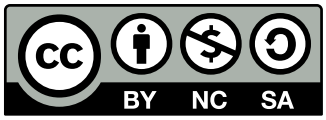
Aalto University
School of Engineering

Master's programme in Mechanical Engineering

Towards a Standardized Framework for Collaborative Ship Powertrain Design using the System Structure and Parameterization Standard

Sashank Neelamraju

Master's Thesis
2024



Aalto University
School of Engineering

Author Sashank Neelamraju

Title Towards a Standardized Framework for Collaborative Ship Powertrain Design using the System Structure and Parameterization Standard

Degree programme Mechanical Engineering

Major Mechatronics

Supervisor Assistant Professor Raine Viitala

Advisor D.Sc (Tech.) Riku Ala-Laurinaho and M.Sc (Tech.) Reino Ruusu

Date 27 March 2024 **Number of pages** 73 **Language** English

Abstract

The maritime sector plays a vital role in global trade and transportation but there is a need for constant innovation within this industry to meet sustainability goals. Optimizing powertrains through full-system simulations during the initial design phase is a key solution. However, the manual selection of suitable component models due to multiple vendors offering similar components leads to inefficiencies. This thesis aims to contribute to the development of a collaborative design framework to automate component selection and simulations, integrating OEMs and system designers for model sharing and powertrain simulation. Standardization is key to this framework's development. This thesis focuses on standardizing component requirements and powertrain simulation model descriptions.

A literature review was conducted to investigate the suitable standardized methods. The System Structure and Parameterization (SSP) standard was found to be a suitable candidate for describing powertrain simulation models, while an ontological approach was chosen for describing component requirements. A case study applied the SSP standard to describe a simple powertrain simulation model while incorporating the Vehicle Signal Specification ontology for component requirements description to evaluate whether the selected methods fit within the framework.

A qualitative analysis evaluated the SSP standard's suitability based on literature review findings and practical experiences with SSP model creation and simulation. Criteria such as ease-of-use, interoperability, modularity, and accuracy were assessed. The results demonstrated the suitability of the SSP standard, combined with the Vehicle System Simulation (VSS) ontology, for automated component selection and powertrain simulation. While effective, drawbacks in both the SSP standard and VSS ontology were identified, with suggestions proposed to address them.

The findings affirm the viability of standardized methods, particularly the SSP standard and VSS ontology, in automating component selection and powertrain simulation. Addressing identified drawbacks will enhance the framework's efficiency and effectiveness in supporting sustainable innovation within the maritime sector.

Keywords System Structure and Parameterization, Functional Mock-up Unit, Co-simulation, Powertrain, Standardization, Component Requirements

Preface

I would like to express my heartfelt gratitude to Professor Raine Viitala, my esteemed supervisor, for his invaluable guidance and unwavering support throughout the entire journey of my thesis. His knowledge and guidance have been very helpful in molding my work.

I am also deeply appreciative of my advisor, Riku Ala-Laurinaho, for introducing me to such a captivating and meaningful topic. Riku's belief in my abilities and continuous encouragement have been pivotal in my successful completion of this thesis. His guidance has truly been a cornerstone of my academic journey.

I am extremely grateful to Reino Ruusu, who was also my advisor, for constantly supporting me with the technical aspects of the thesis. His guidance has helped me to understand the concepts better and apply them in practice.

As a team member of the Digital transformation of collaborative powertrain design (Co-Des) project, I would like to thank all my colleagues who have provided me inputs to constantly improve and learn through my thesis.

Lastly, I would want to express my gratitude to my family and friends, whose unwavering encouragement and support have been a continual source of strength throughout the writing of my thesis. Their belief in me and their encouragement to embark on this academic endeavour have meant the world to me.

Otaniemi, 27 March 2024

Sashank Neelamraju

Contents

Abbreviations

AAS	Asset Administration Shell
CT	Continuous Time
Co-Des	Collaborative Design
DE	Discrete Event
DTD	Digital Twin Document
DTW	Digital Twin Web
HLA	High-Level Architecture
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
OSP	Open Simulation Platform
SSD	System Structure Definition
SSP	System Structure and Parameterization
VSSo	Vehicle Signal Specification Ontology

1 Introduction

1.1 Background

The maritime sector has long been an essential part of international trade and transportation, enabling the reliable and efficient transit of commodities over long distances. However, to align with sustainability objectives, the sector must constantly innovate and adopt new methods to minimize emissions and enhance efficiency. One key strategy is to optimize the powertrain during the initial system design phase using full system simulations.

System designers perform full-system simulations of ship powertrains by integrating multiple component models which are typically developed by Original Equipment Manufacturers (OEMs) of the components. However, OEMs are often hesitant to share these models due to concerns about intellectual property rights (IPR) infringement. To address this issue, black box models, which conceal the internal workings of the model, are found to be an appropriate solution for sharing while protecting IPR. To perform simulations, a full powertrain system needs to be described showing how these black box models are connected with each other. However, there is a challenge to select suitable component models for describing the system. There are often multiple such vendors offering similar components black box models, making the selection of suitable component models for simulation a laborious task. Currently, System designers have to contact individual OEMs and request the component models and when received, they manually browse through the specifications and verify whether they are compatible with the requirements of the powertrain. If suitable component models are found, they are assembled together to form the powertrain system and simulations are performed. If the simulation results are not satisfactory, the entire cycle has to be repeated which makes this process highly inefficient. Figure 1 shows the traditional powertrain design process.

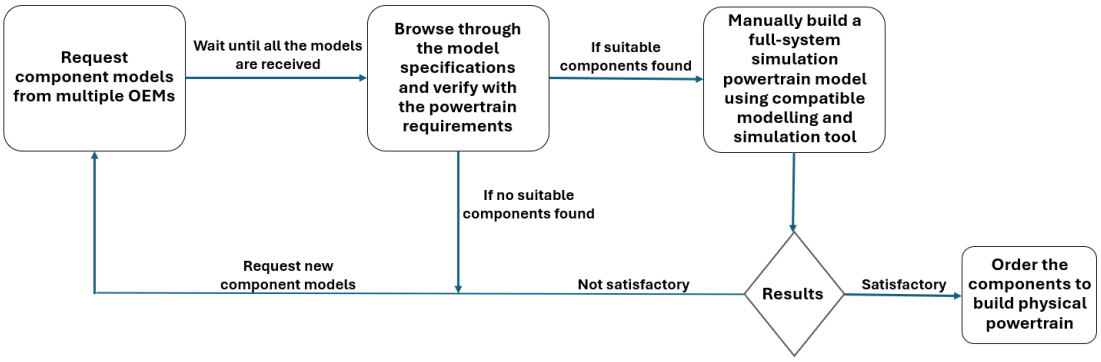


Figure 1: Traditional powertrain design process.

In order to streamline the powertrain design process, there is a need to automate the component selection and simulations. To achieve such automation, a collaborative design framework has to be established to integrate the OEMs and system designers.

This thesis aims to contribute towards the realization of such a framework and refers to it as the **Co-Des Framework**. The Co-Des framework would consist of four main components: Component Descriptions created by OEMs, a Digital Design Template created by system designers, an open-source platform accessible by both OEMs and system designers to collaboratively exchange the component models and finally a simulation interface to perform simulations using the shared component models. The Co-Des framework is explained in more detail in chapter 2.

To implement such a framework, standardization plays a key role. Each component of the framework has to be standardized so that they are compatible and interoperable resulting in a smooth and efficient simulation process. Therefore, this thesis focuses on identifying suitable methods to standardize the components of the Co-Des framework, especially focusing on standardizing the elements of the Digital Design Template (DDT) which describes the system design.

1.2 Research Problem

Previous research [1] has identified methods to standardize the sharing of component digital twins while safeguarding intellectual property rights (IPR). However, a significant gap exists in standardizing the description of the powertrain models and the component requirements that interact with these standardized digital twins. The following research questions formulated for this study aim to address the challenges associated with achieving this standardization, providing insights and solutions

to enhance the collaboration between ship powertrain designers and component manufacturers:

1. What are the methods to define a system structure for a ship powertrain system simulation in a standardized format?
2. How can the requirements for ship powertrain components be standardized?
3. How can these methods be implemented in the Co-Des framework to automate the component selection and simulation for a collaborative and streamlined powertrain design?

1.3 Objective

There are three key objectives of this thesis which would be key in realizing the Co-Des framework. The primary objective is to investigate the standardized methods to describe a ship powertrain simulation model. This description aims to efficiently manage the component models and their properties such as inputs, outputs and parameters and their connections allowing seamless integration into a system.

The second objective is to find a suitable method for describing standardized requirements for the ship powertrain components in order to automate the component selection process. Additionally, a simulation interface is developed to automate the simulation of the selected component digital twins.

The final objective is to present a method to implement these standardized methods within the Co-Des framework and use qualitative analysis to determine whether the chosen methods are suitable for the collaborative ship powertrain design.

1.4 Scope

Tevajärvi [1], concluded that the Functional Mock-Up Interface (FMI), a tool-independent standard for model sharing is a suitable standard to share component digital twins in the form of Functional Mock-Up Units (FMU) since the standard provides a good level of IPR protection for the component manufacturers. Therefore, this study can be considered a continuation of their study and focuses on investigating standardized methods to describe a ship powertrain simulation model which are compatible with the FMI standard. While other standards maybe suitable to this application, they are left out of the scope of this thesis. The FMI standard will be explained in more detail in chapter 2.

This thesis aims to contribute towards realising the Co-Des framework for ship powertrain design focusing on standardizing the descriptions and component requirements for ship powertrain simulation models. While the framework has the potential to be applied in other domains, the thesis limits its scope to the maritime environment to provide a concentrated and in-depth analysis of its capabilities and limitations in this specific context.

In the context of ship powertrain design, component requirements can be classified into two types: static and dynamic requirements. Static requirements can be considered as the basic requirements for component selection. These requirements generally include the dimensional, physical and mechanical requirements. These requirements help to filter out suitable components from a large database containing multiple component models. The dynamic requirements on the other hand are the functional requirements that need to be satisfied during the simulation of the powertrain model. These requirements help to trace back to a specific area where the simulation results do not meet the desired output. The focus of this thesis is limited to standardizing the static requirements.

The study employs a qualitative approach to evaluate the selected standard, primarily focusing on assessing its strengths and limitations rather than the performance of the simulations. This approach serves the purpose of providing a comprehensive understanding of the overall effectiveness of the standard in the maritime domain.

Only open-source tools were employed for modelling and simulating the powertrain models. This decision was made to ensure accessibility, affordability and neutrality. However, the limitations of open-source tools should be acknowledged, and the results may not necessarily reflect the performance of commercial tools, which may offer higher accuracy or faster execution times.

1.5 Methods

The study conducted a thorough literature review and a case study followed by a quantitative analysis. To begin, in chapter 2, a comprehensive literature review was conducted to investigate the suitable standardized methods available to describe the ship powertrain system for simulations that were compatible with the FMI standard and also investigated the methods to describe component requirements. From the literature review, the "System Structure and Parameterization" was found to be a suitable standard for describing a powertrain system for simulations. Furthermore, an

ontological approach emerged as the most effective means for capturing the specific requirements of individual components.

Subsequently, in chapter 3, a case study was applied to implement the Co-Des framework using the SSP standard and ontologies. The case study presented a step-by-step process from the creation of physics-based models based on the physical components to simulation of SSP models. The FMUs were generated using the OpenModelica software from the physics-based models created using a system of differential equations. The powertrain model was created utilizing the SSP standard in the using the EasySSP modelling tool. The Vehicle Signal Specification Ontology (VSSo) was used to define the component requirements of the powertrain model. An algorithm was proposed to use the ontology to find suitable components. The simulation interface was simultaneously developed as a command-line interface using Python. Finally, the suitable models were simulated together and the results were plotted. The learnings from the literature review along with the practical experiences from the case study were used to perform a qualitative analysis of the SSP standard.

In chapter 4, a qualitative analysis was employed to evaluate the SSP standard's usability, interoperability, modularity, and accuracy in ship powertrain design. Usability was assessed based on the ease of creating and using SSP models, considering factors such as the clarity of documentation, the availability of tools and utilities, and the overall level of user-friendliness. Interoperability was evaluated by assessing the ability of SSP models to be exchanged and integrated with various simulation platforms and software applications. Modularity was assessed by evaluating the extent to which SSP models can be modified into reusable components, facilitating the creation of complex models from simpler building blocks. Accuracy was assessed by comparing the simulation results of SSP models to physics-based ones by varying the input parameters.

2 Literature review

This chapter discusses the background studies and concepts presented in previous works. This chapter is divided into six sections. The literature begins with the background of the Co-Des framework presented in section 2.1. A brief introduction to Digital twins and their role in the modelling and simulation is given in section 2.2. The subsequent sections present the standardized methods available for describing system structure and component requirements for ship powertrains. Section 2.3 presents the digital twin description standards, and section 2.4 delves into the SysML modelling language and its role in describing complex systems. Section 2.5 studies how ontologies can be implemented to describe system structures and component requirements and finally, section 2.6 discusses the various co-simulation standards and tools. The literature review aims to study all the suitable methods available for describing the system structure and component requirements of the ship powertrain, select the suitable candidate and perform further investigation on the selected method.

2.1 Co-Des Framework

The Collaborative Design (Co-Des) Framework aims to make ship powertrain design truly collaborative by automating the component model selection and simulation process. The four key components of the Co-Des framework are Component Description, Digital Design Template, an open-source platform for model sharing and a simulation Interface.

Component Description (CD)

Component Description (CD) is a document created by OEMs to share their component models. This document contains three components:

- A link to a standardized and IP-protected component model known as **Component Digital Twin**.
- Standardized component specifications describing the features of the component.
- Metadata describing basic details of the component such as its name, ID and manufacturer details.

Digital Design Template (DDT)

A Digital Design Template (DDT) is a document created by system designers to browse for suitable components and perform simulations from the selected components. This

document contains three components:

- A standardized powertrain simulation model template compatible with the component digital twins. The simulation model template would consist of the place-holder components along with inputs, outputs and connections connecting the outputs of one components with inputs of others.
- Standardized component requirements describing the requirements containing requirements for each component of the powertrain system.
- Metadata describing basic details of the powertrain such as its name, ID and system designer details.

Open-Source Platform

Autiosalo et al. [2] and Ala-Laurinaho et al. [3] proposed a platform to store a network of digital twins known as the **Digital Twin Web** (DTW) which is implemented using an open-source server called Twinbase. The OEMs would upload the CD documents onto the DTW. The DTW would be continuously updated with new component models as new manufacturers upload new component descriptions. The system designers, who need component models to perform simulations, upload their DDT document to DTW. The DTW would consist of an algorithm which would compare the component requirements inside DDT with the component specifications inside every CD document and filter out and present the suitable Component digital twins. The system designer selects a set of component digital twins required to form a full powertrain model. An automated simulation of the powertrain model would then be executed with a click of a button.

Simulation Interface

An algorithm would be used to automatically replace the stand-in components in the standardized powertrain simulation model template with the selected component digital twins. The simulation would then be carried out using a compatible simulation tool and the results would be presented to the system designers in the form of plots. If they are satisfied with the results, they would go ahead to order the components from the manufacturers to build a physical powertrain and if not, they can go back to DTW and select another set of component digital twins and repeat the simulations. If none of them produce satisfactory results, they can then contact manufacturers to provide custom component models. Figure 2 describe how the Co-Des framework can be utilized for collaborative powertrain design.

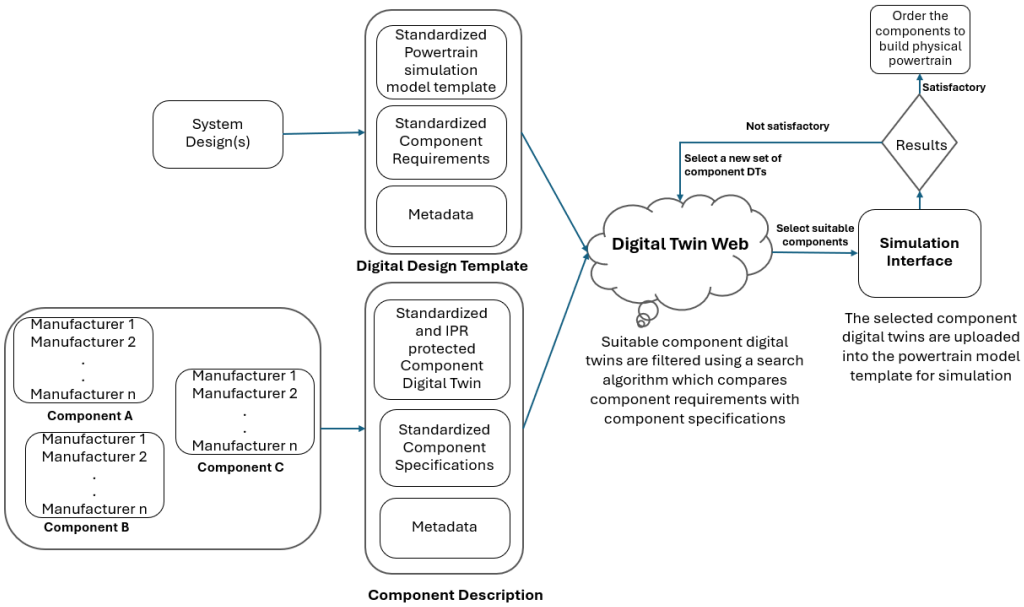


Figure 2: Co-Des framework for collaborative powertrain design.

2.2 Digital Twins

The idea of a "Digital Twin" was first introduced by Michael Grieves at the University of Michigan during a lecture on product lifecycle management (PLM). [4]. The term 'Digital Twin' gained further prominence when it was used by the National Aeronautics and Space Administration (NASA) in its integrated technology roadmap in 2010, where John Vickers used it as an alternate designation for the 'Virtual Digital Fleet Leader' (VDFL) [5].

Michael Grieves introduced the concept of digital twins, portraying them as intricate replicas of physical products, nearly indistinguishable in virtual space. Grieves identified three essential characteristics of a digital twin: physical objects positioned in physical space, virtual equivalents residing in digital space, and the complex network of data and information linking these virtual and real components. [6].

Figure 3 visually represents the concept of a digital twin. Here, data migrates from the real space to the virtual realm, which is gathered via various tools like sensors, lasers, and gauges. Conversely, information and processes flow from the virtual domain back into reality, derived from simulations based on collected data.

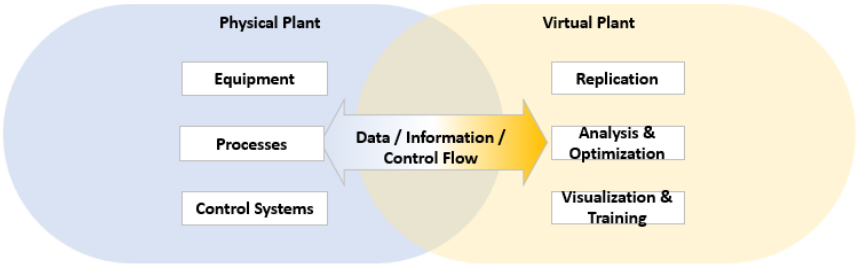


Figure 3: Concept of Digital Twin [6].

Digital twins have been adopted in various industrial sectors. The definition of digital twins varies based on the application and the field in which it is being used. In the present day, the three major purposes for using digital twins in any industry are for modelling and simulation, monitoring and controlling [7]. Modelling and simulation may involve applications which require the behaviour of physical assets to be replicated in the virtual space and perform simulations on them which allows optimization of products without the need to perform tests on real objects. Monitoring refers to the applications where the digital twins can be used to monitor the current state of physical assets, which helps maintain the physical object's performance and find and fix any faults. Controlling covers applications where the physical objects can be controlled using their digital twin, for instance, a robot arm, which might be controlled using its digital twin. This thesis focuses on utilizing digital twins in the maritime industry for modelling and simulation.

According to Giering and Dyke [8], the maritime industry could benefit from the use of digital twins in shipbuilding and shipping operations, which could combine suppliers, co-makers, and vessel owners throughout a vessel's lifecycle. According to Fonseca and Gaspar [9], digital twins have two distinct roles in the maritime industry. The first is decision support for ship operations, which focuses on the calibration of simulation models using actual operational data and condition monitoring. Coraddu et al. [10], for example, used a simulation model built with neural networks to predict the speed loss resulting from maritime fouling. The second type of digital twins are employed as staff training and system integration tools. A system for simulating a control system was reported by Tofte et al. [11], which links controllers for hardware testing and operation planning to a comprehensive simulation model of the lay tower clamp in a pipe-laying vessel.

Giering and Dyke explained how digital twins are used in different life-cycle stages of a vessel. They classified the life cycle into six phases: Conceptual design, basic and detailed design, construction and assembly, testing and commissioning, operation and decommissioning. Early Stage twins are utilized during the conceptual and detailed design phases for simulation and analyses. During the construction phase, Digital twin prototype is created which consists of all the necessary information for describing and producing a physical model from the virtual version [12]. Experimental digital prototypes are fully functional, goal-oriented virtual prototypes that are used during the testing phase for testing, operation, and efficient development. A Digital Twin Instance is kept connected to a physical product for its entire existence and functions as a stand-in for the real product during the maintenance and decommissioning phases. Figure 4 represents the specific digital twin concepts in relation to product lifecycle phases of maritime products.

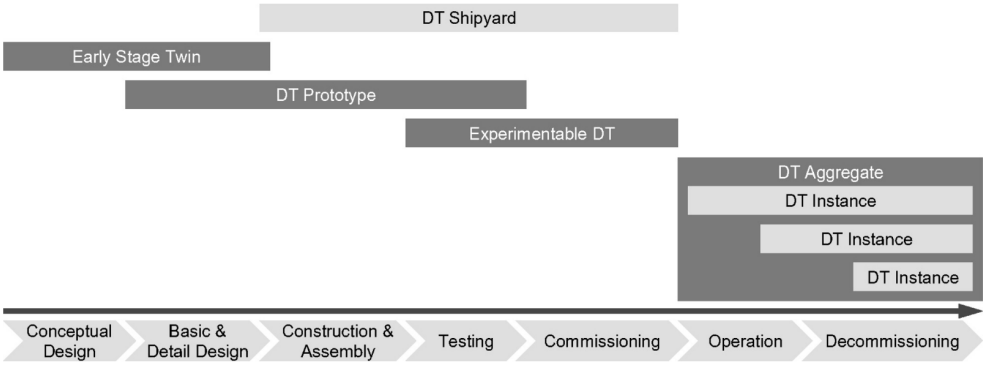


Figure 4: Concept of DT in maritime lifecycle [8].

The early stage digital twins provide a way to optimize the system design. Digital twins in the design stage can be used to evaluate the behaviour of a system even if a physical representation of it does not exist. This is achieved by simulating digital twins of components created using mathematical or data-driven models which represent the behaviour of individual components as a full system. This would help engineers to optimize system performance and automate component selection before the physical system is built which would reduce the high prototyping costs.

Giering and Dyke [8] suggested that an Early Stage Twin may serve as the primary tool for collaborative ship design. The current study aims to provide a framework to utilize digital twins in such a collaborative design of ship powertrains. The goal is to create standardized and IPR-protected digital twins of ship powertrain components to simulate the behaviour of the ship powertrain system.

In this thesis, the concept of digital twins is applied to the design of ship powertrains, where digital twins represent standardized and IP-protected virtual models along with the properties and metadata of physical components of a ship powertrain which are known as component digital twins. These digital twins are used by system designers who perform simulations at the design stage to choose the most suitable components to build their physical powertrain model. In order to describe complex systems such as ship powertrains, which consist of multiple such component digital twins, a few standardized description languages have been proposed which are presented in the next section which would help in automating the component selection during the design stage.

2.3 Standardized Digital Twin descriptions

The Co-Des framework aims to utilize a standardized Digital Twin Document (DTD) to describe the contents of the DTW. The document has to be in a format that would ensure seamless collaboration among digital twins created by different organizations. There are already standards available for describing the digital twins which are discussed below. These standardized descriptions could be a suitable candidate for describing the system structure as well as the requirements of the ship powertrain system.

Currently, the Digital Twins Definition Language by Microsoft Azure, the Web of Things Thing Description by the World Wide Web Consortium, and the Asset Administration Shell by Plattform Industrie 4.0 are the three most well-known standards available for characterizing digital twins [2]. Each of these standards will be studied in more detail below.

Asset Administration Shell

The concept of the **Asset Administration Shell** (AAS) was created by the German network of businesses, associations, trade unions, science, and politics known as Plattform Industrie 4.0 [13]. In addition to the current Internet of Things (IoT) and Embedded Systems (ES) subgroups of Cyber-Physical Systems (CPS), Industrie 4.0 developed a new subgroup called Industrie 4.0-components (I4.0-components) for creating mandatory industrial standards.

The definition of the I4.0 Component is given as “worldwide identifiable participant able to communicate consisting of Asset Administration Shell and Asset including a digital connection” [14]. A CPS must represent an entity in order to be classified as

an I4.0 component. The entity can either be a physical or a virtual asset [15]. Several new terminologies and definitions have been introduced in relation to Industry 4.0's realization and application. Asset Administration Shell is one such term that refers to the logical collection of available information and the administration interface for a particular asset within an organization[14].

AAS is a digital representation of a resource which can be a component, system or subsystem as a part of the digital twin. In AAS terminology, the resource is known as an 'Asset' [13]. The AAS makes the properties and capabilities of these assets available to be actively or passively interacted with by the digital world. It bundles the characteristics of the assets in sub-models which consist of structured information[16]. The submodel is a container which can include properties, files, operations, collections, etc. Figure 5 represents an Asset inside the AAS.

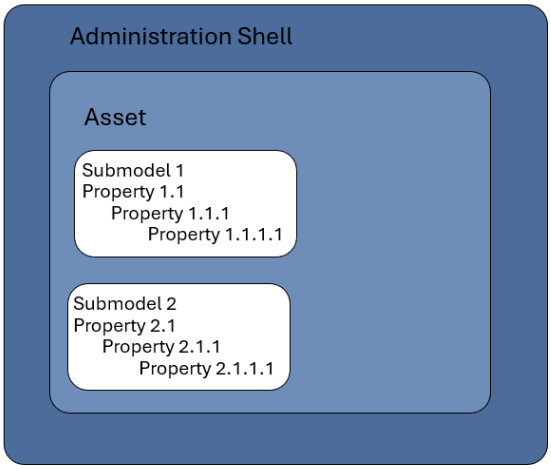


Figure 5: The structure of the Asset Administration Shell.

The goal of utilizing submodels as opposed to directly adding elements to the AAS is to enable submodel standardization. When referencing to external resources for semantic annotations, AAS is flexible because it supports many standard resource IDs, including IRI and IRDA [13]. AAS descriptions can be expressed in a number of formats, including AutomationML, XML, JSON, RDF, and the OPC UA data model.

Web of Things Thing Description

Thing Descriptions was introduced in 2016 by the World Wide Web Consortium as an ontology for the Web of Things which supports the architecture which relies on

established web technologies and RESTful interfaces for Internet of Things (IoT) [17]. It was made an official recommendation by the W3C in 2020. The Thing Descriptions are like building blocks which complete or enhance the existing standards. To represent the current IoT resource as a Digital Twin, they offer a meta-model called 'Thing'. The Thing is made up of properties, actions and events [13]. Figure 6 represents a Thing within the WoT.

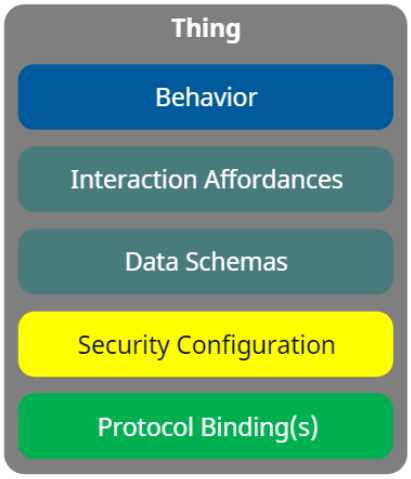


Figure 6: The structure of the Thing Description.

The Web of Things uses the Web as an interoperable platform for the communication and interaction of 'Things'. The examples of Things in this case may be anything from building rooms, manufactured products, mechanical systems or any other real-world or virtual entity. For WoT Things to exchange their data with other agents, they need to be able to describe themselves. Such a description is known as the Thing Description [17]. An additional definition for Thing Descriptions is a semantic resource that describes a distinct WoT thing that can be interacted with by a software agent.

Another important function of WoT Things is to provide a set of interactions to the web which correspond to their communication with the physical world. These interactions are web resources of arbitrary format which act as a digital replica of real-world entities such as natural phenomena like temperature change, object motion and on/off state [17].

Digital Twin Definition Language

Microsoft came up with the **Digital Twin Definition Language** (DTDL) to define their Azure Digital Twins. DTDL consists of a collection of meta-model classes that specify how each digital twin behaves. [13]. The main meta-model class is called the Interface which is the resource and consists of Commands, Components, Properties, Relationships and Telemetry. Figure 7 shows the structure of a DTDL interface.

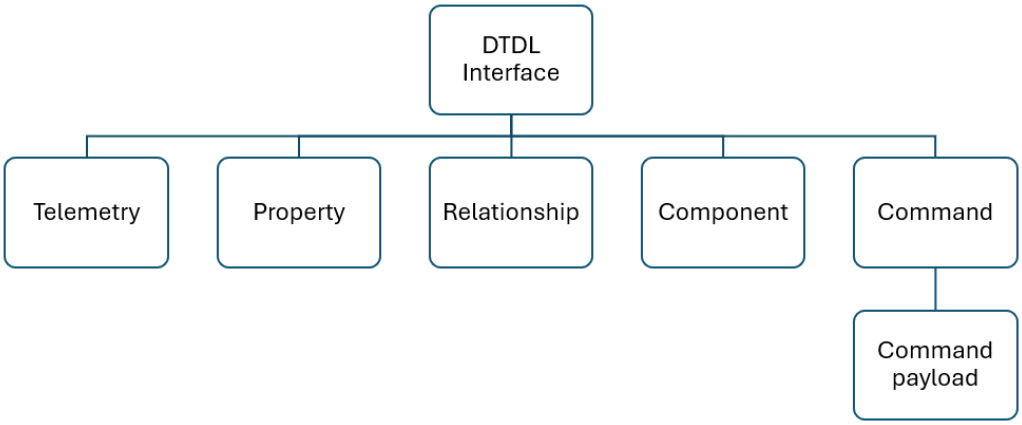


Figure 7: The structure of the DTDL interface

Commands correspond to the functions or operations that can be performed on any DT. The concept of components is comparable to the AAS submodel. The components allow 'Interfaces' to contain other interfaces. The connection between two digital twins or interfaces is defined by a 'Relationship'. The data stream that the digital twin emits is referred to as 'Telemetry' [13].

DTDL is used as an open-source standard with the Azure Digital Twins (ADT) service that enables communication with the products of other software providers [3]. ADT provides a graphical model editor and a web service, which allows users to create, read, modify or delete DTDL models.

DTDL employs a unique schema known as Digital Twin Schema Definition Language, designed to seamlessly integrate with widely used serialization formats like JSON, JSON-LD, and RDF. Devices can be connected to the Azure platform via Azure IoTHub utilizing JSON DTDL specifications, which several tools offered by this platform can leverage [18].

Each of these standards is particularly suitable for certain applications of digital twins. AAS focuses on industrial applications aiming to standardize the description of industrial assets and their properties for interoperability and integration in smart manufacturing environments. While AAS has rich semantics, it focuses only on high-level asset management and it is difficult to describe complex systems like ship powertrain. WoT TD is focused on describing IoT devices and their capabilities in a web-centric manner, enabling easier discovery, integration, and interaction of IoT devices over the web. Since most of the powertrain components are not IoT devices, this might not be a suitable method to describe powertrain systems. DTDL is specifically designed for creating digital twins, with features available to model complex systems and their components. It provides a comprehensive framework for modelling the digital representation of physical assets, making it well-suited for describing ship powertrains, their components and requirements within a digital twin environment.

Although DTDL is suitable for describing the powertrain components, the standard does not perform simulations from the descriptions. The ultimate goal of the Co-Des framework is to automate the simulations of the powertrain model using the descriptions of the ship powertrain structure which is not straightforward using these standardized descriptions. This standard can however be utilized for describing the metadata of the component digital twins inside the Digital Twin Document.

Another standardized method widely studied for describing complex systems is SysML. The following section delves into SysML.

2.4 SysML

SysML [19] is a popular system modelling language for Model-Based System Design. The Object Management Group (OMG) proposed SysML as a standard for modelling systems or system-of-systems [20]. It is an extension to the Unified Modelling Language (UML) specifically designed to facilitate system modelling, however, it can be utilized for systems engineering in general. SysML is a visual modelling language designed for large-scale, complex, and multi-disciplinary systems [21]. SysML offers discrete diagrams that can be used to explain the structure and components of a system, define design requirements and examine allocation policies that are important for system design [22].

SysML uses diagrams to describe the systems. These diagrams represent the system or system-of-systems in a multi-layer fashion to describe the behaviour, structure and

requirements of the system. The three core diagrams in SysML are the behaviour diagram, the requirements diagram and the structure diagram. The figure below represents the organization of the SysML diagrams.

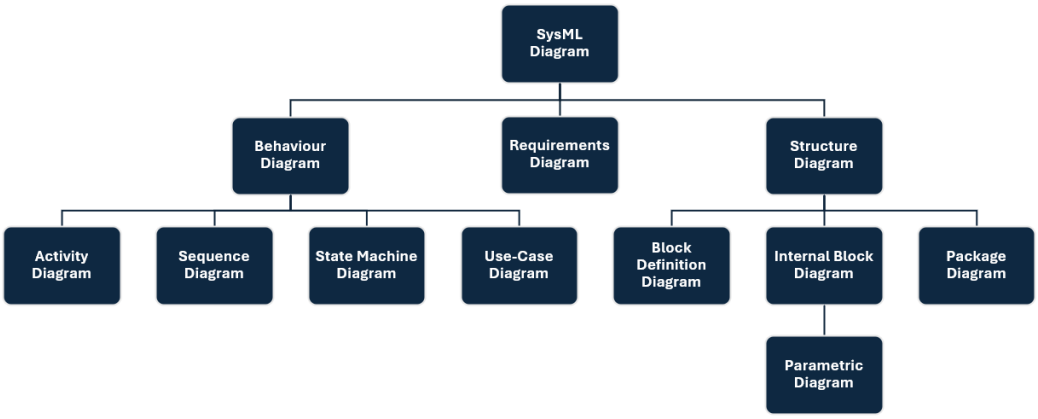


Figure 8: SysML diagram taxonomy [19].

The **Behaviour** diagram describes the dynamic behaviour of a system and its components. The behavior diagram is categorized into four types: activity diagram, sequence diagram, state machine diagram, and use-case diagram [23]. The Activity diagram captures the flow of actions within a system, including control flow, decisions, and concurrency. The Sequence diagram shows how objects interact over time, depicting message exchanges and lifelines, useful for understanding interactions and potential issues. The State Machine diagram displays the sequence of states that the system undergoes throughout its lifecycle, while the Use-case diagram provides a high-level description of the system's functionality. This thesis delves into the fundamental building blocks of ship powertrains, focusing on their structure and requirements rather than their intricate behaviour. Therefore, the key interest would be in studying the Structure and Requirements diagrams.

The system structure is depicted using three diagrams: Block Definition Diagram, Internal Block Diagram, and Package Diagram. The Block Definition diagram is used to create system components using blocks. Blocks are modular structural units that encapsulate the contents (Properties, Behaviors, Constraints) of the components. The block can be defined recursively as nested blocks representing the hierarchy of the system. After defining all the blocks of the system, the Internal Block Diagram

illustrates the interconnections among the components to enable the flow of information. The diagram defines the internal structure of a system, including properties, ports such as inputs and outputs, connectors, and interfaces. A Parametric diagram is a type of Internal Block Diagram (IBD) that uses mathematical equations defined by Constraint Blocks to govern the behaviour of the components of the system. The package diagram is often used when there are multiple diagrams. These diagrams can be organized into groups for easy identification.

The **Requirement** diagram is utilized to define the system requirements which specify the conditions that the system must satisfy. They are categorized into functional and non-functional requirements. Functional requirements are those which specify a particular function or task that the system needs to achieve. These requirements usually are defined to analyse the system output and test whether the system produces the desired output. The Non-functional requirements are used to define other requirements such as the physical design requirements, UI requirements and system performance requirements.

The SysML can be considered for describing the system structure and requirements for ship powertrains since it is standardized and is supported by multiple tools [21]. The Block Definition diagram is useful for describing the powertrain components such as the variable-frequency drive, electric motor, thruster, and propeller. The Internal Block diagram can then be created to define the inputs, outputs and the connections between the components to form the internal structure of the powertrain. The Requirements diagram can be defined to describe the simulation requirements and design constraints of the powertrain.

There are, however, challenges in utilizing the models created using SysML for simulations. Nikolaidou et al. [20] have presented the challenges in simulating the SysML models. In their study, they found methods to generate simulation code from the SysML diagrams by exporting the simulation-specific profiles in XMI format and consequently transforming them into simulation models using model transformation languages. However, they claimed that those methods were not standardized and not supported by many simulation tools. Niger et al. [24] noted that the profiles require frequent updates and lack interoperability.

The Co-Des framework aims to automate simulations of the component digital using the system structure description of the powertrain. However, using SysML directly for powertrain simulations presents challenges. Converting SysML models into

simulations is cumbersome, and while future versions like SysMLv2 may natively support simulation of the SysML models [24], it's unavailable during this study.

An additional limitation of SysML lies in defining component-level requirements. While its requirements diagrams excel at specifying system-level needs, SysML lacks specific mechanisms for detailing component requirements. This poses a significant hurdle as this thesis aims to find a way to browse suitable component digital twins through component requirements within the DTW. The thesis acknowledges the importance of system-level requirements for analysing the performance of the powertrain model, but they are excluded from the scope of this thesis.

The limitations of SysML for automated simulations and component-level requirements underscore the need for exploring alternative methods for capturing system structure and component specifications. The next section of the thesis presents an investigation into one such alternative which is ontology.

2.5 Ontologies

Ontological representation, also known as Ontology can be described as a detailed explanation of a particular concept [25]. In simpler terms, ontology is a formal way of representing knowledge or concepts and their relationships within a specific domain. In the system design domain, an ontology can be used to define systems and their properties, relationships, constraints, and behaviours [26]. There are several standards which are available for implementing ontologies and the most common standards are Web Ontology Language (OWL) [27], Resource Description Framework (RDF) [28] and JSON-LD [29]. These standards allow the ontologies to be both interoperable and machine-readable which makes it easier for the ship designers to create the system structure descriptions.

Ontologies have been used to describe system structures in previous literature. Durak and Ören [30] used the System Entity Structure (SES) ontology [31] to evaluate the fidelity of a flight simulator, which refers to the degree of similarity between the simulator and the real aircraft. The SES ontology offers a collection of components and principles for defining the structures of systems. The elements included entities which represented system components, the relationships between entities and the taxonomy of the entities.

McGinnis et al. [32] demonstrated the use of domain ontology to describe the structure and behaviour of systems in various domains, such as manufacturing, logistics, and

aviation. Banerjee and Sarkar [33] introduced Generalised Ontology Modelling (GOM) to represent structural, functional, and behavioral knowledge in domain-specific systems. The GOM defines the components of a domain-specific system and their relationships.

Ontologies and SysML are both helpful in describing system structures, but they share similar modelling challenges. An ontology serves as a conceptual model that defines the concepts, behaviors, and relationships of systems. These conceptual models cannot be used to simulate the systems and hence they have to be translated into computational models [32]. The ontology has to be integrated with tools which allow simulation based on the descriptions specified in the ontology. Several studies have been conducted on such translations where ontologies have been integrated with simulation tools. For instance, Durak and Ören [30] implemented the aircraft simulator ontology with MATLAB/Simulink to simulate their model. Silver [34] proposed a three-step approach for transforming a domain ontology into an executable model. Initially, concepts from domain ontologies are mapped to a modelling ontology to depict models as ontology instances. Subsequently, the ontology instances are translated into an intermediate XML markup language. Finally, executable simulation models are generated from the markup language representations of the models. However, this process is inefficient and demands substantial manual effort, making it time-consuming and difficult.

Applying ontology to specify the specifications and requirements of components is an additional consideration. A major issue in the collaborative design of a complex system such as a ship powertrain is to create a suitable representation of requirements and specifications for the components. It is common for component manufacturers and ship designers to use different terminology when describing component specifications and requirements, respectively. As a result, the same term may be applied to different concepts, while different terms may be used to refer to the same entity [26]. To address this challenge, a standardized representation of component specifications and requirements is essential.

Ontology offers a formalized method to achieve this standardization. An ontology can be defined using a specific set of terms which can be agreed upon by the partners involved in the collaborative design. Lin et al. [26] developed an ontology to describe the requirements for a desk spot lamp that contains multiple sub-systems and components. The ontology was divided into different classes, including part, feature, parameter, requirement, and constraint. A part refers to a component within the system, while a feature represents different aspects such as geometrical, functional, assembly,

mating, and physical features. Parameters define the properties of the components such as weight, color, diameter, material, surface finish, etc. The requirement is then defined using the part, feature and parameter by assigning constraints which are mathematical expressions such as less than, greater than or equal.

Turnitsa and Tolk [35] explained that ontologies can help in in finding and selection of components using ontology-based search in order to assist in automation of search and selection task. Silver et al. [34] utilized the Discrete Event Modelling Ontology (DeMO) to help identify and connect elements from a medical ontological representation and then help combine those elements with a discrete event system. Teo and Szabo [36] utilized the Component Oriented Simulation and Modeling Ontology (COSMO) to locate components that fulfil semantic requirements for Composable Discrete Event Simulation.

An approach similar to Lin et al. can be utilized in the Co-Des framework to describe the requirements for various components of a ship powertrain where a hierarchy of components, each with a set of parameters and constraints can be defined. Once the requirements are described, an algorithm can be developed to perform an ontology-based search to find suitable components from DTW similar to Turnitsa and Tolk.

2.6 Co-Simulation

The concept of collaborative simulation emerged to overcome challenges when simulating complete systems with complex subsystems and components developed in different environments [37]. Co-simulation allows individual component models created using different tools to be combined as a system and perform simulations by seamlessly exchanging data between them.

There are two approaches for combining the individual components into a full simulation model: tight coupling and loose coupling [38]. Tightly coupled systems refer to those where the various components or subsystems interact with each other frequently and directly, often in real-time or with minimal delay. In such systems, there is a high level of interdependency between the simulation models, requiring synchronized and coordinated execution to ensure accurate results. Conversely, loosely coupled systems involve components or subsystems that interact less frequently or indirectly. While tightly coupled systems offer more accurate and detailed simulations of closely integrated processes, loosely coupled systems provide flexibility and

scalability, enabling the integration of disparate simulation models with varying levels of complexity and fidelity. Co-simulation can be applied to both types of systems, depending on the application.

The Co-simulation approach is being adopted rapidly in the maritime industry for the simulation of complex subsystems using digital twins. Hatledal et al. [37] introduced a methodology for constructing a virtual replica of the Gunnerus research vessel using co-simulation. They undertook a case study to investigate the potential applications of digital twins for ship manoeuvring and onboard decision support. To establish a real-time twin of the Gunnerus vessel, they employed pre-recorded data encompassing several parameters such as longitude and latitude, wind speed and velocities from the Gunnerus vessel to simulate its behaviour.

Perabo et al. [39] modelled a standard AC ship powertrain using the digital twin approach. Furthermore, they relied on co-simulation to allow accurate simulation with digital twins. To accurately model the powertrain, the power stage, relevant local controllers and a high-level controller were included in the simulation model.

Sadjina et al. [38] introduced an open virtual prototyping framework (VPF) designed to enable users to develop reusable subsystem or component models and seamlessly integrate them into comprehensive system simulations. This framework serves various purposes including prototyping, verification, training, and performance analysis. The primary aim of the authors was to enhance communication among customers, designers, and product developers throughout the design process, ultimately aiming to enhance efficiency.

To ensure the smooth operation of Co-simulations, it becomes imperative to establish protocols and guidelines that foster a seamless integration of simulations originating from distinct tools and platforms. These protocols govern communication, synchronization, and data interchange, thereby facilitating the creation of intricate and precise simulation scenarios. These standards play a pivotal role in facilitating the integration and interaction of diverse models within a simulation environment.

The prominent standards defined for co-simulation include High-Level Architecture (HLA), Functional Mockup Interface (FMI) and System Structure and Parameterization (SSP) which are discussed further in more detail. Discrete Event System Specification (DEVS), Simulink S-functions and the C++ library SystemC are also considered standards for co-simulation [40].

2.6.1 High-Level Architecture (HLA)

The High-Level Architecture (HLA) was developed by The US Department of Defense in 1996 to address the challenges of reusability and interoperability in simulations which later became an IEEE standard for distributed simulation [41]. Distributed simulation refers to a simulation scenario where the simulation environment is distributed across multiple computing nodes or systems which can be geographically distributed or located within the same physical area [42]. In such scenarios, HLA acts as a message-oriented middleware which facilitates communication between software components in a distributed simulation environment.

The High-Level Architecture (HLA) is composed of three main functional elements. The first element comprises simulation systems that are modelled as Federates. These Federates can interact with other simulation models through data exchange. The second integral component is the Runtime Infrastructure (RTI), which provides a variety of services to facilitate interactions between Federates and support functions for managing Federations. Finally, the third component is the runtime interface specification, which establishes a standardized pathway for Federates to interface with the Runtime Infrastructure. This specification enables Federates to invoke Runtime Infrastructure services to facilitate runtime interactions between Federates and respond to requests from the Runtime Infrastructure [43].

HLA utilizes the publish-subscribe mechanism to update a shared object-oriented model that represents the simulation environment among distributed simulation components (known as federates). When a federate publishes updates to the shared object model, it notifies the HLA runtime infrastructure about the changes. The RTI then distributes these updates to other federates that have subscribed to receive updates for the same object or attribute. This ensures that all federates have a consistent and up-to-date view of the simulation environment. This means that the updates are not continuously shared with all the federates, moreover, a new RTI implementation is required for every update published. Figure 9 describes the working of the HLA standard based on the publish-subscribe model.

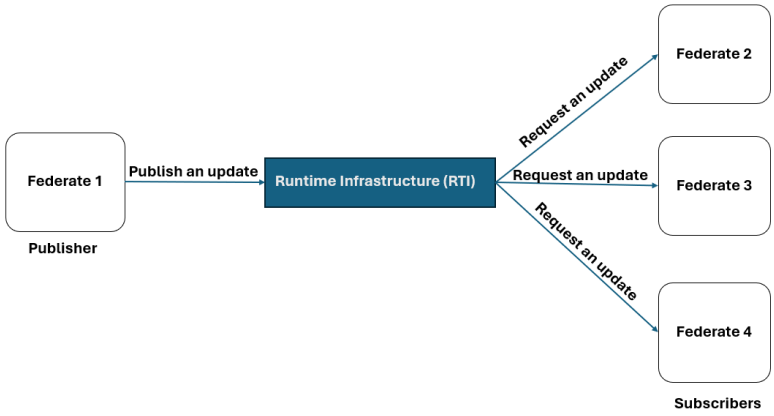


Figure 9: Publish-subscribe model of the HLA standard.

HLA was initially created for use in engineering, analysis, and training in the military and defence sectors. Petty and Gustavson [44] presented the use of HLA in combat modelling. HLA was used in battlefield simulations, combat trauma patient simulations and virtual flight simulations. Each of these mostly used HLA as an event-based interaction of essentially autonomous entities such as tanks, aircraft or individual humans as federates to continuously share location data and messages with each other.

Although HLA ensures interoperability among systems, there are certain challenges associated with it when it comes to modelling of complex systems such as ship powertrains. The primary challenges with creating HLA Federates are their complexity and resource requirements. This is not only due to the intricate nature of the IEEE 1516 family standards but also due to insufficient documentation and lack of readily available use cases [42, 45]. While HLA is a message-oriented middleware which can share data over a network, it cannot be used for model sharing. The standard does not allow sharing of the federates themselves over a network, rather it allows sharing of information between the federates. Another important drawback is the delay in sharing data. HLA introduces a delay in terms of communication and synchronization between distributed simulation components. Although HLA could be one option to perform simulation of ship powertrains without model sharing, it is not considered a suitable candidate since execution time is critical to execute simulations with several combinations of component models. Utilizing HLA would bring unwanted delays in simulation which would make the process highly inefficient.

2.6.2 Functional Mockup Interface (FMI) standard

The development of Functional Mock-up Interface (FMI) began in 2010 and was led by Daimler AG as part of the ITEA2 project MODELISAR [46]. After the MODELISAR project concluded in 2011, the Modelica Association took over the maintenance and further development of FMI [47].

The Functional Mockup Interface (FMI) is a tool-independent standard for exchanging dynamic models and co-simulation. The purpose of FMI is to facilitate model sharing for simulation between suppliers and OEMs, even when a large variety of tools are used [46]. FMI is a standard for connecting mathematical models of continuous-time dynamic systems represented as virtual components in the form of dynamically loaded libraries (DLL). The FMI standard is comprised of two primary components: FMI for Model Exchange and FMI for Co-simulation.

FMI for Model exchange (FMI-ME)

FMI for Model Exchange describes models through differential equations, algebraic equations, and discrete equations that include time, states, and step-events [47]. These equations are compiled into executable code as Dynamic Link Libraries (DLL) [46]. Once the model is imported into a simulation tool as a Functional Mock-up Unit (FMU), the Model Exchange interface exposes the ordinary differential equations (ODE) to a solver that is present inside the tool [48]. This solver is responsible for progressing time, defining states, managing events, and extracting and solving differential equations. Figure 10 represents the FMI standard for Model Exchange used for the simulation of two FMU models.

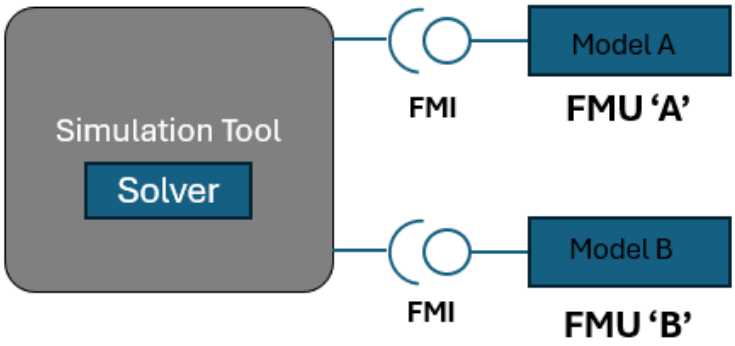


Figure 10: FMI for Model Exchange [48].

FMI for Co-Simulation (FMI-CS)

The FMI for Co-simulation works on the master-slave concept. In FMI for Co-Simulation, the model (FMU) is exported to the simulation tool along with its solver compiled into a runnable code [46]. The simulation tool acts as a master which coordinates with the solvers of individual FMUs which act as slaves. Slaves interact with the master using the FMI communication interface by exchanging inputs and outputs and synchronizing their simulation time with the master's simulation time. The master controls the overall simulation flow, including initializing the simulation, advancing the simulation time, exchanging inputs and outputs with the FMUs, and synchronizing the simulation time between the FMUs [49]. Figure 11 represents the simulation of two FMU models using the FMI for the Co-simulation standard.

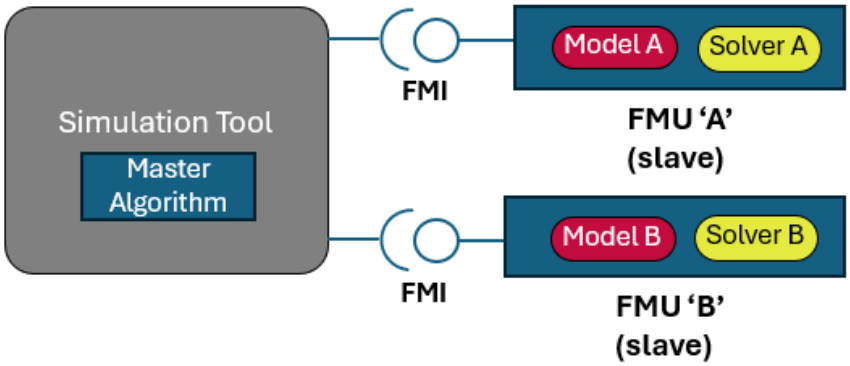


Figure 11: FMI for Co-simulation [48].

Functional Mock-Up Unit (FMU)

The FMI standard is implemented by the Functional Mockup Unit (FMU) component. A FMU is a Zip file with the extension ".fmu" that includes all the files needed to implement the FMI standard. [47]. An FMU contains the following components:

1. **The Model Description file** is an XML document that consists of metadata such as the name of the model, the type of FMU, the FMI version, and the definition of variables including inputs, outputs and parameters that are exposed to the simulation tool for simulation of the FMU [46]. This file is similar in both FMI-ME and FMI-CS.

2. Dynamic Link Libraries (DLL) are files which consist of executable codes required for communication of FMU with the simulation tool. In the case of

FMI-ME, the DLL contains the equations of the model compiled into executable code and in the case of FMI-CS, the DLL contains the simulation algorithm of the model which communicates with the master to solve the model. The FMI standard shares these interfaces in a binary format which helps hide details in the model [1].

3. An FMU consists of other files which might be specific to an operating system used for simulation. Figure 12 represents the generalized contents of an FMU zip file.

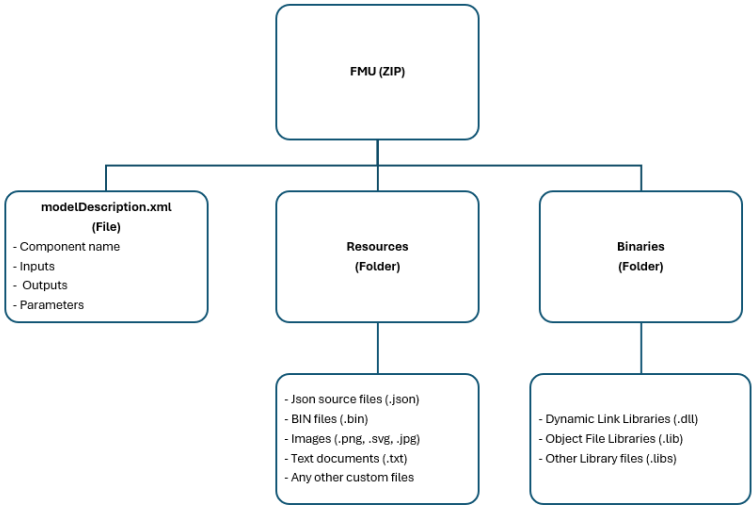


Figure 12: Elements of an FMU zip file.

The FMI standard is suitable for sharing standardized models of individual components. However, to combine these FMUs into full system simulation models and store the structure of the model including the FMUs themselves along with information on how the FMUs are connected, the System Structure and Parameterization (SSP) standard was developed as an extension to the FMI standard, which is discussed in the next section.

2.6.3 System Structure and Parameterization (SSP) standard

The System Structure and Parameterization standard [50] was proposed by the Modelica Association in 2016 and was publicly released in 2019 as an additional standard which complements the existing Modelica Language as well as the well-established FMI standard. The motivation was to use this new standard to help in exchanging system

structures seamlessly [51].

The SSP standard, like the FMI standard, is also a tool-independent standard. It is used for describing, packaging and exchange of system structures and their parameterization. The SSP standard can be utilized to define complete systems describing several interconnected FMUs with multiple inputs, outputs and parameters along with signal flow between them. For example, in Figure 13, a system consisting of three FMU components is shown whose inputs are denoted with 'u' and outputs are denoted with 'y' and the direction of signal flow are shown by the arrows between the FMUs.

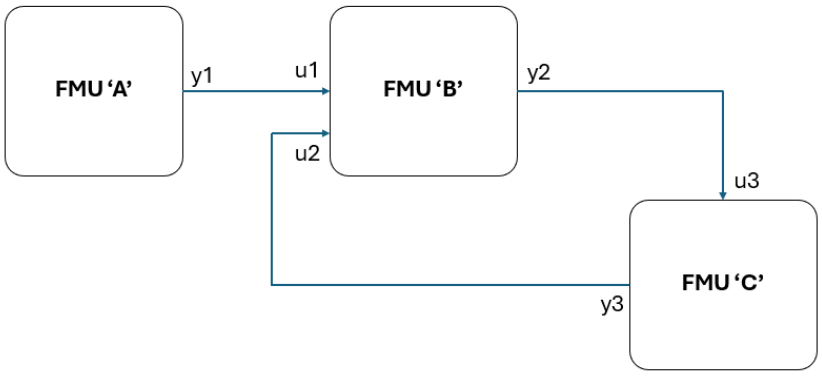


Figure 13: An SSP model created using three FMU components.

The SSP can be used to store all this information in a single document in a standardized format facilitating a more efficient workflow for system designers. This document eliminates the need for repetitive model creation for each simulation run. This information of inputs, outputs and parameters along with the signal flow of the simulation model is stored in an XML document known as the System Structure Description [52]. This XML document along with the FMUs used in the model are packaged as a zip file. The zip file consists of a resources folder to store all the referenced FMUs and other resources which may be necessary for the simulation of the SSP model depending on the operating system and the simulation tool being used [52]. Figure 14 represents an SSP zip file consisting of two FMUs. The figure also describes the contents of the SystemStructureDescription.xml file.

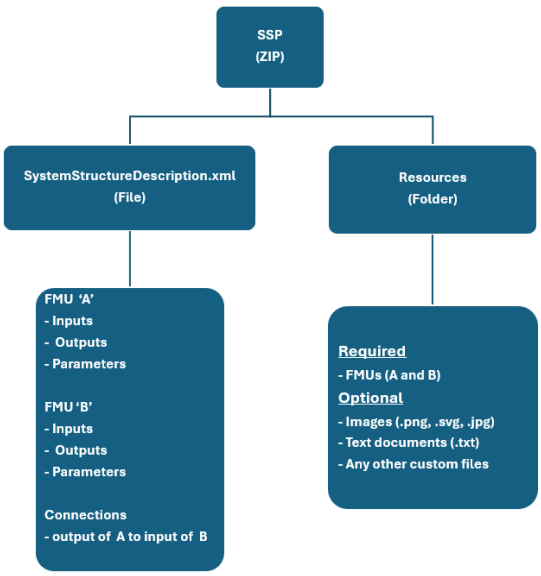


Figure 14: Structure of an SSP zip file.

The SSP file might also contain other documents such as the System Structure Parameter values (.ssv) which has the data of multiple parameter sets for different simulation runs and System Structure Parameter Mapping (.ssm) which contains the mapping of the parameter sets to the system or component parameters during each simulation run [51]. These are included inside the resources folder.

The design of the SSP standard is based on three basic principles which are tool Independence, Simplicity and Maximum Reusability. The goal of SSP is to allow partial or complete simulation models between any tools. There would not be any tool-specific data stored inside SSP by default but if required, they can be added as resources. The SSP is made of a very simple XML-based text which is human readable. It consists of only the data about the components and simulation-specific information and some graphical information for using it in a Graphical user interface. Since all the required files are packaged as a zip archive, it can be used as a template which can be reused across different tools, applications and scenarios.

There are several use cases for the SSP standard. One such application lies in designing systems with interchangeable components. This functionality allows assigning "stand-in components" with designated connectors for inputs, outputs, and parameters. These stand-ins prove valuable when system designers aim to define the systems with fixed connections and signal flow. This approach helps them to explore various component combinations within the established framework for simulations

The SSP can store multiple parameter sets for the same model. This makes it easy to run multiple simulation scenarios with the same SSP model using different sets of parameters each time instead of creating new models each time for different parameters. The SSP once created, is a ready-to-simulate file, which can be reused multiple times without needing to create or modify the model.

PMSFIT is currently developing a layered standard on top of the SSP standard known as the SSP Traceability Specification to support traceability of simulation tasks [53]. Traceability refers to the ability to track and understand the complete information and decisions related to a simulation. Traceability would allow for collecting relevant information to reconstruct how simulation results [54]. The information can include defining problem statements and objectives, specifying the requirements, specifying the design, implementation, execution, evaluation and verification. It is essential to document these simulation tasks transparently since design decisions often need to be made based on the simulation results. This is essential not only to make reliable design decisions but also to be able to understand design decisions retrospectively.

The SSP standard can be considered a suitable candidate for the description of the ship powertrain simulation model since it is compatible with the FMI standard and can be utilized to create a simulation model template and use it to automate the simulation of the component digital twins from DTW. In order to create the SSP models and perform simulations, suitable tools need to be selected. The tools supporting these standards are presented in the next section.

2.6.4 Co-Simulation Tools

The FMI and SSP standards are supported by several commercial and open-source tools. The support for the FMI standard is greater since it was introduced much earlier than SSP. The tools which support both standards are shown in table 1. Each tool has a different user interface and modelling language. A few of the most significant tools are discussed in the subsequent section. It is important to note that this thesis only utilized open-source tools to ensure accessibility, affordability and neutrality.

Name	Vendor	License	FMI support	SSP support
SYNECT Model Management	dSPACE	commercial	X	X
OMSimulator	OpenModelica	free	X	X
Model.CONECT	AVL	commercial	X	X
FMI Bench	PMFS	commercial	X	X
easySSP	eXXelent solutions	free + commercial	X	X
Simcenter System Architect	Siemens	commercial	X	X
Simcenter Studio	Siemens	commercial	X	X
Dymola	Dassault Systemes	commercial	X	X
libcosim	Open Simulation Platform	free	X	X
PyFMI	Modelon	free	X	
FMI Kit	Dassault Systemes	free	X	
FMI Toolbox	Modelon	commercial	X	

Table 1: Tools supporting FMI and SSP standards [37].

OpenModelica Simulator

OpenModelica Simulator is an open-source tool that supports the FMI and SSP co-simulation which is a part of the OpenModelica Tool Suite [55]. It is integrated into OpenModelica but is also available as a standalone library. It supports the FMI standard for both import and export. During the export of a modelica model into an FMU, the modelling tool automatically generates the model description file, which is an integral part of the FMU when it is exported. The OMSimulator is an integrated tool that supports both Model Exchange and co-simulation FMUs. In the case of Model Exchange, the tool generates the c code for the model which can be utilized by other simulation environments and in the case of Co-Simulation, it provides a master algorithm for coupling the simulation tools within a simulation environment [55].

OMSimulator is created using C-API in the form of a simulation library. It provides users with a command line and scripting interfaces with Python and Lua. Using these interfaces, the OMSimulator can be easily integrated into various third-party tools and applications. It is also provided with a graphical user interface known as OpenModelica Connection Editor (OMEdit) for component-based modelling. The graphical interface is developed using C++ and supports the OpenModelica Standard

Library. The OMEdit provides a simple user interface to create models or use standard models available in the OpenModelica Standard Library [56].

OMEdit can also be utilized to create SSP models using FMUs. The SSP can be created using either a weakly coupled system or a strongly coupled (also referred to as tightly coupled) system. An SSP model can be created by connecting the imported FMUs and simulated as a system and the model can be saved locally as a .ssp file. The tool automatically stores all the necessary files related to the SSP model. The results of the simulations can be visualized as plots inside OMEdit and the variables can also be exported to a CSV or a MATLAB file.

Open Simulation Platform

The Open Simulation Platform (OSP) [57] was introduced as a Joint Industrial Project (JIP) by DNV GL, Kongsberg Maritime (formerly Rolls-Royce Marine), SINTEF Ocean, and Norwegian University of Science and Technology (NTNU). OSP was developed as an alternate solution to the existing co-simulation platforms like FMIPy and FMIGo! and OMSimulator to support SSP 1.0 and enable distributed execution of FMUs [58].

The Open Simulation Platform was developed to establish a digital-twin ecosystem within the maritime industry to facilitate efficient and safe model sharing and co-simulations [59]. The key objective of the OSP is to streamline the construction of digital twin marine systems, thereby simplifying the process of planning, manufacturing, assembly, commissioning, and operation of complex integrated systems [58].

The Open Simulation Platform has a Core Simulation Environment (CSE) that contains Libcosim (C/C++ library), a web-based graphical interface, a command-line interface, a model interface validator, and a Java wrapper [59]. Libcosim is one of the main components of OSP. It is a C/C++ cross-platform library which links the FMU models together into system structures using the SSP standard [59]. Moreover, Libcosim provides a robust API which end users can incorporate into their own higher-level applications [58].

The demo application was originally created to demonstrate a basic simulation application that makes use of Libcosim's functionality and a straightforward graphical user interface; nevertheless, it is now a functioning application that can be used for real-world scenarios. The command-line interface can be utilized to execute the

co-simulation. Model interface validator verifies that the simulation model conforms to the Marine System Model Interface (MSMI). The Java wrapper was developed so that Libcosim could be used within Java applications [59].

OSP supports both the FMI 1.0 and FMI 2.0 co-simulation standards. However, OSP only supports Co-Simulation FMUs. While Model Exchange models are not directly supported, they can be transformed into Co-Simulation models by employing the relevant standalone tools.

One of the downsides of the OSP are that it is only a simulation platform which means that it cannot be utilized to create FMU and SSP models. It requires the SystemStructure.ssd file to be uploaded in order to perform the simulation. This would make OSP a less prominent option since SSP modelling is a key aspect of this thesis. Moreover, OSP consists of only one master algorithm for solving the simulations which is a fixed-step algorithm. Applications which require variable step simulations cannot utilize this platform. Moreover, the graphical user interface is not completely developed. It does not have proper error-handling tools to help users resolve the errors during simulations. Although the Open Simulation Platform is developed for maritime applications, these drawbacks make it less suitable to be adopted within this thesis but can be a prospect if the platform is further developed.

Other tools

EasySSP [60] is a cloud-based simulation architecture and co-simulation platform developed by EXXcellent Solutions for system validation based on the FMI and SSP standards. The tool has a model editor which allows users to create systems, define parameters, and integrate multiple FMUs and SSPs. The simulator used is developed by PMSF IT consulting and can execute and validate simulations based on FMI and SSP. It also consists of a traceability editor which can be used to document the simulation process as a part of the SSP package. A few of the features are open-source and others require a fee to be paid.

Dymola [61] is a modelling and simulation tool developed by Dassault Systemes. Dymola has also extended its support for FMI and SSP standards. Dymola supports the import and export of FMUs and the creation of SSP models. The models can also be simulated using Dymola. However, this tool is out of the scope of this thesis since it is a commercial tool.

MATLAB/Simulink also supports the FMI standard, however, it requires external

libraries. Dassault Systemes has developed an open-source Simulink library called FMIKit to import and export FMUs [62]. Modelon is another organization which has developed an FMI toolbox for MATLAB/Simulink [63] which has more features when compared to the FMIKit but is not open-source. These libraries allow Simulink models to be converted and exported as FMUs and also import FMUs for simulation. However, there are no libraries developed that support the SSP standard.

The literature review found four methods to standardize ship powertrain system descriptions: standardized digital twin languages, SysML, ontology and the co-simulation standards. While the former three methods were quite suitable to describe complex systems and their requirements, implementing them as simulation models was found to be difficult. The SSP standard was found to be a suitable option for describing the system structure of the ship powertrain as well as utilising it to simulate the component digital twins (FMUs) from the DTW. The reason being the SSP standard is created on the fundamentals of the FMI standard which means that the component digital twins can be easily integrated with the SSP powertrain model to perform simulations. Moreover, the SSP standard allows multiple documents to be packaged as a resource inside one single file which makes it easier for ship designers to manage all the documents related to the ship powertrain design. The standard is also open-source making it very feasible.

The literature review presents the SSP standard as a suitable candidate. However, in order to test its applicability in the field of ship powertrains, a case study was performed where the SSP standard was employed to create and simulate a simple ship powertrain model. The case study is discussed in more detail in section 3 below.

3 Methods

The comprehensive literature review presented the SSP as a possible candidate to describe the system structure in a standardized format. To test the potential of the SSP standard for automating the component selection and powertrain simulation within the Co-Des framework, a case study is presented in this section. Figure 15 gives an outline of the case study to implement the SSP standard and VSS Ontology within the Co-Des framework.

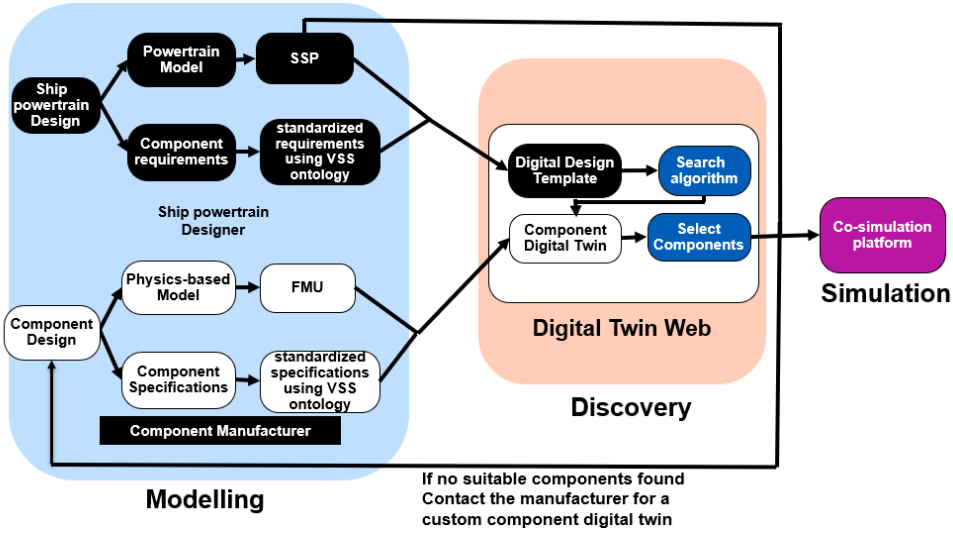


Figure 15: Co-Des framework for collaborative ship powertrain design.

The automated ship powertrain design consists of the following phases: Modelling, Discovery and Simulation which are presented below:

The first stage is **Modelling**. In this stage, the component manufacturers who supply various ship powertrain components such as variable frequency drives, electric motors, thrusters and propellers create physics-based models of their corresponding components which accurately describe the physical behaviour of their components. These models can be created using any of the modelling tools. The next step would be to convert the model into a standardized FMU model which would ensure the IPR protection of the models. The manufacturers also define the specifications of their components in an XML format using the VSS Ontology. Once these are ready, the FMU model and the component specifications are uploaded into the Digital Twin Web as a Digital Twin Document which also consists of the metadata of the component such as its name, description and supplier details. The DTW is always updating as new manufacturers upload their component digital twins.

The ship powertrain designers use tools to create SSP description of the ship powertrain. The SSP model consists of stand-in components with inputs, outputs, parameters and the connections defined. The next step is to define the component requirements which is done in the form of an XML document using the standardized terms specified in the VSS ontology. This XML is uploaded as a resource into the SSP model.

The next step is **Discovery**. In this stage, the SSP model is uploaded into the DTW and instantly, a search algorithm identifies the XML document inside the SSP and compares the requirements defined with all the component specification documents present inside DTW and outputs the suitable components. When ship designer selects the FMU models of all the components required by the powertrain, they would automatically replace the stand-in components defined in the SSP model. Figure 16 shows the flowchart of how the suitable components are discovered. The working of the algorithm is explained in more detail in section 3.2.1.

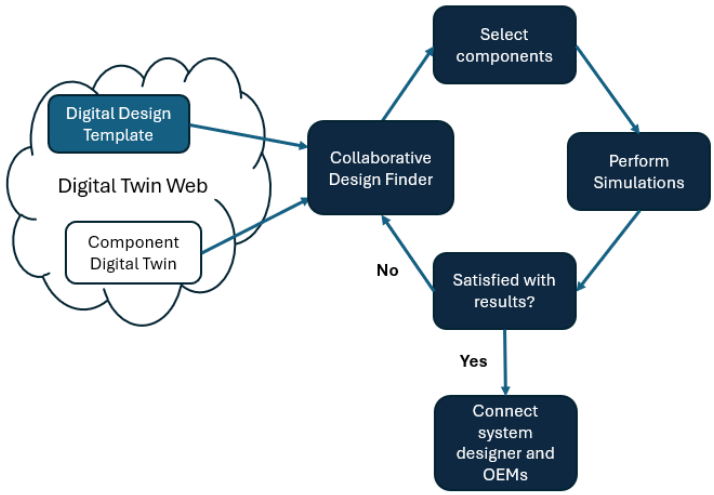


Figure 16: Flowchart for the discovery of suitable components from Digital Twin Web.

The final step is **Simulation**. The DTW would be linked to the co-simulation platform developed in this thesis. The SSP model is then simulated and the results are plotted. If the designer is satisfied with the results, they can approach the individual components manufacturers to order the specific components to build the physical powertrain. In case they are not satisfied with the results, they can choose a different set of components from DTW and repeat the simulation.

The case study applies this framework for designing a simple ship powertrain model containing three simple components. From the experiences of the case study, the SSP standard is evaluated qualitatively based on four criteria: Ease of use, interoperability, modularity and accuracy. Based on the qualitative analysis, a conclusion is presented on the use of the SSP standard in the simulation of ship powertrain.

The case study is organized into six distinct sections. In Section 3.1.1, the creation of the physics-based model for the ship powertrain using OpenModelica is presented. Section 3.1.2 focuses on exporting the physics-based model to a Functional Mock-up Unit (FMU), outlining various export options. In Section 3.1.3, the process of creating an SSP model for the powertrain using the EasySSP modelling tool is presented. Section 3.1.4 introduces a standardized approach for specifying requirements for ship powertrain components. The subsequent Section 3.2.1 illustrates how these requirements can be utilized to identify suitable FMUs from an extensive database. In Section 3.2.2, the replacement of components within the SSP models with the chosen FMUs based on specified requirements is explained. Finally, Section 3.3 outlines the execution of SSP simulation.

3.1 Modelling

3.1.1 Physics-based Modelling

For any simulation, the first step is to create virtual models of the components to be simulated. The models are usually created using a set of mathematical equations which define the system. These equations are usually linear and non-linear differential equations. For modelling such components, several tools such as MATLAB/Simulink, Dymola and OpenModelica are available.

In this thesis, the OpenModelica software is utilized for creating the physics-based models of the components of a ship powertrain. A simple ship powertrain with a Variable-Frequency Controller, a motor and load are modelled. The Variable Frequency Controller (VFC) controls the angular speed of the electrical motor by adjusting the frequency of the electricity that is supplied to the motor. Regular AC power has a fixed frequency, which determines the speed of the motor. The VFC takes in this AC power and converts it to DC electricity and then adjusts the frequency of the DC electricity before converting it back to AC. This change in frequency causes a change in motor speed.

The electric motor is a device which converts electrical energy into mechanical energy.

The electric motor is responsible for generating the power to propel the ship. The load in the ship powertrain is the amount of work that the motor needs to overcome to propel the ship. This could include factors such as friction, water resistance, air resistance, and other external forces acting on the ship.

These components can be modelled in the OpenModelica Connection Editor, which is a graphical user interface either using the pre-defined blocks from the Modelica Standard Library or by creating custom component models by defining the component using the text-based Modelica Language.

The standard library encapsulates common physical components, such as electrical elements, mechanical components, thermal elements, fluid components, mathematical operations, etc. These pre-built classes have a standardized structure and interface, allowing for easy integration and exchange of models among different simulation environments that support the Modelica language. The required components can be found by typing the name of the class in the Filter Classes box. The graphical editor has a diagram view where the required components can be dragged and dropped onto the editor canvas and connections can be established by drawing lines between them, mimicking the physical or logical relationships in the real-world system.

OpenModelica Connection Editor uses Modelica, an object-oriented modelling language specifically designed for the modelling and simulation of complex physical systems, and the relationships and interactions between components in a simple and modular way. All the pre-defined components in the standard library are also written in the Modelica Language. The text view in the editor allows one to view and modify the code written for a particular component using the Modelica Language.

The components for the ship powertrain were modelled using both the pre-defined standard library components as well as the custom components defined using the Modelica Language. A component is usually defined using input signals, output signals, parameters and sometimes mathematical equations which define the relationship between the input, output and the parameters.

The Variable Frequency controller was modelled using three pre-defined blocks of Trapezoid, VFController and toSpacePhasor blocks. The trapezoid block is a source which outputs a signal according to the defined parameters.

The output of the trapezoid block was connected to the input of the three-phase VfController block which considers the signal as the frequency. The Vfcontroller has two parameters which are VNominal and fNominal which are nominal RMS voltage

per phase and nominal frequency respectively. The VfController block outputs voltage signals in the form of three-phase sine-waves whose amplitude is linearly dependent on the input signal which is the frequency but is limited by the nominal RMS voltage per phase.

This voltage signal was then connected to the input of the toSpacePhasor block which converts the three-phase voltage signal into a space-phasor signal. The block produces a two-element vector as its output, with the first element denoting the real part of the space phasor and the second element its imaginary part. The output from the toSpacePhasor block is connected to the Electric Motor block. The Variable Frequency Controller component model consisting of the three blocks is shown in Figure 17.

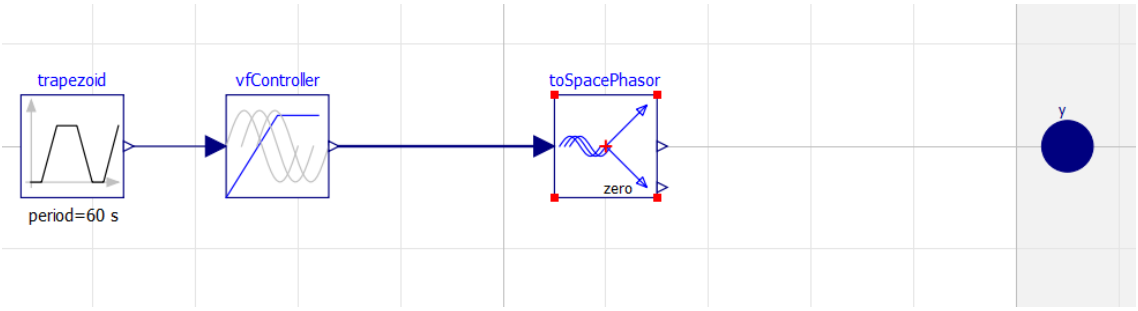


Figure 17: Variable Frequency controller model of the ship powertrain.

The electric motor component was modelled based on an induction motor for torsional vibration analysis of electrical drive trains presented by Holopainen et al. [64] using a custom block where the inputs, outputs, parameters and equations were defined using the Modelica language. The block has two inputs: the first is the vector output with two elements from the toSpacePhasor block and the other is the load torque which is input from the Load block. The electric motor is made of two parts which are the stator and the rotor. The parameters for each part of the motor are defined in table 2.

Parameter	Description	Value	Unit
p	Number of Poles	6	
Rs	Resistance of Stator	0.24141	Ohm
Rr	Resistance of Rotor	0.28808	Ohm
Ls	Inductance of Stator	0.1720	H
Lr	Inductance of Rotor	0.1705	H
Lm	Mutual Inductance	0.1705	H
J1	Moment of Inertia	52.5	Kg.m^2
J2	Moment of Inertia	1.5	Kg.m^2
K12	Linear Torque Coefficient	3.4e6	N.m/rad
α_1	Damping coefficient	0	

Table 2: Parameters for the electrical motor

These parameters were used to define a set of electromagnetic and mechanical equations of the induction motor. These differential equations then convert the voltage signal into angular velocity of the shaft. The equations of the induction motor are presented by Holopainen et al. [64]. From this set of equations, the output current is calculated which is then used in the mechanical equations to calculate the output angular velocity.

The output angular velocity was then connected to the load component. The Load component takes the angular velocity (w) as input and outputs the load torque which is connected back to the electric motor. The Load block has four parameters which are torque, moment of inertia, and two coefficients. Two test cases are considered in this case study. In the first case, no load is applied on the motor so all the parameters are considered to be zero. In the subsequent case, a load moment of 20 $kg.m^2$ is applied. The parameters for each part of the load are defined in table 3.

Parameter	Description	Value	Unit
T	Constant torque component	0	Nm
J	Moment of inertia	20	kgm^2
K1	Linear coefficient	0	-
K2	Quadratic coefficient	10	-

Table 3: Parameters for the Load.

The parameters are used as an input to the equation to calculate the output torque. Once all the components were modelled, the inputs and outputs of all the components were connected to form the full ship powertrain system.

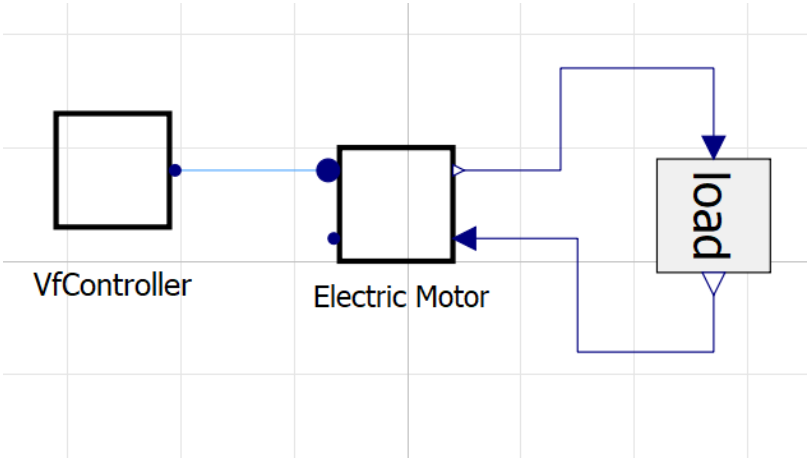


Figure 18: Physics-based model of the ship powertrain with the VfController, Electric Motor and the Load components.

3.1.2 Generation of FMUs from physics-based models

The physics-based modelica model was subsequently converted into Functional Mock-up Units (FMUs). Open Modelica provided an option to export the model as FMU for Model Exchange or Co-Simulation model or a combination of both. Based on the selected type, the required files such as the binaries and c source code files along with the model description file are generated. The solver for co-simulation can also be selected. OpenModelica currently provides two solvers which are Explicit Euler and CVODE. An explicit Euler model can be used if the model has a non-stiff behaviour and larger step sizes can be used and the CVODE solver can be used when the equations of the model are stiff and require smaller step sizes.

Another key feature that the OpenModelica Editor provides for FMI export is Model Description Filters which gives flexibility in exporting the variables within the equations of the models. There are four options available to modify the level of exposure the of variables during the export which are listed below.

- **none**: Applying this filter makes all of the model's variables visible, including those added by the symbolic transformations. This is mostly for troubleshooting.
- **internal**: By using this filter, all model variables including the protected ones would become visible. However, variables added by the symbolic transformations are filtered out.
- **protected**: This filter exposes all the public variables while hiding the Internal and protected variables.
- **blackBox**: This filter only exposes the interface. All other variables are either hidden or exposed using protected names.

This thesis utilized the protected filter to protect the internal and protected variables from being exposed. It is important to conceal the variable names connected to the equations because although FMU hides the equations behind the model, there is still a possibility to trace the equations from the variables using reverse engineering. The blackBox filter was not used since it hides all the variable names which would make it difficult to identify the required variables while plotting the simulation results. All the options provided by OpenModelica for exporting FMUs are shown in figure 19.

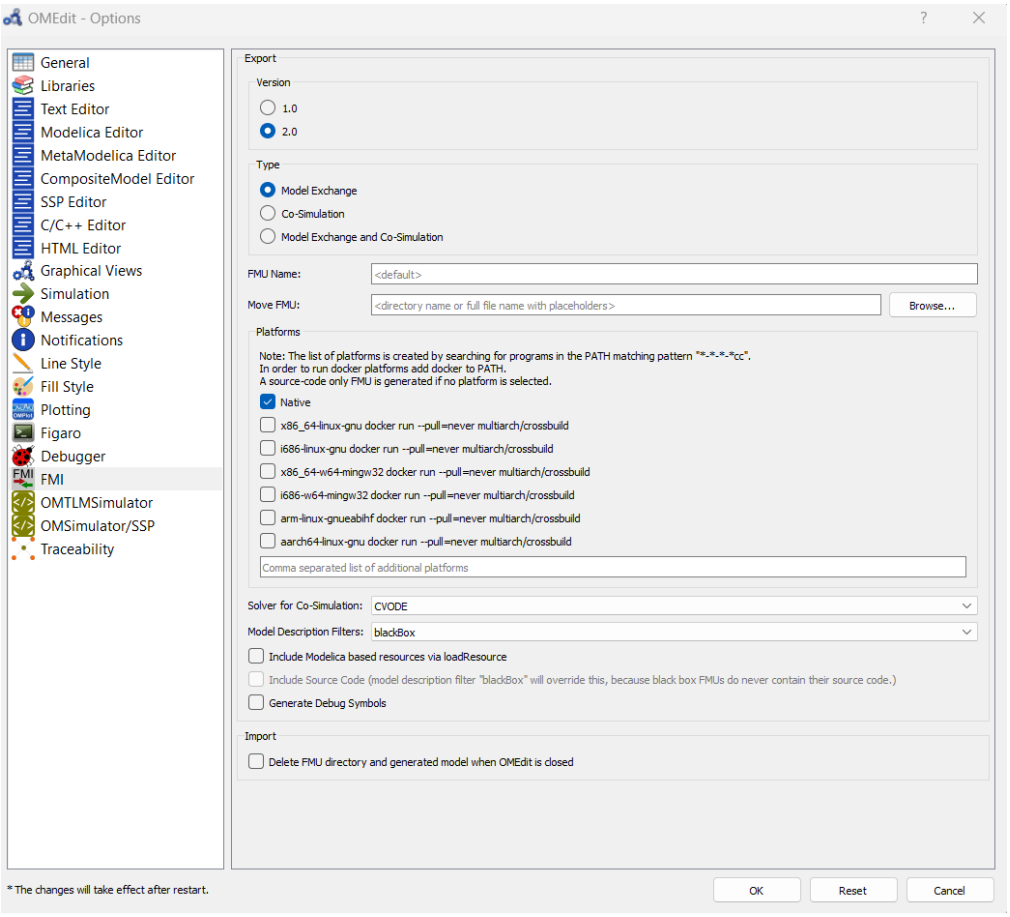


Figure 19: FMI Export options provided by OpenModelica.

In this thesis, the physics-based components of all three components were exported to FMUs using the FMI2.0 standard and were exported as both Model Exchange and Co-Simulation FMUs to study the differences between the simulation results. The Explicit Euler solver was used by default since the time step for the solver was not crucial.

3.1.3 Powertrain Modelling using SSP

In this thesis, EasySSP [60] was used to create the SSP model of the powertrain. EasySSP by Exxcellent Solutions is an open-source tool which is far more intuitive than OpenModelica for creating SSP models. The simple graphical user interface of EasySSP simplifies the process of building complex system models by using a simple drag-and-drop method to add components to the system and establish connections between them using lines or arrows.

EasySSP can be used to create a hierarchy of systems. Multiple sub-systems consisting

of several components can be connected to form a complex system. For each system, EasySSP allows us to define units, enumerations and resources for an SSP model. All the units for the parameters of the system can be defined in the Units section. While Enumerations can be a list of items or actions, and Resources are any other files or documents such as images or text files which can be uploaded and stored inside the SSP zip file.

At both the system and component level, the properties, connectors and parameters can be defined. The properties include the name of the component, the type of component which can be either an FMU or another SSP model which describes a subsystem, the source of the component where the actual FMU source can be uploaded, the implementation type which is either Model Exchange or Co-simulation and the description of the component as shown in figure 20.

Figure 20: Component Properties defined using EasySSP.

EasySSP also allows to define the connectors for each component. Connectors are defined to connect one component with another. The connectors can be an input, output, InOut, parameter or a calculated parameter. The connectors can also be assigned the units which have been defined in the Units section. Parameters for the components can also be defined for each component which would be required for the simulation.

Once all the required fields are filled, the components are created and the connections between the components are defined, the model can be downloaded as a .ssp file which consists of all the required files stored inside. It consists of the System Structure Description file which contains all of the information of the system structure including component names, inputs, outputs and parameters. It also consists of all the resources for the components.

For this thesis, the SSP model of the ship powertrain was created using EasySSP similar to the physics-based model created using the OpenModelica. The VfController, Electric Motor and Load components were modelled and their inputs and outputs

were defined and the connections between the three components were established as shown in figure 21. These components were considered to be stand-in components since actual FMUs were not used to create the model. The model was then saved as powertrain.ssp file. This SSP model would contain a basic structure of the powertrain which would then be utilized to find suitable FMUs based on the defined model requirements which would be discussed in more detail in the next section.

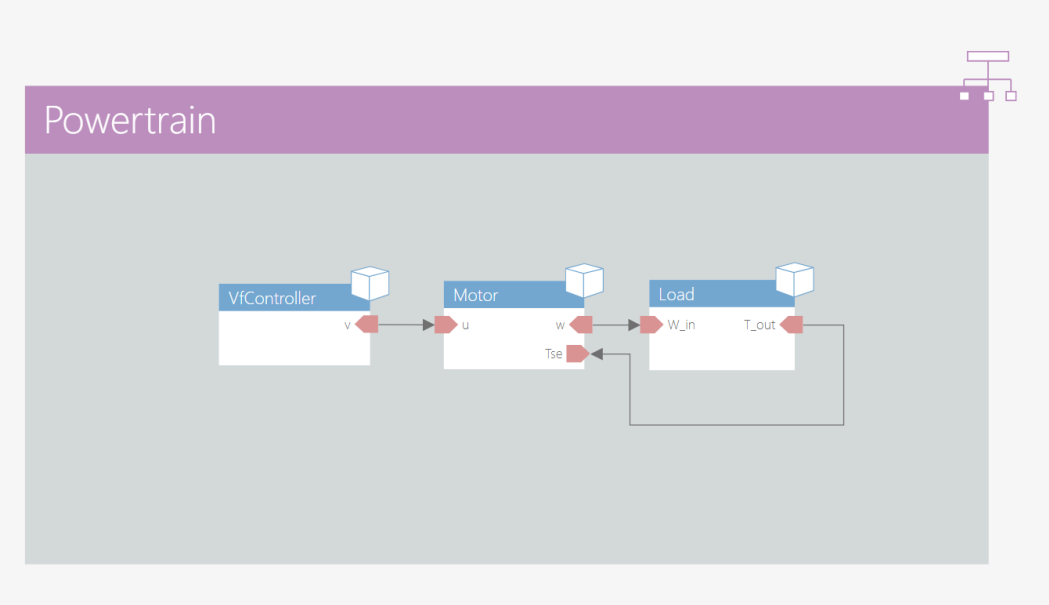


Figure 21: SSP model of the ship powertrain created using EasySSP.

3.1.4 Standardizing Model Requirements and Component Specifications

In this study, an ontology was applied to describe the static requirements of the powertrain components since the goal is to filter the suitable component digital twins from DTW. To define the standardized powertrain requirements and component specifications, the Vehicle Signal Specification Ontology (VSSo), developed by W3C was utilized [65]. The VSS Ontology provided a standardized and structured framework for defining and representing vehicle signals. While initially developed for automotive applications, VSSo's modular and extensible nature made it adaptable to other domains, including ship powertrains.

The terms specified in the VSSo were utilized to specify the basic requirements of the electric motor, however, more standardized terms were required to improve the accuracy of the search algorithm to find suitable components. Therefore, a few new terms were proposed to be added to the VSS ontology to describe the requirements for

the ship powertrain components. Table 4 shows the list of terms existing as well as newly proposed terms to be added to the VSS ontology.

General Term	Ontology Term	Description	Units	Existing / Proposed
Electric Motor	PowertrainElectricMotor	The electric motor component		Existing
Maximum Torque	ElectricMotorMaxTorque	The peak torque that the motor can generate		Existing
Maximum Power	ElectricMotorMaxPower	The peak power that the motor can generate		Existing
Motor Speed	MotorRpm	Motor rotational speed measured as rpm		Existing
Motor Efficiency	MotorEfficiency	The efficiency of the electric motor	%	Proposed
Voltage Rating	MotorVoltage	The operating voltage of the motor	V	Proposed
Current Rating	MotorCurrent	The maximum current drawn by the motor	A	Proposed
Shaft Length	MotorShaftLength	Length of the output shaft of the motor	m	Proposed
Shaft Diameter	MotorShaftDia	Diameter of the output shaft of the motor	m	Proposed
Motor Weight	MotorWeight	The weight of the motor	kg	Proposed
Motor Dimensions	MotorDimensions	Dimensions of motor in LxWxH	m	Proposed

Table 4: Terms existing and terms proposed to be added to the VSS ontology.

Once the terms were chosen, the requirements were structured in a standardized format using these terms. The ontology accommodates various formats like XML, JSON, and JSON-LD. For this thesis, the XML format was adopted to delineate the powertrain requirements. This decision was taken since both the VSS ontology and SSP standard predominantly support XML. Utilizing XML offers a straightforward and standardized approach to represent the ontology, ensuring that model requirements and component specifications are articulated uniformly. Consequently, this simplifies communication between component manufacturers and ship powertrain designers. The XML document containing the requirements for the Electric Motor for the ship

powertrain based on VSS ontology is shown in figure 22.

```
<?xml version="1.0"?>
<rdf:RDF xmlns="https://github.com/w3c/vsso#"
    xml:base="https://github.com/w3c/vsso"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xml="http://www.w3.org/XML/1998/namespace"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:skos="http://www.w3.org/2004/02/skos/core#"
    xmlns:vann="http://purl.org/vocab/vann/"
    xmlns:vsso="https://github.com/w3c/vsso#"
    xmlns:dcterms="http://purl.org/dc/terms/"
    xmlns:vsso-core="https://github.com/w3c/vsso-core#">

    <vsso:PowertrainElectricMotor>
        <rdfs:label xml:lang="en">Motor 1</rdfs:label>
        <vsso:ElectricMotorMaxTorque rdf:datatype="xsd:float">200.0</vsso:ElectricMotorMaxTorque>
        <vsso:ElectricMotorMaxPower rdf:datatype="xsd:float">150.0</vsso:ElectricMotorMaxPower>
        <vsso:MotorRpm rdf:datatype="xsd:float">150.0</vsso:MotorRpm>
        <vsso:ElectricMotorEfficiency rdf:datatype="xsd:float">89.0</vsso:ElectricMotorEfficiency>
        <vsso:MotorShaftDia rdf:datatype="xsd:float">0.5</vsso:MotorShaftDia>
        <vsso:MotorShaftLength rdf:datatype="xsd:float">1</vsso:MotorShaftLength>
    </vsso:PowertrainElectricMotor>
</rdf:RDF>
```

Figure 22: The requirements for the Electric Motor defined using XML based on VSS ontology.

The XML document is stored as 'Requirements.xml' which is then added as a resource to the SSP file created earlier using EasySSP. This file would be stored inside the 'Resources' folder inside the SSP zip archive along with the System Structure Description file and other resources in the SSP. The same format is used to define the 'Specifications.xml' and is uploaded as a resource to the FMU component files which represent the actual specifications of the real-world electric motors given by the component manufacturers. The Requirements.xml document in the SSP file is then used to match the values of the terms in the Component specifications to find suitable components for simulation.

3.2 Discovery

3.2.1 Finding suitable FMU components using the SSP file

The DTW would contain the FMUs in the form of component digital twins provided by various component manufacturers. Each FMU would be placed in a component Digital Twin document containing the metadata of the component along with a 'ComponentSpecifications.xml' document which would use the same ontology to define the specifications of the component. The system designer creates a Digital

Design Template (DDT) which would contain the SSP file and its metadata which is uploaded to the DTW. Once the DDT is uploaded, a collaborative design finder containing a search algorithm would be used to compare the values within the 'Requirements.xml' document inside the SSP file with corresponding values within the 'ComponentSpecifications.xml' and find suitable FMUs which satisfy the requirements. Since the DTW is still under development, an algorithm is presented in this thesis, demonstrating the filtering operation of the components digital twins from DTW using the requirements.xml file.

A Python code was developed to demonstrate the process of finding suitable components by comparing the values of the properties defined in the requirements and those in the component specifications. The requirements were specified in an XML document using the VSS ontology named 'Requirements.xml,' which detailed the desired characteristics of the powertrain components, including parameters such as maximum torque, power, efficiency, voltage, current, and various dimensions. The algorithm of the Python code is shown in Figure 23.

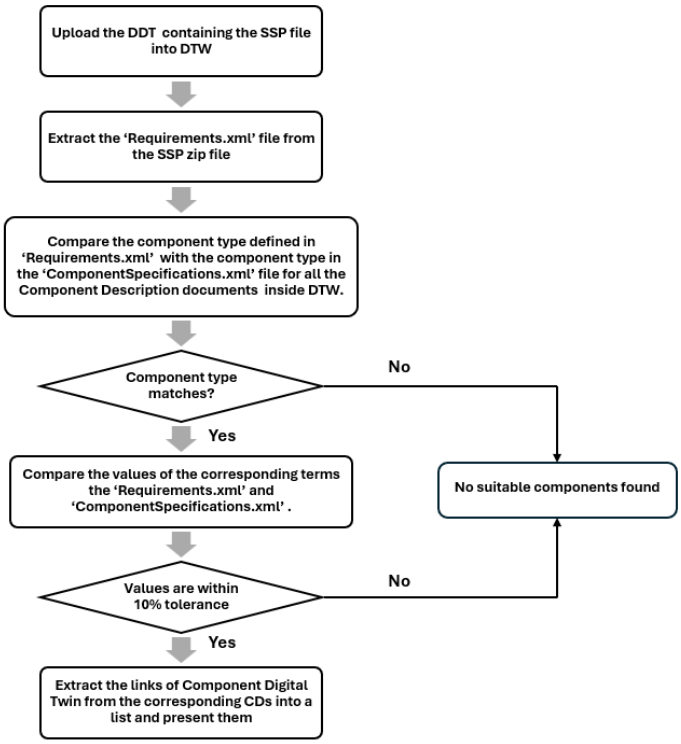


Figure 23: Algorithm to filter suitable component digital twins from DTW.

.

3.2.2 Replacing the components in SSP with selected FMUs

Assuming that the optimal component digital twins were filtered and selected using the algorithm, the subsequent step was to replace the stand-in components from the powertrain model created using EasySSP with the selected FMUs. This was done by writing a Python code using the OpenModelica library. The Python code was used to create a web application that seamlessly integrated the OpenModelica simulation environment. The OMSimulator class was utilized to import and initialize the SSP file. The algorithm of the code can be broken down into the following steps:

1. Extract the System Structure Description file from the SSP model inside the Digital Design Template.
2. Extract the stand-in component names along with their inputs and outputs from the SSD and store them in an array.
3. Display drop-boxes representing each stand-in component with their corresponding name on the web interface.
4. The user drags the FMU corresponding to its name into the drop boxes.
5. When all the FMUs are dropped into corresponding drop-boxes, use the 'replaceSubModel' function from the OMSimulator Python library to swap the stand-in components with the FMUs. The SSD would then be automatically updated with the new FMUs.

The replaceSubModel function was used to replace the FMUs with the stand-in components. It is an inbuilt function inside the OpenModelica library which allows users to replace one component with another. The selected FMU models of all the components including the VfController, Electric Motor and the Load components were dragged and dropped onto the web interface which automatically replaced the stand-in components. The replaceSubModel function automatically modifies the System Structure Description with the new components and inputs, outputs, parameter values and connections specific to each component. Once all the component models are uploaded, the simulate button is displayed which executes the simulation of the SSP model.

Figures 24 and 26 represent the web interface created using Python before and after the replacement of the stand-in components with the FMU components respectively. Figure 25 shows how the FMUs are swapped with the stand-in components inside the SSD using the replaceSubModel fuunction.

Figure 24: Simple web interface for replacing the stand-in components with FMUs inside the SSP model.

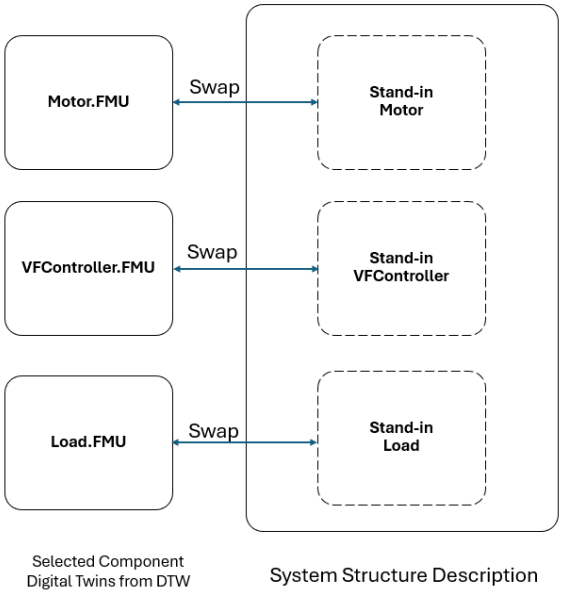


Figure 25: The execution of the replaceSubModel function.

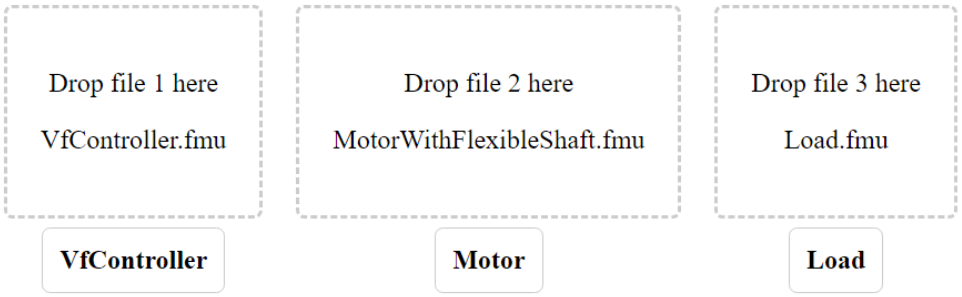


Figure 26: Web interface after replacing the stand-in components with FMUs inside the SSP model.

3.3 Execution of simulation of the SSP models

A Python code was created to simulate the SSP model and to visualize the results using the OpenModelica Simulator python library. To simulate the SSP model, the model is first imported using the importFile function. The instantiate function is used to create a new instance of the model. The initialize function is used to initialize the variables in the model and the stop time of the simulation is set to 60 seconds.

Two SSP models were created using both Model Exchange and Co-simulation FMUs and simulated. Two conditions were chosen to simulate the ship's powertrain. For the first condition, no load was applied on the electric motor. The output torque (τ_{12}) and motor angular velocity (ω_2) for the physics-based model, SSP model with Model Exchange FMUs and the SSP model with Co-simulation FMUs were plotted. For the second case, a load inertia (J) of 20 $Kg.m^2$ was applied on the Electric Motor and all three models were simulated and plotted.

Once the simulate button is clicked on the web application, the simulation is executed using the simulate function. Once the simulation is performed, the results are exported to a CSV file using the setResultFile function. The CSV file contains all the variable names and the results with respect to an individual variable.

The matplotlib function was then utilized to generate scatter plots. The create plot function generates a scatter plot using the specified x-axis and y-axis variables which can be selected from the dropdown menu as shown in Figure 27. This interactive platform facilitates iterative exploration of simulation scenarios, ultimately contributing

to a more efficient and user-friendly simulation workflow. The motor torque with respect to time plotted on the web interface is shown in Figure 28.

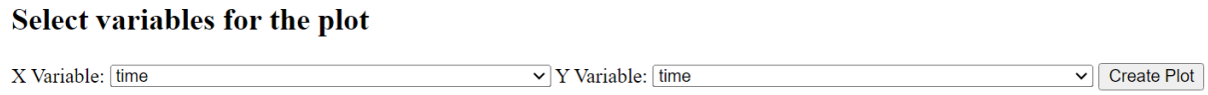


Figure 27: The web interface for plotting the simulation results by selecting the x and y variables.

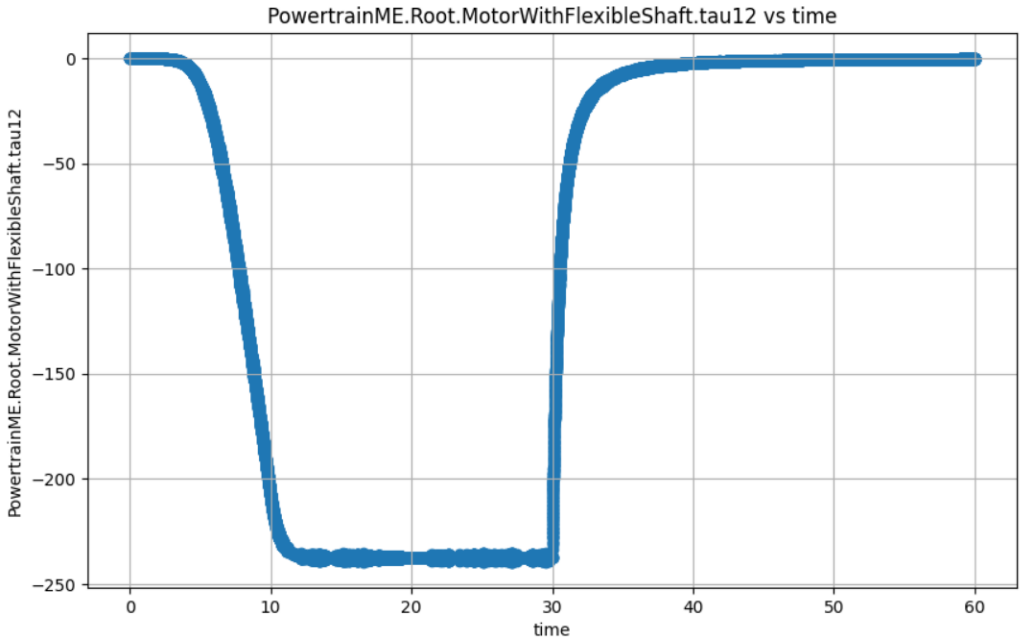


Figure 28: The web interface displaying the plot of motor torque w.r.t time.

4 Results

4.1 Qualitative analysis of the applicability of the SSP standard

This section presents the results from the qualitative analysis based on practical experiences while applying the SSP standard for the case study according to the following criteria chosen: Ease-of-use, Interoperability, Modularity and Accuracy. These criteria help in evaluating whether the SSP standard is suitable for the simulation of ship powertrains or not. Section 4.1.1 delves into the aspect of Ease-of-Use, while section 4.1.2 focuses on Interoperability. Subsequently, section 4.1.3 addresses the Modularity of the SSP standard, and section 4.1.4 explores the Accuracy.

4.1.1 Ease-of-Use

The clear and precise documentation of the SSP standard made it easier to understand the basic concepts of the SSP standards, its features, syntax and use cases. The documentation utilized examples and illustrations to explain the use cases and applications. Additionally, tool-specific documentation played a crucial role in guiding through the necessary steps required for creating SSP models.

Developing SSP models proved to be a straightforward process for the selected case study, using the user-friendly interfaces of both OpenModelica and EasySSP. The graphical user interfaces offered an intuitive approach, facilitating the creation of SSP models from scratch. This simplicity was particularly advantageous, as it allowed for efficient model creation without the need for extensive programming knowledge. There are not many other graphical tools to create SSP models. Without EasySSP, the models have to be created either by using the command line interface or by writing down the System Structure Description in the XML format manually which is both time consuming and less intutitve.

Importing Functional Mock-up Unit (FMU) models, establishing connections between them, as well as assigning and modifying parameters, was straightforward. The tools supported modelling based on both Model Exchange and Co-simulation FMUs, facilitating simulations for both variants. Saving the SSP model followed a standard procedure similar to other simulation models.

The utilization of the OpenModelica Python library expanded the scope of exploring various use cases for the SSP standard. Modifying SSP models became a straightforward task through the implementation of simple Python scripts with the assistance of

these libraries. OpenModelica's documentation offered comprehensive insights into all the functions embedded within the libraries, providing detailed information on how to effectively make modifications to SSP models using Python scripts.

Furthermore, the System Structure Description of the SSP model in XML format proved to be an intuitive way to understand about the structure of the model, even for users who have looked at the model for the very first time. It provided a clear representation of components, their inputs and outputs, parameters, and connections. Overall, the combination of user-friendly interfaces and well-documented standards contributed to the efficiency and ease of the SSP model development process in the case study.

4.1.2 Interoperability

One of the standout features of the SSP standard lies in its emphasis on interoperability. The standard's definition explicitly declares it as a tool-independent standard, allowing SSP models to be seamlessly imported and exported across different tools supporting the standard without any hindrance.

However, in the case study, the experience of sharing SSP models between both tools revealed some challenges. Each tool had a few problems when SSP models created using one tool were imported into the other. When the model created using OpenModelica was imported into EasySSP, it failed to recognize the parameters for one of the two components. Some of the parameters were misinterpreted as connectors and those had to be manually modified to parameters

Importing the EasySSP model into OpenModelica was complicated. When it was first imported, it threw an 'internal error'. Upon conducting a detailed investigation into this issue, it was discovered that the solver used for simulating models in EasySSP was not compatible with OpenModelica. The solver information is stored within the System Structure Description file of the SSP model, and manual editing was required to incorporate the solver information compatible with the OpenModelica simulation tool. Once this modification was implemented, OpenModelica successfully recognized the SSP model.

The developers of the SSP simulation in OpenModelica assured that they were actively working on a solution for this interoperability issue. Nonetheless, it is important to highlight that despite the ongoing efforts, challenges in interoperability persist. These issues could potentially pose difficulties for users seeking to switch between multiple

tools that support the SSP standard. Given that SSP is a relatively recent standard, developers may require additional time to address major issues associated with its implementation.

While the interoperability challenge encountered in the case study did pose initial difficulties for simulations, it was eventually resolved. In the end, the necessary adjustments were made, and the simulations were executed successfully.

4.1.3 Modularity

The modularity of the SSP standard underscores its flexibility, allowing for effortless modifications and the incorporation of supplementary data. This characteristic proved pivotal in the case study, where the integration of standardized requirements into the SSP model was a key objective. The goal was to eliminate the need for creating separate documents for these requirements. With the SSP standard, it became possible to store these requirements seamlessly, regardless of the file type.

The OpenModelica Python library emerged as a valuable tool for replacing FMU models within the SSP model without the need to modify all connections or create an entirely new SSP model. The 'replaceSubModel' function, provided by the library, played a crucial role in simplifying this process. With this function, older models could be effortlessly replaced with the selected FMUs, streamlining the model modification process. The model automatically replaced the names of the components, their inputs and outputs and the parameters.

This modularity enhanced the adaptability of the SSP standard, allowing for efficient adjustments and updates to the model structure as requirements evolved. The ability to modify the SSP model using the OpenModelica Python library ensured a practical and time-saving approach to accommodate changes in the simulation environment.

Moreover, the modularity of the SSP standard extended to the storage of requirements, offering a centralized and integrated approach. This not only reduced redundancy but also contributed to the overall clarity and organization of the simulation model. The combination of modularity and the capabilities of the OpenModelica Python library played a crucial role in enhancing the efficiency and effectiveness of the SSP model development and maintenance processes in the case study.

4.1.4 Accuracy

The accuracy of the results of the simulations of the SSP model was crucial since they have to closely align with the Physics-based model. The FMUs hide the equations which are based on real-world physics.

Figures 29 and 30 show the plots of motor torque and motor angular velocity for all the physics-based models, the Model Exchange SSP model and the Co-Simulation SSP model for the no-load condition. Figures show the plots of Motor Torque and Motor angular velocity for the Physics-based models, the Model Exchange SSP model and the Co-Simulation SSP model when a load torque and load inertia are applied.

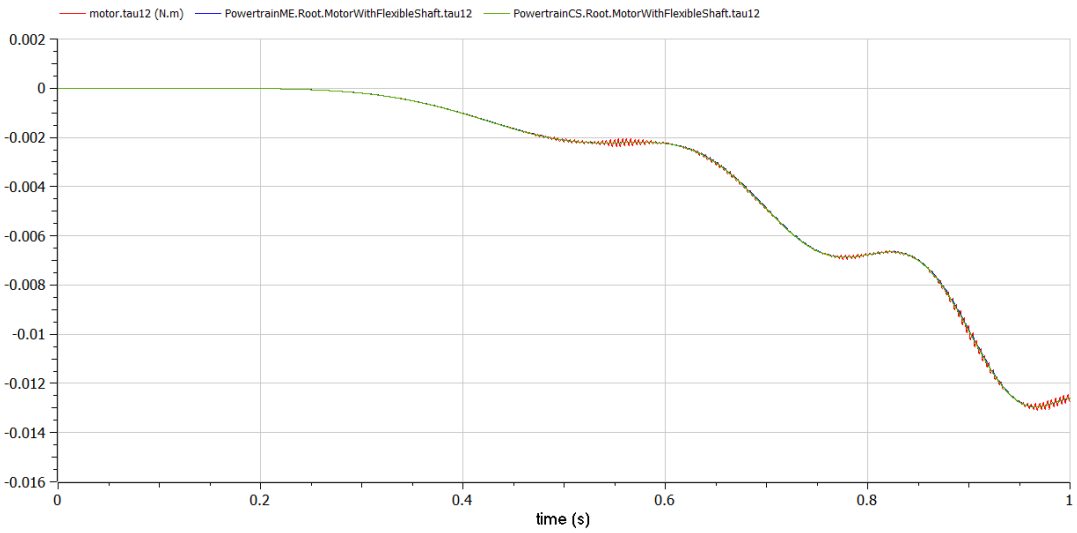


Figure 29: The plot of Motor Torque when no load is applied on the motor. The figure includes the plots from the physics-based model, the Model Exchange SSP model and the Co-simulation SSP model. The simulation is run for 60 seconds, however, the plot is zoomed from 0 to 1 seconds to distinguish the three plots.

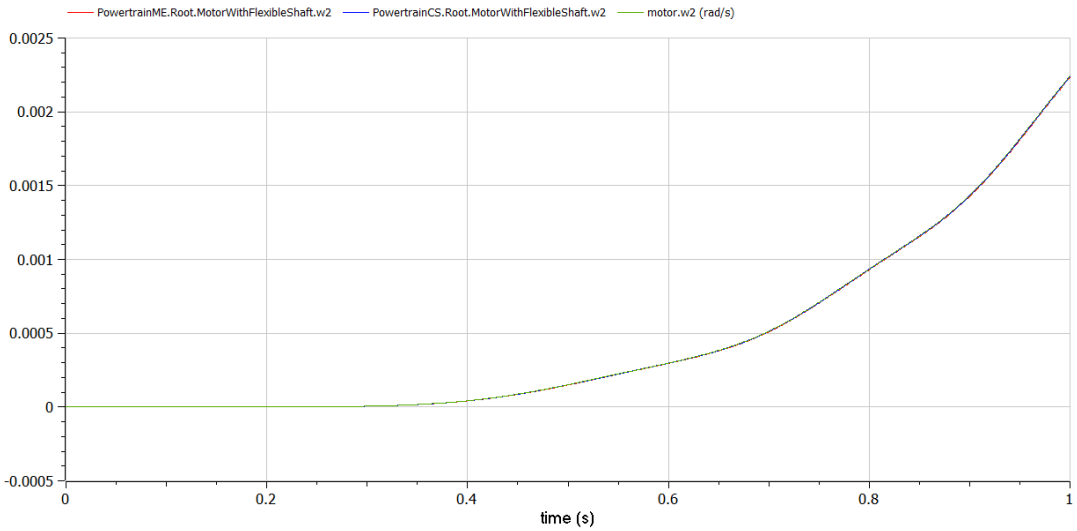


Figure 30: The plot of Motor angular velocity when no load is applied on the motor. The figure includes the plots from the physics-based model, the Model Exchange SSP model and the Co-simulation SSP model. The simulation is run for 60 seconds, however, the plot is zoomed from 0 to 1 second to distinguish the three plots.

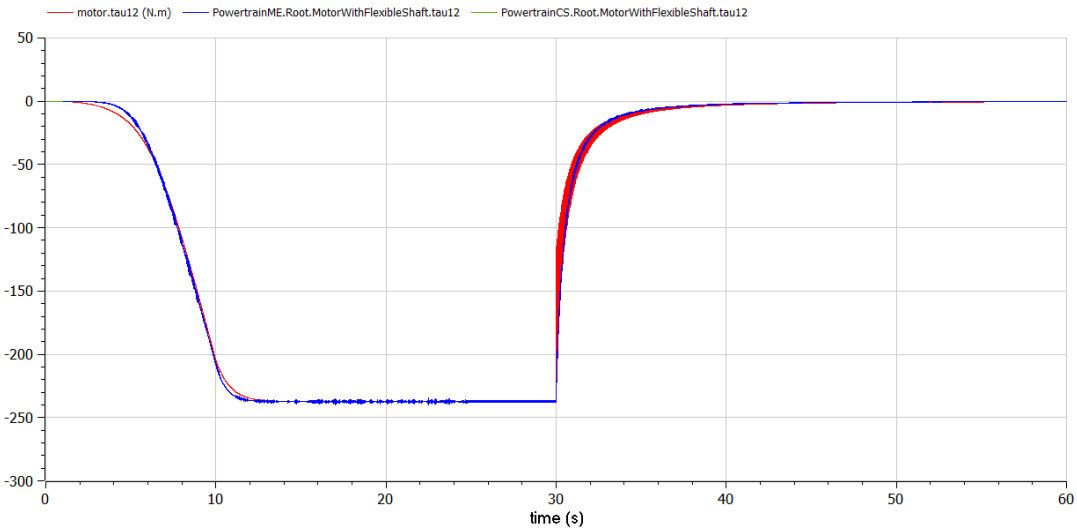


Figure 31: The plot of Motor Torque when an inertia load 20 $Kg.m^2$ is applied on the motor. The figure includes the plots from the physics-based model, the Model Exchange SSP model and the Co-simulation SSP model. The simulation is run for 60 seconds.

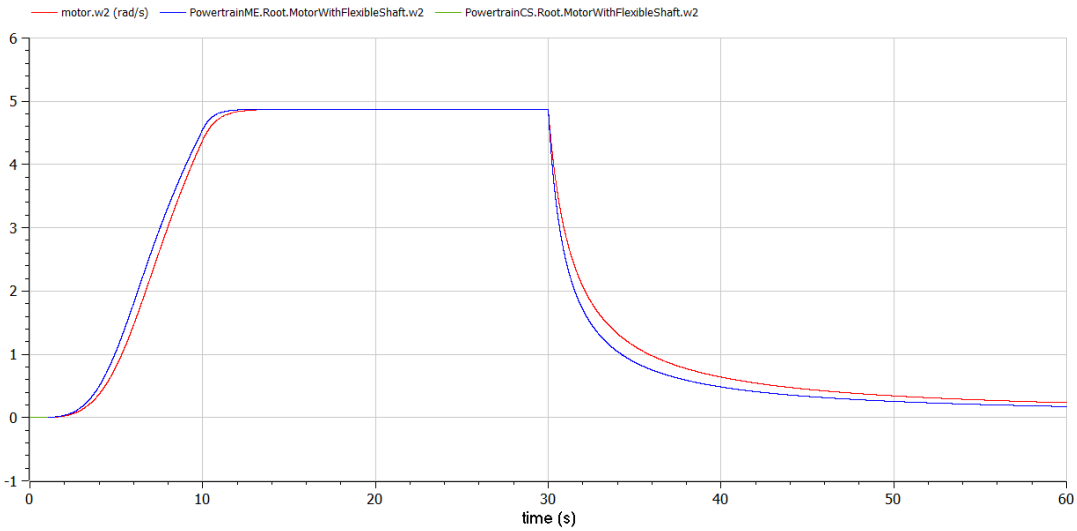


Figure 32: The plot of Motor angular velocity when an inertia load of 20 $Kg.m^2$ is applied on the motor. The figure includes the plots from the physics-based model, the Model Exchange SSP model and the Co-simulation SSP model. The simulation is run for 60 seconds.

Upon analyzing the simulation results, it becomes evident that the plots from all three models exhibit coherence in the no-load conditions. The curves of the Physics-based model, Model Exchange SSP model, and Co-Simulation SSP model closely follow each other. However, some noticeable variations are observed when load inertia is applied. In theory, the results should be the same in all three cases since all three approaches aim to represent the same system and its behaviour.

These variations could be attributed to the step size considered for the simulation. The physics-based model employed the dassl solver with an interval of 0.002, while the Model Exchange SSP model utilized the CVODE solver, a variable-step solver with a step size ranging between 1e-12 and 1e-3, which are configured using the 'setsolver' function in the OpenModelica Python library. On the other hand, the Co-Simulation SSP model employed a fixed-step master solver with a step size of 1e-6. The reason for utilizing different step sizes was the exponential increase in the execution time of the simulations. Using a smaller step size took considerably longer to perform the simulation in the case of Model Exchange and Co-Simulation FMUs. Although these variations in step sizes might contribute to some differences in results, the overall observation is that the results from all three models are in close agreement with each other.

The simulation results for both the SSP models are almost identical in every case.

The difference is only in the amount of execution time. The physics-based model and Model Exchange SSP model take almost the same time to execute the simulation whereas the Co-simulation model takes about ten times more amount of time for the same simulation settings. The reason for this could be due to the additional time required for communication and synchronization. In model-exchange FMUs, the simulation engine is responsible for managing the entire simulation process, including the integration of the model's equations and the communication between the model and other FMUs or external systems.

In contrast, co-simulation FMUs require the involvement of multiple simulation engines, each responsible for simulating a separate FMU. These simulation engines must coordinate their actions to ensure that the simulation progresses in a synchronized manner. This coordination adds an additional layer of complexity to the simulation process, which can lead to slower simulation speeds compared to model-exchange FMUs.

Based on practical experiences derived from the case study, the SSP standard is assessed on a three-level scale **Low, Average, High** for its suitability in standardized model sharing and simulation of ship powertrains. A rating of High denotes strong applicability, indicating confidence in the SSP's effectiveness. An Average rating suggests that while the SSP standard is applicable, enhancements are required for optimal performance. Conversely, a Low rating indicates minimal applicability. The ratings were given while being as objective as possible. Table 5 presents the evaluation of the SSP standard based on the chosen criterion.

Criterion	**Ease-of-Use**	**Interoperability**	**Modularity**	**Accuracy**
Rating	High	Average	High	High

Table 5: Rating of the applicability of the SSP standard ship powertrains based on Ease-of-Use, Interoperability, Modularity and Accuracy.

4.1.5 Summary

The qualitative analysis of the case study in terms of applicability of the System Structure and Parameterization (SSP) standard to other cases based on the criteria of ease-of-use, interoperability, modularity, and accuracy indicates that SSP is a promising solution for standardized model sharing in ship powertrain simulation.

The SSP standard received a **High** rating for ease of use. This is due to the availability

of user-friendly tools like EasySSP and the existence of comprehensive documentation and example use cases. Additionally, the SSP standard is well-organized and easy to understand, making it suitable for users with varying levels of modelling expertise.

For interoperability, the SSP standard was rated **Average**. This is because while SSP is theoretically interoperable with a wide range of modelling tools, there are currently issues with sharing the SSP models from one tool to another although both tools support the standard. This can make it challenging for users to seamlessly integrate with their existing modelling workflow. However, these issues are temporary and as the SSP standard develops further, these issues will ultimately be resolved by developers.

In terms of modularity, the SSP standard was rated **High**. The SSP standard natively allows users to integrate additional documents such as the standardized requirements as a part of the SSP model. Additionally, the SSP models can be easily modified to integrate different sets of FMU components to simplify the simulations.

Finally, the accuracy of the simulations using the SSP standard was also **High**. The simulation results of the SSP models were quite close to the simulation results from the physics-based models. Although Co-simulation SSP models take much longer to solve the model, the results still match with the Model Exchange SSP model.

Overall, the qualitative analysis indicates that SSP is a promising standard for standardized model sharing in ship powertrain simulation. It is relatively easy to use, interoperable with a wide range of tools, modular, and accurate. However, further research is needed to address the limitations of SSP in terms of interoperability and to evaluate its performance in more complex powertrain models.

5 Discussion

This section provides a detailed examination of the findings obtained from the study. The methods and the results presented in the previous sections are analysed and important conclusions and recommendations are given, answering the research questions presented at the outset of the thesis.

Section 5.1 discusses the practicalities of the implementation of the Co-Des framework. Section 5.2 discusses how the current study compares with the previous literature. Section 5.3 and 5.4 present the scientific and real-world contributions of the thesis and finally 5.5 suggests the future research topics based on the findings of this thesis.

5.1 Integrating the SSP standard and VSS ontology into Co-Des Framework

The case study presented a process of implementing the Co-Des framework using the FMI and SSP standards. The results from the case study have shown that the SSP standard and the VSS ontology were quite suitable to be integrated into the framework in order to optimize the collaborative ship powertrain design.

The SSP standard proved to be a suitable method to standardize the ship powertrain simulation model descriptions. A few shortcomings, however, were observed regarding interoperability. Nevertheless, the SSP standard is a relatively recent development and eventually, these shortcomings will be addressed. The seamless communication between the FMI and SSP standards makes it a strong candidate for describing complex system structures and performing simulations using those descriptions.

The VSS ontology also provided standardized terms to define the static requirements of the powertrain components. A few additional terms were required to improve the filtering algorithm to suggest better outcomes. Therefore, additional terms were recommended in this thesis to be added into the ontology.

Integrating the SSP standard and VSS ontology into the Co-Des Framework represents a significant step towards streamlining collaborative ship powertrain design. The SSP standard's ability to standardize simulation model descriptions and the VSS ontology's provision of standardized terms for component requirements demonstrate their effectiveness in enhancing the framework's efficiency and effectiveness.

5.2 Comparison to previous research

This thesis builds upon and extends the existing research of Tevajärvi [1] who presented the FMI standard as a suitable method for IPR-protected model sharing of ship powertrain components. This study investigated standards compatible with the FMI standards for describing powertrain system simulation models and found the System Structure and Parameterization (SSP) standard to be a suitable standard.

While previous studies have explored the potential of SSP in various contexts [37, 38, 59], this thesis provided a more comprehensive and in-depth analysis specifically focused on the modelling and simulation of ship powertrains using case study to create and simulate a simple powertrain model using EasySSP and OpenModelica. It evaluates the effectiveness of SSP using a case study involving a simplified powertrain system and identifies the potential and shortcomings of the standard.

Previous literature provided insights into utilizing an ontology to describe the requirements of components and using the ontology to perform web-based search [35, 36], however, no suitable studies were found that used ontologies defined explicitly for powertrains. The thesis identified the Vehicle Signal Specification Ontology as a suitable method to describe the component requirements of ship powertrains.

5.3 Scientific impact

The current study significantly expands the limited existing literature on the System Structure and Parameterization (SSP) standard and its applications in ship powertrain simulation. While previous studies have explored the potential of SSP for model sharing in various domains, this thesis delves specifically into its application for creating and simulating ship powertrain component models. This focused approach provides a deeper understanding of the specific challenges and opportunities associated with using SSP in the context of ship powertrain modelling.

The thesis demonstrates the practical application of SSP for developing and simulating ship powertrain component models. It presents a detailed methodology for transforming physics-based models into SSP models, highlighting the ease of use and effectiveness of the SSP standard in this context. The case study involving a simplified ship powertrain model demonstrates the ability of SSP to accurately capture the behaviour of individual components and their interactions within complex powertrain systems.

The thesis acknowledges the limitations of SSP and identifies areas for further

research. One key limitation is the interoperability of the SSP standard, which can hinder the seamless sharing and exchange of models between different tools and platforms. To address this issue, the thesis recommends exploring approaches to enhance interoperability and ensure compatibility with a wider range of modelling tools.

Another important limitation identified by the thesis is the lack of standardized model requirements within the SSP standard. The thesis emphasizes the need for defining and incorporating standardized requirements to ensure consistency and compatibility across different models. For this purpose, the thesis proposed the use of ontologies to describe the requirements and specifications in a standardized manner using domain-specific terminology. This would facilitate the exchange of models without compromising their accuracy or applicability.

The thesis proposes additional terms for the Vehicle Signal Specification (VSS) ontology, which is used to describe standardized requirements for powertrain components. These proposed terms would significantly improve the accuracy and granularity of component descriptions, enabling more detailed and comprehensive model sharing. By enhancing the VSS ontology, the thesis contributes to a more robust and standardized approach to modelling and simulating ship powertrain components.

In summary, the current study makes useful contributions to the scientific understanding of the SSP standard for standardized model sharing in ship powertrain simulation. By expanding the literature, demonstrating the practical application of SSP, addressing limitations, and proposing enhancements, the thesis provides valuable insights into the potential and challenges of the SSP standard in this domain. These contributions lay the foundation for further research and development, paving the way for wider adoption of SSP as a standardized practice for ship powertrain modelling and simulation.

5.4 Practical impact

This thesis has the potential to impact collaborative ship powertrain design by introducing the System Structure and Parameterization (SSP) standard for standardized powertrain system descriptions to automate the component model selection for simulations. Traditionally, ship powertrain design has relied on diverse modelling and simulation tools and data formats, hindering collaboration and communication between different stakeholders involved in the process.

By promoting the use of SSP, this thesis paves the way for a more streamlined and

collaborative design approach. SSP offers a standardized structure for representing ship powertrain systems, enabling the creation of interoperable models that can be easily shared and reused by different design teams and software tools. This allows for seamless integration of expertise from various disciplines, fostering a more collaborative environment throughout the design process.

Furthermore, this thesis explores the potential of SSP for automating component selection during the design phase. By establishing a standardized framework for describing component properties and functionalities within the SSP models, the thesis lays the groundwork for the Co-Des framework that the industries could leverage to understand how SSP standards can be utilized for collaborative ship powertrain design.

This automation of component selection would offer several advantages. Firstly, it would significantly reduce the time and resources required for the design process. Secondly, by automating the selection of components that meet specific performance requirements, the thesis contributes to the optimization of ship powertrain design, potentially leading to more efficient and environmentally friendly powertrain configurations.

5.5 Future research

This thesis presents the methodology to support the implementation of the Co-Des framework, however, it is important to note that this thesis does not demonstrate its actual implementation. The development of the Digital Twin Web, a crucial component of the framework, is ongoing at the time of this study. Although Twinbase has demonstrated the implementation of DTW [2], it is still in a very early stage and is not widely adopted. Consequently, the full implementation of the framework should be carried out once all its components are completed and operational.

The thesis utilized a case study with a simple ship powertrain model with only three simple components and all the results applicable to this model alone. If the complexity of the model increases, it is difficult to predict whether these results still hold good. It is therefore advised to test multiple models with varying complexity and number of components using the methods used for the current case study.

The FMU models of components to create SSP models are either all Model Exchange FMUs or Co-simulation FMUs. This study did not involve a case where a combination of Co-simulation and Model FMUs were simulated together. This case might be crucial in certain applications where a few of the component models require the subsystems to be tightly coupled which is possible using Model Exchange FMUs while others

are modelled using Co-simulation FMUs. For instance, the thruster mechanics of a powertrain are extremely complex and they require to be modelled as a single system while other components such as the electric motor and variable-frequency drive can still be modelled as co-simulation FMUs. Therefore, a further study is required to implement such a case using the SSP standard.

For describing the standardized requirements, the ontology approach was selected. This approach was quite suitable for the current case study. For other applications, more terms might need to be proposed or field-specific ontologies might have to be selected. Further research on recognizing other methods for standardizing the requirements and integrating them into the SSP standard could help solve the current issue.

Moreover, the ontology was applied to the static requirements which do not have an impact on the simulation performance. A further study is required to understand the dynamic requirements for the ship powertrains and find suitable methods to standardize those dynamic requirements.

6 Conclusions

The thesis has provided a valuable contribution in taking a step closer towards the realization of the Co-Des framework by identifying standardized methods for describing and simulation of ship powertrains. This thesis has investigated the potential of the System Structure and Parameterization (SSP) standard as a standardized framework for representing, integrating, and simulating ship powertrain models.

Through a comprehensive literature review and a case study involving a simple electric propulsion powertrain, the thesis has demonstrated that the SSP standard effectively stores the structure and requirements of ship powertrain systems, enabling the integration of component models using Functional Mock-up Units (FMU). The high accuracy of SSP-based simulations in replicating real-world ship powertrain behaviour further validates its applicability in this domain.

This thesis presents the methodology to support the implementation of the Co-Des framework which could be valuable to industries which are willing to optimize the component selection process by collaboratively sharing component models. This framework takes a step forward towards facilitating seamless cooperation among industries, fostering improved practices in model-sharing. Streamlining component selection processes paves the way for enhanced efficiency and optimization in ship powertrain design.

Following the qualitative analysis, the SSP standard received high ratings in terms of ease of use, modularity, and accuracy. However, its interoperability was rated as average. This assessment resulted from challenges encountered in sharing SSP models across different tools. While the results of this study are encouraging, there are still several areas for further research. Expanding the scope of SSP to encompass more complex powertrain models, including hybrid and multi-domain systems, is essential to fully realize its potential. Experimental validation with real-world ship powertrains is also critical to provide evidence of SSP in engineering design and decision-making.

In conclusion, this thesis has provided a compelling case for the adoption of SSP as a standardized framework for ship powertrain modelling and simulation. As the SSP standard continues to develop, more tools would support the standard which would help in integrating seamlessly with the broader ecosystem of ship designers. It has the potential to revolutionize the way maritime engineers design, optimize, and validate ship powertrains, leading to safer, more efficient, and more sustainable maritime transportation.

References

[1] J. Tevajärvi, "Protecting intellectual property in multi-supplier ship powertrain co-simulation," 2024.

[2] J. Autiosalo, J. Siegel, and K. Tammi, "Twinbase: Open-source server software for the digital twin web," *IEEE Access*, vol. 9, pp. 140779–140798, 2021.

[3] R. Ala-Laurinaho, J. Autiosalo, A. Nikander, J. Mattila, and K. Tammi, "Data link for the creation of digital twins," *IEEE Access*, vol. 8, pp. 228675–228684, 2020.

[4] F. Tao, Q. Qi, L. Wang, and A. Nee, "Digital twins and cyber–physical systems toward smart manufacturing and industry 4.0: Correlation and comparison," *Engineering*, vol. 5, no. 4, p. 653 – 661, 2019.

[5] R. Piascik, J. Vickers, D. Lowry, S. Scotti, J. Stewart, and A. Calomino, "Technology area 12: Materials, structures, mechanical systems, and manufacturing road map," *NASA Office of Chief Technologist*, pp. 15–88, 2010.

[6] M. Grieves, "Digital twin: manufacturing excellence through virtual factory replication," *White paper*, vol. 1, no. 2014, pp. 1–7, 2014.

[7] M. R. Enders and N. Hoßbach, "Dimensions of digital twin applications-a literature review," 2019.

[8] J.-E. Giering and A. Dyck, "Maritime digital twin architecture: A concept for holistic digital twin application for shipbuilding and shipping," *at-Automatisierungstechnik*, vol. 69, no. 12, pp. 1081–1095, 2021.

[9] Í. A. Fonseca and H. M. Gaspar, "Challenges when creating a cohesive digital twin ship: a data modelling perspective," *Ship Technology Research*, vol. 68, no. 2, pp. 70–83, 2021.

[10] A. Coraddu, L. Oneto, F. Baldi, F. Cipollini, M. Atlar, and S. Savio, "Data-driven ship digital twin for estimating the speed loss caused by the marine fouling," *Ocean Engineering*, vol. 186, p. 106063, 2019.

[11] B. L. Tofte, O. Vennemann, F. Mitchell, N. Millington, and L. McGuire, "How digital technology and standardisation can improve offshore operations," in *Offshore Technology Conference*, p. D041S055R004, OTC, 2019.

[12] M. Grieves and J. Vickers, "Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems," *Transdisciplinary perspectives on complex systems: New findings and approaches*, pp. 85–113, 2017.

[13] M. Jacoby and T. Usländer, "Digital twin and internet of things—current standards landscape," *Applied Sciences*, vol. 10, no. 18, p. 6519, 2020.

[14] C. Wagner, J. Grothoff, U. Epple, R. Drath, S. Malakuti, S. Grüner, M. Hoffmeister, and P. Zimermann, "The role of the industry 4.0 asset administration shell and the digital twin during the life cycle of a plant," in *2017 22nd IEEE international conference on emerging technologies and factory automation (ETFA)*, pp. 1–8, IEEE, 2017.

[15] E. Tantik and R. Anderl, "Integrated data model and structure for the asset administration shell in industrie 4.0," *Procedia Cirp*, vol. 60, pp. 86–91, 2017.

[16] J. Fuchs, J. Schmidt, J. Franke, K. Rehman, M. Sauer, and S. Karnouskos, "I4.0-compliant integration of assets utilizing the asset administration shell," pp. 1243–1247, 09 2019.

[17] V. Charpenay, S. Käbisch, and H. Kosch, "Introducing thing descriptions and interactions: An ontology for the web of things.," in *SR+ SWIT@ ISWC*, pp. 55–66, 2016.

[18] J. Pfeiffer, D. Lehner, A. Wortmann, and M. Wimmer, "Modeling capabilities of digital twin platforms-old wine in new bottles?," *J. Object Technol*, vol. 21, no. 3, p. 3, 2022.

[19] OMG, "Systems modeling language (sysml) specification, version 1.3." Accessed: 9 February 2024. [Online]. Available http://www.omg.org/spec/SysML/1.3/PDF.

[20] M. Nikolaidou, G.-D. Kapos, A. Tsadimas, V. Dalakas, and D. Anagnostopoulos, "Simulating sysml models: Overview and challenges," in *2015 10th System of Systems Engineering Conference (SoSE)*, pp. 328–333, IEEE, 2015.

[21] E. Huang, R. Ramamurthy, and L. F. McGinnis, "System and simulation modeling using sysml," in *2007 Winter Simulation Conference*, pp. 796–803, IEEE, 2007.

[22] M. Nikolaidou, G.-D. Kapos, A. Tsadimas, V. Dalakas, and D. Anagnostopoulos, "Challenges in sysml model simulation," *Advances in Computer Science: an International Journal*, vol. 5, no. 4, pp. 49–56, 2016.

[23] M. Hause *et al.*, "The sysml modelling language," in *Fifteenth European Systems Engineering Conference*, vol. 9, pp. 1–12, 2006.

[24] C. Nigischer, S. Bougain, R. Riegler, H. P. Stanek, and M. Grafinger, "Multi-domain simulation utilizing sysml: state of the art and future perspectives," *Procedia CIRP*, vol. 100, pp. 319–324, 2021.

[25] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing?," *International journal of human-computer studies*, vol. 43, no. 5-6, pp. 907–928, 1995.

[26] J. Lin, M. S. Fox, and T. Bilgic, "A requirement ontology for engineering design," *Concurrent Engineering*, vol. 4, no. 3, pp. 279–291, 1996.

[27] W3C, "Web ontology language (owl) (2012)." Accessed: 9 February 2024. [Online]. Available https://www.w3.org/OWL/.

[28] W3C, "Resource description framework (rdf) (2014)." Accessed: 9 February 2024. [Online]. Available https://www.w3.org/RDF/.

[29] Contributors, "Json-ld - json for linking data (2022)." Accessed: 9 February 2024. [Online]. Available https://json-ld.org/.

[30] U. Durak and T. Ören, "Towards an ontology for simulation systems engineering," in *Proceedings of the 49th Annual Simulation Symposium*, pp. 1–8, 2016.

[31] B. P. Zeigler and P. E. Hammonds, *Modeling and simulation-based data engineering: introducing pragmatics into ontologies for net-centric information exchange*. Elsevier, 2007.

[32] L. McGinnis, E. Huang, K. S. Kwon, and V. Ustun, "Ontologies and simulation: a practical approach," *Journal of Simulation*, vol. 5, no. 3, pp. 190–201, 2011.

[33] S. Banerjee and A. Sarkar, "Ontology-driven approach towards domain-specific system design," *International Journal of Metadata, Semantics and Ontologies*, vol. 11, no. 1, pp. 39–60, 2016.

[34] G. A. Silver, O. A.-H. Hassan, and J. A. Miller, "From domain ontologies to modeling ontologies to executable simulation models," in *2007 Winter Simulation Conference*, pp. 1108–1117, IEEE, 2007.

[35] C. Turnitsa, J. J. Padilla, and A. Tolk, "Ontology for modeling and simulation," in *Proceedings of the 2010 Winter Simulation Conference*, pp. 643–651, IEEE, 2010.

[36] Y. M. Teo and C. Szabo, "Codes: An integrated approach to composable modeling and simulation," in *41st Annual Simulation Symposium (anss-41 2008)*, pp. 103–110, IEEE, 2008.

[37] L. I. Hatledal, R. Skulstad, G. Li, A. Styve, and H. Zhang, "Co-simulation as a Fundamental Technology for Twin Ships," *Modeling, Identification and Control*, vol. 41, no. 4, pp. 297–311, 2020.

[38] S. Sadjina, L. T. Kyllingstad, M. Rindarøy, S. Skjong, V. Æsøy, and E. Pedersen, "Distributed co-simulation of maritime systems and operations," *Journal of Offshore Mechanics and Arctic Engineering*, vol. 141, no. 1, p. 011302, 2019.

[39] F. Perabo, D. Park, M. K. Zadeh, Ø. Smogeli, and L. Jamt, "Digital twin modelling of ship power and propulsion systems: Application of the open simulation platform (osp)," in *2020 IEEE 29th International Symposium on Industrial Electronics (ISIE)*, pp. 1265–1270, IEEE, 2020.

[40] G. Schweiger, C. Gomes, G. Engel, I. Hafner, J. Schoeggl, A. Posch, and T. Nouidui, "An empirical survey on co-simulation: Promising standards, challenges and research needs," *Simulation modelling practice and theory*, vol. 95, pp. 148–163, 2019.

[41] "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules," *IEEE Std 1516-2000*, pp. 1–28, 2000.

[42] A. Falcone, A. Garro, A. Anagnostou, and S. J. Taylor, "An introduction to developing federations with the high level architecture (hla)," in *2017 Winter Simulation Conference (WSC)*, pp. 617–631, IEEE, 2017.

[43] J. S. Dahmann, "High level architecture for simulation," in *Proceedings First International Workshop on Distributed Interactive Simulation and Real Time Applications*, pp. 9–14, IEEE, 1997.

[44] M. D. Petty and P. Gustavson, "Combat modeling with the high level architecture and base object models," *Engineering principles of combat modeling and distributed simulation*, pp. 413–448, 2012.

[45] B. Möller, A. Garro, A. Falcone, E. Z. Crues, and D. E. Dexter, "Promoting a-priori interoperability of hla-based simulations in the space domain: the siso space reference fom initiative," in *2016 IEEE/ACM 20th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pp. 100–107, IEEE, 2016.

[46] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, *et al.*, "The functional mockup interface for tool independent exchange of simulation models," in *Proceedings of the 8th international Modelica conference*, pp. 105–114, Linköping University Press, 2011.

[47] T. Blockwitz, M. Otter, J. Akesson, M. Arnold, C. Clauss, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, *et al.*, "Functional mockup interface 2.0: The standard for tool independent exchange of simulation models," in *Proceedings*, 2012.

[48] M. Association, "Funtional mock-up interface." Accessed: 9 February 2024. [Online]. Available: https://fmi-standard.org/docs/3.0.1/.

[49] J. Bastian, C. Clauß, S. Wolf, and P. Schneider, "Master for co-simulation using fmi," in *8th International Modelica Conference*, vol. 2011, Linköping University Electronic Press, Linköpings universitet Dresden, Germany, 2011.

[50] M. Association, "System structure and parameterization (ssp) specification, version 1.0.1." Accessed: 9 February 2024. [Online]. Available https://ssp-standard.org/publications/SSP101/SystemStructureAndParameterization101.pdf.

[51] J. Köhler, H.-M. Heinkel, P. Mai, J. Krasser, M. Deppe, and M. Nagasawa, "Modelica-association-project "system structure and parameterization"–early insights," 2016.

[52] L. I. Hatledal and E. Fagerhaug, "Enhancing ssp creation using sspgen," in *Modelica Conferences*, pp. 115–119, 2022.

[53] PMSFIT, "Ssp traceability specification, version 1.0.0-beta2, unreleased.." Accessed: 9 February 2024. [Online]. Available https://pmsfit.github.io/SSPTraceability/master/.

[54] M. Ahmann, F. Eichenseer, F. Steimann, M. Benedikt, *et al.*, "Towards continuous simulation credibility assessment," in *Modelica Conferences*, pp. 171–182, 2022.

[55] L. A. Ochel, R. Braun, B. Thiele, A. Asghar, L. Buffoni, M. Eek, P. Fritzson, D. Fritzson, S. Horkeby, R. Hällquist, *et al.*, "Omsimulator-integrated fmi and tlm-based co-simulation with composite model editing and ssp.," in *Modelica*, pp. 157–007, 2019.

[56] S. A. Asghar, S. Tariq, M. Torabzadeh-Tari, P. Fritzson, A. Pop, M. Sjölund, P. Vasaiely, and W. Schamai, "An open source modelica graphic editor integrated with electronic notebooks and interactive simulation," in *8th International Modelica Conference (Modelica'2011), Dresden, Germany, March 20-22, 2011*, vol. 63, pp. 739–747, Linköping University Electronic Press, 2011.

[57] NTNU, "Open simulation platform." Accessed: 9 February 2024. [Online]. Available https://opensimulationplatform.com/.

[58] L. I. Hatledal, R. Skulstad, G. Li, A. Styve, and H. Zhang, "Co-simulation as a fundamental technology for twin ships," 2020.

[59] F. Perabo, D. Park, M. K. Zadeh, Ø. Smogeli, and L. Jamt, "Digital twin modelling of ship power and propulsion systems: Application of the open simulation platform (osp)," in *2020 IEEE 29th International Symposium on Industrial Electronics (ISIE)*, pp. 1265–1270, IEEE, 2020.

[60] ExxcellentSolutions, "Orchideo | easyssp." Accessed: 9 February 2024. [Online]. Available https://www.easy-ssp.com/.

[61] DassaultSystemes, "Dymola." Accessed: 9 February 2024. [Online]. Available https://www.3ds.com/products/catia/dymola.

[62] DassaultSystemes, "Fmi kit for matlab/simulink." Accessed: 9 February 2024. [Online]. Available https://github.com/CATIA-Systems/FMIKit-Simulink.

[63] Modelon, "Fmi toolbox for matlab/simulink." Accessed: 9 February 2024. [Online]. Available https://modelon.com/fmi-toolbox/.

[64] T. Holopainen, J. Roivainen, and T. Ryyppö, "Digital twin of induction motors for torsional vibration analysis of electrical drive trains," in *12th International Conference on Vibrations in Rotating Machinery*, pp. 564–574, CRC Press, 2020.

[65] W3C, "Vehicle signal specification ontology (vsso)." Accessed: 9 February 2024. [Online]. Available https://www.w3.org/TR/vsso/.