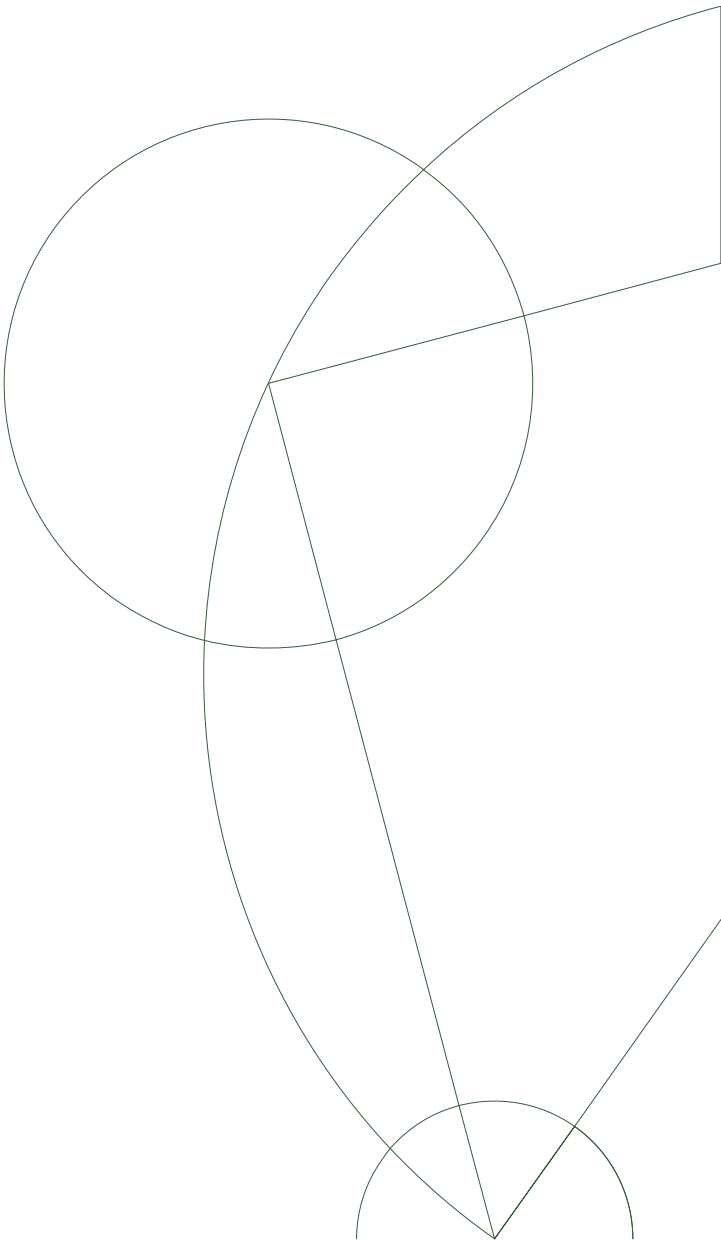
FACULTY OF SCIENCE
UNIVERSITY OF COPENHAGEN

MSc thesis

Søren Lund Hess-Petersen - pws412

Using Model-Based testing to test MCP Instance-Specifications



Academic advisor: Michael Kirkedal Thomsen
<m.kirkedal@di.ku.dk>
Submitted: October 31, 2019

Using Model-Based testing to test MCP Instance-Specifications

Søren Lund Hess-Petersen - pws412
DIKU, Department of Computer Science,
University of Copenhagen, Denmark

October 31, 2019

MSc thesis

Author:	Søren Lund Hess-Petersen - pws412
Affiliation:	DIKU, Department of Computer Science, University of Copenhagen, Denmark
Title:	Using Model-Based testing to test MCP Instance-Specifications /
Academic advisor:	Michael Kirkedal Thomsen <m.kirkedal@di.ku.dk>
Submitted:	October 31, 2019

Abstract

Dansk Resumé

Preface

This Master's thesis is submitted in fulfilment of the master programme in Computer Science at the University of Copenhagen for Søren Lund Hess-Petersen.

Contents

Preface **iv**

1 Introduction **1**
1.1 Learning Objectives 1
1.2 Scope 1
1.3 Limitations 1
1.4 Structure 1

2 Background **2**
2.1 The Maritime Connectivity Platform 2
2.1.1 Identity Registry 3
2.1.2 Service Registry 3
2.1.3 Messaging Service 3
2.2 Monadic Parsing 3
2.3 Property-Based Testing 3
2.4 Model-Based Testing 4
2.4.1 Finite State Machines (FSM) 4
2.4.2 State Charts 4
2.4.3 Unified Modeling Language (UML) 4

3 Analysis **6**
3.1 Testing the MCP 6
3.2 Model Structure 6
3.3 Automated Model-Based testing of MCP 6

4 Work/Design **7**
4.1 Parser Grammars 7
4.1.1 ServiceSpecificationScema 7
4.1.2 ServiceDesignSchema 7
4.1.3 ServiceInstanceSchema 7
4.1.4 General Grammar 7

4.2 Implementation in the MCP 7
4.3 Technical Implementation 7
4.3.1 Executing Instructions 7
4.4 Testing 7
4.5 Experiments 7

5 Results **8**

6 Discussion **9**

7 Conclusion **10**
7.1 Conclusions 10
7.2 Future work 10
7.3 Summary 10

Bibliography **11**

A Appendix **12**

Introduction

1

1.1 Learning Objectives

1.2 Scope

1.3 Limitations

1.4 Structure

- **Chapter 2: Background**
  This chapter describes the techniques, theories, and components that are used or analyzed throughout the thesis.
- **Chapter 3: Analysis**
- **Chapter 4: Work/Design**
- **Chapter 5: Results**
- **Chapter 6: Discussion**
- **Chapter 7: Conclusion**

Background

2

2.1 The Maritime Connectivity Platform

The Maritime Connectivity Platform (MCP), which is formally known as the Maritime Cloud is a platform, developed by EfficienSea2 [1], which is led by the Danish Maritime Consortium. MCP is a communication framework, that is to ensure efficient, reliable and secure communication, and exchange of information in the maritime sector. The goal of the platform is to connect maritime stakeholders with maritime information services.

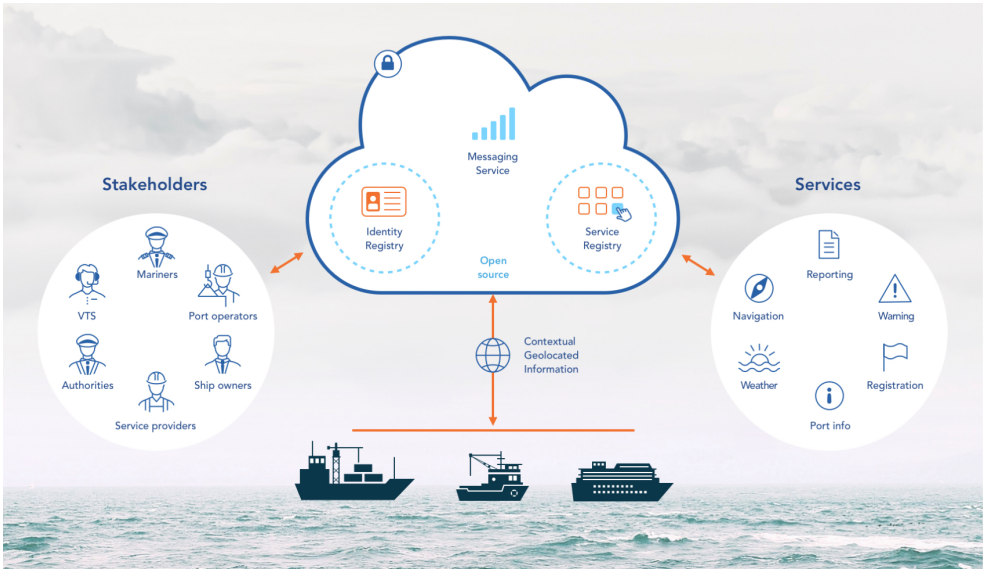


Figure 2.1: Diagram, describing the structure of MCP [1]

A high-level diagram, describing the structure of the MCP can be seen in Figure 2.1. Here it is shown that maritime stakeholders are connected with the maritime services through the Identity Registry, the Service Registry and the Messaging Service.

2.1.1 Identity Registry

Here the relevant information regarding maritime stakeholders are stored. This information needs to be authorized and stored at a safe location in order for the security of the MCP to be sufficient. The Identity Registry on the MCP is equivalent to the Central Person Registry of a country. The Identity Registry is vital to the MCP in it ensures the solution's authenticity, integrity, and confidentiality. Maritime stakeholders are, through the Identity Registry provided with a single login to all Maritime Services. [1]

2.1.2 Service Registry

The Service Registry acts to Maritime Services as the Identity Registry acts to the Maritime Stakeholders. Here all Maritime Services are registered and stored. The Service Registry holds both commercial and non-commercial, as well as authorized and non-authorized services, either free of charge or for a fee. The Service Registry is comparable with the App Store or Google Play in that it distributes services of all kinds to registered users. [1]

2.1.3 Messaging Service

"An information broker that intelligently exchanges information between communication systems connected to the platform, taking into account the current geographical position and communication links available to the recipient." [1]

2.2 Monadic Parsing

2.3 Property-Based Testing

Property based testing, also known as Automation Testing, is the principle of testing properties of code, rather than instances, as is done in manual unit testing. Unit tests are fast and easy to set up and execute, however, in most instances they only provide coverage to a certain degree. Depending on the complexity of the given program, most programmers can come up with tests, covering most of the given program, however many edge-cases are unintuitive and would require extensive testing to determine manually.

Property-based testing solves this problem by setting up test cases that take into account mathematical models, describing the desired behavior of the code. Once such a test case has been created, semi-random execution instances can be run to the point of literal exhaustion, and thus a relatively small program can test for thousands of occurrences at once.

An example of this can be seen in Listing 2.1, that describe a snippet of Haskell quicksort code. A property of quicksort is that a list, sorted once is

equivalent to a list, sorted twice, which is what the function `prop_idempotent` validates.

```
-- file: ch11/QC-basics.hs
import Test.QuickCheck
import Data.List

qsort :: Ord a => [a] -> [a]
qsort [] = []
qsort (x:xs) = qsort lhs ++ [x] ++ qsort rhs
    where lhs = filter (< x) xs
          rhs = filter (>= x) xs

prop_idempotent xs = qsort (qsort xs) == qsort xs
```

Listing 2.1: Example of Property-Based Testing. [2]

2.4 Model-Based Testing

The principle of Model-based testing is to check behavior of software against predictions made by a model. Behavior can be described in a variety of manners, including data flow, control flow, dependencies, decision trees/cycles, and state transition machines. Model-based testing is good at describing the behavior of a system, when it reacts to a specific action, which is determined by another specific model. Using this technique, the behavior of a system can easily be determined and validated.

2.4.1 Finite State Machines (FSM)

A finite state machine is, as the name suggests, a mathematical machine, consisting of a collection of states. An arbitrary FSM has a start state along with a collection of states that are accessed through various status- and/or input-combinations. An example of this can be seen in Figure 2.2 as well as Table 2.1. These describe the control flow of a turnstile, which allows for one pass through, whereafter it prompts for a coin before allowing another pass through.

2.4.2 State Charts

2.4.3 Unified Modeling Language (UML)

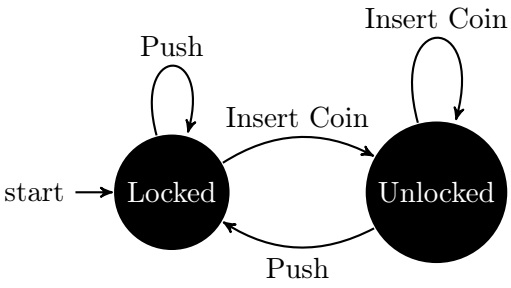


Figure 2.2: Example of a finite state machine, describing a turnstile. This figure relates to Table 2.1.

State	Action	Next State	Output
Locked	Insert Coin	Unlocked	Unlocks turnstile, allowing passage.
Locked	Push	Locked	Blocks passage.
Unlocked	Insert Coin	Unlocked	Returns coin.
Unlocked	Push	Locked	Allows passage and locks.

Table 2.1: Example of a finite state machine, describing a turnstile. This table relates to Figure 2.2.

Analysis

3

In this chapter, I will present the data, made available by the

3.1 Testing the MCP

3.2 Model Structure

3.3 Automated Model-Based testing of MCP

Work/Design

4

4.1 Parser Grammars

4.1.1 ServiceSpecificationScema

4.1.2 ServiceDesignSchema

4.1.3 ServiceInstanceSchema

4.1.4 General Grammar

4.2 Implementation in the MCP

4.3 Technical Implementation

4.3.1 Executing Instructions

4.4 Testing

4.5 Experiments

Results

5

Discussion

6

Conclusion

7

7.1 Conclusions

7.2 Future work

7.3 Summary

Bibliography

[1] EfficienSea2's website
https://efficiensea2.org/solution/maritime-connectivity-platform/

[2] Real World Haskell, Bryan O'Sullivan, Don Stewart, and John Goerzen, Chapter 11
http://book.realworldhaskell.org/read/testing-and-quality-assurance.html

Appendix A