Modeling, Identification and Control, Vol. 41, No. 4, 2020, pp. 297–311, ISSN 1890–1328

Co-simulation as a Fundamental Technology for Twin Ships

L.I. Hatledal [1] R. Skulstad [1] G. Li [1] A. Styve [2] H. Zhang [1]

[1] *Department of Ocean Operations and Civil Engineering, Norwegian University of Science and Technology, Larsgrdsvegen 2, 6009 Ålesund, Norway. Norway. E-mail:* {laht,robert.skulstad,guoyuan.li,hozh}@ntnu.no

[2] *Department of ICT and Natural Sciences, Norwegian University of Science and Technology, Larsgrdsvegen 2, 6009 Ålesund, Norway. E-mail:* asty@ntnu.no

Abstract

The concept of digital twins, characterized by the high fidelity with which they mimic their physical counterpart, provide potential benefits for the next generation of advanced ships. It allows analysis of data and monitoring of marine systems to avoid problems before they occur, and plan for the future by using simulations. However, issues related to integration of heterogeneous systems and hardware, memory, and CPU utilization makes implementing such a digital twin in a monolithic or centralized manner undesirable. Co-simulation addresses this problem, allowing different sub-systems to be modelled independently, but simulated together. This paper presents the ongoing work towards realizing a digital twin of the Gunnerus research vessel by applying co-simulation and related standards. The paper does not present a complete, ready-to-use digital twin. Rather it presents the preliminary results, procedure, and enabling technologies used towards realizing one. In order to accommodate this goal, a novel co-simulation solution, developed in cooperation by members of the Norwegian maritime industry, is presented. Furthermore, a maneuvering case-study is carried out, utilizing pre-recorded sensor data obtained from the Gunnerus. Through a comparative study with the real maneuver in terms of speed, course, and power consumption, the proposed approach is verified in simulation.

Keywords: Co-simulation, Digital twin, FMI, SSP, R/V Gunnerus

1 Introduction

There is a strong demand for innovation and efficiency within operations, life cycle services, and design of marine systems. Modern marine vessels operate increasingly autonomously through strongly interacting subsystems. These systems are dedicated to a specific, primary objective of the vessel or may be part of the general essential ship operations. The sub-systems exchange data and make coordinated operational decisions, ideally without any user interaction. The task of designing, operating, and integrating life cycle services for such vessels is a complex engineering task that requires an efficient development approach, which must consider the mutual interaction between the inherent multi-disciplinary on-board sub-systems. Digitalization thus has become a key aspect of making the maritime industry more innovative, efficient, and fit for future operations Sanchez-Gonzalez et al. (2019); Sullivan et al. (2020).

A digital twin can be defined as a virtual representation of a physical asset enabled through data and simulators for real-time prediction, optimization, monitoring, controlling, and improved decision making Rasheed et al. (2020). The digital twin should be able to take advantage of all digital information available for an asset, such as the system and data information models, 3D models, mathematical models, de-

doi:10.4173/mic.2020.4.2

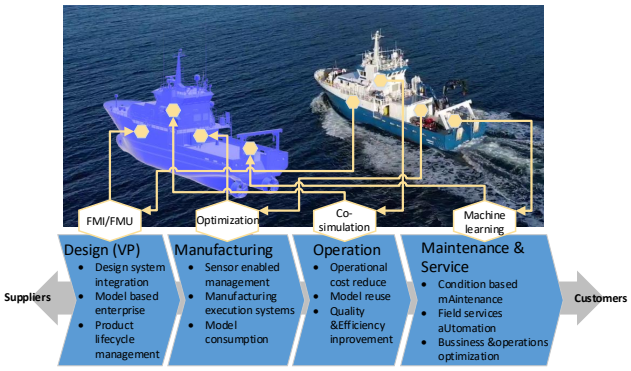

Figure 1: A plausible development procedure of digital twins system for marine industry.

pendability models, condition and performance indicators, and data analytics.

The maritime industry will benefit from digital twin technology Perabo et al. (2020). These proxies of the physical world will help maritime companies in developing enhancements to existing products, operations, and services, and can even help drive the innovation of new businesses. Additional benefits for the maritime industry as a whole is highlighted in Bekker (2018). The eventual goal of this research is to develop digital twins of maritime systems and operations, not only allowing configuration of systems and verification of operational performance, but also to provide early warning, life cycle service support, and system behaviour prediction. As illustrated in Fig. 1, the use of co-simulation together with data related optimization, like data purification, and machine learning methods will be seamlessly combined from the design phase to maintenance phase to achieve heterogeneous simulation, data analytics and behavioural prediction of maritime systems.

As stated in Schleich et al. (2017), the scientific literature has reported that challenges persist in the vision of the implementation of the digital twin, such as insufficient synchronization between the physical and the digital world to establish closed loops, a lack of high-fidelity models for simulation and virtual testing at multiple scales, lacking uncertainty quantification for such models, difficulties related to the prediction of complex systems, and challenges related to the gathering and processing of large data sets. Overcoming these limitations will require a sound conceptual framework and comprehensive reference models. An open platform would ensure that all companies in the surrounding maritime cluster could potentially benefit from and contribute to it. The platform should allow companies to benefit from each other's models and data without necessarily exposing their intellectual property Durling et al. (2017).

In this paper we seek to promote an open-source framework that can leverage the possibilities provided by a digital twin in order to support ongoing work in the Knowledge-building Project for Industry (KPN) *Digital Twins for Vessel Life Cycle Service (TwinShip)* [1]. In order to establish such an open framework for digital twins that enables users to easily develop, integrate, and combine their own components into a complete system, e.g. for the purpose of maritime industry design, operation, service, and maintenance, it is essential to realize effective co-simulation mechanisms and related auxiliary tools. To support the KPN project, DNV GL, Kongsberg Maritime (formerly Rolls-Royce Marine), SINTEF Ocean, and NTNU initiated a Joint Industrial Initiative (JIP) nicknamed the Open Simulation Platform (OSP) Open Simulation Platform (2020) in 2019. The purpose of the OSP is to lay the foundation for an ecosystem where the maritime industry can perform co-simulation and share simulation models in an efficient and secure way. The ultimate goal of the OSP is to facilitate building of digital twin systems and vessels, making it easier to solve challenges related to designing, building, integrating, commissioning, and operating complex integrated systems. Thus it will enable the realization of complex cyber-physical-systems (CPS) like the vessel model illustrated in Figure 2, where the complete vessel model is an aggregation of several independent sub-models that connect through a standardized co-simulation interface.

In this work, we make use of the NTNU owned research vessel (R/V) Gunnerus, as shown in Fig. 1, as the test-bed to demonstrate a sound conceptual framework that uses co-simulation as a fundamental technology towards realizing a digital twin for ship maneuvering. The contributions of the paper include:

1. Employment of a novel co-simulation library as a platform for digital twins.

2. Utilization of a freely available tool-box of marine black-box models, provided by the OSP, in order to accelerate modeling of the Gunnerus.

3. Real-life application of FMU-proxy — enabling co-simulation of otherwise incompatible simulation models.

4. Demonstration of the System Structure and Parameterization (SSP) standard for defining the structure, connections, and the parameterization of the full system to be simulated. Additionally, we demonstrate that components other than

[1] https://org.ntnu.no/intelligentsystemslab/project/twinship.html

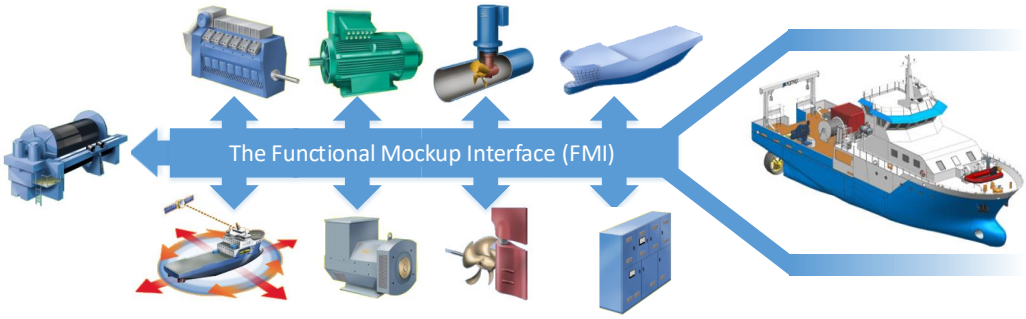

Figure 2: The vessel model depicted in the figure is an aggregate of several different sub-components. *Vessel sub-component figures courtesy of the Virtual Prototyping of Maritime Systems and Operations project (Research Council of Norway, grant nr. 225322).*

Table 1: Open source co-simulation master tools supporting FMI.

	FMI support										
	CS		*ME*								
Name	*v1.0*	*v2.0*	*v1.0*	*v2.0*	**SSP**	**Distributed**	**API**	**CLI**	**GUI**	**Version**	**License**
Coral	✓	✓				✓	✓	✓		0.10.0 (Dec. 2018)	MPLv2
DACCOSIM		✓				✓			✓	2.4.0 (Feb. 2020)	AGPL
FMI Go!	✓	✓	✓	✓	✓[a]	✓		✓		0.5.0 (Nov. 2019)	MIT
Maestro		✓				✓	✓[b]		✓	1.0.10 (Apr. 2020)	GPLv3
MasterSim	✓	✓					✓	✓	✓	0.8.2 (Dec. 2019)	LGPLv3
Ptolemy II	✓	✓	✓	✓			✓		✓	11.0.1 (Jun. 2018)	MIT
FMPy	✓	✓	✓	✓	✓[a]		✓	✓	✓	0.2.17 (Feb. 2020)	BSD
OMSimulator		✓		✓	✓		✓	✓	✓	2.0.1 (Jan. 2019)	GPLv3

[a] Draft version
[b] HTTP API

Functional Mock-up Units (FMUs), such as FMU-proxy, may be used within the context of SSP.

5. Implementation of a preliminary digital-twin model of the Gunnerus, together with subsequent simulation results comparing the power consumption of the model and its real-life counterpart.

The rest of the paper is organized as follows. Section 2 introduces some related work for co-simulation and digital twin platforms. An introduction to the employed co-simulation framework is given in Section 3. The following section provide some implementation notes on the work towards realizing a digital twin of the Gunnerus. Section 5 introduces the case-study, with results and discussions following in Section 6. Finally, some concluding remarks are provided in Section 7.

2 Related work

This section presents an overview of co-simulation technology and related tools, as well as a brief overview of digital twin platforms. Co-simulation as a technology was born out of the idea that no one simulation tool is suitable for all purposes, and complex heterogeneous models may require components from several different domains, perhaps developed in separate, domain-specific tools. In a co-simulation, different sub-systems are modeled separately and composed into a global simulation where each model is being executed independently, sharing information only at discrete time-points. A comprehensive state-of-the-art survey on this topic is given in Gomes et al. (2018). Compared to more traditional monolithic simulations, co-simulation encourages re-usability, model sharing, and fusion of simulation domains. Thus it is in line with the OSP's vision of establishing an eco-system for model sharing within the maritime industry.

Two noteworthy standards for co-simulation exist. The High Level Architecture (HLA) Dahmann et al. (1997) mainly for discrete event co-simulation and the Functional Mock-up Interface (FMI) Blochwitz et al. (2012) for continuous time co-simulation. This work primarily addresses the latter, due to the high number

of supporting tools and the ease with which models can be created and shared. Moreover, a recent survey showed that experts consider the FMI standard as the most promising standard for continuous time, discrete event and hybrid co-simulation Schweiger et al. (2019). Some efforts have also been devoted towards combining the two standards as demonstrated in Yilmaz et al. (2014); Falcone and Garro (2019).

The FMI, currently at version 2.x, is a tool-independent standard that aims to improve the exchange of simulation models between suppliers and original equipment manufacturers. The standard supports both model exchange (ME) and co-simulation (CS) of dynamic models. The key difference between these two variants is that CS models embed a solver, making it easier to deploy at the cost of flexibility. An FMU is a model which implements the FMI standard. It is distributed as a zip-file with the extension *.fmu*. This archive contains:

- An XML-file that contains meta-data about the model, named *modelDescription.xml*.
- C-code implementing a set of functions defined by the FMI standard.
- Other optional resources required by the model implementation.

Since the inception of the FMI standard, a myriad of libraries and software tools have been created or adapted to support it. At the time of writing, the official FMI web page lists over 140 tools, which clearly shows that the standard is being adopted in force. Examples of FMI based co-simulation applied within the maritime domain can be found in Bulian and Cercos-Pita (2018); Hassani et al. (2016); Chu et al. (2018, 2019). Although this standard has reached acceptance in industry, it provides only limited support for simulating systems that mix continuous and discrete behavior, which are typical for CPS Cremona et al. (2018). A future version of the standard (FMI 3.0) will introduce clocks for synchronization of variables changes across FMUs, allowing co-simulation with events.

The Distributed co-simulation protocol (DCP) Krammer et al. (2018) is a standard for real-time and non-real-time system integration and simulation, which the Modelica Association has adopted as a Modelica Association Project. The DCP is compatible with FMI, and just like FMI, it leaves the design of the master out of scope from the specification.

FMU-proxy Hatledal et al. (2019a,b) is an open-source framework that enables language and platform independent access to FMUs. In short, FMU-proxy provides remote procedure call (RPC) mapping to the FMI 2.0 for co-simulation interface. This is achieved by wrapping one or more FMU in a server program supporting multiple schema-based and language-independent RPC systems over several network protocols. The use of schema-based RPCs allows users to easily auto-generate client/server code for a wide range of common programming languages. The framework is independent of the master algorithm, and can therefore be re-used in different software projects.

The System Structure and Parameterization (SSP) Köhler et al. (2016) is a tool-independent standard to define complete systems consisting of one or more components (such as FMUs) including their parameterization, which can be transferred between simulation tools. Version 1.0 of the standard was released in March 2019. The SSP standard is closely aligned with the FMI standard, using the same definition of units and variable types. While FMI is the only model format explicitly mentioned in the standard, a component, which is a blueprint for a model in this context, does not necessarily need to be an FMU. This allows other model formats to be referenced within a SSP archive, such as FMU-proxy or DCP.

Table. 1 provides an overview of open-source tools that are able to orchestrate and run systems of FMUs Gómez et al. (2019); Liu et al. (2001); Nicolai (2017); Lacoursière and Härdin (2017); Ochel et al. (2019); Thule et al. (2019); Catia-Systems (2019); Sadjina et al. (2019). This excludes low-level libraries like the FMI Library JModelica (2017), JavaFMI Galtier et al. (2017), and similar, which only handle loading of individual FMUs. Although a number of existing co-simulation master tools exist and usage of co-simulation to facilitate the digital twin has been presented in, e.g., Yun et al. (2017); Jung et al. (2018); Scheifele et al. (2019); Negri et al. (2019), the OSP partners decided to develop their own alternative, introduced in the following section, due to requirements related to the licensing model, performance, implementation language, maritime ontology, distributed model execution and support for key technologies like FMI 1.0 & 2.0, DCP, and SSP. None of the tools listed support DCP and only FMPy, FMIGo! and OMSimulator support SSP. However, the SSP draft version used by FMPy and FMIGo! is outdated and incompatible with the 1.0 version. Due to the inner workings of some of the models involved, not all models can co-exist within the same process. To overcome this, distributed model execution is required. Neither, FMPy nor OMSimulator supports this. In this way there is sufficient reasoning behind developing an alternate solution that among other things supports SSP 1.0, enables optional distributed execution of FMUs, and which plans to sup-

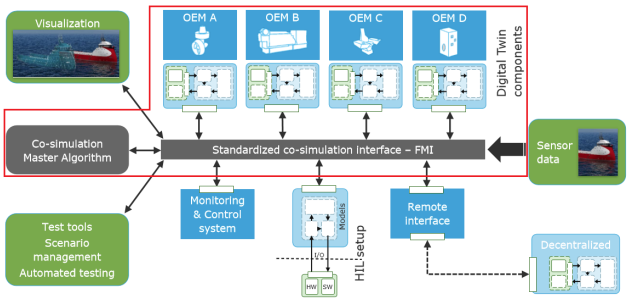

Figure 3: The OSP architecture, with the scope *libcosim* highlighted. *Figure courtesy of the Open Simulation platform.*

port the DCP standard in the future. However, a significant reason for developing yet another co-simulation platform is to maintain control over the software, which allows the collaborators to decide on issues regarding licensing, which features to support and so on.

3 Co-simulation environment

This section introduces the co-simulation environment employed for this research, termed the OSP. The OSP is a collection of software packages developed in collaboration by DNV-GL, SINTEF Ocean, Kongsberg Maritime, and NTNU to facilitate co-simulations and building of digital twin systems and vessels. One of the fundamental deliverables of the OSP is a software library for orchestrating and performing co-simulation named *libcosim*, which is described further below.

3.1 libcosim

libcosim is a cross-platform C/C++ library enabling co-simulations to be orchestrated and run. The scope of *libcosim* within the context of the overall vision of the OSP is illustrated by the framed region in Fig. 3. The OSP will create the foundation for an ecosystem where the maritime industry can perform co-simulation and share simulation models in an efficient and secure way to facilitate building of digital twin systems and vessels. *libcosim* is the cornerstone of the system, enabling users to easily integrate and combine their own components into a complete system, e.g. for the purpose of maritime industry design, operation, service, and maintenance. The co-simulation interface is based on the FMI, with both FMI 1.0 and 2.0 for CS being supported. ME models are not directly supported, however such models may be converted to CS using some appropriate stand-alone tool. Additionally, distributed execution of FMUs is supported through integration with FMU-proxy. Support for the DCP is also planned, which will enable hard real-time integration of hardware devices.

The *libcosim* is written in modern C++, making heavy use of features found in C++11 and above. In order to more easily support integration with other tools, a separate C library is maintained that provides access to most of the functions found in the C++ library. The library is loosely based on Coral with some elements added from CyberSea developed by DNV-GL. Both of whom were developed by collaborating partners. Compared to similar co-simulation libraries and frameworks, *libcosim* is mostly concerned with establishing a solid API that can be embedded in higher-level applications developed by end-users. For convenience, a CLI, which makes the software accessible to non-developers and that simplifies the realization of a number of use-cases, has been developed.

Some of the features of *libcosim* are:

- Integration with Conan dependency manager—making building and distributing the software easier.
- A separate C-API for easier integration with other applications.
- Support for both version 1.0 & 2.0 of the FMI standard for CS.
- Basic support for version 1.0 of the SSP standard, which allows complete simulation systems to be represented in a standardized way.
- Bulk read/write and caching of variable data for efficient access.
- FMU-proxy integration, enabling (optional) distributed execution of FMUs. This in turn enables models to be run regardless of platform, license and software dependencies.
- An extensible design, where master algorithms, slaves, observers, and manipulators are pluggable–allowing library users more control over the simulation.
- The ability to specify events, inline or through configurations files, to occur at specified trigger points, through so-called *scenarios*.

The design of *libcosim* is centralized, with all data flowing through the master. This makes for a less complicated, easier to maintain, easier to debug, and more flexible design compared to similar co-simulation engines such as Coral, where data flows directly between slaves. For instance, entities that want to observe or manipulate the simulation can do so directly as all data

Listing 1: Specifying FMU-proxy sources using libcosim & SSP. Components can be loaded from either an URL, the file system, or using the guide of an already-loaded FMU.

```
<ssd:Component name="model1" source="fmu-proxy://localhost:9090?file=Component.fmu">
<ssd:Component name="model2" source="fmu-proxy://localhost:9090?guid=85bb6608-13d0-46b8-9b8e">
<ssd:Component name="model3" source="fmu-proxy://localhost:9090?url=http://example.com/Component.fmu">
```

is obtainable from a single source. Pure distributed co-simulation masters such as Coral and FMI Go! dictate that all slaves are to be run distributed, whereas *libcosim* makes this entirely optional. Support for this is currently implemented through integration with FMU-proxy, which communicates with remote FMUs using Thrift over TCP/IP. Listing. 1 shows how FMU-proxy components are configured using SSP. A plug-in based system is used to resolve component URIs, allowing custom component sources like FMU-proxy to be added with ease. The support for SSP is not feature complete, but includes the ability to apply linear transformations to connections and multiple parameter sets, both defined inline and as external files.

A crucial part of any co-simulation tool is the available master algorithms. Currently, the library only ships with a single algorithm. A fixed-step algorithm that supports individual FMUs to run at separate step-sizes. However, the API facilitates the creation of additional master algorithms, and as time passes, hopefully more algorithms will be added.

C++ can be a challenging language to learn. Especially compared to higher-level languages like Python or Java. For instance Java has fewer features to learn, is garbage-collected and comes with a richer standard library. Additionally, the tooling, in the form of integrated development environments (IDEs), build systems, and package managers, is state-of-the-art. Therefore, and in order to aid developers that would rater develop in Java, NTNU has developed *cosim4j*, a Java wrapper for *libcosim* introduced in more detail below.

3.2 cosim4j

cosim4j is a Java wrapper for *libcosim*. The goal of the Java API is to be generally easier to use and provide more high-level features than its native counterpart. It uses the Java Native Interface to efficiently interact with the native library. To make the library accessible, it is made available as a Maven artifact at `https://bintray.com/open-simulation-platform/maven/cosim4j`. Furthermore, the artifact include pre-built native binaries for Linux and Windows, which means that no prior installation of *libcosim* is required.

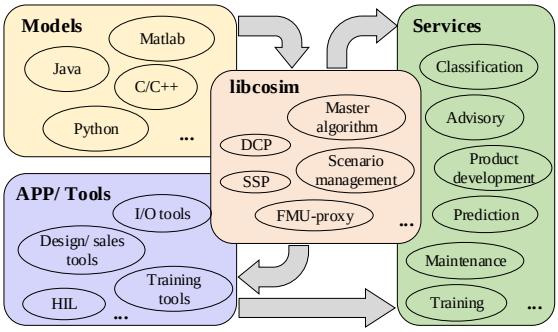

Figure 4: Proposed structure for digital twin implementation.

4 Implementation

Inspired by the overall vision of the OSP in Fig.3, a plausible digital twin framework based on *libcosim* is proposed, as shown in Fig. 4. In the following, implementation details for the proposed framework are provided.

The workflow used towards realizing a digital twin model of the Gunnerus, or digital twins in general, is as follows.

1. Establish the purpose of the model. What should it communicate?
2. Any existing models related to the vessel are collected.
3. Any missing pieces of the puzzle are mapped and consequently implemented using the appropriate software, e.g. FMI4j, PythonFMU or some domain-specific tool, and exported as FMUs.
4. Define the structure of the simulation using the standardized SSP format.
5. Run the simulation using an appropriate tool.

Currently, *cosim4j* is used to run the simulation. The configuration of the system to be simulated is done using SSP. Much as FMI allows us to decouple from the modeling tools, SSP allows us to decouple from the co-simulation master. However, as some of the models currently in use by the digital twin dictate that the full simulation may not run within a single process, the SSP

implementation should support components that can run in separate processes such as FMU-proxy or DCP. An alternative approach to solve this issue, is to run all models in a distributed fashion like e.g. FMIGo! does. However, selective distributed execution, as found in the proposed implementation, has some benefits like easier debugging and less communication overhead in the general case.

As it stands, a complete public overview of available tools that supports SSP is lacking, and implementations, as they become available, will most likely only support FMUs loaded from the file-system—the basic requirement of such an implementation. Using SSP, the structure of a simulation is defined in an XML configuration file. At least one configuration file named *SystemStructure.ssd* must be present. However, additional configurations may optionally be defined, allowing a single SSP archive to contain multiple simulation configurations. Simply explained, an *.ssd* defines which models make up a simulation (components), which variables are exposed (connectors), how they are connected (connections), and how they are parameterized (parameter-sets). Annotations are used to define tool-specific features. The *.ssd* files are packed, together with any resources required, like FMUs, in a zip archive with an *.ssp* extension. While SSP makes it easier to configure systems that can be simulated in a standardized way, it may still be challenging to manually create valid SSP archives due to the shear amount of XML that might have to be written and the packaging of files that goes into the archive. To ease this process, NTNU has developed *SSPgen* Hatledal (2020)—a domain specific language for generating self validating SSP archives. Aside from getting the SSP archive validated prior to simulation, *SSPgen* drastically reduces the amount of code required.

Also embedded in the workflow for realizing the digitalization of the Gunnerus is the use of a set of in-house developed open-source tools for creating FMI 2.0-compatible models in Java (FMI4j) and Python (PythonFMU). Their ease of use makes them ideal for rapid prototyping. FMI4j is an open-source cross-platform Java framework for importing and exporting FMUs. Initially created with FMI import in mind, it has been updated to enable Java code to be exported as FMUs in order to support the work addressed in this paper. Compared to the similar JavaFMI package, FMI4j relies on the Java Native Interface rather than a message passing system making it significantly faster. A Gradle plugin and an easy-to-use CLI that exports conforming Java code as cross-platform FMUs is provided. Listing. 2 shows the minimal required code to write FMI 2.0 compatible models in Java using FMI4j. PythonFMU Hatledal et al. (2020) is a lightweight, open-source, and cross-platform Python 3.x framework for building FMUs readily available through the pip package manager. It has been specifically designed to enable data scientists in the team to contribute with models as the work progresses. Listing. 3 shows the minimal required code to write FMI 2.0 compatible models in Python using PythonFMU.

Listing 2: Writing slaves in Java using FMI4j.

```
public class JavaSlave extends Fmi2Slave {

    @ScalarVariable(causality=output)
    private double realOut;

    public JavaSlave(Map<String, Object> args) {
        super(args);
    }

    @Override
    public void doStep(double t, double dt) {
        realOut = ...
    }

}
```

Listing 3: Writing slaves in Python using PythonFMU.

```
class PythonSlave(Fmi2Slave):

    def __init__(self, **kwargs):
        super().__init__(**kwargs)

        self.realOut = 0.0
        self.register_variable(Real("realOut",
            causality=output))

    def do_step(self, t, dt):
        self.realOut = ...
        return True
```

When designing co-simulations, there might be issues related to causality when two models, which theoretically would be a good match for coupling, have inputs and outputs flowing in the wrong direction compared to each other. Thus, declaring causalities when writing models like shown in Listing. 2 and 3 must be done with great care and in collaboration with other model developers.

5 Case study

Here, the configuration of a case study utilizing the Gunnerus is presented. Its purpose is to test possible applications of digital twin for ship maneuvering and on-board decision support.

The Gunnerus, as seen in Fig. 5, is equipped with the latest technology for a variety of research activities within biology, technology, geology, archaeology, oceanography, and fisheries research. In addition to research, the ship is used for educational purposes and is an important platform for marine courses at all levels and disciplines. Some main dimensions of the vessel are given in Table 2.

Figure 5: Starboard view of the R/V Gunnerus.

Table 2: Main dimensions of the Gunnerus.

Parameter	**Value**
Length overall (Loa)	36.25 m
Length between pp (Lpp)	33.90 m
Waterline length (Lwl)	29.90 m
Breadth middle (Bm)	9.60 m
Breadth extreme (B)	9.90 m
Depth mld. Main deck (Dm)	4.20 m
Draught, mld (dm)	2.70 m
Deadweight	165 t

In this preliminary work, pre-recorded data from the Gunnerus in the form of comma-separated values files are used, as neither the infrastructure for establishing a live link to the vessel nor the means to access recorded data from the cloud are ready. The pre-recorded data from the Gunnerus is wrapped in an FMU, hiding this particular implementation detail and making it possible to add a cloud-connected drop-in-replacement in the future. For this, the plan is to leverage the Cognite[2] cloud platform for data cleaning, analytics, and contextualisation.

One of the deliverables of the OSP is a set of free-of-charge reference models, including models of the most common marine systems and ship dynamics components. The case-study makes use of a number of these models to realize the digital twin. The point of this case study is not to go into detail about how these models are implemented, which in general are black-boxes that could hide proprietary information. Rather, the point is to showcase how co-simulation technology, open-source software, open standards and a library of readily available marine models can be used to develop a digital twin scenario.

The following list briefly describes each of the FMUs used to create the digital Gunnerus.

[2] https://www.cognite.com/

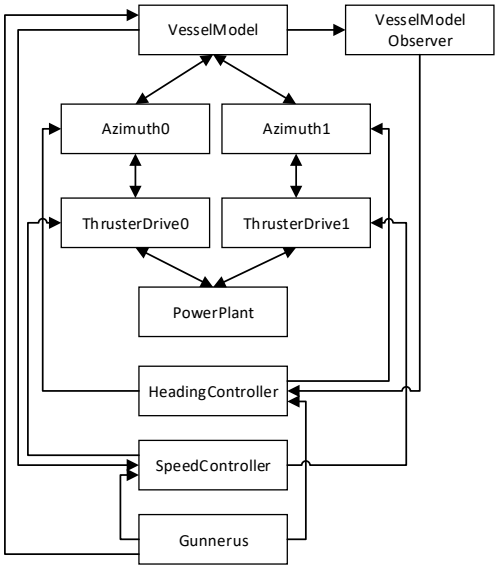

Figure 6: Diagram showing the logical relationship of the involved components.

1. **Gunnerus -** This model contains previously recorded sensor data measured during operation of the Gunnerus. The time-series data is sampled at 1Hz and includes information such as:
   - Heading angle and percent-wise commanded RPM of the tunnel-thruster in the bow as well as the two azimuth thrusters in the aft.
   - Longitude and latitude.
   - Surge, sway, and heave.
   - Yaw, pitch, and roll.
   - Wind direction and speed.
   - Positional and rotational velocities.

   The FMU implements linear interpolation of the recorded data, which is convenient given the low sample rate of the sensor data relative to the simulation, which runs at 20 Hz. In this work, the model acts as a stand-in for what eventually should become a stream of data originating from the real asset.

2. **VesselModel -** This model computes the vessel hydrodynamics such as the radiation forces, mass, and restoring forces as well as manoeuvring forces (resistance and cross flow drag as well as semi-empirical corrections). The equations of motions are solved by this model, summing up all the external forces acting on the vessel. SINTEF Ocean originally implemented the *VesselModel* to model

Table 3: FMUs involved in the case study.

Component	Tool	Vendor	FMI version	canBeInstantiated-OnlyOncePerProcess
HeadingController	FMI4j	NTNU	2.0	False
SpeedController	FMI4j	NTNU	2.0	False
Gunnerus	FMI4j	NTNU	2.0	False
VesselModelObserver	FMI4j	NTNU	2.0	False
ThrusterDrive[a]	20sim	SINTEF Ocean	1.0	True
PowerPlant[a]	20sim	SINTEF Ocean	1.0	True
VesselModel[a]	VeSim	SINTEF Ocean	1.0	True[b]
PMAzimuth[a]	VeSim	Kongsberg Maritime	1.0	True[b]

[a] OSP reference model.
[b] Additionally, only one instance of any model generated by this tool may be instantiated within the same process.

the Gunnerus as part of the SimVal Hassani et al. (2015) project. It was later updated to better approximate an elongated Gunnerus vessel as part of the MAROFF KPN: Digital Twins for Vessel Life Cycle Service (TwinShip). While the model was validated during the SimVal project, it has yet to be validated against the elongated version of the vessel.

3. **VesselModelObserver -** A simple model that computes the direction of travel and speed over ground of the *VesselModel* based on its current and previous position.

4. **SpeedController -** A general-purpose proportional-integral-derivative (PID) controller. It is used to regulate the force required by the *ThrusterDrives* so that the speed of the *VesselModel* and the *Gunnerus* are aligned.

5. **HeadingController -** A special-purpose PID controller where the input data used to compute the controller error is treated as angles in the range $[-180°, 180°]$. This unwinds any input angles that lie outside of the specified range.

6. **PMAzimuth -** The hydrodynamic model of the azimuth thrusters without actuator/motor, implemented by Kongsberg Maritime using VeSim as part of the ViProMa project. Given a certain RPM command (issued by the *ThrusterDrive* FMU), location on the hull, azimuth angle, vessel speed, and the loss factor, the model will output the 3DOF (surge, sway, heave) force generated.

7. **ThrusterDrive -** A drive that converts force commands from the *SpeedController* into RPMs for the *PMAzimuth*.

8. **PowerPlant -** A marine power plant with two equally large gensets, including auxiliary load and circuit breakers.

Fig. 6 shows the logical relationship of the different FMUs, with additional information about the FMUs being provided in Table 3. As illustrated by Fig. 7, the system is far from trivial with a total of 48 variable connections between the models involved. Note that, instances of the *ThrusterDrive* and *PowerPlant* models generated by 20Sim, using an early version of their FMI exporter, cannot co-exist within the same process. This is also true for VeSim Fathi (2013) generated FMUs like the *VesselModel* and *PMAzimuth*. Moreover, these models cannot co-exist within the same process as any other models generated by this tool due to shared library symbol conflicts. To overcome this challenge, execution of the various model instances that cannot co-exist within the same process are split across multiple processes. This is easily solvable using FMU-proxy. Running two instances of FMU-proxy provides two additional processes, which is sufficient for this scenario to run. The distribution of FMUs across the available processes can be seen in Table 4.

To realize the simulation, *cosim4j* is used. The case study presented in this paper is challenging to execute due to the fact that some of the FMUs cannot co-exist within the same process. This makes it impossible to run in non-distributed co-simulation software. This

Table 4: Distribution of FMUs across processes.

Process	FMUs
libcosim	PowerPlant, Gunnerus, VesselModel, VesselModelObserver, SppedController, HeadingController
fmu-proxy1	PMAzimuth, ThrusterDrive
fmu-proxy2	PMAzimuth, ThrusterDrive

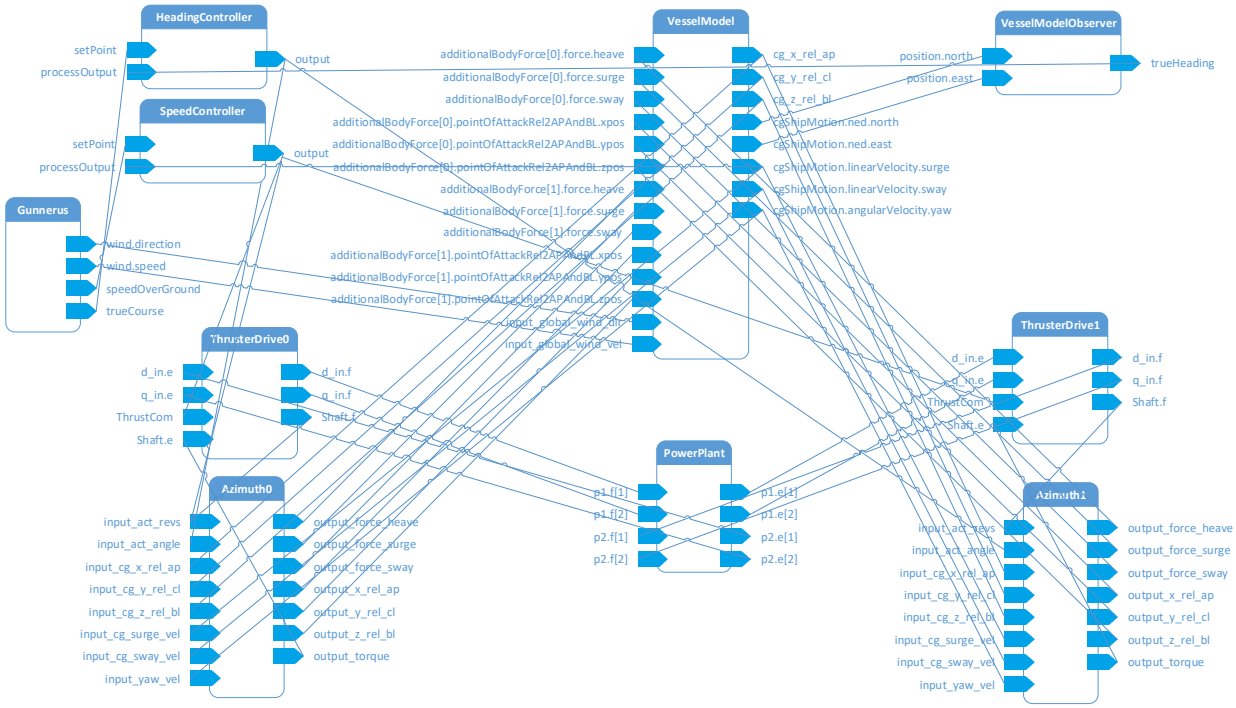

Figure 7: FMU connection graph.

challenge can be solved by means of FMU-proxy, which allows FMUs to be selectively chosen for distributed execution. This enables other parts of the simulation to run within the main application process, which provides the benefit of faster execution times and easier debugging. As noted, when taking the requirements for distributed model execution and the use of SSP 1.0 for creating a standardized system representation, *libcosim* is currently the only library that is applicable of the libraries presented in Section 2.

The idea of the case study is to compare the power consumption of the real vessel and the preliminary digital twin, using data collected from the Gunnerus while performing experiments in the open sea near the city of Trondheim. This is done by feeding the speed and true heading of the real vessel into a set of controllers used to regulate the motion of the twin. To simplify the case study, the equipped thunnel-thruster is not utilized and the command signals to both azimuth thrusters in the aft are equal. Ideally, the power consumption should be comparable, which would indicate a good model fit. However, environmental effects such as current, which are very difficult to measure, could introduce discrepancies between the real and simulated vessel. Yet, the Gunnerus is able to measure and record both wind direction and speed, which are being fed into the model. These measurements are illustrated in Fig. 8. The case study can be run both with and without 3D visualisation enabled. When it is enabled, the simulation is interactive and can be paused/resumed and real-time simulation can be toggled on/off. With real-time on, the execution will try to run in real-time. When successful, the real-time-index (RTI) of the simulation will stay close to 1.0. The other option is to run the simulation as fast as possible. In this preliminary work, it is not necessary to run in real-time as pre-recorded data is used. The case study runs with an RTI of about 30 using a 7th generation Intel i7-8700 CPU on Windows 10. This means that the current models should not be a potential bottleneck once online data becomes available.

6 Results and Discussion

In the following, the simulation results from the case-study are shown. The simulation lasts for approx. 33 minutes, in which the Gunnerus is performing maneuvers in the open sea outside the city of Trondheim. Fig. 9 shows the position and heading of the real and simulation vessel during the case study. Furthermore, the wind direction and normalized magnitude are also shown. To see the actual magnitude of the measured wind speeds, refer to Fig. 8. A comparison of the course of the two vessels is shown in Fig. 10. As can be seen, they are aligning quite well during the entire simula-

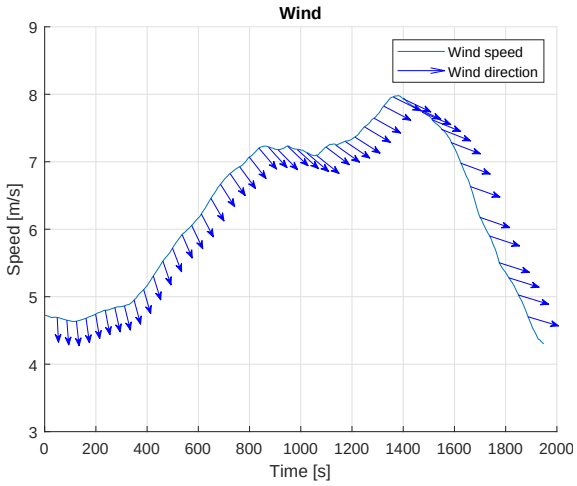

Figure 8: Wind speed and direction measurements obtained from the Gunnerus. The arrow indicates the wind direction according to north.

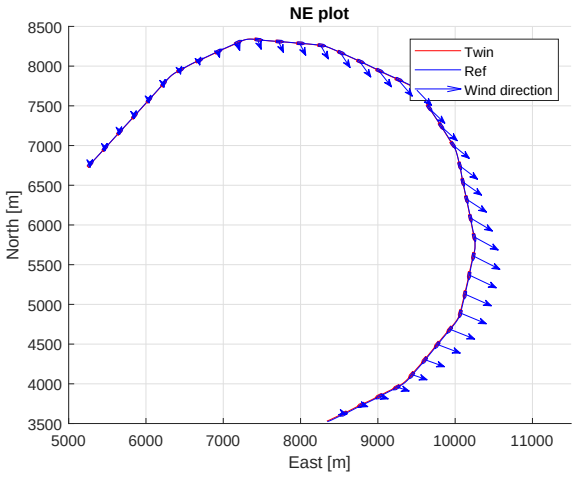

Figure 9: North-east plot showing the trajectory and heading of the vessels during the experiment. The blue arrow indicates the wind direction according to north and normalized magnitude of the speed.

tion. However, the heading controller is a bit aggressive, leading to some oscillations around the set-point, which the authors have not been able to eliminate without sacrificing accuracy over time. The effect of this would be that the power consumption of the twin vessel is increased to some extent. Furthermore, a comparison of the surge speed is shown in Fig. 11. As seen in Fig. 12, speed transients for the twin relates to changes in course made by the Gunnerus. The power consumption is shown in Fig. 13. Interestingly, the power consumption calculated from the twin is showing higher correlation with the speed than that of the real vessel. After approximately 1100s, the power measurement for the real vessel is actually reduced as the speed increases. This could indicate that the vessel is affected by external forces that the model is not aware of, such as current. Therefore, this discrepancy does not necessarily indicate a weakness in the model, but actually provides potentially valuable information regarding external environmental forces acting on the real hull.

From these results, it is clear that some of the underlying models could be more accurately tuned to better reflect the current vessel design. As noted, the employed hull model used has not been thoroughly validated after the Gunnerus underwent an elongation. Doing so might improve the observed difference in terms of overall power consumption.

In order to improve the usability of the digital twin, the offline approach of using pre-recorded operational data should be discarded in favour of a cloud-connected solution with live access to the real asset. This will enable stakeholders and crew members to benefit from the insights provided by the model. This is perhaps the most challenging part, as it requires significant upgrading of the vessel's infrastructure. Today, data from the vessel is buffered on board and transmitted in bulk at intervals measured in minutes. One pragmatic solution to this could be to run the model on board. Implementation-wise, going from pre-recorded data to live data is only a matter of performing a drop-in replacement of the FMU that emits sensor data from the vessel. The simulation structure would not have to be updated as the replacement would share the same model interface. However, using a model connected to a real asset would imply that the simulation would have to be performed in real-time. This mode is supported by *libcosim* and the models used in the simulation are all lightweight enough for the simulation to achieve real-time execution speeds.

7 Conclusion

This paper presents the preliminary results, procedure, and enabling technologies related to our ongoing work to establish a fully operational digital twin of R/V Gunnerus. Co-simulation allows the CPS that the vessel represents to be simulated using models from different vendors and tools. This is absolutely crucial for an aggregate model in the maritime domain, as many different vendors and domain-specific tools are usually involved. Not only does the use of co-simulation allow building of aggregate systems from different vendors, it also allows the simulation to be performed in freely available open-source tools. Furthermore, it makes it possible to decorate the system with models implemented in the tools that best fit the objective.

Figure 10: Course of the vessels. Revisit Fig. 9 for an alternative representation.

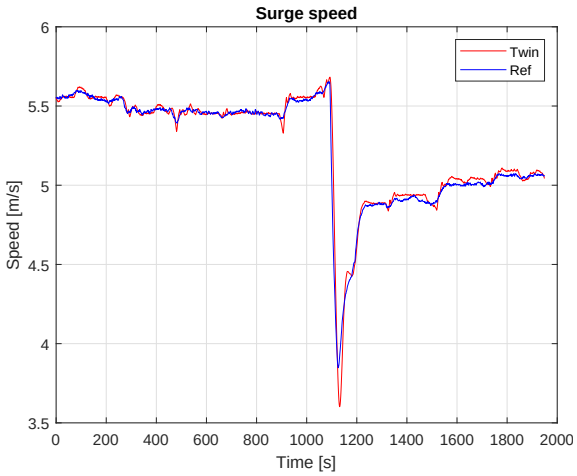

Figure 11: Speed of the vessels.

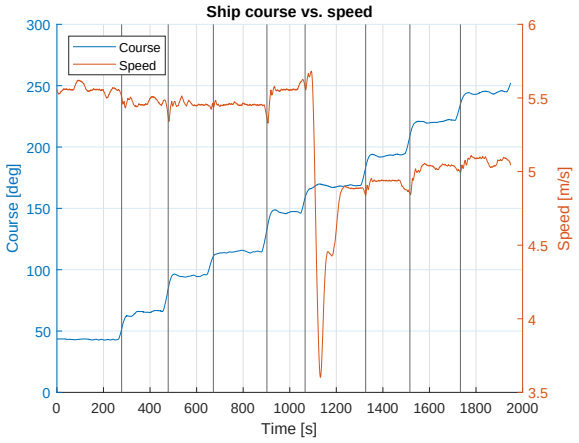

Figure 12: Twin surge speed with respect to course changes by the Gunnerus.

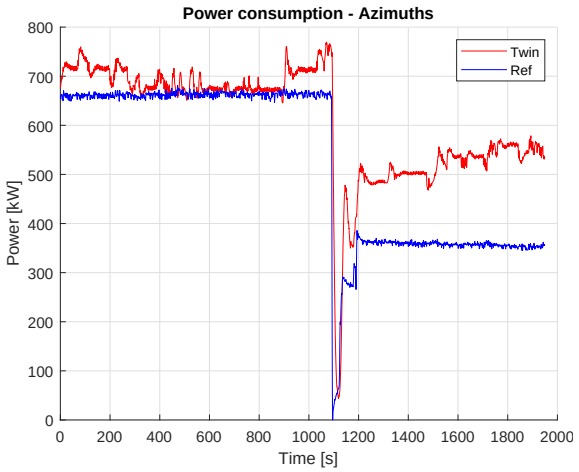

Figure 13: Power consumption comparison. The power output shown is the sum of the two azimuths.

Using the presented simulation framework, model library and tools presented in this paper, NTNU will continue its work towards realizing a digital twin of the Gunnerus, gradually improving its accuracy. Continued development of use-cases will provide meaningful on-board decision support for the crew on-board the Gunnerus. A plausible next step would be to expand on the presented case study by applying a force correction to the hull model in order to offset any differences in position and/or yaw. The amount of force required for this correction could be used as an estimation of environmental forces being applied to the real hull. Being able to quantity these forces would provide substantial support for the crew.

Acknowledgement

This work was supported in part by the Project “Digital Twins for Vessel Life Cycle Service”, under Grant 280703 from Research Council of Norway, and in part by the Project “SFI Offshore Mechatronics”, under Grant 237896 from Research Council of Norway.

The authors would like to thank the members of the Open Simulation Platform for their work related to the employed co-simulation library. A special thanks goes to Stian Skjong and Martin Rindary at SINTEF Ocean for their contribution in terms of models and invaluable insights regarding their usage.

References

Bekker, A. Exploring the blue skies potential of digital twin technology for a polar supply and research vessel. In *Proceedings of the 13th International Marine Design Conference Marine Design XIII (IMDC 2018)*, volume 1. pages 135–146, 2018. doi:10.1201/9780429440533-11.

Blochwitz, T., Otter, M., Akesson, J., Arnold, M., Clauss, C., Elmqvist, H., Friedrich, M., Junghanns, A., Mauss, J., Neumerkel, D., et al. Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. In *Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany*, 076. Linköping University Electronic Press, pages 173–184, 2012. doi:10.3384/ecp12076173.

Bulian, G. and Cercos-Pita, J. L. Co-simulation of ship motions and sloshing in tanks. *Ocean Engineering*, 2018. 152:353–376. doi:10.1016/j.oceaneng.2018.01.028.

Catia-Systems. Fmpy. 2019. URL `https://github.com/CATIA-Systems/FMPy`. (Date accessed 10-December-2020).

Chu, Y., Hatledal, L. I., Æsøy, V., Ehlers, S., and Zhang, H. An object-oriented modeling approach to virtual prototyping of marine operation systems based on functional mock-up interface co-simulation. *Journal of Offshore Mechanics and Arctic Engineering*, 2018. 140(2). doi:10.1115/1.4038346.

Chu, Y., Pedersen, B. S., and Zhang, H. Virtual prototyping for maritime winch design and operations based on functional mock-up interface co-simulation. *Ships and Offshore Structures*, 2019. 14(sup1):261–269. doi:10.1080/17445302.2019.1577597.

Cremona, F., Lee, E., Lohstroh, M., Masin, M., Broman, D., and Tripakis, S. Hybrid co-simulation: It's about time. In *21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2018, 14 October 2018 through 19 October 2018*. Association for Computing Machinery, Inc, 2018. doi:10.1145/3239372.3242896.

Dahmann, J. S., Fujimoto, R. M., and Weatherly, R. M. The department of defense high level architecture. In *Proceedings of the 29th conference on Winter simulation*. pages 142–149, 1997. doi:10.1145/268437.268465.

Durling, E., Palmkvist, E., and Henningsson, M. Fmi and ip protection of models: A survey of use cases and support in the standard. In *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*, 132. Linköping University Electronic Press, pages 329–335, 2017. doi:10.3384/ecp17132329.

Falcone, A. and Garro, A. Distributed co-simulation of complex engineered systems by combining the high level architecture and functional mock-up interface. *Simulation Modelling Practice and Theory*, 2019. 97:101967. doi:10.1016/j.simpat.2019.101967.

Fathi, D. Marintek vessel simulator (vesim), user manual. *MARINTEK. Report*, 2013.

Galtier, V., Ianotto, M., Caujolle, M., Tavella, J.-P., Gómez, J. É., Cabrera, J. J. H., Reinbold, V., and Kremers, E. Experimenting with matryoshka co-simulation: Building parallel and hierarchical fmus. In *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*. pages 663–671, 2017. doi:10.3384/ecp17132663.

Gomes, C., Thule, C., Broman, D., Larsen, P. G., and Vangheluwe, H. Co-simulation: a survey. *ACM Computing Surveys (CSUR)*, 2018. 51(3):1–33. doi:10.1145/3179993.

Gómez, J. É., Cabrera, J. J. H., Tavella, J.-P., Vialle, S., Kremers, E., and Frayssinet, L. Daccosim ng: co-simulation made simpler and faster. In *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*, 157. Linköping University Electronic Press, 2019. doi:10.3384/ecp19157785.

Hassani, V., Rindarøy, M., Kyllingstad, L. T., Nielsen, J. B., Sadjina, S. S., Skjong, S., Fathi, D., Johnsen, T., Æsøy, V., and Pedersen, E. Virtual prototyping of maritime systems and operations. In *ASME 2016 35th International Conference on Ocean, Offshore and Arctic Engineering*. American Society of Mechanical Engineers, pages V007T06A018–V007T06A018, 2016. doi:10.1007/s00773-017-0514-2.

Hassani, V., Ross, A., Selvik, Ø., Fathi, D., Sprenger, F., and Berg, T. E. Time domain simulation model for research vessel gunnerus. In *ASME 2015 34th International Conference on Ocean, Offshore and Arctic Engineering*. American Society of Mechanical Engineers Digital Collection, 2015. doi:10.1115/OMAE2015-41786.

Hatledal, L. I. sspgen. 2020. URL `https://github.com/NTNU-IHB/sspgen`. (Date accessed 10-December-2020).

Hatledal, L. I., Collonval, F., and Zhang, H. Enabling python driven co-simulation models with pythonfmu. In *ECMS*. pages 235–239, 2020. doi:10.7148/2020-0235.

Hatledal, L. I., Styve, A., Hovland, G., and Zhang, H. A language and platform independent co-simulation framework based on the functional mock-up interface. *IEEE Access*, 2019a. 7:109328–109339. doi:10.1109/ACCESS.2019.2933275.

Hatledal, L. I., Zhang, H., Styve, A., and Hovland, G. Fmu-proxy: A framework for distributed access to functional mock-up units. In *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*, 157. Linköping University Electronic Press, 2019b. doi:10.3384/ecp1915779.

JModelica. Fmi library. 2017. URL `http://www.jmodelica.org/FMILibrary`. (Date accessed 10-December-2020).

Jung, T., Shah, P., and Weyrich, M. Dynamic co-simulation of internet-of-things-components using a multi-agent-system. *Procedia CIRP*, 2018. 72:874–879. doi:10.1016/j.procir.2018.03.084.

Köhler, J., Heinkel, H.-M., Mai, P., Krasser, J., Deppe, M., and Nagasawa, M. Modelica-association-project system structure and parameterization–early insights. In *The First Japanese Modelica Conferences, May 23-24, Tokyo, Japan*, 124. Linköping University Electronic Press, pages 35–42, 2016. doi:10.3384/ecp1612435.

Krammer, M., Benedikt, M., Blochwitz, T., Alekeish, K., Amringer, N., Kater, C., Materne, S., Ruvalcaba, R., Schuch, K., Zehetner, J., et al. The distributed co-simulation protocol for the integration of real-time systems and simulation environments. In *Proceedings of the 50th Computer Simulation Conference*. Society for Computer Simulation International, page 1, 2018. doi:10.22360/summersim.2018.scsc.001.

Lacoursière, C. and Härdin, T. Fmi go! a simulation runtime environment with a client server architecture over multiple protocols. In *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*, 132. Linköping University Electronic Press, pages 653–662, 2017. doi:10.3384/ecp17132653.

Liu, H., Liu, X., and Lee, E. A. Modeling distributed hybrid systems in ptolemy ii. In *Proceedings of the 2001 American Control Conference.(Cat. No. 01CH37148)*, volume 6. IEEE, pages 4984–4985, 2001. doi:10.1109/ACC.2001.945773.

Negri, E., Fumagalli, L., Cimino, C., and Macchi, M. Fmu-supported simulation for cps digital twin. In *International Conference on Changeable, Agile, Reconfigurable and Virtual Production*, volume 28. pages 201–206, 2019. doi:10.1016/j.promfg.2018.12.033.

Nicolai, A. Mastersim - a simulation master for functional mockup units. 2017. URL `https://bauklimatik-dresden.de/mastersim/index.php?aLa=en`. (Date accessed 10-December-2020).

Ochel, L., Braun, R., Thiele, B., Asghar, A., Buffoni, L., Eek, M., Fritzson, P., Fritzson, D., Horkeby, S., Hällquist, R., et al. Omsimulator–integrated fmi and tlm-based co-simulation with composite model editing and ssp. In *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*, 157. Linköping University Electronic Press, 2019. doi:10.3384/ecp1915769.

Open Simulation Platform. Open simulation platform - joint industry project for the maritime industry. 2020. URL `https://opensimulationplatform.com/`. (Date accessed 10-December-2020).

Perabo, F., Park, D., Zadeh, M. K., Smogeli, Ø., and Jamt, L. 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)*. IEEE, pages 1265–1270, 2020. doi:10.1109/ISIE45063.2020.9152218.

Rasheed, A., San, O., and Kvamsdal, T. Digital twin: Values, challenges and enablers from a modeling perspective. *IEEE Access*, 2020. 8:21980–22012. doi:10.1109/ACCESS.2020.2970143.

Sadjina, S., Kyllingstad, L. T., Rindarøy, M., Skjong, S., Æsøy, V., and Pedersen, E. Distributed co-simulation of maritime systems and operations. *Journal of Offshore Mechanics and Arctic Engineering*, 2019. 141(1). doi:10.1115/1.4040473.

Sanchez-Gonzalez, P.-L., Díaz-Gutiérrez, D., Leo, T. J., and Núñez-Rivas, L. R. Toward digitalization of maritime transport? *Sensors*, 2019. 19(4):926. doi:10.3390/s19040926.

Scheifele, C., Verl, A., and Riedel, O. Real-time co-simulation for the virtual commissioning of production systems. *Procedia CIRP*, 2019. 79:397–402. doi:10.1016/j.procir.2019.02.104.

Schleich, B., Anwer, N., Mathieu, L., and Wartzack, S. Shaping the digital twin for design and production engineering. *CIRP Annals - Manufacturing Technology*, 2017. 66(1):141–144. doi:10.1016/J.CIRP.2017.04.040.

Schweiger, G., Gomes, C., Engel, G., Hafner, I., Schoeggl, J., Posch, A., and Nouidui, T. An empirical survey on co-simulation: Promising standards, challenges and research needs. *Simulation modelling practice and theory*, 2019. 95:148–163. doi:10.1016/j.simpat.2019.05.001.

Sullivan, B. P., Desai, S., Sole, J., Rossi, M., Ramundo, L., and Terzi, S. Maritime 4.0–opportunities in digitalization and advanced manufacturing for vessel development. *Procedia Manufacturing*, 2020. 42:246–253. doi:10.1016/j.promfg.2020.02.078.

Thule, C., Lausdahl, K., Gomes, C., Meisl, G., and Larsen, P. G. Maestro: The into-cps co-simulation framework. *Simulation Modelling Practice and Theory*, 2019. 92:45–61. doi:10.1016/j.simpat.2018.12.005.

Yilmaz, F., Durak, U., Taylan, K., and Oğuztüzün, H. Adapting functional mockup units for hla-compliant distributed simulation. In *Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden*, 096. Linköping University Electronic Press, pages 247–257, 2014. doi:10.3384/ecp14096247.

Yun, S., Park, J.-H., and Kim, W.-T. Data-centric middleware based digital twin platform for dependable cyber-physical systems. In *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE, pages 922–926, 2017. doi:10.1109/ICUFN.2017.7993933.