Report

AMADEA

Adaptable Maritime Decision Support Architecture

Author(s)

Lars T. Kyllingstad, Stian Skjong, Jarle Ladstein, Joakim Haugen

Report Number

2023:00409 — Unrestricted

Client(s)

SFI MOVE consortium

SINTEF Ocean

Postal address:
Postboks 4762 Torgarden
NO-7465 Trondheim
Norway

Telephone: +47 46415000

Enterprise Number:
NO 937 357 370 MVA

KEYWORDS:
Marine operations
Decision support
Simulation

Report

AMADEA

Adaptable Maritime Decision Support Architecture

VERSION	DATE
1.0	2022-03-06

AUTHOR(S)
Lars T. Kyllingstad, Stian Skjong, Jarle Ladstein, Joakim Haugen

CLIENT(S)	CLIENT'S REFERENCE
SFI MOVE consortium	RCN grant no. 237929

PROJECT NUMBER	NUMBER OF PAGES AND ATTACHMENTS
302003937	66

ABSTRACT
Here, we present a reference model for maritime decision support systems. We establish a common terminology for the components and structure of such systems, describe a number of general-purpose components that can be developed once and then reused in many different contexts, and give practical advice for implementers. We also provide concrete examples of decision support systems for a variety of maritime systems and operations, including offshore lifting operations, fisheries, and marine seismics.

PREPARED BY	SIGNATURE
Stian Skjong	

CHECKED BY	SIGNATURE
Egil Giertsen	

APPROVED BY	SIGNATURE
Arne Fredheim	

REPORT NUMBER	ISBN	CLASSIFICATION	CLASSIFICATION THIS PAGE
2023:00409	978-82-14-07795-7	Unrestricted	Unrestricted

Document History

VERSION	DATE	VERSION DESCRIPTION
1	2023-03-28	First public version of the AMADEA architecture description.

Contents

Acronyms

ADCP acoustic doppler current profiler
AMADEA Adaptable Maritime Decision Support Architecture
AMQP Advanced Message Queuing Protocol
API application programming interface

CoAP Constrained Application Protocol
COTS commercial of-the-shelf

DDS Data Distribution Service
DFT Discrete Fourier transform
DP dynamic positioning
DSS decision support system

FFT fast Fourier transform
FIR finite impulse response
FMC Fundamental Modeling Concepts
FMI Functional Mock-up Interface
FMU functional mock-up unit
FT Fourier transform

GRP glass fibre reinforced polymer
GUI graphical user interface

IDL interface definition language
IIoT industrial Internet of things
IIR infinite impluse response

MDSS maritime decision support system
MOM message-oriented middleware
MQTT Message Queue Telemetry Transport

NLP nonlinear programming

OMG Object Management Group
OPC UA Open Platform Communication Unified Architecture
OSP Open Simulation Platform
OSP-IS Open Simulation Platform Interface Specification

QoS quality of service

RGPS relative global positioning system
ROV remotely operated (underwater) vehicle
RTDS real-time data space

TLS Transport Layer Security
TRL technology readiness level

VPN virtual private network

WAN wide area network

Chapter 1

Introduction

In this document, we present the *Adaptable Maritime Decision Support Architecture* (AMADEA), a flexible and platform-agnostic software architecture for maritime decision support systems. We use the term *maritime decision support system* (MDSS) in a very broad sense, to mean *a system which collects data from sensors and instruments on a ship, combines it with data from other relevant sources, processes it to obtain useful information, and presents the information to users in a timely manner*. The *users* of an MDSS will often be the ship's crew, but they can also have other roles, such as remote operators. What constitutes "useful" information will depend on the ship, the operation, and the users. The same goes for "timely". We will therefore not describe a *particular* MDSS—though we do give some illustrative examples—but rather a general architecture upon which many different types of MDSS may be built. The architecture is a more mature and elaborate version of the ideas presented in a 2019 paper by Skjong et al. [1].

The primary target audience for this document consists of engineers and researchers who work on maritime software systems. In it, we hope they will find useful knowledge, ideas, and tips they can apply to develop powerful MDSS using their preferred tools and software frameworks.

Some knowledge of software development is required to fully understand the contents of the document. That said, we have aimed to make parts of it readable—and interesting, hopefully—to engineers and researchers in adjacent fields, such as marine operations analysis. In particular, this introductory chapter is aimed at a wider audience, as is most of Chapter 3, where we present examples of operation-specific MDSS. The proposed architecture itself is described in technical terms in Chapter 2. Finally, Chapter 4 rounds the whole thing off with some thoughts and advice concerning future development and use of the MDSS architecture.

1.1 Scope and technology readiness level

Upfront we should be clear on what AMADEA is—and, perhaps as importantly, what it isn't. In particular, *this document does not describe a system for which there exists a complete and production-ready implementation*. Instead, AMADEA should be considered a *reference model*, a concept which is defined by the Organization for the Advancement of Structured Information Standards (OASIS) as:

> [...] an abstract framework for understanding significant relationships among the entities of some environment. It enables the development of specific reference or concrete architectures using consistent standards or specifications supporting that environment. A reference model consists of a minimal set of unifying concepts, axioms and relationships within a particular problem domain, and is independent of specific standards, technologies, implementations, or other concrete details. [2]

The concepts and software modules described herein have all, to some extent, been implemented and tested by the authors and/or our collaborators. However, the technological maturity of these implementations varies greatly. Some parts, such as the data acquisition, transfer, and storage subsystems, are quite mature.

In terms of the European Commission's *technology readiness level* (TRL) scale [3], which goes from 1 to 9, our implementation is currently at TRL 8, "system complete and qualified". This is SINTEF Ocean's *Ratatosk* software for marine data acquisition [4]. Other parts are much less mature, some all the way down to TRL 3, "experimental proof of concept". One such example is the model parameter tuning component described in Section 3.4, which currently only exists as a researcher's MATLAB script. This is one of two reasons why this document is not even a detailed technical specification.

The other reason is that a complete implementation or specification would most likely be *less useful* than this document. In developing such an implementation or specification, we would necessarily have to make certain choices: communication protocols, storage formats, middleware, programming language, and so on. Most of our target audience—maritime system suppliers—have existing, in-house frameworks where *these choices have already been made*. No company should be forced to throw ninety percent of their code base out the window in order to accommodate some researcher's idea of a good software system. Instead, what is needed is a way in which the knowledge gained from MDSS research can be easily transferred into these companies' existing systems. That is the primary purpose of this document. The aim is therefore to describe the architecture in a way that is as concrete as possible, yet as abstract as necessary.

A separate but related point worth making is that AMADEA is not limited to any specific type of vessel or marine operation. We will give some examples of use cases and decision support systems that can be built using it in Section 1.3 and Chapter 3. Hopefully, these are varied enough to prove this point.

1.2 Background

What we present in this document is a result of several years of research and development, both within the research centre SFI MOVE and in other projects. This section provides a bit of context. (Readers who are not interested in the history may skip it without missing anything of importance.)

1.2.1 SFI MOVE and Project 6

SFI MOVE is a *Centre for Research-based Innovation* hosted by the Norwegian University of Science and Technology (NTNU). The centre is a collaboration between NTNU, the research institute SINTEF Ocean, and twelve partners from the maritime industry.[1] The centre is jointly funded by the Research Council of Norway (grant no. 237929) and the centre partners for a period of eight years from mid-2015 to mid-2023.

The main goal of SFI MOVE is to perform research that helps to increase the safety and efficiency of demanding marine operations—specifically, installation and maintenance of marine structures under harsh weather conditions. The centre has four main areas of research: *Vessel performance*, *Numerical models and tools*, *On-board systems* and *Integrated simulator environments*.

Multiple projects have been carried out within SFI MOVE, most of them cutting across several of the research areas. Examples include *Safe, all-year, cost-efficient subsea operations*; *Installation of offshore wind power systems*; *Subsea mining*; and *Remote operations/dispersed teams*. The present report is a result from one such project, *Project 6: On-board decision support system*.

The main motivation for Project 6 is to widen the weather window for marine operations. This is important because downtime due to "waiting on weather" is a major cost driver in this industry. The key idea behind the project is that we can achieve this by providing the crew and operators with better, more up-to-date information in the time leading up to and during the operations.

The project is part of a broader effort within SFI MOVE to enable the industry to move from *rule-based* to *response-based* planning and execution of marine operations. What we mean by this will be made clear in a later section, but in brief, it means to base decisions off a system's *actual* response to the *real* situation, rather than rules defined by its *predicted* response to an *imagined* situation.

[1] At the time of writing, the industry partners are: DNV, Equinor, Havfram, Havila Shipping, Kongsberg Maritime, NTNU Ocean Training, Offshore Simulator Centre, Olympic Shipping, Subsea7, TechnipFMC, Ulstein International, and the Vard Group.

Project 6 has comprised a wide variety of activities to support these goals, spanning all of SFI MOVE's four research areas. The activities have included:

- laboratory experiments to study the hydrodynamics of submerged marine structures
- data collection on ships during full-scale operations
- development of methods for numerical modelling and model tuning
- formulation of practical modelling guidelines for engineers
- modelling of ships, equipment, and lifted objects using the abovementioned data and methods
- development of a flexible architecture for on-board decision support systems that brings it all together.

This document represents the outcome of the final activity in this list. With respect to the others, we refer the reader to the large number of other reports and articles that have been published during the course of the project [5].

1.2.2 Other projects; due credit

While this report is to be considered a result from SFI MOVE, in fairness we should say that a lot of the work leading up to it actually predates the centre, and we've also drawn on research performed in projects that have been running parallel to SFI MOVE. Many people besides these authors have played important roles; some of them are mentioned in the citations in the following paragraphs.

We started developing methods and tools for maritime data collection and analysis more than a decade ago, in the research projects *ImproVEDO*[2] [6] and *DANTEQ*[3] [7]. Since then, we have worked steadily to improve the framework, most recently in SFI MOVE and the EU projects *DataBio*[4] [8] and SMARTFISH H2020[5] [9]. Along the way, we have applied and refined the methods and tools in a number of industry projects, some examples of which will be presented in Chapter 3.

Our ideas about a framework for model sharing and co-simulation of maritime systems and operations were born in the *ViProMa*[6] project [10]. We developed them further in SFI MOVE and *TwinShip*[7], and they were finally borne to fruition as the *Open Simulation Platform* (OSP) [11], more on which later.

In several of the case studies described in Chapter 3, we have made use of *SIMO* [12], a computer program for simulation of marine operations. SIMO has been developed and maintained at SINTEF Ocean (formerly MARINTEK) for three decades, but still, new features are added on a regular basis. In SFI MOVE, support for the *Functional Mock-up Interface* (FMI) was added to SIMO so that models can be exported from it and "plugged into" an MDSS.

Besides SIMO, the case studies have also made use of *fmiCpp*, a C++ framework for developing models for co-simulation developed in the Ph.D. work presented in [13], and *FhSim*, a time-domain simulation software for coupled marine systems [14, 15], both continuously developed and maintained by SINTEF Ocean.

[2] *Improved ship design and operation, by operational data aggregation, key performance indices and numerical optimization* (2010–2016), funded by the Research Council of Norway (grant no. 199570), the Norwegian Seafood Research Fund (grant no. 900426), and Kongsberg Maritime CM AS (formerly Rolls-Royce Marine AS).

[3] *Development and assessment of technology improving fishing operation and on board processing with respect to environmental impact and fish quality* (2010–2015), funded by the Research Council of Norway (grant no. 199447) and industry partners.

[4] *Data-driven bioeconomy* (2017–2019), funded by the European Union's Horizon 2020 research and innovation programme (grant no. 732064).

[5] *Smart fisheries technologies for an efficient, compliant and environmentally friendly fishing sector* (2018–2022), funded by the European Union's Horizon 2020 research and innovation programme (grant no. 773521).

[6] *Virtual prototyping of maritime systems and operations* (2013–2016), funded by the Research Council of Norway (grant no. 225322) and industry partners.

[7] *Digital twins for life cycle service* (2018–2022), funded by the Research Council of Norway (grant no. 280703) and industry partners.

1.3 Motivation

As we've already touched upon, a central motivation for the present work, and much of the other research in SFI MOVE, is to widen the weather window for marine operations. However, MDSSes can have many other uses and benefits, which we also discuss in this section.

1.3.1 Widening the weather window

A "wider weather window" for an operation usually means that a ship's expected number of operational hours offshore is increased. However, it may also mean that the operation can now be performed by a ship that was previously considered incapable of the task. The former helps to lower costs by reducing the waiting times for weather windows suitable for offshore operations; the latter helps to make better use of available ships. Ships are generally less expensive to operate the smaller they are. The number of ships that are available to perform a task may also have an impact on the market price.

Today, operators often base their decisions on operational criteria defined through analyses performed during planning of the operation. This planning often takes place weeks or months before the operation is executed. A ship and its operating environment together constitute a highly complex system with many unknowns. An operation analysis must therefore make several simplifying assumptions. Due to the uncertainty, analyses are perhaps more conservative than strictly necessary.

Typical uncertainties are the ship's exact loading condition and the environmental conditions offshore. Shortly prior to the operation, we know more about these things, at least in principle. For example, we can *measure* the sea state and how the ship responds to it, and near-term weather forecasts are much more reliable. With this information, and the tools to process and present it, we believe we can widen the weather window for marine operations significantly without compromising safety. And perhaps more importantly, we may find that we have to *shrink* the weather window if we find ourselves in an unexpected situation where our previous analyses turn out to be unreliable.

So, what are the limiting factors that might prevent a ship from carrying out its operations? First and foremost, it is safety. The crew need to be safe in all situations, and there should be minimal risk of damage to the ship, its equipment, and the handled object. This means that:

- Ship motions and wind must not exceed the limits of safe working conditions for the crew (e.g. on deck) or for safe transfer of people.
- Ship motions and wind must not exceed the limits for safe handling of the object (e.g. through resonant motions).
- Ship motions and forces must not exceed the weather restrictions for the involved equipment such as *remotely operated (underwater) vehicles* (ROVs), cranes, and other lifting equipment.
- Limiting conditions for the positioning system must not be exceeded.
- Limiting conditions for system redundancy and contingency plans should not be exceeded to ensure that the operation can be safely completed, paused, or aborted in the face of certain failures.
- Any other limitations identified by risk assessments, operational experience, and so on must be heeded.

An MDSS cannot in itself affect any of these; they are what they are. What it *can* do is to more accurately *predict* or *monitor* the motions, forces, and events before and during an operation. Thus, it can support the personnel in making decisions that enable them to stay just within the safe limits of an operation.

1.3.2 Other use cases

Of course, MDSSes in general have many uses other than increasing operability in marine operations. We give several concrete examples in Chapter 3, but here are some general categories:

- *Safety*: An MDSS can help to improve safety and reduce risk in general, not only *maintain* the current safety level while widening the weather window.
- *Energy efficiency*: An MDSS can advise the crew on the most energy-efficient way to operate the ship's propulsion systems and other equipment.
- *Operational optimisation*: As a superset of the previous point, an MDSS may help to optimise other aspects of vessel operation or marine operations besides energy, for example time.
- *Information gathering*: Data collection and curation is a central aspect of almost any conceivable MDSS. The gathered information can often be used for *post hoc* learning and improvement too.
- *Crowdsourcing and collective intelligence*: An MDSS need not be limited to using information from, nor on, a single vessel. Information can be shared among multiple vessels, and with shore-based entities, which in some cases may help to see a bigger picture.
- *Training*: An MDSS can be a good way to codify operational expertise and ensure a smooth transfer of knowledge to less experienced personnel.

1.3.3 Towards autonomy

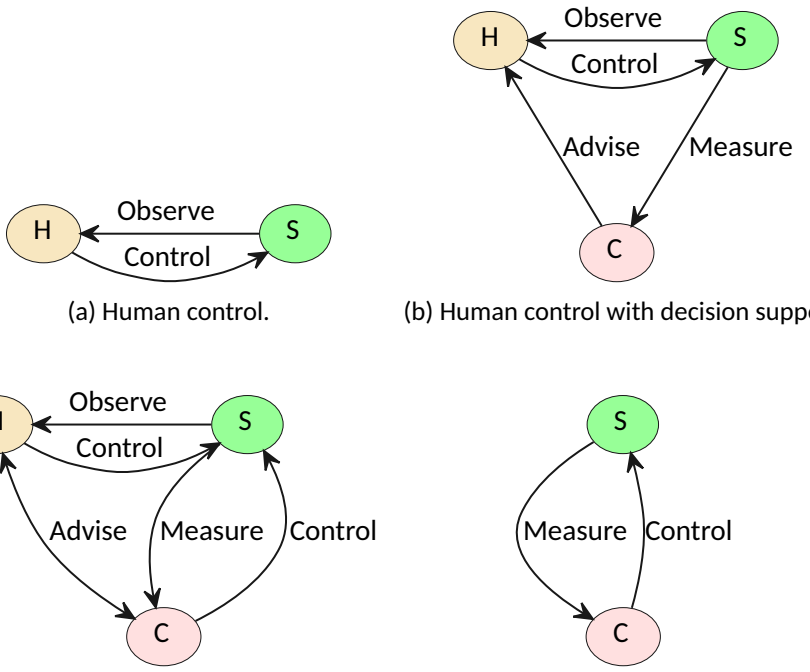
In a longer-term perspective, MDSSes can be stepping stones on the way to building autonomous systems. We can imagine a progression like the one shown in Figure 1.1, where humans are gradually taken out of the loop and control is increasingly left to computers. In certain cases, some or all of the intermediate steps may be skipped. But for complex and high-risk operations, it seems wiser to move cautiously. In those cases, step 2 of the figure is crucial, as it allows computers—specifically, MDSSes—to *suggest* courses of action, but leaves it to the humans to evaluate the suggestions, make the decisions, and execute them. Once a particular system has proven its reliability as an MDSS, it can be further developed into an autonomous system by closing the control loop.

1.4 Decision support systems

In the very first paragraph of this chapter, we gave a definition of MDSS which bears repeating, namely as "a system which collects data from sensors and instruments on a ship, combines it with data from other relevant sources, processes it to obtain useful information, and presents the information to users in a timely manner". This definition is pretty good in terms of scope, but less so in terms of precision. Before we can move on to describing a software architecture for an MDSS, we need to be a bit more specific. What types of decisions should be supported? What kind of information processing is needed? How fast is "timely"? The goal of this section is to get a better handle on these questions, and to establish a terminology that we can use in later chapters.

A quick search for "decision support system" in any of the major scholarly literature databases will reveal that the term was not invented by the authors of the present report. In fact, *decision support systems* (DSSes) constitute a whole field of research, with a well-established literature. A venerable and much-cited source is Sprague [16], who observes the following characteristics of DSSes:

- "they tend to be aimed at [...] less well structured, underspecified problems [...];



(a) Human control.

(b) Human control with decision support.

(c) Semi-autonomous system (i.e., humans control only the most challenging aspects of an operation.

(d) Fully autonomous systems.

Figure 1.1: A progression from human control to fully autonomous systems. Here, *H* stands for "human", *C* for "computer", and *S* for "system", "ship", or "situation".

- they attempt to combine the use of models or analytic techniques with traditional data access and retrieval functions;
- they specifically focus on features which make them easy to use by noncomputer people in an interactive mode; and
- they emphasize flexibility and adaptability to accommodate changes in the environment and the decision making approach of the user."

Even though Sprague's work was in the context of business DSSes aimed at managers and knowledge workers, all of these seem equally applicable to the world of on-board decision support. We can take this as a first sign that there are lessons to be learned from the DSS literature.

1.4.1 Technology layers

Sprague [16] goes on to identify three levels of technology within a DSS:

- *Specific DSS*: "the hardware/software that allows a specific decision maker or group of decision makers to deal with a specific set of related problems".
- *DSS generator*: "a 'package' of related hardware and software which provides a set of capabilities to quickly and easily build a Specific DSS".
- *DSS tools*: "hardware or software elements which facilitate the development of a specific DSS *or* a DSS Generator".

As it turns out, this distinction between levels is central to the design of AMADEA. In fact, the main purpose of the present document is to describe a general architecture for what we might paraphrase as an "MDSS generator": a platform upon which many different types of MDSS may be built.

Extending the nomenclature further, a *specific MDSS* is then a system that provides a specific type of decision support to certain users in a specific marine operation or situation. An example (which we will return to) could be a system that advises the captain about the optimal heading for a lifting operation.

Finally, the *MDSS tools* will be the fundamental building blocks: communication middleware, modelling and simulation tools, optimisation algorithms, and so on. AMADEA itself is not tied to any particular tool; instead, it provides a structure into which many different tools can be fit. We will, however, give examples and recommendations, and we will show how individual tools can be combined to form greater wholes.

1.4.2 Relationship to user: active, passive, or cooperative

Hättenschwiler [17] has proposed to classify DSSes as either *passive*, *active*, or *cooperative* depending on how they present information to, and interact with, their users. A passive DSS is one that presents information, but does not suggest specific decisions or courses of action. The information is intended only as neutral input to the user, who must make their own decisions based on their interpretation of it. An active DSS, conversely, suggests specific decisions or solutions for the user. The user can then choose whether to take or ignore the advice. A cooperative DSS allows for an iterative process between the user and the system, where the user is allowed to adjust or refine the suggestions provided by the DSS.

To make this more concrete, let's look at a couple of examples of how MDSSes could be classified according to this scheme.

First, consider a system for improving a ship's energy efficiency. A *passive* version of this could be a simple "econometer"—an application that continuously calculates and displays various measures of the ship's energy efficiency and fuel consumption rate. If the system in addition suggests a specific power mode that it considers optimal for the current situation (for example "diesel-electric propulsion with two gensets running"), then we would call it *active*. Finally, if we allow the user to input information that the system cannot infer from the current data, which can change the program's estimate of the optimal solution (for example, "we expect intermittent high power loads and therefore need additional spinning reserve"), then we would have a *cooperative* MDSS.

As another example, take a system that predicts the motions of a ship and its payload during a lifting operation. The prediction is made based on simulations of the operation, with weather forecasts and operational parameters as input. A passive version of this MDSS will simply present the predicted motions in some way, perhaps with some confidence interval and in relation to prescribed operational limits. If we give the MDSS the capability to try out different ways of performing the operation and presenting the best one to the user, then it would be an active MDSS. (For example, the program could try to determine the optimal heading through a simple parameter sweep or some more sophisticated optimisation procedure, with an aim to minimise forces in the crane wire.) If the user is allowed to experiment with the operational parameters, such as changing the number of tugger wires attached to the payload, then it would be considered cooperative.

The *cooperative* category provides us with a specific technical requirement for the underlying MDSS generator: It must provide facilities that enable user input and control; it cannot just support one-way data processing "pipelines" that start at a data source and end with passive information on a display.

1.4.3 Relationship to time: monitoring versus prediction

AMADEA is primarily aimed at *online* decision support. By this, we mean that the analyses make use of data that describe the current state of the system or operation, and that they provide results while the data are still (relatively) fresh.

We can further classify online decision support into two types based on its relationship to time:

- *monitoring*, where the MDSS presents useful information about the current state of the system
- *prediction*, where the MDSS tries to predict how the system will behave in the future

Each of these pose different technical requirements to the MDSS generator. For monitoring, we must be able to acquire and process data, and to present the results of the processing, in real time. The architecture therefore needs strong support for fast and reliable data exchange and process synchronisation. This may also be needed for predictions, which will often take the current state as a starting point. But here, we may also need to incorporate information about the future that comes from outside the MDSS itself, for example in the form of weather forecasts or user input.

Of course, many MDSSes will make heavy use of historical data too, and the architecture has strong support for this. The point is that if one *only* uses historical data, in what we may call *offline* analysis, one is likely better off using one of the many well-established data analysis/statistics software packages. (In that case, AMADEA still has a use, namely to build a powerful and flexible data *acquisition* system.)

Finally, we should mention a hybrid case that blurs the boundary between monitoring and prediction, which we may call "alternative present". Here, we run a simulation of the system in real time, that is, in parallel with the physical system, but we change some important detail. For example, if the physical ship hasn't lifted its payload off deck yet, we could run a simulation of how the payload would behave *were we to lift it off right now*. Assuming that the load is small compared to the ship, we can feed the measured ship motions into the simulation in real time and see how they affect the load. This can be described as *monitoring* an alternative reality, or as making a *prediction* assuming that future conditions are similar to present ones, which is why we may call it a hybrid of the two.

1.4.4 Method: data-driven versus model-driven

Power [18] has developed a taxonomy for decision support systems based on the *mode of assistance*. Like so much of the decision support literature, the focus there is on managerial decision support in organisations. Still, part of the taxonomy transfers well over to on-board decision support. In particular, we shall borrow the categories named *model-driven* and *data-driven* decision support.[8]

A model-driven DSS is, as the name implies, based on a model of the actual system or operation for which decision support is needed. "Model" here means a simplified mathematical representation of the system, for example a statistical model or a set of equations describing the system dynamics. Any DSS that attempts to make predictions about the future must necessarily be model-driven, since any assumption about the system's behaviour constitutes a kind of model.

A data-driven decision support system, conversely, is based on access to real data about the actual system or operation, especially historical time series. The analysis can be based on traditional statistical methods or more modern data mining techniques. A *purely* data-driven DSS would make no assumptions about the system behaviour, only about the veracity of the data; it would let the data speak for themselves.

However, the boundaries between these two categories are frequently blurred. Firstly, full-scale data are often used during model development, for parameter estimation, or as model input. In fact, as we shall see, automatic model tuning and model-based state estimation are key elements of the architecture. Secondly, a DSS will often rely on *both* model-driven and data-driven methods. In fact, one rarely sees cases where *no* simplifying assumptions are made about the system being studied, that is, where the system is treated as a "black box". Similarly, purely theoretical "white box" models are seldom useful outside of academia. Most real-world applications fall somewhere on the spectrum between these two extremes and may be referred to as "grey box" models. Still, the distinction between *model-driven* and *data-driven* is useful to indicate in which half of the spectrum a given DSS lies.

[8] The remaining categories, which we will not go into here, are *communication-driven*, *document-driven*, and *knowledge-driven* decision support.

Chapter 2

Architecture

In this chapter, we will formally describe AMADEA. In doing so we have three main goals. The first is to establish a terminology that enables one to speak concisely and unambiguously about the structure and components of an MDSS. This makes it easier for ourselves to describe the architecture and our case studies, of course, but a more important motive is that it facilitates collaborative development of complex, multidisciplinary MDSSes.

The second goal is to describe a set of general-purpose MDSS components. These are components that can be developed once and thereafter reused as building blocks in a wide variety of systems and contexts. (This reusability is one of the main things we wish to demonstrate in the case studies in Chapter 3.) Each of the "logical" components we describe will usually perform a rather narrowly-defined task. In practice, MDSS developers may see fit to create "physical" components that are more specialised and/or combine multiple of them. There could be good reasons to do this, for example to improve performance, accuracy, or maintainability in certain cases.

The third and final goal is to provide concrete advice to developers. There are many ways to construct an MDSS for a given purpose, and many tools to choose from. We will discuss a few of them and share our experience with their use.

In this chapter and the next, we will use the *Fundamental Modeling Concepts* (FMC) block diagram notation to graphically show the substructure of components and systems. An advantage of FMC is that it is simple and intuitive enough that readers without prior knowledge of the formal notation can get a good idea of what the diagrams mean. That said, we do recommend that readers familiarise themselves with it by reading the (rather short) notation reference [19]. There are a few subtle details and nuances in the diagrams that may otherwise be lost.

2.1 Overview of architecture

AMADEA is a *service-oriented architecture* [2]. An MDSS built according to it will consist of a number of services, each of which provides a specific capability. A capability can for example be to obtain a certain type of data or to process data in a certain way. The services communicate with each other through a common infrastructure, so that the output of one service can be used as input to another. In that way, they can be combined as building blocks to form a greater whole. Figure 2.1 shows an example of what this might look like. (The terms used in the diagram will be defined and explained later in this chapter, and diagrams for more concrete MDSSes will be presented in Chapter 3.)

In the figure, we clearly delineate what we have termed *external systems*. These are systems that an MDSS is connected to, and from which it obtains information, but which are not considered part of the MDSS itself. Examples include machinery, instruments, weather services, and more.

We also distinguish between *back-end* components, which process and store information, and *front-end* components, which interact with human operators. AMADEA does not include guidelines or generic components for developing front ends. From the perspective of the architecture, front ends are services like everything

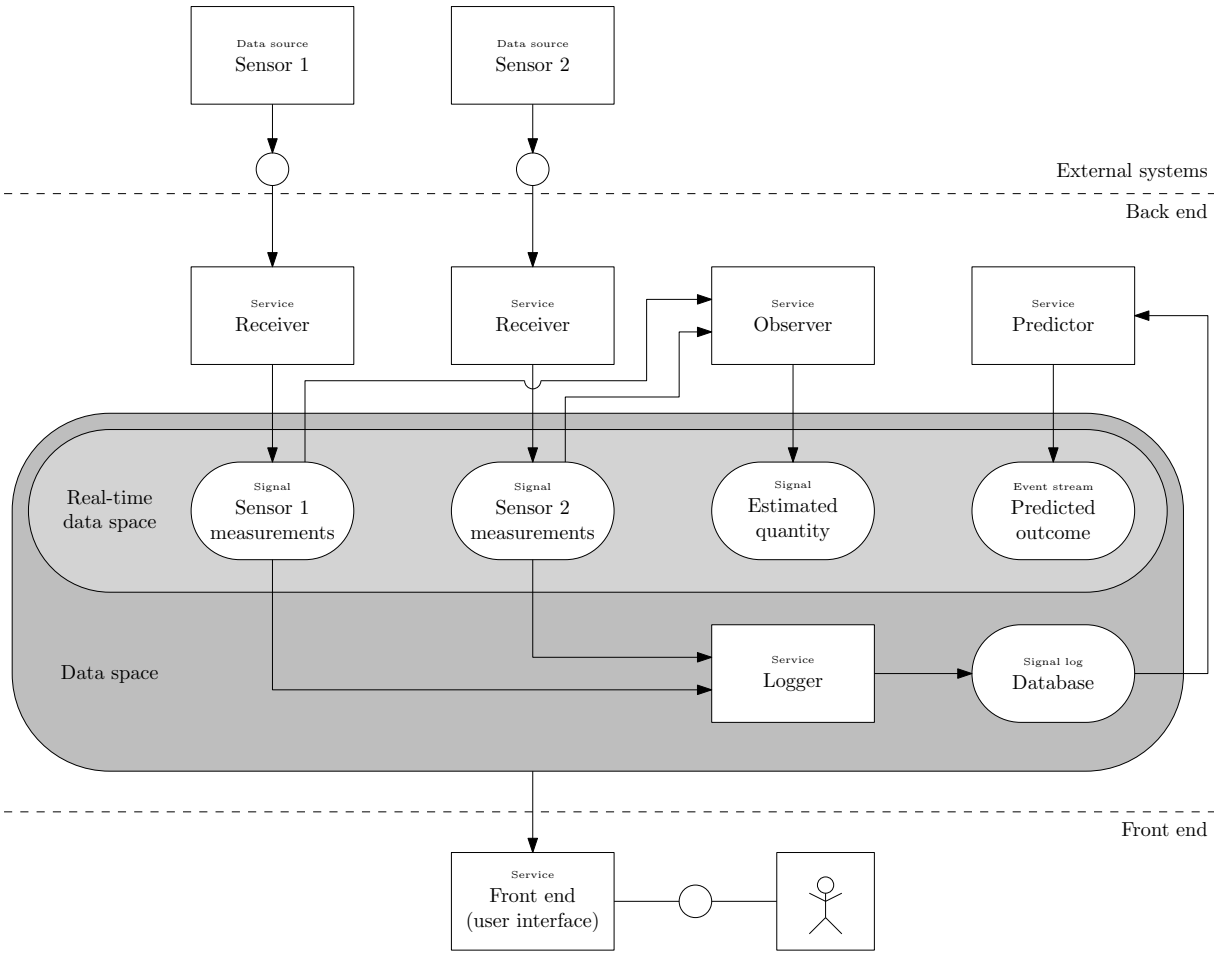


Figure 2.1: Mock-up FMC block diagram of an unspecified MDSS. It consists of multiple services, some of which receive data from the outside, some of which process data to produce results (*Observer* and *Predictor*, in this example), and some of which present data and results to users. The services are linked by an infrastructure that supports both real-time communication and data storage. The dashed lines indicate the boundaries between the MDSS and external systems on one side, and between back end and front end on the other.

else. The topic of user interface design or human–machine interfaces more generally is outside the scope of this document. That said, we will provide some simple examples, mock-ups, and proof-of-concept *graphical user interfaces* (GUIs) in Chapter 3.

2.2 Concepts and terminology

In this section, we will establish the terminology we shall use to describe the AMADEA architecture, including a taxonomy of concepts and properties for the different elements that make up an MDSS. We will take two different perspectives:

- the *information* perspective, where we are concerned with describing the information that flows through the system, *how* it flows through the system, and how it relates to real-world processes
- the *software* perspective, where we concern ourselves with describing the software components that make up an MDSS

First, however, we need to clarify a few things concerning *time* and its significance in an MDSS.

2.2.1 Time

As we discussed in Section 1.4.3, time is of the essence in an MDSS. Decisions usually have to be made within a certain time limit, and the computations performed by the MDSS therefore have to be completed well before that time. This, again, means that the requisite *data* have to be available within a certain time. And finally, most of the data we care about, whether they're measurements or estimates, will be associated with some particular time point or interval and have some lifetime. Beyond their lifetime, the data are no longer considered current.

Commonly, the data in an MDSS come from several different sources, and part of the MDSS's job is to collect them in one place for analysis. A key element of this is to ensure *synchronisation*: Two pieces of data which were generated simultaneously, for example by measurement, should be perceived as simultaneous by all entities in the system. If they get time stamped for later use, their time stamp should be the same, up to the required level of precision. Thus, if the data are time stamped at their source, one must always make sure that the data sources' clocks are synchronised. In practice, this is difficult to ensure and therefore error prone, especially when there are many data sources of different make and kind. AMADEA is therefore based on the principles of *real-time data transfer* and *centralised time stamping*.

Real-time data transfer

No communication is instantaneous, so by "real-time", we simply mean that the time it takes for a sample to be transferred from producer to consumer—the *communication latency*—is small compared to the sampling period.[1] What "small" means here will be somewhat dependent on the use case. Fortunately, for most MDSS use cases, contemporary digital communication methods such as Ethernet are more than fast enough. It is then up to the implementation to ensure that excessive additional latency does not get introduced by the software. Here, the choice of *middleware*[2] may be the most critical point; we return to this topic in sections 2.2.3 and 2.5.1. Henceforth, we shall assume that real-time data transfer is available.

[1] And the sampling period will of course be dependent on the characteristics of the sampled signal.

[2] A *middleware* is a set of software services that provide some feature beyond those provided by the operating system. In this report, we use *middleware* as shorthand for *communication middleware*.

Table 2.1: Message temporality classes.

Temporality class	Description	Examples
Sampled	Message represents the value of some (continuous or discrete) variable at a particular point in time.	vessel speed, crane wire tension
Isolated	Message represents a unique, discrete event that happened at a particular point in time.	user keypress, engine alarm
Non-temporal	Message is not associated with a particular point in time.	configuration parameter, average value

Centralised time stamping

Under the assumption of real-time data transfer, modules that consume data can act as if they have *direct, immediate access* to the data at the source. This is especially important for modules that explicitly time stamp and store data for future use, such as data loggers (see Section 2.4.3). It means that such modules can simply use their own local clock to generate the time stamps. Of course, if there is more than one such module in the system, *their* clocks need to be synchronised. Thus, the problem has not been entirely eliminated, but it has been confined to just a few modules, all inside the MDSS, rather than every single data source in the system. This also means that, in most cases, it has a very simple solution: Just run all time-stamping modules on the same computer.

2.2.2 Information concepts

We shall define three primary concepts that pertain to information: *message*, *channel*, and *data space*. In simple terms, a message is any "chunk" of data that exists in the system, a channel is a way for messages to be stored or communicated, and a data space is a collection of channels. The following sections will put this in more precise terms.

Message

We borrow the word "message" from information theory, and use it to mean *a discrete unit of communication*. In an MDSS context, a message could be a measurement coming from a sensor, a chunk of data stored in a database, a notification about a user keypress, or any number of other things.

Messages are often associated with specific points in time. Measurements are prime examples of this. But this is not always the case. Since we have defined "message" very broadly, the term also includes time-independent information such as configuration settings. To enable us to speak precisely about different cases, we will introduce a message property called *temporality*, which describes whether and how a message is related to a particular point in time. We define three temporality classes, *sampled*, *isolated*, and *non-temporal*, explained in Table 2.1.

Channel

We will use the word "channel" to refer to any means of obtaining messages. Common examples include network connections, databases, configuration files, and so on.

One can distinguish between *physical channels* and *logical channels*, where the latter is limited to carrying only one kind of message. In other words, if we receive two different types of messages via the same network connection, we have two logical channels and one physical. Unless otherwise is specified or obvious from context, we will use the unqualified term "channel" to refer only to *logical* channels.

We will define two useful channel properties:

Table 2.2: Channel persistence classes.

Persistence class	Description	Examples
Volatile	Messages are obtained through a communication medium that is not backed by persistent storage, and once received, it is generally not possible to re-obtain it.	network connection, serial connection
Persistent	Messages are obtained from a persistent storage, meaning that they may be re-obtained arbitrarily many times.	database, file on disk

Table 2.3: Channel latency classes.

Latency class	Description	Examples
Real-time	The channel supports real-time data transfer (as defined in Section 2.2.1).	network connection, serial connection
Non-real-time	The channel can not be relied on to support real-time data transfer.	database, file on disk

- *Persistence* refers to whether whether a single message can be retrieved arbitrarily many times from the channel. We define two persistence classes, *volatile* and *persistent*, explained in Table 2.2.
- *Immediacy* describes whether the channel is able to transfer messages in real time (as defined in Section 2.2.1). As such, the two latency classes are *real-time* and *non-real-time*.

These two properties are related, in the sense that real-time message exchange usually happens via volatile channels (e.g. a network), while non-real-time message exchange often takes place on persistent channels (e.g. a database). But the relation is not one-to-one. Some real-time communication systems enable persistence even if the underlying communication channel is volatile. For example, middleware based on the *publish–subscribe* or *message queue* paradigms can often be configured to maintain arbitrarily long message histories.

We can identify some general channel types which are common enough that it is useful to invent terms to refer to them. Table 2.4 contains a list of named channel types, defined in terms of the taxonomy we have developed in the preceding section and this one.

Data space

The total set of channels available in an MDSS will be referred to as its *data space*. For the subset of channels which are classified as real-time, we will use the qualified term *real-time data space* (RTDS). This is illustrated in Figure 2.2. This distinction makes sense because the RTDS will usually have a unified implementation based on a particular middleware (see Section 2.2.3), while the non-real-time channels will often be a more varied collection of file systems, databases, and so on.

2.2.3 Software concepts

The software components of an MDSS can be grouped into various software concepts that characterise their roles. It can be computer resources, such as physical devices, or running instances of a computer program, commonly denoted *processes*.

Job A process that runs either intermittently or at scheduled intervals with a limited time duration. The purpose of a job is often to perform a task that produces a result, which can be accessed through a prescribed interface or resource. A job is typically triggered by an event or by a predefined time schedule.

Table 2.4: Channel types and their properties.

Channel type	Temporality	Immediacy	Persistence	Description
Signal	sampled	real-time	volatile	The messages in a signal typically represent sensor measurements or the results of having processed other signals. The messages arrive in real time, usually at a fixed sampling frequency.
Event stream	isolated	real-time	volatile	Messages arriving in an event stream contain information about various types of events, such as a user action or a discrete change in some system. The messages arrive immediately after each event occurs, and usually not at any fixed frequency (unless the event itself is periodic).
Signal log	sampled	non-real-time	persistent	Signal logs are stored time series. Usually, the data will originate in various signals, which are timestamped as they are recorded.
Configuration	non-temporal	non-real-time	persistent	Information that specifies how the system should operate, possibly customising it for a specific vessel, situation, or user group.

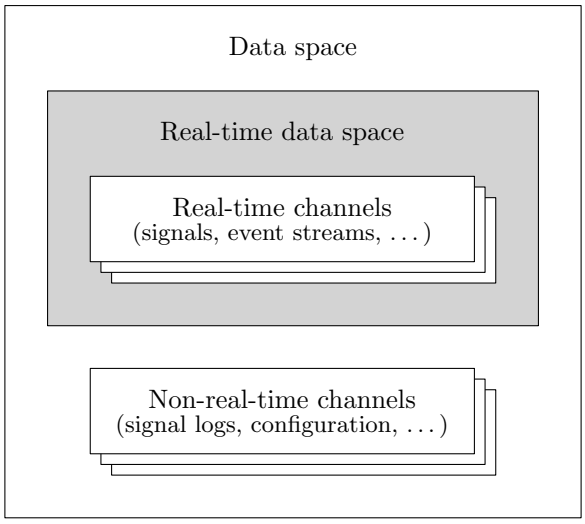


Figure 2.2: Data spaces and the channels they contain.

Service A process that runs continuously. A service is a mechanism to provide capabilities through a prescribed interface [2]. Such a process will usually run as long as the MDSS is operational.

Resource Physical devices such as a disk drives, communication interfaces, or more concrete resources such as databases, files, network connections.

Process manager A set of processes that monitor and control other processes and resources that make up the MDSS. It is the responsibility of a process manager to ensure that services are running and that resources are available, et cetera. For distributed MDSSes, i.e., ones for which different services may run on different machines in a network, there will usually be one process manager running on each machine. Operating systems usually provide software that can accomplish this; see Section 2.5.4 for examples.

Middleware A shorthand for *communication middleware*; this is a set of services that provides data communication through the means of a common *application programming interface* (API). Such middleware enables otherwise separate software components to exchange information. It should provide unified real-time communication, which typically include concepts such as publish–subscribe and request–reply using logical network channels. See Section 2.5.1 for more details.

2.3 Service categories

It is useful to categorise services according to how they connect to the *data space*. In particular, we will make a categorisation according to how the services are connected to *signal* channels in the RTDS. Two possible connections exist: input and output. A service has an input signal if messages are received from the RTDS and feed into the service. If messages are produced by the service and made available in the RTDS, the service is said to have an output signal. We will use the following categorisation of services to describe how services interact through signals:

Signal source Service that only has output signals. A signal source service provides data to the system through signal channels in the RTDS. Examples of signal source services:

- Receiver: Reads data from the “outside world” (e.g. sensors) and makes them available via the middleware.
- Playback: Reads stored data and plays them back in real time.

Signal sink Service that only has input signals. A signal sink service uses data that is available from signals in the RTDS. Examples of signal sink services:

- Transmitter: Reads data off the middleware and transmits them to the “outside world”.
- Logger: Stores data.

Signal processor Service with both input signals and output signals. Typically data from the input signals will be processed to produce data that are transmitted through the output signals.

- Observer: Produces estimates of (possibly unmeasured) state variables by combining measurements with a model of a system’s dynamics.
- Virtual observer: Like an observer, but tries to estimate what would happen in an "alternative reality" by using a model which differs from the actual system in some specific, deliberate way. (For example simulating how a payload lifted off deck would respond to the ship’s current motions when the real-world payload is still safely placed on deck, as described in Section 3.1.)

Non-signal service Service that has neither input nor output signals. Typically a non-signal service may interact with the RTDS through event streams.

- Scheduler: Start up task/job/process according to predefined schedule or received event messages.
- GUI: User interactions through graphical elements. These elements can have properties like an event stream source, sink, or even processor. For instance, a button click (event source), an alarm indicator (event sink). Note that usually a GUI is a combination of several service categories, including, but not limited to: *Signal sink*, *Event stream source*, and *Event stream sink*.

This categorisation allows the flow of signal data through the system to be described in form of a directed graph [20]. *We suggest that a decision support system should limit signal connections to an acyclic directed graph. Otherwise, great care should be made to ensure that any cyclic data flow does not create instabilities through feed-back loops or similar.*

2.4 General-purpose services

A number of services can be implemented in a generic manner so that the service can be used in different situations without any need to revise the program code or similar. Typically, the operation of the service will be tailored to each specific through configuration files or similar measures. These general-purpose services are very useful and reduce the development time and cost to set up a new MDSS.

2.4.1 Job scheduler

In many cases there will be a number of jobs in the MDSS that should be executed at specified times, regularly or in certain situations. This could be maintenance tasks—such as removing old log files to ensure available storage space etc.—that do not directly affect the MDSS, but are important to the stability of the system. Other jobs may be essential for the inner workings of the MDSS, such as performing necessary calculations at certain events. One example could be that if new weather forecast data becomes available, a new job should run to simulate a planned operation under the forecasted weather conditions. Interaction with the MDSS could for example allow the job scheduler to start a specific job if a specified event occurs in an event stream in the RTDS, the event could be emitted by a button press in a user interface or by an “Event trigger”, see Section 2.4.9. Another possibility is to integrate the job scheduler with a custom made process manager that ensures all necessary services in the MDSS are running, see Section 2.5.4

2.4.2 Protocol converters

These services are concrete cases of “Receiver” or “Transmitter” services. A protocol converter translates between a specific external protocol and the internal MDSS middleware protocol. Well-defined industry standard protocols, such as e.g. NMEA [21] and Modbus [22], are well suited for implementation of general-purpose protocol converter services. Typically, these protocol converters should be configurable to the specific use case by configuration arguments or a configuration file. For some protocols it may be useful, and in some cases necessary, to implement both the “Receiver” and “Transmitter” services in combination. Note that such a combined service should not be categorised as a signal processor, as the signals to and from the RTDS will not interact. Some external protocols may require a “receiver” to acquire samples through polling a buffer, this may pose a few challenges, particularly if no information about when the sample was written into the buffer can be obtained (or alternatively, are we getting a new sample or the same as at last request?) First, buffering typically introduces jittering in the signal relayed to the RTDS, i.e. the timing of the final signal may deviate from the source signal. This can be particularly pronounced if the polling frequency is similar to the frequency of a periodic update of the buffer values from the signal source, in such cases minor deviations from

the desired sampling times can result in skipped sample points or reusing the same sample for two consecutive output messages. If it is known that the source varies much slower than the periodic update of the buffer, the problem of signal jitter in the RTDS can be reduced by lowering the polling frequency (still keeping it above the Nyquist frequency of the highest frequency components of the source signal). This leads to the second point, namely that if the "Receiver" is configured to poll regularly at some frequency required by the user of the signal in the RTDS for example, there is a risque that aliasing from higher frequencies may distort the signal if the source signal contains frequency components above the Nyquist frequency given by the polling frequency. For external protocol implementations that *push* data to the "Receiver", the above issues will generally not be of importance (although small amounts of jittering could be introduced). However, the signal conversion may be complicated if the "Receiver" is configured to send messages comprised of non-atomic pieces of information that are *pushed* from the source in separate messages. In such cases, one must allow for some time period to receive all the relevant information and have logic in place for handling missing information.

2.4.3 Logger

A general-purpose logger (or recorder) service enables easy storage of real time data from an MDSS. The output of the logger may be that the data is inserted in a database or alternatively stored in files. Our advise is to use a self-describing storage format, i.e. information such as signal names, description, origin, units and so on should be included in the stored files or database. This simplifies use of and reduces risk of misinterpreting historic signal logs especially in cases in which the configuration of the MDSS have gone through changes. In many cases, operations on persistent storage such as writing, opening and closing of files, may take more time than the available time between successive samples at the specified sampling frequency. Therefore, multiple threads will normally be needed to separate sampling of signal values from the RTDS and writing the values to the persistent storage. Some kind of mutexed buffer storage in memory is needed to pass the signal values to the *write thread*. Depending on the specific case, different requirements may have to be satisfied by the logger. One case would be that a selection of signals should be polled at a set frequency and the values stored together with a common time stamp. This case could be extended by having groups of signals with different poll frequencies. A different case would be that all incoming samples—whether they arrive regularly or not—should be stored, possibly together with a timestamp, for instance using a time series database, such as for instance InfluxDB or TimescaleDB. A specific implementation of a general-purpose logger does not necessarily have to able to work for all such different use cases.

2.4.4 Co-simulation service

This service provides a way to run a co-simulation in real time, connected to the real-time data space. That is, input and output variables in the simulation can be associated with signals in the RTDS, so that the simulated system appears to the MDSS much like a physical system would.

Compared to other simulation methods, co-simulation has some features which are advantageous in this context. First of all, the data exchange among entities in a co-simulation is based on relatively low-frequency signal sampling, just like the RTDS. This allows the two domains to be connected seamlessly. Secondly, the use of co-simulation facilitates "pluggability", especially if a common interface like OSP-IS/FMI is used.[3] For a more in-depth discussion of the advantages and best practices of co-simulation in a maritime systems context, see [10].

A co-simulation service can be built using the *libcosim* software library developed by the OSP project, which provides the necessary simulation algorithms and subsystem interfaces. The main thing that is needed in addition is a bridge module that reads signal samples from the RTDS and injects them into the simulation via input variables, and does the reverse operation for output variables. This is illustrated in Figure 2.3. *libcosim*

[3] *Open Simulation Platform Interface Specification* (OSP-IS), a common interface specification for maritime co-simulation models. FMI is the *Functional Mock-up Interface*, a lower-level interoperability standard upon which OSP-IS is based. See `https://opensimulationplatform.com` and `https://fmi-standard.org` for more information.

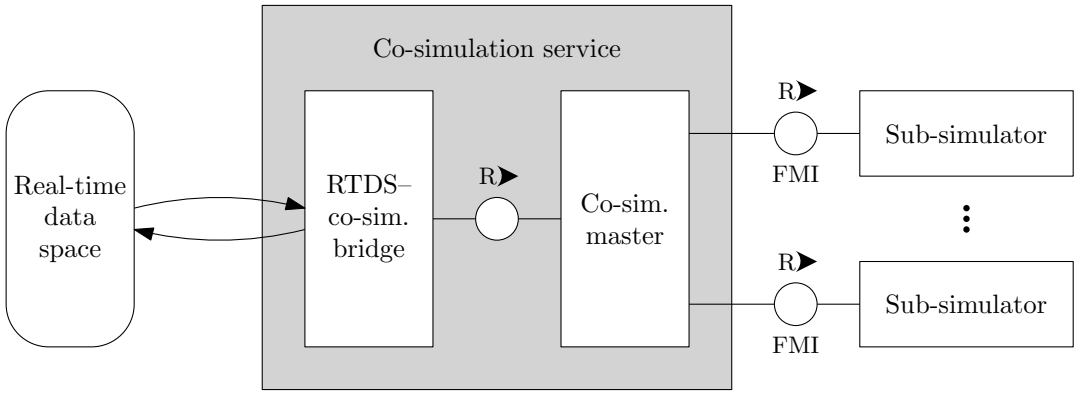


Figure 2.3: FMC block diagram of a co-simulation service.

provides more than one way to achieve this (but the most straightforward is probably via its `observer` and `manipulator` interfaces).

2.4.5 Mathematical function

These are useful components that can use input signals as input to a mathematical expressions and transmit the result on a different signal, and are thus examples of *Signal processors*. The usefulness of these components is in how configurable they are, let's consider the case of transforming a temperature signal measured in Fahrenheit to a temperature signal measured in Celcius. One could implement a special purpose component in which the conversion is hard coded as:

$$C^\circ = (F^\circ - 32)\frac{5}{9} \tag{2.1}$$

However, that component would have very limited use. A simple improvement would be to make the component configurable by replacing the constants, 32 and $5/9$ by parameters that can be set in a configuration file (signal names should also be configurable). A much more valuable improvement is to enable entire mathematical expressions to be configurable, that is, output signals can be specified as a general mathematical expression using input signals, common mathematical operators and functions. In this case the configuration file could include specifications of mathematical expressions such as:

```
signal_out_1 = signal_in_1 / signal_in_2
singal_out_2 = sin(signal_in_1 + signal_in_3) + signal_in_2 * signal_in_4
```

Such a general purpose mathematical function component can be used for many different calculation tasks without any need to implement new code or any compilation of the service code. Further, adjustments can be made by simply adjusting configuration files and restart the software. Typically, a *Mathematical function* service can be implemented by using a (third party) library for parsing mathematical expressions. Such libraries are available for several programming languages. Another approach would be to implement the service as a wrapper that enables passing input signals into a runtime environment of an interpreted language such as e.g. Julia or Python, and to pass the output values from the interpreted language runtime back to the RTDS. The expressions for calculating the output signals with the given input signals would then be specified in the interpreted language in a script file or similar. This approach would typically be more flexible and allow for a wider range of calculations. However it will introduce additional dependencies and may be more complicated to implement and to deploy/install.

2.4.6 Filters

Signal filtering is often needed when real-time signals from real sensors are used as input to an MDSS. In a real-time context, *causal digital filters*[4] are relevant. There is a wide range of useful discrete filters, and they are well-defined *mathematical functions*, which can be entirely configured by parameters to become operational services. Filtering can have several goals, for example: noise suppression, outlier detection, extraction of slow trends, fast variations, or specific frequencies. It can be advantageous to include dedicated filter services, because it reduces the implementation burden instead of integrating it into other, more complex services. In addition, some filtered signals may be needed by several services, so sharing these signals are then made possible using the middleware. Finding a suitable filter and its parametrisation either requires knowledge on the characteristics of the signal source, or a representative dataset, for which analyses can be done. It is important to know the properties of a filter, including expected performance, but also its limitations and possible causes of issues. Some filters may become unstable if not properly tuned, or if the assumptions regarding input signals are violated. In some cases, a filter may require a non-standard modifications, in which case a dedicated filter service may not be applicable. Some classes and examples of relevant filters are:

***Finite impulse response* (FIR) filters:** moving average, {low,band,high}-pass, bandstop

***Infinite impluse response* (IIR) filters:** autoregressive, Butterworth, Chebyshev, Cauer, Bessel

State-space model filters: Kalman, Luenberger observer

Other filters: median, wavelet denoising, outlier detection

2.4.7 Fourier transform

To analyse the characteristics of signals or sequences, such as logged time series, often *Fourier transform* (FT) is used [23]. Signal characteristics, such as the frequency components of the signal, can reveal information about the data sequence that are not necessarily easily obtained just by looking at the raw signal. The signal characteristics are in some situations crucial for detecting important system behaviours. One example is that monitoring the frequency components in a roll motion of a vessel in an offshore lifting operation can help to detect unstable resonance situations and help stabilising the vessel by adjusting the ballast. Figure 2.4 shows logged roll motions and a corresponding FT analysis. Depending on the use case, an FT service may transmit signal characteristics for a specified time period regularly, on request, or continuously for a sliding window on the latest incoming signal. In general, the service will include buffering of the incoming signal to ensure the required data is available. In practice, FTs are most often implemented using *fast Fourier transform* (FFT), a fast algorithm for computing the *Discrete Fourier transform* (DFT). Note that the input signal is required to have a constant sampling rate for the FFT to be valid. This requirement could easily be violated if signal polling is used and no signal low-pass filtering is used.

2.4.8 Playback

A playback service is here considered a service that reads *signal logs* from the persistent data space and transmits the values as *signals* in the real-time data space. In effect, historic data will be played of as if they were incoming signals. The service could use the time stamp of the signal log to reproduce the rate of messages in the real time data space, or even to e.g. play off the historic signals at a specified higher frequency. The time stamps may be stripped in the process and not transmitted in the real-time data space. The "Playback" service can be useful for testing and development of the MDSS, or to enable the user to review historic operations for example. It could also be used to feed signal logs (historic data) to other components that expect signal inputs from the RTDS—e.g. digital twins, simulations, statistical analysis etc.—but specialised components that are expected to use historic signal logs could alternatively use the signal logs directly.

[4] A causal digital filter only depends on past and current sampled, discrete-time inputs

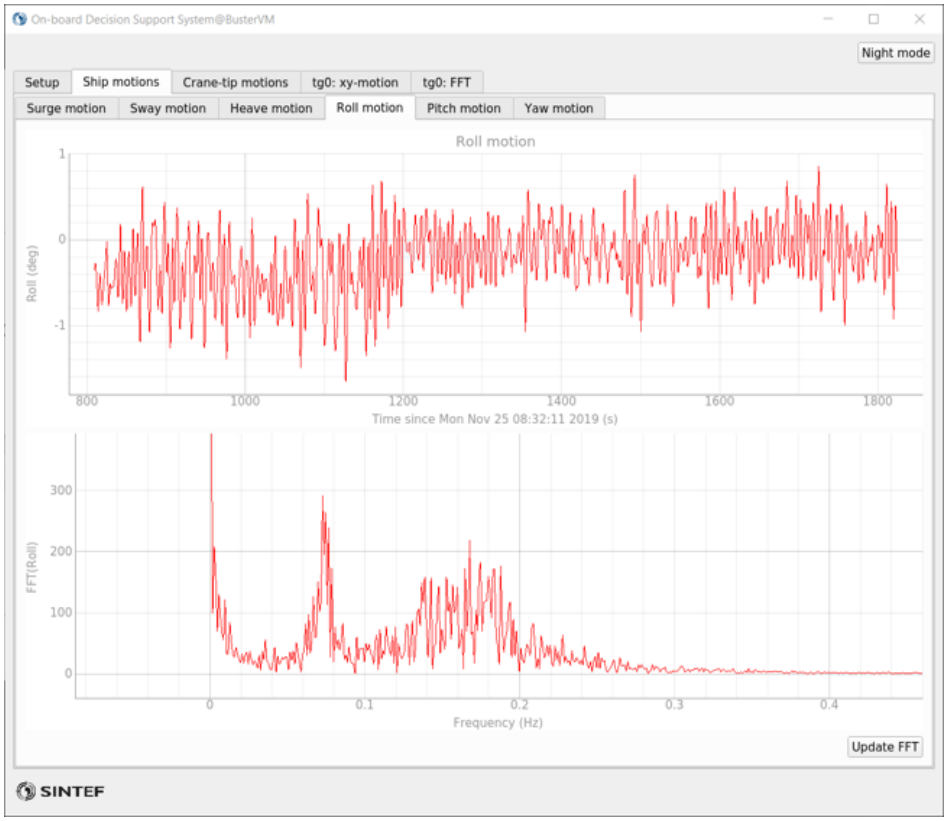


Figure 2.4: Example illustrating the use of FFT to analyse the roll motion of a vessel in DP.

2.4.9 Event trigger

An *event trigger* is a service that has an event stream output channel. The input message(s) may come be any combination of signals or event streams. In the case that the input messages fulfil a set of conditions, an event is emitted. A condition can in general be any mapping from input to output, but is typically a mathematical or boolean expression. Boolean signals (button pushed), exceeding threshold value (high temperature), and flatline detection (signal loss) are all examples of useful conditions. The purpose of an event trigger is typically to notify another service or resource regarding special conditions so that actions can be taken. For example, a triggered event can be used by a process manager to restart some service or job. Other examples include notifying a human operator by means of visual or audible alarms.

2.5 Practical considerations

We noted in the introduction that AMADEA is not a complete, implemented architecture. The first “A” stands for “adaptable”, which refers to the fact that it should be possible to implement the architecture on top of existing systems, using a variety of technologies, for a variety of purposes. Even so, we do wish to give some concrete suggestions and advice. In this section, we will discuss things that are not part of the architecture per se, but which will be important for anyone working on an actual implementation. In particular, we will describe several specific choices of technologies and standards that can be used to implement the concepts and services defined previously in this chapter.

2.5.1 Communication middleware

AMADEA is a service-oriented architecture [2], primarily with a message-oriented data-sharing scheme. Such an architecture may imply that a sensible choice is a *message-oriented middleware* (MOM). A key feature of a MOM is its ability to send and receive messages between processes on a distributed system in a simple manner. The entities in a MOM are loosely coupled, with a universal interface definition format[5], which facilitates development of software components that can exchange messages. Only an agreed interface definition is needed to implement the message passing.

A benefit of using a middleware is that the developers can keep focus on the main objective of a product or service instead of core features provided by a *commercial of-the-shelf* (COTS) middleware framework. An appropriately chosen middleware simplifies the development, with standardised APIs, maybe with a location transparent addressing scheme, a reliable publish / subscribe paradigm, service discovery, *quality of service* (QoS) configuration capabilities, and interoperability.

Suppose that an MDSS consists of software services from a range of vendors. The ideal situation would be a mutually agreed communication middleware. Within the realm of MOM, there exists several protocol standards, such as *Message Queue Telemetry Transport* (MQTT), *Advanced Message Queuing Protocol* (AMQP), *Constrained Application Protocol* (CoAP), and DDS [25]. Corsaro [26] compares these standards and concludes that DDS is the best candidate. The same author later did another evaluation of two popular *industrial Internet of things* (IIoT) standards, namely DDS versus *Open Platform Communication Unified Architecture* (OPC UA) [27]. Since 2016, this technology landscape has evolved, for instance the *OPC Unified Architecture* [28] standard has been extended to also include a publish / subscribe paradigm in addition to [29], which mainly concerns controller-to-controller communications[6]. Simultaneously, the *Object Management Group* (OMG) has extended the family of standards pertaining to DDS, with notable additions such as:

***Remote Procedure Call over DDS* [30]** Adds support for a so-called request / reply paradigm over DDS.

***DDS Security* [31]** Standardise security features such as authentication, encryption and access control.

***DDS for eXtremely Resource Constrained Environments* [32]** Allow extremely resource constrained devices (microcontrollers) to communicate with the DDS data space via a broker.

***Web-Enabled DDS* [33]** Define means for applications using standard web protocols to participate as publisher / subscriber in the DDS data space.

***OPC UA/DDS Gateway* [34]** Interoperability and information exchange between systems that use DDS and systems that use OPC UA.

The authors of this document has experience using the *DDS Interoperability Wire Protocol* [35] for various MDSSes, and it was chosen due to its key benefits[7]. This standard has been implemented by several vendors, for which both closed source software [37, 38] and open source software [39, 40] exist. An up-to-date list of software vendors can be found in *Data Distribution Service (DDS) Guidebook* [41]. The *Data Distribution Service (DDS) Guidebook* is a vendor-independent resource that provide useful information such as real-world user scenarios involving DDS.

2.5.2 Storage

The best choice of storage solution will depend on several characteristics of the MDSS. First, the topology of the system can be important; will the MDSS have services running on multiple computers or will all services be running on the same computer? Second, who will need access to the stored information, or from where?

[5] The *interface definition language* (IDL) format [24] is a recent ISO standard for *Data Distribution Service* (DDS).
[6] Since the OPC UA standard comes from the automation industry, a *controller* is a low level field device or I/O peripheral.
[7] "Performance, scalability, robustness, Reliability and QoS" [36].

Third, is it important that no data will be lost? Answering these questions may help deciding on which will be the best storage solution. A few key choices will be discussed here.

Will local storage or cloud storage suit the specific MDSS best? Local storage may be practical and easy to set up if the stored data is primarily used locally, or e.g. regularly transferred to some remote location. Conversely, for situations in which access to the stored data is needed from multiple computers running the MDSS, a cloud storage solution may be better. A private cloud solution on the local network may serve this purpose well. If a cloud solution that allows for sharing data with other vessels or on shore users is needed, available communication may be an issue, see Section 2.5.3.

Should the data be stored in files or a database? Again, this will depend on the intended use. One case could be that some *signal logs* should be stored and transferred for on shore analysis or permanent storage. In that case, a series of files in which the files contain successive temporal intervals of the signal logs may be a good choice. This simplifies transfer of the data to a remote location and will also limit the amount of data that may be corrupted if e.g. the writing process should be abruptly aborted by some external event and the current file is not closed correctly. A very different case could be that some mapping of values for different conditions—e.g. statistical values for vessel motions under different weather conditions—should be updated as new operational data becomes available and perhaps simultaneously be queried by another component in the MDSS. In this case a database accessed through a central service in the MDSS, may be the best choice.

By using standardised file formats and following conventions for how the data can be self describing—this may include variable naming and expected metadata—exploration and use of the data can be greatly simplified. A number of tools to handle the data may be available and e.g. software for plotting of the data may be able to read the necessary information without any further adaptation.

2.5.3 Ship–shore communication

Ship-to-shore communication is in some circumstances necessary, but the requirements for such a communication link greatly depend on the specific decision support system in question. The offshore communication capacity is high, with a range of capabilities including sub sea fibres, 4G, line-of-sight, as well as satellite-based communication links. These different communication solutions can in most circumstances provide the necessary quality of service requirements, such as bandwidth and latency—often at a premium cost. As a result, a tradeoff between a real-time communication link and acceptable operational expenses is made. If, for instance, a vessel performs a time-limited operation at sea and returns to shore in reasonable time, one can exploit a cheaper and more accessible communication infrastructure close to shore by delaying the data transfer. This is particularly useful in cases where data collection is performed during offshore operations, but availability of these data are not needed until a later time point, e.g. for retrospective analyses and service development.

When timely data access is desired, one can devise a variety of approaches to achieve this. One example is edge-based data analyses with the purpose of insight harvesting and data reduction, so that the bandwidth requirements are reduced. This approach may only expose a subset of the data that are available in the RTDS, usually by other means than extending the RTDS to shore; for instance achieved by transferring databases or files on disk in the form of an encrypted, session-based transfer. Suppose that the entire RTDS is to be extended to an operations centre located on shore. In most circumstances, this would require the use of a *wide area network* (WAN), which necessitates additional security measures and route configurations. Common techniques involving a *virtual private network* (VPN) or *Transport Layer Security* (TLS) could be elements in such a solution, but challenges such as company compliance requirements and firewall settings may be hurdles on the way to success. In the case of DDS as middleware, different vendors provide approaches to deal with {edge, cloud}-to-{edge, cloud} permutations. *DDS-Router* [42] is an example of software that deal with inter-site communication.

2.5.4 Service and job management

Service and job management is an essential part of an MDSS system; their task is to ensure that all necessary services are started and running and that necessary jobs are started according to a predetermined schedule or when they are demanded by the MDSS. The service management should be carried out by a *Process manager*, as described in Section 2.2.3, that can be used to monitor the process and ensure that it is running, not only start the process without any subsequent monitoring. The job management is a little different as it is mainly concerned with starting jobs, not necessarily monitoring the running job, see Section 2.4.1.

Both of these task could be carried out by software implemented as part of the MDSS, either specialised *Process manager* and *Job scheduler* or alternatively a service that is adapted to both purposes. This approach has some advantages over using available software: the software can be made cross platform if needed and it can be given the ability to interact with the RTDS. Interaction with the RTDS could be used to improve fault detection in monitoring services for a *Process manager* or to enable a *Job scheduler* to start a job if an events occurs in an event stream, among other things.

If the above mentioned features are not needed, existing software that is available through the operating system, package repositories or elsewhere may be able to efficiently do the service and job management. Such software will typically be mature, robust and well tested, as they have been been used in a lot of different situations and many bugs will have been discovered and fixed. Some examples of software that could be used for service or job management:

- *systemd*—software used by many linux distributions to initalise the user space and manage user processes, among other things. Can also manage resources and run scheduled tasks.
- *crond*—software on UNIX-like systems that runs scheduled jobs, that is: commands or shell scripts.
- *supervisord*—a process control system for UNIX-like systems: monitor and control processes.
- *Windows task scheduler*—Windows software that start programs or scripts according to schedule.

2.5.5 Existing systems

Within the maritime industry of Norway, there exist several systems that can form the backbone for a MDSS. The decision to choose an existing in-house or a new solution based on third-party software depends of course on many factors and criteria. Usually, a company has an existing family of products / solutions that are based on former technological choices. The decision to migrate from one technology to another may be associated with considerable development cost and cannot be taken lightly. If interoperability with systems delivered by other—possibly competing—actors in the sector is important, a common middleware would ease integration considerably, see Section 2.5.1. In practice, integration between dissimilar systems is typically achieved by using protocol converters. Norwegian companies with activity within maritime industry known by the authors to be using DDS software are:

Ulstein Group Makes use of *X-CONNECT®* [43] in various automation solutions;

SINTEF Ocean Uses [4] for on-board data collection and MDSS demonstrators;

Brunvoll *BruCon* [44] makes use of DDS;

Kongsberg Gruppen Kongsberg Defence & Aerospace offers *InterCOM DDS* [38], but it is not known to be utilised within their maritime solutions;

Maritime companies also make use of several other middlewares and ecosystem solutions to build MDSSes. Large companies such as Kongsberg Gruppen offers several solutions targeting various aspects of maritime operations, including EcoAdvisor, K-IMS, K-Pro, K-SAFE, CAF, and Kognifai. Elaboration of these and other ecosystems is beyond the scope of this report.

Chapter 3

Case studies

In this chapter, we give several examples of how concrete decision support systems can be built using AMADEA, and more specifically using the building blocks described in Chapter 2. The case studies cover a number of different operations. We give particular attention to offshore lifting operations, with four cases that have been studied during the course of SFI MOVE. In addition, we have included three examples of DSSes for fisheries and offshore seismics from other projects. We provide somewhat less information about these, and instead refer the reader to previously published work for more details. In all cases, we have omitted many details concerning the involved models and analyses, as the main purpose is to demonstrate the use of the architecture, not to provide complete MDSS specifications.

Each of the MDSSes described here make use of at least one component that has *not* been described in Chapter 2. These are *special-purpose components*—components which perform a task that is so specific to the MDSS in question that it makes little sense to make them part of a generic architecture. In all cases, it is a design goal to keep the number of such components at a minimum, and instead try to employ reusable general-purpose components.[1]

We will use the FMC block diagram notation to graphically show the structure of each MDSS in terms of services, jobs, signals, and so on. Some of the FMC diagrams are quite large, and we have therefore made a small simplification in an attempt to keep them uncluttered: By nature, *jobs* are always run by some service, but these services are often not shown in the diagrams. Instead, whenever an event stream is linked to a job, there is an implicit, hidden scheduler that runs the job. This is illustrated in Figure 3.1.

[1] Reusability is by no means *the only* design goal, though. As we noted in the introduction to Chapter 2, sometimes it makes sense to combine or specialise what could otherwise be general-purpose components to meet requirements for performance, accuracy, and so on.

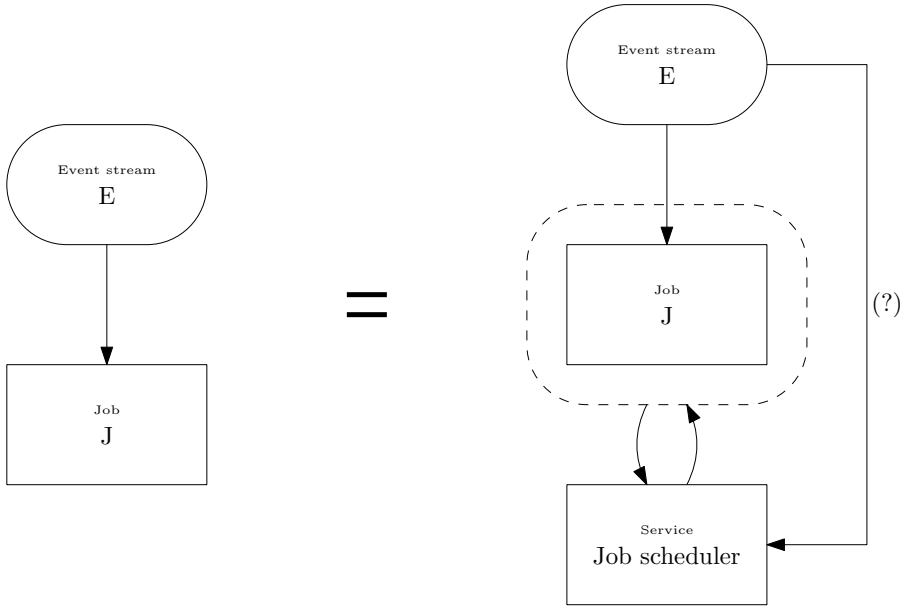


Figure 3.1: A simplification of the FMC block diagrams in this chapter: Whenever an event stream *E* is linked to a job *J*, there is an implicit hidden job scheduler that runs the job. The question mark indicates that the job may be run in response to the event (or some other event for that matter), but it may also be run on a periodic schedule. This should be clear from the context.

3.1 Wind- and ship-induced motions of a lifted GRP cover

Lifting a *glass fibre reinforced polymer* (GRP) cover from a vessel's deck, over the side and sub-sea is a common marine offshore operation. Such covers can be relatively light, typically about 5-10 tons, and therefore, such lifts are considered light lifting operations in the scope of marine offshore installation vessels[2]. Nevertheless, lifting GRP covers in strong winds and/or in high sea states might accelerate the pendulum motions and, in worst case, hit equipment or personnel positioned on the vessel's deck. Hence, having decision support regarding the safety of lifting such a GRP cover off the deck in the current environmental state could give valuable inputs for determining operational safety and feasibility. And, if severe conditions that affect the operation are predicted, actions can be taken to further increase the operational safety. Moreover, the type(s) of action(s) can also be determined based on decision support provided by the MDSS, such as for instance suggesting adding tugger wires to stabilise the payload, see Section 3.2.

3.1.1 Methods

In this case study, the vessel is assumed to be in a DP-operation, in preparation for starting the lifting operation. Before starting the lifting operation, a *what-if* analysis is to be executed based on the current vessel motions and environmental conditions, to determine the safety of lifing a GRP cover off deck. Real-time-, or historical, data for the vessel motion and environmental conditions, such as wind speed and direction, are fed into the MDSS, which passes these data on to a simulator that predicts the outcome of the operation. Here, logged data from Olympic Challenger is used. Note that only the vessel's roll, pitch and yaw motions are fed to the simulator because the surge, sway and heave data were not logged, and, the wind speed and direction are not known. Hence, from the model's perspective, the environment is considered windless even though the model supports wind speed and direction inputs.

The GRP cover is modelled in *fmiCpp*, an in-house developed C++ framework for rapid creation of co-

[2]Meaning that the effect of the lifted GRP cover on the vessel's motions is negligible.

simulation *functional mock-up units* (FMUs). The reason for modelling the system as a co-simulation model using FMU is to facilitate a stand-alone simulator and to provide a generic model API such that the software code used for running the model can be made more or less generic. The cover itself is modelled as a massive rectangular mass which can "land" on the vessel's deck. This mass is connected to a crane through a wire and two payload slings. The wire and the slings are modelled as lumped masses using Baumgarte stabilisation [45], a method for solving constrained dynamical systems explicitly. The state-space model is integrated using the variable Runge-Kutta 2 method.

The vessel measurements are also coded as an FMU in this case. However, this FMU only contains functionality for playback of historically logged data, and interpolates between the logged data points, for time synchronisation purposes. Later, the vessel measurements is expected to come from a different data source. A step towards such an MDSS setup is demonstrated in Section 3.2. To setup the analysis, the Kopl[3] software is used, which also can run *cosim*[4] in the background.

3.1.2 Components

Here, the MDSS setup consists of multiple components, as shown in Figure 3.2. In addition to services collecting data from the ship's sensors, a co-simulation service is running. Based on previously logged vessel data, or live data from the vessel, the co-simulator can predict what will happen if the GRP cover is being lifted under given conditions. The simulation data are distributed and the advisory module processes them to provide further operational advise to the operator. These advises are distributed through the RTDS and sent to a suited front-end system where the decision support is presented.

The data connections between the GRP model and the real-time data space is shown schematically in Figure 3.3.

3.1.3 Results

The GRP cover is in the simulation first hoisted up from deck, and the length of the crane wire is about 8.3 m (slings excluded) before the hoisting stops. A simple graphical representation of the GRP cover when hoisted is shown in Figure 3.4 for four different time points. Note that the red colour in the figures shows the trace of the position of the (x, y)-position for the GRP plate from $t = 0$ s until the current time point.

Figure 3.5 shows a density plot of the motions of the GRP plate in the relative North/East plane (bird-view). Note that relative here means that the initial position for the payload in (North,East)-coordinates is (0,0). Such results may be helpful to plan the lifting operation with respect to the position of the cover on deck. In this case the heading of the vessel is more or less constant, which also means that the relative positions in the figure could be directly interpreted as relative positions on the vessel's deck. Moreover, one could further analyse the motions of the payload to figure out if one should add e.g. tugger wires or not. Figure 3.6 shows a different representation of the GRP cover motions in addition to FFT analyses of the results. Note that FFT can be a separate module in the MDSS, as described in Section 2.4.7, but this is omitted here and the FFT is assumed to be run in the advisory module. Such FFT analyses can be of interest to check if the payload is able to excite any natural frequencies for the vessel, causing resonance and, in worst case, resonance and unstable system behaviour.

Figure 3.7 shows the crane wire tension along with its FFT. As seen, the tension is about zero in the beginning of the simulation, only affected by the weight of the crane wire itself, since the GRP cover is resting on the vessel's deck. However, the tension is increased when the hoisting is started, and for a short period of time about half the weight of the cover is taken up by the crane wire. After about 50 seconds of hoisting, the cover is airborne, solely held by the crane wire. Note that the peak frequency in the FFT is different in comparison to the isolated axis motions shown in Figure 3.6.

[3] `https://open-simulation-platform.github.io/kopl`
[4] A command line application for running co-simulations

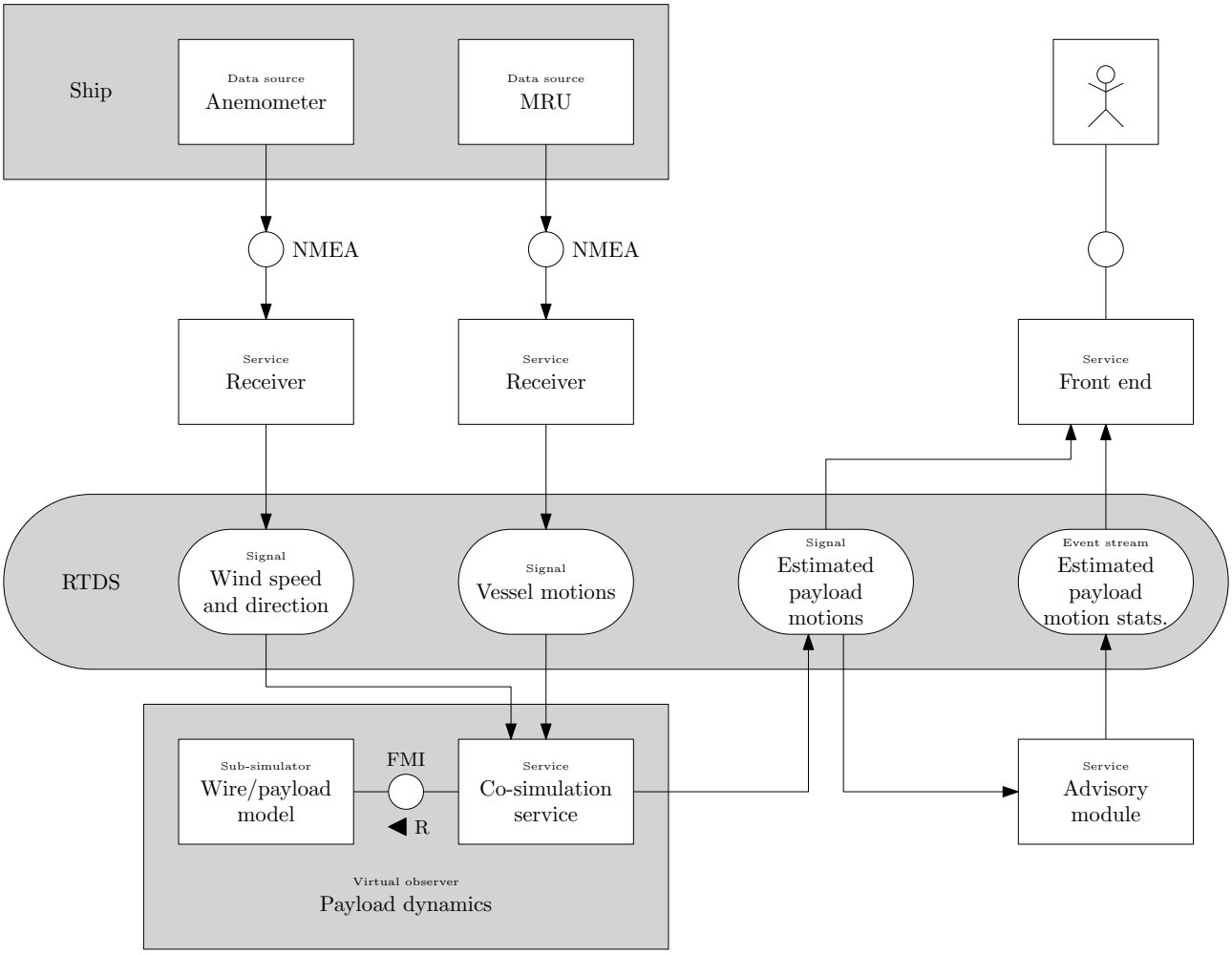


Figure 3.2: FMC block diagram of the MDSS for simulated lifting of a GRP cover based on live data from the vessel.

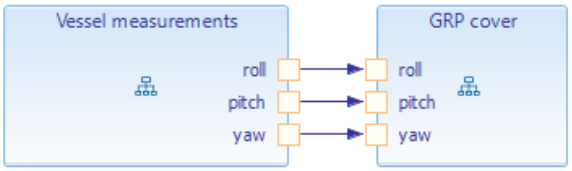
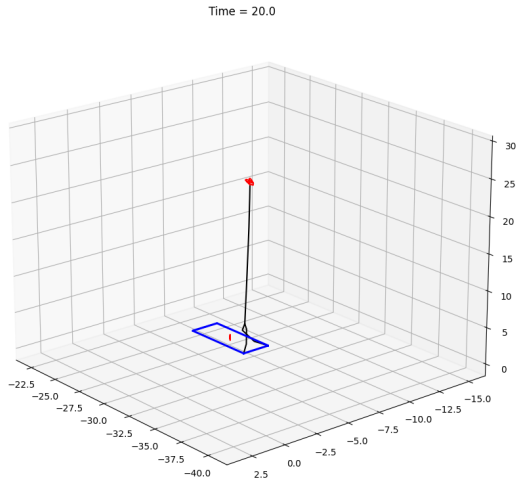
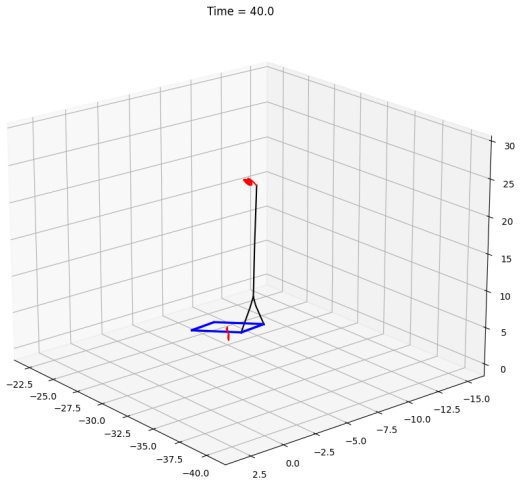


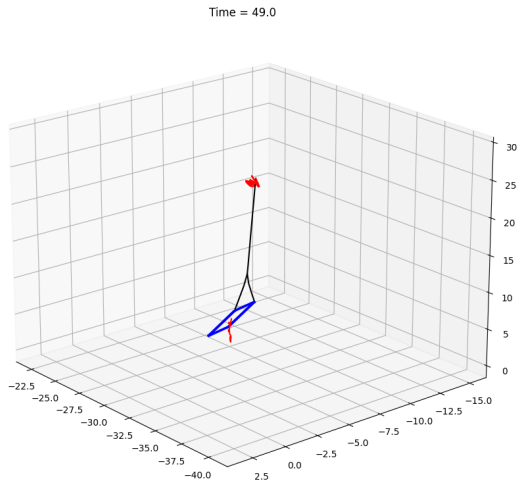
Figure 3.3: Simulator components and connections. Note that only roll, pitch and yaw measurements are logged and sent to the GRP-model in this case.



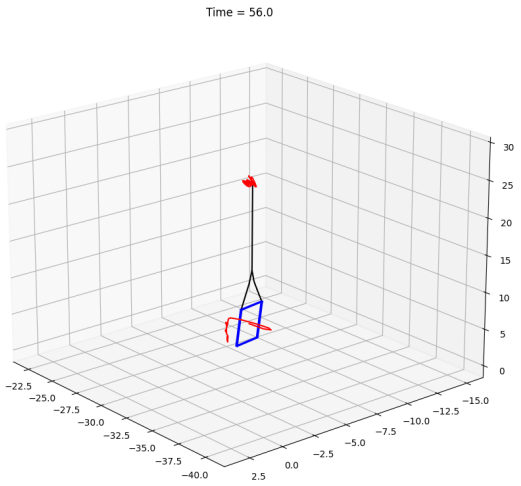
(a) GRP cover at deck, crane wire being reeled in.



(b) Starting to lift the cover off deck.



(c) GRP cover almost off deck.



(d) GRP cover off deck.

Figure 3.4: A simple 3D visualisation of the GRP plate being lifted off the vessel's deck. Note that the vessel's deck is not shown in the figures.

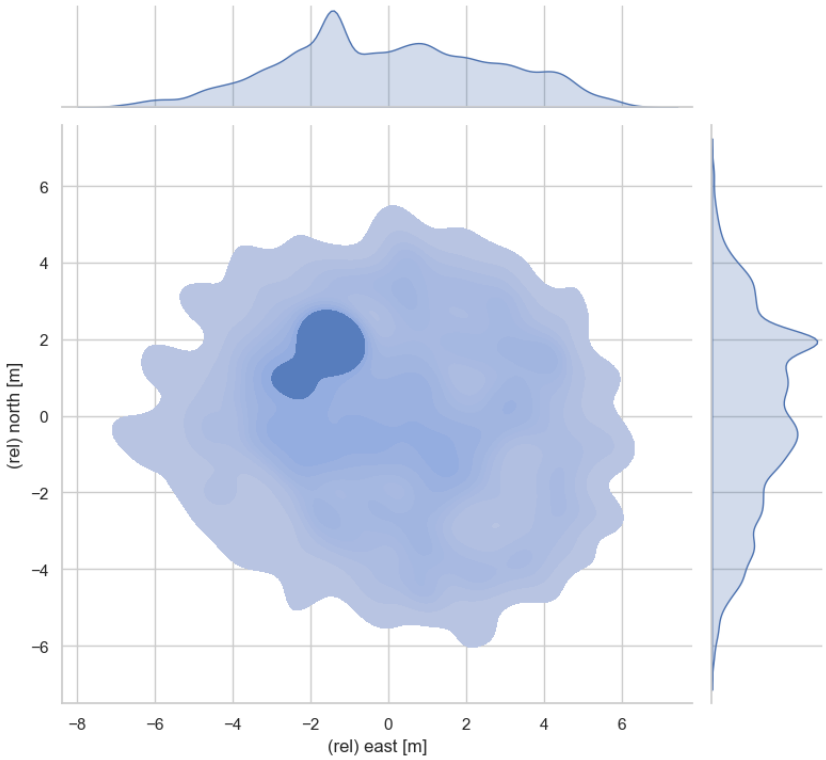


Figure 3.5: (Relative) North/east position density plot for the GRP plate (bird view of motion density).

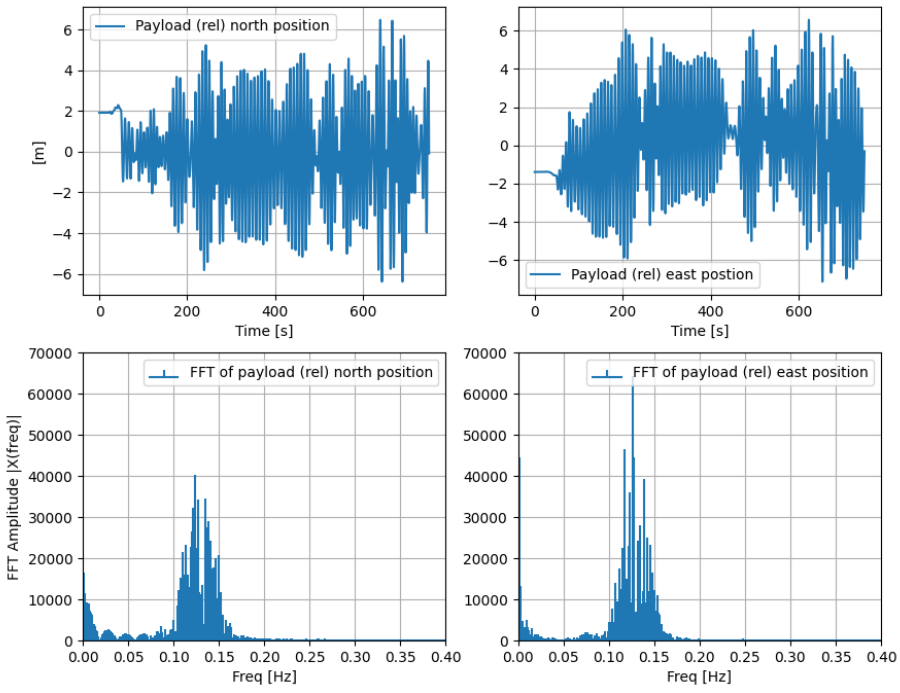


Figure 3.6: (Relative) North/east position and corresponding FFT for the GRP plate.

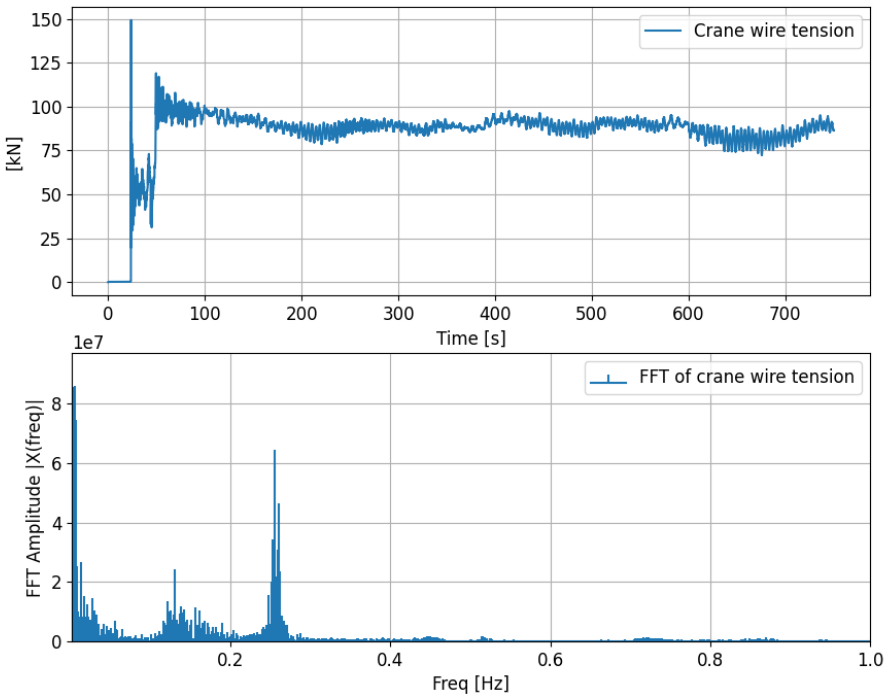


Figure 3.7: Crane wire tension for the GRP plate case.

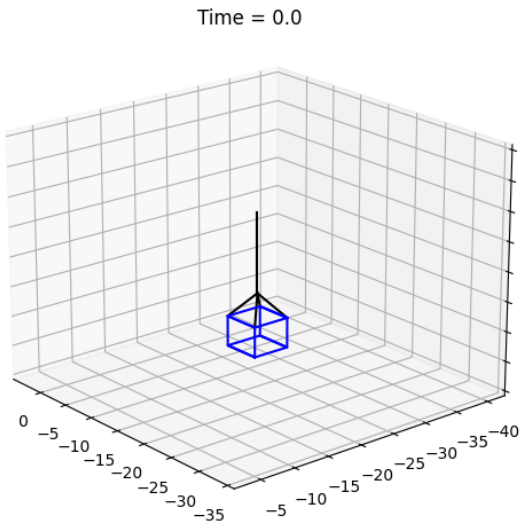
3.2 Tugger configuration

A similar simulation case to lifting a GRP cover, presented in Section 3.1, is to use simulations as a tool for planning tugger wire configuration and setup in a lifting operation. Tugger wires are applied to in-air payloads to restrict the pendulum motion of the payload by providing additional damping (tension controlled tugger winches). However, the configuration of the tugger winches, e.g. how many and on which axis they should be applied, and the tension control set-point are highly dependent on the payload characteristics, the operation and the environmental conditions. Using *just-in-time* simulations right before starting the operation, with freshly logged vessel data (assuming a light lifting operation) or a vessel simulator tuned for the current conditions, can provide useful insight in determining the configuration and execution of the operation. In this case study, we demonstrate the use of both logged data and a vessel simulator for determining the need of tugger wires in a specific operation. Note that it is possible to setup many experiments testing different settings, from tugger wire cable dimensions, length of crane wire, tugger winch positions to tension set-points.

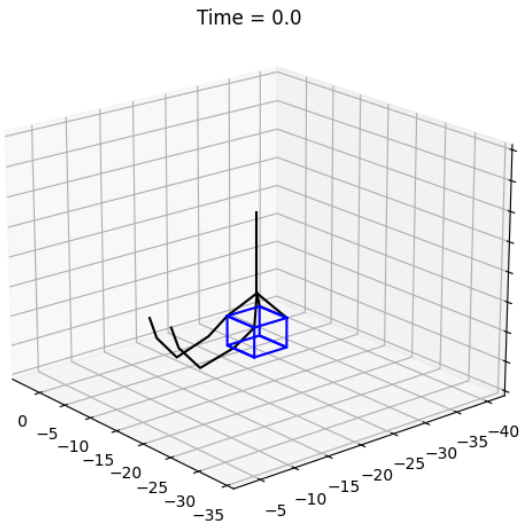
3.2.1 Methods

All wire models are assumed to be lumped bar elements connected by constraints and modelled using *fmiCpp*. The constraints are expressed as constraint forces using Baumgarte stabilisation [45]. The payload is assumed a rectangular box which is connected to the crane wire using four slings. In the the first study, in which historical vessel motion data are used, one configuration with two tugger wires and winches is compared to one configuration without any tuggers. The tuggers have a tension set-point of 7500 N, the payload is 5000 kg, and lifting this mass is considered a light lift. The tugger winch controllers that are controlling the tension of the tugger wires have the main objective of damping the pendulum motion of the payload and to restrict the motions in their installed direction. In this case, both tugger wires are fixed to the crane pedestals, which

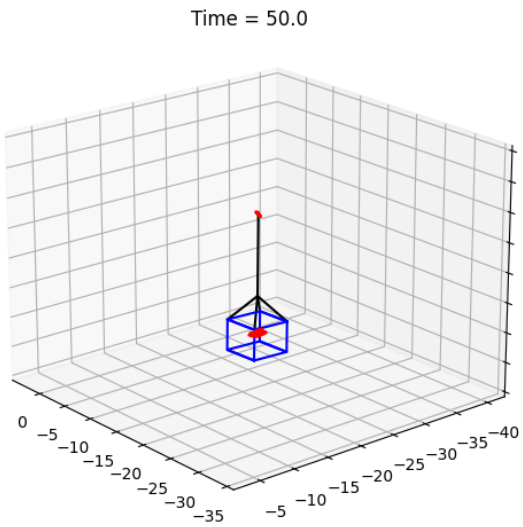
means that they restrict the motion of the payload in the crane's extended direction. On the payload side, the tugger wires are connected in two of the upper neighbouring corners of the payload, in the same locations as two of the slings are connected. Figure 3.8 shows a simple presentation of the payload motions for the two cases, for two different time steps.



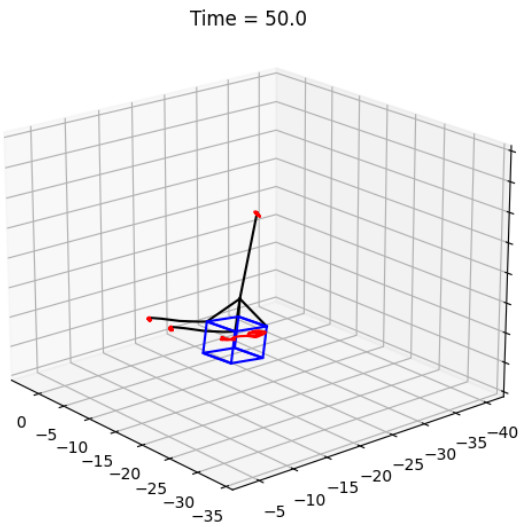
(a) In-air payload without tugger wires at $t = 0$ s.



(b) In-air payload with two tugger wires at $t = 0$ s.



(c) In-air payload without tugger wires at $t = 50$ s.



(d) In-air payload with two tugger wires at $t = 50$ s.

Figure 3.8: Simulation setup for two different virtual observers, one without any tugger wires and one with two tugger wires. Note that the tugger wires are slack in the start of the simulation, but are tensioned by tugger winches to a given set-point during the simulations.

For all models, except the tugger winch controllers, the variable Runge-Kutta 2 integrator is used. The tugger winch controllers are integrated using the forward Euler integration method.

For the logged vessel motions, the same vessel measurement model used in Section 3.1 is used here. A vessel model describing Olympic Challenger in DP is exported from SIMO [12] as an FMU and to be used as the source of vessel motion data in the simulations. The simulated vessel motions will be useful when the loads are too heavy for the light weight assumption or in near future prediction cases for which the currently logged vessel motions would not be valid (e.g. due to changes in the weather conditions).

3.2.2 Components

The different components needed in the MDSS and their connections are shown in Figure 3.9. The co-simulation service orchestrates the co-simulation which contains multiple sub-simulators in this case. Also, the co-simulation service sets up the connections between the sub-simulators and feeds the simulation with relevant data from the RTDS. The first case using logged vessel motions is shown in Figure 3.10 and consists of four sub-simulators and a vessel motion source. In the latter case the logged vessel motions are replaced by a vessel sub-simulator. Also, the logger stores all logged data from the real-time data space for later usage. Note that each tugger configuration requires its own vessel model and that the vessel simulators get force feedback from the payload sub-simulators — from the crane wire and the tugger wires, as well as the forces from the tugger winches. When using vessel sub-simulators, we also employ all degrees of motion for the vessel model[5]. From the simulation results, which are distributed on the RTDS, an FFT-service analyses the results for the virtual observer case and forwards the corresponding results to the front-end, through the RTDS. In the predictor case a predictor-job containing a similar simulation setup as the virtual observer, but also with a vessel model, execute simulations based on event streams and distribute the results as new event stream data on the RTDS. These data are further fed to a statistical analysis job, which analyses the simulation results and forwards the analysis results to the front-end service through the RTDS.

3.2.3 Results

Virtual observer

Some of the simulation results are compared for the two tugger wire configurations in Figure 3.11. Figure 3.11a compares the crane wire tension (time series) and the corresponding FFTs for the two tugger winch configurations while Figure 3.11b shows the tension in the tugger wires and the corresponding FFTs. Maybe the most interesting results are the FFTs, which show critical frequency regions. As an example, in the case with two tugger wires, there is a new peak region of frequencies (≈0.27 Hz) in the FFT for the crane wire tension, Figure 3.11a, in comparison to the unrestricted payload case. The same peak regions can be found in the tugger tension results in Figure 3.11b.

Figure 3.12 compares the x-y motions, the north-east motions respectively, for the two tugger wire configurations. As can be seen in the leftmost, vertically stacked plots, which shows the x-motion and the corresponding FFT of the payload pendulum motion, the motion component does not seem to be damped that much in this direction. Actually, according to the FFT, the frequency peak area of this motion is slightly increased. Nevertheless, the rightmost, vertically stacked plots show the payload's pendulum motion in the y-direction, which is the same direction as the tugger winched are installed. As the results show, the y-motions in the case with tugger wires are significantly damped and get an offset of about 5 m due to the tugger tensions, dragging the payload towards the tugger winches. The FFT of the y-motion clearly shows that this pendulum motion is significantly damped in comparison to the unrestricted payload case.

Predictor

Figure 3.13 shows a screen shot from comparing the two different tugger configurations when using a vessel model instead of logged data as input to the payload model(s). Note that in this case the waves do not hit the vessel model dead on the bow, but comes slightly in from the port side. The corresponding payload motions are shown in figure 3.14, where the upper-most plots shows the x-position of both the crane tip and the payload, the plots in the middle shows the corresponding y-positions while the z-positions are show in the last plots in Figure 3.14a and 3.14b. As shown when comparing the Figure 3.14a and 3.14b, both the x and the y motion of the payload are significantly reduced. In comparison to the previous case, now the forces from both the crane

[5]We only have logged data for the roll, pitch and yaw motions from Olympic Challenger which are provided by the playback module in the virtual observer case.

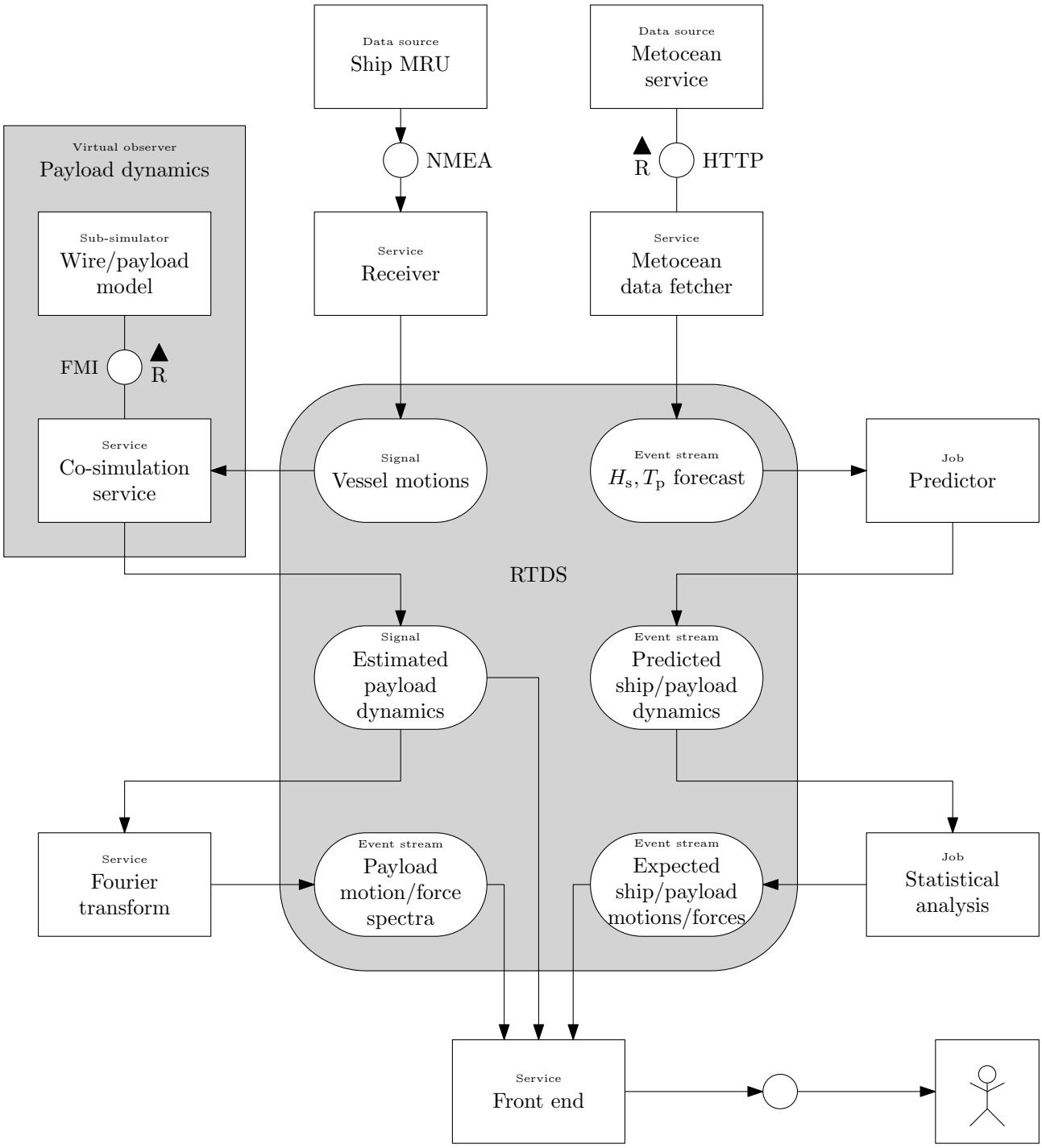


Figure 3.9: FMC block diagram for the tugger configuration MDSS.

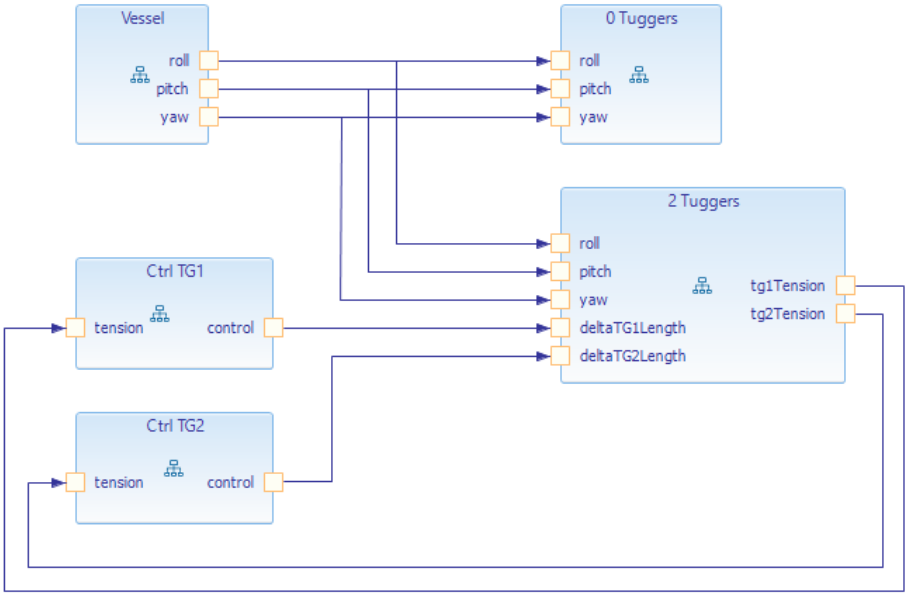
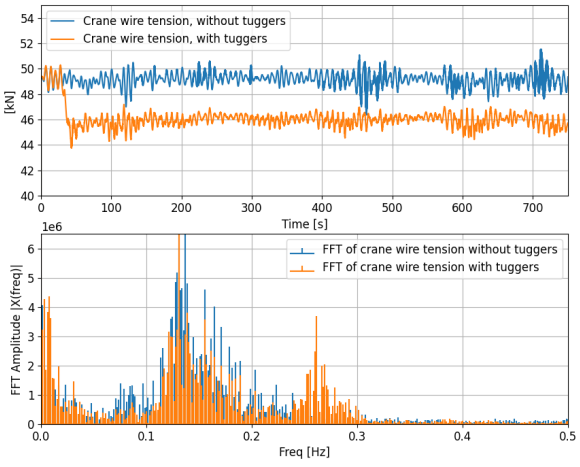
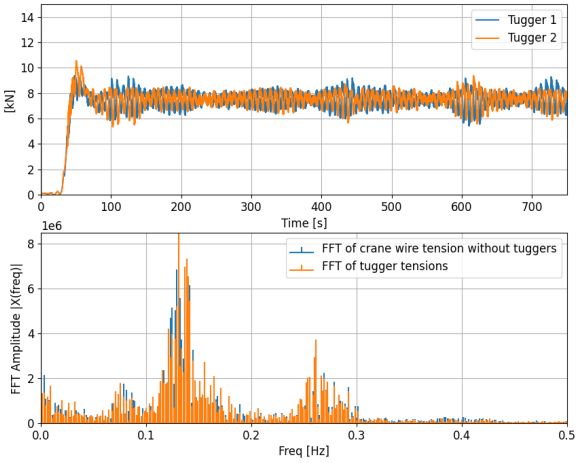


Figure 3.10: Testing two different tugger configurations as virtual observers being fed with vessel motions (first case). The figure shows connections between simulation models and sensor data provided as input.



(a) Tension in crane wire and corresponding FFT. The case with tugger wires adds a new peak frequency area around 0.25 Hz, which should be taken into account.



(b) Tension in tugger wires, and corresponding FFTs. The tugger winch controllers seem to be able to control the tension to around the desired tension set-point of 7500 kN.

Figure 3.11: Tensions in crane wire and in tugger wires.

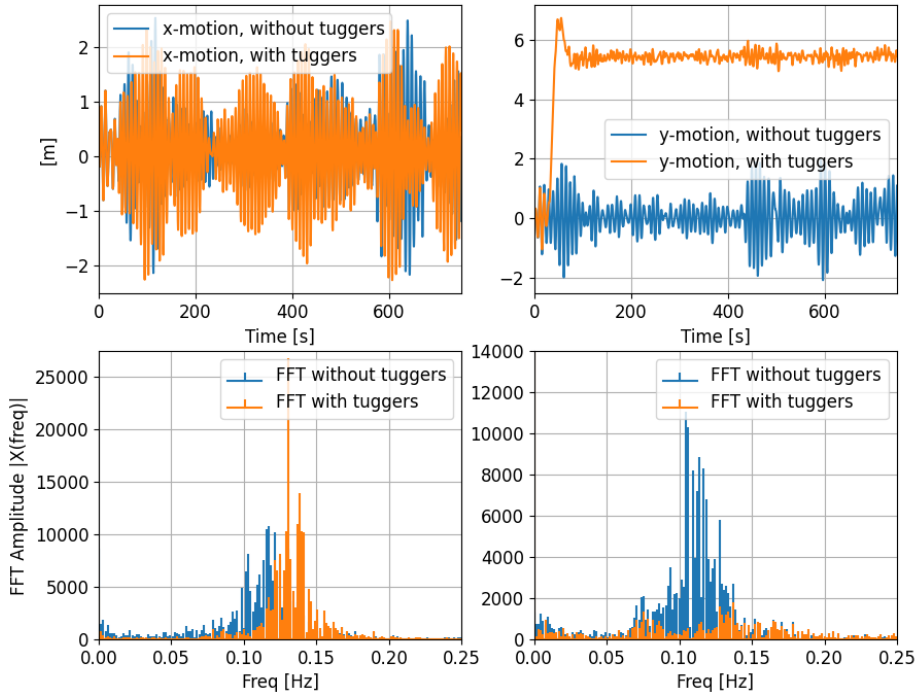


Figure 3.12: Payload position and FFT. Note that the tugger wires restricts the payload motion in the y-direction.

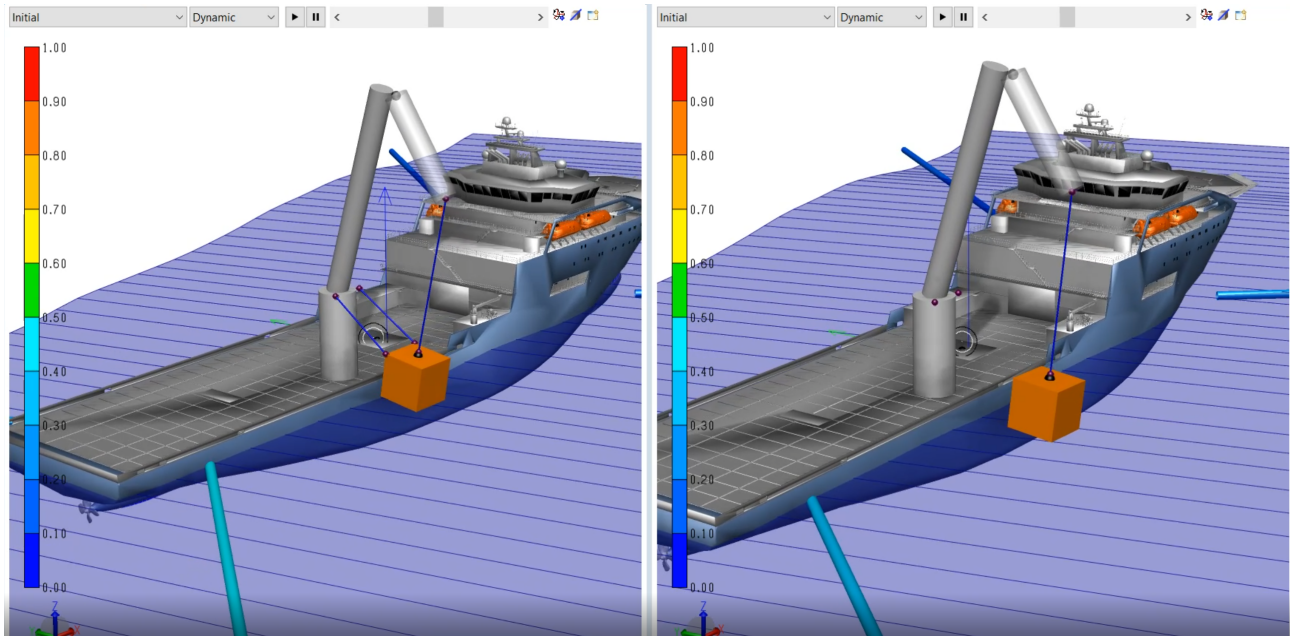
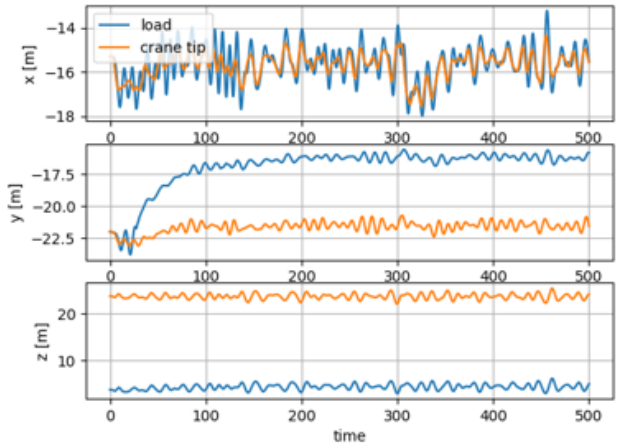
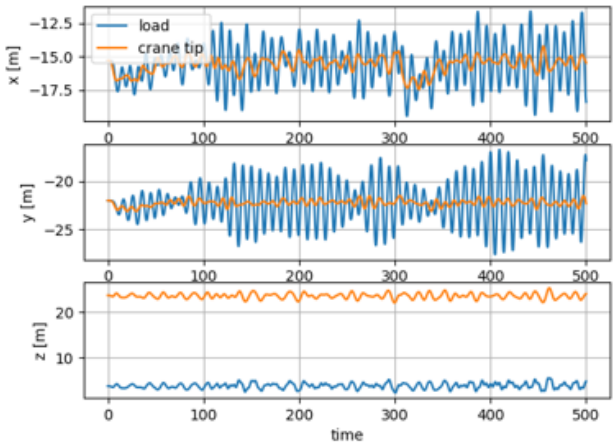


Figure 3.13: Replacing vessel measurement with a simulation model of the vessel (Olympic Challenger). The tugger configuration using two tuggers connected to the crane pedestal is shown to the left in the figure while the configuration without tugger wires is shown to the right.



(a) Payload and crane tip motions with two tugger wires, in closed-loop connection with a vessel model.



(b) Payload and crane tip motions without tugger wires, in closed-loop connection with a vessel model.

Figure 3.14: Simulation setup for two different predictors, one without any tugger wires and one with two tugger wires.

wire and the tugger wires are given back to the vessel model, hence, the payload motions affects the vessel motion as well.

Figure 3.15 compares the tension in the crane wires and the tugger wire tensions for the two different tugger configurations. As before, the tension in the crane wire is slightly lower than in the case without any tuggers since the tuggers take some of the weight of the payload. The amplitude for the crane wire tension oscillations seems to be lower in the case with tugger wires in comparison to the case without. The last plot in the figure shows the tension in the two tugger wires. As can be seen, the two tugger winches seem to have a suited control, being able to keep the tugger tensions of close to the set-point of 7500 N.

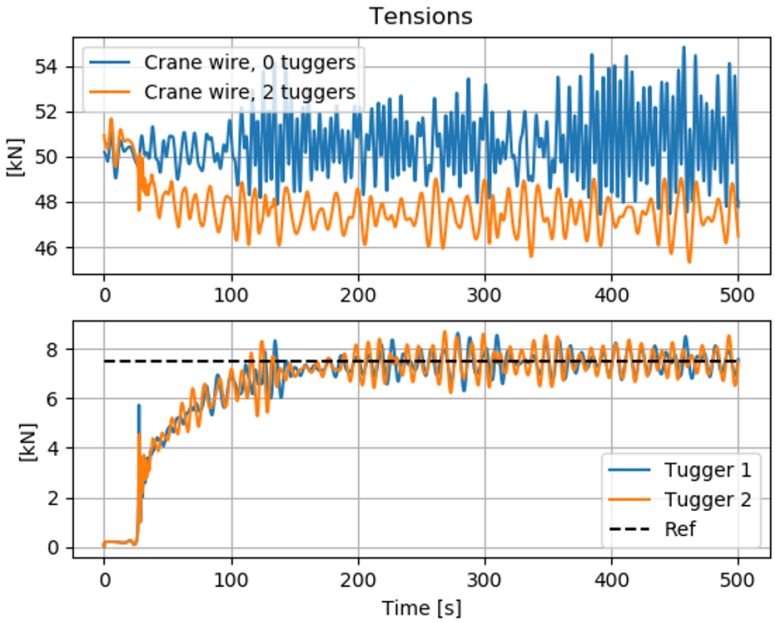


Figure 3.15: Tugger wire tension.

3.3 Light lifting operation, load in splash zone – predictor

When performing an offshore lifting operation through the splash zone, it is important to be aware of the environmental forces, as well as the forces generated by the difference in motion between the lifted payload and the vessel. Changing the heading of the vessel can have a significant impact on these quantities since the waves then encounter the vessel from another angle. This is demonstrated here, where we use a predictor for estimating extreme values in a lifting operation. The payload, a cylindrical suction anchor, is situated in the splash zone, connected to the vessel's crane through a wire and four slings, and motion-stabilised by two tugger wires, as shown in Figure 3.16. The predictor is used to provide decision support prior to the lifting operation to increase both safety and operational efficiency. In particular, in this case the tension in one of the slings is extracted from the simulations and studied. In such an operation it is not desirable that the tension in each sling gets too low, nor too high. Low sling tensions can cause slack slings, which again can cause snap loads; high, dynamic tension peaks.

3.3.1 Method

To predict interesting states in the operation, and the effect of changing the vessel heading, a suitable model representation is needed; a digital twin. This model must be able to represent the vessel, at least its main characteristics, in the given operation. Note that one can use logged data, either "live" or historical, to tune such models, see Section 3.4. In this case study, we use a SIMO [12] model of Olympic Challenger performing a lifting operation with a suction anchor. This model is exported from SIMO as an FMU. The MDSS loads the model and initialises it with relevant user inputs, such as the current weather condition, or the forecast weather conditions, and the vessel heading. The main idea here is to produce statistics through multiple simulations with different realisations of the weather conditions, as well as changing the vessel heading to purposefully find the best vessel orientation for the operation. In this case, 24 different vessel headings are tested, 0-360 ° with increments of 15 °, and for every heading ten different realisations of the wave conditions (the same ten are used for each heading change) are run for producing statistics, resulting in 240 different simulations.

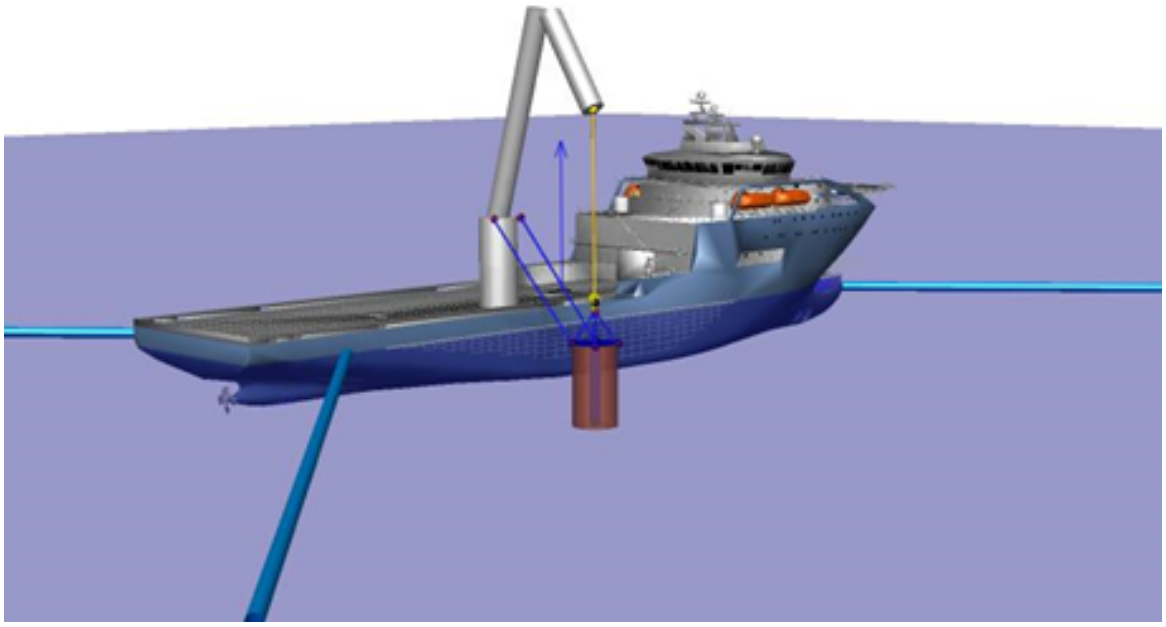


Figure 3.16: System setup for Olympic Challenger performing a lifting operation where a suction anchor is to be installed. Note that the turquoise bars in the figure illustrates the simplified DP-system used, consisting of sets of springs and dampers for the three relevant degrees of freedom.

3.3.2 Components

The components in the MDSS for this case are shown in Figure 3.17. Here, a predictor job is used, which based on an event stream of data executes batch simulations to run multiple "what-if" analyses by sweeping the vessel heading set-point, generating statistics for the operation using different seed numbers for the waves, resulting in different realisations of the waves. A *Metocean data fetcher service* obtains relevant weather forecasts from a *Metocean data provider service*, and feeds the predictor with useful predictions about the near future environmental conditions. The simulation results obtained in the predictor are fed to a statistical analysis job through the RTDS as an event stream. This statistical analysis job analyses the simulation results from the predictor and forwards a new event stream of data to the front-end, which is to be presented as useful decision support for the end-user.

3.3.3 Results

The simulation results for the tension in one of the payload slings, which are given in kilo Newton, for ten different realisations of the waves[6], for a heading of 0 °, are shown in Figure 3.18.

Based on the time series generated by the simulations, the extreme values (both from lower and upper regions) are extracted from all ten weather realisations. It is out of scope here to detail specific methods for extracting these values, but there exist multiple strategies, e.g. extracting only the most extreme values (both upper and lower) from the time series, or to extract the most extreme values from a moving time-window in the time series.

The extracted extreme values from the time series are used for curve-fitting extreme distributions such as the Gumbel distribution or the Weibull/Fréchet distribution. Figure 3.19 shows some of the results from the curve fitting using the Gumbel distribution. Note that the two left-most columns of plots in the figure shows

[6]The realisations are produced by changing the seed number for the phase shift random generator of wave components.

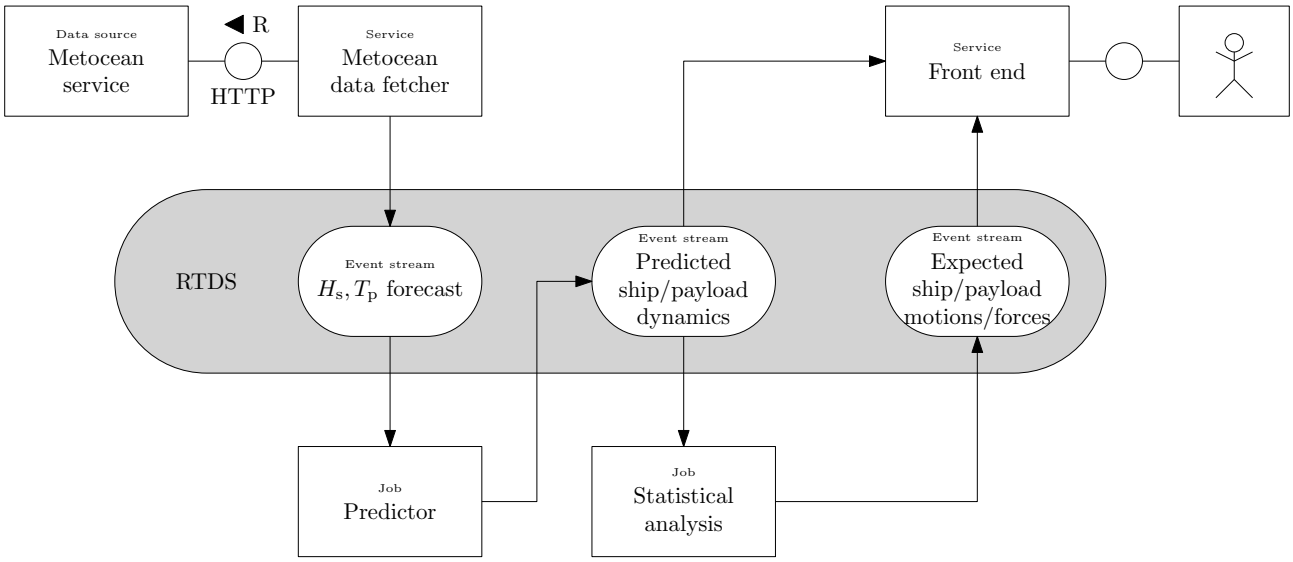


Figure 3.17: FMC block diagram for the "load in splash zone" MDSS.

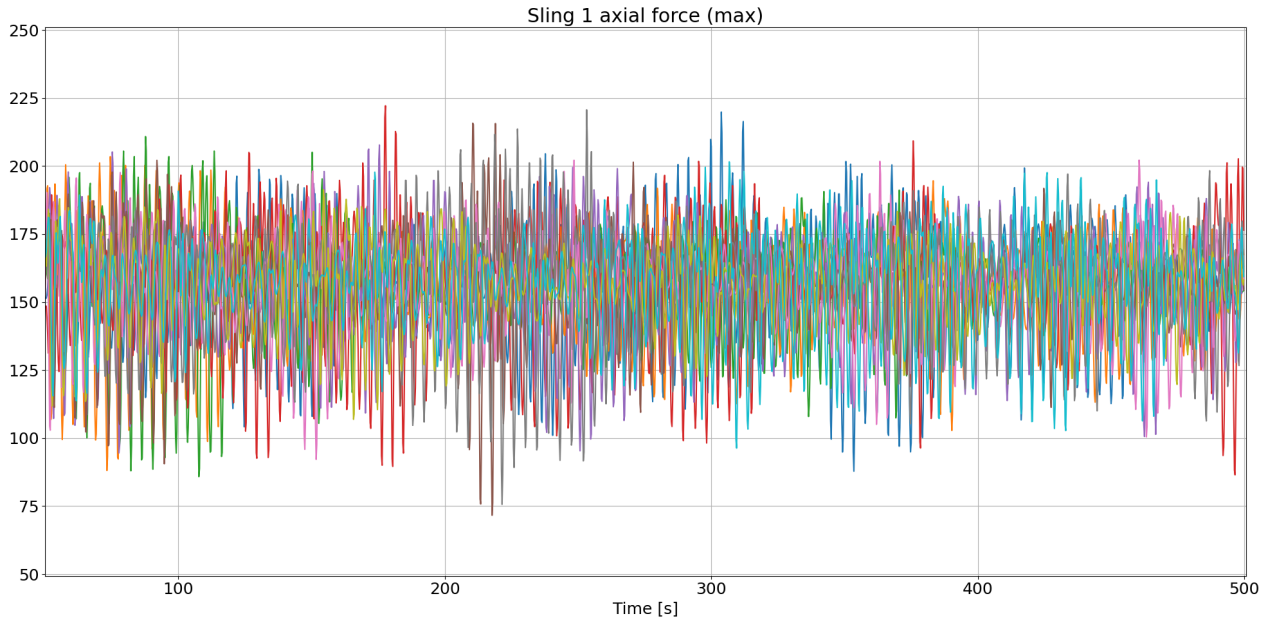


Figure 3.18: Running multiple realisations of the environmental conditions help to map the expected extremes of the studied system state(s). This figure shows the tension in one of the payload slings for tin different realisations of the environmental conditions. The unit on the y-axis is kilo Newton.

the probability density function for the Gumbel distribution for the minimal extreme values followed by the corresponding cumulative distribution, while the two right-most columns shows the corresponding plots for the maximal extreme values. From the extreme value distribution it is possible to extract the expected extreme

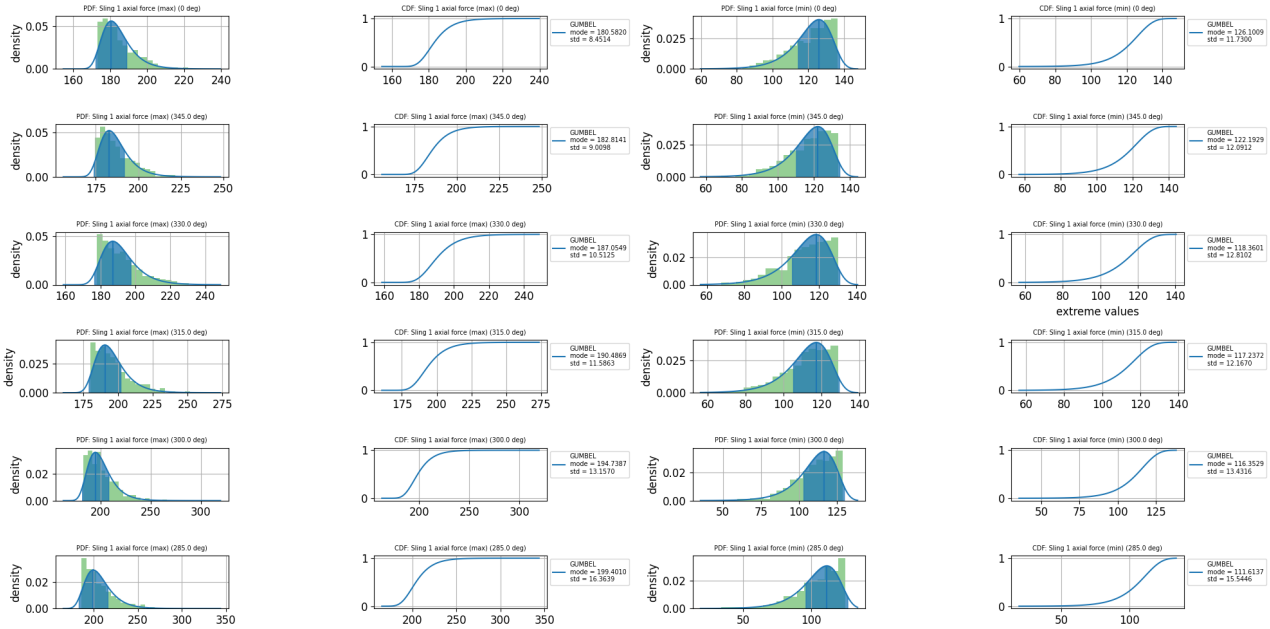


Figure 3.19: Running multiple realisations for the environmental conditions can help produce extreme value distributions of interesting quantities. Here, running to different realisations for the environmental conditions, for six different wave directions relative to the vessel, help mapping the lower- and upper extreme values for the expected sling tension. Note that also a lower extreme is added to be able to determine if the motions of the payload cause the sling to be slack during the operation, which is not desired and can cause critical events.

values and the corresponding standard deviations. These values can be used to determine expected maximum and minimum values in tension for the payload sling for each given vessel heading. Graphically, these results can be visualised in a polar plot, as shown in Figure 3.20. Note that the heading in the figure, the angular axis, shows the heading relative to the incoming waves, where the waves are assumed to come from North in the simulations. As an example, at 0 ° in the figure, the waves hits the vessel on the bow, while at 90 ° the waves hits the vessel at the port side (dead-on).

For this particular case, visual inspection of Figure 3.20 indicates that orienting the vessel such that the waves hits the vessel at 22.5 °, at the port side, gives the lowest maximal tensions and the highest minimal tensions in the sling, which is the best operation condition.

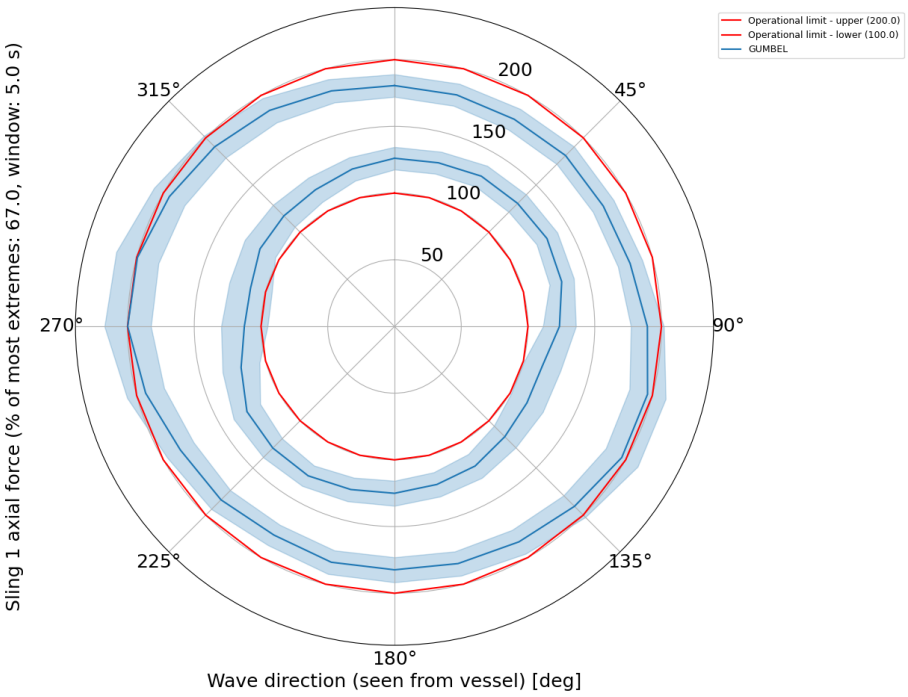


Figure 3.20: Running different realisations of the environmental conditions and different orientations of the vessel relative to the incoming waves help produce a polar plot showing the extreme maxima and extreme minima for the sling tension. Note that the red lines are the limits while the blue line are the max/min expected extremes, while the light blue filling is the standard deviation.

3.4 Waiting for lifting operation – model parameter tuning

The understanding of a system's characteristics and operation using simulations is highly dependent on the simulation components being able to represent the system characteristics accurately enough for its purpose. When planning demanding offshore campaigns simulations are often used, which are tuned based on both known system characteristics and experience, and on qualified guesswork and historical data regarding environmental conditions and the state of the operating equipment. Hence, an operation is conservatively planned when it comes to safety and time frame needed for executing the mission. This is done in order to compensate for the many uncertain quantities which are possibly only known right before the operation is started.

One always want to increase the operational window at the same time as maintaining operational safety, both for equipment, payload and personnel. It is also important to use sufficiently accurate mathematical models for its purpose in an MDSS to provide the appropriate decision support. This can be achieved by using logged data, both live and historical, for tuning the mathematical models. Moreover, since some models have parameter sets that are only valid for one specific operational condition, such as for example a vessel model, it is also of interest to store tuned parameter sets along with the actual vessel state, operation, and the environmental condition. This both enables for warm-starting future tuning procedures and facilitates building knowledge databases on shore. Knowledge databases can be used in planning future operations, as well as for making improved system designs.

In this case study, the Olympic Challenger model from SIMO will be tuned based on previously generated data. We assume that some specific parameters are set with $\pm 10\,\%$ offset in comparison to the actual values in an on-shore analysis. Then, the goal is to be able to tune the vessel model based on previously generated data with the correct parameter set for the vessel model. This is done to demonstrate the concept. Nevertheless, the methods employed in this case study have been tested with realistic vessel data as well, as stated in [46].

3.4.1 Methods

A general schematic representation of an model tuning algorithm, which is closely related to a traditional observer/estimator design, is shown in Figure 3.21. As shown in the figure, comparing live data to simulated data gives a measure for how well the model represents the real system in its current situation and condition. Nevertheless, it is difficult to directly compare time series from physical sensors with time series generated in simulations due to stochastic variations in the environmental conditions. If this is the goal, the simulation model needs to experience the exact same environmental conditions as the realistic system. This is hard to realise in practice, so we seek to compare statistical data instead. It is out of scope here to go into details, but a thorough description is given in [46]. Comparing statistics are done by producing different spectra and comparing their spectral values.

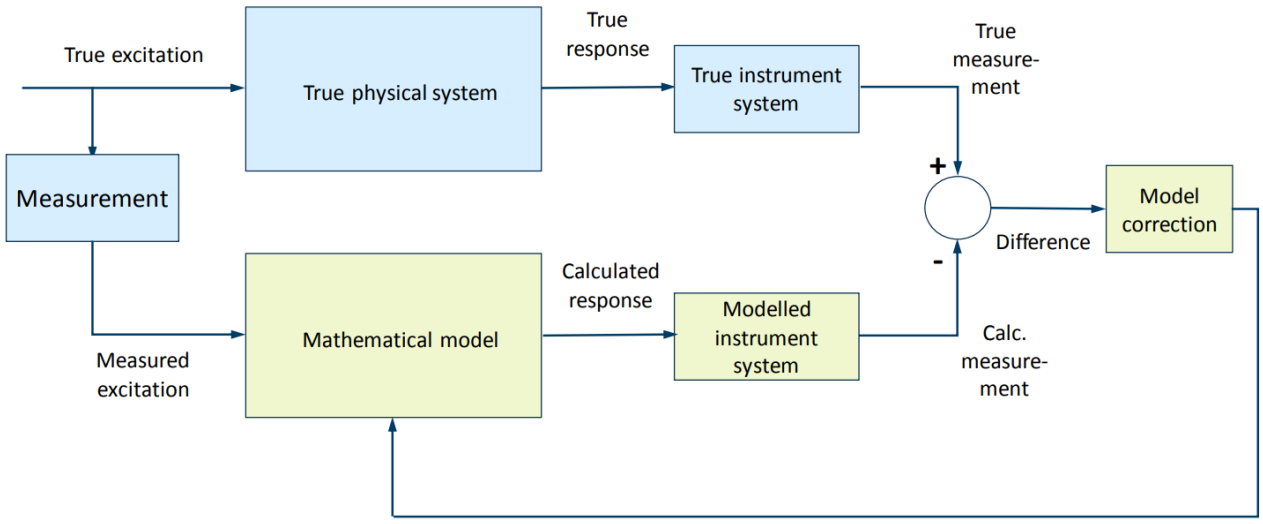


Figure 3.21: Illustration of principle of dynamic system state estimation. Blocks in light blue represent the real world. Blocks in light green represent the mathematical model.

3.4.2 Components

The different components needed in the MDSS for tuning the vessel model are shown in Figure 3.22. In this case, the parameter tuning job writes new simulation configuration files, triggers new simulations with new configurations, and reads the results files. The inner dashed box in the figure contains the simulation part of the parameter tuning with simulation configuration files and results files. The outer dashed box denotes the entire parameter tuning service which is run by a job scheduler. The freshly tuned model parameters are fed to other simulation services through the RTDS, after being thoroughly tested and validated. The vessel model uses information from the *Metocean data source*, and possibly a wave radar, to set the current environmental conditions in the simulations. In parallel, the measured ship states are logged and stored to be used for comparison when calculating the new vessel parameters in scope.

3.4.3 Results

In this case study we are tuning the linear roll damping of the model, d_p, the restoring moment in roll, k_p and the z-component in the centre of gravity. As initial values, the parameter set used for the on-shore analysis will be used. Figure 3.23 shows the normalised values for these variables as function of the iteration number in the tuning procedure. As can be seen in the figure, the three different vessel parameters to be tuned get

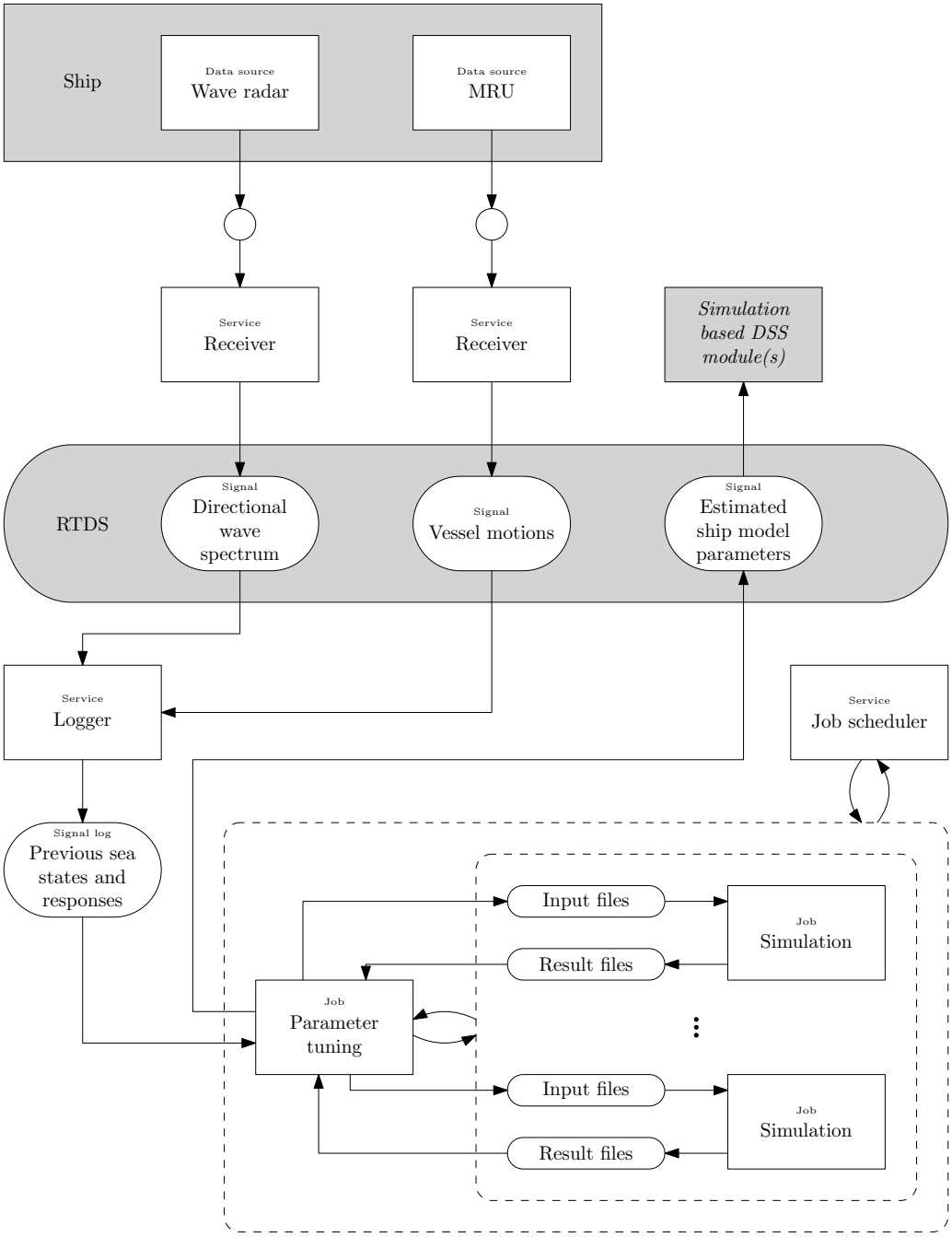


Figure 3.22: FMC block diagram of the ship model parameter tuning system. The grey box marked “Simulation based DSS module(s)” represents any number and type of DSS modules that use a dynamical ship model, and therefore can benefit from up-to-date model coefficients.

close to 1 — in the figure after nine iterations. Note that the values in the figure are normalised, but the actual values and the results are shown in Table 3.1.

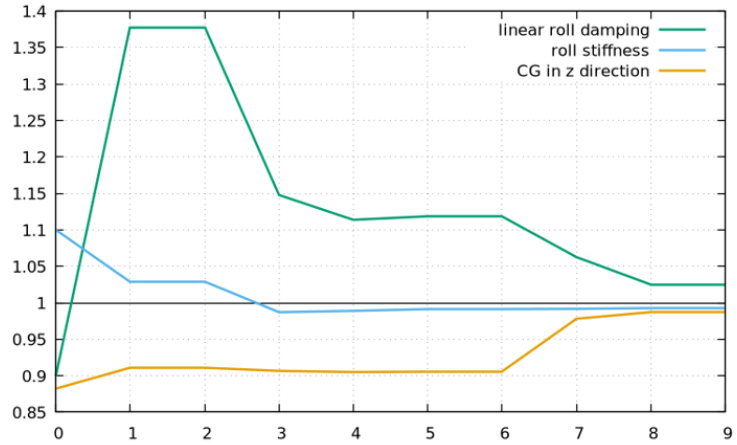


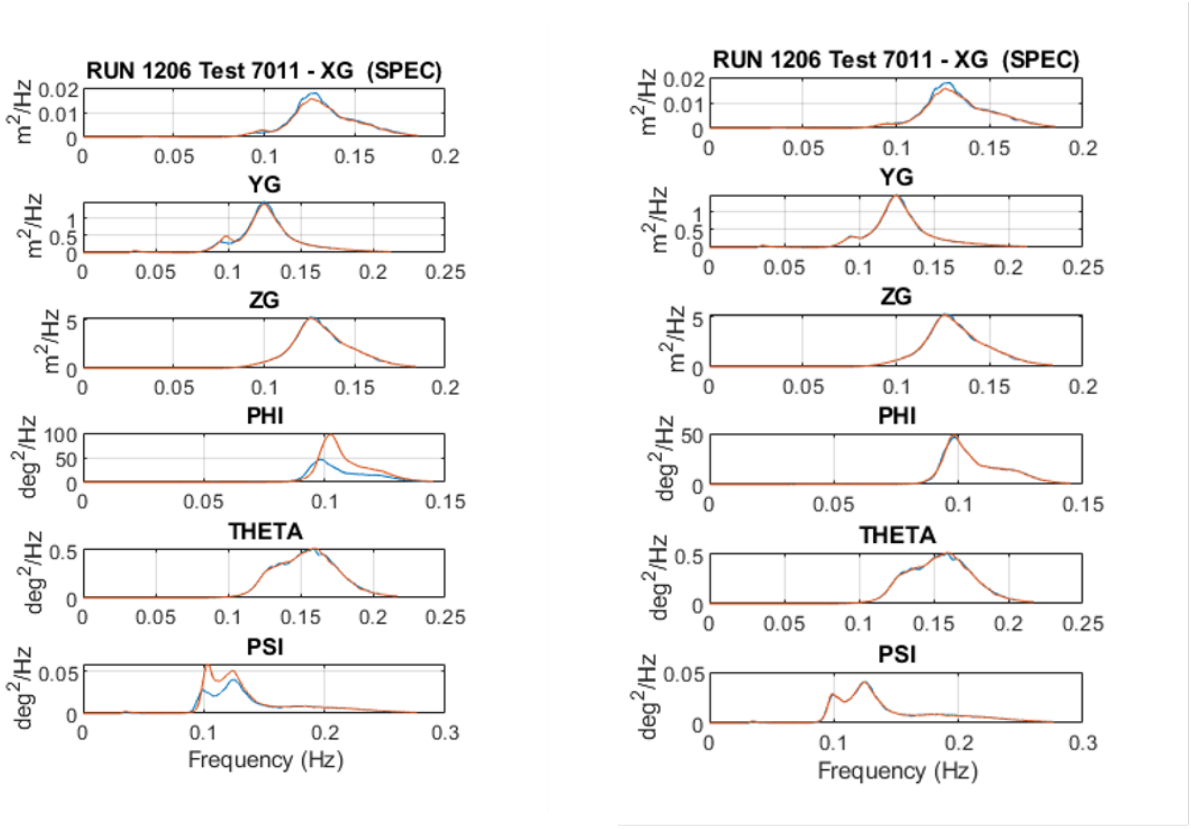
Figure 3.23: Iterations in the tuning algorithm. Note that the results are normalised on the y-axis, hence, 1 is the desired value.

Table 3.1: Numerical results from model tuning compared to actual values and the initial condition.

Parameter set	**Linear roll damping**	**Roll stiffness**	**zcog**
Actual	40290	256137	1.7
On-shore analysis	36 26261	270751	1.5
Tuned	41314	244430	1.68

To understand the effect of the model tuning, simulation results using the initial parameter set and the tuned parameter set are used to generate spectra for the degrees of freedom for the vessel model. These spectra are compared in Figure 3.24. From the figure we can see that the **YG**, **PHI** and **PSI** seem to be significantly improved.

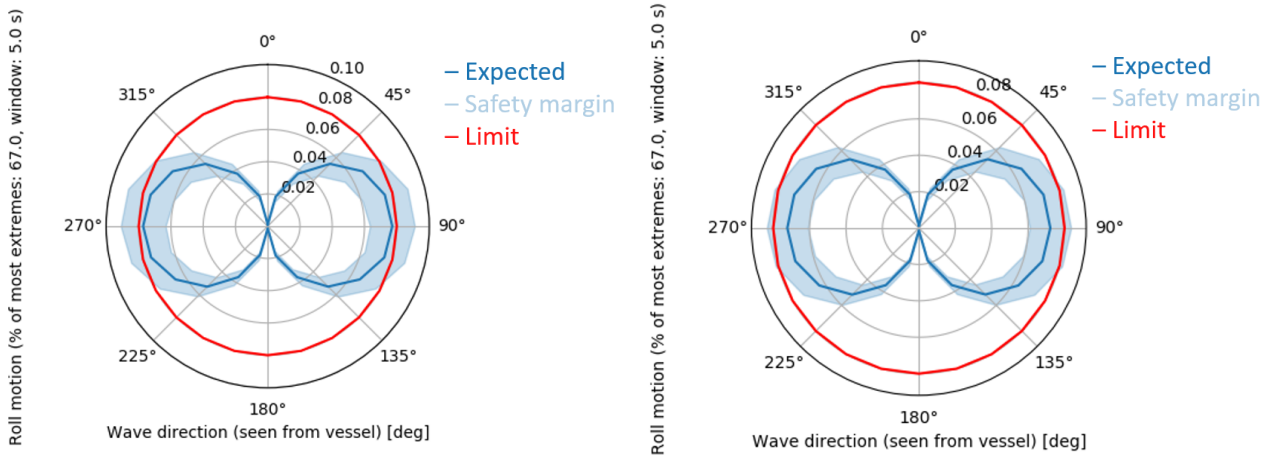
To demonstrate the effect of using a more accurate parameter set for decision support applications, let us assume that we run a similar analysis as the one presented in Section 3.3, but now we are interested in the expected roll motion for the vessel as function of the wave direction encountering the vessel. Assuming that we do not want a roll angle amplitude of more than 4 ° (≈ 0.07 rad), we can make a comparison when using the on-shore parameter set and the tuned parameter set. Such a comparison is shown in Figure 3.25. As shown in Figure 3.25a, the capability plot for the roll motions using the on-shore parameter set shows that the expected extremes are within the preset safety limit, but the standard deviations, the safety margins are not. The situation is not as severe for the analysis using the tuned parameter set, as seen in Figure 3.25b. Note that for this comparison, the MDSS setup with components and corresponding connections as sketched in Figure 3.17 is needed.



(a) Spectra for the six degrees of freedom for the vessel model generated from simulation results using the on-shore parameter set.

(b) Spectra for the six degrees of freedom for the vessel model generated from simulation results using the tuned parameter set.

Figure 3.24: Comparing vessel motion spectra for the parameter set used on-shore with the tuned parameter set.



(a) On-shore parameter set

(b) Tuned parameter set

Figure 3.25: Expected roll motions and corresponding standard deviation (safety margin) compared to the limiting criterion for a given operation, before and after model tuning.

3.5 Energy-efficient operation of hybrid propulsion systems

The decision support system described in this section was developed in the *PurSense* project[7][47]. For a more detailed description than what we give here, see [48].

In this case an on-board decision support system for four purse seiners was developed. On modern vessels with hybrid propulsion systems installed, the fuel consumption can vary greatly depending on the selected operational mode of the machinery system. Typically, the vessel has a main engine that can produce electrical power—by driving a generator or shaft generator—and/or transfer power mechanically to the propulsion system. A number of auxiliary engines connected to generators—gensets—could also deliver electrical power to the system. The propulsion system can thus be powered mechanically or electrically or even a combination of mechanical power from the main engine with boost from electrical power from gensets. A number of operational modes will be available, these modes will differ in which power produces are running—main engine, gensets—, how the propulsion power is delivered and which power producers that deliver electrical power to the remaining electrical consumers. Making the most energy efficient choices of operational modes on such vessels can be difficult for several reasons:

- The vessels have complicated operational patterns in which changes between very different operations may occur frequently, without being planned before the last minute.
- Operational demands may be different for similar operations due to e.g. how much fish is stored in the fish holds, weather conditions or shoal behaviour.
- The fishing operations are the main concern for the crew, and they may therefore have less attention towards choosing the optimal operational mode to lower fuel consumption.

3.5.1 Methods

The key concept for this case is to maintain a database of the *best* way the vessel has fulfilled similar operational demands historically. The operational demands were the *speed*, *thrust* and *electrical consumption* of the vessel, while the indicator of better performance will be lower fuel consumption. The *thrust* signal was replaced by a *normalised thrust* in order to make it independent of the *speed*. Similarly, the fuel consumption was normalised to remove the direct dependence on speed and electrical consumption. By dividing the continuous operational demands into finite bins (intervals), the *best-so-far* solutions for each bin can be stored. The current performance can then be compared to the stored *best-so-far* solution and the user can be presented with the alternative (better performing) solution. The presented solution would then comprise the alternative fuel consumption and the operational choices for which it was achieved. Here, operational choices are the mode the hybrid propulsion was run in, engine rpms and propeller pitch and rpm. Importantly, update and suggestion of best-so-for solution and evaluation of performance are only carried out when the vessel demands are in a steady state.

3.5.2 Components

Figure 3.26 shows a schematic view of the different logical components and the data flow in the system. Several general-purpose components as described in Section 2.4 were used in the system: *Protocol converters*, *Logger* and *Mathematical functions*. The *Advisor* component carries out the main work in evaluating the current performance and comparing with historic *best-so-far* solutions for similar demands. The *best-so-far* solution and its fuel consumption are transmitted back to the RTDS. The *Database updater* will update the *best-so-far database* if the current solution outperforms historic solutions. It could be noted that the *Advisor* and *Database*

[7] *PurSense: Operation monitoring and decision support for purse seiners* (2013–2017), funded by the Research Council of Norway (grant no. 226378), the Norwegian Seafood Research Fund (grant no. 900886), Ervik & Sævik AS, Eros AS, Herøyhav AS, and Kings Bay AS.

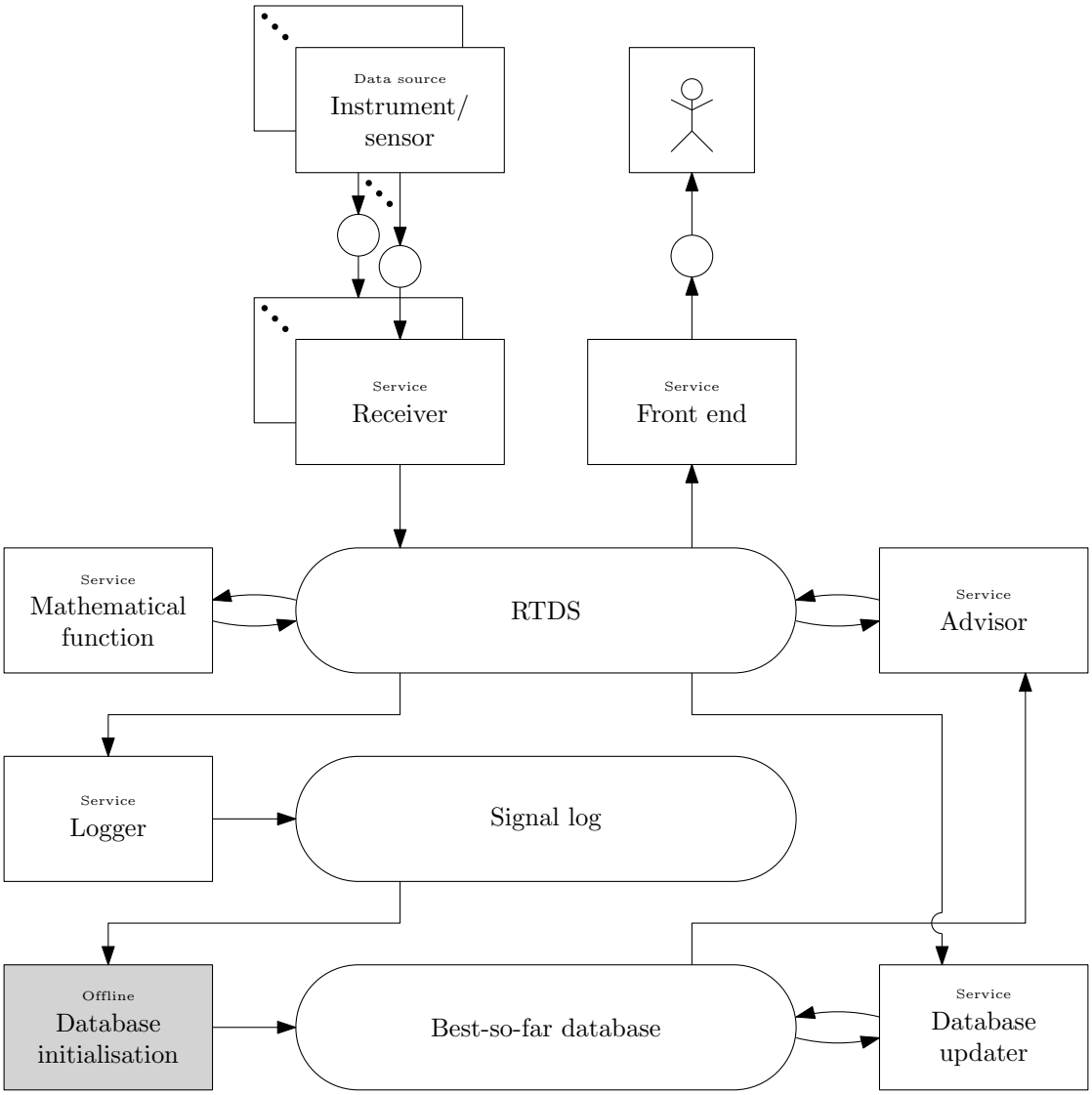


Figure 3.26: FMC block diagram for the PurSense system. The RTDS in this case carries a large number of signals. To keep the diagram simple, the individual channels are not shown.

updater services include some filtering of the input signals that could alternatively have been carried out in a separate general-purpose *Filter* component. The *Mathematical function* service is here used to calculate input signals to the *Advisor* and *Database updater* services. The calculation of input signals has two purposes: one is to harmonise the signal input between different vessels, which may have different configurations of the power systems and also different available signals from the vessel systems, the second is to estimate values that are not directly available based on the values of other signals (one example for the included vessels is that there is no direct measurement of the propeller thrust). Figure 3.26 also includes an offline component, *Database initialisation*. This process is not running while the MDSS system is running, but is used to create an initial *best-so-far database* from signal logs that with historic operational data from the vessel. This means the *best-so-far database* can also be updated if any changes to the configuration of the system is made, e.g. updates to the configuration of the *Mathematical function* or the *Advisor* services.

3.5.3 Results

An example screen shot from the user application is included in Figure 3.27. The user is here shown a plot of the best-so-far fuel consumption for different vessel speeds with the current electrical demand, different curves show the results for different operational modes. By comparing the curves for the different operational modes, the user—with his or hers knowledge of the imminent operations—can more easily select the best operational mode going forward. For more results, see the paper by Reite, Ladstein and Haugen [48].

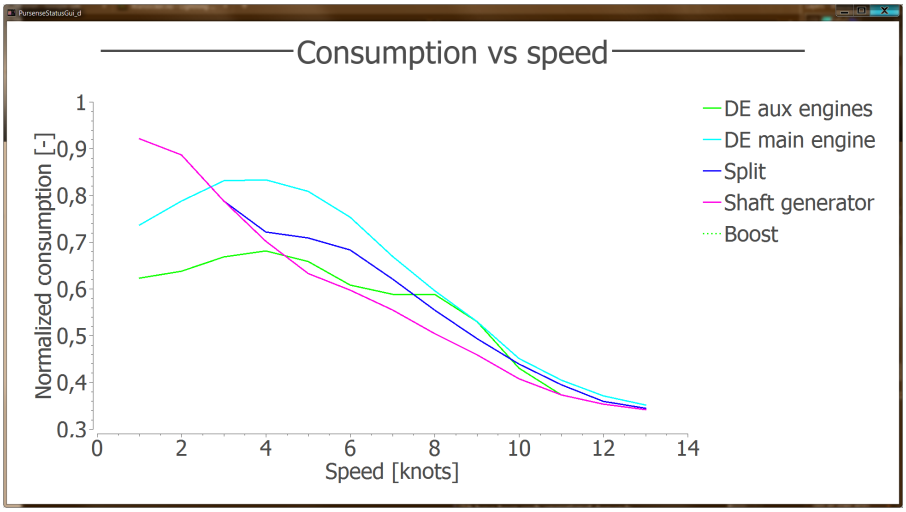


Figure 3.27: A screenshot from the *PurSense* user application showing normalised consumption with varying speed for the current electrical consumption [47].

3.6 Monitoring of towed seismic operations

This section describes results from the *Seismic RTDT* project[8], with even more details found in Ref. [49]. A seismic front end, as seen in Figure 3.28, is the rope-spread arrangement in which to the streamer cables are attached. In general, a complete seismic cable spread can be as large as 1.8 km wide and 10 km long, consisting of up to 20 streamer cables, being the largest man-made moving object on the earth to this date. The main purpose of the front end is to spread out the streamer cables, which has the sole purpose of measuring sound waves reflected from the sea floor. The sound waves are generated by gun-arrays using pressurised air, to cover

[8] *Real Time Digital Twin for Boosting Performance of Seismic Operations* ("Seismic RTDT", 2018–2020), funded by Kongsberg Maritime CM AS (formerly Rolls-Royce Marine AS) and the Research Council of Norway (grant no. 282378).

a large area behind the towing vessel. The front end itself is divided into two mirrored arrangements, that may or may not be connected. Each of the sides of the front end consists of a deflector, a wide-tow, connecting the deflector to the vessel, lead-in cables, spread-ropes, head buoys and a spur-line, as shown in Figure 3.28. The deflector is responsible for spreading out the streamer cables, whereas the rope arrangement's purpose is to mainly balance the forces between the vessel, the deflectors and the streamer cables.

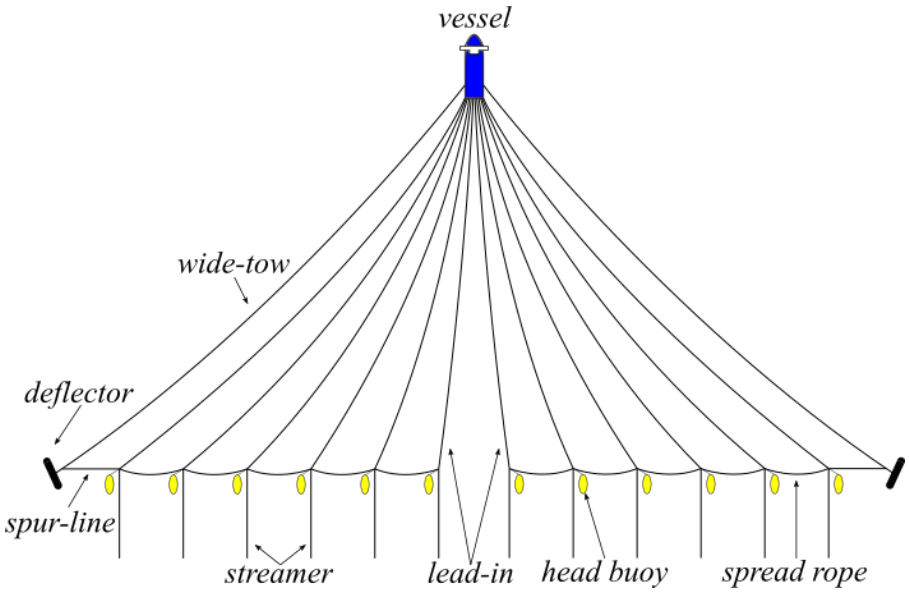


Figure 3.28: Sketch of a seismic front end with named gear. Figure obtained from Ref. [49].

One challenging task when conducting an offshore seismic survey is to balance the forces in the seismic spread at the same time as keeping the geometry of the spread — as widely and evenly spread as required for making accurate enough measurements, and its success is highly influenced by experienced crew. Nevertheless, an MDSS consisting of a front end observer and real-time measurements can help the crew to balance forces and better understand the geometry of the front end in real-time.

3.6.1 Methods

As in many of the previously presented case studies, the various cables and ropes in the front end are modelled as lumped bars connected by constraints, which are solved explicitly using Baumgarte stabilisation [45]. It is assumed that each streamer cable has a force-cell, such that the forces can be measured and used as input to the front-end model. Moreover, the head buoys, the deflectors, and the floater for the gun array are assumed to have *relative global positioning system* (RGPS), such that their positions, relative to the towing vessel, can be used as input to the front end model. These positions are used in position constraints, and even though the gun array tie the port side and starboard side of the front end together, it is possible to separate them because of the position constraint for the gun-array, since there are two different cables connecting the floater to the different front end sides.

3.6.2 Components

The components in the MDSS for the virtual observer for the seismic front-end are sketched in Figure 3.29. In this setup the vessel states, as well as streamer load cell information, head-buoy-, deflector and gun-array positions, and some environmental conditions, such as currents, are measured and logged in real-time. These measurements are fed to the co-simulator which runs observers for the starboard and port side front-end spread as sub-simulators in a co-simulation setup. The simulation results are distributed on the RTDS, where

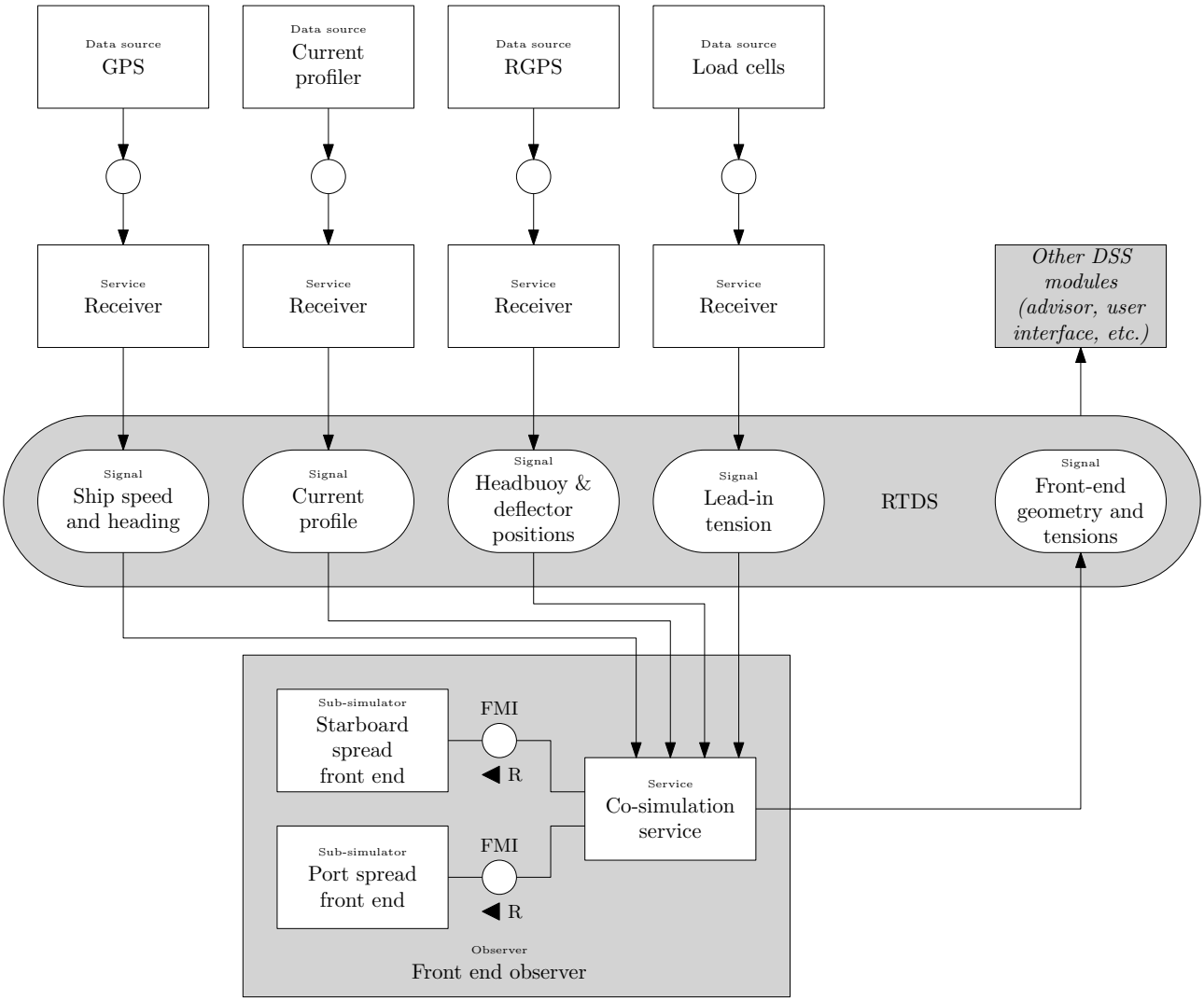


Figure 3.29: FMC block diagram for the seismic front end state observer and its inputs and outputs. "RGPS" stands for "relative global positioning system", and refers to a high-precision positioning system for floating equipment near the vessel (head buoys and deflectors in this case). The grey box labeled "Other DSS module(s)" is a placeholder for any and all modules that use the information produced by the state observer to provide decision support.

different modules and services pick them up and process the data, providing useful insight regarding both the front end geometry and the load distribution amongst the lead-in cables, spur-lines, spread ropes and the wide-tows. This insight can be used both for visualising the cable geometry, as well as for monitoring and surveillance purposes, and load balancing control through adjusting the lead-in winches. Figure 3.30 shows the connections in the front end simulator. Note that the simulator itself is required to run in real-time in this

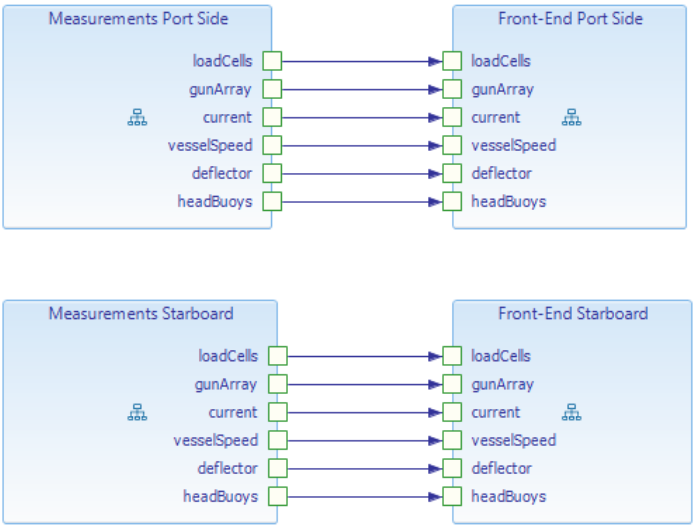


Figure 3.30: Setup for seismic front-end observer.

case.

3.6.3 Results

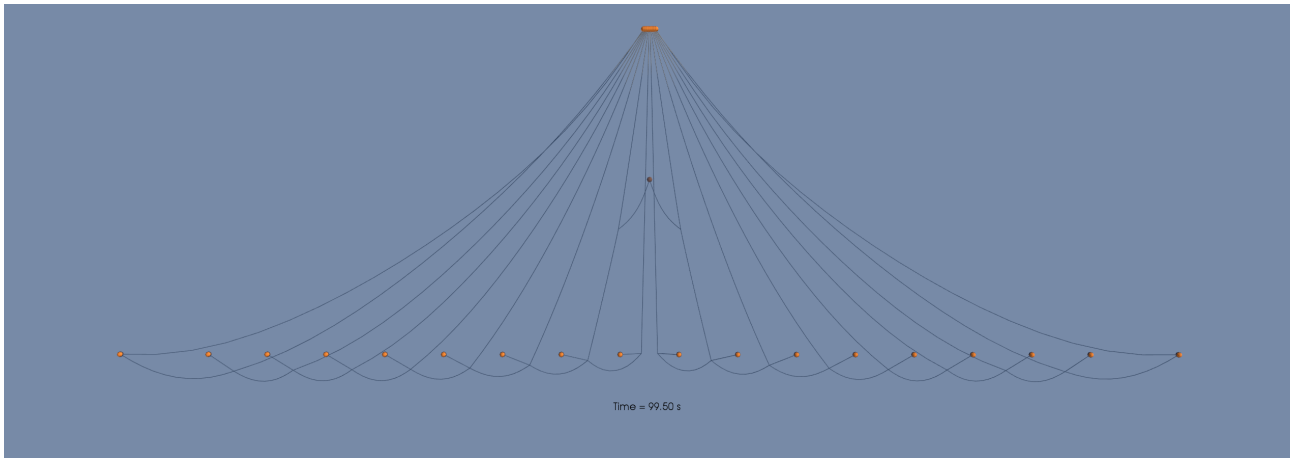


Figure 3.31: Visualisation of the seismic front-end observer, seen from above in a bird-view.

From the MDSS for the seismic front end it is possible to obtain tension forces in all cables, as well as the geometry of the entire spread. It is out of scope here to present specific results and details about the simulated seismic survey, but a 3D representation of the front end in bird view is shown in Figure 3.31. Note that the orange spheres in the visualisation illustrates all the position measurements fed to the observer. Both the spheres, some of them representing the head buoys, and the cable thickness in the figure are made larger than in reality to increase figure readability.

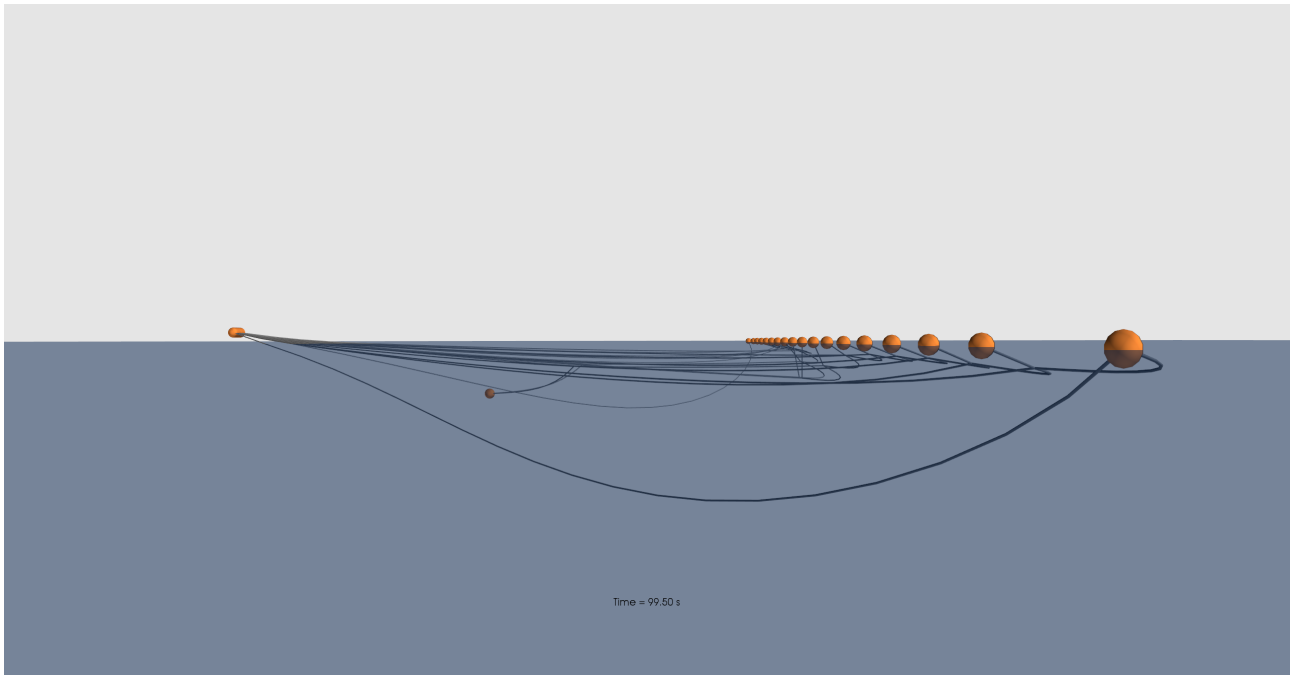


Figure 3.32: Visualisation of the seismic front end observer, seen from the port side towards starboard side.

Figure 3.32 shows the front end in 3D from port side towards the starboard side. Note that the wide-tows in the figure seem to be situated a bit deep and should be reeled in by the wide-tow winches. Also, the first couple of minutes in the simulation the model is converging towards its operational equilibrium from the initial conditions and these screenshots shown in Figures 3.31 and 3.32 are taken before the system has converged.

3.7 Catch control in purse seine fisheries

In this section, we describe a decision support system developed in the project *Catch control in purse seine fisheries*[9]. Haugen and Kyllingstad [50] provides a detailed description of the project work. During the phase before purse seine deployment the purse seine master ("the purser") needs to acquire and maintain a situational awareness. This is a key success factor in purse seining. Elements of the situational awareness include a comprehension of the main process components in play. Mastery of purse seining is often correlated with experience. Regardless of the level of experience, there is a fair amount of burden placed on the purser. For this reason, a useful aid would be a tool that could both ease the burden and help acquire the necessary situational awareness before action is taken.

Modern purse seiners are equipped with sophisticated instruments, which help the captain in executing their job. The wheel house has many monitors with diverse and scattered information, from which some are very important during the pre-deployment phase. A proficient purser possesses both perceptive and predictive abilities. This means that based on the available sensory information about the processes in play, the purser is capable of predicting ahead in time a plausible outcome based on a series of actions. These actions are typically, i) to purposefully manoeuvre the vessel, and ii) to determine an appropriate time point for purse deployment in such a way that the fish is caught as intended. Figure 3.33 displays the main phases in purse seine deployment.

[9]The actual project name is Norwegian: *Fangstkontroll i notfiske etter pelagiske arter: Fase 2.* Funded 2017–2021 by the Norwegian Seafood Research Fund (grant no. 901350), the Institute of Marine Research, and the Norwegian Directorate of Fisheries.

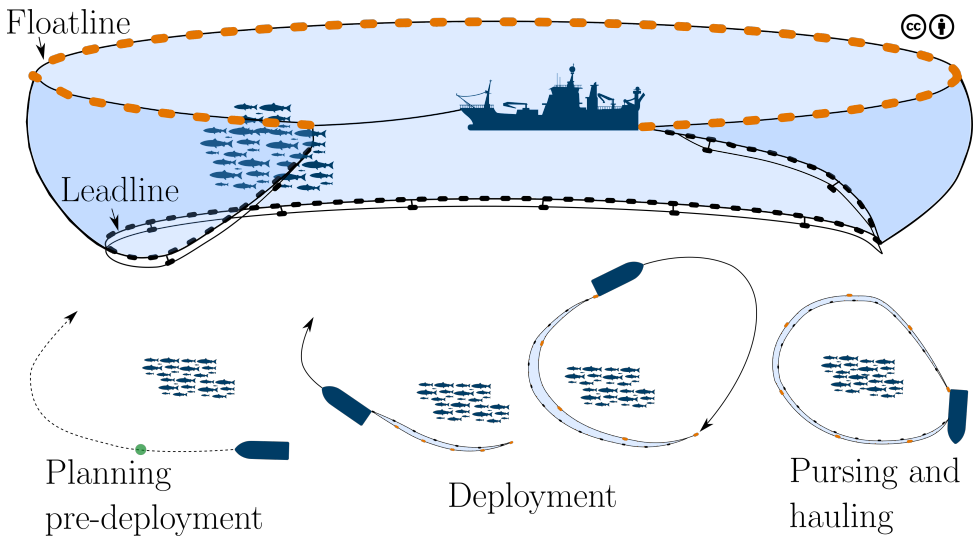


Figure 3.33: Illustration of purse seine elements and main deployment phases. Image credit: Ref. [50].

3.7.1 Methods

The goal is to create an MDSS for purse seine deployment. We pose the proposed solution as an optimal control problem formulation, specifically, as a so-called path planning problem. The solution to such a formulation contains the following:

- A suggested planar path for the fishing vessel to follow
- An indicated point in time and space for initiation of purse deployment
- Auxiliary information about the involved actors, which helps situational awareness

Typically, such solutions should be presented to the purser in an informative manner, for instance as a graphical visualisation including a map plot, which we will provide later. The arrows in Figure 3.34 indicate information flow and interactions between important elements of the presented decision support. Instruments are capable of acquiring relevant environmental data together with vessel-specific measurements in real-time. The system make use of a machine-to-machine application programming interface that are capable of sharing this information, in particular, with the help of appropriate middleware, protocol converters, and other services.

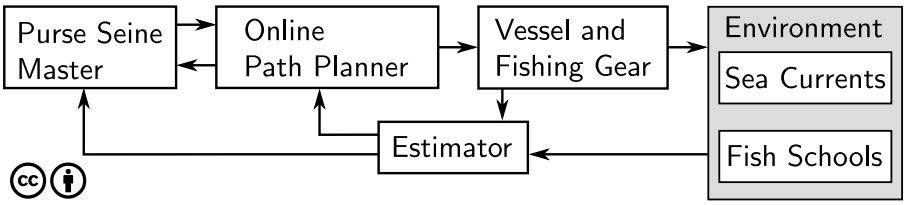


Figure 3.34: Key elements in a purse seine deployment operation. Image credit: Ref. [50].

The online path planner consists of simplified models of key elements of the process, namely:

- Kinematic vessel with manoeuvrability constraints
- Expected fish school movement model
- Anticipated sinking response of the leadline midpoint

- Environmental conditions
- User preferences

These elements are mathematically described as ordinary differential equations with constraints in a *nonlinear programming* (NLP) problem. This formulation also includes various criteria, such as deployment initiation time point, sinking margin, trap the fish, pre-deployment positioning, and more. The NLP problem is re-solved in a receding horizon fashion, meaning that the problem is solved periodically, making use of the newest available data on the RTDS, which include changes in vessel and fish position and orientation, new environmental conditions, and updated user settings. This decision support system is implemented in two parts: the algorithm and the user interface.

During the various phases of the development we made use of several different tools and libraries, for which central items are indicated below:

Data acquisition and data sharing *Ratatosk* [4] and *OpenSplice DDS* [51];

Decision support development Casadi [52] for automatic differentiation and interfaces to NLP-solvers, Ipopt [53] for NLP solver, Qt [54] for graphical user interface implementation;

Case study fmiCpp [13]: simulator models of the vessel and fish school, *cosim* [55]: co-simulation engine.

3.7.2 Components

The decision support consists of several components as indicated in Figure 3.35. There is a series of data sources, namely *Anemometer*, *ADCP*, *Sonar*, and *GPS*, with accompanied data receiver services that publish a series of sampled signals on the RTDS. These signals are subscribed by the decision support services *Path planner* and *Front end*, with the purpose to either compute value-increased decision support or directly visualise the data. The *Path planner* is a service that produces *Suggested path* on a regular interval, or at best effort if the solution time exceeds the prescribed execution interval. The results are published as event streams to the RTDS. Available to the end-user is a *Front end* service, which is a graphical user interface. This service publish event stream *User settings* to the RTDS. The user settings allow the end-user to change the behaviour of the path planner while it is running by means of adjusting algorithm parameters. The path planner and front end services likely run on separate processors, so user settings via the RTDS is an example of event stream data sharing between separate processes (and possibly machines) using a middleware communication service.

3.7.3 Results

The user can provide preferences and other configuration settings in the settings tab of the GUI, see Figure 3.36a, which is an event stream source. The algorithm component acts as an event stream sink and signal processor, because it makes use of the user settings and measurements of the environment, fish school, and vessel. Once a solution to the algorithm problem is available, it is sent from the algorithm component to the user interface component, which usually is periodically on regular intervals. A dashboard contains visual elements that are signal sinks (graphs and number displays), but also event stream sinks (algorithm status) and sources (buttons and sliders). A screenshot of this view is shown in Figure 3.36b.

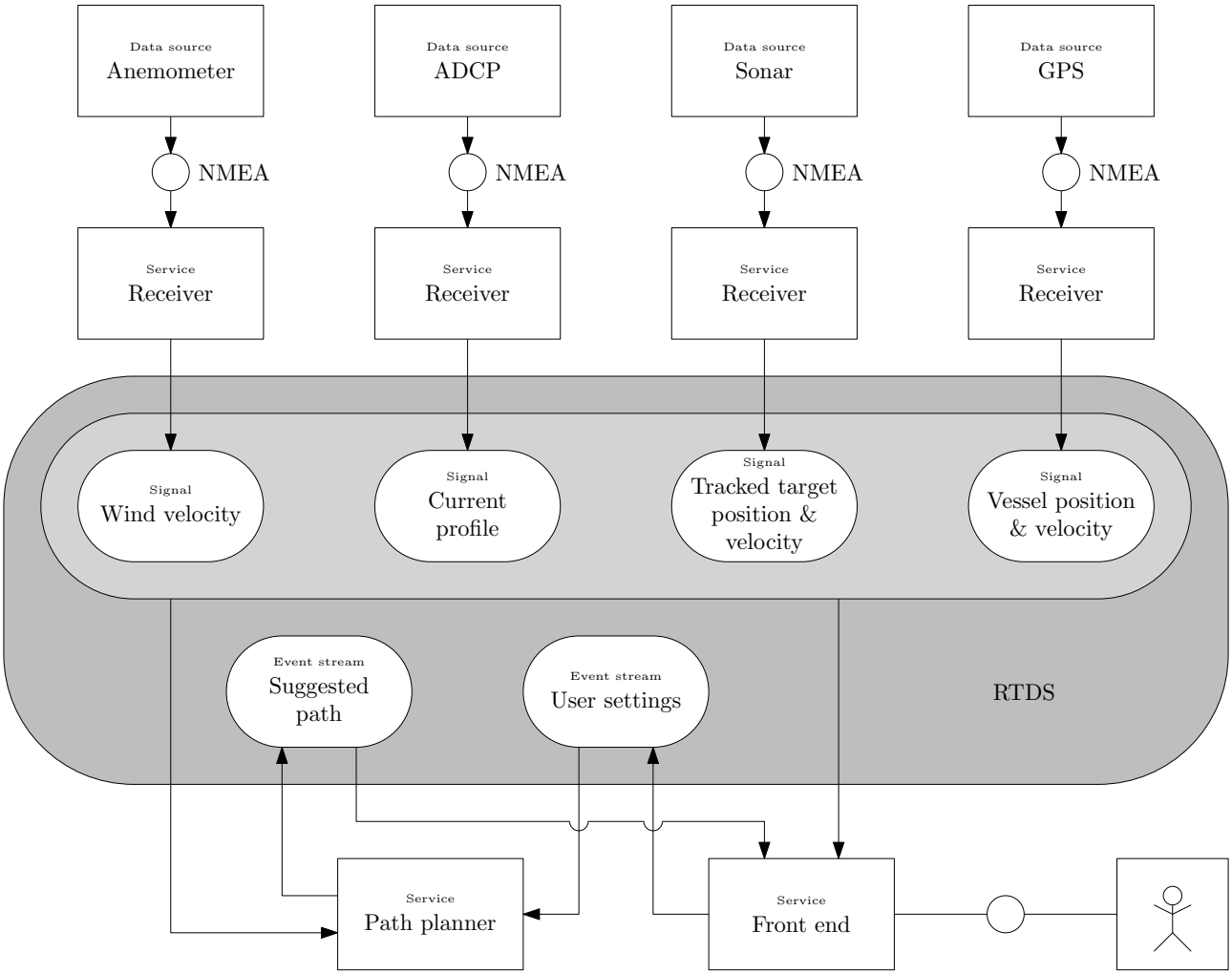
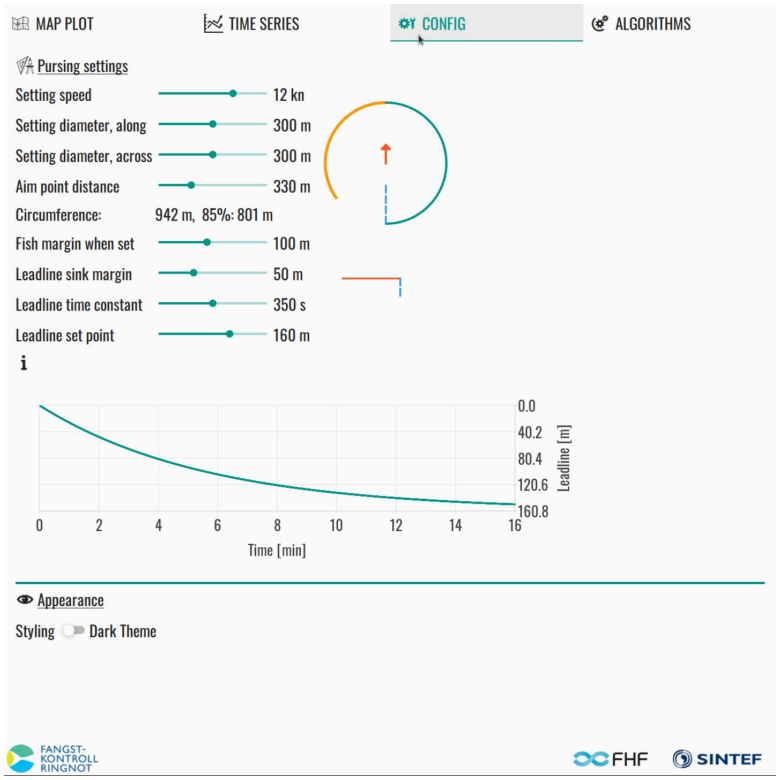
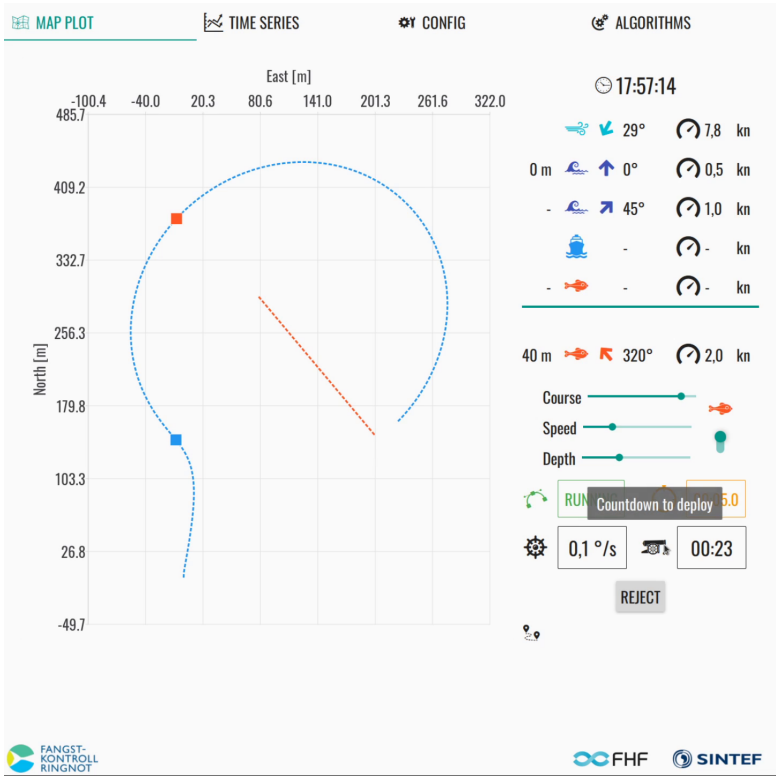


Figure 3.35: FMC block diagram for the purse seine deployment advisor. “ADCP” stands for “acoustic doppler current profiler”.



(a) View with preferences configuration.



(b) Decision support dashboard with visual elements.

Figure 3.36: Screenshots of decision support user interface. Image credit: Ref. [50].

Chapter 4

Summary and outlook

In this document, we have presented AMADEA—the Adaptable Maritime Decision Support Architecture—a reference model for maritime decision support systems. We have established a common terminology for the components and structure of such systems, described a number of general-purpose components that can be developed once and then reused in many different contexts, we have given practical advice on the implementation of MDSSes, and provided concrete examples for a variety of maritime systems and operations. Our hope is that this will give maritime system suppliers ideas, inspiration, and guidance for developing MDSSes, and that it will facilitate *collaborative* development. By this, we mean collaboration between different engineering disciplines, between different companies, and between industry, research institutes, and academia.

There is also much room for further development of AMADEA itself. For one thing, it could easily be extended with more general-purpose services—AI-based ones might be an idea. Another would be to describe it in a more standardised and precise manner, for example by developing a formal, machine-readable ontology of terms and concepts related to maritime systems. This would facilitate collaboration to an even greater extent, and allow things like auto-discovery of external systems and self-adapting MDSSes.

We close this document with a quote from Sprague [16]:

> Improving the performance is the ultimate goal of information systems—not the storage of data, the production of reports, or even "getting the right information to the right person at the right time". The ultimate objective must be viewed in terms of the ability of information systems to support the improved performance of people [...].

AMADEA provides a blueprint for how to do the storage, data processing, and information exchange, but the more exciting part about "the improved performance of people" is left to the actual MDSS implementations.

References

[1] Stian Skjong et al. 'Generic On-Board Decision Support System Framework for Marine Operations'. In: *Proceedings of the ASME 2019 38th International Conference on Ocean, Offshore and Arctic Engineering*. OMAE2019 (Glasgow, Scotland, 9th–14th June 2019). 2019. Chap. Volume 7A: Ocean Engineering, OMAE2019-95146. DOI: 10.1115/omae2019-95146.

[2] C. Matthew MacKenzie et al., eds. *Reference Model for Service Oriented Architecture*. soa-rm. OASIS Standard. Version 1.0. 12th Oct. 2006. URL: http://docs.oasis-open.org/soa-rm/v1.0/ (visited on 30/01/2023).

[3] European Commission. '13. General Annexes. European Commission Decision C(2022)2975'. In: *Horizon Europe Work Programme 2021–2022*. 10th May 2022. URL: https://ec.europa.eu/info/funding-tenders/opportunities/docs/2021-2027/horizon/wp-call/2021-2022/wp-13-general-annexes_horizon-2021-2022_en.pdf (visited on 12/07/2022).

[4] SINTEF Ocean. *Ratatosk*. Version 4.6.6. 20th May 2022. URL: https://ratatosk.smd.sintef.no/.

[5] Research Council of Norway, ed. *Project bank: Marine operations center (MOVE)*. See "Publications from Cristin" listed on the web page. URL: https://prosjektbanken.forskningsradet.no/en/project/FORISS/237929 (visited on 20/02/2023).

[6] Karl-Johan Reite, Jarle Ladstein and Lars T. Kyllingstad. *ImproVEDO åpen rapport. Åpne resultater fra KMB-prosjektet ImproVEDO*. Norwegian. SINTEF report A277720. SINTEF Fiskeri og havbruk, 24th June 2016. HDL: 11250/2447402.

[7] Ida Grong Aursand et al. *Development and assessment of novel technologies improving the fishing operation and on board processing with respect to environmental impact and fish quality (DANTEQ). Final Report*. SINTEF report A27309. SINTEF Fisheries and Aquaculture, 13th Nov. 2015. HDL: 11250/2466707.

[8] Karl-Johan Reite et al. 'Sustainable and Added Value Small Pelagics Fisheries Pilots'. In: *Big Data in Bioeconomy. Results from the European DataBio Project*. Ed. by Caj Södergård et al. Springer, Cham, 14th Aug. 2021, pp. 389–409. ISBN: 978-3-030-71069-9. DOI: 10.1007/978-3-030-71069-9_30.

[9] Lars T. Kyllingstad, Karl-Johan Reite and Per Gunnar Auran. *FishData system specification*. SMARTFISH H2020 deliverable D5.1. Version 2. Trondheim, Norway: SINTEF Ocean, 29th Oct. 2019. HDL: 11250/2999077.

[10] Severin Sadjina et al. 'Distributed Co-simulation of Maritime Systems and Operations'. In: *Journal of Offshore Mechanics and Arctic Engineering* 141.1, OMAE-17-1037 (Mar. 2019), p. 011302. DOI: 10.1115/1.4040473.

[11] Øyvind Smogeli et al. 'Open Simulation Platform. An Open-Source Project for Maritime System Co-Simulation'. In: *Proceedings of COMPIT'20*. 19th International Conference on Computer and IT Applications in the maritime industries. COMPIT'20 (Pontignano, Italy, 17th–19th Aug. 2020). Ed. by Volker Bertram. Hamburg: Technische Universität Hamburg–Harburg, 2020, pp. 239–253. ISBN: 978-3-89220-717-7. URL: http://data.hiper-conf.info/compit2020_pontignano.pdf (visited on 13/07/2022).

[12] SINTEF Ocean. *SIMO*. URL: https://www.sintef.no/globalassets/project/oilandgas/pdf/simo.pdf.

[13] Stian Skjong. 'Modeling and Simulation of Maritime Systems and Operations for Virtual Prototyping using Co-Simulations'. Doctoral Thesis. Trondheim: Norwegian University of Science and Technology, 2017, p. 301. ISBN: 978-82-326-2820-9.

[14] Karl-Johan Reite et al. 'FHSIM. Time Domain Simulation of Marine Systems'. In: *Proceedings of the ASME 2014 33rd International Conference on Ocean, Offshore and Arctic Engineering*. OMAE2014 (San Francisco, California, USA, 8th–13th June 2014). 2014. Chap. Volume 8A: Ocean Engineering, OMAE2014-23165. DOI: 10.1115/OMAE2014-23165.

[15] SINTEF Ocean. *FhSim*. 2022. URL: https://fhsim.com/ (visited on 13/07/2022).

[16] Ralph H. Sprague Jr. 'A Framework for the Development of Decision Support Systems'. In: *MIS Quarterly* 4.4 (Dec. 1980), pp. 1–26. DOI: 10.2307/248957.

[17] Pius Hättenschwiler. 'Neues anwenderfreundliches Konzept der Entscheidungsunterstützung'. German. In: *Absturz im freien Fall – Anlauf zu neuen Höhenflügen. Gutes Entscheiden in Wirtschaft, Politik und Gesellschaft*. Ed. by Hansjürg Mey and Daniel Lehmann Pollheimer. Forum für Universität und Gesellschaft, Universität Bern. Zürich: vdf Hochschulverlag, 2001, pp. 189–208. ISBN: 978-3-7281-2703-7.

[18] Daniel J. Power. *Decision Support Systems. Concepts and Resources for Managers*. Quorum Books, 2002. ISBN: 1-56720-497-X.

[19] FMC Research Staff. *FMC Notation Reference*. Ed. by Rémy Apfelbacher and Anne Rozinat. 2003. URL: http://www.fmc-modeling.org/notation_reference (visited on 08/02/2022).

[20] Jørgen Bang-Jensen and Gregory Z. Gutin. *Digraphs. Theory, Algorithms and Applications*. 2nd ed. Springer Monographs in Mathematics. Springer London, 2009. DOI: https://doi.org/10.1007/978-1-84800-998-1.

[21] National Marine Electronics Association. *NMEA 0183 Standard*. 0183 (NMEA). URL: http://www.nmea.org/content/nmea_standards/nmea_0183_v_410.asp (visited on 31/10/2022).

[22] *MODBUS APPLICATION PROTOCOL SPECIFICATION. V1.1b3*. Version 1.1b3. 2012. URL: https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf (visited on 31/10/2022).

[23] R. G. Brown and P. Y. C. Hwang. *Introduction to Random Signals and Applied Kalman Filtering*. 4th. J. Wiley, 1992. ISBN: 0-471-52573-1.

[24] Object Management Group. *Interface Definition Language*. Version 4.2. 19516 (ISO/IEC). Aug. 2018. URL: https://www.omg.org/spec/IDL/ISO/19516/PDF (visited on 30/01/2023).

[25] Object Management Group. *OMG Data Distribution Service*. formal/2015-04-10. OMG DDS. Version 1.4. Apr. 2015. URL: https://www.omg.org/spec/DDS/1.4 (visited on 30/01/2023).

[26] Angelo Corsaro. 'A Comparative Study of Data-Sharing Standards for the Internet of Things'. In: *Cutter IT Journal* 27.11 (Nov. 2014), pp. 23–29. ISSN: 1554-5946. URL: https://www.cutter.com/article/comparative-study-data-sharing-standards-internet-things-470036.

[27] Angelo Corsaro. *A Tale of Two Industrial IoT Standards: DDS and OPC-UA*. https://www.rtinsights.com/dds-opc-ua-industrial-iot-standards/. Accessed: 2023-01-30. RTInsights, 16th Aug. 2016.

[28] Open Platform Communications Foundation. *OPC Unified Architecture*. Version 1.05.02. 62541 (IEC). Nov. 2022. URL: https://reference.opcfoundation.org (visited on 30/01/2023).

[29] Open Platform Communications Foundation. *OPC Unified Architecture Field Exchange*. 10000-{80–84}. OPC UAFX. Version 1.00. Nov. 2022. URL: https://reference.opcfoundation.org (visited on 30/01/2023).

[30] Object Management Group. *Remote Procedure Call over DDS*. formal/17-04-01. DDS-RPC. Version 1.0. Apr. 2017. URL: https://www.omg.org/spec/DDS-RPC/1.0 (visited on 30/01/2023).

[31] Object Management Group. *DDS Security*. formal/2018-04-01. DDS-SECURITY. Version 1.1. June 2018. URL: https://www.omg.org/spec/DDS-SECURITY/1.1 (visited on 30/01/2023).

[32] Object Management Group. *DDS for eXtremely Resource Constrained Environments*. formal/20-02-01. DDS-XRCE. Version 1.0. Feb. 2020. URL: https://www.omg.org/spec/DDS-XRCE/1.0 (visited on 30/01/2023).

[33] Object Management Group. *Web-Enabled DDS*. formal/2016-03-01. DDS-WEB. Version 1.0. Mar. 2016. URL: https://www.omg.org/spec/DDS-WEB/1.0 (visited on 30/01/2023).

[34] Object Management Group. *OPC UA/DDS Gateway*. formal/20-01-01. DDS-OPCUA. Version 1.0. Jan. 2020. URL: https://www.omg.org/spec/DDS-OPCUA/1.0 (visited on 30/01/2023).

[35] Object Management Group. *DDS Interoperability Wire Protocol*. formal/2022-04-01. DDSI-RTPS. Version 2.5. Apr. 2022. URL: https://www.omg.org/spec/DDSI-RTPS/2.5 (visited on 30/01/2023).

[36] Object Management Group. *Why choose DDS?* https://www.dds-foundation.org/why-choose-dds. Accessed: 2023-02-01. 2023.

[37] Real-Time Innovations. *RTI Connext*. Version 6.1.2. 15th Dec. 2022. URL: https://www.rti.com/products/connext-lts.

[38] Kongsberg Defence & Aerospace. *InterCOM DDS*. URL: https://www.kongsberg.com/kda/what-we-do/defence-and-security/c4isr/intercom-dds/ (visited on 30/01/2023).

[39] Eclipse Foundation. *Cyclone DDS*. Version 0.10.2. 9th Sept. 2022. URL: https://github.com/eclipse-cyclonedds/cyclonedds.

[40] eProsima. *Fast-DDS*. Version 2.9.1. 19th Jan. 2023. URL: https://github.com/eProsima/Fast-DDS.

[41] Object Management Group. *Data Distribution Service (DDS) Guidebook. Distributed Computing Real-Time Messaging*. Ed. by R. W. Stavros. Version 1.0. Aug. 2021. URL: https://www.omgwiki.org/ddsf/doku.php?id=ddsf:public:guidebook:guidebook (visited on 30/01/2023).

[42] eProsima. *DDS-Router*. Version 1.1.0. 14th Dec. 2022. URL: https://github.com/eProsima/DDS-Router.

[43] Blue Ctrl. *X-CONNECT®*. https://bluectrl.io/. Accessed: 2023-02-02. 2023.

[44] Brunvoll. *Brunvoll*. https://www.brunvoll.no. Accessed: 2023-02-02. 2023.

[45] Stian Skjong, Karl-Johan Johan Reite and Karl Gunnar Aarsæther. 'Lumped, constrained cable modeling with explicit state-space formulation using an elastic version of Baumgarte stabilization'. In: *Journal of Offshore Mechanics and Arctic Engineering* 143.December (Mar. 2021), pp. 1–15. ISSN: 0892-7219. DOI: 10.1115/1.4050422.

[46] Karl Erik Kaasen. *Automatic Estimation of Parameters in Numerical Vessel Models from Laboratory Tests*. True Hydrodynamics Pilot Project deliverable OC2019 F-030. Version 1. Trondheim, Norway: SINTEF Ocean, 15th Feb. 2019.

[47] Norwegian Seafood Research Fund, ed. *Prosjektbasen: Operation monitoring and decision support for purse seiners PurSense*. URL: https://www.fhf.no/prosjekter/prosjektbasen/900886/ (visited on 20/03/2023).

[48] Karl-Johan Reite, Jarle Ladstein and Joakim Haugen. 'Data-Driven Real-Time Decision Support and its Application to Hybrid Propulsion Systems'. In: *Proceedings of the ASME 2017 36th International Conference on Ocean, Offshore and Arctic Engineering*. OMAE2017 (Trondheim, Norway, 25th–30th June 2017). 2017. Chap. Volume 7B: Ocean Engineering, OMAE2017-61031. DOI: 10.1115/OMAE2017-61031.

[49] Severin Sadjina et al. 'Seismic RTDT: Real-Time Digital Twin for Boosting Performance of Seismic Operations'. In: *Proceedings of the ASME 2019 38th International Conference on Ocean, Offshore and Arctic Engineering*. OMAE2019 (Glasgow, Scotland, 9th–14th June 2019). 2019. Chap. Volume 7A: Ocean Engineering, OMAE2019-95885. DOI: 10.1115/omae2019-95885.

[50] Joakim Haugen and Lars T. Kyllingstad. *Baneplanlegging for ringnotfartøy. Sluttrapport fra arbeidspakke 2 i prosjektet "Fangstkontroll i notfiske etter pelagiske arter: Fase 2"*. Norwegian. SINTEF report 2021:00578. Trondheim: SINTEF Ocean, 1st June 2021. HDL: 11250/2991650.

[51] ADLINK Technology Ltd. *OpenSplice DDS*. Version 6.9.210323. 23rd Mar. 2021. URL: https://github.com/ADLINK-IST/opensplice.

[52] Joel A E Andersson et al. 'CasADi – A software framework for nonlinear optimization and optimal control'. In: *Mathematical Programming Computation* 11.1 (2019), pp. 1–36. DOI: 10.1007/s12532-018-0139-4.

[53] Andreas Wächter and Lorenz T. Biegler. 'On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming'. In: *Math. Programming* 106 (1 May 2006), pp. 25–57. ISSN: 0025-5610. DOI: 10.1007/s10107-004-0559-y.

[54] Qt Group. *Qt*. Version 5.15.8. 4th Jan. 2022. URL: https://qt.io.

[55] Open Simulation Platform. *cosim*. Version 0.7.0. 26th Apr. 2022. URL: https://github.com/open-simulation-platform/cosim-cli.

SINTEF

Technology for a better society