  :=MAIN.state0.dvar_M_M_HW_Upstream_Gate_Q_Close;
11 GVLS.Actuators[10].oToggle :=MAIN.state0.dvar_M_M_HW_Upstream_Paddle_Q_Open;
12 GVLS.Actuators[11].oToggle :=MAIN.state0.dvar_M_M_HW_Upstream_Paddle_Q_Close;
```

Listing 4: Original PLC code

```
1  GVLS.Actuators[0].oToggle  :=MAIN.state0.dvar_M_M_HW_Downstream_Gate_east_Q_Open;
2  GVLS.Actuators[1].oToggle  :=MAIN.state0.dvar_M_M_HW_Downstream_Gate_east_Q_Close;
3  GVLS.Actuators[2].oToggle  :=MAIN.state0.dvar_M_M_HW_Downstream_Gate_West_Q_Open;
4  GVLS.Actuators[3].oToggle  :=MAIN.state0.dvar_M_M_HW_Downstream_Gate_West_Q_Close;
5  GVLS.Actuators[4].oToggle  :=MAIN.state0.dvar_M_M_HW_Downstream_Paddle_East_Q_Open;
6  GVLS.Actuators[5].oToggle  :=MAIN.state0.dvar_M_M_HW_Downstream_Paddle_East_Q_Close;
```

```
GVLS.Actuators[6].oToggle  :=MAIN.state0.dvar_M_M_HW_Downstream_Paddle_West_Q_Open;
GVLS.Actuators[7].oToggle  :=MAIN.state0.dvar_M_M_HW_Downstream_Paddle_West_Q_Close;
GVLS.Actuators[8].oToggle  :=MAIN.state0.dvar_M_M_HW_Middle_Gate_east_Q_Open;
GVLS.Actuators[9].oToggle  :=MAIN.state0.dvar_M_M_HW_Middle_Gate_east_Q_Close;
GVLS.Actuators[10].oToggle :=MAIN.state0.dvar_M_M_HW_Middle_Gate_West_Q_Open;
GVLS.Actuators[11].oToggle :=MAIN.state0.dvar_M_M_HW_Middle_Gate_West_Q_Close;
GVLS.Actuators[12].oToggle :=MAIN.state0.dvar_M_M_HW_Middle_Paddle_East_Q_Open ;
GVLS.Actuators[13].oToggle :=MAIN.state0.dvar_M_M_HW_Middle_Paddle_East_Q_Close;
GVLS.Actuators[14].oToggle :=MAIN.state0.dvar_M_M_HW_Middle_Paddle_West_Q_Open;
GVLS.Actuators[15].oToggle :=MAIN.state0.dvar_M_M_HW_Middle_Paddle_West_Q_Close;
GVLS.Actuators[16].oToggle :=MAIN.state0.dvar_M_M_HW_Upstream_Gate_Q_Open;
GVLS.Actuators[17].oToggle :=MAIN.state0.dvar_M_M_HW_Upstream_Gate_Q_Close;
GVLS.Actuators[18].oToggle :=MAIN.state0.dvar_M_M_HW_Upstream_Paddle_Q_Open;
GVLS.Actuators[19].oToggle :=MAIN.state0.dvar_M_M_HW_Upstream_Paddle_Q_Close;
```

Listing 5: Adjusted PLC code

By adding the additional actuators the array of actuators grows from 12 to 20, which has to be taken into account. This is done by changing the value 11 to 19 on line eight in `GVLs > GVLS`, the change can be seen in Listing 6 and Listing 7.

```
Actuators : ARRAY [0..11] OF ACTUATORS; //Automatically Exported Collection of POU's
    of type 'Actuators'
```

Listing 6: Original array actuators

```
Actuators : ARRAY [0..19] OF ACTUATORS; //Automatically Exported Collection of POU's
    of type 'Actuators'
```

Listing 7: Adjusted array actuators

When testing now nothing happens since the actuator variables from TwinCAT were never connected to Unity. To do this a link has to be made between the variable inside Unity and the variable in TwinCAT. To do this, first a variable needs to be created within Unity. This can be done by going to the correct folder in the hierarchy `PreLogicSimulator > Actuators`, adding an empty GameObject and giving it a clear name. The name of the GameObject does not matter in this case as the PLC code in TwinCAT uses indexing instead of the name to identify the actuator. For this step to work properly, make sure that the order of actuators in Unity matches the order of actuators in TwinCAT, in this case the order in Listing 5. The next step would be to add the `Closed_State_logic` script to each of the created GameObjects. In the editor, a few parameters need to be changed. The "Sim Address Path Format" and "PLC Address Path Format" need to be changed to {{*INST_NAME*}}.{{*IO_NAME*}} and {{*INST_NAME*}}.{{*IO_NAME*}} respectively. Also the "Instance Name Rule" needs to be changed into the following: *GVLs.Actuators*[{{*INDEX_IN_PARENT*}}]. The last step is to connect the Gate State to the script. These can be found under the "Bar Linkage solver" folder within each prefab. This should be done for all the missing gate and paddle variables. Figure 5.2 shows what the hierarchy should look like when everything is done properly.

Figure 5.2: PreLogicSimulator Actuators

At this point, TwinCAT can send signals to the DT but it does not receive any signals back. To send signals back the sensors need to be connected. To do this the same steps can be applied as those that are used for the actuators. However, this time different scripts and settings need to be used depending on the function of the sensor. In this case the easiest and most reliable method is to copy-paste the sensors of the downstream gates and paddles, changing their names and linking them to the correct script of the correct object. The downstream sensors can serve as a reference for this. Once these steps are performed for all the gates and paddles of the middle gates, it must not be forgotten to export the policy inside the `PreLogicSimulator` to include all the (updated) connections to the variables. The middle set of gates behaves now as expected.

Downstream set of gates

Considering that both doors of the middle set of gates are behaving as expected, it is possible to resolve the last problem. To this end the downstream set of gates can be removed and the middle set of gates can be copied and moved to the downstream position. Removing the original downstream set of gates will un-link all the variables connected to this GameObject within Unity. This means that all the actuators and sensors for the gates and paddles need to be linked again via the same process as mentioned before for the middle set of gates. However, in this instance no new GameObjects need to be created as the old ones can be reused. Again, it must not be forgotten to export the policy once done for the changes to be applied properly.

Control room

By removing the old downstream set of gates, also a link with the interface in the control room is broken, which results in some warnings or errors. This can be easily solved by linking the correct GameObjects of the new downstream set of gates to a script within the MainCamera which is part of the control room. The control room is turned off as this part is replaced by a GUI in the HIL setup as well as the fact that the CCTV cameras are turned off for the moment, which further reduces the need for the control room in this project.
Once all the steps above are executed the policy file in the `PreLogicSimulator` can be exported one more time and the DT should behave now as expected.

CCTV

To complete the old DT, the control room needs to be added and made functional again. Besides linking the new downstream set of gates, as mentioned above, the main objective is to fix the CCTV system. The main cause for the bad performance when using the CCTV system is the large number of cameras trying to 'stream' their image to the screen inside the control room all at the same time. There are a few possible solutions to this problem, one being the use of less screens and another where one screen is used but where it is possible to switch between various camera views or by grouping the cameras in batches. The latter option shows to be the best option as it still allows the use of multiple screens within the control room. The only catch is that the screens are updated less frequently and in batches. Still this solution should work as there is no high frame rate required for the CCTV system. More information about batching the cameras and the various scripts used can be found in [20]. Chapter 5 in [20] describes how this method can be applied for a CCTV system of a tunnel.

5.3 Configurator

Now the problems with the old DT are resolved it is possible to start with creating the configurator. The configurator is going to be a tool that is used inside the editor of Unity, the reason for this decision is that it avoids the need to use Unity in `Play Mode`, as `Play Mode` is more processor intensive compared to the editor. This should help with performance, it also allows the use of (other) standard tools of the editor. Finally, it makes saving the configuration easier. When a configuration is created in the editor, it can be saved by saving the project. However, if a configuration is made inside `Play Mode` it will not be saved unless it is explicitly done through code in a script. If the configuration is saved inside `Play Mode` it results in the following problem, what is done with the saved configuration from `Play Mode`? In the enumeration below are a few suggestions for possible solutions:

- The configuration is set up every time `Play Mode` is activated.
- The configured configuration from `Play Mode` is (re-)created inside the editor after `Play Mode` is exited from the saved data.

As can be seen from the solutions above, most rely on the editor anyway. This leads to the decision to create the configurator in the editor of Unity.
Only one cluster needs to be implemented in the configurator for this project but the configurator needs to have the possibility to be extended by more clusters. To accomplish this, tabs are used where each tab represents a cluster. The layout of the configurator should guide the user in the right direction, to accomplish this a minimalistic design is selected. The layout is split in two sections, one section where modules can be selected, like doors and traffic lights and the other section is used for environment components, like train tracks and bridges. The environment components can be used to mimic the environment of an existing lock in a configuration, making a configuration easier to recognize. It can also be used to shape the environment of a new configuration. Besides the drop down menus for the selection of the modules and environment components, a slider is implemented to change the dimensions of the selected object. This is used for instance for the doors, because the lock chamber width can range between 10 and 20 meter in Cluster 5. Lastly, a button with the text "Spawn Object" is added. This button adds the preferred object, and scale, in the configuration. This results in the configurator that is depicted in Figure 5.3.

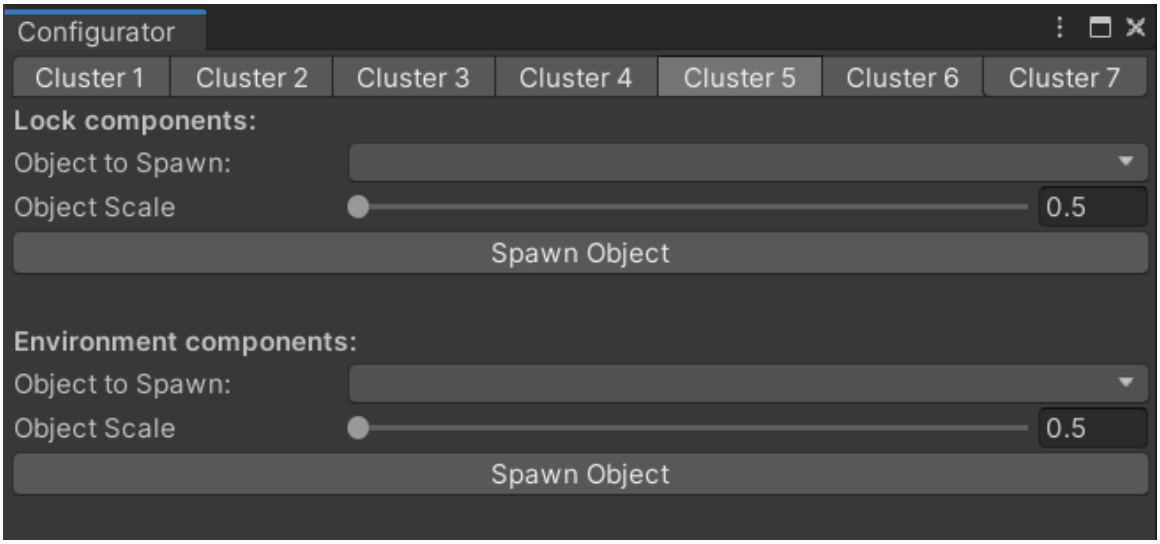

Figure 5.3: Configurator

The script used for the configurator is shown in Appendix B, Listing 10. Currently, once an object is added to the configuration it is placed randomly in the scene and the user needs to position the object in the correct place. The object is added to the main folder of the scene in the hierarchy. When there are a lot of objects in a configuration this can become cluttered and more difficult to debug or find an object. A solution for this is the use of folders, which in practice are empty GameObjects, to divide modules and environment components in an orderly manner. An attempt of creating such a folder structure is implemented in Appendix B, Listing 10 from line 110 onwards. The script checks if a GameObject, that is added to the hierarchy, contains a certain string of characters, e.g. "Gate" or "Traffic light". When this is the case, the GameObject can be moved to a folder, "Gates" for example. At this moment in time, creating the folder structure and moving the GameObjects into the correct folder is not implemented.

Suggestions for follow up

For a follow-up project the following suggestions are made:

- Upgrade to the newest, stable, non-beta version of Unity and Prespective. This will ensure that the configurator can last longer before official support from Unity and Prespective is halted. Making use of newer software can help with removing bugs from both Unity and Prespective. However, there is also a chance of introducing new bugs this way. Due to the encountered problems with this version of Unity and Prespective (2019.3.10f1 and 2020.1.1500.2, respectively), it is recommended to take this risk. Another downside of upgrading to a newer version is that the software used for the creation of the old DT becomes obsolete and requires the DT to be adjusted.
- The prefabs from the old DT that are fixed need to be implemented.
- Scripts from [23] can be used for setting up the connection between Unity GameObjects and Prespective components. Using these scripts will help with quickly setting up new configurations, as the old scripts required a lot of manual work.
- Scripts from [20] can be used for the CCTV system in the control room, since these scripts make use of the mentioned "batching" system that should improve the performance of the DT.
- Besides the method described above for the folder structure, another solution based on tags assigned to specific prefabs can be used. This will require additional initial setup compared to

the earlier method described but will allow GameObjects to be added to certain folders based on the value of the assigned tag.

Lastly, an idea is proposed for the layout of the interface of the configurator to solve the problem of the user having to move the components to the correct location manually. A 2D top-down representation of the configuration can be visualized in the tool where the user can design the configuration, Figure 5.4 shows an illustration of this idea. Here it is possible to change the components, orientation and dimensions of the lock. Once the user is satisfied with the configuration, the "Create" button can be pressed and the 2D configuration is created in 3D.

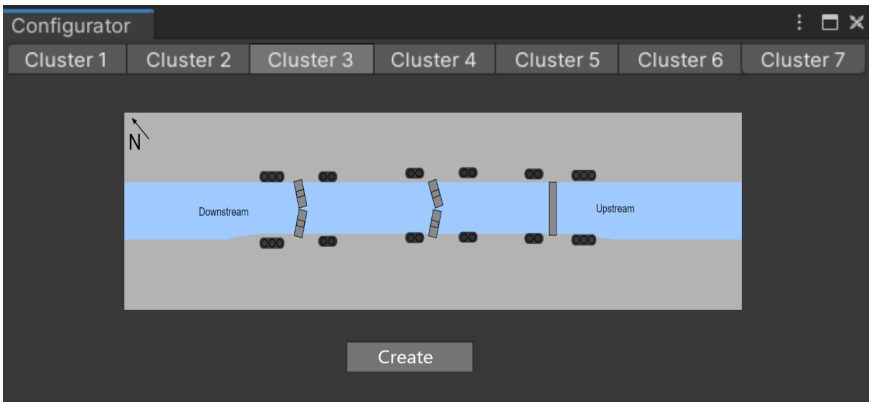

Figure 5.4: Configurator with 2D configuration panel

5.4 GUI

When a configuration is created with the configurator, it can be controlled manually or potentially by the control room when using TwinCAT. The control room is used as a digital counterpart of the real life control room in which operators operate the locks, in the control room only operations that are allowed by the controller are executed. Manual mode allows for testing of the DT by changing the values of actuators and sensors that are normally not possible due to the controller preventing that action from being executed. For the HIL setup it is preferred to use a GUI, or otherwise referred to as operator interface, on a separate laptop instead of making use of the control room inside the DT. Making use of a GUI allows the operator interface to be separate from the plant, this is also how it is done in practice. This also gives the possibility to replace the DT by another DT while keeping the same operator interface. As mentioned in Section 3.5, the reason for choosing Cluster 5 is because there is some material available for the Prinses Marijke lock complex. As can be seen in [37], Figure 13. After looking around with multiple people the files for an Ignition Designer Launcher project of a GUI could not be found. It turns out that the Human Machine Interface (HMI), similar to a GUI, from Figure 13 in [37] is a svg used for the simulation model in Eclipse to test the performance of the supervisory controller. This means that a GUI has to be created from scratch in Ignition Designer Launcher. The GUI has to be made in this program as it is directly integrated in the Ignition software suite of the HIL setup. This makes it easier for actions in the GUI to be connected to PLC variables since Ignition Designer Launcher can directly access the Ignition tag database. The (functional) components of the GUI are made as templates. This makes it easier to create GUI configurations to accommodate the new matching DT configuration made with the configurator.

Prinses Marijke lock complex

For the operator interface the Prinses Marijke lock complex is considered. It is located in the Amsterdam-Rhine canal near Ravenswaaij in Gelderland. The lock complex consists of two water lock chambers, a fixed bridge for road traffic and a storm surge barrier. In this project, only the two

water locks are considered. The object was built in 1939 and put into use after WWII, in 1952. The storm surge barrier is realized in 1979 [38]. The two locks can be found on rows 75 and 78 in Cluster 5 of Figure 3.8. Under normal operating conditions the water locks and the storm surge barrier are open and ships can pass through them without any effort. When the water level reaches 5,55 meters above NAP the storm surge barrier is closed and water traffic has to go through the water locks. In Figure 5.5 the Prinses Marijke lock complex can be seen with the storm surge barrier on the right.

Figure 5.5: Prinses Marijke lock complex [38]

For the creation of the GUI the HMI of the Prinses Marijke lock complex from [37] is used as reference material. This HMI can also be seen in Figure 5.6. Also the simulation model from Github is used for finding the correct variables [18]. The GUI is made in accordance with the style guide rules from RWS for the control station of nautical objects which can be found in [39]. It is designed to fit on a 22 inch Samsung monitor (Samsung LF22T450FQUXEN).

It should be noted that upstream is located on the right in Figure 5.6 and hence downstream on the left. Under normal operating conditions the water levels on both sides are equal. In Section 3.2, it is mentioned that mitre gates point in the upstream direction. Normally this is the case as the water level on the upstream side is higher compared to the downstream position. However, the Prinses Marijke lock complex is an exception to this rule as the downstream side becomes the side with the higher water level when there is a storm in the North Sea, causing the water level to rise from the downstream direction. For this reason the mitre gates are pointing in the downstream direction as they are only operational when the water level downstream is higher than the water level upstream. This scenario is also depicted in Figure 5.6.

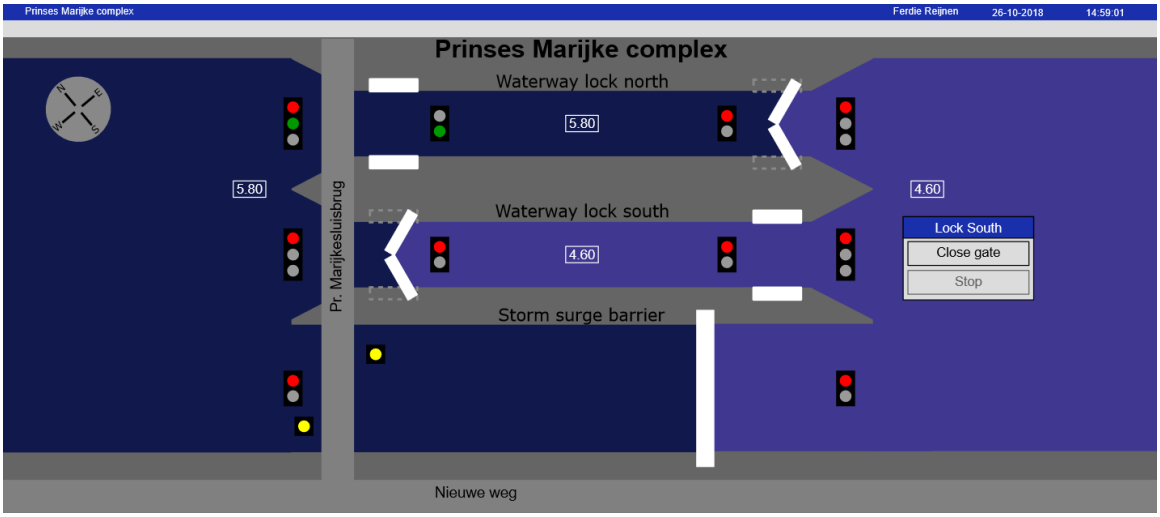

Figure 5.6: HMI Prinses Marijke lock complex [37]

Ignition Designer

A new project is created in Ignition Designer to create the GUI. Ignition has two visualization modules: Vision and Perspective, not to be confused with Prespective, which is the tool used to connect Unity to a OPC UA server as described in Chapter 4. Vision has been around for years and is in extensive use in just about every industry. Perspective is Ignition's newest visualization module. Figure 5.7 gives a schematic overview of the devices for which of each module is used. The GUI for this project is created in the *Vision* module, because this is the module that is used in the CST group of the TU/e. It is important to note that there are some differences in the workspace between the two modules. This means that the settings shown in Appendix B can vary when using Perspective. In addition, attention should be paid when looking for information regarding documentation online (always mention the used module).

Figure 5.7: Vision versus Perspective [40]

The vision module can be found in the `Project Browser`. This opens the Vision Designer Workspace, which is depicted in Figure 5.8.

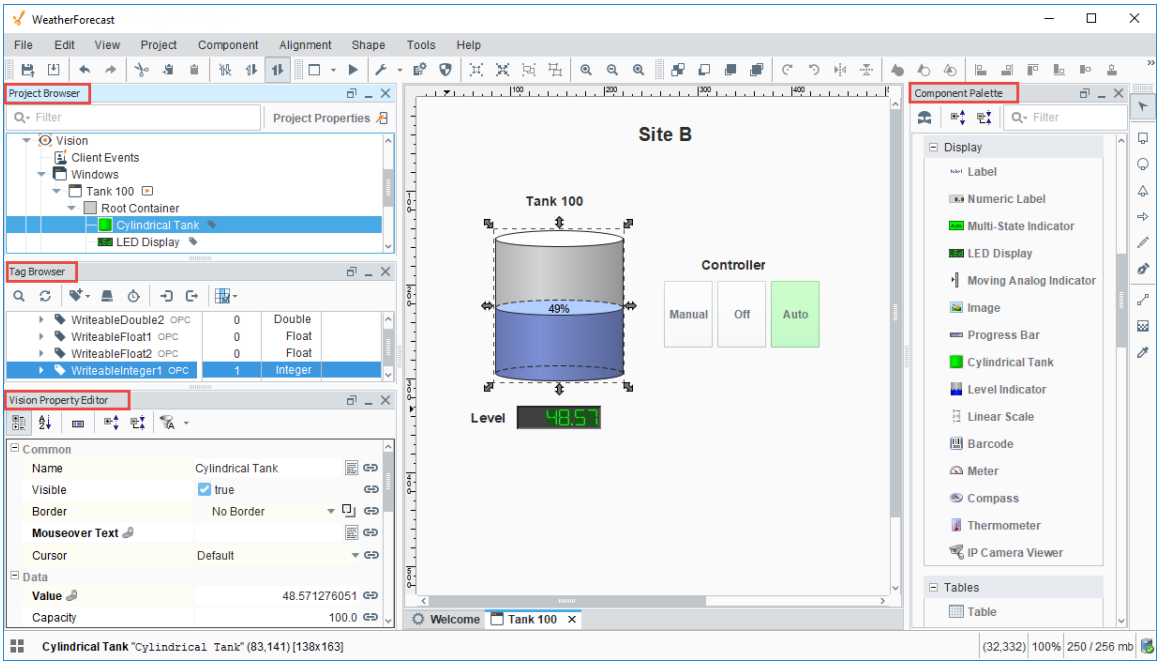

Figure 5.8: Vision Designer Workspace [41]

In the `Project Browser`, a new window can be added called "Main". This is the canvas on which the GUI is made. The following step is to add rectangles and triangles to represent the shape of the lock complex. These shapes can be found in the far-right toolbar seen in Figure 5.8. With these shapes the road, bridge, quays and water from Figure 5.6 can be modelled, according to the style guide rules from RWS.
Text can be added with labels. The font and rotation of the text can be changed in the `Vision Property Editor`. The compass rose seen in the top left corner in Figure 5.6 is imported as a template from another project. Lastly, a rectangle is added that represents the storm surge barrier. With this, all the non-functional (static) objects from the GUI are created.
The goal of the operator interface is to exchange data from the GUI to the PLC and vice versa. To accomplish this, use is made of Ignition tags. There is a variety of Ignition tag types, one being an OPC tag, this tag is linked to a PLC variable. A memory tag stores a value of a certain data type. To properly assign the values of all the used PLC variables, the same number of OPC tags is created as the number of PLC variables used. For this, the folder structure of Figure 5.9 is used. Each OPC tag can be added manually as discussed in Appendix A.4. However, this takes a long time because each time the database has to be queried which takes some time. An alternative method is to export the folder structure from Figure 5.9, which contains only one OPC tag, as a JSON file. After opening the JSON file, the JSON code representing the OPC tag is duplicated as many times, in each folder, as required. The only thing that has to be changed is the *dataType*, *opcItemPath* and *name*. The *dataType* should be the same as the data type of the PLC variable. The reason for exporting the folder structure with at least one configured OPC tag is because all PLC variables are stored in the same location. This means that the *opcItemPath* is the same for each tag and can be easily copied in this manner. Only the name of the PLC variable is different. An example of this is Applications.Application_1.Diagram1.state0.**S_Up_G_E_Act_state**, in which only the bold part needs to be replaced for each tag. It should be noted that the tag names cannot exceed 32 characters due to a PLC limit. The *name* parameter of the tag can be anything but the name is taken of the PLC variable to which it is coupled, in this project. The final JSON file is shown in Appendix B. This file is imported in Ignition Designer which results in a folder with the structure from Figure 5.9 being added that is filled with Ignition tags. Currently the process of adding all the tags in the JSON file is done manually but this can be done automatically by making use of a

generator. A generator is going to be useful when a new configuration is made with the configurator that makes use of different PLC code. For example, the PLC code is given as input to the generator and it outputs a JSON file with Ignition tags.

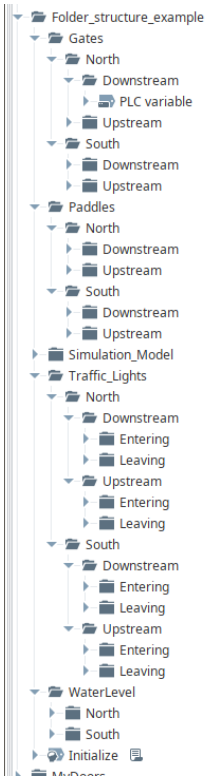

Figure 5.9: Folder structure Ignition tag database

Besides the OPC tags also self-created variables are used. These are used for controlling the movement of the mitre gates, keeping track of water level heights inside the locks and simulating storms. The scripts used are found in Appendix B.
The traffic lights are made as a template. This is done by creating a template under *Templates* in the `Project Browser`. The outline of the entering traffic light is made with a rectangle and three circles. To make the template as easy in use as possible, use is made of "Template Parameters" and "Internal Properties". These can be added by selecting the template and `Component > Customizers > Custom Properties`. This gives the custom properties window, Figure B.6, in which four template parameters are added, one for the actuator and three for the sensors. Additionally, an internal property is added called *Mode*.

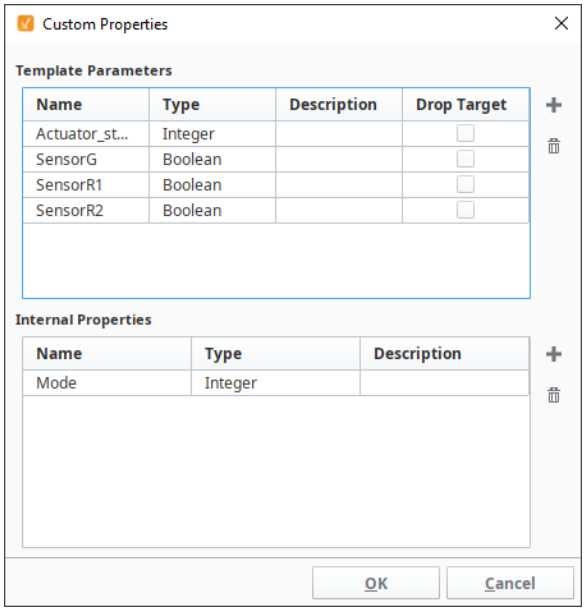

Figure 5.10: Custom properties window

These parameters and property form the variables of the template. The mode variable is directly linked to the actuator state value. The actuator state value is an integer which represents the state of an actuator. Only the last digit of this value is of interest for this template. To do this, an expression is added to the mode variable which takes modules 10 of the actuator state. With this mode variable, the different states of the traffic light can be represented. Only the *Template Parameters* are accessible when using the template in the main window Figure B.2. These variables are used to link to the corresponding tags. By making use of variables, it becomes easy to connect all the tags in a short amount of time. Otherwise, one would have to go through each component of the traffic light to connect the tag to the corresponding property. The last step is to link the state of the traffic light to the mode variable. By doing this, the color of the ”lamp” shows the correct state of the traffic light. The leaving traffic lights can be made in the same manner as the entering traffic lights. Only this time with one ”lamp” less.

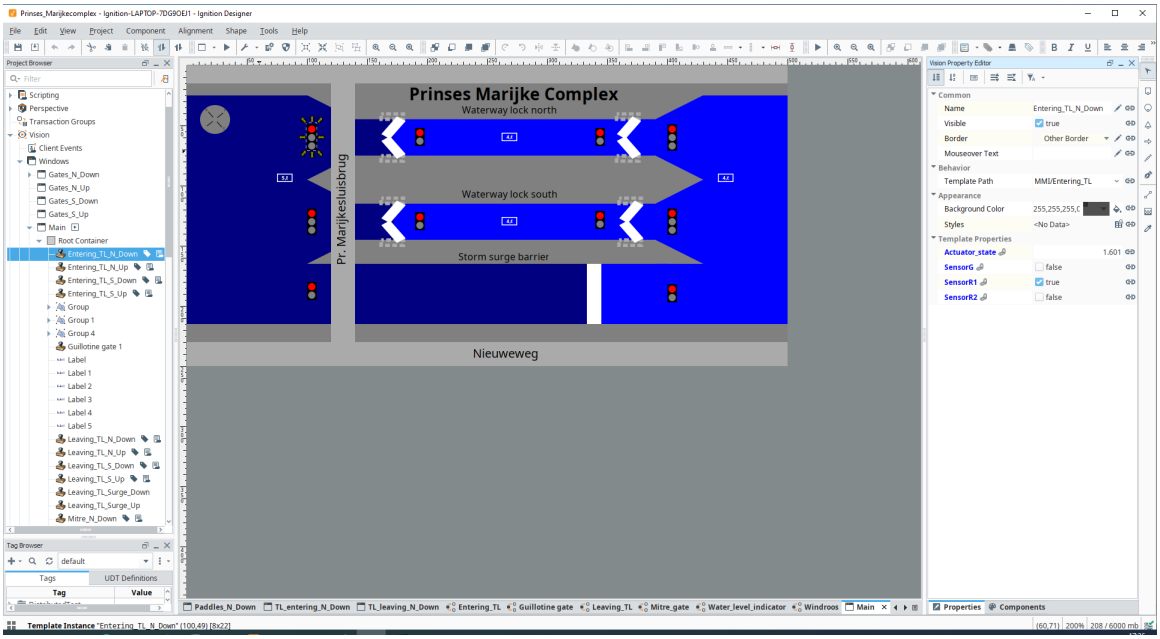

Figure 5.11: Linked entering traffic light template

The water level indicators are also made as a template. It is made by drawing a rectangle with the fill color set to transparent. This only leaves the stroke line which creates a border. Inside this border a label is added that is connected to a *Template Parameter* that should be connected to a tag that represents the water level.
The mitre gates are made as a template, as well. Four rectangles, with transparent fill, are added, of which two have dashed lines. These represent the "open" position of the doors. The other two rectangles represent the doors. For the rectangles to rotate around the pivot point corresponding to the one used in [39] the anchor point needs to be moved, from the center of the rectangle to the corner of the rectangle. Just like the other templates, `Template Parameters` are used. The following variables are used for the gates: the angle of the gate (this is a continuous angle), the position of the gate (this is a discrete angle used for the visualization according to RWS style guide rules) and two variables for sensors for open and closed.
The next step is to create pop-up menus that are activated by clicking on a traffic light or gate in the GUI. To make the pop-up menus, windows are added to the "Windows" folder in the `Project Browser`. These form the canvases of the pop-up menus. A label and a certain number of buttons, depending on the application of the pop-up menu (leaving traffic light: 2; gates, paddles: 3; entering traffic light: 4), is added to each pop-up window. Each specific pop-up menu (gates, paddles, entering and leaving traffic lights) needs to be duplicated four times. Since the pop-up menus are the same for each gate it should be possible to create a template of them. It is possible to create a template from these windows by dragging the window from the *Windows* folder to the *Templates* folder inside the `Project Browser`. However, when trying to use the templates by dragging one into the *Windows* folder nothing happens. This is not a big problem as each button of a pop-up menu needs to be linked directly to an Ignition tag which prevents the use of any variables. Lastly, the pop-up menus are linked to a module in the main window, like the traffic lights or the traffic lights. When clicking on a module, the pop-up menus appear that correspond with that module, for example, the paddles and gates pop-up menus when clicking on a gate.
The last step is to connect all the previously created tags to the corresponding template parameters of the different modules. This can be done by dragging and dropping the correct tag from the `Tag Browser` to the corresponding template parameter. Also the buttons of the pop-up menus need to be connected. The system can be tested by going to `Tools > Launch Project > Extended window` and filling in the RWS credentials. It should be noted that the controller can prevent some actions from

being executed, like opening both gates at the same time when there is a difference between the upstream and downstream water level.
When performing an operation in real life, it can take a few seconds or even minutes for the tasks to be executed entirely, like opening the gates. In the GUI, it is possible to speed up these processes by a certain (variable) factor. To visualize the time that has passed, a label is added to the GUI. This label is linked to a memory tag called *SimulationTime*. This tag is of the data type "Double", it is linked to a Unity variable in the same manner as the other tags. When `Play Mode` is activated the time should increase, starting from zero. When a speed factor different from one is selected, the speed of the executions in the DT should be adjusted accordingly, the same applies to the time step of the timer in the GUI.
A PLC makes use of cycles. It first reads all PLC values, makes calculations and gives an output. This process repeats many times per second. In the time that the PLC is making calculations, it does not read any PLC values, meaning that if a PLC value is changed while the PLC is making a calculation it is not going to detect this until it is done with making the calculations and producing an output. This becomes more pronounced when a large speed factor is selected, because more changes can occur during the time that the PLC is busy making calculations. It is even possible that while the PLC is busy, a PLC value is changed and some time later it is returned back to its previous value before the PLC is done making the previous calculations. This can cause problems. It should be noted that the cycle time of a PLC is in the order of milliseconds while changes in state often happen in the order of seconds in real life. Solving the problem of changing PLC values while the PLC is busy making calculations is outside the scope of this project. The final end result of the Prinses Marijke lock complex GUI is shown in Figure 5.12.

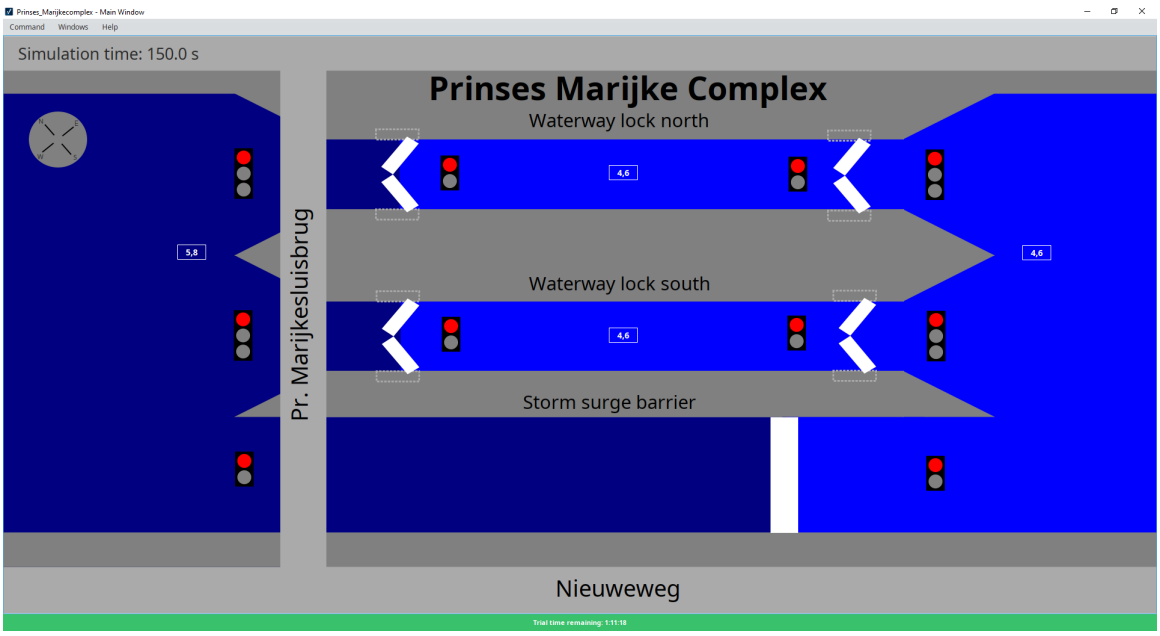

Figure 5.12: GUI of the Prinses Marijke lock complex

Simplifications

During the creation of the GUI some simplifications are made, which are:

- Dimensions - the GUI has some dimensions that are slightly off, according to the RWS style guide rules, due to the useable screen area being less than the 22 inches the GUI is designed for.
- Storm surge barrier - the storm surge barrier has no tags or actions, also the accompanying traffic lights do not work.

- Water level indicator - the value of the indicator jumps between different levels, this can be made such that the value changes gradually.
- The tags used for increasing or decreasing the downstream water level are not yet connected to a button in a simulation model.
- Messages that are shown during certain processes, such as "levelling running" are not incorporated into the GUI as of yet. The same applies for windows, buttons or text that need to change color depending on the state of the GUI.
- There is no information about the angle of the doors in the two positions beside the open position. These positions have been estimated with the use of the simulation model and the RWS style guide rules.
- The text is in English, based upon the simulation model. If the GUI is going to be used for operator training the text should be changed to Dutch.
- Sensors - currently the GUI writes the states of the sensors back to the PLC. When a DT is used, the DT should write back the sensor states to the PLC instead.

5.5 Conclusion

This chapter discusses the need for and the required functionality of the configurator by giving some background information. With this background information a start is made on the configurator. During the construction of the configurator problems with the components of the old DT are discovered. Because the configurator is based upon these components, the problems observed need to be fixed first, how these problems are solved is described in this chapter. Once this step is completed, progress on the configurator continued. An interface is created for the configurator which allows basic configurations to be made, it is possible to scale components but the sizes of the components are not equalized, an example of this could be a traffic light that is larger than a lock door (for the same scaling factor). In addition, no logic is implemented in the configurator that controls the behavior of the components.

Parallel to the construction of the configurator work is performed on the construction of a GUI of an example configuration, that of the Prinses Marijke lock complex. This GUI will be used in HIL simulations to send signals to the configuration DT, made with the configurator. The GUI is made as modular as possible, allowing it to be easily adjusted to match the configuration. The GUI is operational but some simplifications are made along the way, the GUI can be improved by fixing these simplifications. Currently the GUI writes the values of the sensors back to the PLC depending on the input. When the DT is used, the DT should write back the values of the sensors. Tasks that are performed manually at the moment when creating a new GUI configuration, like adding the OPC tags, should be automated through the use of a generator. This generator would take PLC code as input and output a JSON file with Ignition tags.

6 Conclusion and recommendations

In this chapter, the research questions from Section 1.2 are answered. This is done by drawing conclusions from the findings in this report and evaluating them. Afterwards, a recommendation is given for future work.

6.1 Conclusion

DTs have several advantages over 2D visualizations. For example, a DT can be used for operator training. To make operator training as realistic as possible, it is desirable to use the DT in a setup with the final control hardware that is or will be used in real life. For this, it is necessary that it can be included in a HIL simulation. To achieve this, the first research question is formulated:

1. What is needed to connect a DT to a PLC in a HIL setup?

To connect a DT, made in Unity, to a PLC in a HIL setup, an OPC UA adapter is required. This adapter can be supplied by several companies, but in this project Prespective is chosen, because the DT is also made with software from this company. Due to a problem with sending credentials from Prespective to the OPC UA server, it is necessary to remove the credentials from the OPC UA server and allow *Anonymous Access*. Next, the `Gateway Settings` must be adjusted in Unity, for which the settings of the OPC UA server are used. The last step is to create and trust certificates. Because Prespective is not yet able to do this, the program UaExpert is used. This program creates a certificate for the PC and trusts a certificate from the OPC UA server. It is also important that the certificate of the PC on the OPC UA server is trusted. With these steps it is possible to connect a DT locally to a PLC in a HIL setup. For a remote connection, where the DT runs on a separate laptop compared to the OPC UA server, the Endpoint URL must be modified and the port over which the signal is sent and received must be opened in the firewall. With this the first research question is investigated and answered. It should however be noted that the security of the OPC UA server is compromised due to removing the credentials. At the moment, it is possible to use the DT locally or remote on one of the laptops included with the HIL setup. It is not yet possible to use the DT on another, more powerful PC in the HIL setup. When using an arbitrary PC, better hardware can be used when running the DT.

Now that it is possible to use a DT in a HIL simulation, the next step is to look at the DT itself. Currently there is only a DT for the Prins Bernhardsluis, but because of the advantages of a DT it is desirable to also make DTs for other locks. Making a DT takes a lot of work and time, but there is a solution. Many components of locks are similar, therefore it is possible to group locks into clusters. It would be nice if it would be possible to create a configurator based on these clusters. This gives rise to the second research question:

2. What should be done to make the current DT of the Prins Bernhardsluis a suitable configurator basis for a family of water locks

For the second research question, research is performed and the following points are identified as being necessary for the adaptation of the DT of the Prins Bernhardsluis to make it configurable for a family of water locks:

- Developing a GUI that can be used for HIL simulations.
- Making adjustments to the old DT to make it functional.
- Developing an interface for the configurator to create DTs.
- Updating Unity and Prespective to a newer version.
- Enable the sensors of the lock to detect a difference in state.

- Enable the actuators in the DT to influence the components of the lock.
- Enable Unity and the PLC to exchange data.

In this report, the first two steps are worked out and a start is made on the third step. First, a GUI of an example configuration, the Prinses Marijke lock complex, is made in Ignition designer. The GUI is made as modular as possible, so that it can be easily adapted for a different configuration. Some simplifications have been made to the GUI, but the majority of these simplifications has to do with the appearance of the GUI. Second, problems with the old DT that caused certain parts, such as the mitre gates, not to work are solved. The question now is, how much of the old DT can be used, because the old DT is made a lot by hand, which is less than ideal for a configurator where the intention is that all settings are applied automatically through the use of scripts. Third, a start is made with the configurator, for which major steps have already been taken with the interface and placing most prefabs in the configuration.

6.2 Recommendations

The following recommendations are made for a follow-up study. For the HIL connection, other software than Prespective can be looked at, to see if it would allow to send credentials, so they can be used on the OPC UA server again and thus improve security. The remote connection with another, more powerful PC can be worked out, making it possible to use better hardware in the HIL setup. The RWS PCs are not that powerful and can therefore form a bottleneck for larger and more complex DTs that require more memory to run. Therefore, it is also recommended to upgrade the RWS PCs from 8 GB RAM to at least 16 GB. With respect to HIL, it is recommended to automate all labor intensive tasks as much as possible, think of a generator for the Ignition tags, where the PLC code is given as input and the generator outputs a JSON file with Ignition tags.
The GUI can be improved by fixing the simplifications mentioned in Section 5.4.
The configurator can be improved by working on the points mentioned above, under the second research question. If the DTs made with the configurator are going to be used for operator training, it is important that the control room, with a CCTV system, is implemented in the DT. A final suggestion is to add a 2D interface of the lock configuration in the configurator. Currently all components are randomly added to the configuration. If the design is first created in 2D, limiting certain aspects such as the orientation of mitre doors, it is possible to add the entire configuration to Unity at once. This has the advantage that a user no longer has to place all individual parts in the correct position. It will also lower the knowledge threshold, because the 2D interface can influence certain aspects of a configuration. For Cluster 5, the 2D interface can be used to ensure that all mitre gates are in the same direction, so the configuration meets the requirement that the lock must be mono-directional.

References

[1] M. Broy, H. Krcmar, S. Kirstan, and B. Schätz, "What is the benefit of a model-based design of embedded software systems in the car industry?" *Software Design and Development: Concepts, Methodologies, Tools, and Applications*, vol. 1-4, pp. 310–334, 2013.

[2] Wikipedia, "Programmable logic controller," 2022. [Online]. Available: https://en.wikipedia.org/wiki/Programmable_logic_controller

[3] ——, "Model-based design," 2022. [Online]. Available: https://en.wikipedia.org/wiki/Model-based_design

[4] A. Frederiksen, "Model-Based Design of Advanced Motor Control Systems," Tech. Rep., 2013. [Online]. Available: http://www.analog.com/media/en/technical-documentation/technical-articles/Model-Based-Design-of-Advanced-Motor-Control-Systems-MS-2577.pdf

[5] L. Quan and L. Li, "The Study of Soft PLC Running System," *Procedia Engineering*, vol. 15, pp. 1234–1238, 2011.

[6] Wikipedia, "Digital twin," 2022. [Online]. Available: https://en.wikipedia.org/wiki/Digital_twin

[7] E. Negri, L. Fumagalli, and M. Macchi, "A Review of the Roles of Digital Twin in CPS-based Production Systems," *Procedia Manufacturing*, vol. 11, pp. 939–948, 2017.

[8] AI Multiple, "Best Digital Twin Applications & Use Cases in Healthcare in 2022," 2022. [Online]. Available: https://research.aimultiple.com/digital-twin-healthcare/

[9] Deloitte, "Industry 4.0." [Online]. Available: https://www2.deloitte.com/nl/nl/pages/energy-resources-industrials/topics/industry-4-0.html

[10] ——, "Industry 4.0 and the digital twin: Manufacturing meets its match," 2017. [Online]. Available: https://www2.deloitte.com/us/en/insights/focus/industry-4-0/digital-twin-technology-smart-factory.html

[11] i-Scoop, "Digital twins and digital twin technology in an industrial context." [Online]. Available: https://www.i-scoop.eu/internet-of-things-iot/industrial-internet-things-iiot-saving-costs-innovation/digital-twins/

[12] KanooElite, "Using IoT and Digital Twins to Reduce Costs," 2021. [Online]. Available: https://kanooelite.com/using-iot-and-digital-twins-to-reduce-costs/

[13] IEN, "Digital Twins Helping to Save Time, Money," 2018. [Online]. Available: https://www.ien.com/advanced-manufacturing/blog/21008391/digital-twins-helping-to-save-time-money

[14] RockwellAutomation, "Leveraging Digital Twins for Operator Training," 2019. [Online]. Available: https://www.rockwellautomation.com/en-hu/company/events/webinars/leveraging-digital-twins-for-operator-training.html

[15] Rijkswaterstaat, "About us." [Online]. Available: https://www.rijkswaterstaat.nl/en/about-us

[16] J. J. Verbakel, M. E. Vos De Wael, J. M. Van De Mortel-Fronczak, W. J. Fokkink, and J. E. Rooda, "A configurator for supervisory controllers of roadside systems," in *IEEE International Conference on Automation Science and Engineering*, vol. 2021-Augus. IEEE, 2021, pp. 784–791. [Online]. Available: https://ieeexplore.ieee.org/document/9551485/

[17] J. Dijkstra, "Digital Twin of the Prins Bernhardsluis," *BSc Thesis Eindhoven University of Technology*, 2021.

[18] F. F. H. Reijnen, "GitHub - ffhreijnen/PrinsesMarijkeComplex: CIF models for the HIL set-up described in the CCTA 2019 paper," 2019. [Online]. Available: https://github.com/ffhreijnen/PrinsesMarijkeComplex

[19] TU/e, "Digital twins of the festo MPS workstations for the course model-based system engineering on the TU/e." 2022. [Online]. Available: https://cstweb.wtb.tue.nl/4tc00/festo/digital-twin/index.html

[20] J. H. M. van Eijk, "Hardware-in-the-loop simulation using a digital twin," *BSc Thesis Eindhoven University of Technology*, 2022.

[21] M. A. Goorden, L. Moormann, F. F. Reijnen, J. J. Verbakel, D. A. van Beek, A. T. Hofkamp, J. M. van de Mortel-Fronczak, M. A. Reniers, W. J. Fokkink, J. E. Rooda, and L. F. Etman, "The road ahead for supervisor synthesis," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12153 LNCS, 2020, pp. 1–16.

[22] L. Moormann, P. Maessen, and M. A. Goorden, "Design of a tunnel supervisory controller using synthesis-based engineering," *ITA-AITES World*, no. May, 2020. [Online]. Available: https://research.tue.nl/en/publications/design-of-a-tunnel-supervisory-controller-using-synthesis-based-e

[23] J. van Hegelsom, "Development of a 3D Digital Twin of the Swalmen Tunnel in the Rijkswaterstaat Project," *BSc Thesis Eindhoven University of Technology*, 2021.

[24] Wikipedia, "Falkirk wheel," 2022. [Online]. Available: https://en.wikipedia.org/wiki/Falkirk_Wheel

[25] Narrow Boating For Beginners, "Locks, bridges and tunnels." [Online]. Available: http://narrowboatingforbeginners.com/wordpress/locks-bridges-and-tunnels/

[26] Dreamstime, "Ship lock." [Online]. Available: https://www.dreamstime.com/royalty-free-stock-photo-ship-lock-image28637755

[27] R. A. Daniel, "Mitre gates illustration," 2013. [Online]. Available: https://www.researchgate.net/figure/Lock-crown-with-mitre-gates-for-the-new-IJmuiden-sea-lock-Smits-Langedijk-2012_fig8_310751411

[28] Wikipedia, "Guillotine lock," 2022. [Online]. Available: https://en.wikipedia.org/wiki/Guillotine_lock

[29] T. Wilschut, L. F. P. Etman, J. E. Rooda, and J. A. Vogel, "Similarity, Modularity, and Commonality Analysis of Navigation Locks in the Netherlands," *Journal of Infrastructure Systems*, vol. 25, no. 1, 2019. [Online]. Available: https://ascelibrary.org/doi/abs/10.1061/%28ASCE%29IS.1943-555X.0000468

[30] W. Molenaar, "Hydraulic Structures," Delft University of Technology, Tech. Rep., 2011. [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid:36e5bade-8c61-4963-8b70-c83ecd79b863/datastream/OBJ/download

[31] AT-Automation, "Documentatie testopstelling," *Classified document used in the Rijkswaterstaat project*, 2021.

[32] Inductive Automation, "OPC COM Tunneller Module - Ignition User Manual 8.0 - Ignition Documentation." [Online]. Available: https://docs.inductiveautomation.com/display/DOC80/OPC+COM+Tunneller+Module#OPCCOMTunnellerModule-CreateRemoteConnectionsthroughOPC-UAhttps://support.inductiveautomation.com/hc/en-us/articles/360047615731-A-Primer-to-the-OPC-COM-Tunneler-Module

[33] ——, "Connecting to Ignition's OPC UA server - Ignition - Inductive Automation Forum," 2019. [Online]. Available: https://forum.inductiveautomation.com/t/connecting-to-ignitions-opc-ua-server/26137

[34] In2sight, "OPCUA4Unity — Utilities Tools — Unity Asset Store," 2022. [Online]. Available: https://assetstore.unity.com/packages/tools/utilities/opcua4unity-143980

[35] Unity forum, "Remove all missing reference Behaviours - Unity Forum," 2014. [Online]. Available: https://forum.unity.com/threads/remove-all-missing-reference-behaviours.286808/

[36] Pastebin, "Find and remove missing monobehaviour - Pastebin.com," 2017. [Online]. Available: https://pastebin.com/DLuE2Ze9

[37] F. F. Reijnen, J. J. Verbakel, J. M. van de Mortel-Fronczak, and J. E. Rooda, "Hardware-in-the-loop Set-up for Supervisory Controllers with an Application: The Prinses Marijke Complex," *CCTA 2019 - 3rd IEEE Conference on Control Technology and Applications*, pp. 843–850, 2019.

[38] Rijkswaterstaat, "Aanbesteding renovatie Prinses Marijkesluis — Rijkswaterstaat," 2021. [Online]. Available: https://www.rijkswaterstaat.nl/nieuws/archief/2021/03/aanbesteding-renovatie-prinses-marijkesluis-van-start

[39] ——, "Style Guide Bedienplek Nautische Objecten Colofon," pp. 1–119, 2020.

[40] Inductive Automation, "Ignition Vision Module — Real-Time & Historical Data on HMIs." [Online]. Available: https://inductiveautomation.com/ignition/modules/vision

[41] ——, "Designer - Ignition User Manual 8.0 - Ignition Documentation." [Online]. Available: https://docs.inductiveautomation.com/display/DOC80/Designer

A Step-by-step HIL plan

This appendix contains the step-by-step guide that explains how to obtain a connection between the Unity client and the Ignition OPC UA server, as well as an explanation on how to connect the PLC variables to the Unity variables used in the Digital Twin. This appendix has been made in collaboration with J.H.M. van Eijk [20].

A.1 Set up of software and hardware

This section explains what software and hardware are needed before starting the steps.

A.1.1 Unity and Prespective

It is assumed that Unity has been downloaded and installed. The versions of Unity can be found on the website of Unity[16]. Unity version 2019.4.19f1 was used as proof of concept, but any stable newer version should also work. In addition to Unity, the Prespective asset needs to be added and licensed. The Prespective asset can be downloaded and installed from the Unity asset store[17], and the license can be obtained via the Prespective website[18].

A.1.2 Ignition and PLC

It is assumed that the PLC used has PLC code active and is connected to an OPC DA server, which in turn is able to communicate with the Ignition OPC UA server. One way to validate whether this has been done correctly is to see whether an Ignition Designer Launcher can access the variables of the OPC DA server by adding an Ignition OPC tag described in Section A.4. Alternatively, the connections can be checked within the Ignition gateway under `Config > OPC Client> OPC Quick Client`. This should show a similar structure as in Figure A.1, where the OPC DA server is indicated with *ABB OPC Server*.

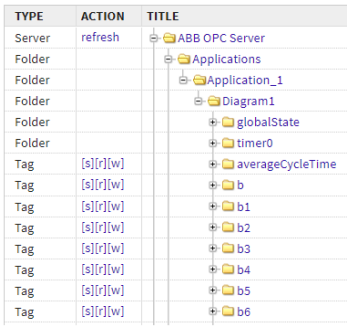

Figure A.1: OPC Quick Client initial view

A.1.3 UaExpert

UaExpert is a program that acts as an OPC UA client. UaExpert is mainly used to accept certificates and to troubleshoot the connection. UaExpert can be downloaded from the download page of Unified Automation[19]. Documentation on how to use UaExpert can be found on the Unified Automation website[20]. The relevant functions of UaExpert are discussed later on.

[16] https://unity3d.com/get-unity/download/archive
[17] https://assetstore.unity.com/packages/tools/utilities/prespective-digital-twin-software-161664#description
[18] https://prespective-software.com/prespective-shop/
[19] https://www.unified-automation.com/downloads/opc-ua-clients.html
[20] https://documentation.unified-automation.com/uaexpert/1.4.2/html/first_steps.html

A.1.4 Ignition Designer Launcher

The Ignition Design Launcher is part of Ignition. This program is used to create OPC tags which are linked to PLC variables. These tags are used for the connection between Unity and the PLC.

A.1.5 ABB Compact Control Builder

ABB Compact Control Builder is a piece of software which is used to upload the PLC code on the PLC. This program is used to test the connection between the PLC and Unity once a connection is set up and the variables of the PLC and Unity are linked.

A.2 Configuring the Ignition OPC UA server

This section gives a step-by-step guide on how to configure the Ignition OPC UA server to allow for local and remote connection for a Unity digital twin. Step 1 is necessary for both local and remote connection, while steps 2-5 are only necessary for remote connection.

1. It is important to remove the need for a username and password for the Ignition OPC UA Server, and to allow for anonymous access. Removing the username and password can be done in the Ignition Gateway under `Config > opc client > opc connections > Ignition OPC UA Server > edit`. Leave the username and password empty, and save the changes. Allowing for anonymous access can be done via `Config > OPC UA > Server Settings > Anonymous Access Allowed` and toggling it to true.

The following steps configure the Ignition OPC UA server for remote connection.

2. Go to `Config > OPC UA > Server Settings`. Change the *Bind Addresses* to the IPv4-adres of the PC which hosts the Ignition OPC UA server, e.g. `172.16.0.10`. Change the *Endpoint Addresses* to: "`<IPv4-adres>, <localhost>`", where `<IPv4-adres>` is the same as the *Bind Addresses*.

3. The endPoint URL of the Ignition OPC UA server needs to be updated. Go to `Config > opc client > opc connections > more > endpoint`. Change the endPoint URL from `opc.tcp://localhost:62541/discovery` to `opc.tcp://<IPv4-adres>:62541/discovery`, where `<IPv4-adres>` is the same IPv4-adres used in step 2. Continue clicking next to get to the overview page where the settings are saved.

4. The firewall of the PC which hosts the Ignition OPC UA server interferes with the remote connection. This problem is solved by going to the firewall of the PC which hosts the Ignition OPC UA server, by typing "Firewall" in the search box of the Taskbar. Next go to `Advanced settings > Inbound Rules`. Here, click `New Rule > Port > TCP > Specific local ports > 62541`. Select *Allow the connection* and apply to all networks.

5. The Ignition Gateway first needs to be restarted for the changes to take effect. This can be done by either restarting the PC or by opening Command Prompt as an administrator and typing `net stop ignition` in the Command Prompt window, waiting until the messages say that Ignition is stopped, followed by `net start ignition`.

A.3 Connection between Unity and OPC UA server

This section gives a step-by-step guide on how to establish a connection with the Ignition OPC UA Server using the Unity client.

1. First, a logic simulator needs to be added to Unity. This can be done by adding an empty GameObject in a Unity project and adding the *Pre Logic Simulator* script to this GameObject.

2. The next steps relate to the Gateway settings tab within the Pre Logic Simulator. See Figure A.2 for a visual indication of the settings within the Pre Logic Simulator.

(a) The *Adapter Target* should be set to OPC_UA via the drop-down menu.

(b) The *XmlFilePath* needs to be named. This will create a file containing the policy of the Adapter Settings. A standard name that can be used for this is *policy.xml*. When another name is used, make sure it has the *.xml* extension.

(c) The *EndPointUrl* should contain the Endpoint URL of the Ignition OPC UA server. This can be found in Ignition under `Config > opc client > opc connections > more > endpoint.`

(d) The *ConfigurationFilePath* contains a path to the Configuration file of Prespective, this should be correct by default.

(e) In the case of connecting to the Ignition OPC UA server, the IdentifierTypeOpc should be set to *String*.

(f) The other three settings can be left alone for now. More information on the Gateway Settings can be found in the Prespective documentation of OPC UA[21].

(g) To save the Gateway settings, the *Export Policy to XML* button needs to be pressed.

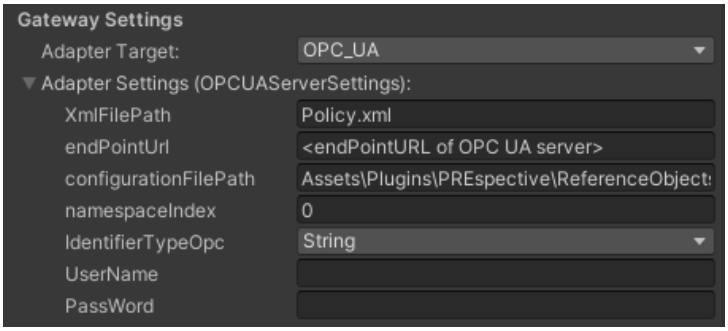

Figure A.2: Gateway Settings in Unity

3. Click play within Unity, the error *ServiceResultException: Error establishing a connection: Error received from remote host: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target* should pop up, since certificates need to be validated to establish the legitimicy of the connection.

4. The next steps relate to using UaExpert.

   (a) Open UaExpert and click the plus button in the task bar (see number 1 in Figure A.3).

   (b) In *Custom Discovery*, double click to add a server. The URL of the server should be the same as the endpoint used for Unity.

   (c) If done correctly, the tree can be extended, and the *Basic256Sha256* security level can be chosen, click *OK*. The server should now be visible under the Servers tab (left-hand side of the window).

   (d) Click on the plug connector in the taskbar to connect to the server (see number 2 in Figure A.3). A pop-up should now show up similar to Figure A.4. On this pop-up click *Trust Server Certificate*, this accepts the certificate of the server on the client side.

[21] https://unit040.atlassian.net/wiki/spaces/PUD/pages/1212979579/2020.1.60.3+OPC+UA

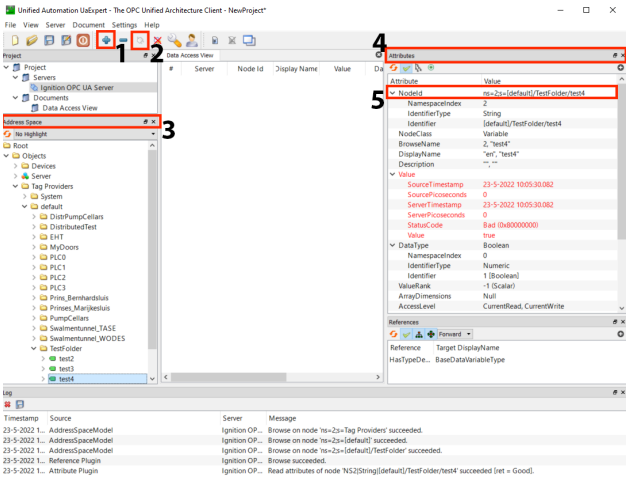

Figure A.3: Overview of UaExpert, with important parts highlighted.

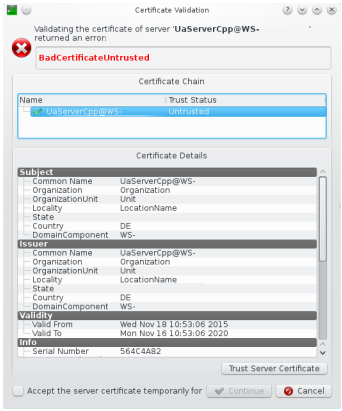

Figure A.4: Certificate validation[22]

5. The certificates on the server side also need to be accepted, to do this, go to `Config > opc client > security > server`. Trust the untrusted certificates of UaExpert and Unity.

6. Click the plug connector in the taskbar of UaExpert once more (see number 2 in Figure A.3), if done correctly, UaExpert should now be able to connect to the Ignition OPC UA server.

7. Click play within Unity, if done correctly, info messages similar to the ones in Figure A.5 should be seen, indicating a successful connection to the OPC UA server of Ignition.

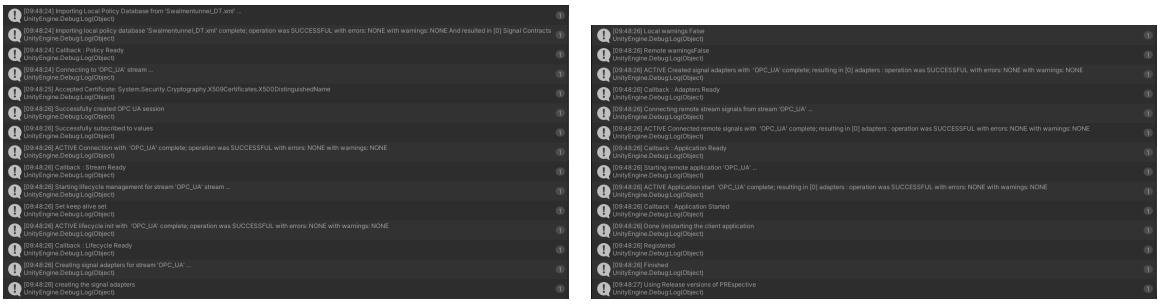

Figure A.5: Succesful connection info messages

A.4 Linking PLC variables to Unity variables

After establishing connection to the OPC UA server, the next step is to read or write variables from the PLC in Unity. The main challenge is that Unity is connected to the OPC UA server of Ignition, while the desired variables are on the OPC DA server of the PLC. To solve this, it is possible to link the Unity variables to OPC UA tags, which in turn are connected with the PLC variables. This process is visualized in Figure A.6. This section describes how to setup the required variables required for data interchange.

[22] https://documentation.unified-automation.com/uaexpert/1.4.2/html/first_steps.html

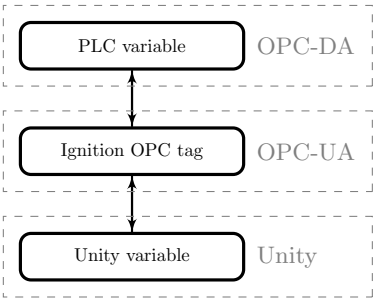

Figure A.6: Variable linkage

1. First, make sure the tag providers are exposed. This can be done in the Ignition gateway via `Config > OPC UA > Server Settings > Show advanced properties > Expose Tag Providers`. If done correctly, the Tag providers folder should be visible in the OPC quick client in the Ignition gateway (`Config > OPC Client > OPC Quck Client`). Alternatively, this can be checked via UaExpert by looking in the Address Space tab (see number 3 in Figure A.3) and opening the *Tag Providers* folder while connected to the Ignition OPC UA server.

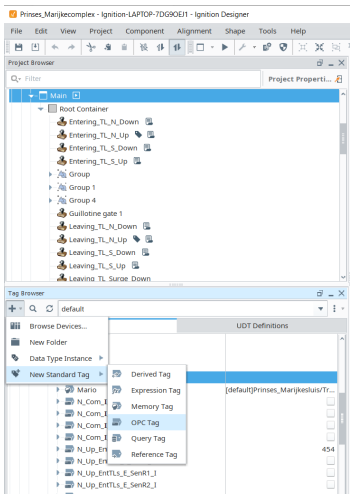

Figure A.7: Design launcher adding tag

2. Add an Ignition OPC tag to the Ignition OPC UA server. This can be done by opening the `Ignition Designer Launcher> Opening an arbitrary project` once inside the project a tag can be added by going to `Tag Browser > Plus sign > New Standard Tag > OPC Tag`, as in Figure A.7. Make sure the target server is set to the ABB OPC server. In the tag path, choose the PLC variable to which the OPC tag should be connected. Verify that the *Data type* of the OPC tag is the same as the date type of the PLC variable. Give the tag a clear name, as this will be needed in Unity. Figure A.8 shows the settings page of a tag. Here the tag can be given a *Name* under `Basic Properties`, the *Data type* can be adjusted under `Value` settings, the *OPC Server* setting needs to be set to "ABB OPC server" and *OPC Item Path* should point to the path of a PLC variable.

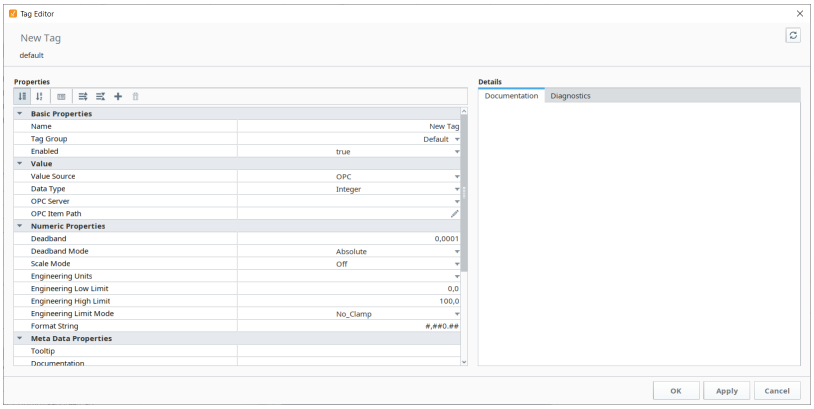

Figure A.8: Design launcher editing tag

3. Within Unity, variables need to be created. These variables are connected to the tags on the OPC UA server which allows the variables in Unity to read and write signals to the PLC. To achieve this, a script needs to be added to an empty GameObject, depending on the type of the variable and whether it needs to read a PLC variable (output) or write a PLC variable (input). Section A.6 provides an example of an Output Boolean variable. Next to this, the corresponding toggle file needs to be added, depending on the type of the variable, an example of a toggle file can also be found in Section A.6. The toggle file contains the variable which can be written or read by other Unity GameObjects. Figure A.9 provides an example of a GameObject with a (bool) toggle script.

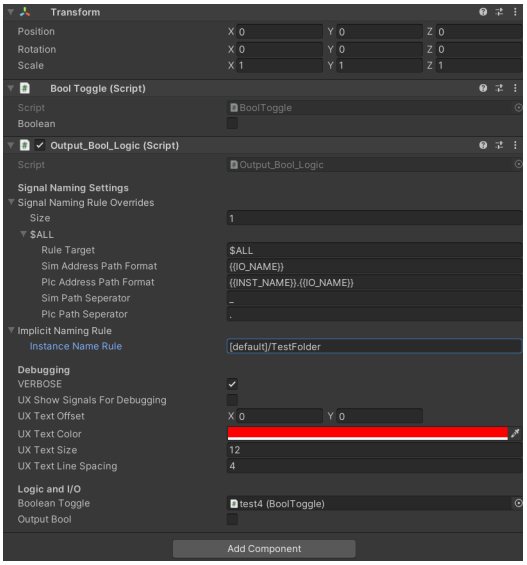

Figure A.9: Example of Output_Bool_Logic script

4. Figure A.9 shows an example of the *Output_Bool_Logic* script. To link an Unity variable to the created Ignition OPC tag, the following steps need to be taken.
   (a) Change the GameObject name to the name of the Ignition OPC tag. By right-clicking on the GameObject in the hierarchy window and selecting `Rename` or by changing the name at the top of the inspector window after selecting the GameObject.
   (b) Change the *Sim Address Path Format* under `Output_Bool_Logic > signal Naming Settings > Signal Naming Rule Overrides > $ALL` to {{*IO_NAME*}}.

(c) Change the *Plc Address Path Format* under `Output_Bool_Logic > signal Naming Settings > Signal Naming Rule Overrides > $ALL` to {{*INST_NAME*}}/{{*IO_NAME*}}.

(d) Change the *Instance Name Rule* under `Output_Bool_Logic > signal Naming Settings > Signal Naming Rule Overrides > Implicit Naming rule` to the Identifier of the OPC tag, minus the tag name and leading forward slash. This can either be found via the Ignition Designer Launcher, by right clicking the tag and clicking *Copy Path*, or via UaExpert, by selecting the tag in the Address Space and copying the Identifier from the Attributes tab (see numbers 3&4 in Figure A.3). Both options will give the following result for the path of tag test4: `[default]/TestFolder/test4`, in Unity the following needs to be filled in for the *Instance Name Rule*: `[default]/TestFolder`.

(e) Additionally, the namespaceIndex setting and the IdentifierTypeOpc need to be changed in the Gateway Settings of the Prelogic Simlator to match the NodeID. This can be found in UaExpert in the Attributes tab of the tag (see numbers 4 and 5 in Figure A.3). These settings should be the same for all OPC tags.

(f) To save these settings, the *Export Policy to XML* button needs to be pressed in the Gateway Settings tab.

5. If done correctly, a change of a writable variable in Unity should result in a change of the corresponding Ignition OPC tag value, and the corresponding PLC value. The Ignition OPC tag can be checked in the Ignition Designer Launcher, and the PLC variable changing can be checked in the ABB Compact Control Builder. The change of a readable variable in Unity should happen after changing the corresponding PLC variable value in the ABB Compact Control Builder.

The PLC address in the xml policy file from Section A.4, which can be opened by clicking "Open Policy File" in the Gateway settings, see Figure A.2, should contain the same information as the Identifier of the OPC tag for each variable. The Unity variable can now be used within Unity for e.g. an IO-script as described in a previous report [23].

A.5 Troubleshooting the connection

It can be difficult to pinpoint the cause of a problem when connecting fails. It is important to note that this appendix section assumes that the connection from Unity to the Ignition OPC UA server is done locally. When trying to connect remotely, these steps will likely not result in a successful connection. A helpful software program that may help troubleshooting is the OPC UA ANSI C Demo Server[23]. This demo server sets up a very simple and bare boned OPC UA server, allowing the user to connect to this server via either UaExpert or via Unity. Remember that connecting via Unity first requires a connection via UaExpert in order to accept the certificates. Using the Demo Server can thus help with identifying whether the problem lies within Unity or the Ignition Gateway. Below are a few errors and problems defined with their solutions.

- **ServiceResultException: Error establishing a connection: Error received from remote host: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target**

  The origin of this error has to do with certificates. To solve this problem the certificates need to be accepted locally as well as on the OPC UA server. See step 4 and 5 of the step-by-step guide of Section A.3.

- **Unable to access the Ignition Design Launcher on a remote laptop**

  This can be solved by opening the port of the Ignition Gateway (8088 by default). To do this, first search for `Firewall & Network protection`. In this window, choose for `Advanced settings` and `Inbound Rules` or `Outbound Rules`. It is recommended to do the following steps for both the

[23] https://www.unified-automation.com/downloads/opc-ua-servers.html

`Inbound Rules` and `Outbound Rules`. Click `New Rule > Port > TCP > Specific local ports`. On this screen the port number can be added (8088 by default). Select `Allow the connection` and apply to all networks.

- **FormatException: Input string was not in a correct format**

  Set the *IdentifierTypeOpc* in the Gateway Settings of the Pre Logic Simulator to *String*. This step can also be found back in step 2e of Section A.3.

- **ServiceResultException: Error establishing a connection: BadNotConnected**

  This error can have multiple sources. One source is that a previous connection has not been terminated properly. The main source is probably caused by trying to use a remote connection which is not yet possible. A solution for this problem is to continue locally. Meaning, using Unity on the same PC that hosts the OPC server.

- **ServiceResultException: Error establishing a connection: BadChannelSecureClosed**

  Most likely a wrong port is used within Unity for the *EndPointUrl*. Check if the correct *EndPointUrl* is used by going to `Config > opc client > opc connections > more > endpoint`.

A.6 Script examples

```
  using   System;
using System.Collections.Generic;
using System.Reflection;
using u040.prespective.prelogic;
using u040.prespective.prelogic.component;
using u040.prespective.prelogic.signal;
using UnityEngine;


public class Output_Bool_Logic: PreLogicComponent
{
#if UNITY_EDITOR || UNITY_EDITOR_BETA
    [HideInInspector] public int toolbarTab;
#endif

    [Header("Logic and I/O")]

        [Tooltip("BooleanToggle component for the in or output")]
        public BoolToggle BooleanToggle;
        [Tooltip("Input Boolean (used for input to the PLC)")]
        public bool OutputBool;

    #region <<PLC Signals>>
    #region <<Signal Definitions>>
    /// <summary>
    /// Declare the IO signals
    /// </summary>
    ///

    public override List<SignalDefinition> SignalDefinitions
    {
        get
        {
            return new List<SignalDefinition>
            {
                new SignalDefinition(gameObject.name, PLCSignalDirection.OUTPUT,
    SupportedSignalType.BOOL, "", "Value", onSignalChanged, null, false),
            };
        }
    }
    #endregion
    #region <<PLC Outputs>>
    /// <summary>
    /// General callback for the IOs
    /// </summary>
    /// <param name="_signal">the signal that has changed</param>
    /// <param name="_newValue">the new value</param>
    /// <param name="_newValueReceived">the time of the value change</param>
    /// <param name="_oldValue">the old value</param>
    /// <param name="_oldValueReceived">the time of the old value change</param>
    void onSignalChanged(SignalInstance _signal, object _newValue, DateTime
    _newValueReceived, object _oldValue, DateTime _oldValueReceived)
    {
        if (_signal.definition.defaultSignalName == gameObject.name)
        {
            OutputBool = (bool)_newValue;
            if (BooleanToggle.Boolean != OutputBool)
            {
                BooleanToggle.Boolean = OutputBool;
            }
        }
        else
        {
            Debug.LogWarning("Unknown Signal received:" + _signal.definition.
    defaultSignalName);
        }
```

```
64      }
65      #endregion
66  }
67  #endregion
```

Listing 8: Unity Script *Output_Bool_Logic.cs*

```
1   using   UnityEngine;
2
3   public class BoolToggle : MonoBehaviour
4   {
5       [Tooltip("Boolean Value of the toggle")]
6       public bool Boolean = false;
7
8       public void Toggle(bool oToggle)
9       {
10          Boolean = oToggle;
11      }
12  }
```

Listing 9: Unity Script *BoolToggle.cs*

B Details configurator and GUI

In this appendix, an overview is given of the code of the configurator. In addition, all scripts and settings used for the creation of the GUI, of the Prinses Marijke lock complex, are detailed in this appendix.

Configurator

Listing 10 shows the script used for the configurator. A thing to note is the word: "EditorWindow" on line 6, which indicates that this script is used in the editor of Unity. Lines 8 till 25 are used to define various variables, like lists of possible components and variables that keep track which component is selected from the drop down menu. Lines 27 till 31 adds the configurator to the editor of Unity. The function `OnGUI()` creates the interface and adds all the components, like tabs, drop down menu's and labels for text. The function `SpawnObject()` adds the selected component to the configuration in the scene of Unity. The remainder of the code from line 110 onwards is part of an experiment to add the components to a specific folder, as mentioned in Chapter 5.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEditor;
5
6  public class Configurator : EditorWindow
7  {
8      // Editor
9      int toolbarIndex = 0;
10     string[] toolbarStrings = {"Cluster 1", "Cluster 2", "Cluster 3", "Cluster 4", "
       Cluster 5", "Cluster 6", "Cluster 7"};
11
12     // Prefabs
13     // string objectBaseName = "";
14     private int objectID = 1;
15     GameObject objectToSpawn;
16     float objectLockScale, objectEnvironmentScale;
17     private float spawnRadius = 50f;
18     public string[] prefabLockOptions = new string[] {"Mitre gate", "Walls", "Traffic
        light entering", "Traffic light leaving"};
19     public int prefabLockIndex = -1;
20     public string[] prefabEnvironmentOptions = new string[] {"Water", "Kade", "Road",
        "Train track", "Traffic light", "Bridge"};
21     public int prefabEnvironmentIndex = -1;
22
23     // Prefabs spawning
24     private string[] spawnLockOptions = new string[] {"mitre gate", "lock_wall", "
       TF_three", "TF_two"};
25     private string[] spawnEnvironmentOptions = new string[] {"Water/Water4/Prefabs/
       Water4Advanced", "Kade", "road", "train_track", "Traffic Light", "brug"};
26
27     [MenuItem("Tools/Lock creator")]
28     public static void ShowWindow()
29     {
30         GetWindow(typeof(Configurator));      //GetWindow is a method inherited from
       the EditorWindow class
31     }
32
33     private void OnGUI()
34     {
35         toolbarIndex = GUILayout.Toolbar(toolbarIndex, toolbarStrings);
36         switch(toolbarIndex)
37         {
38
39             case 3:
40                 GUILayout.Label("Basic Modules", EditorStyles.largeLabel);
41
42                 EditorGUILayout.Popup("Object to Spawn:", -1, prefabLockOptions);
43                 objectLockScale = EditorGUILayout.Slider("Object Scale",
       objectLockScale, 0.5f, 3f);
```

```
44
45                  if (GUILayout.Button("Spawn Object"))
46                  {
47                      // SpawnObject();
48                  }
49                  break;
50
51              case 4:
52                  GUILayout.Label("Lock components:", EditorStyles.boldLabel);
53
54                  prefabLockIndex = EditorGUILayout.Popup("Object to Spawn:",
    prefabLockIndex, prefabLockOptions);
55                  objectLockScale = EditorGUILayout.Slider("Object Scale",
    objectLockScale, 0.5f, 3f) ;
56
57                  if (GUILayout.Button("Spawn Object"))
58                  {
59                      SpawnObject(spawnLockOptions[prefabLockIndex], prefabLockIndex,
    prefabLockOptions[prefabLockIndex]);
60                      Debug.Log(prefabLockIndex);
61                      Debug.Log(prefabLockOptions[prefabLockIndex]);
62                  }
63
64                  GUILayout.Space(20);
65
66                  GUILayout.Label("Environment components:", EditorStyles.boldLabel);
67
68                  prefabEnvironmentIndex = EditorGUILayout.Popup("Object to Spawn:",
    prefabEnvironmentIndex, prefabEnvironmentOptions);
69                  objectEnvironmentScale = EditorGUILayout.Slider("Object Scale",
    objectEnvironmentScale, 0.5f, 3f) ;
70
71                  if (GUILayout.Button("Spawn Object"))
72                  {
73                      SpawnObject(spawnEnvironmentOptions[prefabEnvironmentIndex],
    prefabEnvironmentIndex, prefabEnvironmentOptions[prefabEnvironmentIndex]);
74                      Debug.Log(prefabEnvironmentIndex);
75                      Debug.Log(prefabEnvironmentOptions[prefabEnvironmentIndex]);
76                  }
77                  break; //Used for switch/case
78          }
79
80      }
81
82      private void SpawnObject(string options_prefabs, int index, string name_prefab)
83      {
84          if(index == -1)
85          {
86              Debug.LogError("Error: Please assign an object to be spawned.");
87              return;
88          }
89
90          Vector2 spawnCircle = Random.insideUnitCircle * spawnRadius;
91          Vector3 spawnPos = new Vector3(spawnCircle.x, 0f, spawnCircle.y);
92
93
94          UnityEngine.Object objectToSpawn = Resources.Load("Prefabs/" +
    options_prefabs); // note: not .prefab!
95          Debug.Log(objectToSpawn);
96
97          GameObject newObject = (GameObject)GameObject.Instantiate(objectToSpawn,
    spawnPos, Quaternion.identity);
98
99          newObject.name = name_prefab + " " + objectID;
100         newObject.transform.localScale = Vector3.one * objectLockScale;
101
102         if (options_prefabs == "TF_two")
103         {
```

```
104             newObject.transform.Rotate(-90,0,0);
105         }
106 
107         objectID++;
108     }
109 
110     // private void SpawnObject(options_prefabs, index, name_prefab)
111     // {
112     //     if(prefabLockIndex == -1)
113     //     {
114     //         Debug.LogError("Error: Please assign an object to be spawned.");
115     //         return;
116     //     }
117 
118     //     Vector2 spawnCircle = Random.insideUnitCircle * spawnRadius;
119     //     Vector3 spawnPos = new Vector3(spawnCircle.x, 0f, spawnCircle.y);
120 
121 
122     //     UnityEngine.Object objectToSpawn = Resources.Load("Prefabs/" +
    spawnLockOptions[prefabLockIndex]); // note: not .prefab!
123     //     Debug.Log(objectToSpawn);
124 
125     //     GameObject newObject = (GameObject)GameObject.Instantiate(objectToSpawn,
    spawnPos, Quaternion.identity);
126 
127     //     newObject.name = prefabLockOptions[prefabLockIndex] + " " + objectID;
128     //     newObject.transform.localScale = Vector3.one * objectLockScale;
129 
130     //     if (spawnLockOptions[prefabLockIndex] == "TF_two")
131     //     {
132     //         newObject.transform.Rotate(-90,0,0);
133     //     }
134 
135     //     objectID++;
136     // }
137 
138 
139     // Use this to create a folder structure
140     // Perhaps create all folders the first time you press one of the spawn buttons,
    change bool variable which says the folders are already created
141     // Next place each gameobject in each corresponding folder
142 
143     // private void getInfo()
144     // {
145     //     foreach (GameObject obj in Object.FindObjectsOfType(typeof(GameObject)))
146     //     {
147     //         if (obj.transform.parent == null && obj.name != "Main Camera" && obj.
    name != "Directional Light")
148     //         {
149     //             string checker = "test";
150     //             GameObject objToSpawn;
151     //             if (obj.name.Contains(checker))
152     //             {
153     //                 Debug.Log("Exist");
154 
155     //                 objToSpawn = new GameObject("folder");
156     //                 // Debug.Log(obj.name);
157     //             }
158     //             else
159     //             {
160     //                 Debug.Log("Does not exist");
161     //             }
162     //         }
163     //     }
164     // }
165 }
```

Listing 10: Configurator

GUI

Down below all scripts with the extension ".py" are used on components to update sensors or to control the behavior of the component in the visualization, e.g. changing the angle of the doors. The expressions mentioned, are applied to their corresponding tag in the Ignition tag database. These expressions are used to update the value of the tag.
The figures show the used settings and overall layout of the individual components, this can be used as reference material in a follow-up project. The *DoubleToggle.cs* and *Input Double Logic.cs* scripts are used to update the visualization of the simulation timer in the GUI.

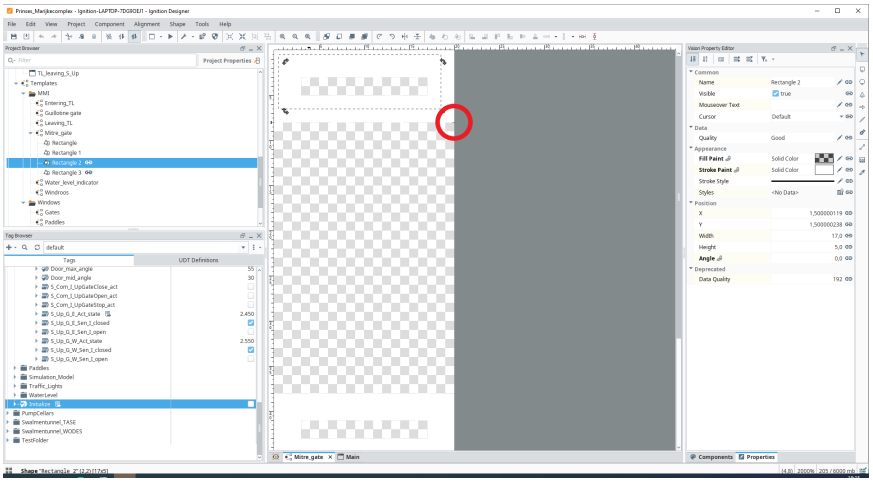

Figure B.1: Anchor positioned in the corner

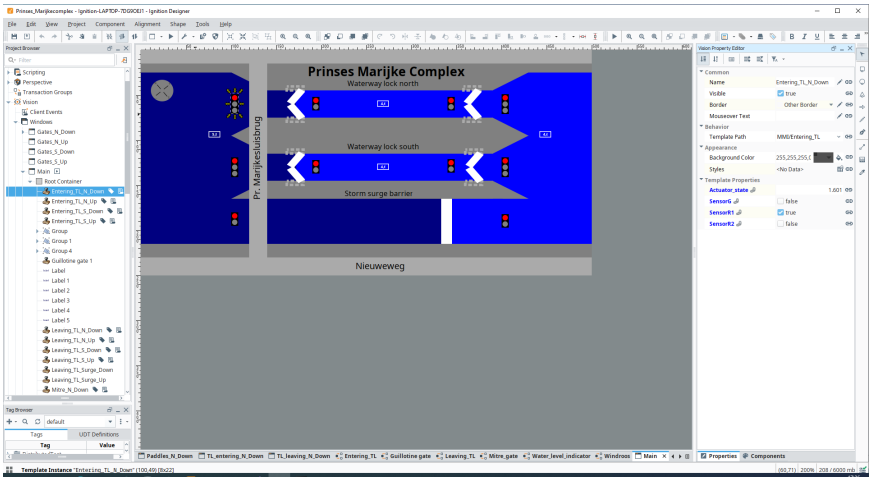

Figure B.2: Connected TL

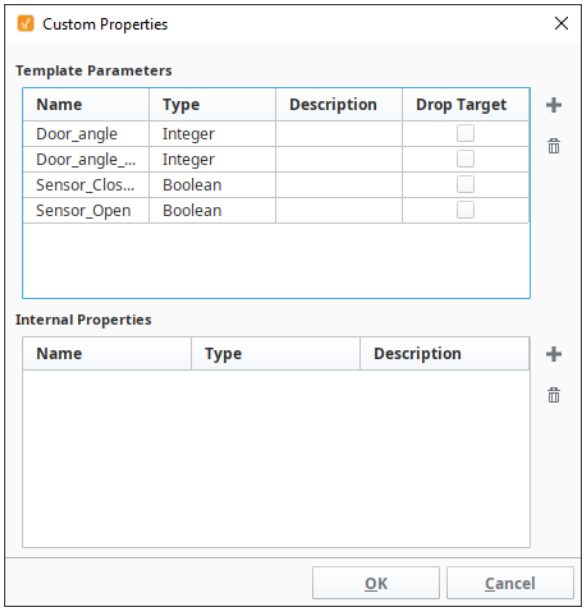

Figure B.3: Custom properties window mitre gates

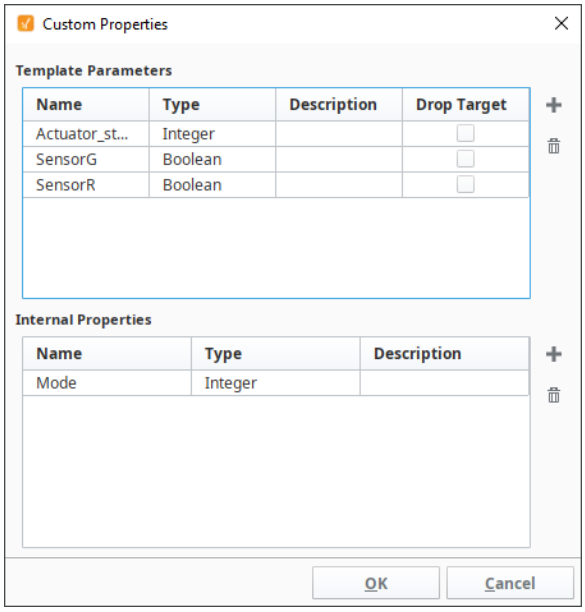

Figure B.4: Custom properties window TL Leaving

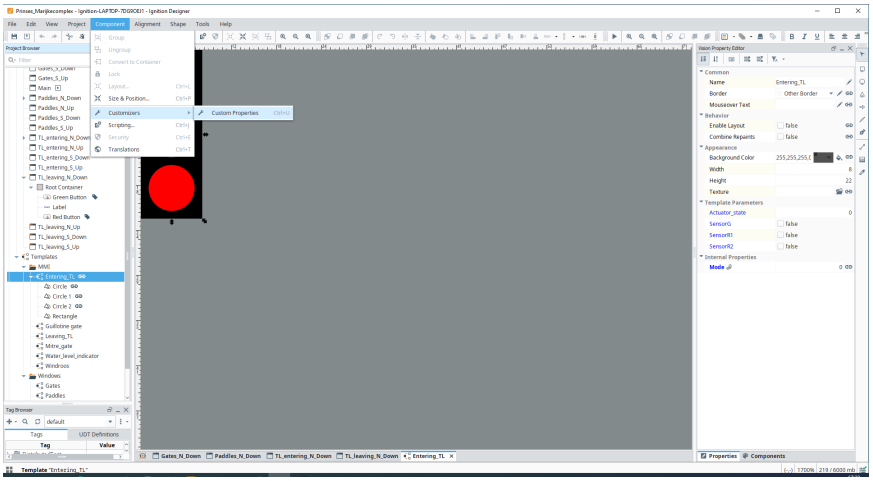

Figure B.5: Custom properties window

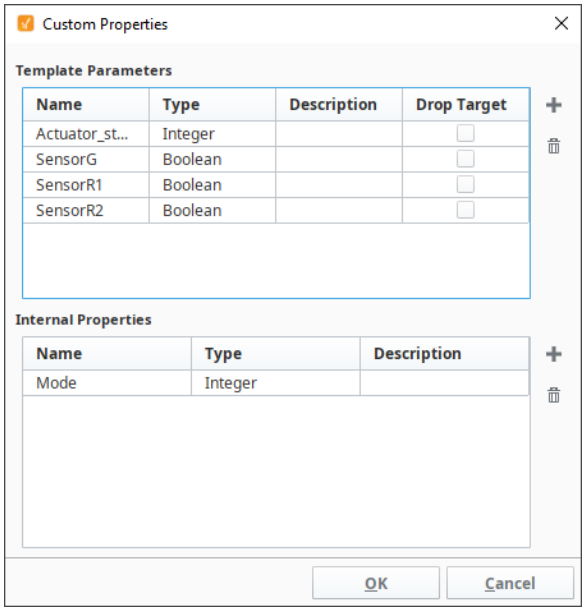

Figure B.6: Custom properties window TL entering

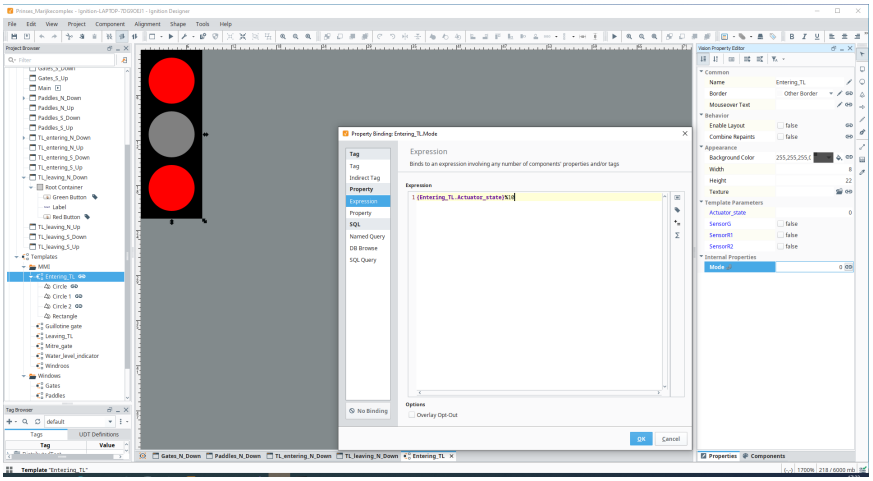

Figure B.7: Mode settings

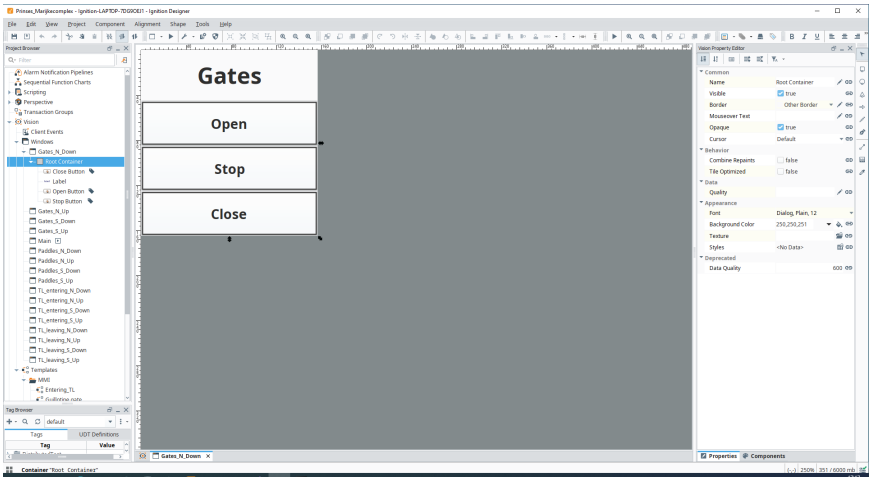

Figure B.8: Popup Window Gates

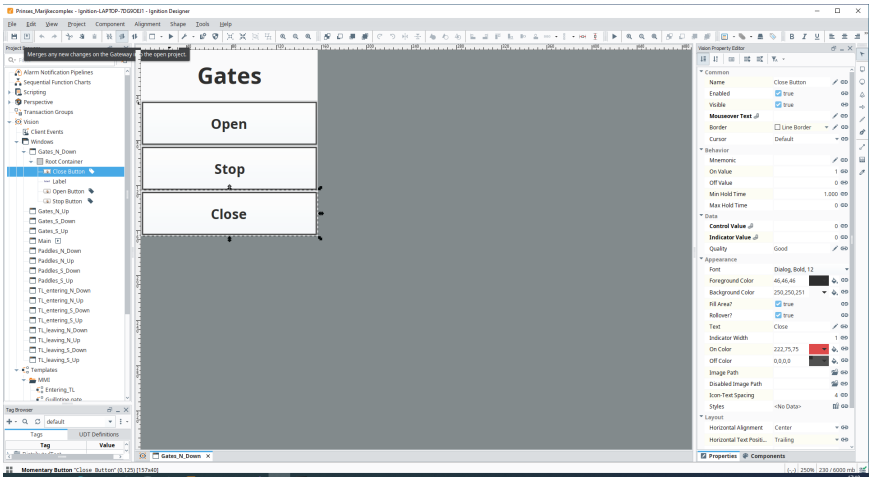

Figure B.9: Popup Window Gates settings button

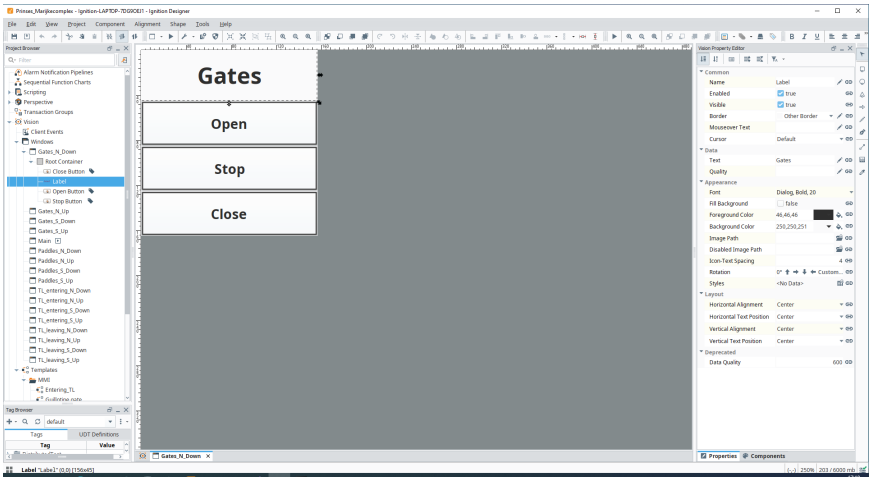

Figure B.10: Popup Window Gates settings label

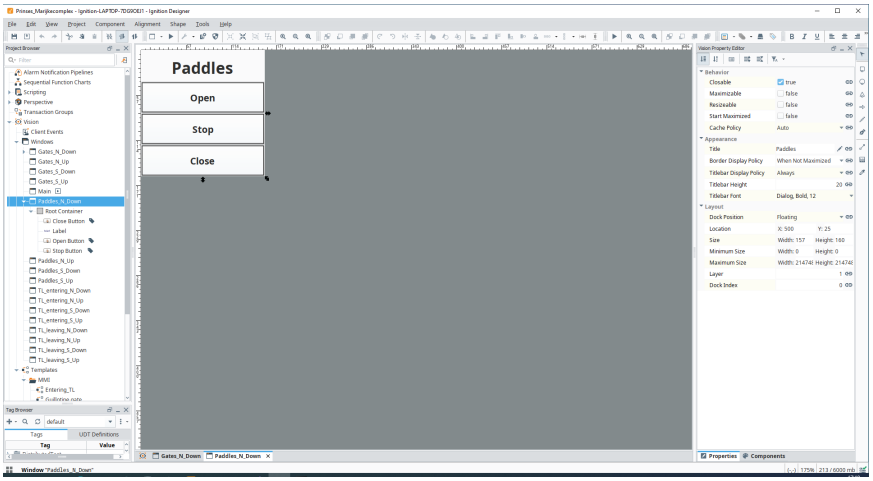

Figure B.11: Popup Window Paddles

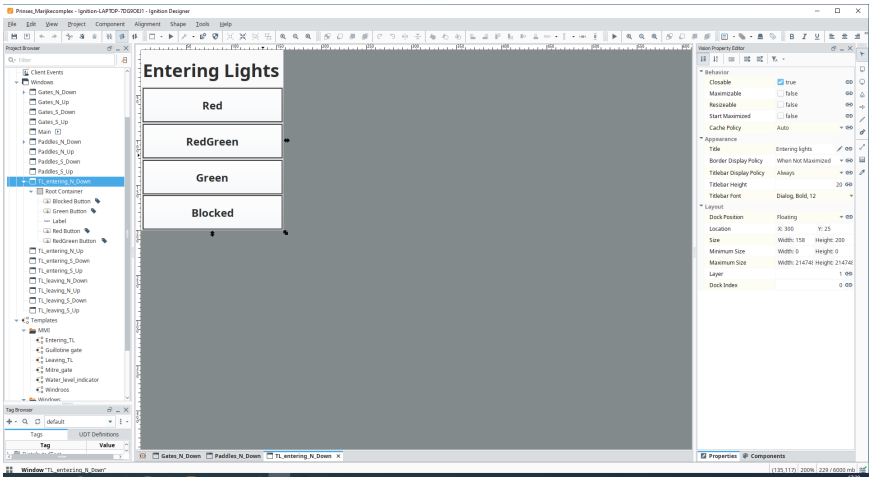

Figure B.12: Popup Window TL Entering

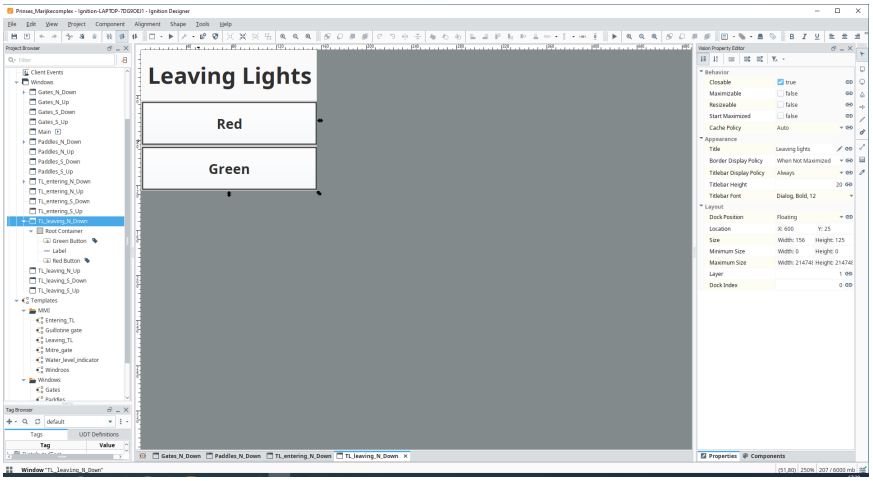

Figure B.13: Popup Window TL Leaving

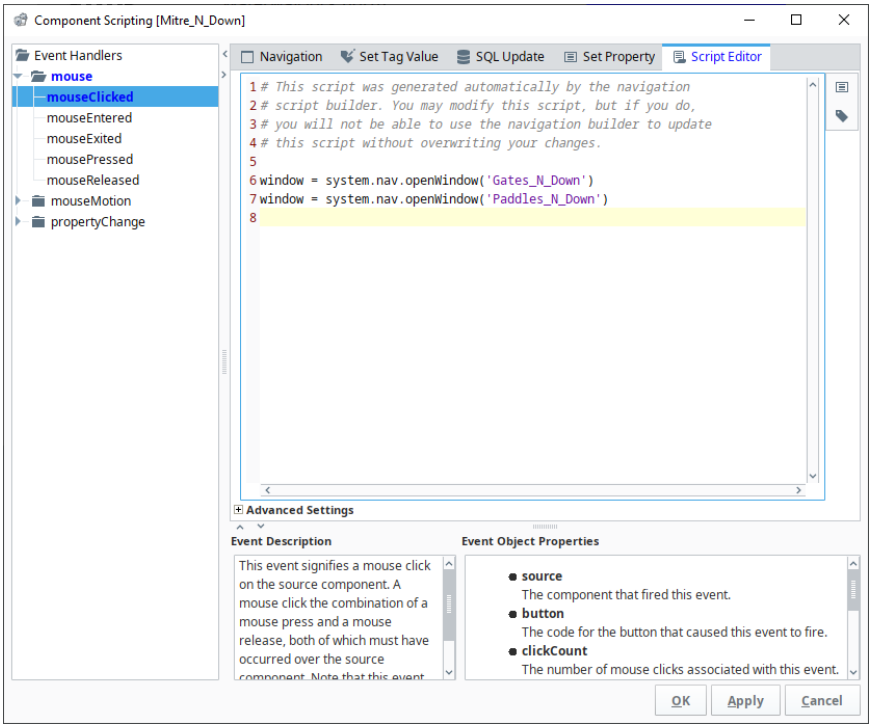

Figure B.14: Scripting on template for popup multiple windows

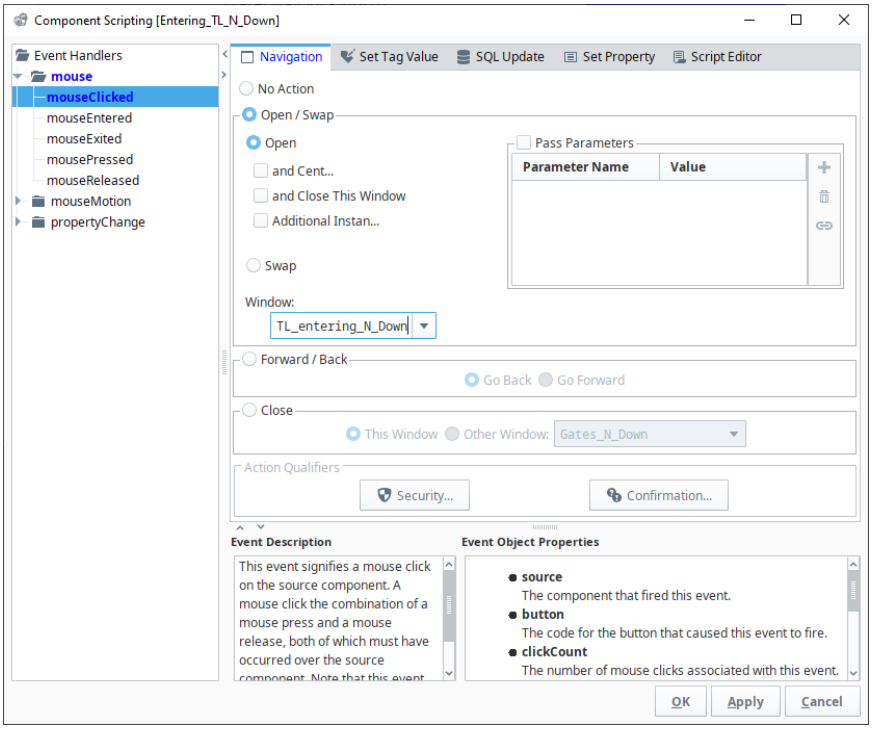

Figure B.15: settings for popup menu display

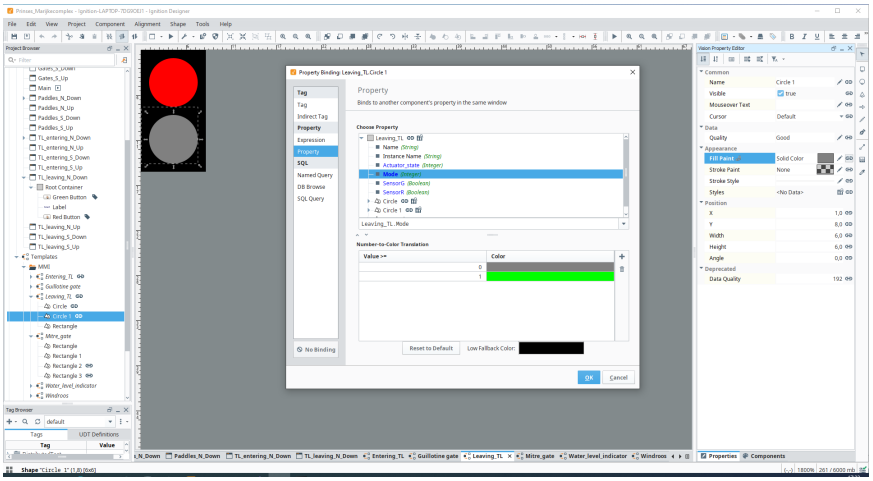

Figure B.16: Settings Leaving Lights downstream

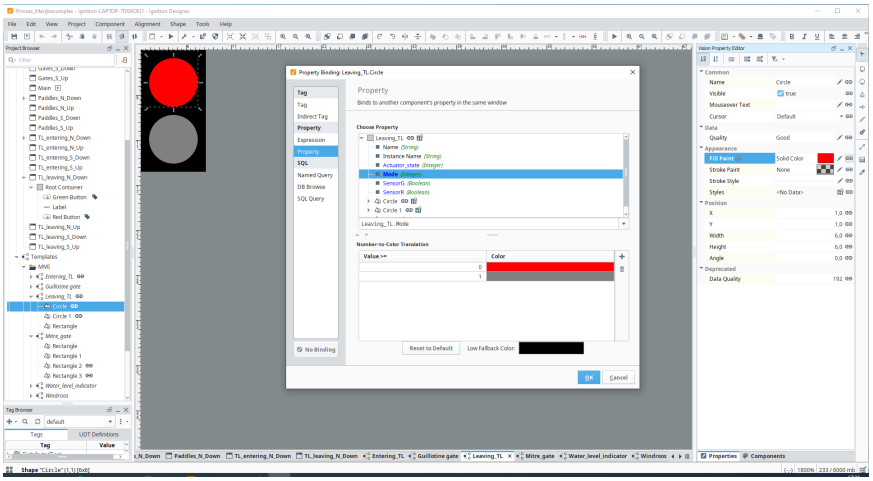

Figure B.17: Settings Leaving Lights

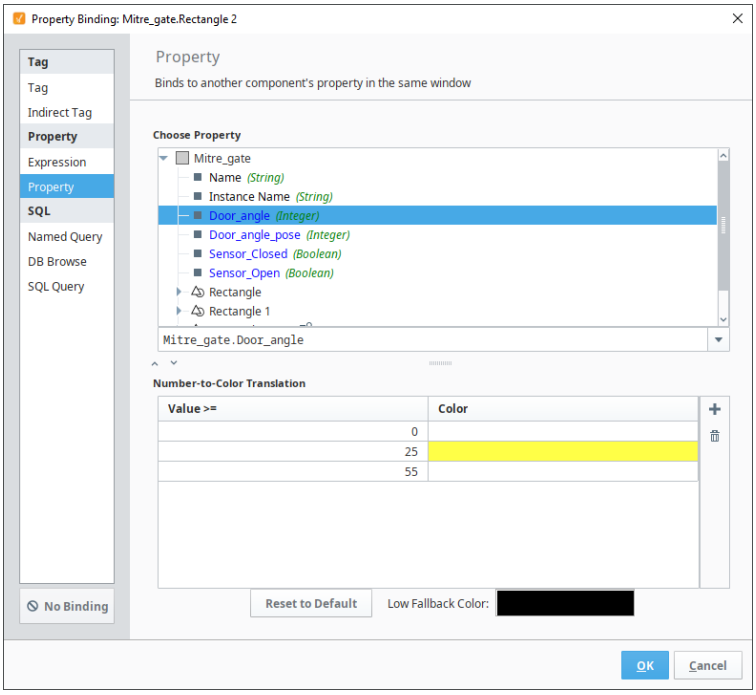

Figure B.18: Settings mitre gate door fill

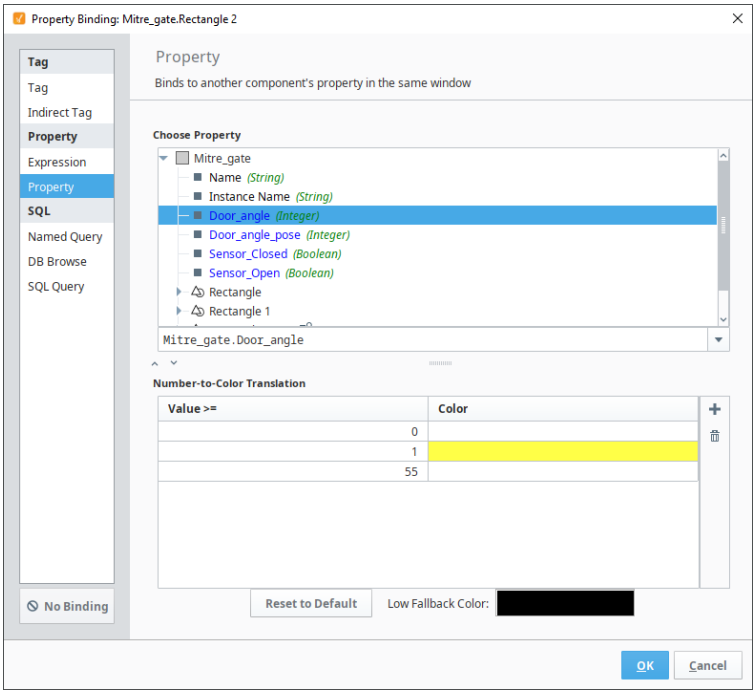

Figure B.19: Settings mitre gate door stroke

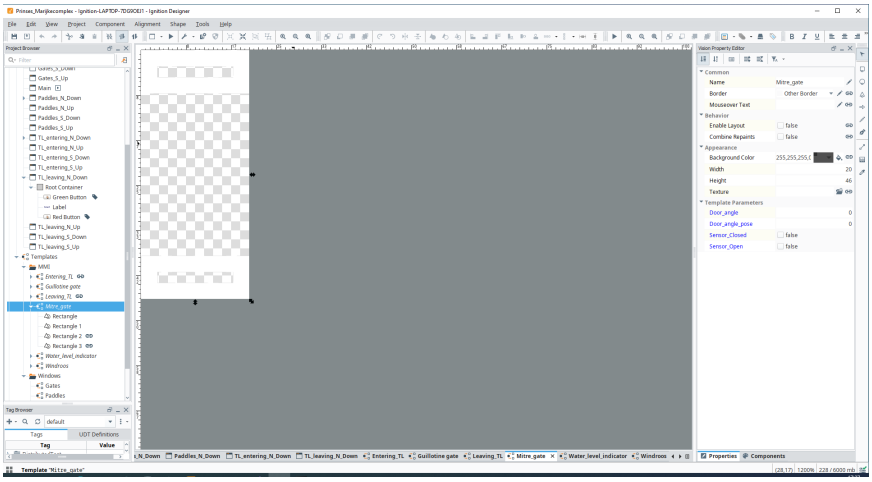

Figure B.20: Template Mitre Gates Overview

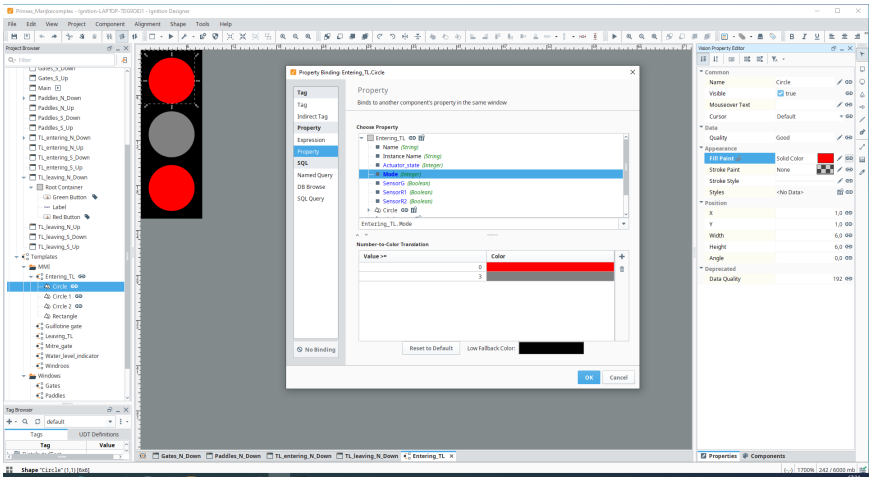

Figure B.21: Template TL Entering Lamp Connected

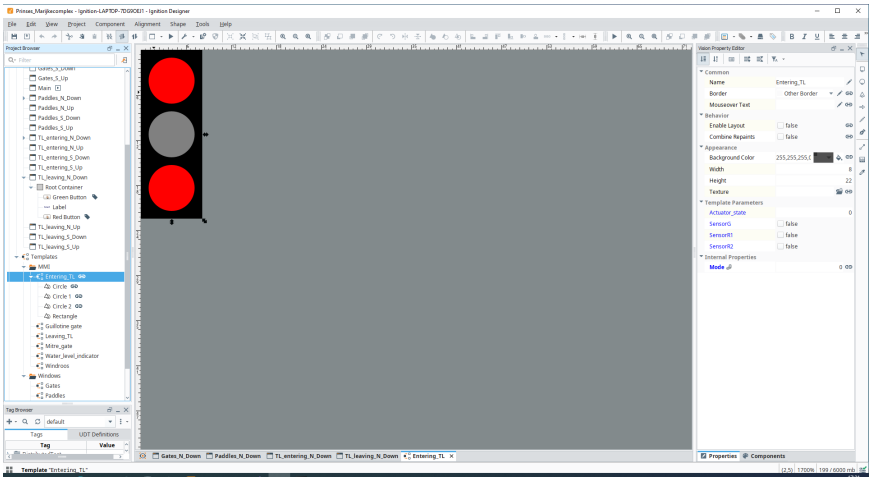

Figure B.22: Template TL Entering Overview

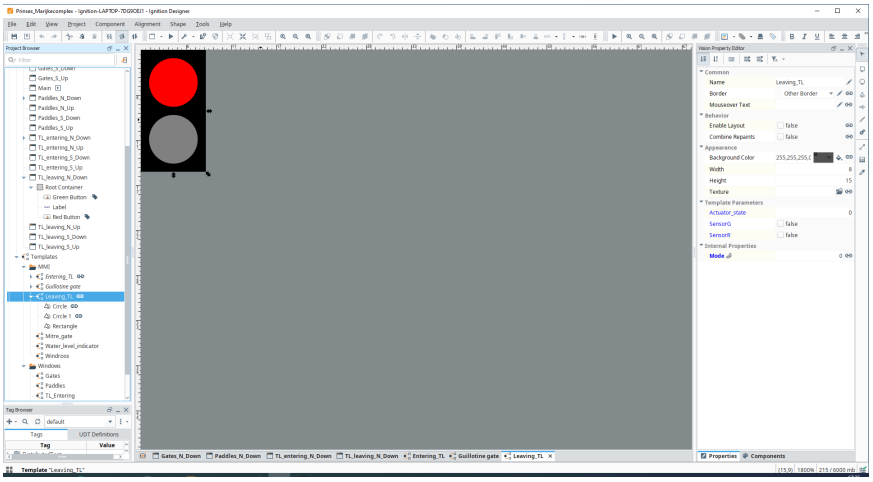

Figure B.23: Template TL Leaving Overview

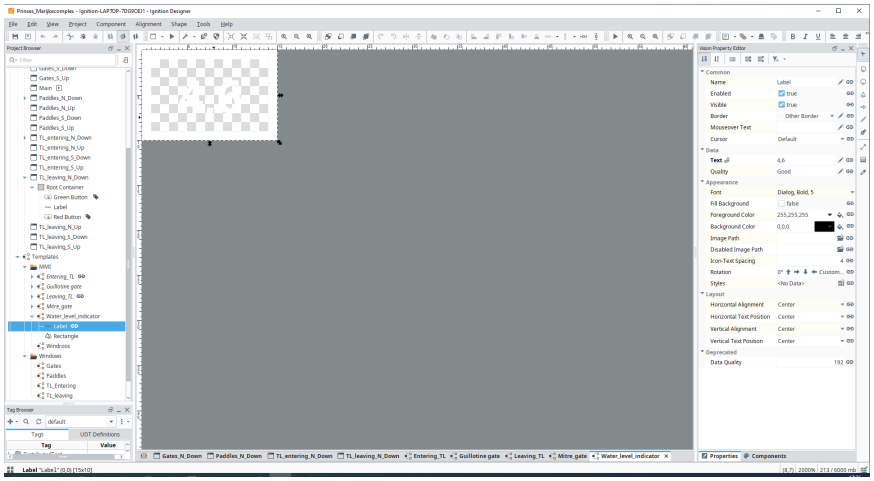

Figure B.24: Template Water Level Indicator

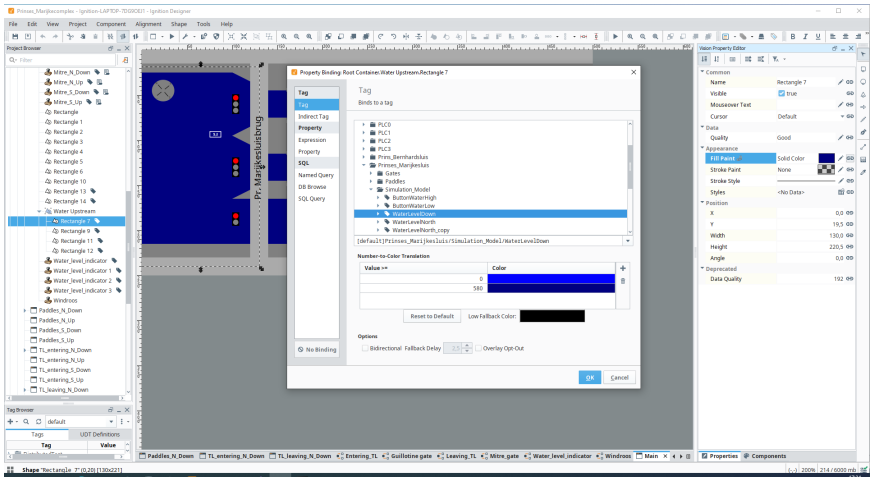

Figure B.25: Water upstream

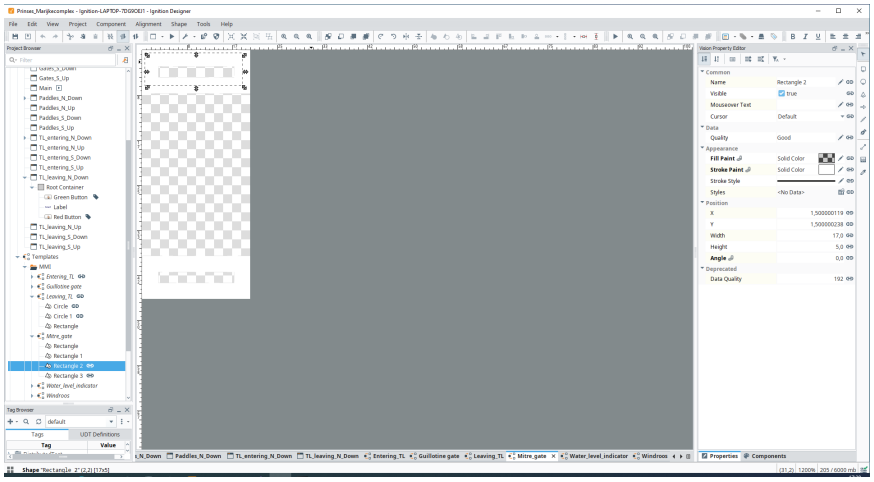

Figure B.26: Window Mitre gate settings

```
using UnityEngine;

public class DoubleToggle : MonoBehaviour
{
    [Tooltip("Double Value of the toggle")]
    public double Double= 0;

    public void Toggle(double oToggle)
    {
        Double = oToggle;
    }
}
```

Listing 11: DoubleToggle.cs

```
using System;
using System.Collections.Generic;
using u040.prespective.prelogic;
using u040.prespective.prelogic.component;
using u040.prespective.prelogic.signal;
using UnityEngine;

public class Input_Double_Logic : PreLogicComponent
{
    #if UNITY_EDITOR || UNITY_EDITOR_BETA
        [HideInInspector] public int toolbarTab;
    #endif

    [Header("Logic and I/O")]

        [Tooltip("BooleanToggle component for the in or output")]
        public DoubleToggle DoubleToggle;
        [Tooltip("Input Boolean (used for input to the PLC)")]
        public double InputDouble;

    #region <<PLC Signals>>
    #region <<Signal Definitions>>
    /// <summary>
    /// Declare the IO signals
    /// </summary>
    ///

    public override List<SignalDefinition> SignalDefinitions
    {
        get
        {
            return new List<SignalDefinition>
            {
                new SignalDefinition(gameObject.name, PLCSignalDirection.INPUT,
    SupportedSignalType.DWORD, "", gameObject.name, null, null, 0.0),
            };
        }
    }
    #endregion
    #endregion

    #region <<Update>>
    /// <summary>
    /// update the simulation component
    /// </summary>
    /// <param name="_simFrame">the current frame since start</param>
    /// <param name="_deltaTime">the time since last frame</param>
    /// <param name="_totalSimRunTime">total run time of the simulation</param>
    /// <param name="_simStart">the time the simulation started</param>
    protected override void onSimulatorUpdated(int _simFrame, float _deltaTime, float
     _totalSimRunTime, DateTime _simStart)
    {
```

```
        readComponent();
    }
    void readComponent()
    {
        if (DoubleToggle.Double != InputDouble)
        {
            InputDouble = DoubleToggle.Double;
            WriteValue(gameObject.name, InputDouble);
        }
    }
}
#endregion
```

Listing 12: *Input_Double_Logic.cs*

The expression used for the (mitre) door angle:

```
max(min({[.]Door_angle_copy}+{[.]Door_angle_speed},{[.]Door_max_angle}),0)
```

```
def valueChanged(tag, tagPath, previousValue, currentValue, initialChange,
    missedEvents):
  """
  Fired whenever the current value changes in value or quality.

  Arguments:
    tag: The source tag object. A read-only wrapper for obtaining tag
         properties.
    tagPath: The full path to the tag (String)
    previousValue: The previous value. This is a "qualified value", so it
                   has value, quality, and timestamp properties.
    currentValue: The current value. This is a "qualified value", so it has
                  value, quality, and timestamp properties.
    initialChange: A boolean flag indicating whether this event is due to
                   the first execution or initial subscription.
    missedEvents: A flag indicating that some events have been skipped due
                  to event overflow.
  """
  door_location = "N_Down"

  system.tag.write("[.]Door_angle_copy", currentValue.value)

  if currentValue.value < (system.tag.read("[.]Door_max_angle").value/2 - 10):
    system.tag.write("[.]Door_angle_pose", 0)
  elif currentValue.value >= (system.tag.read("[.]Door_max_angle").value/2 - 10) and
    currentValue.value <= (system.tag.read("[.]Door_max_angle").value/2 + 10):
    system.tag.write("[.]Door_angle_pose", system.tag.read("[.]Door_mid_angle").value
    )
  elif currentValue.value > (system.tag.read("[.]Door_max_angle").value/2 + 10):
    system.tag.write("[.]Door_angle_pose", system.tag.read("[.]Door_max_angle").value
    )

  if currentValue.value<=5:
    system.tag.write("[.]" + door_location + "_G_E_Sen_I_open", 1)
    system.tag.write("[.]" + door_location + "_G_W_Sen_I_open", 1)
  elif currentValue.value>=50:
    system.tag.write("[.]" + door_location + "_G_E_Sen_I_closed", 1)
    system.tag.write("[.]" + door_location + "_G_W_Sen_I_closed", 1)
  else:
    system.tag.write("[.]" + door_location + "_G_E_Sen_I_open", 0)
    system.tag.write("[.]" + door_location + "_G_E_Sen_I_closed", 0)
    system.tag.write("[.]" + door_location + "_G_W_Sen_I_open", 0)
    system.tag.write("[.]" + door_location + "_G_W_Sen_I_closed", 0)
```

Listing 13: *Door_angle.py*

Expression used for the water level north:

```
({[.]../Paddles/North/Downstream/N_Down_P_E_Act_state}=1401)*({[.]WaterLevelDown})+({[.]../Paddles/North
/Upstream/N_Up_P_E_Act_state}=251)*({[.]WaterLevelUp})+({[.]../Paddles/North/Downstream/N_Down_P_E_Act_state
}=1400)*({[.]../Paddles/North/Upstream/N_Up_P_E_Act_state}=250)*{[.]WaterLevelNorth_copy}
```

```
def valueChanged(tag, tagPath, previousValue, currentValue, initialChange,
    missedEvents):
  """
  Fired whenever the current value changes in value or quality.

  Arguments:
    tag: The source tag object. A read-only wrapper for obtaining tag
         properties.
    tagPath: The full path to the tag (String)
    previousValue: The previous value. This is a "qualified value", so it
                   has value, quality, and timestamp properties.
    currentValue: The current value. This is a "qualified value", so it has
                  value, quality, and timestamp properties.
    initialChange: A boolean flag indicating whether this event is due to
                   the first execution or initial subscription.
    missedEvents: A flag indicating that some events have been skipped due
                  to event overflow.
  """
  system.tag.write("[.]WaterLevelNorth_copy", currentValue.value)

  if currentValue.value == system.tag.read("[.]WaterLevelDown").value:
    system.tag.write("[.]../WaterLevel/North/N_Down_EqualWaterSen_I_equal.value", 1)
  else:
    system.tag.write("[.]../WaterLevel/North/N_Down_EqualWaterSen_I_equal.value", 0)

  if currentValue.value == system.tag.read("[.]WaterLevelUp").value:
    system.tag.write("[.]../WaterLevel/North/N_Up_EqualWaterSen_I_equal.value", 1)
  else:
    system.tag.write("[.]../WaterLevel/North/N_Up_EqualWaterSen_I_equal.value", 0)
```

Listing 14: *WaterLevelNorth.py*

Expression used for the water level south:

```
({[.]../Paddles/South/Downstream/S_Down_P_E_Act_state}=3801)*({[.]WaterLevelDown})+({[.]../Paddles/South
/Upstream/S_Up_P_E_Act_state}=2651)*({[.]WaterLevelUp})+({[.]../Paddles/South/Downstream/S_Down_P_E_Act_state
}=3800)*({[.]../Paddles/South/Upstream/S_Up_P_E_Act_state}=2650)*{[.]WaterLevelSouth_copy}
```

```
def valueChanged(tag, tagPath, previousValue, currentValue, initialChange,
    missedEvents):
  """
  Fired whenever the current value changes in value or quality.

  Arguments:
    tag: The source tag object. A read-only wrapper for obtaining tag
         properties.
    tagPath: The full path to the tag (String)
    previousValue: The previous value. This is a "qualified value", so it
                   has value, quality, and timestamp properties.
    currentValue: The current value. This is a "qualified value", so it has
                  value, quality, and timestamp properties.
    initialChange: A boolean flag indicating whether this event is due to
                   the first execution or initial subscription.
    missedEvents: A flag indicating that some events have been skipped due
                  to event overflow.
  """
  system.tag.write("[.]WaterLevelSouth_copy", currentValue.value)

  if currentValue.value == system.tag.read("[.]WaterLevelDown").value:
    system.tag.write("[.]../WaterLevel/South/S_Down_EqualWaterSen_I_equal.value", 1)
  else:
    system.tag.write("[.]../WaterLevel/South/S_Down_EqualWaterSen_I_equal.value", 0)

  if currentValue.value == system.tag.read("[.]WaterLevelUp").value:
    system.tag.write("[.]../WaterLevel/South/S_Up_EqualWaterSen_I_equal.value", 1)
  else:
    system.tag.write("[.]../WaterLevel/South/S_Up_EqualWaterSen_I_equal.value", 0)
```

Listing 15: *WaterLevelSouth.py*

Scripts used for raising and lowering the water level on the downstream side:

```
def valueChanged(tag, tagPath, previousValue, currentValue, initialChange,
    missedEvents):
  """
  Fired whenever the current value changes in value or quality.

  Arguments:
    tag: The source tag object. A read-only wrapper for obtaining tag
         properties.
    tagPath: The full path to the tag (String)
    previousValue: The previous value. This is a "qualified value", so it
                   has value, quality, and timestamp properties.
    currentValue: The current value. This is a "qualified value", so it has
                  value, quality, and timestamp properties.
    initialChange: A boolean flag indicating whether this event is due to
                   the first execution or initial subscription.
    missedEvents: A flag indicating that some events have been skipped due
                  to event overflow.
  """
  if currentValue.value:
    system.tag.write("[.]WaterLevelDown", 580)

  #North downstream
  if system.tag.read("[.]WaterLevelDown").value == system.tag.read("[.]
    WaterLevelNorth").value:
    system.tag.write("[.]../WaterLevel/North/N_Down_EqualWaterSen_I_equal.value", 1)
  else:
    system.tag.write("[.]../WaterLevel/North/N_Down_EqualWaterSen_I_equal.value", 0)
  #South downstream
  if system.tag.read("[.]WaterLevelDown").value == system.tag.read("[.]
    WaterLevelSouth").value:
    system.tag.write("[.]../WaterLevel/South/S_Down_EqualWaterSen_I_equal.value", 1)
  else:
    system.tag.write("[.]../WaterLevel/South/S_Down_EqualWaterSen_I_equal.value", 0)
  #North upstream
  if system.tag.read("[.]WaterLevelUp").value == system.tag.read("[.]WaterLevelNorth
    ").value:
    system.tag.write("[.]../WaterLevel/North/N_Up_EqualWaterSen_I_equal.value", 1)
  else:
    system.tag.write("[.]../WaterLevel/North/N_Up_EqualWaterSen_I_equal.value", 0)
  #South upstream
  if system.tag.read("[.]WaterLevelUp").value == system.tag.read("[.]WaterLevelSouth
    ").value:
    system.tag.write("[.]../WaterLevel/South/S_Up_EqualWaterSen_I_equal.value", 1)
  else:
    system.tag.write("[.]../WaterLevel/South/S_Up_EqualWaterSen_I_equal.value", 0)
```

Listing 16: *ButtonWaterHigh.py*

```
def valueChanged(tag, tagPath, previousValue, currentValue, initialChange,
    missedEvents):
  """
  Fired whenever the current value changes in value or quality.

  Arguments:
    tag: The source tag object. A read-only wrapper for obtaining tag
         properties.
    tagPath: The full path to the tag (String)
    previousValue: The previous value. This is a "qualified value", so it
                   has value, quality, and timestamp properties.
    currentValue: The current value. This is a "qualified value", so it has
                  value, quality, and timestamp properties.
    initialChange: A boolean flag indicating whether this event is due to
                   the first execution or initial subscription.
    missedEvents: A flag indicating that some events have been skipped due
                  to event overflow.
  """
```

```
  if currentValue.value:
    system.tag.write("[.]WaterLevelDown", 460)

  #North downstream
  if system.tag.read("[.]WaterLevelDown").value == system.tag.read("[.]
    WaterLevelNorth").value:
    system.tag.write("[.]../WaterLevel/North/N_Down_EqualWaterSen_I_equal.value", 1)
  else:
    system.tag.write("[.]../WaterLevel/North/N_Down_EqualWaterSen_I_equal.value", 0)
  #South downstream
  if system.tag.read("[.]WaterLevelDown").value == system.tag.read("[.]
    WaterLevelSouth").value:
    system.tag.write("[.]../WaterLevel/South/S_Down_EqualWaterSen_I_equal.value", 1)
  else:
    system.tag.write("[.]../WaterLevel/South/S_Down_EqualWaterSen_I_equal.value", 0)
  #North upstream
  if system.tag.read("[.]WaterLevelUp").value == system.tag.read("[.]WaterLevelNorth
    ").value:
    system.tag.write("[.]../WaterLevel/North/N_Up_EqualWaterSen_I_equal.value", 1)
  else:
    system.tag.write("[.]../WaterLevel/North/N_Up_EqualWaterSen_I_equal.value", 0)
  #South upstream
  if system.tag.read("[.]WaterLevelUp").value == system.tag.read("[.]WaterLevelSouth
    ").value:
    system.tag.write("[.]../WaterLevel/South/S_Up_EqualWaterSen_I_equal.value", 1)
  else:
    system.tag.write("[.]../WaterLevel/South/S_Up_EqualWaterSen_I_equal.value", 0)
```

Listing 17: *ButtonWaterLow.py*

Script used on the paddles to update the state of to the sensors:

```
def valueChanged(tag, tagPath, previousValue, currentValue, initialChange,
    missedEvents):
  """
  Fired whenever the current value changes in value or quality.

  Arguments:
    tag: The source tag object. A read-only wrapper for obtaining tag
         properties.
    tagPath: The full path to the tag (String)
    previousValue: The previous value. This is a "qualified value", so it
                   has value, quality, and timestamp properties.
    currentValue: The current value. This is a "qualified value", so it has
                  value, quality, and timestamp properties.
    initialChange: A boolean flag indicating whether this event is due to
                   the first execution or initial subscription.
    missedEvents: A flag indicating that some events have been skipped due
                  to event overflow.
  """
  mode = currentValue.value%10

  if mode == 1:
    system.tag.writeBlocking([tagPath[:-11] + "E_Sen_I_closed", tagPath[:-11] + "
    E_Sen_I_open", tagPath[:-11] + "W_Sen_I_closed", tagPath[:-11] + "W_Sen_I_open"],
     [False, True, False, True])
  elif mode == 2:
    system.tag.writeBlocking([tagPath[:-11] + "E_Sen_I_closed", tagPath[:-11] + "
    E_Sen_I_open", tagPath[:-11] + "W_Sen_I_closed", tagPath[:-11] + "W_Sen_I_open"],
     [True, False, True, False])
```

Listing 18: *Paddles.py*

Script used to change the state of the mitre gate, stationary, opening and closing, respectively:

```
def valueChanged(tag, tagPath, previousValue, currentValue, initialChange,
    missedEvents):
  """
  Fired whenever the current value changes in value or quality.

  Arguments:
    tag: The source tag object. A read-only wrapper for obtaining tag
         properties.
    tagPath: The full path to the tag (String)
    previousValue: The previous value. This is a "qualified value", so it
                   has value, quality, and timestamp properties.
    currentValue: The current value. This is a "qualified value", so it has
                  value, quality, and timestamp properties.
    initialChange: A boolean flag indicating whether this event is due to
                   the first execution or initial subscription.
    missedEvents: A flag indicating that some events have been skipped due
                  to event overflow.
  """
  if currentValue.value%10==0:
    system.tag.write("[.]Door_angle_speed", 0)
  elif currentValue.value%10==1:
    system.tag.write("[.]Door_angle_speed", -1)
  elif currentValue.value%10==2:
    system.tag.write("[.]Door_angle_speed", 1)
```

Listing 19: *Mitre_gate.py*

Script used to update the sensors of the traffic lights:

```
def valueChanged(tag, tagPath, previousValue, currentValue, initialChange,
    missedEvents):
  """
  Fired whenever the current value changes in value or quality.

  Arguments:
    tag: The source tag object. A read-only wrapper for obtaining tag
         properties.
    tagPath: The full path to the tag (String)
    previousValue: The previous value. This is a "qualified value", so it
                   has value, quality, and timestamp properties.
    currentValue: The current value. This is a "qualified value", so it has
                  value, quality, and timestamp properties.
    initialChange: A boolean flag indicating whether this event is due to
                   the first execution or initial subscription.
    missedEvents: A flag indicating that some events have been skipped due
                  to event overflow.
  """
  mode = currentValue.value%10

  if mode == 0:
    system.tag.writeBlocking([tagPath[:-9] + "SenG_I", tagPath[:-9] + "SenR1_I",
    tagPath[:-9] + "SenR2_I"], [False, True, True])
  elif mode == 1:
    system.tag.writeBlocking([tagPath[:-9] + "SenG_I", tagPath[:-9] + "SenR1_I",
    tagPath[:-9] + "SenR2_I"], [False, True, False])
  elif mode == 2:
    system.tag.writeBlocking([tagPath[:-9] + "SenG_I", tagPath[:-9] + "SenR1_I",
    tagPath[:-9] + "SenR2_I"], [True, True, False])
  elif mode == 3:
    system.tag.writeBlocking([tagPath[:-9] + "SenG_I", tagPath[:-9] + "SenR1_I",
    tagPath[:-9] + "SenR2_I"], [True, False, False])
  else:
    system.tag.writeBlocking([tagPath[:-9] + "SenG_I", tagPath[:-9] + "SenR1_I",
    tagPath[:-9] + "SenR2_I"], [False, False, False])
```

Listing 20: *TL.py*

```
GVLS.Actuators[0].oToggle  :=MAIN.state0.dvar_M_M_HW_Downstream_Gate_east_Q_Open;
GVLS.Actuators[1].oToggle  :=MAIN.state0.dvar_M_M_HW_Downstream_Gate_east_Q_Close;
GVLS.Actuators[2].oToggle  :=MAIN.state0.dvar_M_M_HW_Downstream_Gate_West_Q_Open;
GVLS.Actuators[3].oToggle  :=MAIN.state0.dvar_M_M_HW_Downstream_Gate_West_Q_Close;
GVLS.Actuators[4].oToggle  :=MAIN.state0.dvar_M_M_HW_Downstream_Paddle_East_Q_Open;
GVLS.Actuators[5].oToggle  :=MAIN.state0.dvar_M_M_HW_Downstream_Paddle_East_Q_Close;
GVLS.Actuators[6].oToggle  :=MAIN.state0.dvar_M_M_HW_Downstream_Paddle_West_Q_Open;
GVLS.Actuators[7].oToggle  :=MAIN.state0.dvar_M_M_HW_Downstream_Paddle_West_Q_Close;
GVLS.Actuators[8].oToggle  :=MAIN.state0.dvar_M_M_HW_Upstream_Gate_Q_Open;
GVLS.Actuators[9].oToggle  :=MAIN.state0.dvar_M_M_HW_Upstream_Gate_Q_Close;
GVLS.Actuators[10].oToggle :=MAIN.state0.dvar_M_M_HW_Upstream_Paddle_Q_Open;
GVLS.Actuators[11].oToggle :=MAIN.state0.dvar_M_M_HW_Upstream_Paddle_Q_Close;
```

Listing 21: Original PLC code

The JSON file containing the Ignition tags of the Prinses Marijke lock complex can be opened by double clicking the paperclip icon: 
The script containing the code used for the reset button, to set the values of all used Ignition tags back to initial values, in Ignition can be opened by double clicking the paperclip icon: 

C DT missing scripts

This appendix contains the code for the missing script finder in Unity, including a figure of the missing script finder tool. In addition, it contains the simulation model and synthesized controller of the Prins Bernhardsluis. At last, the IP address error from Section 4.4 is shown.

```
using System.Collections.Generic;
using System.Linq;
using UnityEditor;
using UnityEngine;

public class FindMissingScriptsRecursivelyAndRemove : EditorWindow
    {
        [MenuItem("Window/Missing Script Window")]
        private static void Init()
        {
            GetWindow<FindMissingScriptsRecursivelyAndRemove>("Missing Script Finder"
    ).Show();
        }

        public List<GameObject> results = new List<GameObject>();

        private void OnGUI()
        {
            if (GUILayout.Button("Search Project"))
                SearchProject();
            if (GUILayout.Button("Search scene"))
                SearchScene();
            if (GUILayout.Button("Search Selected Objects"))
                SearchSelected();

            // src: https://answers.unity.com/questions/859554/editorwindow-display-
    array-dropdown.html
            var so = new SerializedObject(this);
            var resultsProperty = so.FindProperty(nameof(results));
            EditorGUILayout.PropertyField(resultsProperty, true);
            so.ApplyModifiedProperties();
        }

        private void SearchProject()
        {
            results = AssetDatabase.FindAssets("t:Prefab")
               .Select(AssetDatabase.GUIDToAssetPath)
               .Select(AssetDatabase.LoadAssetAtPath<GameObject>)
               .Where(x => IsMissing(x, true))
               .Distinct()
               .ToList();
        }

        private void SearchScene()
        {
            results = FindObjectsOfType<GameObject>()
               .Where(x => IsMissing(x, false))
               .Distinct()
               .ToList();
        }

        private void SearchSelected()
        {
            results = Selection.gameObjects
               .Where(x => IsMissing(x, false))
               .Distinct()
               .ToList();
        }

        private static bool IsMissing(GameObject go, bool includeChildren)
        {
```

```
60              var components = includeChildren
61                  ? go.GetComponentsInChildren<Component>()
62                  : go.GetComponents<Component>();
63
64              return components.Any(x => x == null);
65          }
66
67          public class RemoveMissingScripts : Editor
68          {
69              [MenuItem("GameObject/Remove Missing Scripts")]
70              public static void Remove()
71              {
72                  var objs = Resources.FindObjectsOfTypeAll<GameObject>();
73                  int count = objs.Sum(GameObjectUtility.
    RemoveMonoBehavioursWithMissingScript);
74                  Debug.Log($"Removed {count} missing scripts");
75              }
76          }
77
78  }
```

Listing 22: FindMissingScriptsRecursivelyAndRemove.cs

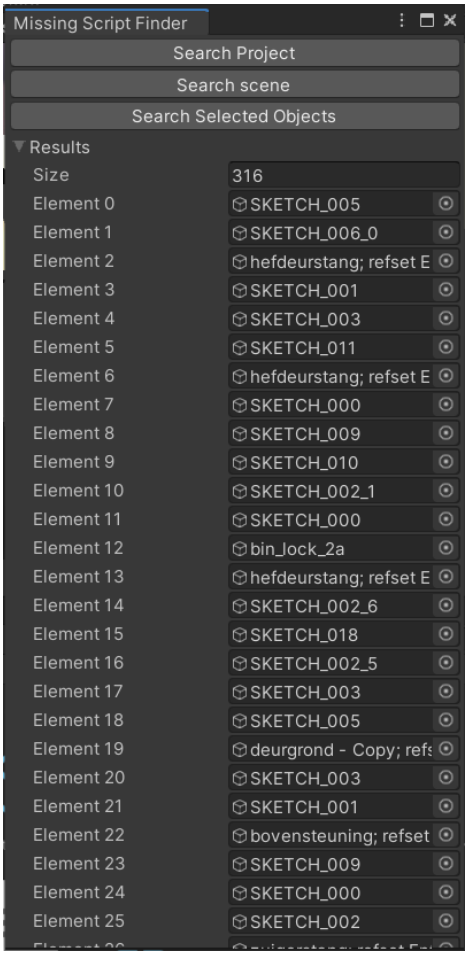

Figure C.1: Missing scripts finder

Prins Bernhardsluis simulation model and synthesized supervisory controller: (Open by double clicking the paperclip icon.)

```
ArgumentException: IP Address is invalid
Parameter name: name
Org.BouncyCastle.Asn1.X509.GeneralName..ctor (System.Int32 tag, System.String name) (
    at <4b876a5efd1b42a095fbfedce2ffc961>:0)
CertificateFactory.CreateSubjectAlternateNameDomains (System.Collections.Generic.
    IList`1[T] domainNames) (at <dc6e3f13df7d40c7a65f31d64b38a0b7>:0)
CertificateFactory.CreateCertificate (System.String storeType, System.String
    storePath, System.String password, System.String applicationUri, System.String
    applicationName, System.String subjectName, System.Collections.Generic.IList`1[T]
     domainNames, System.UInt16 keySize, System.DateTime startTime, System.UInt16
    lifetimeInMonths, System.UInt16 hashSizeInBits, System.Boolean isCA, System.
    Security.Cryptography.X509Certificates.X509Certificate2 issuerCAKeyCert, System.
    Byte[] publicKey, System.Int32 pathLengthConstraint) (at <
    dc6e3f13df7d40c7a65f31d64b38a0b7>:0)
Opc.Ua.Configuration.ApplicationInstance+<CreateApplicationInstanceCertificate>d__55.
    MoveNext () (at <f0c8c481657f471c9bea6b518609558a>:0)
--- End of stack trace from previous location where exception was thrown ---
System.Runtime.ExceptionServices.ExceptionDispatchInfo.Throw () (at <9577
    ac7a62ef43179789031239ba8798>:0)
System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess (System.Threading.
    Tasks.Task task) (at <9577ac7a62ef43179789031239ba8798>:0)
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification (
    System.Threading.Tasks.Task task) (at <9577ac7a62ef43179789031239ba8798>:0)
System.Runtime.CompilerServices.TaskAwaiter.ValidateEnd (System.Threading.Tasks.Task
    task) (at <9577ac7a62ef43179789031239ba8798>:0)
System.Runtime.CompilerServices.TaskAwaiter`1[TResult].GetResult () (at <9577
    ac7a62ef43179789031239ba8798>:0)
Opc.Ua.Configuration.ApplicationInstance+<CheckApplicationInstanceCertificate>d__51.
    MoveNext () (at <f0c8c481657f471c9bea6b518609558a>:0)
--- End of stack trace from previous location where exception was thrown ---
System.Runtime.ExceptionServices.ExceptionDispatchInfo.Throw () (at <9577
    ac7a62ef43179789031239ba8798>:0)
System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess (System.Threading.
    Tasks.Task task) (at <9577ac7a62ef43179789031239ba8798>:0)
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification (
    System.Threading.Tasks.Task task) (at <9577ac7a62ef43179789031239ba8798>:0)
System.Runtime.CompilerServices.TaskAwaiter.ValidateEnd (System.Threading.Tasks.Task
    task) (at <9577ac7a62ef43179789031239ba8798>:0)
System.Runtime.CompilerServices.TaskAwaiter`1[TResult].GetResult () (at <9577
    ac7a62ef43179789031239ba8798>:0)
u040.prespective.prelogic.adapters.opcua.OPCUAClient+<CreateSession>d__21.MoveNext ()
     (at <8676f7efb13e4f2193727a070f7c44cb>:0)
--- End of stack trace from previous location where exception was thrown ---
System.Runtime.ExceptionServices.ExceptionDispatchInfo.Throw () (at <9577
    ac7a62ef43179789031239ba8798>:0)
System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess (System.Threading.
    Tasks.Task task) (at <9577ac7a62ef43179789031239ba8798>:0)
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification (
    System.Threading.Tasks.Task task) (at <9577ac7a62ef43179789031239ba8798>:0)
System.Runtime.CompilerServices.TaskAwaiter.ValidateEnd (System.Threading.Tasks.Task
    task) (at <9577ac7a62ef43179789031239ba8798>:0)
System.Runtime.CompilerServices.TaskAwaiter.GetResult () (at <9577
    ac7a62ef43179789031239ba8798>:0)
u040.prespective.prelogic.adapters.opcua.OPCUAStreamAdapter+<Connect>d__13.MoveNext
    () (at <8676f7efb13e4f2193727a070f7c44cb>:0)
--- End of stack trace from previous location where exception was thrown ---
System.Runtime.ExceptionServices.ExceptionDispatchInfo.Throw () (at <9577
    ac7a62ef43179789031239ba8798>:0)
System.Runtime.CompilerServices.AsyncMethodBuilderCore+<>c.<ThrowAsync>b__6_0 (System
    .Object state) (at <9577ac7a62ef43179789031239ba8798>:0)
UnityEngine.UnitySynchronizationContext+WorkRequest.Invoke () (at <
    a555715ef291462eb190c2c939543f55>:0)
UnityEngine.UnitySynchronizationContext:ExecuteTasks()
```

Listing 23: IP address error