



**DEVELOPING A MODULAR SIMULATION TOOL CAPABLE OF SIMULATING
ELECTRIC VESSELS IN REAL-TIME**

Lappeenranta-Lahti University of Technology LUT

Degree Programme in Electrical Engineering, Master's Thesis

2022

Aleksi Kettunen

Examiners: Assistant Professor Niko Nevaranta

D.Sc. (Tech.) Tuomo Lindh

Supervisor: M.Sc. (Tech.) Anssi Mattus

Abstract

Lappeenranta-Lahti University of Technology LUT
LUT School of Energy Systems
Electrical Engineering

Aleksi Kettunen

Developing a Modular Simulation Tool Capable of Simulating Electric Vessels In Real-Time

Master's Thesis

2023

73 pages, 33 figures, 3 tables.

Examiners: Assistant Professor Niko Nevaranta and D.Sc. (Tech.) Tuomo Lindh

Keywords: Electric Vessel, Functional Mock-Up Unit, Real-Time Simulation, Simulation

The aim of this thesis was to design and develop a simulation tool capable to simulate electric vessels, with various component configurations, in real-time. Thesis was done for Danfoss Editron. Simulation tool was to be capable to represent a physical vessels inputs and outputs in software, with the aim of saving costs in the testing and deployment of electric vessel management software, running on Beckhoff PLCs. Python was chosen as the programming language of the simulation tool, and simulations were done using Functional Mock-up Units (FMUs). Simulation models were developed using MathWorks Simulink. Operational simulation accuracy was targeted for the simulation tool, along with configurability and modularity. This was to allow for different electric vessel configurations. As the aim was to achieve real-time simulation speed, was simultaneous multiprocessing of Python programming language utilized. Simulation tool development was separated into 3 segments. Logic simulation segment was into represent the logic and represent inverters to the PLCs. Logic simulator communicates with the PLCs using Automatic Device Specification (ADS). Electric simulation segment executed all the dynamics simulations. For switching-type circuits, such as inverters and converters, was duty-cycle based approximation applied, to achieve real-time calculation speed. Calculation speed was also improved upon, by segmenting simulations to different processes, where simulation blocks would only communicate with certain intervals. Graphical user interface (GUI) was developed for end-user to interact with the simulation tool. The results from the simulation tool where compared against results from a real world vessel, where operational accuracy of the simulation tool was validated. Simulation tool as an entity can be considered fulfilling its requirements, and a good foundation to build upon. Performance results of simulation tool implicates that the type of approach with different communication intervals in simulations can achieve major simulation time improvements and similar methodologies could be utilized in simulations, where accuracy is not the most important factor.

Tiivistelmä

Lappeenrannan-Lahden teknillinen yliopisto LUT
LUT Energiajärjestelmät
Sähkötekniikka

Alexi Kettunen

Sähköisten Alusten Reaaliaikaiseen Simulointiin Kykenevän Modulaarisen Työkalun Kehittäminen

Diplomityö

2023

73 sivua, 33 kuvaa, 3 taulukkoa.

Tarkastajat: Apulaisprofessori Niko Nevaranta ja TkT Tuomo Lindh

Hakusanat: Sähkölaiva, Functional Mock-Up Unit, Reaaliaikainen simulointi, Simulaatio

Tämän diplomityön tavoitteena oli suunnitella ja kehittää simulointityökalu, joka kykenee simuloimaan sähköisiä aluksia ja niiden erilaisia variaatioita reaaliajassa. Diplomityö tehtiin Danfoss Editron Oy:lle. Simulointityökalun tulisi kyetä esittämään fyysisen aluksen rajapintaa, tavoitteena vähentää aluksen ohjausjärjestelmän testaamiseen ja käyttöönottoon kuluja. Ohjausjärjestelmän alustana toimi Bechhoff:n valmistamat PLC:t. Työkalun ohjelmointikieleksi valittiin Python. Simulaatiot toteutettiin Functional Mock-up Unit (FMU):lla. Simulaatiomallit kehitettiin MathWorks Simulink:lla. Simulaation tarkkuusvaatimukseksi asetettiin toiminnallinentarkkuus. Simulaatio työkalun tulisi myös olla modulaarinen, sekä helposti muunneltavissa eri alusprojektien tarpeisiin. Jotta reaaliaikaisuus olisi saavutettavissa, samanaikaista moniydin prosessointia hyödynnettiin kehityksessä. Simulointityökalu voidaan jakaa kolmeen osa-alueeseen. Logiikkasimulaattori vastaa logiikan ja inverttereiden emuloimisesta PLC:lle. ADS-kommunikointiprotokollaa käytettiin kommunikointiin PLC:n kanssa. Dynamiikkasimulaattori vastasi dynamiikan simuloinnista. Simuloinnin nopeutta parannettiin ajamalla eri komponentit eri prosesseissa. Nämä prosessit kommunikoiivat keskenään tietyin aikavälein. Graafinen käyttöliittymä kehitettiin, työkalun ja käyttäjän väliseen vuorovaikutukseen. Simulointityökalun lopputuloksia verrattiin oikeaan alukseen. Vertailun pohjalta voitiin työkalun toiminnallinen tarkkuus varmistaa. Simulointityökalun voidaan todeta täyttävän sille asetetut vaatimukset. Simulointityökalu on myös hyvä pohja tulevaa kehitystyötä varten. Simulointityökalun suorituskyvyn perusteella voidaan myös todeta, että on mahdollista saavuttaa merkittäviä simulaatioaika vähennyksiä. Samankaltaisia metodeja voitaisiin käyttää simulaatioissa, missä simulaation tarkkuus ei ole sen pääprioriteetti.

Acknowledgements

This thesis was completed for Danfoss Editron during 2022. Thank you for providing me with this subject, and all support received inside the company from many individuals. Especially I'm grateful to Anssi Mattus, for acting as the supervisor for the thesis, and providing the possibility to test the vessel in real world in Estonia. Trip was important for this thesis, but was a great journey besides that. Special thanks to Andrey Lana, for acting as a guide and a reference for the simulations, and for general interest on the subject. Thank you to all my co-workers at Danfoss Editron, for providing great working environment, and atmosphere.

I would also express my gratitude towards Assistant Professor Niko Nevaranta, and D. Sc. Tuomo Lindh, who acted as the instructors and examiners of the thesis. I want to thank for the feedback, guidance and advices during the process of making this thesis.

Finally, thanks to all my friends for fruitful and unforgettable study time here at Lappeenranta. For making those late night study sessions at Sätky guild room almost bearable. And making the free time here something one would not want to give up, and move on from.

Aleksi Kettunen
Lappeenranta, 30.12.2022

Symbols and abbreviations

Symbols

Δt	Time difference
D	Inverter Duty-cycle
I_{gen}	Current produced by generator, towards DC-link
I_{in}	Inverter input-side current
I_{load}	Current consumed by load, from DC-link
I_{out}	Inverter output-side current
k	Moving average window length
t_0	Start time
t_1	End time
t_n	Time
t_s	Step interval
t_{end}	Simulation end time
t_{it}	Idle time
t_{or}	Overrun time
t_{start}	Simulation start time
U_{dc}	DC-link Voltage
V_{in}	Inverter input-side voltage
V_{low}	Lowest measured DC-link voltage
V_{out}	Inverter output-side voltage
AC	Alternating Current
ADS	Automatic Device Specification
AFE	Active Front End
API	Application programming interface
CAN	Controller Area Network
CLI	Command-Line Interface
DC	Direct Current
DT	Digital Twin
EMS	Editron Marine System
FAT	Factory Acceptance Test
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
FPGA	Field Programmable Gate Array
GUI	Graphical User Interface
HAT	Harbour Acceptance Test
IGBT	Insulated-gate bipolar transistor

IO	Input-Output
JSON	JavaScript Object Notation
MAC	Media Access Control
MOSFET	Metal-oxide-semiconductor field-effect transistor
PC	Personal Computer
PLC	Programmable Logic Controller
PS	Portside
rpm	Rounds per minute
RTC	Real-Time Clock
SAT	Sea Acceptance Test
SB	Starboard
UI	User Interface
XML	Extensible Markup Language
ZMQ	ZeroMQ

Table of contents

Abstract

Tiivistelmä

Acknowledgements

Symbols and abbreviations

1	Introduction	11
1.1	Background	12
1.2	Aim of the Project	13
1.3	Advantages of performing simulations	14
2	Requirements for the software simulation tool	18
2.1	Main Components of the Electric Vessel	18
2.1.1	DC-Grid	20
2.1.2	Power sources	21
2.1.3	Loads	21
2.1.4	Fuses	22
2.2	Electric Converters	22
2.3	Communication with PLC	25
2.4	Selecting the tools for development	26
2.5	Functional Mock-Up Interface and Function Mock-Up Units	28
2.6	Impact of Programming Languages On Runtime Performance	30
2.7	Threading and Multiprocessing	30
3	Programming of the simulation tool	33
3.1	Simulation Environment Description	33
3.2	Logic Simulator	34
3.3	Vessel Simulations	34
3.4	Software Requirements	35
3.5	Design and Structure of the Simulation Tool	37
3.5.1	Logic Simulator part for simulation tool	41
3.5.2	Vessel Simulator Structure	42
3.5.3	Graphical User Interface	45
3.5.4	Configurability	46
3.5.5	Outside Communication	47
3.5.6	Communication with other software	48
4	Electrical Simulations	50
4.1	Digital Twin Classification	50
4.2	Simulation Models	51
4.2.1	Diesel Engine	51
4.2.2	Store model	52
4.2.3	Modelling DC-Link and bus tie	54
4.2.4	Load Model, Grid Model and Shore Connection	56
4.2.5	Propulsion	57

4.3	Connections Between Models	58
5	Results	60
5.1	Simulator Performance Analysis	60
5.2	Real World Test Case	61
5.2.1	Vessel Setup	61
5.2.2	Tests	61
5.2.3	Test Results	62
5.2.4	Conclusions about the simulations	68
5.3	Runtime Analysis	68
6	Conclusions	70
6.1	Future work	70
	Bibliography	72

List of figures

1.1	Example of an electric vessel case, from Danfoss Editron marketing material.	12
1.2	Illustration between real-time system and offline system	17
2.1	Generalized main components of an electric vessel	19
2.2	Example of a medium voltage shipboard system	20
2.3	Simplified averaging converter	23
2.4	ADS protocol transport layer description	25
2.5	ADS communication description	26
2.6	ADS and CANopen communication difference	27
3.1	Environment where the simulator working in and its relations to the other components.	33
3.2	Generalized simulator function block diagram	37
3.3	Simulation timeline of an average simulation cycle.	39
3.4	Simulator process division	40
3.5	Logic simulator block diagram	42
3.6	Dynamics simulation block diagram	43
3.7	Dynamics simulation timeline	44
3.8	Simulator variable step size visualized.	44
3.9	ZMQ Request – Reply messaging pattern	49
4.1	Simplified diesel engine and generator Simulink-model	52
4.2	DC-DC converter and battery Simulink-model	54
4.3	Bus tie Simulink-model	55
4.4	DC-link Simulink model	55
4.5	General load Simulink model	56
4.6	Shore and grid combined Simulink model	57
4.7	Propulsion Simulink model	58
4.8	Simulation models communication generalized	59
5.2	DC voltage on vessel cold boot	64
5.3	DC-link voltage reaction to sudden load	65
5.4	Propulsion comparison	66
5.5	Propulsion comparison, moving average	67
5.6	Propulsion comparison RPM	67
5.7	Relative time difference between simulators	69

List of tables

1	Use cases of electric converters in shipboard applications	23
2	Requirements for developing the simulation software	36
3	Program requirements	46

1 Introduction

An electrical vessel can be classified as a subcategory of vessels, that are powered partly or fully by electricity. Such technologies are an emerging trend, trying to reduce marine exhaust emissions. Fuel costs are a large portion of a ships life-cycle costs, and this has lead to an economic development, where power electronics based drives are now used for propulsion, and different solutions such as diesel generator are used to provide power for those power electronics. This allows for decoupling the running speed of diesel generators from the propulsion. As diesel generators have optimal speed range where they are efficient, this decoupling allows for diesel generators to run at optimal speeds for fuel savings, and propulsion to run at its required speed. As other needs have grown, so has the complexity of the systems. For example today's vessels shipboard systems can have battery systems instead or along with the diesel generator, hotel grids providing power to all the ship, and hybrid power drive systems with both diesel and electric propulsion.

So for companies designing and manufacturing, those complex systems, it is of great importance, to develop them and deploy them as efficiently as possible. Fig. 1.1 shows an example electric vessel, containing basic components that most electric vessels will contain. A supplier of such systems needs to master every component from propulsion to power plants, to design or manufacture such system. All of these entities contain smaller components that need to be thoroughly tested. While the components themselves are important, behind these systems, need to be a functional system controlling them. Programmable logic controllers (PLCs) are used as control and management systems, and the users have the possibility to interact with these via different interfaces. Software running on PLCs needs to be developed and tested. Developing and deploying software for ships can be a time-consuming task, as it requires completing standardized tests before it can be approved. This includes the time needed to develop the software and deploy it in a real working environment. Therefore, being able to simulate and test the software beforehand is an advantage for the developer of such systems.

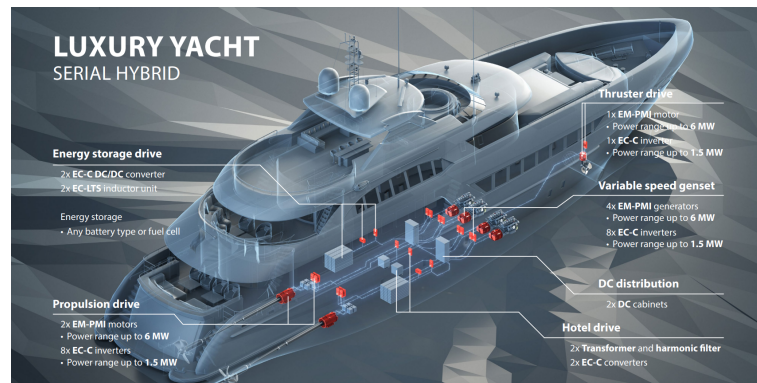


Figure 1.1. Example of an electric vessel. The Figure has been taken from Powerpoint, 2022.

1.1 Background

Danfoss Editron is a company that provides hybrid powertrain solutions for industrial usage in on-highway, off-highway and in marine applications. When considering marine solutions only, Danfoss Editron provides systems for both hybrid-vessels as full electric vessels. Power system is responsible for both, generation of electrical power and the consumption of it in propulsion and other use cases. The PLC software is used to control these devices and to provide a quality experience for the client. Such system is internally named Editron Marine System (EMS). To ensure this experience, the PLC software needs to be tested, and best approach to replicate this is to simulate the real situation what the PLC faces after deployment. Doing this, can make programmers work easier, and reduce time needed to deploy it. Both research and resources have been dedicated by Danfoss Editron to perform exactly this; simulate PLC environment to test PLC software. Previous research conducted (Gräsbeck, 2021), was focused to simulating the logic of the vessel. This is important, but if electrical functionality is not considered simultaneously it might result in variables that are not accounted for in the testing. As a result, the PLC software might give wrong instructions to inverters, resulting in wrong voltages being driven in end product. These kinds of errors are usually encountered during the deployment process and requires patching. But finding these issues earlier in the process, can remove most of them entirely, thus saving money and time.

Testing the software to be deployed to vessels, is a required process by classification societies. Before a vessel can be taken into use, it is required by country-specific regulations, that the software is to be checked for the safety and reliability reasons, before it can be deployed. Qualifications must be achieved for many of the vessels systems, including the

software. Factory acceptance test (FAT) is performed on the upon the software that is developed, to be make sure that software at least in ideal conditions, performs as expected. In many cases, it is not necessary to have a physical representation of the entire vessel for these tests. For many of the different factory acceptance tests (FATs), it is sufficient to simulate the interfaces of the vessel to prove that the software works correctly. This means that the simulation or testing environments do not need to be perfect representations of real-world applications for the software to pass these tests.

The testing procedure for vessel software also includes Harbour Acceptance Test (HAT) and Sea Acceptance Test (SAT), where the vessels systems are tested under real conditions. In these tests, the functionality of the software can be evaluated and tested in real-world conditions, but any issues that arise during the testing process can significantly delay the deployment of the vessel. Often, the deployment team is not working alone on the vessel, as typically other workers are focusing on other aspects of the vessel. Non-working software could also be a major set-back for other teams working on the dock, especially if other systems require something that depends on the automation software. But on the contrary, a fully working software that does not need any additional work done at the deployment site makes things much easier for the deployment team.

Fixing issues related to software is easier, cheaper and saves more time, the earlier in its lifetime it is done. To find the possible issues developing a proper simulation environment for vessels can be beneficial, as potential problems could be found earlier, and time used in deployment could be shortened.

1.2 Aim of the Project

The aim of this thesis is to develop a software package that can simulate the electrical systems and other relevant dynamics, such as a diesel engine, of a vessel in real time, building upon all previous work done at Danfoss Editron on logic simulations. Simulation tools will be used by software developers, and setting up simulations, should be straight forward, cost-effective, and time effective. This thesis is straight follow-up to previous thesis work done at Danfoss Editron (Gräsbeck, 2021). While the objectives remain the same, a method is to be

accomplished that expands on what was previously done. So instead of producing software that can simulate just the communication as the PLC sees it, electrical system simulation and other system dynamics will be added.

Entirety of the simulator to be developed could be classified as digital twin (DT) of an electric vessel. DT is a virtual model that is made to represent a physical object, in this case a vessel. But in this case the DT development is limited to only the relevant parts of the physical object, that are directly connected on how the electrical systems work. This means other systems that the vessel has and that are not related to electric powertrain and hotel systems of the vessel are ignored. (IBM, 2022)

Vessels electrical systems consists of both AC and DC circuits. The focus point of the electrical simulations considered in this thesis will be on the DC components of the vessel. This keeps simulations less demanding to solve, while still retains the most important parts of the components and their dynamics in terms of the needs of this thesis.

Dynamic systems that are simulated thus include plausible diesel engines and rotating electrical machines. These components of the vessel directly affect how the electrical systems that are provided by Danfoss Editron to the customer, act and perform during operation. While these simulations need to be done to describe the overall system in a more detailed manner, the accuracy of these simulations are kept at a lower accuracy level. The focus of this thesis is on the electrical systems, particularly the DC-bus, and the software required to perform the simulations, rather than the simulations themselves. In most cases, operational accuracy for simulations is sufficient, and the goal is not to create the most accurate simulation, but rather to create an environment in which to run those simulation models.

1.3 Advantages of performing simulations

Performing simulations or other types of tests, and the advantages they bring are not new concept in the industry. While tools are widely available for the majority of systems, this does not apply to more specific systems, and for instance such tools do not exist for combining the specific needs of performing IO-simulation to PLCs and simultaneously also simulating

the dynamics of the connected systems that the PLCs are meant to control. Software and hardware provider for Beckhoff PLCs does provide a software suite, that makes it possible to run software, and integrate dynamics models from other software, such as MathWorks Matlab Simulink. Software like this, can be used, and have been used (Kaikko, 2015) and (Haapaniemi, 2016), but despite its solid properties there are some trade-offs that limits usage in this use-case. While simulink offers tools to develop models, using it alone, without any other tools does not provide options to easily change the configuration and components of the model, without additional development time. Other tools are required for performing simulations such as MeVEA. Models also has to be developed for that specific use case. Simulinks provided environment lacks the flexibility required for different configurations that the electric vessel projects can have. However, this could be solved, with a dynamic model generation system, where models are generated on demand. As vessel component configurations vary largely between projects, a flexible simulation environment is required, where the amount of components is easily interchangeable.

So even if the software suite makes it possible to develop systems required, it might be the case that is not flexible enough solution. Using this type of solution to test the same PLCs also has an issue with where the simulation is run on. The simulation program can be running on same PLC as the tested PLC program. But this results that the tested PLC simulation not being the same as the one that is actually deployed to customer and could result in unexpected behaviour.

Testing the final version of PLC software on against a specifically developed hardware test system, represents the reality most closely. It is the closest environment to the final environment of tested software, but it has compromisses that need to be made. First is the costs involved for the hardware. If the simulation software is running on a separate PLC, this brings additional hardware requirements. Moreover, if inputs and outputs of the system are simulated, devices to make that possible are also required. Such testing system also might need changes after testing, as a result of the testing, or for outside reasons. Work required to make changes, and to build this test setup also need to be accounted. It is also possible, that hardware based solution is not cost-efficient solution to serve all required needs of such system. By using testing software instead of hardware test system, the hardware changes and costs can be kept to minimum.

In the final stage software is being deployed to a vessel with final hardware and tests need to be done anyway. While testing hardware on hardware provides advantages, those advantages diminish if the software requires changes in the commissioning phase. If a software simulation is accurate enough the advantages gained by hardware on hardware simulation in accuracy can be outweighed by the flexibility and cost-effectiveness of a software-based simulator. With a properly implemented simulating suite, the software can be developed faster, and also in a cost-effective manner. Also, there are some test situations, that might be troublesome to execute using hardware based test system, or just testing on-site with the final hardware. Doing these types of tests virtually on software is much more feasible approach.

In past similar studies, simulator software suites have been developed and research about software simulating has been done. In thesis (Kuusela, 2019) a real-time simulator was developed, that could simulate multi-drive systems and a common DC system. This was achieved using ABB closed source software. While the work done cannot be directly applied in this thesis, but proof of concept remains, that it is plausible for switching systems like the ones considered in this thesis, is calculationally plausible to carry out in real-time.

In the same thesis work, results of the executed simulator were reported and compared to a high fidelity simulator. The results show that when properly executed, variables like DC link voltage in steady-state simulations are within few percents, even if the run time performance of the simulators is very different. Larger changes in accuracy can be seen in smaller timesteps. While accuracy in with small timesteps is important, in a simulator running at real-time speed, that information can be harder to extract and its usefulness is debatable for the use cases in this thesis. The main goal of the simulations in this thesis is to provide realistic reference values for the PLCs. The simulations are not intended to act as a simulation model that accurately represents an actual electric vessel on more than an operational level.

Most commonly simulator solutions are based on running software locally on a PC. But running software on PC is not a must, and simulations can be done on separate hardware running on Field Programmable Gate Arrays (FPGA), or similar microcontrollers. In (Mustafa et al., 2019) was an approach presented to model and simulate models on FPGA. The work was executed using Simulink Coder and Xilinx Vivado HDL.

The previous work done in (Mustafa et al., 2019) can act as proof of concept for this project. Approaching software simulation with a different hardware than traditional the operating systems modern personal computers have, has advantages. Simulation could be run in a hard real-time system instead of a soft real-time system achieved in personal computers. Running hard real-time software provides some obvious advantages over soft real-time calculations. With hard real-time systems, the system responds in a manner that is predictable. On the contrary, in the case of soft real-time systems, there is always a question if the system can respond within a given deadline. For example, a programs run time is not always the same, can be varying, this could mean that the run time of a software is longer than the expected running time. In a use case where it is expected to run along the real-time clock, these unexpectancies can cause the program to run slower, hence not running in real time. With hard real-time systems, runtime is consistent, and program will execute within given time limits. Hard real-time systems are more predictable and consistent because of this and can be considered to be more reliable because of these properties. However, this does not mean soft real-time calculations necessarily cannot be run in real-time, it in practice means that the latency of such system might be higher than similar system executed in hard real-time, and reliability of system staying consistently in real-time, without slowdowns. System is by definition running in real-time, when its per cycle execution time is shorter than its sampling time. In Fig. 1.2, is an illustration between real-time and offline simulation. (Omar Faruque et al., 2015)

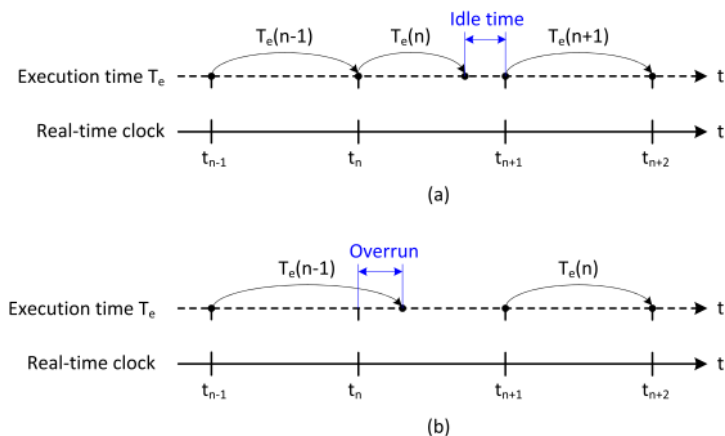


Figure 1.2. Illustration between (a) Real-time simulation. (b) Non-real-time simulation (Omar Faruque et al., 2015). Overrun (t_{or}), means the time simulator runs over the time allocated for the next simulation cycle. Idle time (t_{it}) on the contrary, means the time that is spent waiting until next allocated time period.

2 Requirements for the software simulation tool

In this chapter, the components and the fundamental system parts of a vessel are explored, for which knowledge of is required to execute a simulator for such systems. Moreover, the tools that are to be used in the development of the simulation environment(tool) are also introduced, and foundations that the simulations, and the tool is based on is explored.

2.1 Main Components of the Electric Vessel

At a central point of this thesis, is an electric vessel. The definition of an electric vessel is a seafare vehicle that is mainly, or partly powered with electricity. Vessels also use many voltage levels for different parts of the ship e.g. electrical distribution and internal dc-grid. Powerful diesel generators used in the vessel generate larger voltage levels, to what for example 230V 50Hz AC-voltage that the hotel grid network requires. There might also be batteries used for storing energy, and those are in DC-voltage. The mentioned components can also have different configurations required by different use cases, adding even more variety. This thesis focuses on vessels using DC based power systems and other vessels, that could be identified as electric vessels by definion are ignored. Different system configurations for these electric vessels are almost limitless. For instance, the system configuration required for vessel designated for longer travels for multiple days can be vastly different, compared to 10 minute per trip on a ferry. Upon closer examination, it becomes apparent that the main components of vessels with vastly different use cases are often constructed from the same types of components that have the same functionality, even if the specific components themselves are different. The main component options for electrical vessel are shown in Fig. 2.1. Combining factor for these components is the DC-link. By distribution system topologies, the power systems are isolated micro grids, and as such, have same operational challenges as any other micro grid. For instance, the micro grids often encounter issues with management of instantaneous active power balance. As micro grids are powered by inverter interfaced sources, that are inertia-less, voltage changes caused power variance could result in black-outs of entire micro grid. The electrical vessels usually contain 2 DC-links, 1 for portside of the vessel, and one for the starboard. These types of systems, where

constant voltage network is used as the common point for all components are called voltage link systems. It is also possible, for system to be current linked, where current sources are used instead of voltage sources as stiff points between converters in the system, but in this thesis, voltage link systems are considered as those type systems are developed at Danfoss Editron (Trzynadlowski, 2016b, Kayhn, 2012).

In this thesis, DC-link is used to describe that for example starboards dc systems form, while DC-bus is used to describe the entity DC-links from when bus tie is closed, so both starboard and portside.

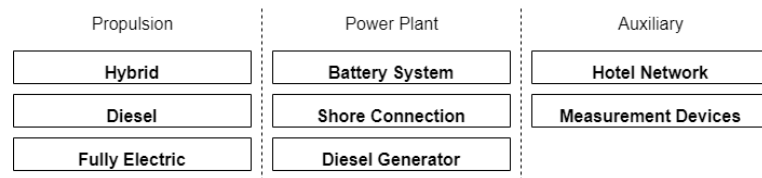


Figure 2.1. Generalized components of an electric vessel categorized. A vessel can contain multiple items from each category, depending on needs of the vessel.

Loads in a shipboard system can be categorized as propulsion, ship service and hotel or grid loads. In all cases significant portion of generation is utilized by the propulsion. But this is naturally dependent on type of ship, as for example in a passenger ship, the hotel load can be much larger, than in navy ship. (Trzynadlowski, 2016b)

All the main components of an electric vessel are connected with one or multiple DC-grids, and then power electronic converters are used to supply power in voltage and frequency that the use case requires. All the components in the vessel could either be classified as producers, or consumers, but there are exceptions, where some components, can act as both depending on situation. As all components are connected to this DC-grid, it is an essential part of a vessel functionality. In Fig. 2.2, is power system example of a typical medium voltage ship distribution. From this example, it can be noticed that vessel electrical grid is constructed from two identical entities. These entities mirror each other, and have exactly the same components, and are connected via bus tie.

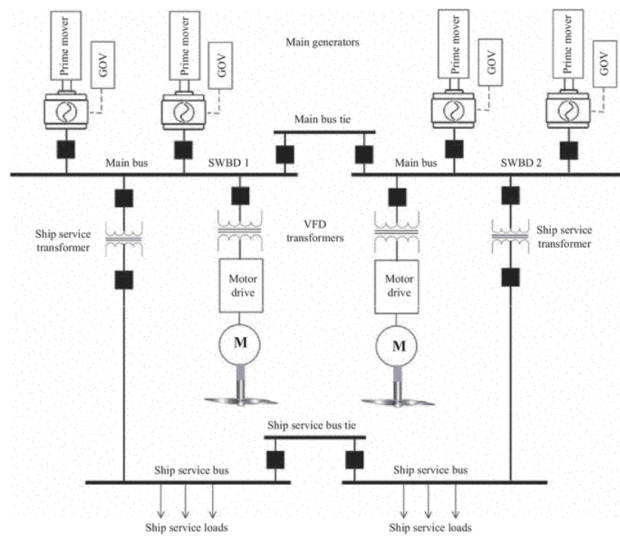


Figure 2.2. Example of a medium voltage shipboard system (Trzynadlowski, 2016b).

2.1.1 DC-Grid

The DC-grid, often referred as DC-link (or DC-bus) is what connects different parts of the vessel to each other, using DC voltage. This DC-link can have its power source from many components. DC-links voltage is a product of its connected components, and different components affect it in different ways. These components include diesel generators, grid systems, electric machines, DC-DC converters, batteries and others such as customer provided systems, depending on the configuration. Different sources and loads affect DC-links voltage behaviour and this in turn affects how the loads and generators work. While the dynamics of DC-link can be complex, the electrical components themselves results in a capacitors, inductors and resistances of the wiring. The DC-link is the central point of the electric grid, combining everything to one and the complexity comes from connected components, not the electrical circuit of it itself. Predicting how the DC-links voltage reacts in different situations, can be a complex task. From the simulation perspective, this also means that simulating DC-link properly, is a crucial part if the goal is to have accurately modelled simulation of the entire vessel.

The DC-link can be connected to another DC-link, and this is often the case in vessels. When a DC-link is connected to another DC-link, they are connected to each other with a connector, referred as bus tie. The power can flow in both directions through this connection.

This approach allows for power outages or other failures on one of the DC-links, whilst the other one keeps the vessels electrical functionality intact.

2.1.2 Power sources

Vessel can have different types of power sources, that we can classify into two groups; electric, and non-electric. In general the vessel can be powered electrically in two main ways. The energy is either stored in the battery system of the vessel or it can be connected to shore power supply, the power grid in-land. Naturally, the latter power source requires that the vessel is stationary and in port, so it cannot be used while moving. On the contrary, the energy stored in batteries is limited, and often a diesel generator is used, that then generates power to the DC-bus. There are multiple types of diesel generators and different purposes for their usage. These can be for instance just to power the grid. In this case diesel generator only exists to provide power to electrical systems, and doesn't have any mechanical load, such as propulsion. The power electronic generator can also be connected to the propeller axle of the power train that is driven by a diesel motor. The generator can be then used to generate power to the DC-bus. In such configuration where generator is connected to propulsion, the generator can also be used in the other way; using power from the DC-bus to propel the vessel in a hybrid or fully electric way.

Other means of powering the vessels electrical grid, aren't covered in this thesis, such as gas turbines or other types of power plants.

2.1.3 Loads

Loads that vessels electric grid can also have many variations, but can be divided into three categories. Either the electricity is used to power something that moves the vessel, that is, the propulsion. It can also be used to power other devices or so-called hotel grid, that provides electricity, for the entire vessel, and all its gadgets. This grid is an AC-grid, representing standard AC-grid with fixed voltage and frequency i.e. 400V 50Hz. For hotel grid, developers do not have much information upon, and it cannot really be modelled accurately, to reflect actual loads and devices connected to it. As last load type, there is ship service. This

is everything else connected to vessels DC-bus. This in practice means measurement devices and other such devices to cover various ship functionalities.

2.1.4 Fuses

While fuses are not important in simulation in terms of power consumption, and differences in load with or without them, they are integral part of the logic of the ship as tripped fuse or a failed system definitely cannot be ignored. Fuse trips are also vital to test beforehand, before the real world testing. If vessels control system does not react to fuse tripping it can cause valuable equipment to break in the system. For these reasons proper testing of fuses is valuable to extract from simulation.

2.2 Electric Converters

At Danfoss Editron, electric converters are very common components in electric vessels electrical grid. There are several use-cases, and electric converters are used as DC to DC converters, AC to DC rectifier, DC to AC inverter, with direction of power also changing depending on the needs of the system. This means that DC to AC inverter can act as active front end (AFE), for example. To develop simulation of such devices there needs to be understanding of fundamentals these are based on. In this thesis, the focus is on shipboard applications of electric converters, and thus only such applications only relevant to that scope are researched and presented.

Power electronic converter such as electric converter is a switching circuit, that uses a power semiconductor as a switch that are either on or off, switching the configuration of the circuit. Common semiconductors used, are power diode, thyristor, bipolar junction transistor, insulated-gate bipolar transistor (IGBT) and metal-oxide-semiconductor field-effect transistor (MOSFET) (Trzynadlowski, 2016a).

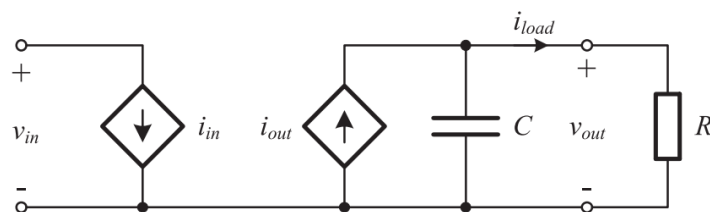
Typical applications of power electronic devices in shipboard applications are presented in Table 1. This has been derived based on (*IEEE guide for the design and application of power electronics in electrical power systems on ships*. 2009).

Table 1. Different use cases for electric converters in shipboard applications.

Acronym	Description
ACFC	AC frequency converter
AC/DC	AC rectifier
DC/AC	AC inverter fed from dc source
DC/DC	DC converter fed from dc source
AFE	Active Front End
STS	Solid-state transfer switch
STATVAR	Static reactive power (VAR) compensator
AHF	Active harmonic filter
VFD	Variable frequency drive, typically fed from fixed-frequency ac source
UPS	Typically fed from fixed-frequency ac source
HPSP	High-power ship propulsion
ES	Energy storage or pulsed power loads
Built-in power supplies	Any of the many power supplies built into other shipboard equipment

A typical 3-phase AC/DC rectifier, has 2 semiconductors per phase, totaling 6 semiconductors. Each phase has three plausible positions; high conducting, low conducting, and not conducting. This results in 9 plausible circuit configurations. Moreover, the switching frequency of rectifiers switching frequency can be in range of several kHz. Shipboard system can include over 10 electronic converters. Based on this simple analysis, It quickly comes apparent, that scale of calculation required, to simulate an electric converter at real-time is impossible, for most consumer machines.

It is possible to accelerate calculation speed by simplifying the circuit for instance by using duty-cycle based modelling (equivalent circuit) to represent switching, instead of actually switching circuit. In Fig. 2.3 an illustrative example of such equivalent circuit is shown. (Pavlovic, Bjazic, and Ban, 2013)

**Figure 2.3.** Equivalent circuit of simplified averaged converter.

Currents for averaging converter can be calculated using duty-cycle, that can be deduced depending on the control method. If load power is known, the current I_{in} can be calculated when the voltage input V_{in} is also known. As the duty-cycle D is known from control model, the load current I_{out} can be calculated as

$$I_{out} = (1 - D)I_{in} \quad (2.1)$$

In case we use DC-bus voltage as the V_{in} , and for simulation, it is not required to simulate V_{out} , that part can be left out, and only calculated the required power for load using other means. I_{in} can still be derived from power. Similar methodology can be used in all electric converter appliances, where DC-bus is used as voltage input-output. In buck boost DC-DC converters, used for example in battery systems, the studied system can be divided into two DC sides where one can describe the differences between the source and output side voltages, using duty-cycle. The duty-cycle can be defined using following equation for lossless case as follows

$$D = 1 - \frac{V_{in}}{V_{out}} \quad (2.2)$$

This kind of approximation does not reflect accurately switching events, as they are not accounted for in any manner. Simulation only reflects what loads DC-bus might have during modelled loads. Loads also need to be modelled accurately, for simulation to matter. In this thesis, the goal of the simulation tool is not simulate the voltage of DC-bus in high accuracy (i.e. 1 V accuracy), but the main target is to obtain simulation capable of represent how DC-bus voltage reacts in greater power balance changes. For those situations, this averaging calculation method is good enough.

Using averaging simulation-based modelling, the operational simulation accuracy that the project requires can be reached. The power generated or consumed, and the plausible losses can still be calculated, even without considering the dynamics of switching events. By using duty-cycle averaged simulations the usage of calculation intensive switching simulations can be avoided. And at the same time achieving real-time simulation is possible. The control

systems still need to be run at their calculation frequencies, and thus are the main focus point of the simulations. The simulations and its fundamentals are discussed in Chapter 4.

2.3 Communication with PLC

Performing simulations alone is not enough, and those simulation results need to be communicated to the PLC efficiently. Also, the reference values need to be fetched from the PLC, so there is mutual communication between the simulation tool, and the PLCs. The Beckhoff PLCs communicate normally with the inverters using CANopen interface. The CANopen is an interface that is commonly used for communication between multiple nodes, in industries such as marine, and automotive.

Beckhoff PLCs support many communication interfaces, and many of them are used during the development of vessel software. One of them is CANopen, and two other (main) ones are Automatic Device Specification (ADS) protocol and Modbus. The connections via Modbus are mainly used to communicate with 3rd party devices, while CANopen is left to internal device communication. The ADS communication protocol uses ethernet as its interface and runs on top of the TCP/IP or UDP/IP protocols. It also is relatively easy to implement into software as there is no need to map data, like in Modbus or CANopen. In Fig. 2.4, a visualization on how ADS works is depicted. The ADS software interfaces can be implemented in many programming languages, such as Python and C++ (Beckhoff, 2022).

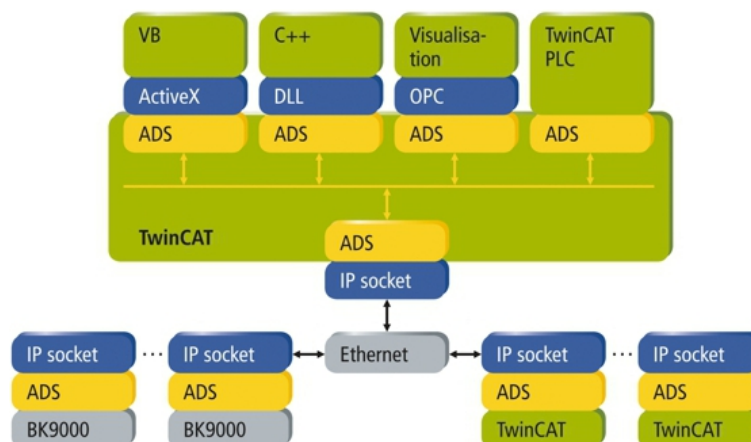


Figure 2.4. ADS protocol transport layer (Beckhoff, 2022).

The ADS protocol requires devices to be in the same network, and connected using ethernet.

This protocol uses AMSNetID to reference to other devices. The AMSNetID is based on devices MAC address, and typically MAC addresses bytes 3–6 are used to form a ADSNetID. On Windows PCs, TwinCAT creates virtual environment and MAC address, where it refers to, when running PLC software locally. A port number is used to distinguish sub-elements within device. By default, port 851, refers to PLC runtime 1, that often is the default port for running software. Then index group and index offset is used to distinguish different data objects within a sub-element. In Fig. 2.5 are the contents ADS communication visualized (Beckhoff, 2022).

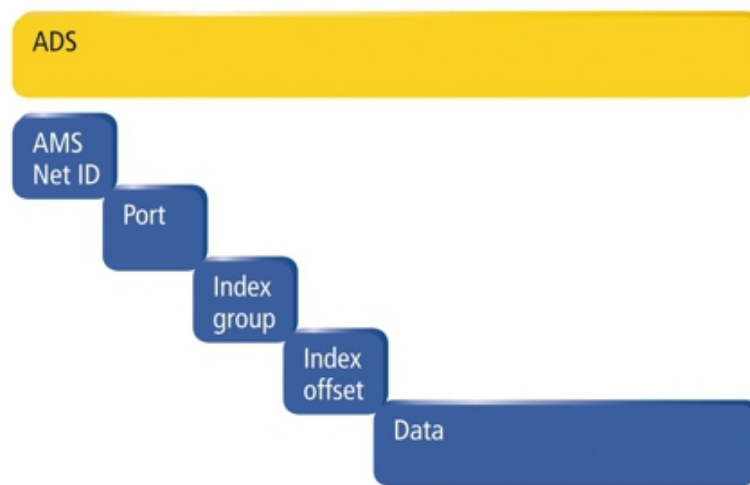


Figure 2.5. ADS communication description (Beckhoff, 2022).

The ADS protocol accesses directly to variables that the PLC uses. Whereas CANopen communication is set as device, that then writes values to its mapped locations. If ADS communication is used to write variables by 3rd parties, it needs to be accounted that values aren't written from multiple sources and cause unexpected behaviour. Difference between ADS communications variable access, and CANopen communication is shown in Fig. 2.6.

2.4 Selecting the tools for development

Selecting the correct programming language, and thus the toolset for a program, is an essential part any software project. To ensure proper tools were used in this thesis, research was conducted on what features are required from the software. The previous work (Gräsbeck, 2021) and the executed logic simulator, was made in Python. This meant that, either the work carried out during this thesis should be done in Python too, or the simulator should be

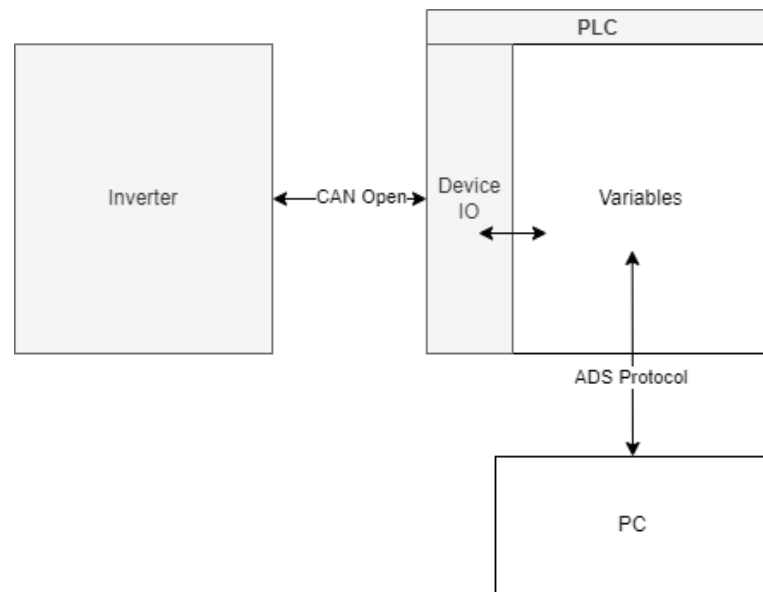


Figure 2.6. Simplified difference between CANopen communication and ADS protocol, on how they write their variables to PLC program memory.

ported to another language. Also, an interface between two simulators, where one handles the logic simulation, and another one electrical simulation, is an option. However, the commercial tools available for real-time simulation, that this research requires, are very limited. A possible solution, is MathWorks software family, that is Matlab, and Simulink. The advantage of using Simulink is that it makes it plausible to accurately model, and then run models, of complex electrical systems. Simulink has large library of accurately modelled electrical components, and capability to model new ones when required. But while using Simulink for electrical simulations is convenient, this would also require re-modelling all logic from previous simulator. There is also the practicality question about commercial simulation environments. These simulation environments often have financial costs in terms of licensing. A thing to consider too, is version control. The version control between different versions of software, could be a potential issue in the future, as not all tools are as flexible when transitioning between versions of software. The issue is not limited to commercial software with yearly major releases, such as Simulink, but other solutions too. As the developed simulator will be used for commercial use at Danfoss Editron, corporate needs to allow usage and acquisition of such licenses if required. Licensing costs increases if the end-user is required to have licenses for the software, compared to just the developer requiring the software.

A widely available tool for performing electrical simulations, is SPICE. This software also

would provide accurate simulations, and robust modelling tools, depending on distribution chosen. Spice has multiple different distributions available, as the software is open source. Multiple different front-ends exist for SPICE, e.g. Altium Designer, TopSpice and LTSpice. There is also an existing Python library for integrating SPICE called PySpice. PySpice is a module which interface Python to Ngspice and Xyce circuit simulators. This would provide easy integration to existing logic simulator as it a Python module. A proof of concept simulator was developed using PySpice as the simulator component. While electrical simulations were possible, tools available in the Python library, weren't robust and flexible enough to provide easy environment simulate the developed models with. Interacting with the models in real-time was not possible, and that is required in this case. Thus, other options were researched.

2.5 Functional Mock-Up Interface and Function Mock-Up Units

When modelling different dynamic systems and differential equations for simulation models, there are multiple different ways and tools one can use to achieve the goal. Different options have all have their compromises and there is no solution for all needs. In case of this project, performing simulations about an electric vessel, in real time, requires fast execution time. Simulation model also needs to support co-simulation. Co-simulation is a type of simulation, where different subsystems of the total system are modelled independently, and subsystems can be communicated with during the ongoing simulation. This way different subsystems can communicate with each other, and form a larger system, but being modular at the same time. Co-simulation also allows for outside intervention, and sources outside the simulation environment can also input values to the simulation. This is very much required, when communication with outside components such as PLCs is required. One option to consider is manually deducting differential equations for the simulation models, then implementing those equations in chosen environment, be that C-programming language in example. Then an interface can be built around those differential equations, allowing for integration of components outside differential equations. This option would provide all wanted features, as this approach would be flexible to the needs of the project. Mathematical accuracy would be only limited by the differential equations themselves. While in theory,

this approach would be ideal for implementing simulations, but this does not consider time needed to create and implement all said systems. Time required to accurately model an entire electric vessel, and then build a robust and flexible communication interface with PLCs is not an option that is reasonable for this project. Developing new models, if required, would be a major work load, and would require expertise knowledge and work from the developer. In future use cases for these simulation models, and frameworks built around it, it is required that models can be easily developed if required. This approach does not allow for easy development, and more time could be lost in developing new models, rather than gained by performing software simulations. Thus, other options were researched.

Functional Mockup Interface (FMI), is a free and open-source simulation format that allows for running simulation models in a more flexible environment, which is easier to develop models for. The FMIs main purpose is to define a free independent standard interface, for a container, for exchange of dynamic models, between variety of different tools. This container holds a combination of XML files, binaries and C code, and the end its distributed as a ZIP file. This would allow running simulation models, in FMI-format, to be run in Python, or other chosen environment. A component that implements FMI, is called Functional Mockup Unit (FMU). While there are different interface types, that FMI defines, in this use case, we will be focusing on co-simulation. The co-simulation models, are a type of interface, where the FMU contains a solver, or a scheduler for calculations. So with proper tools, simulation models, that also include accurate and fast solvers. (*Functional Mock-up Interface Specification 2022*)

FMIs are flexible and allow development of models, that can be then used almost regardless, of what is the chosen environment for final simulation. This means that models won't be proprietary to just for use in this specific simulation use case, lowering the risks associated with development in a major way. The models developed for this simulator could also be used for other Simulink simulation projects at the company. Using a specific simulation tool like Spice, for only this one use case, could result in models being useless in a case where simulation tools development does not go as planned.

2.6 Impact of Programming Languages On Runtime Performance

While possibilities in implementing the models themselves are important, the actual runtime performance of the selected language used in the framework to be built around the simulation models is also very important. As the code need to be running in real-time, the performance of said code needs to be up to that task. While Python has the tools to do almost everything, it also rather slow in runtime performance, when compared to lower level languages, like C or Rust. But learning a new language, like the Rust, can be a time-consuming project, even though it could provide better end results. Also, to be noted, in real-time simulation, it doesn't provide much additional benefit, to run the program faster than real-time. Since its time bound, it only needs to run at real-time and rest of the time is spent waiting. While that waiting time could be spent doing something else, in the case of this simulator, running in real-time, is good enough. Even if the simulator could run much faster, it does not actually benefit the end-user. So if the simulator can accomplish real-time simulation, it can be said to be good enough. Faster than real-time simulation could be used to simulate vessel dynamics for longer periods of time, such as weeks or months, but for this thesis this possibility for such simulations is not taken into account.

2.7 Threading and Multiprocessing

Modern computer processors are compiled of multiple processor cores and threads. Operating system handles the allocation of those said cores, and threads. Having multiple cores and threads allows for multithreading and multiprocessing, making it possible to run multiple programs simultaneously. How fast a program is completed, is usually bottlenecked by one of two things. It is either input-output limited (IO), meaning that most of the time that running the program, is spent on waiting different inputs or outputs. Or it can be calculating bound. In this case, most time is spent, actually calculating and executing the program. These calculation heavy programs, can be split up into smaller pieces, and those smaller pieces, can be running simultaneously. This can reduce runtime dramatically. While in ideal case, splitting up one process to 2 processor cores, equals double performance, this in reality often is not the case, as more variables affect how programs work. But in some unique cases,

the programs can gain performance equal to amount of cores they are allocated. In a project, where runtime is important, looking to minimize the runtime, and at the same time maximize the performance is key. So finding a way to utilize this technology can be key to success.

Multi-threading, by definition means that program can run on multiple threads, but this does not require, that it runs on multiple threads at the same time. Python as a programming language, is multithreaded. But by design, only one Python process can be executed at once. A process is a running program. This means that even though, the code can be run on different cores and threads, only one is executing at a time (Hillar, 2009).

This means, that if one wants to split simulation in to smaller pieces to utilize resources, base-line Python doesn't improve performance, as those smaller pieces aren't run simultaneously. Python allows for multithreading and multiprocessing as a separate library. Multithreading library does provide programmer the ability to allocate code to different physical processor threads, but it still can't run multiple code entities simultaneously. This is caused by Python still wrapping everything inside one process instance, and it was by design, only one can be running at once. Where a process is an independent program, thread is only a part of a program. The multi-threading does still allow for safe communication between different threads, as they remain in same context. The other library, multiprocessing, does allow for simultaneous running of Python code. Multiprocessing, creates new process instance, so those instances can be running simultaneously in parallel. But, this comes with a caveat, that communication between those processes can be slow, due to process instances not having access to same memory. This means, that one can make the program IO-limited by using multiprocessing, when most of the time is spent communicating between different processes, instead of actually computing. So while multiprocessing unlocks more computing power, it might not be the correct choice for every use case, and you can actually make your program slower by using multiprocessing, compared to multithreading, that still allows for faster communication. Multithreading allows for concurrency in running Python code, but multiprocessing allows for true parallelism, as shown Fig. 2.7. (Fernando, 2015)

To determine if a program will benefit from multiprocessing, compared to just multithreading, it's needed to determine what is the limiting factor of the program. In this case, different simulation models will need to communicate with each other, but main time consumer is go-

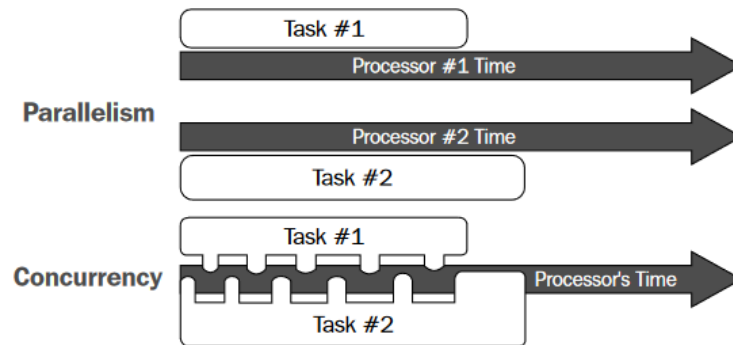


Figure 2.7. Comparison between parallelism and concurrency (Fernando, 2015).

ing to be the simulation and its differential equations. Amount of components in simulation also will affect the gains available, and while in smaller simulations, it might be faster to run them in same process, in larger simulations, multiprocessing will definitely be an advantage as the component count grows. Multiprocessing approach also means user doesn't have to think about the amount of the components in the vessel limiting the performance as long as there are enough processing cores. Multithreading is a better solution when it is not required to achieve true parallelism, but as the requirement is to run multiple simulation processes at the same time, multiprocessing is required to achieve it.

Python multiprocessing library comes with multiple tools to provide the developer easy to implement multiprocessing and multithreading in their programs. One of these tools is concurrent futures. This tool provides developer a scheduler, that automatically allocates functions to multiple processes. These are then concurrently executed and called futures. While tools like this are fast to implement, they do not provide necessary control for user to control what resources are allocated to processes, and when those processes are executed. This could bring unwanted inconsistency to the simulation tool, and for that reason, it was decided that processes would be formed manually. Also processes need to be persistent, where building own scheduler and simulation processes using Python library multiprocessing is advantageous.

3 Programming of the simulation tool

In this chapter, the structure of the simulation tool is described, along with the tools chosen for the development. For this thesis, some material was provided by Danfoss Editron. These were the existing PLC connection simulator, and internal simulations executing averaging simulations and control systems of the inverters. They act as the base for the simulator to be built upon. Both the PLC connection simulator and the averaging simulations are made for internal usage, and are not available to public use.

3.1 Simulation Environment Description

The environment where the simulator will be used is shown in Fig. 3.1.

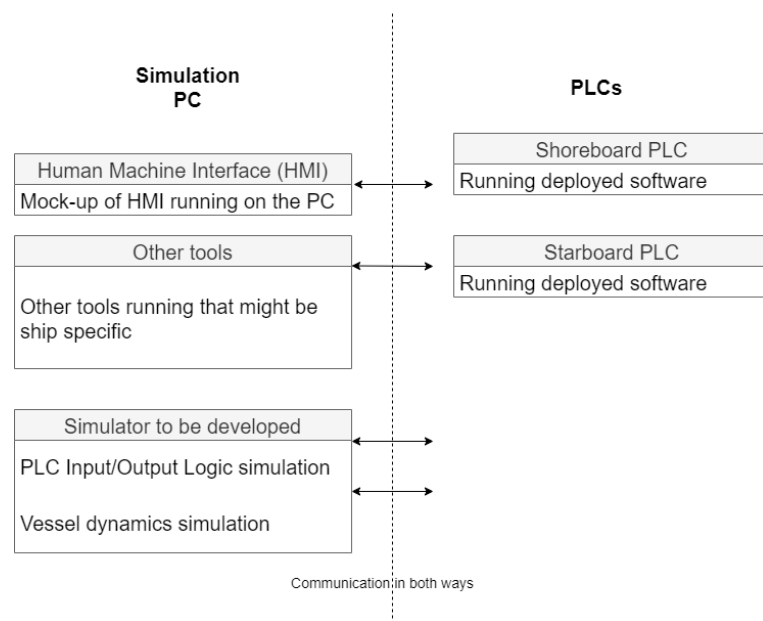


Figure 3.1. Environment where the simulator will be working in. The simulator and all tools, communicate with both PLCs. The simulator that is to be developed will act as input and output simulation for both PLCs..

When simulating a vessel, different tools are used to handle different parts of the vessel. The target of this thesis work is to expand the functionality of the dynamics simulations in the block logic and dynamics simulations. The communication between tools is not necessary, as all tools act as independent entities. Whilst communication between simulators is not required, the simulator developed must not break the functionality of other tools.

3.2 Logic Simulator

A logic simulator for internal use was developed for Danfoss Editron as a thesis work (Gräsbeck, 2021). This simulator was developed entirely in Python, and its main feature is to simulate the interface between the PLC and the vessel. It simulates inputs and the outputs of the PLC, and where dynamics of the vessel are required, simplified simulations are done to represent that part of the vessel. Using this simulator, vessel can be tested for the majority of its functionalities. But what it lacks are the dynamics simulations, and the electrical simulations. Initially the research was done on the source code on how to implement it to the simulator that was to be developed.

3.3 Vessel Simulations

To develop simulations, first a platform for those simulations needs to be chosen. As mentioned in Chapter 2, the main options considered for this simulation tool are Simulink, Spice and FMU-simulations. The selected option would also largely decide the programming language.

Python has a library for performing FMU co-simulations. A proof of concept simulator was then developed using FMPy library for Python. If the performance, of this simulator is good enough, Python could be chosen as the main platform for the program. If it were noticed, that the simulation can't be run in real-time using Python as a language, other options would be researched, that offer faster performance. Options such as using Rust, with Rust-FMI library is an option that offers more performance if required.

With the results from the proof of concept simulator, FMI was chosen as the simulation model format. The proof of concept simulator using Python, was able to achieve run-time for simulation cycle shorter than its selected sampling time. This would allow communication with PLC without it affecting dynamics simulation runtime negatively. Previous experience, great library support, and the fact that existing logic simulator is developed in Python, were the reasons for choosing Python as the development language for the project. For chosen development tool, for the FMUs was MathWorks Matlab Simulink. This was done, for two

main reasons; previous experience with the tool, and previous work done at Danfoss Editron to develop simulations. Simulink also has full support for FMU Co-Simulations export, providing all required features required from the development tool.

Predicting how vessels electrical systems work over longer periods of time (e.g. range from minutes to hours, rather than under a second) is valuable information. As previously concluded in Chapter 2 simulating longer run times using switching simulations is not feasible, as the simulation targets real-time operation. Thus, for this reason the averaging simulations have been implemented for these simulations. Simulation models, implementing averaging simulations, and components on basic level, of several vessels were provided for this thesis by Danfoss Editron, to aid the development of the simulator. The models are done in Math-Works Matlab Simulink. The averaging simulations include almost the whole vessel, from simple diesel generators to batteries and generalized load models. Research on how these models work, and how they could be improved and included in to the simulator was done. Simulation models required, performing simulations and building an interface between them and logic simulator, are to be developed in this thesis from ground up. Developing simulation models themselves is explored in Chapter 4.

3.4 Software Requirements

As a first step towards developing the simulator, a list of requirement were formed. The requirements also include the requirements for the existing logic simulator (Section 3.2), as this simulator will be build upon those foundations. New simulator is only to add functionality, not reduce it. The requirements are provided in Table 2

Other features were discussed, but many of them were chosen to be out of scope. Features such as focus on energy efficiency calculations, and comparison tools between two different configurations where left out to keep to scope of the project more manageable. Language for the tool was not specified in the requirements, and could be chosen freely.

Table 2. Requirements for the software development project.

Category	General Requirements
	<p>Simulator is to implement all features previously included in logic simulator.</p> <p>Simulator is to provide dynamics simulation for vessels electrical systems, with main focus on DC-network, and its connected devices.</p> <p>Dynamics to be able to be simulated separate from logic, as a separate application.</p> <p>Logic simulator to be also runnable as standalone, without electrical dynamics simulation.</p>
Category	Functional Requirements
	<p>Simulator is to be modular in its structure, to allow different configurations. There can be different amount of different components, and components can be changed freely.</p> <p>Configurations for the simulator to be read as JSON-files.</p> <p>In electric dynamics, software structure to be modular, where simulation blocks can be connected to each other.</p> <p>Simulator must have a user-friendly interface, either a graphical user interface (GUI) or a command-line interface (CLI).</p>
Category	Performance Requirements
	<p>Simulator must run in real-time and be able to communicate with the PLC at the same time.</p>
Category	Simulation Requirements
	<p>Simulator control systems must react realistically to the reference values it gets from PLCs.</p> <p>Simulation blocks must be configurable to suit vessels needs.</p> <p>Targeted simulation accuracy for simulation is operational.</p> <p>Simulation blocks to be included:</p> <ol style="list-style-type: none"> 1. Battery 2. Diesel generator 3. Propulsion 4. Hotel grid 5. DC-link 6. Bus tie

3.5 Design and Structure of the Simulation Tool

As the electrical vessels are created with great variety between each one, the simulation environment needs to be flexible. Adding new diesel engines, changing loads, changing different battery models, everything needs to be flexible, and configurable for the end user. Components should be easily swapped out, and added in to vessel, and it should not require any additional modelling to configure new vessel. End product should be easy to use package, that will handle everything from simulation, to multiprocessing. Finalized simulation tool will be capable to complete both logic and dynamics simulations. In Fig. 3.2, the generalized structure of the whole simulation program is shown.

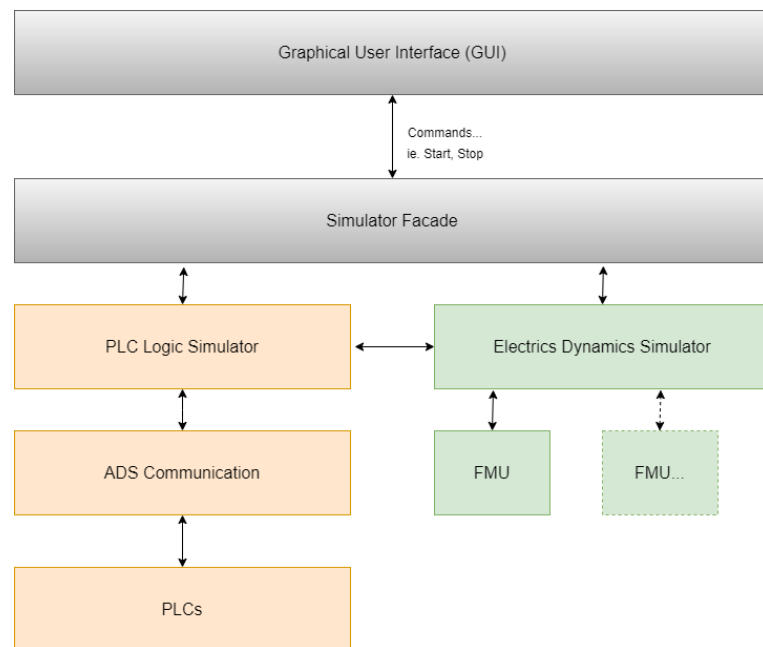


Figure 3.2. Generalized block diagram for the simulation tool. Different simulation commands, and other information are shown to user in GUI, simulators communicate to each other when necessary. i.e. simulation results or instructions.

Simulation tool itself, will be designed and executed with object-oriented programming approach. This was chosen, instead more procedural approach, as the simulation tool is to be modular, with configurable and interchangeable simulation blocks. This makes it more fitting choice for this task. Target operating system for the tool will be Windows 11, as it's the main operating system used at Danfoss Editron. While performance of the tool is important, it is affected by the hardware its ran on. So it is important, that some baseline specifications will be set. It is unrealistic for the program to run at real-time speed on every machine avail-

able. This is because speed of the program usually comes at the cost of reduced accuracy of the simulation. But it is also important not to set requirements too high, since most developers do not have access to the latest and greatest computers, provided by the company. There needs to be a good middle ground between performance and availability, so it is chosen as the minimum requirement for the simulator to be running on at real-time speeds.

Chosen programming language for the simulation tool was Python initially. Other options considered include Rust, C and C++, and remain as fallback options if Python would be deemed not capable for acting as platform for simulations. Python was chosen, for its robust library support, and good enough real-time performance in proof of concept Simulations. Also, previous experience with the programming language, would mean most efficient development cycle for the tool, as no new simulation languages would be needed to be learnt. Simulations will be developed in MathWorks Simulink, then to be exported as Functional Mockup Unit (FMU).

As previously pointed out, the multiprocessing gives the biggest performance gains, when communication between the processes is reduced. Also, different parts of the program, have different sampling frequencies and thus simulation step sizes. Initially scheduler solution was considered as an option. Different simulation blocks would simulate to a given end time, with a step size of the largest step size of a block, then results of that simulation blocks would be given to other blocks. While this method works, it means that all simulations need to communicate with each other, on every cycle, and faster simulations will wait for slower simulations, meaning they are not computationally limited, but instead limited by inputs and outputs. The communication between the PLC, and simulations is done once every 10 milliseconds. So to be running at real-time, once every 10 milliseconds, simulation should advance 10 milliseconds. But simulation can be calculated much faster, and rest of the time can be spent just waiting for PLC inputs. But simulations speed must be limited by something, as for it to stay in sync with real-time. In Fig. 3.3 is cyclic nature of the program visualized.

This does mean, that all components are run separate from each other, and do not form a large electrical system, as they would when doing a regular system-wide simulation. It is to be compared to more traditional simulation method, to see if the electrical simulation

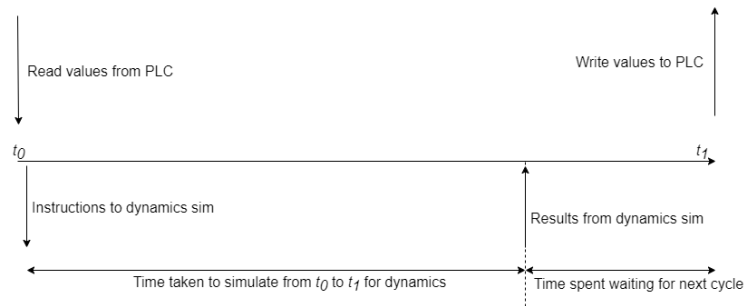


Figure 3.3. Timeline of an average simulation cycle. The simulator simulates faster than real-time, this means that some time is spent waiting for next simulation cycle.

accuracy, with this topology is good enough, or is more traditional simulation method required. Advantages of the approach considered in this thesis are clear. Since the electrical circuits are smaller, and less complex, simulations are faster to calculate. This way faster sampling frequencies are possible, and simulation can be more accurate in this way, as large step sizes can cause simulation to be unstable, and not accurate. As the sampling frequency used directly affects time needed to perform the simulation. Disadvantage is that, the electrical circuit is not complete as for example diesel generator isn't directly connected to battery like in traditional circuit. Instead, an equivalent DC-link is used to represent the rest of the circuit. Accuracy of the simulation might be affected and the dynamics of the system may not represent reality as closely. It is also worth noting, that communication frequency of the blocks could be adjusted, so instead of using communication interval of 10 milliseconds, a shorter interval may be used, for example 5 milliseconds. Effects of changing the communication interval need to be researched and tested. Notably, large and fast voltage swings rarely happen on vessels, where voltage changes much rapidly. This is due to the large capacitance of the circuit, commonly around $5 - 20\mu\text{F}$, used in the circuit, resulting in voltage fluxuating as much in short time periods (10 milliseconds).

Simulator can be separated in 3 main parts, as shown in Fig. 3.4. There are several benefits to keeping the graphical user interface (GUI) and simulations in separate processes. For example, the GUI can update and be responsive without taking run time away from the simulator processes. Additionally, heavy calculation work is hidden from the GUI, which makes it more responsive for the user. Keeping the logic simulations and electrical simulations in separate processes also avoids slowdowns. There are also technical reasons for keeping the GUI and simulations in separate processes. For instance, communication timeouts between

the PLC and the logic simulations could affect cycle lengths if they are not kept in separate processes.

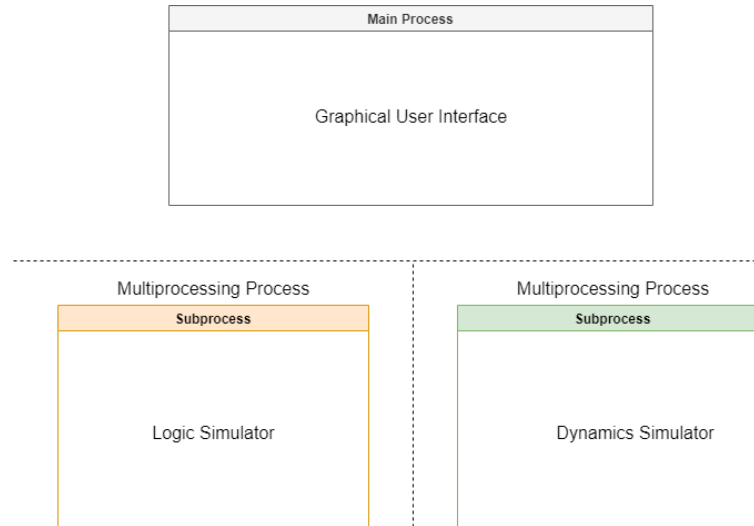


Figure 3.4. Division of processes. Processes communicate with each other using multiprocessing queue.

While having multiple processes gives flexibility, and has other advantages, it does have its disadvantages as well. Implementing multithreading is more work, and more critically, the communication between processes is not as simple, as without it. In Python, you cannot access variables from other process directly. Instead, it is required to send variables from process to another. There are multiple different methods, that Python as a programming language offers, to communicate between different processes. There is also an option to use 3rd party options such as ZeroMQ. The ZeroMQ is explored more in Chapter 3.5.6. As all running processes are running Python programs, naturally a Python specific solution provides an easy implementation solution, and not much overhead, when compared to third party solutions. Python multiprocessing library provides few approaches, for how developer can construct communication between processes. Main methods are multiprocessing values, where value is accessible by multiple processes, and threading safety is provided by mutex locking, so only one process can access the data at the same time. Using multiprocessing values does not work for this use case, as flexibility of the system is key target, and it is also not required for all processes to access the same data. Defining all values, and then planning communication of them requires effort and does not scale as well as other options. One such option is to use data transfer, instead of just providing access to all processes to the same data. Multiprocessing library provides queues, and pipes, for these use cases. Both solutions

transfer the data over from one process to another. Both solutions offer similar feature set, but as the name suggests, queue provides ability to queue multiple data objects at the same time, and the developer does not need to account for, if the data has been processed in the other process yet. (P. S. F. Website, 2022)

Queues from multiprocessing library were chosen as the communication method between the processes. Interface was defined for those queues and communication between processes can be seen in Fig. 3.4.

3.5.1 Logic Simulator part for simulation tool

Entity responsible for plc communication, and simulating its inputs and outputs, is called logic simulator. Simulator was developed in (Gräsbeck, 2021), and source code for this simulator was provided for this thesis. It uses ADS as the communication protocol between Beckhoff PLC and PC.

Simulator is functional, but as the scope of new simulator grew, it was noticed that major changes were needed to be done to the logic simulator, for it to fit the new simulator environment. The simulator was not configurable enough, and the configuration was not as easy as it was wanted for the end user. As the logic simulator was already developed in a modular manner with separate classes, complete full re-programming was not required, but just update on all classes. Majority of the source code could be either recycled or slightly altered.

In common use cases, CANopen is used at Danfoss Editron to communicate between inverters and PLCs, like mentioned in Chapter 2.3. ADS is used by TwinCAT to communicate with the PLC. ADS is not used in the final PLC projects in vessels for communication with any components. This would mean that ideal communication method with the PLCs would be communication through CANopen. There are both digital and analog ports on PLC. Those would still need to be communicated through ADS, but the inverter interfacing could be executed through CANopen. That would be the optimal solution, but ADS communication was settled on to keep the focusing point of the work on dynamics simulation.

Inheritance scheme for logic simulation was implemented in the simulator update. A base class that provides the classes PLC communication capabilities was developed. General structure of the final logic simulator can be seen in Fig. 3.5. Library used for communication with the Beckhoff PLCs was pyads.

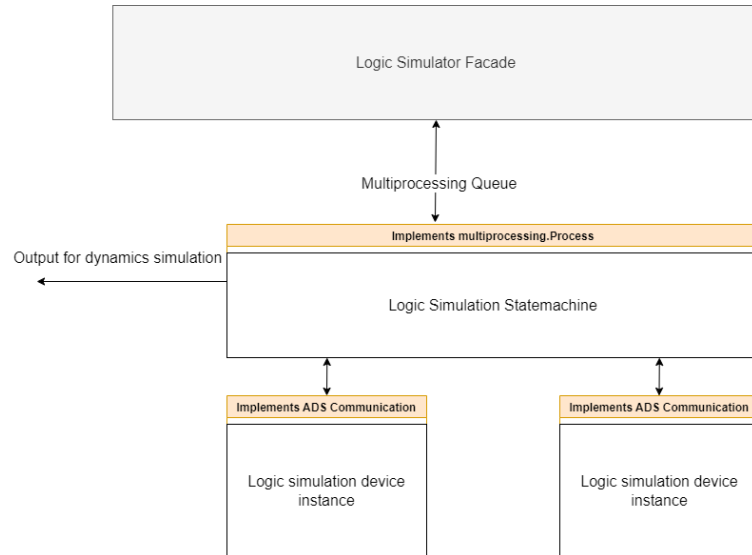


Figure 3.5. Logic simulator block diagram. Amount of simulation device instances are not limited, and they are managed by simulation statemachine. Each device instance implements ADS-communication to handle communication with the PLC.

3.5.2 Vessel Simulator Structure

Dynamics simulations are the heaviest part of the simulator, and here precautions were done on every possible step to make sure that the program runs in real-time. As mentioned in Chapter 2, main options considered for performing simulations were Spice, Simulink and FMU-simulations. From those options FMU-simulations were chosen as the platform for simulations.

As the simulation models themselves run in native C-language code, as an FMU, code that is running on Python should be kept as minimum to achieve the best performance. The structure of the simulator is similar to logic simulator. Base class supplies a singular simulation block FMU support and data processing via inheritance. That is then implemented in different classes, such as diesel generator in example. Different class implementations are required because every simulation has different inputs and outputs, and it allows for changes based on needs specific simulations.

Major difference when compared to logic simulator is implementation of Python multiprocessing, in every simulation block. This is done to gain major simulation speed advantages, by dividing simulation load from one core, to multiple as explained in Chapter 2.7. Generalized block diagram of electric simulator is shown in Fig. 3.6. The simulation blocks receive their instructions from a simulation manager, that is responsible for giving commands to the simulations. Simulation manager gives a command to simulate to a certain point in time from the starting time t_0 . Simulation steps can only be taken forwards in time. Every simulation process contains a FMPy implementation of a specific simulation, in example, diesel generator. The simulation manager also handles the communication between the simulations, and handles values such as dc link voltage to its destination address. The connections between simulations are specified in a separate json-file.

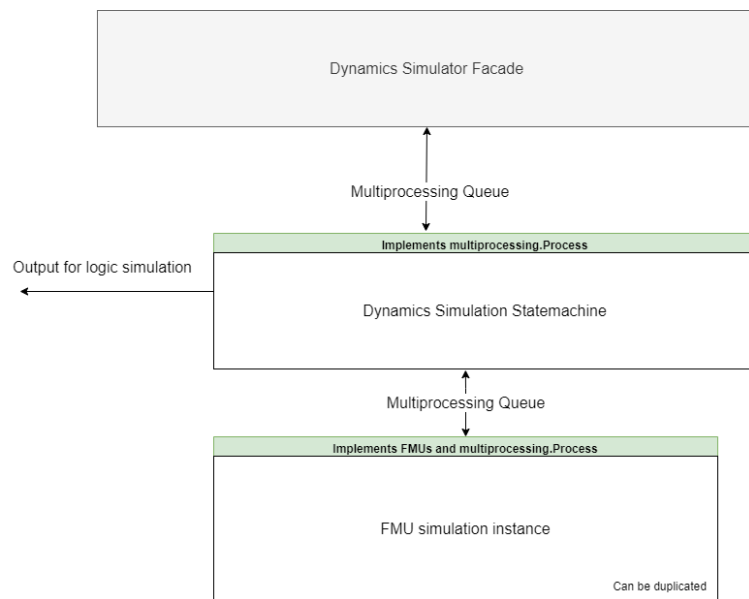


Figure 3.6. Dynamics simulation block diagram. There can be multiple FMU simulation instance, in which case dynamics simulation statemachine handles the communication between different FMU instances..

Electric simulator communicates with logic simulator once every $t_s = 0.01$ seconds as that is the cycle length on the PLCs. The simulator works with a principle, where you give it a time when it should simulate, and the simulator simulates to that time. This approach allows different simulation blocks have different sampling times. More specific simulation timeline is shown in Fig. 3.7.

As the simulator is to run in real-time, it needs to be synchronized to real time. The simulator takes in the internal simulation time, that it is to be simulated to, t_{end} . Simplest possible way

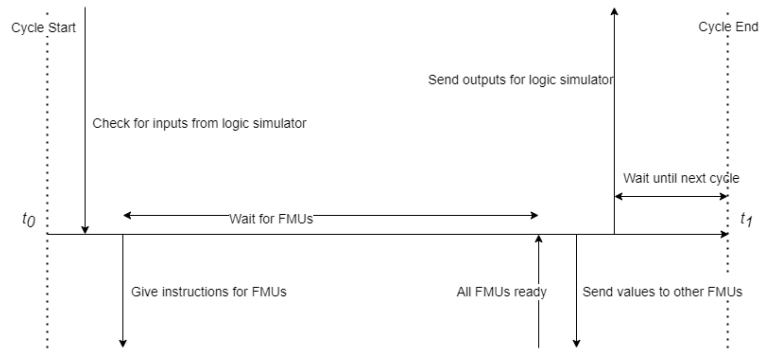


Figure 3.7. Internal timeline of dynamics simulation. The base timeline represents simulation manager. FMUs do the time from t_0 to t_1 using internal step size. After simulation is finished, required values are sent to logic simulator, where they are sent to PLCs..

is to measure time before simulation starts t_0 , then give simulator instructions to simulate a step of t_s . Measure time t_1 again after the simulation. Then to calculate the delta between the starting and ending points two Δt , and compare it to t_s . If Δt was larger than t_s , we use wait the remaining time until t_s has passed in total, for one simulation cycle. While this approach certainly can work, it has some fallbacks. For example, what happens if the simulation cannot reach wanted t_s . The simulator runs slower than real time. Slowdowns during singular simulation cycles can happen, when for example operating system does not allocate enough resources to the simulator. Simulators internal simulation time can be synchronized to computers real-time clock. Instead of giving simulator always simulation steps of t_s , the simulator can be given a variable stepsize, that has offset of simulator starting time t_{start} . With this approach, if the simulator will always take steps with size of t_s , but in case of slowdown, t_s is changed dynamically, and simulator tries to catch up the real time clock. The simulator is then simulating faster than real time until it catches the real-time clock (RTC). In Fig. 3.8, is the two methods pictured and how they act in case of slowdown.

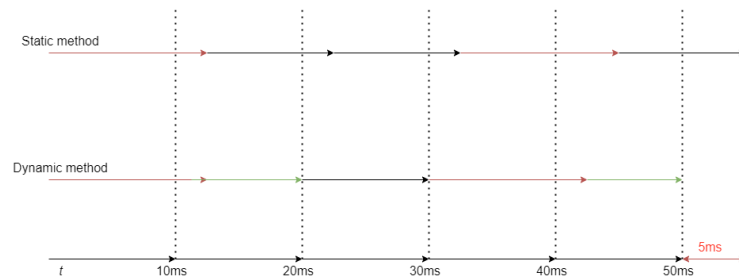


Figure 3.8. Dynamic step size calculation visualization. Each arrow represents one simulation cycle. Red arrow means, simulation took too longer than the configured step size 10ms. Green arrows on dynamic method indicate dynamic step size was used to catch up to real time..

In a case of major slowdowns, approach of dynamic t_s will result in a complete hang of software, since calculation step will grow constantly, to infinity. Where constant t_s would result in just slow responsiveness of the software and the dynamics. Both methods have their advantages and disadvantages, so both are left in to the program, and the user can select what method is to be used for step calculation. But in setups and situations where it's possible to calculate simulations faster than real-time, other method is superior for its synchronization capabilities.

3.5.3 Graphical User Interface

Graphical user interface (GUI), was developed using Qt library PySide6. The PySide6 is the open source version of PyQt6, that is under a commercial license. Qt is a powerful tool for creating complex user interfaces. Qt is and has been used in a good deal of commercial software, in applications by companies such as Peugeot, LG, Nokia and ABB (QT, 2022).

The graphical user interface was developed more with functionality, rather than user accessibility in mind. As the simulator will be running mostly on background, with the UI hidden, UI needs to be functional, but it is not prioritized. The GUI is to have a separate configuration view, and then a control panel, where user can see status of the simulators and other information, in addition to controlling the simulations.

Like other simulations, modular programming structure can be utilized. As there can be basically unlimited amount of different vessel configurations, at least theoretically, the user interface needs to be able to show all essential information about running simulations, no matter the scale of the simulations. All essential information that is required for the GUI are listed in Table 3.

When the simulator program itself is first initialized, its configuration phase is first prompted. There user is asked to configure essential information about the simulation user wants to run. For simulation to be run, needs following information about the simulation to be inserted. The PLC ADS addresses and ports, and device simulations to be initialized under those PLCs. Dynamics simulation of FMU instances, and then information on how to link those variables back to the PLC device instances. Target communication step also needs to be

Table 3. Requirements for the simulator program, and its components.

Category	General user interface
	Simulation controls must include. Start, Stop, Shut-down, Save Simulation Results See current status of simulations, and current internal simulation time Notify about errors during simulation, or during configuration phase
Category	Setup
	Select json-config files for both internal simulators, and connections between them Select running mode of the program from following: Run only Electrical simulations, Run only logic simulator, Run both simultaneously. Select target communication step for PLC ADS communication
Category	Electrics Simulator
	Separated control windows for each FMU Ability to configure initial values for FMU instances Ability to adjust values during simulation for each FMU All values for inputs and outputs listed for each FMU
Category	Logic Simulator
	Connection information for PLCs; Connected, Disconnected, Connected but with errors Current connection status Ability to execute different commands. In example: execute a fuse trip

configured, but as default it is set to 10 milliseconds.

3.5.4 Configurability

The developed simulation tool is to be largely configurable. Everything from used internal tools to choosing what simulation models are used for specific devices, and then the parameters for those simulations, are to be user configurable. A common file format used to configure different software is JSON. XML was also considered to be a candidate for file format, but the JSON was chosen because future users of the tool are most likely to be familiar with that format. As the functionality of the simulator is split up into multiple parts; logic simulation and dynamics simulations, are also the configurations split up into multiple files. The configurations for dynamics simulation is kept separate from logic simulation.

```

1  "plc_sb":{
2      "Devices":[
3          {
4              "DeviceName":
5                  "Diesel Generator SB",
6              "DeviceType": "DieselGen",
7              "Parameters": {
8                  "Nominal Power": 10e3,
9                  "Nominal RPM": 2500
10             }
11         }
12     ]
13 }

```

Listing 1. Example JSON structure for configuring logic simulations. Setups starboard PLC with one diesel generator as its device.

While this complicates configuring process, keeping the configurations separate, also helps in a situation, where only one simulator is to be used. A common situation is that only logic simulation is required, so also needing to configure dynamics is not efficient time usage.

On program startup, user is prompted for configuration files for different parts of the simulation. An example JSON is shown in Listing 1. The simulation tool automatically generates required connections between the dynamics simulator and logic simulator. But these can be configured manually through a JSON file.

3.5.5 Outside Communication

Simulator needs inputs from the PLC and a way to output its result back. Main method chosen for communication with the PLC was ADS. ADS can be used to read and write variables directly, into the PLCs data structures. This does not require any changes on the PLC source code, except disabling inputs and outputs definitions, so values from ADS and its inputs and outputs are not contradicting with each other. The simulator uses PyADS-library for Python, for ADS communication. ADS communication is essentially replacing inputs and outputs for the PLC.

PyADS is a Python library that uses C-API provided by Beckhoff, and implements it to

Python environment. With this library, it is possible to execute communication with the PLCs in a more simple way. In this project, the communication was developed into its own class, that then could be implemented in each device. So logic simulations for devices can also handle the communication. (Lehmann, 2015)

But not all devices use hardware inputs and outputs to communicate with the PLCs. For example, as Danfoss Editron does not directly provide battery systems to customers, as mentioned in Chapter 4.2.2. The battery systems instead might use Modbus for communicating with PLC. The Modbus integration is not in scope of this thesis, but as future development point, this might be considered.

3.5.6 Communication with other software

There is also possibility to account for, that other tools want to extract information from simulation in the future. For this a communication protocol like ZeroMQ (also spelled ZMQ, ØMQ) can be used. ZMQ is a networking library that gives the ability to carry atomic messages for various transports, such as connection of two separate programs. This would allow for example, connecting the simulator to existing company tools. ZMQ is a library that is supported in many languages, including Python (Z. Website, 2022).

A communication interface was developed using ZMQ as the interface. Sockets are for network programming in ZMQ. ZMQ offers different socket types for developer to develop any arbitrary messaging pattern. While ZMQ supports multiple messaging patterns, the selected approach for this software was request (REQ) – reply (REP) pattern. In this pattern, there is only one server, but there can be multiple clients. The simulation tool was set up as server. In the request – reply scheme, there must always be a request message, and then a reply. Sending a request without a reply causes an error. (Hintjens, 2013)

This communication pattern is shown in Fig. 3.9. As, other testing tools want information from the simulation tool, was simulation tool setup as the replier (or REP) of the interface. A client was also developed, with REQ implemented, to act as a testing tool, and aid future development. It also acts as a proof of concept, that simulation tool can in fact communicate over ZMQ interface.

In practical use, client requests a value for variable, such as battery voltage, from the server, and server then replies with the requested variables. The communication uses serialized JSON as the data format, this way requesting and then replying multiple variables is easy.

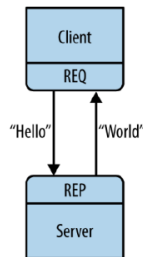


Figure 3.9. ZeroMQ request – reply pattern. When client sends a message (request) with contents of “Hello”, replies with a message (reply) with the content “World” (Hintjens, 2013).

As there are multiple possible communication paths, the communication is executed in modular structure, where it required. The communications class can be inherited, or implemented when required. When all communication methods use unified interface, the communication blocks can be easily interchanged.

4 Electrical Simulations

In this chapter, goals for developing simulation models is explored, and simulation models are developed, to provide operational accuracy for simulating electric vessel.

Electrical simulations can be computationally heavy, especially simulations switching power electronics devices. Aim of this thesis isn't to produce the most advanced and accurate simulation tool. For most use cases for the developed simulation tool, operational accuracy is enough for wanted results. This means accuracy can be sacrificed over speed. Achieving real-time simulation speed is more important, for the end-product, than developing accurate simulation data, as simulation models can be iterated over, on the existing simulation framework. But, if simulations don't reflect reality, data obtained from simulation doesn't have any value. So, balance between accuracy and speed must be found. This is the main goal for developing simulation models for this project. To be more specific, accurate enough foundations to build simulations on, in the future.

Like previously mentioned in Chapter 2, vessel includes many switching circuits, that are calculationally heavy to simulate. An alternative calculation method must be used that can be calculated at real time speeds, and such solution is using averaging simulations, where switching events are simplified to duty-cycle based calculations. As means to speed up the simulation time and make the simulation system more flexible, a modular simulation structure was chosen for the simulation. Simulation system constructs of several simulation blocks, that are then combined using the simulation framework developed in Chapter 3. This enables quick prototyping environment for developers, while still providing simulation results, at the cost of simulation accuracy. As there is no need to develop new simulation models specific to each vessel, can simulations be configured and executed faster. Connections between simulation blocks will be combatted described in Section 4.3 in more detail.

4.1 Digital Twin Classification

A digital twin is a virtual representation of some physical object. The difference between a digital twin, and a simulation is often the scale, and the reference of the real world object.

Simulations, are often based on imaginary components and those then aren't executed in real world. Digital twins by definition are based on real world objects, and dynamics are running in real-time, as in this case. And with real world counter-part, that real world information can be applied to further develop its digital twin. With this feedback loop model, the accuracy of the digital twin rises, improvements can be generated and then applied back to the real world. Digital twins also are often larger in scope and scale than just simulations (IBM, 2022).

Simulations developed here from a digital twin of sorts of an electric vessel, and the data from real world will be used to develop models even further. But the aim of this project is to provide more generalized set of components, that can be used in all kinds of different types of vessels, and the aim is not just one. Models will represent multiple different types of vessels, and the aim is not to develop specific models for every vessel, instead models will be configurable to suit the needs of multiple use cases. So by this definition, system developed might not be classifiable as a digital twin.

4.2 Simulation Models

Simulation models are to be developed independently, making it possible to have multiple different diesel engines, loads for example. Also changing in different variations of models is easier, when the models themselves are modular. Modules are then connected with the simulation framework built in Chapter 3, making it one entity. Solver for the simulations are provided by the development tools. MathWorks Matlab includes export tools that allow for the developed models to be exported as FMUs that includes solver for the model.

As models share some parts with other Danfoss made projects, all details about the simulator can not be shared publicly. For example the structure of the PID-controller models cannot be shared, and the values used in them.

4.2.1 Diesel Engine

Diesel engine model contains the diesel engine governor, and the generator. Whilst simulation of diesel engines is important, since they are controlled by the PLC, diesel engines to

vessels are provided by different manufacturer, and are sourced by the client of the vessel. As diesel engines are proprietary technology, models for those engines are not widely available. The diesel engines also vary between vessels, so accurately modelling a diesel engine is out of the scope for this project. A generalized diesel engine model will be used, in conjunction with diesel engine generator included in modelling tool Simulink Simscape. Accuracy of this general model will be considered to be accurate enough, as the dynamics of the diesel engine aren't critical when the focus is on simulating electrical side of the vessel.

Diesel engine motors generator model includes voltage controller, and implementations for under voltage and overvoltage responses. The diesel engine controls the generated power based on the difference between the high side voltage, and the reference voltage.

The diesel engine is represented as a Simulink diesel engine governor model, and with a discrete time integrator to represent, the rotational forces. Then the generator is represented as a torque load for the diesel engine. Simplified engine Simulink model is presented in Fig. 4.1.

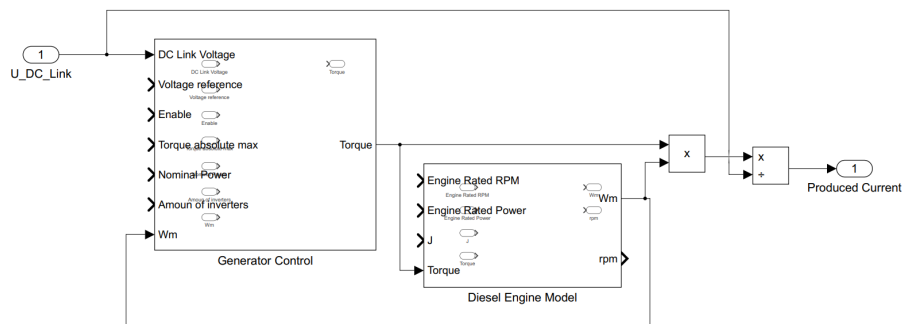


Figure 4.1. Simplified model for the diesel engine and the generator.

4.2.2 Store model

Store model represents both, the battery system, and the DC-DC converter between it and the DC-link connecting all other devices. This is a critical part of the simulations, as voltage levels for both the battery and DC-link need to be accurate, to represent vessels electrical systems. Battery system often acts the main stabilizer of the DC-link, and as such is pivotal point of the simulations. As the primary focus of the simulator is in DC-links and the DC-bus they form, were the 2 DC-links and bus tie combined to one model. This keeps accuracy

of these simulations as high as possible, without the disadvantage the modular simulation model structure brings.

Battery acts as the low side voltage source of the DC-DC converter, while DC-link network is the high side voltage. The battery is represented as a general Simulink model. This is due to the fact that accurate battery specs are unavailable during the time period when the software for electric vessels is developed. The battery systems are developed by a separate third party company, that can be different for every project, thus realistically modelling all different batteries for different projects is unrealistic. Most critical parts of the system's reaction to the usage a battery system, can be simulated using a more generalized battery model. How specific batteries from different manufacturers react, cannot be simulated, without specifically accounted for that.

High voltage side for DC-DC converter is the DC-link itself. All its loads and generators are represented as current sources. Value for current is calculated and inputted by the simulation framework, but it is the sum of all current components. As electric vessel often has two separate DC links, that are then connected by a bus tie, there are two variants of this the model. One with only one DC-link and one battery system. And another one with two battery systems and two DC-links, that are then connected via bus tie model.

The battery energy system is represented as a Simulink general battery model. This was chosen, because the actual battery systems used in vessels can come from different manufacturers, and accurately representing it could be a challenge, as those models are not widely available. Behaviour of the battery itself is not that crucial for judging the performance of the DC-link, as the performance of DC-DC converter is what affects it the most.

The DC-DC converter model can be divided into two parts. One is the control model, and one is then the electrical model. For the electrical model, two different methods were tried. The option one was to use voltage source to represent the high voltage side of the converter, and current source in low voltage side. The option two was to use the Simulink provided D-averaging model for the electric model. As both methods were tried out, the option two was chosen, as it behaves more accurately behaves in non-switching state of the converter.

The DC-DC converter control model was developed by hand, to closely represent what

Danfoss Editron developed control models represent. The modelled entities include reference ramping, linear under- and over-voltage controllers, voltage drooping and several PID-controllers.

The final Simulink-model representing battery system and DC-DC converter is shown in figure 4.2.

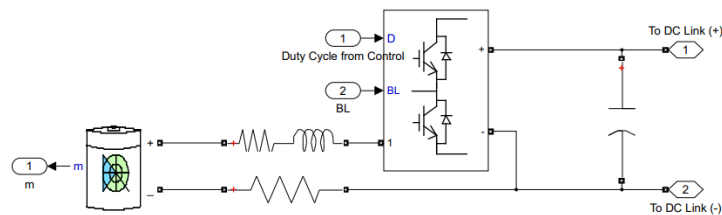


Figure 4.2. Simulink-model for the DC-DC Converter, and the battery unit.

4.2.3 Modelling DC-Link and bus tie

DC-link does not contain any complex circuits itself. It is formed from simple resistors, capacitors and inductors. Since it is essentially just connecting the other components, and has capacitors to have more voltage stability. Also, every inverter and converter has a capacitor of its own, that was chosen to be modelled in the same Simulink-model, as the DC-link. This is due to approach chosen for splitting the simulation into separate models. But keeping all the capacitors in one model, keeps the communication between each model simpler. All producers and consumers, other than DC-DC converters, are represented with current sources in the DC-link model.

The current source approach allows easy scaling for larger vessels, with many components. All currents are summed in the Python simulation environment before inserted in as total current of the DC-link model. No changes need to made to the dc link model itself, to allow different configurations of producers and consumers. This approach does come with some disadvantages, as different components are not actually separated in the modelled circuit.

Bus tie is a component that connects the two identical DC-link circuits to each other, from each side of the vessel. In reality, both DC-link circuits have one bus tie breaker each, that when both connected, are DC-links then connected to each other. In simulation model one

breaker is used, since it can deduct when the breaker needs to be closed, since both breakers need to be commanded to be closed for bus tie to be closed. Simulink model for bus tie is shown in Fig. 4.3.

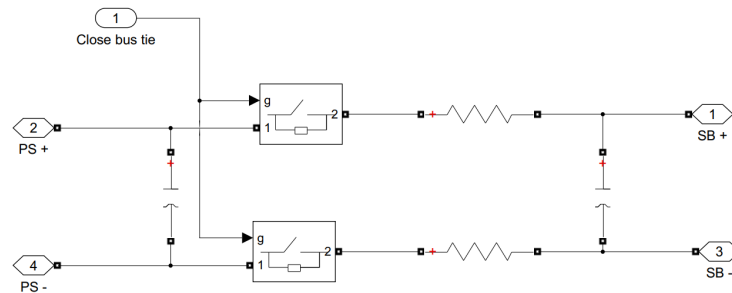


Figure 4.3. Simulink-model for bus tie. Single input signal that sets it either open, or closed.

As the main focus point of the real-time simulation is accurate DC-link voltage representation and behaviour, the DC-links from both starboardside and portside are combined in same simulation model. This is done to not compromise accuracy of said simulations by separating them to different models, and then to different processes in the developed simulation tool.

Two current sources are used per side. Two current sources were chosen, as having a dedicated current source for every component would require changes to model for every vessel, as not all vessels have same configuration. Also, all components connected can be divided generally into producers or consumers, so with this approach, it can all similarly typed devices be connected to same current source. Final Simulink model for battery system and single side of DC-link is shown in Fig. 4.4.

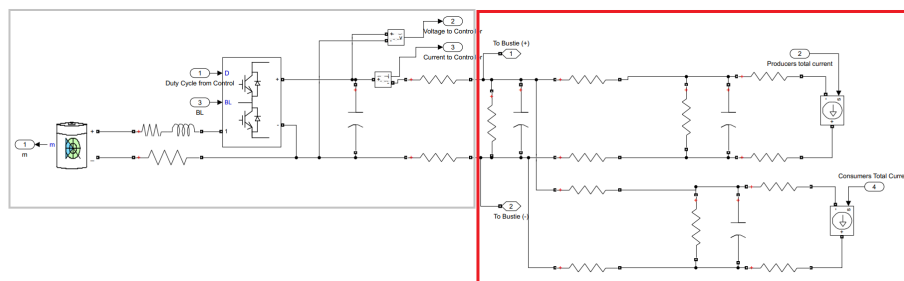


Figure 4.4. Simulink model for the DC-link, with controller model left out. DC-DC converter is highlighted with grey, and the DC-link is highlighted with red.

4.2.4 Load Model, Grid Model and Shore Connection

Load model is used to simulate all possible variable loads connected to the vessels electrical systems. This load can represent any miscellaneous load or source that is not accurately modelled, but needs to be vaguely represented in the vessel. It can also be used to simulate how the ship systems react to extra load. But its main purpose is to represent the hotel electric network of the vessel. This network is the AC grid, that often is required in a vessel. The load model is simple equation that calculates current required from DC-link, for configured load in Watts. Variance in the load is represented as variable multiples of the reference power. The actual AC grid is not simulated in any way, and the reference values PLC requests from this model, are just feedback to the PLC as they are. This is done, since the frequency and the voltage of the grid, do not provide much value to the simulations done. Also developing accurate AC grid simulation is hard and time intensive, since amount of the connected devices is large, and unpredictable. Focusing point was decided to be on DC grid early on simulators developing cycle, so simulating the AC grid accurately would just be not only be computationally intensive, but also not provide any meaningful information. The load model can be seen in Fig. 4.5.

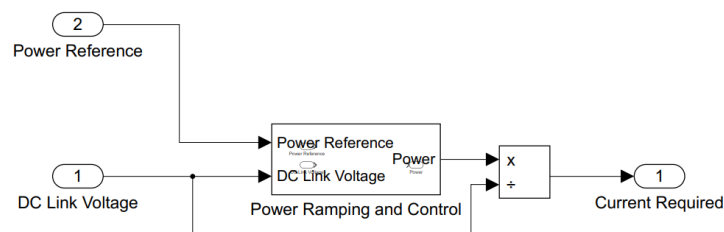


Figure 4.5. Simulink model for general loads. Current is simply calculated using smoothed power reference, received from simulation tool.

Where in load-type cases inverter is supplying power from DC grid to AC grid, or another type loads, the direction of the power can be in the other direction, where the AC grid or another source feeds power into DC grid. This functionality was implemented in the simulation model, as there can be cases, where there is a need to be able to feed power back to the DC grid. Such cases are where the AC grid is connected to shore grid. This configuration is common, as electric vessels need to be chargeable from shore. There is also a possibility of having a diesel generator connected to AC grid. For those situations being able to feed

power to the DC grid is essential.

For finalized simulation model, the model includes functionality for both directions of power from the DC grid. One mode for usage as a load, in use cases such as thrusters or AC voltage grid generation. Other mode for usage as feeding power back to the DC grid in AFE mode. When using as load, the power is simply ramped to user supplied value. The consumed power is calculated based on DC grid voltage. When feeding power back to the DC grid, a PID-controller is used, to control the amount of power fed. The controller is modelled based on the controller on the actual inverter controllers. Both modes have undervoltage and overvoltage linear controllers were also implemented in the model. Final Simulink model is presented in Fig. 4.6.

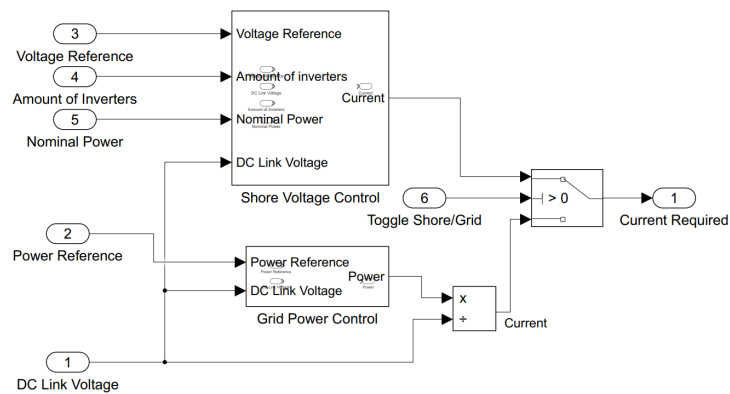


Figure 4.6. Simulink model for combined shore and grid functionality.

4.2.5 Propulsion

For propulsion solutions that are in the scope of this simulation tool, are the hybrid power solutions, where a diesel generator is used with an electric machine, and a full electric solution where the whole propulsion system, in this case an electric motor is driven by the DC grid. The propulsion systems that are powered alone by diesel motors, are overlooked, as they don't affect how electrical grids work, even if such components were in control of EMS. There can be multiple inverters used to power a single propulsion motor, and this needs to be accounted for in the simulation model.

There are several references one can use to control an electric motor, but for this work, was speed control chosen as the focus point for propulsion model. Other two main controls

are power control and torque control, where torque control is especially relevant for hybrid solutions, as motors are controlled with torque reference, to reduce the load of the diesel motors. The speed control was chosen, since it was most relevant for upcoming projects at Danfoss Editron, with torque control left as an option for the future, to be done if there was enough time.

The propulsion model follows similar principle as previously introduced load model, where required current is calculated based on DC grid voltage. But a more sophisticated model was used to simulate the dynamics of the electric motors. A simple integrator model with propulsion curve feedback was used to represent the motor's dynamics. Propulsion curve represents the fact that power required to grow squared in relation to speed of the propeller. Linear over-voltage and undervoltage controllers are also implemented in the controller model.

Speed is then controller via a PID-controller, that controls the duty-cycle of the inverters that then converted back to torque produced. The generated torque is limited as a function of speed, as is in real world applications. Multiple inverters are accounted for by multiplying the produced torque, as control is running for one inverter. It is assumed, that all possible connected inverters are configured identically. Final Simulink model is presented in Fig. 4.7.

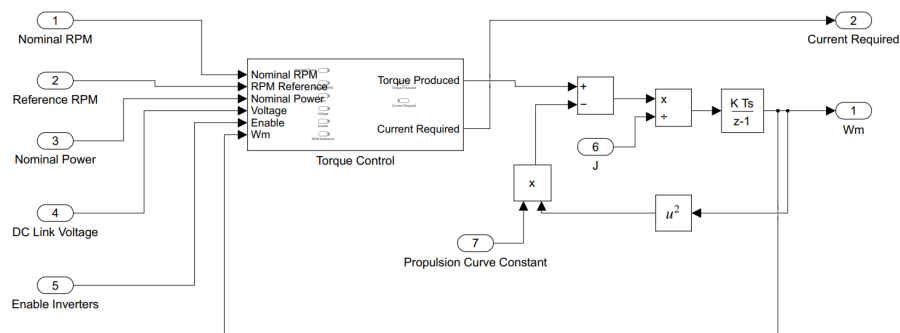


Figure 4.7. Simplified propulsion Simulink model.

4.3 Connections Between Models

Due to design decisions made in programming structure of the simulation tool, the connections between the simulation models, need to be executed in a way, where there isn't constant connection between them. In other words, for every calculation cycle in the model, its inputs

from other models do not change. The simulation model, and the scheduler will run at different frequencies, as previously stated in Chapter 3.5. A sweet spot communication cycle needs to be found, that provides both accuracy and speed provided by multiprocessing.

Also, for models to be compatible, there needs to be standard interface between the different simulation models. In Fig. 4.8, a simplified communication diagram between models shown.

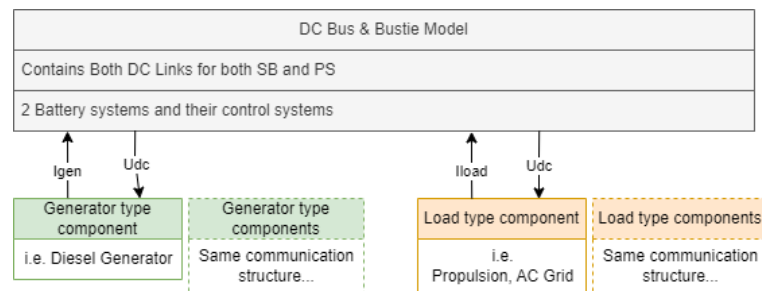


Figure 4.8. Communication structure between different simulation models generalized. All similar type components use the same communication structure. Special inputs and outputs like, engine speed are not included in the figure.

The model for the DC-link, both starboard and portside, acts as the main simulation block, as it holds all DC grid related simulation. It has outputs for both starboard DC-link voltage, and portside DC-link voltage. Both starboard and portside DC-links have inputs for generated current, and consumed current as previously mentioned in Chapter 4.2.3. To those port child components output their required or produced power, that is combined by the Python simulation interface.

Models then have more specific connections based on their needs, for example generator-type diesel generator has output for running speed, and load-type propulsion has input for speed reference. All the models use defined interfaces to communicate with DC-link block that acts as the common link between different models.

5 Results

Performance of the developed simulation tool is validated first against simulation models, and expected behaviour in case of logic simulations, and then accuracy of dynamics simulation is compared against measurements obtained from real world vessel.

5.1 Simulator Performance Analysis

Simulator performance metrics can be individually done to both, the PLC communications and electric dynamics simulations. Since both run separately from each other, and their performance does not really affect each other, both can be graded separately. Thus separate tests can be run to validate the performance of the entire simulator.

To grade electric simulations' performance, it's needed to measure both its accuracy and speed. The accuracy can be measured with a comparison to a more accurate simulator, using switching circuit model. The speed measurement is measuring run-time performance of electric simulations.

While performance of the logic simulations was not directly the scope this thesis, it is part of the simulation tool, thus its performance needs to be validated. The performance of the logic simulations can also be graded in similar manner as electric simulations. That is by evaluating the speed in runtime. The logic simulator can only be graded as working or not working, thus its compared against another simulation solution. If it acts in same way as the existing logic simulator, it can be considered working as intended. As the behaviour of these two simulators should be identical, as they are based on the same source code. This includes testing that simulations work with real deployable PLC software, and then comparing the results.

As functionality of the logic simulator remains largely the same, even after re-programming of entire code base, was performance of logic simulations expected to remain same as compared to the existing simulator. Logic simulator was used to test two separate vessel software packages, and for both simulations resulted in similar behaviour, when comparing to existing

testing software and expected test results. One of these vessel software packages was then deployed in Section 5.2, and in both cases, could the PLC software be acting as expected with given inputs. Thus can be concluded that the functionality of the logic simulator, remains intact after changes done.

5.2 Real World Test Case

Simulator was tested against actual real world vessel. In addition to the ordinary tests performed, was data logged from the test runs, to be compared against the simulators values.

Before a vessel is deployed and given to customer, a set of tests must be run, to qualify if the vessel is working properly and safe. Danfoss Editron also does the deployment testing, in addition of developing the systems required on vessel. This made it possible to have real world test subject to compare against the data from the simulator.

5.2.1 Vessel Setup

Vessel used for tests is slightly longer than 30 meters, with 2 battery systems, 2 electric propulsion motors, and one diesel generator. More accurate information about the vessels internals are not shared in this thesis work, due to confidentiality. The simulations were run with same parameters and same vessel configuration as the real vessel. All inverters used in the vessel were provided by Danfoss Editron, along with the control system.

5.2.2 Tests

Time available on vessel was limited, and that limited the amount of tests to be run. Most important functionalities of the vessel were tested from the viewpoint of the simulator. Different transients are the most important things aside of general accuracy of the simulator. With different transient measurements, the most impactful parts of the simulator can be validated.

The following tests were run on the vessel:

1. DC-link cold start
 - (a) Cold start using DC-DC Converters and Battery Systems
 - (a) Cold start using the Diesel Generator
2. Load types
 - (a) Sudden power requirement (Thrusters)
 - (b) Slowly rising power requirement (Propulsion)

These tests were chosen as with these tests can most important aspects of the vessel be recorded, and most common transitions captured. One part to be more thoroughly tested is load testing, but the state of the vessel was not ready for heavier loads, and only lighter loads could be tested. With this limitation in mind, could not propulsion for example be tested at full power. With two different load types we can emulate how the DC-link voltage reacts to different types of transient loads, that essentially all other loads can be divided into. Loads either ramp up to their peak loads, or they are more instantaneous.

Preceding tests would then be done using the simulation software, and then compared to actual test results. The measurements on the actual vessel were done using Danfoss EC-C1200 inverters included PowerUSER 9.6.0 software, using the internal measurement methods. Same parameters that were used on the inverters, were used on the simulation. Same software was used on PLCs in both situations, with no changes, except for disabling hardware inputs and outputs for the simulation.

PC was connected directly to inverters using a serial cable. PowerUSER software uses variable logging frequency, but logging interval for these tests was measured around 150–200ms.

5.2.3 Test Results

DC-link Cold Boot with Diesel Generator

In DC-link cold boot test, the voltage of DC-link is raised to a level where functionality of

other systems can be guaranteed. This voltage level depends on chosen parameters, but often this voltage level is in the range of 650–800 Volts.

Point of this test is to test if the control systems of generator model work, and it is capable to reliably charge up DC-bus voltage. The accuracy of simulations can be compared to real world measurements. Simulated and measured curves showing the performance on generator powered start-up, are shown in Fig. 5.1. In figure, the DC-link voltage behaviour can be seen, when using diesel generator as the power source.

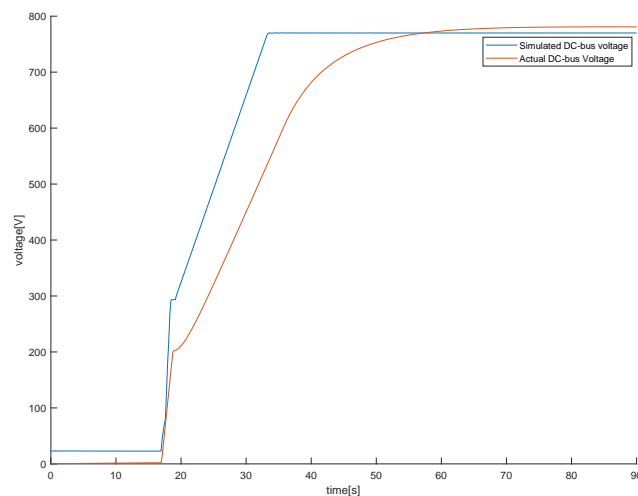


Figure 5.1. DC voltage from high voltage side of DC/DC converter, from both actual measurements and simulations. Plots are matched where voltage starts to rise at same t .

From the results, we can see, that ramping speed, and accuracy are not equal to actual real world system. From the figure can be seen, that real world system starts to follow reference voltage later, at 300 V, instead of 200 V where simulation model starts. Also, the ramping voltage is not sharp, as it is in real world system. In addition to this, generator overshoots the DC-bus voltage reference of 770 V by 10 V. Poor performance of the voltage and power controllers is due to simulation approach of variable communication steps between models. Diesel generator model runs at 16 kHz frequency, while the communication between simulation model is only at 1 kHz. Tightening this down results in better simulation results.

DC-link Cold Boot with Battery Systems

Similar to the test done with diesel generator, the goal of the test is the same, but instead of charging the DC-link with the diesel generator, were the battery systems of the ship used.

For this test, only one of the battery systems were used, while the other one remained in a non-modulating state.

Point of this test is to test simulation quality for DC-DC converters, and how closely the simulation model represents real world in a situation where DC-bus voltage is close to zero, and voltage is raised using battery systems of the vessel. In Fig. 5.2, is the voltage response of the test is shown along the simulated one.

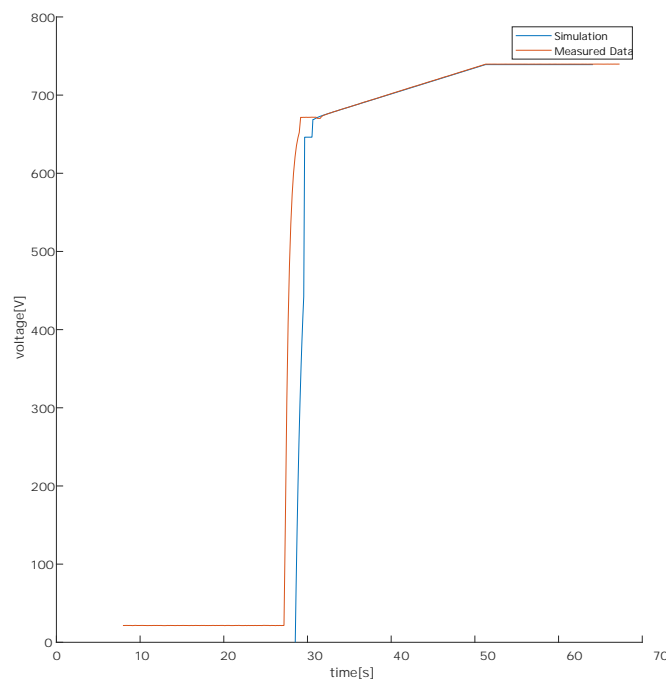


Figure 5.2. DC voltage from high voltage side of DC-DC converter, from both actual measurements and simulations. Plots are matched in a way where voltage reference is reached at same t .

From the results can be seen that the simulation follows real world representation very closely, after power ramping begins, and DC-bus voltage starts to rise. The voltage response from the starting voltage V_{low} that is the lowest measured voltage, is slightly different between simulations. This is due to different timings of the breakers. Simulations for DC-DC converter can be considered to meet their requirements, as the main focusing point of simulations, is the DC-bus voltage, not the step response from V_{low} .

Large Sudden Load Test

Large sudden load test is a general test to see how DC-link voltage reacts to a larger sudden load. This test is important, as if the voltage drops too much, it can cause undervoltage trips in devices. This test was difficult to execute on the real vessel, due to the circumstances. As the load used on the real vessel, does not represent the largest possible load it can suddenly experience, this test cannot be used to verify black-out behaviour and prevention. Simulation has under voltage controllers implemented, to prevent power outages, but these could not be compared against real vessel. Large sudden load test was thus conducted with thrusters of the vessel. Thrusters of the vessel apply sudden load on the vessels AC grid, that then applies large sudden load on the DC grid. Thrusters load profile, is sudden at first, but it then ramps off. This is problematic, as such load ramping features are not programmed into the simulation tool. For this reason, transients, on how DC-link voltage recovers, are not comparable to each other. In simulation tool, large sudden load was applied, and this load was then removed suddenly, when it was ramped off in the real world vessel. In Fig. 5.3 are transients of large sudden load shown. For simulations these three loads were chosen and shown, because the change in DC-link voltage can be clearly seen.

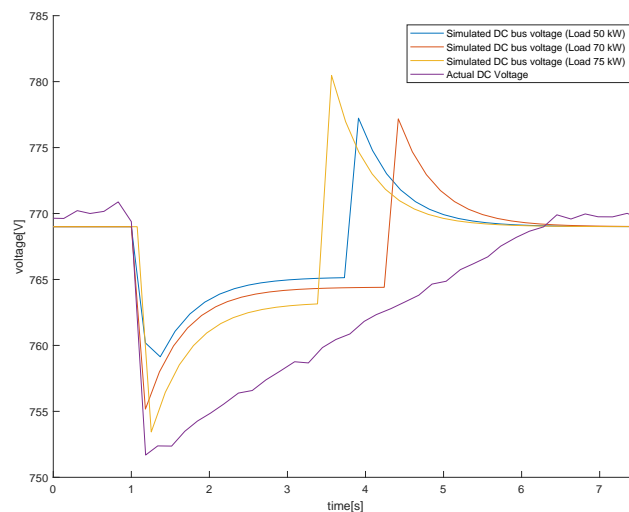


Figure 5.3. DC-link voltage, when a sudden load is applied. Simulation was performed with 3 different power levels. In real world system thrusters are ramped off. On the contrary in simulations loads are suddenly removed due to technical limitations.

From the test, it can be concluded that DC-link voltage reacts to sudden loads as expected, with a sudden voltage drop, where it starts recovering from. Size of that voltage drop, is directly connected to the size of the sudden load. Difference on, how it recovers, is not

comparable, due to types of loads being different. This difference can be improved upon, with better load models, that can represent, the nature of thruster load. But for the purpose of proving that, the voltage reacts to a load, this test does what it is meant to do.

Slowly Rising Load Test

Slowly rising load test can be used to simulate loads such as propulsion, where the power load ramps up slowly, after initial power spike. Test can also be used to see how the systems compare when reacting to a ramping loads. In this test, propulsion was used as the ramping load in both cases. In this is case is also to be noted, that propulsion reference can only be input by the operator, with the propulsion levers at the deck of the vessel. But on simulator, direct position of the lever can be input with certainty, with no possibility for human error.

For these tests, approximately 250 rpm reference was used as propulsion reference. Larger propulsion references could not be used, as the vessel was tied to the dock. Results of the slowly rising load test can be seen in Fig. 5.4.

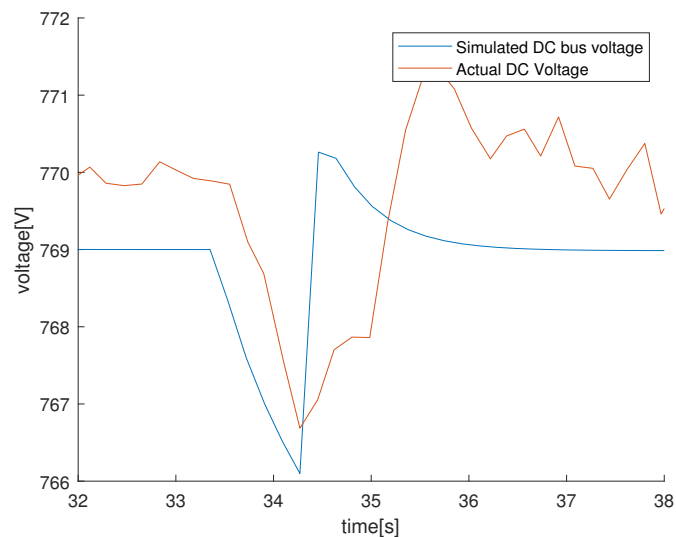


Figure 5.4. DC voltage from high voltage side of DC-DC converter, from both actual measurements and simulations, when propulsion motor reference is raised to 250 rpm. The plots are matched by time when the RPM reference is received.

Largest difference between simulator and real world, is the DC bus voltage, that is in this case, approximately 769 V on simulation, and 770 V on real world simulation. Ignoring this, the voltage behaviour of both is similar, as voltage slightly dips when load is applied, then recovers with slight overshoot, when the propulsion motor reaches its speed reference. As

the voltage ripple is quite high, a moving average with sliding window length k of 3 was used, and an offset of -1 V was applied to measured actual DC voltage. Results can be seen in Fig. 5.5. In Fig. 5.6 actual speed of both propulsion motors is presented. Note that for simulation the reference was kept at 250 rpm.

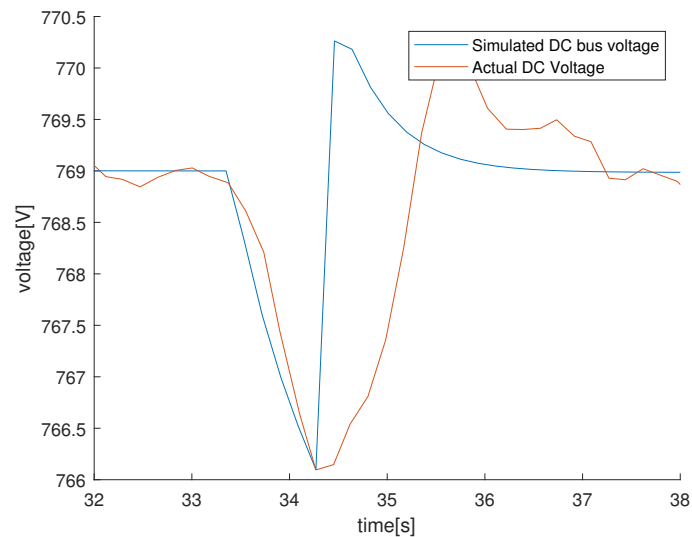


Figure 5.5. DC voltage from high voltage side of DC/DC converter, from both actual measurements and simulations. The measured data ran through moving average with sliding window size $k = 3$. The plots are matched by time when speed reference is received.

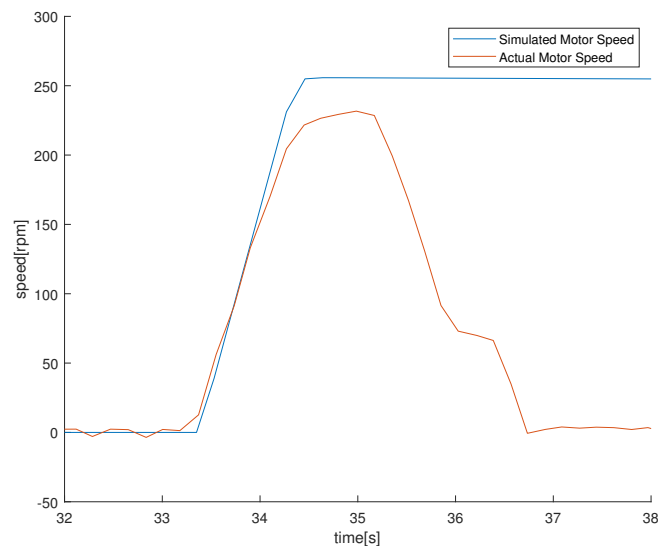


Figure 5.6. Propulsion motor speed with reference of 250 rpm on simulation, on actual motor, by lever. The plots are matched by time when speed reference is received.

When applying a moving average to measured voltage from the real world vessel, the initial voltage is very similar to the simulated voltage as seen in Fig. 5.5. Larger variance occurs,

when the motor reaches reference, where simulated voltage reacts very fast comparing to measured real world data. This could be partly explained, by the simulation environments, incapability to reproduce parabolic speed reference, that was used by the operator with the real world tests. As the motor reference is different, response seen in Fig. ?? is different. As the goal of the simulations was to achieve operational accuracy, can results be seen as a success, as voltage amplitudes are almost the same. Differences in response times and transients can be neglected, and improved upon.

Based on the simulated results, it can be concluded that all the dynamics are not included in the models. The voltage does not have any ripple, and remains almost as a constant. On the contrary in the case of measured data, larger fluxuations of $\pm 0.5V$ can be seen. This can be caused by two factors. The option one is that these fluxuations occur, due to switching events, and as those events are not simulated, naturally their dynamics is cannot be seen in the results. The second option is that these fluxuations are caused by measuring equipment, and as such result appear as ripple in the measured data.

The speed response of the propulsion motor, seen in Fig. 5.6 has a similar rising transient, as the real world propulsion motor, and the operational accuracy can be seen as achieved.

5.2.4 Conclusions about the simulations

By these results, simulation tool developed can be considered working, but there is room for improvement. For instance, the voltage ripple is not simulated at all, as the simulation model reaches stability to a level not seen in real world test case. Simulation of diesel generator leaves some major accuracy gains to the table, and solid improvements can be done on that front. The DC-DC converter model, and DC-link model, can be seen working very well, replicating the behaviour of real world tests in very convincing manner.

5.3 Runtime Analysis

Runtime performance of the simulation was measured to be in real-time. To analyse simulation time savings gained by all measurements taken in this thesis, a comparison was made,

where the same vessel configuration, as used with real world testing was also simulated in MathWorks Simulink, and the time taken was compared against the developed simulation tool. Both tests were performed on same computer. But as computer specifications change the results of this test, more research needs to be done to verify these results. As this was not the focus of this thesis, only simple measurement was done. In Fig. 5.7 a comparison between Simulink performance, and simulation tool performance is shown. Test situation used for the test, was DC-link cold boot test, where the DC-link was charged using the DC-DC converter. Test was performed using simulation step of 60 seconds. The simulation tool was capable of simulating the vessel in real-time, as such simulation took 60 seconds. While using simulink, simulating the model took 5 minutes and 50 seconds. Simulink was configured to run with fixed-step of 62.5 nanoseconds. In simulation tool, internal simulation step size for individual simulations was 62.5 nanoseconds, but the communication interval between the simulation blocks was 10 milliseconds.

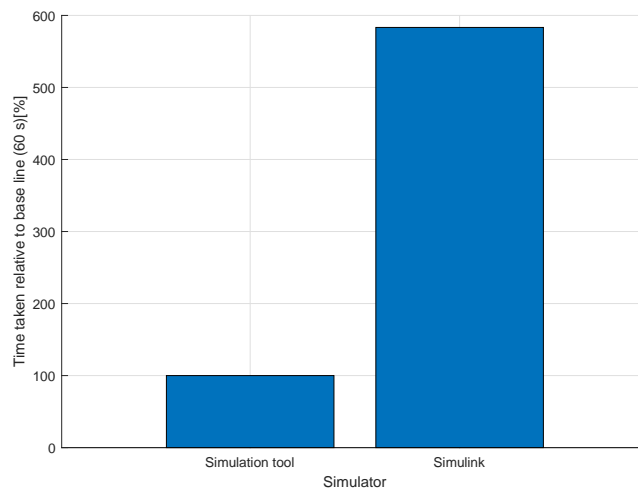


Figure 5.7. Relative performance between simulators, when simulating vessel model for step size of 60 seconds. As the simulation tool was running in real time, the result was 100%. Simulink completed simulating to stop time in 5 minutes and 50 seconds. A result of 583% when comparing to base time (60 seconds) .

As the sample size is small, any definitive conclusions cannot be made about the approach of splitting simulation load to multiple processes. Many more variables need to be tested, and more research be done to achieve more conclusive results. But in this test case in the scope of this thesis, and with this vessel configuration, notable simulation time differences can be seen between the two simulators.

6 Conclusions

In this thesis, a simulation tool capable of real-time simulation of an electric vessel was developed. Operational accuracy was the targeted for the simulations. Simulation tool is Python based program, that utilizes multiprocessing to achieve real-time speeds. Simulations were implemented in FMUs. Whole simulator was developed using modular design principles, and GUI was developed, for user to interact with the simulation tool. The simulation tool was to communicate with the PLCs using ADS. The simulation tool was capable of providing time savings during the testing period of software, and offered additional time savings, when compared against the existing solutions. The performance of the simulation tool was evaluated against a real world vessel, where its characteristics were deemed to be of operational accuracy. Performance of tools logic simulator was compared against the existing logic simulator, where it was found to be working identically. Tools capabilities in simulating dynamics proved mixed results, in terms of mirroring real world data. While some test results were in a good agreement, it is clear that some simulation blocks could use additional time to achieve their fullest potential. From comparison against Simulink model, with same simulation setup, could major simulation speed improvements be seen, but more research needs to be done, to validate and form more conclusive take, on this approach, and the advantages it can provide over more traditional simulation approaches.

Concluding, a simulation tool was developed in Python. Simulation tool fulfilled its requirements that were set. Simulation tool was capable of communicating with the Beckhoff PLCs using ADS whilst simultaneously simulating electric vessels DC grid and its connected components in real-time. Completing the goals that were set out for it in this thesis.

6.1 Future work

While the simulator works quite well in principle, there are still some things that can be done. One main thing is the battery communication. The batteries in vessels can communicate using Modbus, but this simulation tool does not have Modbus communication support. While adding communication support for Modbus is possible, it is most certainly out of this

projects scope. Steps are taken inside Danfoss Editron to add this feature later during tools life cycle. The simulator already has functionality to output requested data for the battery communication using external tools, but those tools, do not have required functionality yet. This feature remains largely untested. The simulator also could be better integrated, to other software suites at Danfoss Editron. This is taken into consideration in the future.

User interface, while perfectly working functionally, also does not have the functionality and the quality of life features of most professional software suites. Designing and executing more demanding software were outside the scope of this thesis, but in the future, could these UI improvements be made, and already some changes and features have been added to the user interface during the development.

Configuring of the simulations is not as simple, as it ideally could be. Operator needs to know what they want to do, and how to do it in the context of the simulation tool, to build a configuration file. For simulating PLC IO, having enough configurability is a requirement, but configuring dynamics simulation, when it is essentially known what components vessel already has from logic simulation configuration is unnecessary. This could be improved upon, and automatic generation utilized, to further simplify the usability of the software.

Simulation solution needs to be further investigated and researched, to deem its worth in simulations. While in this thesis, results were proven positive, not enough research was done to provide deeper analysis on the subject, as it was outside the scope of this thesis.

References

- [1] Beckhoff. *ADS-Communication*. Accessed: 2022-10-30. 2022. URL: https://infosys.beckhoff.com/english.php?content=../content/1033/cx8190_hw/5091854987.html#:~:text=ADS-Communication%20The%20ADS%20protocol%20%28ADS%3A%20Automation%20Device%20Specification%29,the%20communication%20between%20the%20NC%20and%20the%20PLC..
- [2] Doglio Fernando. *Mastering Python High Performance : Measure, Optimize, and Improve the Performance of Your Python Code with This Easy-to-follow Guide*. Community Experience Distilled. Packt Publishing, 2015. ISBN: 9781783989300. URL: <https://search-ebscohost-com.ezproxy.cc.lut.fi/login.aspx?direct=true&db=e000xww&AN=1063765&site=ehost-live>.
- [3] *Functional Mock-up Interface Specification*. Accessed: 2022-11-05. May 2022. URL: <https://fmi-standard.org/docs/3.0/>.
- [4] Krister Gräsbeck. *Simulation of Field Devices in Testing of a Marine Control System*. eng. Tech. rep. LUT University, 2021.
- [5] Marko Haapaniemi. *Traktorin sähköistysanalyysi virtuaalisimulaatiomallilla*. fin. Lappeenranta teknillinen yliopisto, School of Energy Systems, Sähkötekniikka / Lappeenranta University of Technology, School of Energy Systems, Electrical Engineering, 2016.
- [6] Gastón C. Hillar. *2. Processes and Threads*. Packt Publishing, 2009. ISBN: 978-1-847197-10-8. URL: <https://app.knovel.com/hotlink/khtml/id:kt00BE3VF1/c-2008-2005-threaded/processes-threads>.
- [7] P. Hintjens. *ZeroMQ: Messaging for Many Applications*. O’Reilly Media, 2013. ISBN: 9781449334444. URL: https://books.google.fi/books?id=TxHgtl%5C_sFmgC.
- [8] IBM. *What Is a Digital Twin*. Accessed: 2022-10-24. 2022. URL: <https://www.ibm.com/topics/what-is-a-digital-twin>.
- [9] *IEEE guide for the design and application of power electronics in electrical power systems on ships*. eng. New York: IEEE, 2009. ISBN: 0-7381-5840-2.
- [10] Esa-Pekka Kaikko. *Development of Generic Simulation Model for Mobile Working Machines*. eng. Lappeenranta teknillinen yliopisto, Teknillinen tiedekunta, LUT Kone / Lappeenranta University of Technology, LUT School of Technology, LUT Mechanical Engineering, 2015.
- [11] Al. Kayhn. *Smart Power Grids 2011*. eng. Power Systems. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 351–399. ISBN: 9783642215780.
- [12] Simo Kuusela. *Electric drive simulator for system simulations*. eng. Tech. rep. LUT University, 2019.
- [13] Stefan Lehmann. *PyADS Documentation*. Accessed: 2022-10-31. 2015. URL: <https://pyads.readthedocs.io/en/latest/>.
- [14] Hadil Mustafa et al. “Real-Time FPGA Simulation of Electric Ship Power System Using High-Level Synthesis”. In: *2019 IEEE Electric Ship Technologies Symposium (ESTS)*. 2019, pp. 377–381. DOI: 10.1109/ESTS.2019.8847870.

- [15] M. D. Omar Faruque et al. “Real-Time Simulation Technologies for Power Systems Design, Testing, and Analysis”. In: *IEEE Power and Energy Technology Systems Journal* 2.2 (2015), pp. 63–73. DOI: 10.1109/JPETS.2015.2427370.
- [16] T Pavlovic, T Bjazic, and Z Ban. “Simplified Averaged Models of DC-DC Power Converters Suitable for Controller Design and Microgrid Simulation”. eng. In: *IEEE transactions on power electronics* 28.7 (2013), pp. 3266–3275. ISSN: 0885-8993.
- [17] Danfoss Editron Powerpoint. *System Product Overview Marine*. Accessed: 2022-10-28. 2022. URL: https://files.danfoss.com/download/PowerSolutions/Mobile%20Electrification/presentations/Danfoss-Editron_BU-Marine_V6.pdf.
- [18] QT. Accessed: 2022-10-14. 2022. URL: <https://www.qt.io/>.
- [19] Andrzej M. Trzynadlowski. *1.1 Introduction*. Institution of Engineering and Technology (The IET), 2016. ISBN: 978-1-84919-826-4. URL: <https://app.knovel.com/hotlink/khtml/id:kt010RPV5A/power-electronic-converters/introduction>.
- [20] Andrzej M. Trzynadlowski. *15.1 Shipboard Power System Topologies*. Institution of Engineering and Technology (The IET), 2016. ISBN: 978-1-84919-826-4. URL: <https://app.knovel.com/hotlink/khtml/id:kt010RQ454/power-electronic-converters/shipboard-power-system>.
- [21] Python Software Foundation Website. *multiprocessing - Process-based parallelism*. Accessed: 2022-12-17. 2022. URL: <https://docs.python.org/3/library/multiprocessing.html>.
- [22] ZMQ Website. *Documentation*. Accessed: 2022-10-30. 2022. URL: <https://zeromq.org/get-started/>.