Carl von Ossietzky Universität Oldenburg
Abteilung Systemanalyse und -optimierung

OFFIS – Institut für Informatik

Projektgruppe – **Ma**ritime **T**est- und **E**xperimentierplattform

Abschlussdokumentation

Teilnehmer: Luca Gatterdam
Adrian Jagusch
Hendrik Kahlen
Sergej Kidrowski
Thorben Kloppe
Berislaw Kock
Philipp Mudra
Mark Otten
Sebastian Schmidt
Maximilian Wappler
Tim Wiechmann

Betreuer: Prof. Dr.-Ing. Axel Hahn
Ing. Mohamed Abdelaal
M.Sc. Marius Brinkmann
Dipl.-Ing. Arne Stasch
M.Sc. Peter Tank

Oldenburg, 3. Oktober 2017

Inhaltsverzeichnis

Abstract: *Die vorliegende Abschlussdokumentation fasst die Ergebnisse der Projektgruppe „Maritime Test- und Experimentierplattform“ (kurz: PG MATE) zusammen. Die PG MATE hat sich mit der Entwicklung und den Tests eines ganzheitlichen Konzepts für ein autonom fahrendes Schiff beschäftigt.*

Die entwickelte Plattform verfügt über eine komplexe Systemarchitektur mit küsten- und schiffsseitigen Komponenten. Die Plattform wurde in eine vorhandene Systemarchitektur eingebettet. Diese besteht unter anderem aus dem Forschungsschiff Zuse und dem virtuellen Testbed LABSKAUS im Rahmen der eMIR-Initiative. Das Produkt soll im Rahmen weiterer Forschungsprojekte im maritimen Bereich an der Universität Oldenburg und im OFFIS genutzt werden. Die Entwicklung und die Tests fanden mit Hilfe des Forschungsschiffs Zuse statt. Die Plattform wurde jedoch so entwickelt, dass sie über eine Schnittstelle, über die Daten im NMEA2000-Standard ausgetauscht werden, auch auf anderen Schiffen eingesetzt werden könnte.

Geplant war die Schaffung eines autonom fahrenden Schiffs in drei aufeinander aufbauenden Schritten. Der erste Schritt befasst sich mit der Fernsteuerung eines Schiffs. Diese wurde erfolgreich umgesetzt. Der zweite Schritt sah das automatische Abfahren einer vorgegebenen Route durch das Schiff vor. Dieser wurde weitestgehend umgesetzt. Der dritte und letzte Schritt hatte das vollständig autonom fahrende Schiff zum Ziel, dass dynamisch auf diverse Umwelteinflüsse, wie Hindernisse im Wasser oder andere Schiffe auf Kollisionskurs, reagieren kann. Dieser Schritt wurde nicht umgesetzt.

1 Einleitung

Die Entwicklung autonom fahrender Autos ist schon weit vorangeschritten und ein wichtiges Thema, das nicht nur die Automobilindustrie, sondern auch die Gesellschaft im Allgemeinen beschäftigt. Medial nicht ganz so präsent sind Versuche und Fortschritte autonom fahrender Schiffe. Aber auch in dieser Branche hat das Rennen um das erste kommerziell einsetzbare Fahrzeug begonnen. Rolls-Royce rief zum Beispiel 2016 eine Initiative namens „Advanced Autonomous Waterborne Applications“ (Aawa) ins Leben und testet derzeit an der finnischen Küste Konzepte für das autonome Schiff.

Die Projektgruppe „Maritime Test- und Experimentierplattform“ (kurz PG MATE) soll sich mit der Schaffung eines Produkts beschäftigen, welches als Plattform für verschiedene Forschungsprojekte im maritimen Bereich und insbesondere im Bereich des autonomen Fahrens genutzt werden kann. Die PG MATE reiht sich dabei in eine Vielzahl anderer Projektgruppen der Universität Oldenburg ein, die mit ähnlichen Projektaufgaben im maritimen Bereich beschäftigt waren. Ein Beispiel dafür sind die Projektgruppen MOPS I bis IV.

1.1 Motivation

Die Interessen und Ziele für den Einsatz autonom fahrender Schiffe in der Schifffahrt sind, wie autonom fahrende Autos im Straßenverkehr, vielfältig. Zu den Bedeutendsten zählt die Erhöhung der Verkehrssicherheit und die damit verbundene Reduzierung der Unfallzahlen. Durch eine effizientere Fahrweise und optimierte Routen entstehen zudem ökologische und ökonomische Vorteile. Außerdem kann auf diese Weise die notwendige Schiffsbesatzung verkleinert bzw. entlastet werden.

Die Universität Oldenburg arbeitet an verschiedenen Projekten im maritimen Bereich und hat ein Interesse daran, die hier gewonnenen Erkenntnisse zu vertiefen. Die PG MATE und ihre Mitglieder möchten durch die Projektgruppenarbeit unter anderem die Arbeit in einer großen Arbeitsgruppe erlernen und die Kenntnisse in verschiedenen Technologien und Arbeitsweisen vertiefen. Eine detailliertere Analyse der Motivation der Projektgruppenmitglieder und weiterer Stakeholder wird in Abschnitt 5.1 gegeben.

1.2 Einordnung

Die zu entwickelnde Plattform soll im Rahmen von Forschungstätigkeiten im Projekt LABSKAUS verwendet werden, um unter anderem das Testen von Konzepten und Software-Systemen im Projekt „eMaritime Integrated Reference Platform (eMIR)“ durch die Nutzung eines physischen Forschungsboots zu unterstützen. Die Projektarbeit findet in Zusammenarbeit mit Mitarbeitern der Abteilung „Systemanalyse und -optimierung (SOA)“ des Departments für Informatik an der Carl von Ossietzky Universität Oldenburg und des OFFIS, einem An-Institut der Universität Oldenburg, statt. Diese beiden Forschungsstätten arbeiten zusammen an verschiedenen Projekten im maritimen Bereich. Detailliertere Informationen zur Einordnung des Projekts finden sich in Abschnitt 4.

Bei möglichen zukünftigen Produkteinsätzen der Plattform ist die sogenannte Zivilklausel der Grundordnung der Universität Oldenburg in Bezug auf die rein nicht militärische Nutzung zu beachten.

1.3 Problemstellung

Aus der Aufgabenstellung der Projektgruppenbetreuer und der zuvor beschriebenen Motivation ergibt sich folgende zentrale wissenschaftliche Fragestellung der PG MATE:

„Wie muss eine maritime Test- und Experimentierplattform aussehen, auf der im Rahmen der Forschung an der Universität Oldenburg und dem OFFIS Konzepte und Systeme zum autonomen Fahren von Schiffen getestet und weiterentwickelt werden können?“

1.4 Zielsetzung

Aus der wissenschaftlichen Fragestellung, den Vorgaben der Projektgruppenbetreuer und dem eigenen Anspruch der Projektgruppenmitglieder ergibt sich die Zielsetzung der PG MATE:

- Es soll ein ganzheitliches Konzept für das autonom fahrende Schiff entwickelt und getestet werden.
- Das zu entwickelnde Produkt soll als Plattform entwickelt werden, welche weitestgehend unabhängig vom Forschungsschiff Zuse ist und mit möglichst geringem Aufwand auch auf andere Schiffe übertragen werden können soll. Dafür ist es notwendig, das die PG MATE sich bei der Entwicklung an in der Schifffahrt verbreitete Standards und Konzepte hält.

- Bei der Entwicklungsarbeit soll darauf geachtet werden, drei von den Projektgruppenbetreuern vorgegebenen Entwicklungsstufen nacheinander abzuarbeiten. Diese Entwicklungsstufen sind:

  1. Fernsteuerung eines Schiffs
  2. Das automatische Abfahren von vorgebenden Routen
  3. Das vollkommen autonome Fahren und dynamisches Reagieren auf Umweltbedingungen

- Das zu entwickelnde Produkt soll aus einer küsten- und einer schiffsseitigen Plattform bestehen, die sich gegenseitig ergänzen.
- Das zu entwickelnde Produkt soll nur für Forschungszwecke eingesetzt werden. Die Schaffung eines verkaufsfähigen Produkts soll nicht angestrebt werden. Es soll eine Grundlage für weitere Forschungsarbeiten auf den beschriebenen Themengebieten erarbeitet werden.
- Das zu entwickelnde Produkt soll als Grundlage für weitere Forschungen im maritimen Bereich genutzt werden und soll entsprechend in die bereits vorhandene Projekt- und Systemumgebung der Universität Oldenburg und des OFFIS eingegliedert werden. Dies kann zum Beispiel bereits vorhandene allgemeine Projektstrukturen, GUI-Komponenten und Regelungssysteme umfassen.
- Das Ergebnis der PG MATE soll von anderen Projektgruppen im maritimen Bereich bzw. möglichen direkten Folgeprojektgruppen genutzt und weiterentwickelt werden. Deshalb ist auf eine ordentliche Dokumentation der Entwicklung und Ergebnisse zu achten.
- Das zu entwickelnde Produkt soll methodisch und umfangreich getestet werden, um eine hohe Softwarequalität zu erhalten.

2 Projektvorgehen

In diesem Abschnitt soll kurz das allgemeine Projektvorgehen der PG MATE vorgestellt werden. Dies umfasst die Beschreibung des Vorgehensmodells, die Beschreibung der Projektplanung sowie eingesetzte Methoden und Werkzeuge.

2.1 Organisation

In diesem Kapitel werden die Struktur und die Elemente der Projektgruppenorganisation vorgestellt und beschrieben. Dies umfasst die Beschreibung des Auswahlprozesses eines Vorgehensmodells, die Beschreibung des gewählten Vorgehensmodells, die Vorstellung des internen und externen Projektplans sowie die Beschreibung und Aufteilung fester Rollen in der Projektplanung.

2.1.1 Vorgehensmodell

Ein Vorgehensmodell soll im Allgemeinen den Projektverlauf in strukturierte Abläufe einteilen. Den sich daraus ergebenden strukturierten Abschnitte werden entsprechende Methoden und Techniken zugeordnet. Ohne ein geeignetes Vorgehensmodell drohen Projekte oft zu scheitern. Mit einem Vorgehensmodell können verschiedene ähnlich aufgebaute Projekte in der Softwareentwicklung umgesetzt werden (vgl. [BPK15], S. 3 f.). Im Folgenden wird kurz auf die Anforderungen an ein Vorgehensmodell durch die gegebenen Rahmenbedingungen für die PG MATE eingegangen. Im Anschluss werden verschiedene Vorgehensmodelle aufgeführt und die Wahl eines dieser für die Projektgruppe begründetet. Dieses Vorgehensmodell wird im Anschluss detaillierter vorgestellt.

Ermittlung eines passenden Vorgehensmodells

Die zentrale Anforderung an ein Vorgehensmodell für die PG MATE ist die Unterstützung der Arbeit mit einer Gruppe von bis zu 12 Personen. Zudem muss das gewählte Vorgehensmodell über einen längeren Zeitraum von einem Jahr standhalten. Auch muss das Vorgehensmodell schnelle Reaktionen in einem komplexen Softwaresystem ermöglichen. Dabei muss den, durch den Einsatz im Schiffsverkehr, erhöhten sicherheitskritischen Anforderungen Sorge getragen werden. Neben dem klassischen Wasserfallmodell ähnlichen Vorgehensmodellen sind in der Praxis agile Methoden weit verbreitet.

Klassische Modelle sind dabei in mehrere Phasen unterteilt, die aufeinander aufbauen und nacheinander in einer festgelegten Reihenfolge abgearbeitet werden müssen. Wichtig ist hierbei die konsequente Durchführung einer einzelnen vorher geplanten Phase. In einer vorherigen Phase getroffene Entscheidungen können nur schwer rückgängig gemacht werden. Der Vorteil klassischer Modelle liegt in der hohen Planungssicherheit durch die strikte Einhaltung der einzelnen Phasen. Diese ist jedoch nur bei über den Projektverlauf konstanten Anforderungen gegeben. In der Konzeptionsphase entstandene Fehler können in klassischen Modellen nur schwierig behoben werden und zusätzliche oder veränderte Anforderungen nur schwer umgesetzt.

Agile Vorgehensmodelle weisen einen höheren Grad an Flexibilität auf. Ein bekanntes und etabliertes agiles Vorgehensmodell ist das Scrum-Modell. Bei agilen Modellen wird in der Regel kein langfristiger Plan bis zum Ende des Projekts erstellt. Stattdessen wird der gesamte Bearbeitungszeitraum in mehrere Bearbeitungszyklen eingeteilt, in denen jeweils einer oder mehrere Themenbereiche bearbeitet, getestet und abgeschlossen werden. Im Allgemeinen dauern diese Bearbeitungszyklen (Sprints) zwischen einer und vier Wochen. Auf diese Weise kann flexibler auf sich ändernde Anforderungen reagiert werden und Fehler in der Konzeption können besser korrigiert werden. Allerdings geht dadurch die Planungssicherheit der klassischen Modelle verloren (vgl. [Jak15], S. 125-127 und 134).

Gewähltes Vorgehensmodell

Für die Organisation der Projektgruppe wurde ein modifiziertes Vorgehensmodell auf Basis des V-Modells gewählt. Dabei wurde das V-Modell verschlankt und auf zentrale Elemente reduziert. Zudem wird mit einer iterativen Vorgehensweise gearbeitet. In den einzelnen Iterationen wird nach dem V-Modell entwickelt.

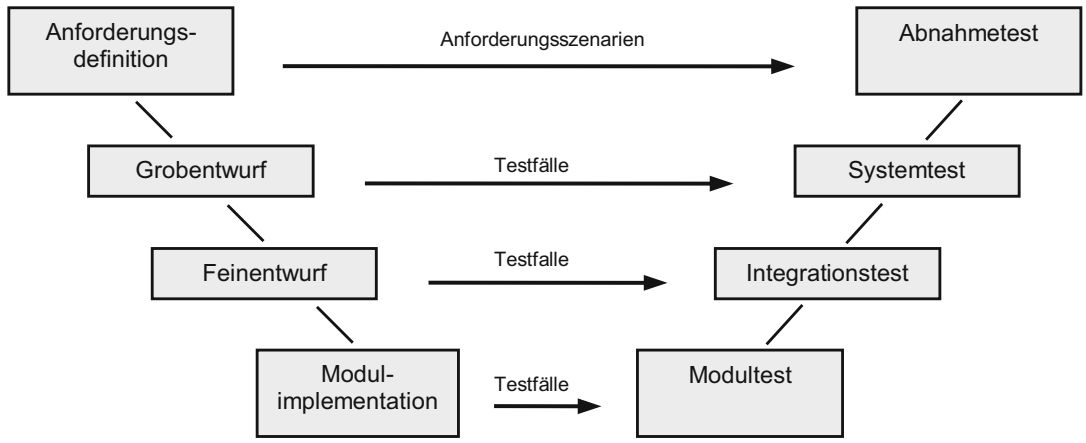


Abbildung 1: Aufbau des V-Modells ([BPK15], S. 24)

Das V-Modell basiert auf dem Wasserfallmodell. Der linke Pfad des V-Modells beschreibt wie beim Wasserfallmodell die einzelnen aufeinander folgenden Phasen (siehe Abbildung 1). Auf dem rechten Pfad werden diese durch die passenden Tests ergänzt (vgl. [BPK15], S. 24).

Die Phasen des V-Modells wurden modifiziert und um eine zusätzliche Phase zum Übergang in den nächsten Zyklus erweitert. Die in den einzelnen Phasen des V-Modells vorgesehenen Tests bzw. Teststufen sind in den Phasen des erarbeiteten Vorgehensmodells zu finden. Zudem wird die Methode der testgetriebenen Entwicklung während der einzelnen Zyklen angewandt (zur Methode siehe Abschnitt 2.2.1).

Je nach Literatur unterscheidet sich die Anzahl der Phasen und deren Zusammenspiel im V-Modell. Konstant ist jedoch die charakteristische „V“-Form und die Gegenüberstellung von Entwurfs- und Testsphasen. In der Praxis exisiteren diverse Vorgehensmodelle die auf dem allgemeinen V-Modell aufbauen. Ein Beispiel ist das V-Modell XT, welches weit verbreitet ist und als Standard für IT-Entwicklungsprojekte der Bundesrepublik Deutschland dient. Das V-Modell XT erlaubt umfangreiche Tailoring-Maßnahmen, um das Standard-Vorgehensmodell auf die jeweilige Projektsituation anpassen zu können (vgl. [Hei07], S. 73-76).

Das Tailoring des V-Modells ist jedoch kritisch zu sehen. Durch dieses kann zwar auf viele Herausforderungen und Probleme reagiert werden, um eine sinnvolle Lösung zu finden. Jedoch besteht das Risiko, dass durch Veränderungen am Vorgehensmodell falsche Annahmen getroffen und relevante Aspekte bei der Analyse nicht berücksichtigt werden.

Bestimmte Informationen und Zusammenhänge zu dem zu schaffenden Projekt ergeben sich erst während ihrer Untersuchung. Dies verändert bestimmte Anfangsannahmen mit welchen das Produkt geplant wurde und erfordert weitere Korrekturschleifen.

Diese Wahl des Vorgehensmodells wurde getroffen, da das V-Modell zum einen ein in der Praxis verbreitetes und etabliertes Vorgehensmodell ist. Zum anderen zwingt das V-Modell die Gruppe zu einem strukturierten Vorgehen. Bei diesem müssen zunächst Anforderungen erhoben und ein Systementwurf erstellt werden. Erst zum Schluss kann die Implementierung erfolgen. Eine Kombination mit einem iterativen Vorgehen ist sinnvoll, da so schon nach kurzer Zeit ein benutzbares Produkt entsteht. Dieses Vorgehen ermöglicht zudem eine schrittweise Verbesserung des Arbeitsprozesses. Außerdem sieht der von den Projektgruppenbetreuern vorgegebene Projektplan mehrere aufeinander aufbauende Meilensteine, bis hin zum autonom fahrenden Schiff, vor. Um diese nacheinander abzuarbeiten, eignet sich besser ein Vorgehensmodell mit mehreren Bearbeitungszyklen. Um den erhöhten sicherheitskritischen Anforderungen gerecht zu werden, ist es notwendig von starren gleichlangen Bearbeitungszyklen abzuweichen. Die Fertigstellung eines Bearbeitungszyklus ist dann von der kompletten und sicheren Fertigstellung aller Teilkomponenten abhängig zu machen.

Deshalb wurde ein angepasstes Vorgehensmodell gewählt. Die Anpassung von standardisierten Vorgehensmodellen an die jeweilige Projektsituation ist in der Praxis stark verbreitet und wird in der Literatur auch empfohlen. Dieser Vorgang wird „Tailoring“ genannt (vgl. [SDW+10], S. 65 f.).

Das Vorgehensmodell mit den Anpassungen am V-Modell gliedert sich wie folgt. Die Iterationen, in denen das V-Modell durchlaufen wird, werden in der PG MATE Zyklen genannt. Innerhalb eines Zyklus wird nach dem V-Modell vorgegangen. Neben den Zyklen gibt es Meilensteine. Die Meilensteine beruhen auf den externen Vorgaben der Stakeholder, wohingegen die Zyklen die interne Planung widerspiegeln (siehe dazu Abschnitt 2.1.2).

Anders als bei agilen Vorgehensmodellen wie Scrum dauert ein Zyklus nicht zwangsläufig immer gleich lang. Die Dauer eines Zyklus ergibt sich aus der in der Entwurfsphase erarbeiteten User-Storys. Eine User-Story ist eine in Alltagssprache formulierte Software-Anforderung. User-Storys werden in agilen Vorgehensweisen wie Scrum zur Spezifikation von Anforderungen eingesetzt. Kurze Zyklen mit einer abgewandelten Form des V-Modells bieten die Möglichkeit, sich immer wieder auf geänderte Anforderungen einstellen zu können und nicht den Planungsprozess wieder von vorn zu beginnen und die abgearbeiteten Phasen erneut zu durchlaufen. Aus den agilen Vorgehensmodellen wurden

zudem einige weiter Artefakte, wie ein Weekly-Update übernommen. Diese werden in Abschnitt 2.3.3 näher erläutert.

Das Ergebnis der Tailoring-Maßnahmen am V-Modell wird im Folgenden, in den in jedem Zyklus von der Projektgruppe zu durchlaufenden Phasen, beschrieben:

- **Konzeption**: Hierbei soll zunächst festgelegt werden, was im jeweiligen Zyklus erreicht werden soll. Dafür werden mit Hilfe der Brainstorming-Methode die notwendigen User-Storys erarbeitet. Eine User-Story ist eine in Alltagssprache formulierte Anforderung (vgl. [WM17], S. 50).

  Diese Phase entspricht damit der Anforderungsdefinition aus dem V-Modell. Im Anschluss sollen diese User-Storys inklusive der zugehörigen Akteure in Issues im GitLab festgehalten werden.

  Der letzte Schritt der Konzeption ist die Abstimmung der erarbeiteten Ergebnisse mit den Stakeholdern. Auf diese Weise können offene Fragen geklärt, etwaige entstandene Fehler frühzeitig identifiziert und beseitigt werden.

- **Entwurf**: In der Entwurfsphase werden zunächst die User-Storys konkretisiert und ergänzt. Dafür ist es notwendig die Beschreibung der entsprechenden User-Story anzupassen sowie Akzeptanzkriterien festzulegen. Akzeptanzkriterien beschreiben, wann eine User-Story als erfolgreich umgesetzt gilt. Wann dies erreicht ist, legt die „Definition of Ready“ fest, die sich wie folgt zusammensetzt:

  - Akzeptanzkriterien definiert
  - Ausführliche Beschreibung vorhanden
  - Spezifikation und Anforderungen sind verlinkt
  - Alle offenen Punkte sind geklärt
  - Fehlerfälle sind identifiziert

  Im nächsten Schritt wird die grobe Architektur in der Spezifikation beschrieben (analog zum Grobentwurf aus dem V-Modell). Dies umfasst die Hard- und Software, sowie die Festlegung, Beschreibung und Abgrenzung der Komponenten. Außerdem erfolgt hier bereits eine grobe Definition der Schnittstellen zwischen den Komponenten sowie der Entwurf der Systemtests. Ein Systemtest ist die Teststufe, bei der das gesamte System gegen alle funktionalen und nichtfunktionalen Anforderungen getestet wird.

Im Anschluss werden die User-Storys aufgeteilt und den Teams zugewiesen. Sollten sich während der Erarbeitung der Entwurfsphase Fragen ergeben haben, sind diese mit Hilfe der User-Storys mit den Stakeholdern abzustimmen.

– **Konstruktion**: In der Konstruktionsphase erfolgt die Umsetzung der zuvor definierten Aufgaben und User-Storys. Dabei wird zunächst für jede Aufgabe der jeweiligen Untergruppe ein Feinentwurf erstellt. In diesem wird zunächst festgelegt, wie die Anforderungen konkret umzusetzen sind.

  Zudem werden die Integrationstestfälle festgelegt. Entscheidungen, die während der Umsetzung getroffen werden, zum Beispiel die Verwendung bestimmter Technologien, sind methodisch zu evaluieren und in den Repositories zu vermerken. Sollten sich während der Konstruktion weitere Probleme ergeben, sind diese in zusätzlichen Tasks im GitLab festzuhalten.

  Nach der Erarbeitung des Feinentwurfs erfolgt die eigentliche Umsetzung der Aufgaben. Dabei wird für jeden Task iterativ zunächst festgelegt, ob und wie Unit-Tests möglich sind. Ein Unit-Test (auch Modul-Test) testet die Funktionalität innerhalb einzelner abgrenzbarer Teile der Software. Im Anschluss erfolgt die Umsetzung und Implementierung der Tasks. Das Ergebnis wird mit der „Definition of Done“ abgeglichen. Dabei ist zu berücksichtigen, ob der Code lauffähig, der Unit-Test erfolgreich und der Quellcode dokumentiert ist. Ist die „Definition of Done“ für alle Tasks einer User-Story erfüllt, sind die Integrationstests für diese zu entwickeln, durchzuführen und zu dokumentieren. Ein Integrationstest testet die Zusammenarbeit voneinander abhängigen Komponenten. Hier bei ergibt sich im Hinblick auf die anzufertigende Abschlussdokumentation, dass die beständig angepassten detaillierten Lösungen nicht im Entwurf beschrieben werden. Diese werden stattdessen in der Umsetzung beschrieben. Hierbei muss auch auf verworfene Lösungen eingegangen werden.

  Im Anschluss muss auch die User-Story mit der „Definition of Done“ abgeglichen werden. Um diese zu erfüllen, müssen die Akzeptanzkriterien erfüllt, der komplette Code lauffähig, alle Unit-Tests und Integrationstests erfolgreich und die Dokumentation in den Repositories vollständig sein. Im letzten Schritt der Konstruktionsphase müssen die Systemtests durchgeführt und dokumentiert werden.

– **Übergang**: In der letzten Phase eines Zyklus werden die Ergebnisse den Stakeholdern vorgestellt und durch diese abgenommen. Dies ist zu dokumentieren. Im Anschluss findet eine Retrospektive statt. Die Projektgruppe erarbeitet dabei, was

im aktuellen Zyklus gut und schlecht gelaufen ist. Aus den Ergebnissen werden konkrete Handlungen und Maßnahmen abgeleitet, welche die Zusammenarbeit im Folgezyklus verbessern sollen (Details zu Retrospektive siehe Abschnitt 2.2.2.

2.1.2 Projektplan intern und extern (Zyklen und Meilensteine)

Zyklen

Der interne Projektplan unterteilt sich in Zyklen. Insgesamt wurden drei Zyklen im Rahmen der Projektgruppe durchlaufen. Im Folgenden werden die drei Zyklen kurz vorgestellt:

- **Zyklus 1**: „Erster Durchstich“

  Im ersten Zyklus soll ein erster Durchstich geschaffen werden, um dadurch die Basis für die späteren Entwicklungen zu legen und den Projektgruppenmitgliedern einen tieferen Einstieg in die einzelnen zu implementierenden Komponenten zu geben. Ziel ist es dabei, zunächst Positionsdaten von der Zuse live zu empfangen, diese durch die einzelnen betroffenen Komponenten zu leiten und anschließend im Küstenleitstand darzustellen.

- **Zyklus 2**: „Fernsteuerung der Zuse“

  Im zweiten Zyklus soll es möglich sein die Zuse vom Küstenleitstand aus, sowie mit einer Backup-Steuerung, aus der Ferne steuern zu können.

- **Zyklus 3**: „Fertigstellung Fernsteuerung der Zuse und Routen abfahren“

  Die Umsetzung der Fernsteuerung der Zuse soll im dritten Zyklus abgeschlossen werden. Außerdem soll hier das automatische Abfahren von vorgegebenen Routen umgesetzt werden.

Meilensteine

Der von Seiten der Projektgruppenbetreuern vorgegebene Projektplan gliedert sich in drei Meilensteine. Ein Meilenstein umfasst dabei eine Projektphase, an deren Ende ein fest definiertes Ergebnis steht. Die Meilensteine orientieren sich dabei an der Klassifizierung von Autonomiestufen für das autonome Fahren, wie unter anderem bei der Bundesanstalt für Straßensicherheit dargestellt (vgl. [Gas12], S. 1). Im Folgenden werden diese drei Meilensteine kurz vorgestellt:

– **Meilenstein 1**: „Remote Control“

  Im ersten Meilenstein soll ein Schiff ferngesteuert werden können. Dafür müssen Steuerbefehle für Ruder und Motor an das Schiff geschickt werden können. Außerdem müssen die Daten der verschiedenen Sensoren auf dem Schiff vom zu entwickelnden Produkt empfangen und dargestellt werden können.

– **Meilenstein 2**: „Remote Action Planning“

  Im zweiten Meilenstein soll das Schiff eine vorgegebene Route, bestehend aus mehreren Wegpunkten, abfahren und dabei selbstständig manövrieren und den Kurs halten können. Das Abfahren der Route soll durch den Anwender überwacht werden können (Monitoring).

– **Meilenstein 3**: „Fully Autonomous“

  Im dritten Meilenstein soll das Schiff völlig autonom eine Route abfahren können und dabei Hindernisse wie andere Schiffe und Sandbänke automatisch erkennen und diesen ausweichen können. Dies ist mit Hilfe der Auswertung der Sensordaten sowie einer kamerabasierten Objekterkennung umzusetzen.

2.1.3 Rollenverteilung in der Projektgruppe

Für wiederkehrende und regelmäßig auftretende Aufgabenbereiche wurden auf Empfehlung der Projektgruppenbetreuer einige Rollen innerhalb der Projektgruppe fest bestimmten Personen zugeordnet. Auf diese Weise steht für diese Aufgaben immer ein fester Ansprechpartner zur Verfügung. Diese Rollen werden im Folgenden genannt und kurz beschrieben. Außerdem wird der Rolleninhaber genannt.

– **Projektleitung**: Der Projektleiter besitzt einen Überblick über den aktuellen Stand des Projektes. Er ist für die Motivation der Projektmitglieder verantwortlich und fungiert als Problemlöser. (Verantwortlicher: Sebastian Schmidt)

– **Teilprojektleiter**: Der Teilprojektleiter unterstützt den Projektleiter bei seinen Tätigkeiten und führt auf Teilgruppenebene dieselben Aufgaben aus. (Verantwortlicher: Verschiedene, wurde bei Bedarf durch Projektgruppe bestimmt)

– **Admin**: Zuständig für Verwaltung unter anderem der OFFIS-Server, Git-Repositories, Testing-Services. (Verantwortlicher: Adrian Jagusch)

- **Dokumentationsbeauftragter**: Pflege, Vorantreiben und Überprüfung der Dokumentation. (Verantwortlicher: Maximilian Wappler)
- **Inventarbeauftragter**: Ist zuständig für das eingesetzte Inventar. (Verantwortlich: Sergej Kidrowski)
- **Qualitätsbeauftragter**: Ansprechpartner bezüglich Codepflege und -qualität. (Verantwortlicher: Hendrik Kahlen)
- **Testbeauftragter**: Ansprechpartner bezüglich Testplanung, -erstellung und -durchführung. (Verantwortlich: Philipp Mudra)
- **Öffentlichkeitsbeauftragter**: Pflege einer Website und Erstellung von Postern (Verantwortlicher: Luca Gatterdam)
- **Raum-/Transportbeauftragter**: Verwaltet Termine und Räume. Lädt zu offiziellen und inoffiziellen Terminen ein. (Verantwortlicher: Mark Otten)
- **Projektplanung**: Aktualisieren und Überwachen bzw. Gegensteuern der Projektplanung (Verantwortlicher: Maximilian Wappler)

2.2 Methodik

In diesem Abschnitt soll auf einige Methoden, die bei der Projektarbeit der PG MATE eine zentrale Rolle gespielt haben, eingegangen werden. Neben einer allgemeinen Beschreibung der Methode wird dabei immer der Einsatz der Methode in der Projektgruppe beschrieben, begründet und kritisch reflektiert.

2.2.1 Testgetriebene Entwicklung

Testgetriebene Entwicklung (engl. Test-Driven-Development) ist eine Methode, die häufig im Rahmen der agilen Softwareentwicklung eingesetzt wird. Kernelement der Methode ist das konsequente Entwerfen von Software-Tests vor der Implementierung der zu testenden Software. Dabei geht es hauptsächlich um die Verwendung von Modultests (Unit-Tests), die auch das von der PG MATE verwendete V-Modell vorsieht. Wobei beim V-Modell Modultests parallel zur Implementierung geschrieben werden. Bei der testgetriebenen Entwicklung werden die Modultests geschrieben, bevor mit der Modulimplementierung begonnen wird. Ein Modultest muss daher, aufgrund der fehlenden Funktionalität, zuerst immer fehlschlagen. (vgl. [Wes06], S. 2 f.).

Für die Nutzung der testgetriebenen Entwicklung spricht die erhöhte Softwarequalität des Endprodukts. Dies ist sowohl in funktionaler als auch in struktureller Hinsicht möglich. Durch die erhöhte Softwarequalität ist das Produkt besser wartbar, was besonders für mögliche Folgeprojektgruppen, die auf den Ergebnissen der PG MATE aufbauen, wichtig ist.

Gegen die Nutzung der testgetriebenen Entwicklung spricht der zu Beginn höhere Aufwand bei der Entwicklung, der den Projektfortschritt hemmt. Entwickler, die bisher nicht mit der testgetriebenen Entwicklung gearbeitet haben, arbeiten vor allem am Anfang mit der Methode langsamer. Dieser Effekt verringert sich jedoch im Laufe der Zeit bzw. verschwindet ganz, wenn die Entwickler sich an die Methode gewöhnt haben. Außerdem funktioniert die testgetriebene Entwicklung nur, wenn die Methode auch von allen Beteiligten konsequent genutzt wird. Ein weiterer Nachteil ist, dass sich mit der testgetriebenen Entwicklung nicht alles testen lässt. Zum Beispiel können GUI-Anwendungen mit dieser Methode nicht ohne Weiteres getestet werden (vgl. [Wes06], S. 3-6).

Die Methode der testgetriebenen Entwicklung wird in der PG MATE letztendlich eingesetzt, da auch die Projektgruppenbetreuer statt Software-Quantität eine hohe -Qualität bevorzugen, welche sich gut mit der Methode erreichen lässt. Auch die Projektgruppenmitglieder haben ein Interesse an der Arbeit mit der Methode, da viele im Vorfeld noch nicht mit dieser gearbeitet haben, diese jedoch in der beruflichen Praxis stark verbreitet ist. Der Lerneffekt, der durch die Arbeit mit der Methode zu erwarten ist, spricht ebenfalls für die Nutzung der Methode.

2.2.2 Retrospektive

Bei der Retrospektive handelt es sich um eine Methode, die in der agilen Softwareentwicklung genutzt wird. Bei Scrum findet die Retrospektive am Ende eines Sprints statt (bei der PG MATE am Ende eines Zyklus). Das Team prüft in der Retrospektive die Arbeitsweise des zurückliegenden Zyklus. Dabei soll offen und ehrlich Kritik geäußert werden können. Auf diese Weise kann erarbeitet werden, was im zurückliegenden Zyklus gut war oder wo gegebenenfalls Verbesserungsbedarf vorhanden ist. Dabei werden zunächst ungefiltert alle Informationen und Eindrücke gesammelt. Im Anschluss werden diese analysiert und verdichtet. Die Analyse ist notwendig, um die tatsächliche Ursache eines Problems zu finden und nicht bloß Symptome zu bekämpfen. Im nächsten Schritt erarbeitet das Team Maßnahmen, um die Arbeitsweise für den nächsten Zyklus zu ver-

bessern. Diese sollten sinnvolle und realistische Schritte umfassen, die von allen Teammitgliedern akzeptiert werden können (vgl. [Glo16], S. 182 f.).

Die Retrospektive bietet die Möglichkeit zum konsequenten Aufdecken von Problemen und Verbessern der Arbeitsweise. Dadurch kann auch der Zusammenhalt des Teams gestärkt werden, da Konflikte nicht verschleppt werden und lange unausgesprochen „aufkochen“. Dies gelingt jedoch nur, wenn bei der Retrospektive eine entsprechende konstruktive Atmosphäre geschaffen wird. Ansonsten kann die Retrospektive auch zu offenen Konflikten führen. Zudem kostet die Planung und Durchführung einer Retrospektive Zeit (vgl. [Glo16], S. 182 f.).

In der PG MATE wird mit der Methode Retrospektive gearbeitet. Die Projektgruppe ist ein komplett neu zusammengestelltes Team, bei dem sich die Mitglieder zunächst noch nicht gut kennen. Anfangsschwierigkeiten können frühzeitig entgegengewirkt und das Arbeitsklima und die Zusammenarbeit verbessern werden.

2.2.3 Brainstorming

Brainstorming ist eine Methode aus dem Bereich der Kreativitätstechniken. Sie kann zur Sammlung von Ideen genutzt werden. Dabei werden zunächst alle Ideen von den Gruppenmitgliedern gesammelt, zum Beispiel an einem Whiteboard. Im Anschluss können diese sortiert und verdichtet werden. Während der ersten Phase sollte es keine „Totschlagargumente“ geben. Was von einem Gruppenmitglied vorgeschlagen wurde, wird zunächst auch notiert. Im Anschluss kann über die gesammelten Ideen gesprochen und diese detailliert ausgeführt werden (vgl. [WB16], S. 16 f.).

In der PG MATE wird diese Methode eingesetzt, um zum Beginn eines jeden Zyklus die anstehenden Aufgaben, die von der Projektgruppe im kommenden Zyklus umgesetzt werden sollen, zu sammeln. Auf Basis der Ergebnisse werden dann die User-Stories erarbeitet.

2.3 Infrastruktur und Werkzeuge

Im Folgenden wird näher auf die Projektmanagementinfrastruktur eingegangen. Mit der Infrastruktur werden die unterschiedlichen Termintypen und Projektartefakte beschrieben, welche während der Termine zum Einsatz kommen. Die Artefakte dokumentieren Prozesse, Fortschritte oder Ergebnisse und werden während der Meetings als Grundlage für Berichte oder Diskussionen verwendet. Um auch das Projektmanagement und die

Kommunikation in einem Projekt zu lenken, werden verschiedene Werkzeuge (Softwarelösungen) benutzt. Diese werden vorgestellt und erläutert.

2.3.1 Terminplanung

Im Rahmen der Projektgruppe fanden in der Regel zwei Sitzungen pro Woche statt. Diese Regelung wurde zum Beginn der Projektgruppe gemeinsam mit den Betreuern vereinbart, um eine stetige Kommunikation zwischen den Betreuern und der Projektgruppe zu gewährleisten. Das jeweils erste Treffen fand für interne Besprechungen innerhalb der Gruppe der Studierenden statt. Die zweite Sitzung wurde zusammen mit den Projektgruppenbetreuern abgehalten. Die Ergebnisse und Inhalte dieser Termine wurden jeweils in einem Protokoll festgehalten (siehe Absatz 2.3.3). Zusätzlich fanden regelmäßig weitere Termine mit unterschiedlichen Anwendungsfällen statt. Teilweise erfolgte die Terminfindung aufgrund der Anzahl an beteiligten Personen über die Web-Anwendung Doodle. Meist konnten die gemeinsamen Termine jedoch innerhalb einer Sitzung abgestimmt werden. Anschließend wurden die Termine als Termineinladung über den Mail-Verteiler versendet, um alle Stakeholder zu informieren.

Für die Termine wurden in Absprache mit Manuela Wüstefeld (Sekretärin der Abteilung Systemanalyse und -optimierung) Räumlichkeiten für die Treffen reserviert und der PG zur Verfügung gestellt. Für die Treffen mit den Betreuern wurde der Raum A02 3-334 genutzt. Er befindet sich in direkter Nähe zu den Büros der Betreuer Marius Brinkmann und Mohamed Abdelaal. Die Nähe zu den Büros ermöglicht kurze Wege für Gespräche vor und nach den Gruppentreffen. Zusätzlich zu dem Besprechungsraum im Universitätsgebäude wurde der Projektgruppe Zugang zu den Arbeitsräumen im Untergeschoss des OFFIS gewährt. Diese konnten zeitweise reserviert werden, um eine Überlastung der Räume durch mehrere Projektgruppen zu vermeiden. Günther Ehmen war der Ansprechpartner für die Arbeitsräume im OFFIS.

2.3.2 Termintypen

Wie bereits erläutert, muss bei den regelmäßig wöchentlich stattfindenden Sitzungen zwischen zwei Arten von Terminen unterschieden werden – den internen und den externen Treffen.

Interne Treffen

Das interne Treffen bezeichnet ein, in der Regel einmal pro Woche stattfindendes, Treffen zwischen allen an der Projektgruppe teilnehmenden Studierenden. Jedes interne Treffen wurde von einem Moderator geleitet, um zu gewährleisten, dass jeder Teilnehmer zu Wort kommt und Diskussionen den vorgesehenen Zeitraum einhalten. Ergebnisse und Informationen wurden jeweils in einem Protokoll festgehalten (siehe dazu Absatz 2.3.3). Die Rollen des Moderators und die des Protokollanten wechselten von Termin zu Termin reihum zwischen den Studierenden. Die Agenda eines Termins wurde vor den entsprechenden Treffen vom Moderator vorbereitet, sodass sich alle Beteiligten bereits im Vorfeld Gedanken zu den jeweiligen Themen machen konnten. Die festen Themenpunkte in jeder Sitzung waren:

- Besprechung der Agenda
- Weekly Update (siehe Absatz 2.3.3)
- Besprechung der Projektrisiken (siehe Absatz 2.3.3)
- Statusberichte (siehe Absatz 2.3.3)
- Weitere optionale aktuelle Agendapunkte
- Terminabschluss (siehe Absatz 2.3.3)

Externe Treffen

Als externes Treffen wird im Rahmen der Projektgruppe ein regelmäßiges Treffen mit den Betreuern verstanden, welches in der Regel einmal wöchentlich stattgefunden hat. Es unterscheidet sich prinzipiell nur durch die Anwesenheit der Betreuer und der leicht abgewandelten Agenda von den internen Treffen. Folgende Punkte standen fest auf der Agenda der externen Treffen:

- Besprechung der Agenda
- Weekly Update (siehe Absatz 2.3.3)
- Besprechung der Projektrisiken (siehe Absatz 2.3.3)
- Status des Projekts (Kurzfassung der Statusberichte, siehe Absatz 2.3.3)
- Weitere optionale aktuelle Agendapunkte

– Terminabschluss (siehe Absatz 2.3.3)

Bei den Projektrisiken wurde häufig über getroffene Entscheidungen diskutiert. Die Entscheidungsfindung und die Gründe für eine Entscheidung wurden dabei den Betreuern erläutert, um so eine hohe Transparenz gegenüber den Betreuern zu gewährleisten.

Testtermine

Bei den Testterminen wurde der aktuelle Stand des Produkts vor Ort in Wilhelmshaven unter Realbedingungen mit der Zuse getestet. Für die Testtermine wurden im Vorfeld Anforderungen erhoben und ein Testdrehbuch verfasst. Dieses wurde vierzehn Tage vor dem Test an die Betreuer übergeben, um die Umweltbedingungen für den Test zu beschreiben und einen wissenschaftlichen Aufbau durch das Testdrehbuch zu gewährleisten. So konnte sichergestellt werden, dass beim Test auch alle Anforderungen an das Testbed erfüllt waren. Detailliertere Informationen zum Ablauf der Tests sind im Abschnitt 8.4 zu finden.

Weitere Termine

Neben den bereits beschriebenen Termintypen fanden diverse weitere Termine statt. Zum Beispiel fanden Treffen von Teilgruppen oder Interviews mit bestimmten OFFIS-Mitarbeitern und direkten Stakeholder statt. Dies waren meistens Treffen, bei denen das Vorgehen bei der Entwicklung detaillierter besprochen wurde. Diese Termine dienten dazu, jederzeit die Anforderungen der entsprechenden Stakeholder zu berücksichtigen und den Fortschritt mitzuteilen.

2.3.3 Projektkommunikation

Zur Kommunikation innerhalb der Projektgruppe und auch in Richtung der Stakeholder standen verschiedene Kommunikationsmittel zur Verfügung, die im Folgenden kurz vorgestellt werden.

Protokolle

In den Protokollen wurden die Ergebnisse der einzelnen internen und externen Treffen festgehalten. Sie sind nach dem Schema eines Ergebnisprotokolls aufgebaut und enthalten ausschließlich die für diese Protokollart erforderlichen Informationen. Die Protokolle

folgen einer festen Struktur, die bei allen Protokollen übergeordnet vorgegeben ist. Der Protokollant wechselte wöchentlich und das Ergebnis wurde stets von einem anderen Projektgruppenteilnehmer sowie dem Projektleiter überprüft.

Weekly-Update

Das Weekly-Update war ein fester Bestandteil jedes internen und externen Treffens. Es ist angelehnt an agile Vorgehensmodelle, wie beispielsweise die Methode Daily-Scrum aus dem Vorgehensmodell Scrum (vgl. [Gol15], S. 9-10). Dabei stellen die Projektgruppenmitglieder kurz in wenigen Sätzen vor, welche Tätigkeiten sie seit dem letzten Treffen im Zusammenhang mit der Projektgruppenarbeit geschafft haben und woran sie als Nächstes arbeiten. Dabei werden auch aufgetretene Probleme erläutert und so allen Projektgruppenmitgliedern bekanntgemacht. Auf diese Weise erhalten alle Projektgruppenmitglieder einen aktuellen Überblick über die Gesamtsituation und können Rückfragen stellen oder Vorschläge unterbreiten. Somit können unter Umständen Probleme direkt angesprochen und mögliche Lösungen entwickelt werden.

Statusberichte

Nach der ersten Retrospektive hat die Projektgruppe beschlossen, die so genannten Statusberichte zu verfassen. Die Projektgruppe hat herausgefunden, dass ein zu großer Wissensdefizit zwischen Teilgruppenmitgliedern und den nicht am Teilprojekt beteiligten Mitgliedern bestand. Die Statusberichte sollten alle Projektgruppenmitglieder auf den aktuellen Stand bringen und somit Wissensdefizite minimieren. Es wurden zwischenzeitlich Untergruppen für Teilprojekte gebildet und nach Vollendung ihres Zwecks wieder aufgelöst. Die Statusberichte waren von den Teilgruppen wöchentlich zu erstellen und beim internen Treffen vorzustellen. Für das Verfassen war der Teilprojektleiter der jeweiligen Untergruppe verantwortlich. Dieser war dafür zuständig, vor den internen Treffen alle wichtigen Informationen in seiner Untergruppe zusammentragen und übersichtlich in Form eines Berichts aufzubereiten.

Die Statusberichte beinhalten unter anderem den aktuell geplanten Fertigstellungstermin der entsprechenden Teilgruppe im Rahmen des jeweiligen Zyklus, die Aufgaben, die in der vergangenen Woche erledigt wurden, die Aufgaben, welche aktuell bearbeitet werden, sowie zukünftig anstehenden Aufgaben. Besonderes Augenmerk wurde auf

die kritischen Tasks geworfen. Des Weiteren wurde unter internen und externen Risiken unterschieden.

Status des Projekts

Der Status des Projekts ist ein fester Punkt in der Agenda der externen Treffen mit den Betreuern. Dabei wurde eine verkürzte Form der Statusberichte durch die einzelnen Teilgruppen vorgestellt, um die Betreuer über den aktuellen Stand, die Aufgaben und die Herausforderungen dieser Teilgruppe zu informieren, ohne jedoch detailliert auf einzelne Tasks einzugehen. Insbesondere wurden in diesem Teil der Sitzung aufgetretende Fragen und Probleme mit den Betreuern besprochen und oft ein gemeinsamer Lösungsansatz entwickelt. Falls an dieser Stelle eine längere Diskussion absehbar war, wurde sie auf ein gesondertes Treffen mit der entsprechenden Untergruppe verschoben (siehe hierzu Absatz 2.3.2).

Projektrisikoliste

Um längerfristige Probleme zu dokumentieren, wurde eine Liste aller projektkritischen Risiken erstellt, die von allen Projektteilnehmern gemeinsam geführt wurde. Bei der Aufnahme und späteren Pflege der Risiken wurde während der wöchentlichen Termine beraten, wie das jeweilige Risiko beseitigt werden könnte. Wenn möglich wurden die Projektrisiken schnell beseitigt, da sie die Planbarkeit des Projekts oder dessen Fortschritt stark gefährden.

Die Liste wurde aus zwei Gründen eingeführt. Zum einen hatte nicht jeder Stakeholder einen Überblick über alle angefallenen Probleme, da diese nicht zentral dokumentiert wurden. Zum anderen konnte die Projektgruppe keine Abhilfe schaffen, wenn sie nicht wusste, dass ein Problem kritisch war und schnell gelöst werden musste. Diese Probleme hatten durch ihre Komplexität den Charakter, den Projektfortschritt zu stoppen und somit andere Aufgaben auf dem kritischen Pfad nach hinten zu verschieben.

Nr.	Beschreibung Projektrisiko	Problem	Abhilfe	Aufnahmedatum	Autor	Status/Datum geschlossen
1	Kommunikation mit Systemen	Das Regelungssystem empfängt keine Daten der Polymorphen Schnittstelle	Problem durch Tests finden und beheben.	01.10.2016	Herr Müller	Geschlossen am 03.05.2017

Tabelle 1: Beispiel eines fiktiven Projektrisikos in der Projektrisikoliste

Für die Umsetzung der Liste wurde eine Tabelle mit mehreren Spalten für verschiedene Attribute der Risiken gepflegt (siehe Tabelle 1). Ein Projektrisiko hat eine eindeutige Nummer, um das Risiko später identifizieren zu können. Des Weiteren sind die Projektrisiken einmal kurz zusammengefasst und wenn bereits ermittelt, ist ein entsprechender Lösungsvorschlag angegeben. Das Aufnahmedatum und ein Verfasser, welcher auch als Ansprechpartner fungiert, sorgen zusätzlich für eine bessere Rückverfolgbarkeit der Risiken.

Die Risiken wurden zusätzlich in zwei Kategorien aufgeteilt. Die internen Risiken sind Probleme, die innerhalb des Produktes der Projektgruppe aufgetaucht sind und von ihr ohne weitere Hilfe von außen gelöst werden konnten. Externe Risiken konnte die Projektgruppe nur bedingt oder nicht alleine bewältigen. Hierfür war die Hilfe der Betreuer erforderlich.

Terminabschluss

Die internen und externen Termine endeten immer mit einem Terminabschluss. Im Terminabschluss verliest der Protokollant noch einmal das Protokoll und die Terminteilnehmer haben die Möglichkeit Ergänzungen oder Richtigstellungen einzubringen. Des Weiteren werden zu diesem Zeitpunkt Termine geplant, die aus der Reihe während des Meetings vereinbart worden sind. Beispiele dafür sind ein Treffen zwischen einer Teilgruppe und einem der Betreuer, um das weitere Vorgehen zu besprechen oder ein Termin für einen Systemtest. Der Terminabschluss endet mit konstruktivem Feedback an den Moderator. Nach dem Terminabschluss endet der Termin offiziell.

2.3.4 Kommunikationswerkzeuge

Damit die Kommunikation innerhalb der Projektgruppe nicht nur einmal in der Woche zum internen Meeting stattfindet, greift die Projektgruppe auf verschiedene Kommunikationswerkzeuge zurück. Zu diesem Zweck wurden ein Git, ein Mail-Verteiler und mehrere Instant-Messenger-Gruppen eingerichtet. Je nach Bezugsgruppe und Dringlichkeit wurden diese Werkzeuge eingesetzt.

Git, GitHub und GitLab

- **Einsatz:** Git ist ein Versionsverwaltungsprogramm und ermöglicht es, verschiedene Versionen einer Datei über eine Versionierung eines Verzeichnisbaums zu erhal-

ten (vgl. [Cha14], S. 1). Des Weiteren ermöglicht Git es, verschiedene Versionen einer Datei zusammenzuführen (vgl. [Cha14], S. 31-33). Dies macht Versionsverwaltungsprogramme interessant für große Projekte, an denen viele Mitarbeiter an zum Teil gleichen Dokumenten arbeiten (vgl. [Cha14], S. 5 ff.). GitLab und GitHub sind dabei serverseitige Managementlösungen für ein Git-Repository (vgl. [Cha14], S. 131 ff.).

Zunächst wurde GitHub und im späteren Verlauf GitLab als Versionsverwaltungs- und Projektmanagement-Tool im Rahmen der Projektgruppe eingesetzt. Zusätzlich wurde die Projektdokumentation in Form von Dateien im Markdown-Format so angelegt, dass diese über einen Browser in der Web-App von GitLab betrachtet werden konnten. Diese Dokumentationsdateien wurden so eingepflegt, dass eine übersichtliche Dokumentation in direkter Nähe zum Quellcode (Lokalität) des Projekts erfolgte. Über eine Browser konnte so zu jeder Zeit der Fortschritt eingesehen werden.

GitLab gibt dem Anwender der Software die Möglichkeit, zu erledigende Aufgaben in Form von Issues zu generieren. Mit der Issue-Funktion konnten die Projektteilnehmer Tasks und User-Stories erstellen. Diese können mit Labels versehen und dem Label entsprechend auf einem Kanban-Board dargestellt werden. Das Kanban-Board ist ein weit verbreitetes Werkzeug, um große Projekte mittels einzelner Arbeitspakete übersichtlich darzustellen. GitLab bietet auch die Möglichkeit über Verknüpfungen unterschiedliche Projektartefakte miteinander zu verbinden. Beispielsweise kann in einem Issue eine Anforderung referenziert werden, welche in einem anderen Dokument aufgeführt wird. Dies verbessert die Nachvollziehbarkeit für den Leser. Für jede zu entwickelnde Komponente wurde eine eigene Teilgruppe gebildet. Jede Teilgruppe bekam im GitLab ein eigenes Repository, damit die verschiedenen Gruppen unabhängig voneinander arbeiten konnten.

– **Begründung:** Beim Auftakttreffen der Projektgruppe wurde vorgeschlagen ein Open-Source-Projekt zu erstellen. Dies trug maßgeblich zu der Entscheidung bei, GitHub als zentrales serverseitige Repository zu verwenden. GitHub bietet die Möglichkeit das Projekt öffentlich freizugeben, um interessierten Personen einen Blick auf das Projekt zu ermöglichen. Einer der Vorteile eines öffentlichen Repositories ist somit eine Referenz, welche die Reputation der Entwickler fördern kann. Leider zwang die Verwendung von urheberrechtlich geschütztem Code die Projektgruppe dazu, nach kurzer Zeit auf eine alternative Lösung zu wechseln. Hierfür bot

sich eine von den IT-Diensten der Universität Oldenburg verwaltete GitLab-Installation an. GitLab ist GitHub sehr ähnlich. Eine Migration zu GitLab konnte mit überschaubarem Aufwand durchgeführt werden.

Telegram-Gruppe

- **Einsatz:** Um einen schnellen unbürokratischen Austausch von Informationen zu realisieren, wurde der Instant-Messenger Telegram verwendet. Der Messenger unterstützt sowohl Gespräche von Person zu Person als auch in Gruppen. Die gesamte Projektgruppe sowie jede Teilgruppe hat auf diese Weise eine eigene Telegram-Gruppe erstellt. So gut wie alle Diskussionen außerhalb der regulären Meetings wurden über den Messenger abgehalten und konnten über die Historie im Nachhinein nachvollzogen werden.
- **Begründung:** Telegram besitzt im Gegensatz zu anderen Instant-Messengern die Möglichkeit die Historie für neu hinzugefügte Gruppenmitglieder freizugeben. So konnten Teilnehmer, welche später einer Teilgruppe beitraten, die bisherigen Ergebnisse der Gruppenkonversation einsehen. Ein weiter Vorteil von Telegram ist die Unabhängigkeit von Smartphones. Telegram unterstützt die gängigen Plattformen (Linux, Android, Windows, Windows Phone, macOS, iOS) und kann ohne eine Registrierung der Mobilfunknummer verwendet werden. Der größte Nachteil bei der Verwendung von Telegram ist, dass durch die Aufteilung in kleinere Teilgruppenchats Informationen nicht an alle Projektmitglieder gehen, was die Transparenz negativ beeinflusst. Des Weiteren ist ein Verlauf eines Chat-Protokolls in diesem Medium gefangen und kann schlecht rekapituliert werden, falls nach bestimmten Informationen gesucht wird.

Mail-Verteiler

- **Einsatz:** Die Mail-Adressen pg-mate@informatik.uni-oldenburg.de und später pg-mate@googlegroups.com wurden zur offiziellen Kommunikation zwischen den Betreuern der Projektgruppe und den Teilnehmern verwendet. Des Weiteren wurden die Einladungen zu den offiziellen Termine über den Verteiler realisiert.
- **Begründung:** Um einen nachvollziehbaren Gesprächsablauf garantieren zu können, bei dem jeder Teilnehmer der PG und die Betreuer auf dem aktuellen Stand bleiben, wird ein verlässliches Werkzeug benötigt. Ein Mail-Verteiler sorgt dafür,

dass kein Stakeholder bei der Informationsverteilung vergessen werden kann. Außerdem werden alle Mitglieder der Gruppe automatisch über Änderungen bei den Terminen auf dem aktuellen Stand gehalten.

Gantt-Diagramm und GanttProject

- **Einsatz:** Ein Gantt-Diagramm ist ein Werkzeug, um die Aufgaben in einem Projekt zu gliedern. Dabei werden die Projektaufgaben in Arbeitspakete unterteilt und in eine chronologische Beziehung zueinander gesetzt. Arbeitspakete bekommen dabei in der Regel Vorgänger und Nachfolger zugeordnet. Somit entsteht ein Graph, in dem alle Arbeitspakete enthalten sind. In den Graph werden auch Meilensteine eingepflegt, welche festgelegte Fortschritte während des Projekts widerspiegeln. Alle Arbeitspakete, die von einem Meilenstein hin zum nächsten Meilenstein erfüllt werden müssen, werden als Projektphase bezeichnet. Des Weiteren können Pakete verschiedene Eigenschaften, wie eine zeitliche Dauer, zugeordnet bekommen. So ist es möglich anhand der Dauer der Arbeitspakete einen kritischen Pfad und Pufferzeiten in dem Graphen zu identifizieren.

  Um das Gantt-Diagramm zu vervollständigen, werden den Arbeitspaketen Mitarbeiter (Ressourcen) zugeordnet. Somit kann der Projektleiter jederzeit die Verfügbarkeit seiner Mitarbeiter im Projekt überschauen und sehen, ob ein Arbeitspaket mehr Zeit benötigt oder ob sich das Projekt anhand des kritischen Pfades verschiebt.

  Für die Erstellung des Gantt-Diagramms wurde die Software GanttProject benutzt. Über dieses Werkzeug konnten auch erstmals die „Human Resources“ innerhalb der Projektgruppe verteilt werden, sodass hierfür eine Übersicht generiert wurde. Haupteinsatz war die Darstellung des kritischen Pfads bei den externen Treffen (siehe Absatz 2.3.2) des Projektes und die Verteilung der Ressourcen (vgl. [Hil05], S. 1 ff.).

- **Begründung:** Nach dem ersten Zyklus wurde beschlossen die Projektplanung weiter auszubauen. Neben den Statusberichten (siehe Absatz 2.3.3) wurde entschieden, ein Gantt-Diagramm zu erstellen. Das Werkzeug sollte für eine höhere Übersichtlichkeit und Transparenz, sowohl für die Projektgruppe als auch für die Betreuer, sorgen. Bis zur Einführung des Gantt-Diagramms war es nicht möglich, alle Aufgaben einer Projektphase chronologisch darzustellen.

Slack

- **Einsatz:** Slack ist ein Instant-Messenger, der speziell für Arbeitsgruppen und deren Bedürfnisse entwickelt wurde. Die Besonderheit von Slack ist eine Schnittstelle, welche es den Benutzern erlaubt, andere Dienste einzubinden. So war es der Projektgruppe möglich, die Kommunikationskanäle von E-Mail, GitLab und Telegram in Slack zu integrieren und damit zu bündeln. Benutzer müssen somit nur noch einen Posteingang prüfen, anstatt verschiedene Posteingänge verschiedener Werkzeuge zu prüfen.
- **Begründung:** Die Projektgruppe hat sich entschieden Slack zu verwenden, um den Kommunikationsaufwand zu verringern.

3 Maritime Grundlagen

In diesem Kapitel werden einige maritime Grundlagen gegeben, die für das Verständnis dieser Arbeit von hoher Bedeutung sind. Dieser Abschnitt kann von Lesern mit grundlegendem Vorwissen zu maritimen Standards und zu Schiffssensoren sowie Navigations--Assistenz-Systemen übersprungen werden. In diesem Kapitel werden in den einzelnen Abschnitten zudem jeweils nur die für die Ausarbeitung letztendlich auch relevanten und tatsächlich verwendeten Themen und nicht, zum Beispiel alle in der Praxis relevanten maritimen Standards vorgestellt.

3.1 Maritime Standards

In diesem Abschnitt werden die maritimen Standards NMEA0183, NMEA2000, RTZ und S-100 vorgestellt. Diese Standards spielen im Rahmen der Umsetzungsphase eine wichtige Rolle und werden für den Leser, der mit diesen Standards nicht vertraut ist, an dieser Stelle kurz erläutert. Dabei wird auf die Aufgabe, den Hintergrund und den Einsatz des jeweiligen Standards eingegangen. Detailliertere Informationen, zum Beispiel zum Inhalt einzelner Nachrichten, werden bei Bedarf in den jeweiligen Abschnitten, in denen diese Standards eine Rolle spielen, gegeben.

3.1.1 NMEA0183

Der National Marine Electronics Association 0183 (NMEA0183) Interface Standard definiert Anforderungen an elektrische Signale, an das Datenübertragungsprotokoll und an die Datenübertragungszeit. Außerdem spezifiziert er Satzformate für einen seriellen 4800-Baudrate Datenbus. Jeder Bus hat nur einen Sender, aber mehrere Empfänger. Dieser Standard wurde entwickelt, um die serielle unidirektionale Datenübertragung von einem einzigen Sender zu mehreren Empfängern zu unterstützen. Die Daten sind in ASCII-Zeichen auslesbar und enthalten Informationen wie Geschwindigkeit, Position und Tiefe (vgl. [ZZ12], S. 490).

Die NMEA hat den Standard 0183 definiert, um verschiedene Sensoren von unterschiedlichen Herstellern miteinander kommunizieren zu lassen. Im Idealfall werden die Daten nach dem RS-232-Standard (seriell) vom Sender ausgegeben. Es ist jedoch kein Stecker definiert, hier bleibt dem Hersteller freie Wahl, wodurch viele individuelle Stecker auf dem Markt verfügbar sind. Da es nur einen Sender pro Netz gibt, benötigt es

keiner Zuordnung von einzelnen Datenpaketen. Andererseits muss es ein Gerät geben, das mehrere Netze zusammenfassen kann, insofern mehrere Sensoren abgehört werden sollen (vgl. [Wei13], S. 301).

Die Daten werden im ASCII-Format in einzelnen separaten Nachrichten übertragen. Dabei sind alle druckbaren Zeichen, sowie Carriage-Return (CR, Wagenrücklauf) und Line-Feed (LF, Neue Zeile) erlaubt (vgl. [HWLW08], S. 450).

3.1.2 NMEA2000

National Marine Electronics Association 2000 (NMEA2000) ist ein serielles Datennetzwerk, welches mit 250 kbit pro Sekunde betrieben wird und den integrierten Schaltkreis des „DeviceNet“ benutzt. DeviceNet verwendet den Controller Area Network (CAN) Standard als Grundlage für ein Kommunikationsprotokoll auf höherer Ebene (vgl. [Nat02]). Abgelöst werden soll DeviceNet in naher Zukunft von der Ethernet-Lösung OneNet (vgl. [nme13b]). Viele Anwender in der maritimen Industrie vertrauen bislang noch auf NMEA0183. Der Hauptunterschied ist, abgesehen von der Betriebsgeschwindigkeit, dass NMEA0183 eine Schnittstelle und NMEA2000 ein Netzwerk ist.

Das NMEA2000 Netzwerk erlaubt es mehreren Elektrogeräten, sich auf einem einzigen Kanal miteinander zu verbinden, um Daten auszutauschen. Da es sich um ein Netzwerk handelt und weil mehrere Geräte Daten übermitteln können, ist eine umfassendere Menge an Regeln nötig, die das Verhalten der Netzwerk-Teilnehmer bestimmen. CAN bietet diese Regeln automatisch an, vor allem um den Zugang zum Netzwerk und die Packet-Übertragung zu kontrollieren. Des Weiteren spielt dies für die Fehlererkennung eine wichtige Rolle. Ähnlich wie NMEA0183 definiert auch NMEA2000 standardisierte Datenformate und -definitionen, verfügt aber gleichzeitig über ausführliche Regeln für das Netzwerk-Management, um Knotenpunkte zu identifizieren, Kommandos an Geräte zu senden und Daten abzufragen. Neben der größeren Menge an Kontrolle und Verflechtung, die angeboten wird, ersetzt NMEA2000 mit nur einem einzigen Kabel alle bestehenden (circa 50) NMEA0183 Verbindungsstücke und kann die Datenmenge von 50 bis 100 NMEA0183 Streams summiert verarbeiten. Das NMEA2000 Netzwerk kann bei einer Betriebsgeschwindigkeit von 250 kbit/s bis zu 200 Meter lang sein (vgl. [Com16] und [Bus16]).

3.1.3 S-100

Der S-100 Datenstandard wurde von der International Hydropgraphic Organization (IHO) entwickelt. S-100 stellt einen modernen hydrographischen Datenstandard dar, welcher ein großes Spektrum an hydrographischen und digitalen Datenquellen unterstützt. Er erfüllt dabei die ISO-19000 und soll den veralteten S-57 Standard ersetzen. S-100 definiert die Spezifikationen der abzubildenden Daten und ist damit präzise gesagt kein Daten-Standard, sondern ein Framework (vgl. [WG11], S. 2).

S-100 ist eine sprachliche Definition und dient als eine Reihe von Richtlinien zur Datenerstellung und zum Datenaustausch zwischen Systemen und Anwendern. Deshalb bietet S-100 mehrere gemeinsame Datenmodelle wie das General Feature Model (GFM), Schemata und Metadaten, sowie Kodierungsformate für Daten (vgl. [PP16] S. 302). S-100 stellt Beschreibungen zu wesentlichen Verfahren, dazu gehören unter anderem die Erstellung von Datenelementen und die Lieferung von Datensätzen, bereit. Zudem existieren Beschreibungen der Komponenten für die Produktspezifikationen und der für die Entwicklung notwendigen Verfahren (vgl. [Mik10], S. 6).

S-100 bietet eine gute Kompatibilität mit webbasierten Diensten und ermöglicht den Datenaustausch mit anderen Standards wie dem NATO DIGEST (vgl. [Roj09], S. 3). Den Weg den S-100 einschlägt, hat den Vorteil, dass neue Komponenten nicht isoliert entwickelt werden. Die IHO bietet mit dem Feature Concept Dictionary (FCD) ein Register für S-100, das als allgemeines Wörterbuch für Spezifikationen nach S-100 von Organisationen, Industrie, Akademien und andere Menschen, die sich mit der Entwicklung von eigenen Datenspezifikationen beschäftigen, genutzt werden kann. Für dieses Register eingereichte Unterlagen werden von der IHO geprüft und können im Anschluss von anderen genutzt werden (vgl. [Roj09], S. 7 f.).

3.1.4 RTZ

Das Route plan Exchange Format (RTZ) ist ein XML basierter maritimer Standard zum Austausch von Routen. RTZ ist in Verbindung mit Electronic Chart Display and Information System (ECDIS) in der IEC-Norm 61174 spezifiziert. Die International Electrotechnical Commission (IEC) ist eine internationale Normierungsorganisation für Elektrik und Elektrotechnik. Routen sind dabei standardisiert. Eine Route besteht aus einem Tupel mit mindestens zwei Wegpunkten. Jeder Wegpunkt enthält Informationen zum „Leg“ des vorherigen Wegpunktes (vgl. [CIR]). Unter „Leg“ wird dabei eine Etappe bzw. ein Abschnitt zwischen zwei Wegpunkten verstanden.

Eine Route in einem RTZ-XML kann aus mehreren Teilen bestehen. Der erste Abschnitt „RouteInfo“ enthält dabei zunächst Informationen, welche die gesamte Route betreffen. Dies könnten unter anderem der Name der Route und der aktuelle Status der Route sein. Der zweite Abschnitt „Waypoints“ beinhaltet die Beschreibungen der Wegpunkte und der jeweiligen „Legs“. Dies können unter anderem die Position und die Geschwindigkeit sein, mit der die Etappe gefahren werden soll. Zudem lässt sich ein Standard-Wegpunkt definieren, dessen Attribute für alle anderen Wegpunkte gelten, sofern die jeweiligen Attribute dort nicht anders belegt wurden. Im dritten und letzten Abschnitt „Schedule“ sind Informationen zum Zeitplan der Route vorhanden. Dies sind zum Beispiel Wartezeiten und Start- bzw. End-Zeitpunkte (vgl. [Ryd12], S. 15).

3.1.5 MMSI

Die Maritime Mobile Service Identity (MMSI), zu deutsch Rufnummer des mobilen Seefunkdienstes, ist eine neunstellige Nummer zur Identifikation von See- und Küstenfunkstellen. Sie ist weltweit gültig und wird in Deutschland von der Bundesnetzagentur (Außenstelle Hamburg) vergeben. Die ersten drei Ziffer stellen die Landeskennung dar, für Deutschland sind dies „211“ und in bestimmten Fällen „218“. Die folgenden sechs Ziffern kennzeichnen die Funkanlage eindeutig (vgl. [Deu15]).

3.2 Schiffssensoren und Navigations-Assistenz-Systeme

In diesem Abschnitt wird ein Überblick über Schiffssensoren und Navigations-Assistenz-Systeme gegeben, die im Rahmen der Projektgruppe MATE eine Rolle spielen können. Dabei wird darauf eingegangen, worum es sich bei den einzelnen Systemen und Sensoren handelt, wie diese grob funktionieren und wo diese eingesetzt werden.

3.2.1 Schiffssensoren

Ein Sensor lässt sich allgemein als technisches Bauteil, welches bestimmte physikalische/chemische Eigenschaften oder die Beschaffenheit seiner Umgebung qualitativ und quantitativ erfasst, definieren (vgl. [Rei16], S. 2 f.). An dieser Stelle soll unter einem Schiffssensor ein technisches Bauteil oder System verstanden werden, das im Rahmen der Schifffahrt wichtige Informationen aus seiner Umgebung aufnimmt und zur Verfügung stellt. Bei nahezu allen Sensoren ist zu beachten, dass bei diesen keine hundertprozentige Genauigkeit garantiert ist und sich die Sensoren unter anderem hinsichtlich

Datenqualität, Signalstärke, Bauart und Alter unterscheiden können. Deshalb wird dieses Merkmal nicht bei jedem Sensor explizit genannt.

DGPS

Eine Quelle von Sensordaten für die Positionsbestimmung ist das Differential Global Positioning System (DGPS) an Bord des Schiffes. Ein DGPS nutzt die Positionsangaben von einem Global Navigation Satellite System (GNSS).

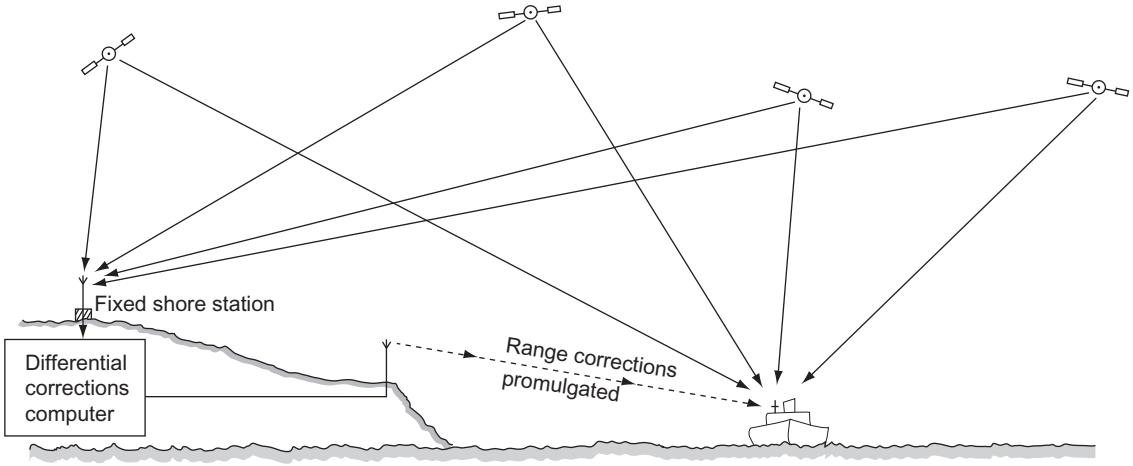


Abbildung 2: Aufbau eines DGPS-Systems in der Schifffahrt (vgl. [BWN14], S. 438)

Meist stammen diese Daten vom Global Positioning System (GPS). Grundsätzlich ist jedoch auch der Einsatz anderer Systeme wie Galileo (EU), GLONASS (Russland) oder Beidou (China) möglich. Die Positionsbestimmung eines Satellitennavigationssystems basiert auf der Tatsache, dass Signale von verschiedenen Satelliten, die unterschiedlich weit vom Empfänger entfernt sind, unterschiedlich viel Zeit zum Empfänger benötigen (Laufzeitdifferenzen).

Dabei wird in der Theorie angenommen, dass die Entfernungsdifferenzen proportional zu den Zeitdifferenzen sind. In der Praxis kommt es jedoch in der Iono- und Troposphäre zu Störungen und Abweichungen. Für räumlich benachbarte Empfänger sind diese Fehler gleich und lassen sich herausrechnen. Je größer der Abstand zwischen den Empfängern ist, desto weniger genau wird diese Fehlerkorrektur. In der Schifffahrt werden deshalb auf einem Schiff oft mehrere GPS-Empfänger genutzt, sowie zusätzliche sich meist an Land befindende Referenzstationen. Die exakte Lage dieser Referenzstationen wurde mithilfe klassischer Vermessungsmethoden bestimmt. Die genaue Position der Referenzstation ist

damit bekannt. Mit dieser und der empfangenen Position aus dem Satellitennavigationssystem errechnet die Referenzstation nun die Abweichungen. Die errechnete Differenz wird dann an die DGPS-Empfänger in der Umgebung übertragen (siehe Abbildung 2). Mit diesem Verfahren lässt sich die Genauigkeit der Positionsbestimmung per DGPS in der Schifffahrt deutlich verbessern (vgl. [BWN14], S. 437 f.).

Radar

Unter Radio detection and ranging (Radar) werden verschiedene Erkennungs- und Ortungsverfahren verstanden. Diese arbeiten auf Basis von elektromagnetischen Wellen im Radiofrequenzbereich (Funkwellen). In der Schifffahrt können auf diese Weise andere Schiffe oder Hindernisse in der erreichbaren Umgebung des Schiffes erkannt werden.

Die Technik hinter dem Radar-Gerät nutzt dabei die Reflexionen der Zielobjekte aus. Das Radar-Gerät (Sender) sendet Radiowellen gebündelt in seiner Umgebung aus (Primärsignal). Objekte in der Umgebung reflektieren diese Radiowellen und werfen sie zurück in Richtung des Senders (Sekundärsignal). Das Radar-Gerät kann die empfangenen Reflexionen (Echos) dann auswerten. Dabei können unter anderem folgende Informationen gewonnen werden: der Winkel bzw. die Richtung zum Objekt, die Entfernung zum Objekt und die Relativbewegung zwischen Sender und Objekt. Das Aneinanderreihen einzelner Messungen liefert sowohl die Wegstrecke, als auch die Absolutgeschwindigkeit des Objektes (vgl. [Bal14]).

Motor Control Unit

Die Motor Control Unit (auch Motorsteuerung) ist ein Steuergerät bzw. Teil der Motorelektronik, welche die Steuerung, Regelung und Überwachung von Motorfunktionen übernimmt. Die Motor Control Unit erhält somit als Sensor indirekt Daten unter anderem über Motor-Temperatur, CO2-Konzentration, Drehzahlen und Drehmomente, Öldruck, Strom, Batterieladung/Generator, sowie Fehlermeldungen (vgl. [Ise10], S. 39 f.).

Ruderlagenanzeiger

Im konkreten Beispiel der Zuse erhält die Motor Control Unit zudem Daten vom Ruderlagenanzeiger. Der Ruderlagenanzeiger zeigt den tatsächlichen Ausschlag des Ruders in Grad an (vgl. [Hou07], S. 180).

3.2.2 Navigations-Assistenz-Systeme

Unter Navigations-Assistenz-Systeme lassen sich verschiedene Systeme verstehen, welche bei der Navigation sowie bei der Routenfindung unterstützen sollen und in der Schifffahrt eingesetzt werden. Dieser Abschnitt befasst sich mit den Navigations-Assistenz-Systemen. Ausgehend vom elektronischen Navigationsinformationssystem ECDIS werden Systeme genannt, die an dieses System Daten liefern und gegebenenfalls auch eigenständig im Rahmen der Schiffsnavigation eingesetzt werden.

ECDIS

Das ECDIS ist ein elektronisches Navigationsinformationssystem und kombiniert dabei Daten aus unterschiedlichen Quellen. Dies ermöglicht es dem ECDIS auf einer Anzeige die Informationen zur Position und Umweltbeschaffenheit des Schiffs und Informationen zum Schiff selbst zusammen darzustellen. Die Datenquellen können dabei GNSS (in der Regel GPS bzw. in Küstennähe DGPS), Radar, Echolot, Seekarten bzw. Electronic Nautical Chart (ENC) und zusätzliche objektbezogene Informationen (zum Beispiel zu Schifffahrtszeichen) sein (vgl. [Kö17]). Durch die Zusammenführung der unterschiedlichen Datenquellen kann das System zusätzliche Informationen ermitteln und zum Beispiel Warnmeldungen ausgeben. Ist zum Beispiel der Tiefgang des Schiffes hinterlegt, kann ECDIS diese Informationen mit den Seekarten abgleichen und anhand der Route überprüfen, ob das Schiff diese Stelle passieren kann (bzw. darf, im Fall von Sperrgebieten) (vgl. [BWN14], S. 442-448).

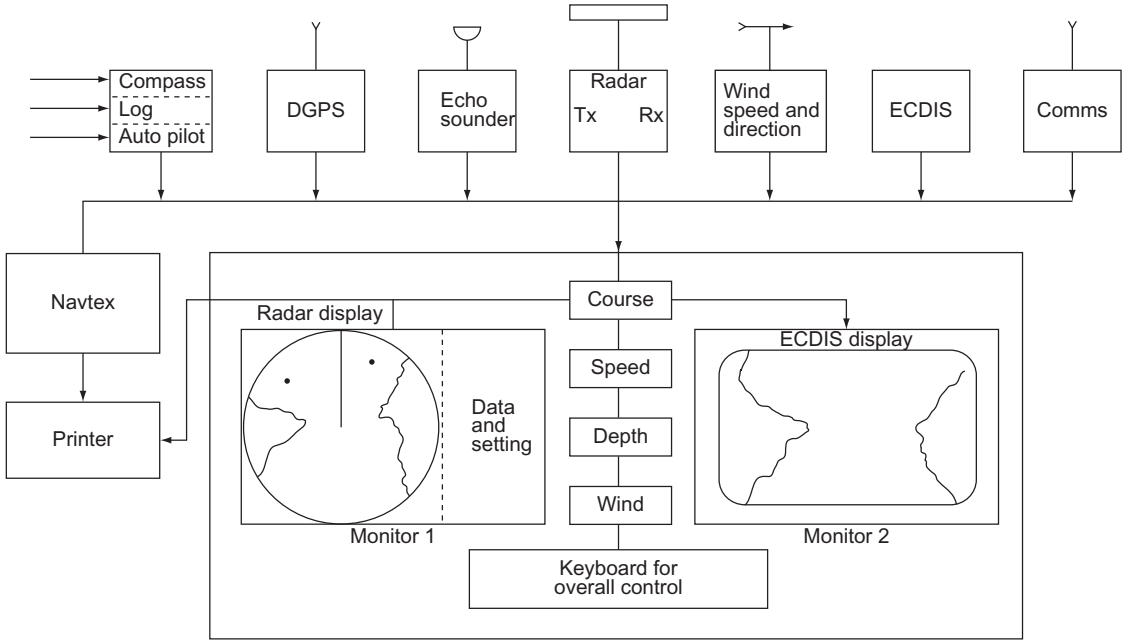


Abbildung 3: Typische integrierte Systemkonsole (mit ECDIS) ([BWN14], S. 450)

AIS

Das Automatic Identification System (AIS) ist ein Funksystem zum Austausch von Navigations- und anderen Schiffsdaten in der Schifffahrt. Das Ziel, was durch den Einsatz des AIS verfolgt wird, ist die Verbesserung der Sicherheit und Lenkung im Schiffsverkehr. Beim AIS werden über Funk verschiedene statische und dynamische (Reise-)Daten übertragen (vgl. [BWN14], S. 255). Diese können zum Beispiel sein (vgl. [BWN14], S. 259f.):

- Statisch: IMO-Nummer, Schiffsname, Rufzeichen, Schiffstyp (zum Beispiel Frachter), Abmessungen des Schiffs
- Dynamisch: Schiffsposition (Lat, Long, in WGS 84), Kurs über Grund (COG), Fahrt über Grund (SOG), Kursänderungsrate (ROT)
- Reisespezifisch: aktueller maximaler statischer Tiefgang, Gefahrgutklasse der Ladung, Reiseziel, geschätzte Ankunftszeit

Der AIS Standard der IMO definiert unterschiedliche Anlagenstandards. Die relevantesten sind an dieser Stelle Class-A und Class-B. Diese unterscheiden sich in ihrem Ein-

satzgebiet sowie in Signalstärken und Frequenzen. Class-A-Transceiver sind für die Berufsschifffahrt vorgesehen und ab bestimmten Abmessungen Pflicht. Class-B-Transceiver sind mit den Class-A-Transceivern grundsätzlich vergleichbar, aber sie übertragen mit einer niedrigeren Signalstärke und Melderate. Sie werden von nicht ausrüstungspflichtigen Schiffen zum Beispiel im Bereich der Fischerei verwendet (vgl. [BWN14], S. 258 f.).

Technisch funktioniert AIS als Self-Reporting-System mit lokaler Ausbreitung und im Zeitscheibenverfahren. Dies bedeutet konkret, dass das AIS abwechselnd auf zwei Kanälen im UKW-Seefunkbereich sendet. Das Senden erfolgt dabei in einem festen Zeitrahmen. Eine Minute wird in 4500 Zeitschlitze (Slots) unterteilt, die sich je zur Hälfte auf die beiden Kanäle aufteilen. Class-A-Transceivern stimmen die Slot-Belegung selbstständig mit den in Funkreichweite befindlichen anderen Class-A-Transceiver ab. Class-B-Transceiver verwenden die übrigen freien Zeitschlitze (vgl. [BWN14], S. 255-258).

Radar

Ach das Radar kann im Rahmen der Navigation und Routenplanung genutzt werden. Ähnlich wie DGPS-Systeme bieten Radar-Systeme meist zusätzliche Assistenz-Systeme und Tools, die zum Beispiel Echos als Schiffe identifizieren (Tracks), Fehler (Shadows) erkennen und diese nicht als Objekte darstellen oder darüber informieren, wenn sich das Schiff auf Kollisionskurs befindet. Am Radar selbst lassen sich meist einige Parameter einstellen, welche die Sensordaten beeinflussen. Dies sind zum Beispiel die Reichweite des Radars, die Deadzone (Bereich in dem das Radar „blind“ ist) oder FTC (Fehler durch Regen oder ähnliches herausrechnen). Die Parameter beeinflussen sich teilweise untereinander. Die optimalen Einstellungen dieser Parameter zu finden ist schwierig und orts-, wetter- und kontextabhängig. Einige Radar-Systeme können die Anpassung dieser Parameter selbst durchführen (vgl. [BWN14], S. 86). Einige der Assistenz-Systeme bei Radar-Anlagen sind:

– ARPA (auch MARPA): Automatic Radar Plotting Aid (ARPA) ist eine automatische Radar-Plotteinrichtung (oder eine Radar-Plotthilfe bzw. ein Radarbild-Auswertegerät). ARPA kann die mit dem Radar erkannten Objekte in der Umgebung als Schiffe (sogenannte Targets) identifizieren und verfolgen. Auf dem Kartenplotter werden dann der Kurs, die Richtung und die Geschwindigkeit angezeigt. Es gibt stark unterschiedlich leistungsfähige Systeme. Diese unterscheiden sich unter anderem in der manuellen und automatischen Peilung sowie beim Tracking der Schiffe

oder auch in der automatischen Anpassung der Radar-Parameter wie Reichweite oder FTC (vgl. [BWN14], S. 407 f.).

– EBL und VRM: Die Electronic Bearing Line (EBL) und der Variable Range Marker (VRM) sind zwei weitere Tools, die eingesetzt werden können, um zu ermitteln, wie weit andere Schiffe vom eigenen Schiff entfernt sind bzw. wie deren Kurs ist und ob sich beide Schiffe auf Kollisionskurs befinden (vgl. [BWN14], S. 297 f.)

4 Projektumfeld

In diesem Kapitel soll das Projekt bzw. die Projektgruppe in dem Umfeld, in dem die Durchführung stattfindet, eingeordnet werden. Dies umfasst zunächst organisatorische Aspekte bezüglich der Einordnung innerhalb der Universität Oldenburg. Zudem wird das Projekt das im Rahmen anderer Projekte stattfindet in diese eingeordnet und von diesen abgegrenzt. Außerdem werden die relevanten Schnittstellen und Systemgrenzen beschrieben.

Dabei wird darauf geachtet, nur das zu beschreiben, was von übergeordneter Bedeutung ist. Details die nur für Teile des Projekts relevant sind, werden in den jeweiligen Abschnitten beschrieben, in denen diese relevant sind. Der Grundgedanke ist dabei das Kapitel so zu strukturieren, dass es von einer mit dem Projektumfeld vertrauten Person übersprungen werden kann und die Grundlagen für die Personen liefert, die bisher mit dem Projektumfeld nicht vertraut ist.

4.1 Organisatorisch

Das Projekt findet innerhalb verschiedener organisatorischer Einheiten der „Carl von Ossietzky Universität Oldenburg“ (kurz Universität Oldenburg) statt. Die Abteilung „Systemanalyse und -optimierung“ (kurz SOA) des Departments für Informatik an der Universität Oldenburg wird geleitet von Prof. Dr.-Ing. Axel Hahn und apl. Prof. Dr.-Ing. Jürgen Sauer (vgl. [SOA17]).

Die Abteilung „Systemanalyse und -optimierung“ beschäftigt sich vor allem mit der Systemanalyse und Optimierung von maritimen Systemen und energieeffizienten Transport und Produktionssystemen. Die Abteilung „Systemanalyse und -optimierung“ arbeitet eng mit dem „Institut für Informatik OFFIS e.V.“ (kurz OFFIS) zusammen (vgl. [SOA17]).

Das OFFIS ist ein An-Institut der Universität Oldenburg. Die Schwerpunkte der Forschung und Entwicklung des OFFIS liegen in den Bereichen Energie, Gesundheit und Verkehr. Im Bereich Verkehr beschäftigt sich die Gruppe „Kooperierende Mobile Systeme“ (kurz KMS) in den Bereichen Automotive und Maritime mit der Entwicklung und Erprobung neuer Automatisierungssysteme. „Im maritimen Bereich entwickelt und betreibt die Gruppe KMS dazu die e-Maritime Integrated Reference Platform eMIR, mit der an der deutschen Nordseeküste und in Simulationsumgebungen neue maritime Sicherheits- und Navigationssysteme getestet werden können.“ ([OFF]).

Im Bereich der praxisnahen maritimen Forschung arbeiten das OFFIS und die Projektgruppe mit dem „Institut für Chemie und Biologie des Meeres“ (kurz ICBM) zusammen. Das ICBM „ist ein interdisziplinäres Forschungsinstitut, in dem grundlegende und angewandte Fragestellungen der Meeres- und Umweltforschung bearbeitet werden. Ziel des ICBM ist es, die Bedeutung mariner Umweltsysteme durch die Zusammenarbeit von Arbeitsgruppen verschiedener naturwissenschaftlicher Disziplinen (Chemie, Biologie, Physik, Modellierung) zu verstehen“ ([ICB17]). Das ICBM unterhält neben dem Universitätsstandort am Campus Wechloy eine weitere Außenstelle in Wilhelmshaven am Innenhafen. Dies ist auch der Heimathafen des Forschungsschiffs Zuse des OFFIS.

4.2 eMaritime Reference Platform

Die „eMIR“ ist eine Initiative der deutschen Schifffahrtsindustrie, um die Sicherheit und Effizienz im Transportwesen zu erhöhen sowie Schifffahrtsunfälle zu verhindern. Dies soll unter Berücksichtigung spezifischer Umweltaspekte geschehen (vgl. [eMI17]).

LABSKAUS und HAGGIS sind die zwei Forschungsschwerpunkte von eMIR. LABSKAUS ist die physische Testumgebung für neue Fortschritte und Technologien im Bereich eNavigation und eMaritime. LABSKAUS wird als Plattform zum Testen von Konzepten und Softwaresystemen genutzt, im Bereich Forschung und Entwicklung sowie zu Demonstrationszwecken. Technisch ist LABSKAUS als „Message passing system“ gedacht, das den Austausch von S-100-kompatiblen Daten ermöglicht. LABSKAUS ist mit der virtuellen Testumgebung HAGGIS verbunden (vgl. [eMI17]).

HAGGIS ist eine Plattform für die co-simulierte Verifizierung der Sicherheit und Effizienz von Transportsystemen. Auch HAGGIS arbeitet mit dem S-100 Datenmodell. HAGGIS wird unter anderem für die Untersuchung von selten auftretenden Zwischenfällen genutzt (vgl. [eMI17]).

Die PG MATE befasst sich mit der Entwicklung einer Plattform zur Erweiterung der LABSKAUS Infrastruktur (siehe Abschnitt 1.4).

Das Forschungsboot „Zuse“ wird im Rahmen der maritimen Forschung an der Universität Oldenburg und vom OFFIS eingesetzt. Dort findet es im Rahmen von eMIR Verwendung und wird in der praktischen Erprobung von Forschungsprojekten eingesetzt.

5 Anforderungsanalyse

Das vorliegende Kapitel Anforderungsanalyse bietet einen lösungsneutralen Überblick über die Anforderungen, die an das zu entwickelnde Produkt der PG MATE gestellt werden. Dazu werden zunächst auf Basis der drei Meilensteine (siehe Abschnitt 2.1.2) die im Rahmen des Projekt Kick-Offs vorgestellt wurden, Anwendungsfälle ermittelt. Diese werden im weiteren Verlauf auf atomare funktionale Anforderungen nach fest definierten Anforderungsschablonen abgebildet.

Neben den Anforderungen, die auf Anwendungsfälle zurückzuführen sind, werden weitere allgemeingültige funktionale und nichtfunktionale Anforderungen ermittelt, die an das Produkt der PG MATE gestellt werden.

5.1 Stakeholderanalyse

Anforderungen an das Produkt der PG MATE können von unterschiedlichen Stellen erhoben werden. Eine Quelle für Anforderungen stellen Stakeholder dar, die wie folgt definiert werden:

> „Ein Stakeholder eines Systems ist eine Person oder Organisation, welche (direkt oder indirekt) Einfluss auf die Anforderungen des betrachteten Systems hat.“ ([RP15], S. 77)

Neben den Anforderungen können Stakeholder direkt oder indirekt Informationen für Ziele und Randbedingungen an das System liefern.

In der Abbildung 4 sind die unterschiedlichen externen sowie internen Stakeholder abgebildet. Diese werden im weiteren Verlauf dieses Abschnitts genauer erläutert.

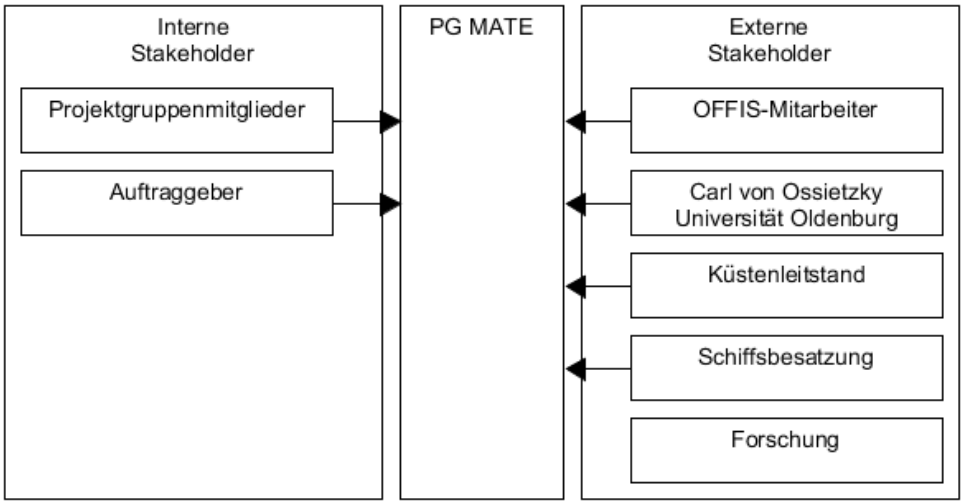


Abbildung 4: Interne und externe Stakeholder der PG MATE

Bei den Stakeholdern wird zwischen internen und externen Stakeholdern unterschieden. Interne Stakeholder bilden Personengruppen, die direkt an dem Projekt beteiligt sind. Entsprechend sind externe Stakeholder Personengruppen und Institutionen, die nicht direkt an dem Projekt beteiligt sind.

Im Folgenden wird näher auf die abgebildeten Stakeholder eingegangen, sowie ihre jeweilige Motivation und Rolle im Projekt erläutert.

Projektgruppenmitglieder: Die Projektgruppenmitglieder der PG MATE sind primär für die Konzeptionierung und Entwicklung des Produkts sowie für die Durchführung von Tests zuständig. Diese nehmen an der PG MATE teil, da dies einen essenziellen Teil der vertretenen Masterstudiengänge Wirtschaftsinformatik und Informatik darstellt. Die Projektgruppenmitglieder haben ein besonderes Interesse daran sich im Rahmen des Projektes persönlich weiterzuentwickeln. Des Weiteren stellen die Ergebnisse des Projektes eine Referenz für die weitere Laufbahn der Studierenden in der freien Wirtschaft und im Forschungsbereich dar.

Auftraggeber: Die Auftraggeber bestehen aus den Professoren apl. Prof. Dr.-Ing. habil. Jürgen Sauer und Prof. Dr.-Ing. Axel Hahn der Carl von Ossietzky Universität Oldenburg Department für Informatik in der Abteilung Wirtschaftsinformatik. Des Weiteren sind die wissenschaftlichen Mitarbeiter Marius Brinkmann und Mohamed Abdelaal unter den Auftraggebern vorhanden.

Diese dienen als beauftragende sowie beaufsichtigende Personen im Rahmen der Projektdurchführung. Ihr Interesse besteht zum einen darin die Ergebnisse des Projektes für eigene Forschungsaktivitäten zu nutzen und zum anderen die Ergebnisse als Basis für die Durchführung weiterer Projektgruppen zu verwenden.

OFFIS-Mitarbeiter: Die Mitarbeiter des OFFIS - Institut für Informatik unterstützen die Projektmitglieder bei der Konzeptionierung und Entwicklung des Produktes der PG MATE sowie bei der Durchführung von verschiedenartigen Tests. Die Systemtests (siehe Abschnitt 8.4) werden im Testbed des OFFIS ausgeführt. Die OFFIS-Mitarbeiter arbeiten im Bereich der Forschung eng mit der Carl von Ossietzky Universität Oldenburg zusammen und haben ähnliche Interessen an der Durchführung des Projektes, die auf den maritimen Forschungsbereich abzielen.

Carl von Ossietzky Universität Oldenburg: Das Projekt der Projektgruppe wird an der Carl von Ossietzky Universität Oldenburg durchgeführt, welches mit dem Projekt die Forschung im maritimen Bereich fokussiert. Dabei gibt die Carl von Ossietzky Universität Oldenburg das angestrebte Forschungsziel vor, stellt alle notwendigen Betriebsmittel zur Verfügung und kann die Ergebnisse des Projektes für weitere Forschung im maritimen Bereich verwenden.

Küstenleitstand: In einem realen Einsatzszenario wird das Produkt der PG MATE in einer küstenseitigen Leitwarte verwendet, um Schiffsbewegungen vorzugeben, die Abfahrt der Route nachzuverfolgen, ein Monitoring der Schiffsdaten durchzuführen und in Notsituationen eingreifen zu können.

Schiffsbesatzung: Neben der genannten küstenseitigen Leitwarte befindet sich schiffsseitig die Besatzung. Diese haben ebenfalls ein Interesse daran, Routen nachzuverfolgen sowie ein Monitoring durchzuführen.

Forschung: Da es sich bei der Umsetzung des Projektes um ein Forschungsprojekt handelt, haben alle Forschungsinteressierten im Bereich der Wissenschaft ein Interesse an der Durchführung sowie den Ergebnissen des Projektes. Auf den Ergebnissen aufbauend können weitere Forschungsarbeiten durchgeführt werden. Forschungsbereiche können beispielsweise im maritimen Umfeld stattfinden sowie im Bereich des autonomen Fahrens oder der Verkehrssicherheit.

5.2 Anwendungsfälle

Anwendungsfälle, die im Englischen als Use-Cases bezeichnet werden, dienen der graphischen oder textuellen Beschreibung von Anforderungen. Dabei beschreiben Anwendungsfälle die Art und Weise, wie ein Akteur mit dem zu erstellenden System interagieren kann. Aufgrund dieser Eigenschaft dienen Anwendungsfälle als Beschreibung für das äußerlich sichtbare Systemverhalten (vgl. [FGG+12], S. 158). Rupp spezifiziert Akteure als eine Rolle, die einen Benutzer, Benutzergruppen oder andere Systeme widerspiegeln und mit einem Subjekt interagieren ([RQ12], S. 251 f.). Das Subjekt stellt im Kontext dieses Projekts das Produkt der PG MATE dar.

Das Ziel bei der Erstellung von Anwendungsfällen ist festzuhalten, was ein System leisten soll, und nicht, wie dies erfolgen soll. Anwendungsfälle sind daher als lösungsneutral anzusehen.

In diesem Abschnitt werden, basierend auf den in Abschnitt 2.1.2 beschriebenen Meilensteinen, Anwendungsfälle identifiziert und in Form von Anwendungsfalldiagrammen sowie Anwendungsfallbeschreibungen veranschaulicht. In den nachfolgenden Tabellen 2, 3 und 4 werden die Meilensteine aufgegriffen, beschrieben und Anwendungsfälle identifiziert.

Meilenstein	Remote Control
Kurzbeschreibung	Der Meilenstein Remote Control beinhaltet die direkte Steuerung des Forschungsschiffs Zuse, sowie die küstenseitige Erfassung relevanter Sensordaten.
Anwendungsfall 1	Verbindung aufbauen
Anwendungsfall 2	Schiff manuell steuern
Anwendungsfall 3	Schiff in Notsituation steuern
Anwendungsfall 4	Steuerdaten senden
Anwendungsfall 5	Sensordaten abrufen
Anwendungsfall 6	Schiffssensordaten darstellen
Anwendungsfall 7	Verbindung abbauen

Tabelle 2: Anwendungsfälle des Meilensteins Remote Control

Meilenstein	Action Planning
Kurzbeschreibung	Der Meilenstein Action Planning beinhaltet die küstenseitige Routenplanung sowie ein Monitoring des Schiffs. Dieses umfasst sowohl relevante Sensordaten als auch die Position des Schiffs. Des Weiteren wird das Schiff befähigt, Wegpunkte der Route zu befahren und dabei den Kurs einzuhalten.
Anwendungsfall 8	Route erstellen
Anwendungsfall 9	Route senden
Anwendungsfall 10	Route bearbeiten
Anwendungsfall 11	Autopilot aktivieren
Anwendungsfall 12	Autopilot deaktivieren
Anwendungsfall 13	Schiffsposition darstellen
Anwendungsfall 14	Schiffsposition abrufen

Tabelle 3: Anwendungsfälle des Meilensteins Remote Action Planning

Meilenstein	Fully Autonomous Driving
Kurzbeschreibung	Der Meilenstein Fully Autonomous Driving beinhaltet neben den Funktionalitäten der vorherigen Meilensteine die bewusste Wahrnehmung von definierten Situationen über Sensoren und Kameras. Des Weiteren ist ein autonomes Abfahren einer Route unter Berücksichtigung des Reiseziels und die Reaktion auf die Umwelt des Schiffs vorgesehen.
Anwendungsfall 15	Autonomes Fahren aktivieren
Anwendungsfall 16	Autonomes Fahren deaktivieren
Anwendungsfall 17	Route anpassen
Anwendungsfall 18	Geschwindigkeit anpassen
Anwendungsfall 19	Umwelt wahrnehmen
Anwendungsfall 20	Auf Umwelt reagieren

Tabelle 4: Anwendungsfälle des Meilensteins Fully Autonomous Driving

Im weiteren Verlauf dieses Kapitels wird in grafischer und textueller Form auf die jeweiligen Anwendungsfälle der aufgeführten Meilensteine eingegangen. Diese verdeut-

lichen zunächst die Interaktion unterschiedlicher Akteure mit dem Produkt der PG MATE und den identifizierten Anwendungsfällen.

Die in Abschnitt 5.1 beschriebenen Stakeholder, Küstenleitstand und Schiffsbesatzung, stellen gleichzeitig Akteure des Produktes der PG MATE dar. Darüber hinaus existieren die Akteure Zuse, das ein Forschungsschiff darstellt und für die Durchführung des Projektes eingesetzt wird, sowie externe Systeme. Externe Systeme stellen beispielsweise AIS-Empfangsstationen dar, welche genutzt werden, um unter anderem Positionsdaten der Zuse und weiteren Schiffen in bestimmten Gebieten abzurufen.

5.2.1 Erster Meilenstein

Der erste Meilenstein besteht aus acht Anwendungsfällen, die in Abbildung 5 dargestellt werden. Darin sind, auf einer zunächst abstrakten Ebene, die Interaktionen der unterschiedlichen Akteure mit dem Produkt der PG MATE einsehbar. Es wird deutlich, welcher Akteur auf die unterschiedlichen Anwendungsfälle einwirkt. Des Weiteren wird dargestellt, wie die Anwendungsfälle untereinander in Beziehung stehen.

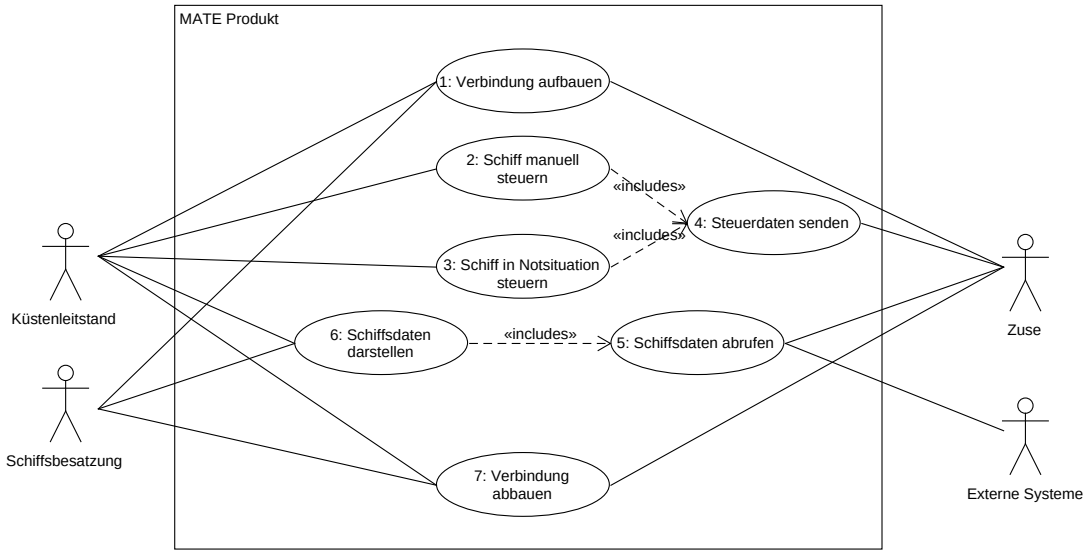


Abbildung 5: Anwendungsfälle des ersten Meilensteins

Im Folgenden werden die dargestellten Anwendungsfälle aufgegriffen und erläutert.

1. Verbindung aufbauen: Um Befehle an die Zuse zu senden und Daten von der Zuse zu empfangen, ist es notwendig, dass eine Verbindung zwischen dem Produkt der

PG MATE und der Zuse aufgebaut werden kann. Dieser Anwendungsfall stellt die Basis für jegliche Interaktion mit der Zuse dar und ist für nahezu alle Folgeanwendungsfälle relevant. Kann keine Verbindung zur Zuse aufgebaut werden, kann keine Interaktion mit der Zuse stattfinden. Darüber hinaus muss die Schiffsbesatzung eine Verbindung zur Zuse aufbauen, um Schiffsdaten abrufen zu können.

Kurzbeschreibung	Verbindungsaufbau vom Küstenleitstand zur Zuse.
Akteure	Küstenleitstand, Zuse, Schiffsbesatzung
Auslöser	Küstenleitstand, Schiffsbesatzung
Vorbedingungen	Verbindungsdaten sind bekannt. Beteiligte Akteure besitzen eine aktive Internetverbindung.
Ergebnis	Verbindung ist aufgebaut.
Ablauf	(1a) Küstenstand gibt Verbindungsdaten in der Benutzeroberfläche des MATE Produkts ein. (1b) Schiffsbesatzung gibt Verbindungsdaten in der Benutzeroberfläche des MATE Produkts ein. (2) Das MATE Produkt verbindet sich mit der Zuse. (3a) Verbindung ist erfolgreich aufgebaut. (3b) Verbindung kann nicht aufgebaut werden.

Tabelle 5: Anwendungsfall 1: Verbindung aufbauen

2. Schiff manuell steuern: Um die Zuse manuell steuern zu können, ist eine aktive Verbindung zur Zuse notwendig. Ist dies sichergestellt, kann der Küstenleitstand manuell Steuerdaten über das MATE Produkt an die Zuse weiterleiten. Steuerdaten bestehen aus den Attributen Geschwindigkeit sowie Ruderstellung. Sobald die Zuse diese Daten erhält, wird das Ruder auf Basis der Steuerdaten positioniert oder die Drehzahl des Motors erhöht bzw. verringert.

Kurzbeschreibung	Küstenleitstand sendet bei bestehender Verbindung Steuerdaten zur Zuse.
Akteure	Küstenleitstand, Zuse
Auslöser	Küstenleitstand
Vorbedingungen	Verbindung ist aufgebaut. Zuse ist in Sichtweite.
Ergebnis	Zuse führt Steuerung des Ruders bzw. des Motors auf Basis der Steuerdaten aus.
Ablauf	(1a) Küstenleitstand gibt Ruderstellung an über eine Benutzeroberfläche. (1b) Küstenleitstand gibt Geschwindigkeit an über eine Benutzeroberfläche.

Tabelle 6: Anwendungsfall 2: Schiff manuell steuern

3. Schiff in Notsituation steuern: Im Vergleich zu dem Anwendungsfall 2 Schiff manuell steuern, beinhaltet dieser Anwendungsfall den schnellen Eingriff und die Übernahme der Steuerung über einen sekundären Kanal. Dies ist notwendig, falls die primäre Steuerung nicht wie gewünscht ausgeführt wird bzw. eine Situation eintrifft, die eine schnelle Reaktion des Küstenleitstands auf die Steuerung des Schiffes erfordert. Dieser Anwendungsfall wird ebenfalls im Rahmen der Weiterentwicklung verwendet und beugt Schäden an der Zuse, Menschen sowie der Umgebung vor.

Kurzbeschreibung	Verhält sich die Steuerung nicht wie gewünscht, es tritt eine Notfallsituation ein oder das Schiff ist über den primären Kanal nicht erreichbar, so wird das Schiff manuell über einen sekundären Kanal gesteuert.
Akteure	Küstenleitstand, Zuse
Auslöser	Küstenleitstand
Vorbedingungen	Verbindung über den sekundären Kanal ist aufgebaut und die Zuse ist in Sichtweite.
Ergebnis	Zuse führt Steuerung des Ruders bzw. des Motors auf Basis der Steuerdaten über den sekundären Kanal aus. Steuerdaten des primären Kanals werden nicht von der Zuse ausgeführt.
Ablauf	(1) Küstenleitstand aktiviert sekundären Kommunikationskanal zur Zuse. (2a) Küstenleitstand gibt Ruderstellung an. (2b) Küstenleitstand gibt Geschwindigkeit an.

Tabelle 7: Anwendungsfall 3: Schiff in Notsituation steuern

4. Steuerdaten senden: Die Steuerbefehle, die aus Anwendungsfall 2 und Anwendungsfall 3 resultieren, müssen an die Zuse weitergeleitet werden. Die Steuerbefehle bestehen aus den Daten Ruderwinkel und Geschwindigkeit und müssen gegebenenfalls zyklisch an das Schiff gesendet werden, um dem Verlust von Steuerbefehlen vorzubeugen.

Kurzbeschreibung	Damit das Schiff gesteuert werden kann, müssen Steuerbefehle in Form von Ruderwinkel und Geschwindigkeit an das Schiff weitergeleitet werden.
Akteure	Küstenleitstand, Zuse
Auslöser	Küstenleitstand
Vorbedingungen	Es besteht eine Verbindung zur Zuse. Steuerbefehle werden sind vorhanden und syntaktisch korrekt.
Ergebnis	Zuse empfängt die gesendeten Steuerdaten und reguliert Aktuatoren gemäß dieser.
Ablauf	(1) Küstenleitstand sendet Steuerdaten an die Zuse. (2) Zuse empfängt die Steuerdaten. (3a) Zuse setzt die Ruderstellung auf den empfangenen Wert. (3b) Zuse setzt die Geschwindigkeit auf den empfangenen Wert.

Tabelle 8: Anwendungsfall 4: Steuerdaten senden

5. Sensordaten abrufen: Um eine permanente Kontrolle der Zuse zu ermöglichen, muss das MATE Produkt die Zuse über unterschiedliche Parameter überwachen können. Dazu ruft das MATE Produkt Sensordaten der Zuse sowie von externen Systemen ab.

Kurzbeschreibung	Sensordaten werden für weitere Aktionen von unterschiedlichen Akteuren abgerufen.
Akteure	Externe Systeme, Zuse
Auslöser	MATE Produkt
Vorbedingungen	Es besteht eine Verbindung zur Zuse und zu externen Systemen.
Ergebnis	Sensordaten liegen vor und können weiter verarbeitet werden.
Ablauf	(1a) Das MATE Produkt ruft Sensordaten von der Zuse ab. (1b) Das MATE Produkt ruft Sensordaten von externen Systemen ab. (2) Das MATE Produkt hält die Sensordaten für die weitere Verarbeitung vor.

Tabelle 9: Anwendungsfall 5: Sensordaten abrufen

6. Schiffssensordaten darstellen: Um eine permanente Kontrolle der Zuse zu ermöglichen, müssen Küstenleitstand und die Schiffsbesatzung ein Monitoring der Zuse

vornehmen können. Dies erfolgt über die Darstellung von allen relevanten Sensordaten über eine Benutzeroberfläche. Auf Basis dieser Informationen können die Akteure einen ordnungsgemäßen Betrieb überwachen oder bei Abweichungen entsprechend eingreifen. Die Sensordaten können direkt von den an der Zuse verbauten Sensoren stammen oder von externen Systemen zur Verfügung gestellt werden.

Kurzbeschreibung	Zur Überwachung eines ordnungsgemäßen Betriebs müssen die Akteure über die Sensordaten der Zuse informiert werden.
Akteure	Küstenleitstand, Schiffsbesatzung, Zuse, Externe Systeme
Auslöser	Zuse, Externe Systeme
Vorbedingungen	Es besteht eine Verbindung zwischen allen Akteuren und dem MATE Produkt. Sensordaten sind von der Zuse und externen Systemen abgerufen.
Ergebnis	Sensordaten werden dem Küstenleitstand und der Schiffsbesatzung dargestellt.
Ablauf	(1) Dem MATE Produkt liegen Sensordaten vor. (2a) Das MATE Produkt stellt dem Küstenleitstand Sensordaten dar und aktualisiert diese laufend. (2b) Das MATE Produkt stellt der Schiffsbesatzung Sensordaten dar und aktualisiert diese laufend.

Tabelle 10: Anwendungsfall 6: Schiffssensordaten darstellen

7. Verbindung abbauen: Hat die Zuse das vom Küstenleitstand verfolgte Ziel erreicht, ist keine Steuerung mehr notwendig und die Verbindung vom Küstenleitstand und der Schiffsbesatzung zur Zuse kann abgebaut werden.

Kurzbeschreibung	Falls keine Notwendigkeit mehr besteht die Zuse zu steuern, kann die Verbindung zu dieser abgebaut werden.
Akteure	Küstenleitstand, Schiffsbesatzung
Auslöser	Küstenleitstand, Schiffsbesatzung
Vorbedingungen	Verbindung ist aufgebaut. Es sind keine weiteren Steuermaßnahmen der Zuse notwendig.
Ergebnis	Verbindung ist abgebaut und es kann keine Steuerung mehr erfolgen.
Ablauf	(1a) Küstenleitstand trennt die Verbindung zur Zuse. (2b) Schiffsbesatzung trennt Verbindung zur Zuse.

Tabelle 11: Anwendungsfall 7: Verbindung abbauen

5.2.2 Zweiter Meilenstein

Der zweite Meilenstein besteht aus sieben Anwendungsfällen, die in Abbildung 6 in einem Anwendungsfalldiagramm dargestellt sind. Darin sind die Akteure Küstenleitstand, Schiffsbesatzung, Zuse sowie externe Systeme gegeben, welche auf die unterschiedlichen Anwendungsfälle zugreifen. Diese weisen teilweise Abhängigkeiten untereinander auf.

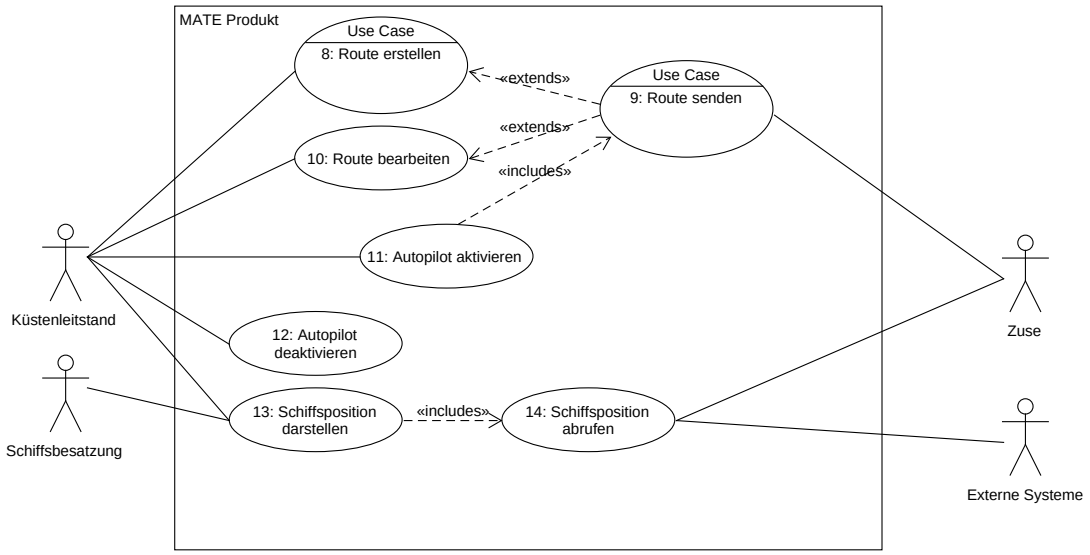


Abbildung 6: Anwendungsfälle des zweiten Meilensteins

Im Folgenden werden die dargestellten Anwendungsfälle aufgegriffen und erläutert.

8. Route erstellen: Gemäß der Beschreibung des zweiten Meilensteins kann neben der direkten Steuerung der Zuse vom Küstenleitstand eine Route erstellt werden. Diese Route kann die Zuse gemäß Anwendungsfall 14 eigenständig abfahren. Dabei sind keine direkten Steuerbefehle vom Küstenleitstand notwendig. Die Erstellung der Route erfolgt über eine Benutzeroberfläche, in die der Küstenleitstand 2 bis n Wegpunkte einer Route einträgt. Es können unterschiedliche Routen bestehend aus einer variierenden Anzahl von Wegpunkten erstellt werden. Die Wegpunkte werden vom Küstenleitstand in eine Seekarte eingetragen und über Längen- und Breitengrade bestimmt.

Kurzbeschreibung	Der Küstenleitstand kann über das Setzen von Wegpunkten Routen erstellen.
Akteure	Küstenleitstand
Auslöser	Küstenleitstand
Vorbedingungen	
Ergebnis	Es sind 2 bis n Wegpunkte gesetzt und als eine zusammengehörige Route gekennzeichnet.
Ablauf	(1) Küstenleitstand definiert 2 bis n Wegpunkte. (2) Wegpunkte werden in der Benutzeroberfläche als zusammengehörige Route gekennzeichnet. (3) Küstenleitstand kann beliebig weitere Routen erstellen.

Tabelle 12: Anwendungsfall 8: Route erstellen

9. Route senden: Eine zuvor erstellte Route muss an die Zuse gesendet werden, damit diese abgefahren werden kann.

Kurzbeschreibung	Route wird vom MATE Produkt an die Zuse gesendet.
Akteure	MATE Produkt, Zuse
Auslöser	Küstenleitstand
Vorbedingungen	Verbindung zur Zuse ist aufgebaut. Route ist erstellt.
Ergebnis	Route ist erfolgreich an die Zuse übertragen.
Ablauf	(1) Erstellte Route wird vom MATE Produkt an die Zuse gesendet.

Tabelle 13: Anwendungsfall 9: Route senden

10. Route bearbeiten: Aufgrund von sich veränderten Bedingungen muss eine erstellte Route, die gegebenenfalls bereits an die Zuse gesendet wurde, bearbeitet werden. Dies beinhaltet, dass ein oder mehrere definierte Wegpunkte einer ausgewählten Route über neue Angaben von Längen- und/oder Breitengrad verändert werden. Eine Bearbeitung der Route führt durch erneutes Senden dieser Route dazu, dass die Route auf der Zuse aktualisiert wird.

Kurzbeschreibung	Einzelne Wegpunkte einer gewählten Route werden angepasst.
Akteure	Küstenleitstand, Zuse
Auslöser	Küstenleitstand
Vorbedingungen	Es existiert eine Route, die bearbeitet werden kann.
Ergebnis	Route ist aktualisiert.
Ablauf	(1) Küstenleitstand wählt eine Route aus, die bearbeitet werden soll. (2) Küstenleitstand verändert Längen- und/oder Breitengrade eines oder von mehreren Wegpunkten. (3) Küstenleitstand bearbeitet bei Bedarf weitere Routen. (4) Aktualisierte Route wird an die Zuse gesendet.

Tabelle 14: Anwendungsfall 10: Route bearbeiten

11. Autopilot aktivieren und 12. Autopilot deaktivieren: Um zwischen einer manuellen Steuerung, die im ersten Meilenstein beschrieben ist, und dem automatisierten Abfahren einer vorgegebenen Route umschalten zu können, ist es notwendig, einen Autopiloten zu aktivieren bzw. zu deaktivieren. Ist der Autopilot deaktiviert, so kann eine manuelle Steuerung über den primären Kanal erfolgen. Ist der Autopilot aktiviert, ist eine manuelle Steuerung über den primären Kanal nicht möglich und der Zuse werden feste Routen vorgegeben, die abzufahren sind. Die manuelle Steuerung in Notsituationen über den sekundären Kanal ist davon nicht betroffen und kann jederzeit erfolgen.

Kurzbeschreibung	Um zwischen manueller Steuerung und Steuerung über die Vorgabe von Wegpunkten zu schalten, ist die Aktivierung bzw. Deaktivierung eines Autopiloten erforderlich.
Akteure	Küstenleitstand
Auslöser	Küstenleitstand
Vorbedingungen	Es ist eine Route erstellt und für den Autopiloten ausgewählt.
Ergebnis	a) Autopilot ist aktiviert und eine erstellte Route wird von der Zuse abgefahren. b) Autopilot ist deaktiviert, es wird keine Route von der Zuse abgefahren. Die Zuse reagiert auf direkte Steuerbefehle.
Ablauf	(1a) Der Küstenleitstand aktiviert den Autopiloten. Die Zuse fährt eine vorgegebene Route ab. (1b) Der Küstenleitstand deaktiviert den Autopiloten. Die Zuse fährt eine bisher vorgegebene Route nicht mehr ab, sondern wird vom Küstenleitstand manuell gesteuert.

Tabelle 15: Anwendungsfall 11 und 12: Autopilot aktivieren bzw. deaktivieren

13. Schiffsposition darstellen: Damit der Küstenleitstand und die Schiffsbesatzung permanent darüber informiert sind, wo sich in der Umgebung noch andere Schiffe befinden, müssen diese in einer Seekarte dargestellt werden. Das vom Küstenleitstand gesteuerte Schiff sollte anders dargestellt werden, damit es von den anderen auf der Seekarte dargestellten Schiffen unterschieden werden kann. Um die Positionen der jeweiligen Schiffe abbilden zu können, ist es notwendig, dass Längen- und Breitengrade der Schiffe bekannt sind und in einem regelmäßigen Intervall aktualisiert werden.

Kurzbeschreibung	Der Küstenleitstand hat Einsicht in die Position von Schiffen in der Umgebung, wobei das eigens gesteuerte Schiff als solches zu erkennen ist.
Akteure	Küstenleitstand, Zuse
Auslöser	MATE Produkt, Zuse, Externe Systeme
Vorbedingungen	Es liegen Positionsdaten der Schiffe in der Umgebung vor. Es ist bekannt, welches Schiff aktuell gesteuert wird.
Ergebnis	Es werden die Schiffe in der Umgebung dargestellt und das Schiff, welches aktuell gesteuert wird, ist hervorgehoben.
Ablauf	(1) Der Küstenleitstand kann die Position der Schiffe in der Umgebung auf einer Seekarte einsehen. (2) Der Küstenleitstand kann die Position des Schiffes erkennen, welches er steuert.

Tabelle 16: Anwendungsfall 13: Schiffsposition darstellen

14. Schiffsposition abrufen: Um die Darstellung von Schiffspositionen in einer Seekarte zu ermöglichen, ist es zunächst notwendig Schiffspositionsdaten in regelmäßigen Intervallen abzurufen. Die Schiffspositionsdaten bestehen jeweils aus einem Längen- und Breitengrad.

Kurzbeschreibung	Abrufen von Schiffspositionsdaten von unterschiedlichen Quellen, um auf Basis dieser Informationen weitere Aktionen durchführen zu können.
Akteure	Küstenleitstand, Schiffsbesatzung, Zuse, Externe Systeme
Auslöser	Zuse, Externe Systeme
Vorbedingungen	Es besteht eine Verbindung zur Zuse und zu Externen Systemen.
Ergebnis	Positionsdaten aller relevanten Schiffe liegen vor und können für weitere Aktionen verwendet werden.
Ablauf	(1) Das MATE Produkt ruft Schiffspositionsdaten von der Zuse ab. (2) Das MATE Produkt ruft Schiffspositionsdaten von externen Systemen ab.

Tabelle 17: Anwendungsfall 14: Schiffsposition abrufen

5.2.3 Dritter Meilenstein

Der dritte Meilenstein besteht aus fünf Anwendungsfällen, die in Abbildung 7 dargestellt sind. Die jeweiligen Anwendungsfälle sind untereinander sowie mit den Akteuren Küstenleitstand und Zuse in Relation gesetzt. Die dargestellten Anwendungsfälle werden im Folgenden aufgegriffen und erläutert.

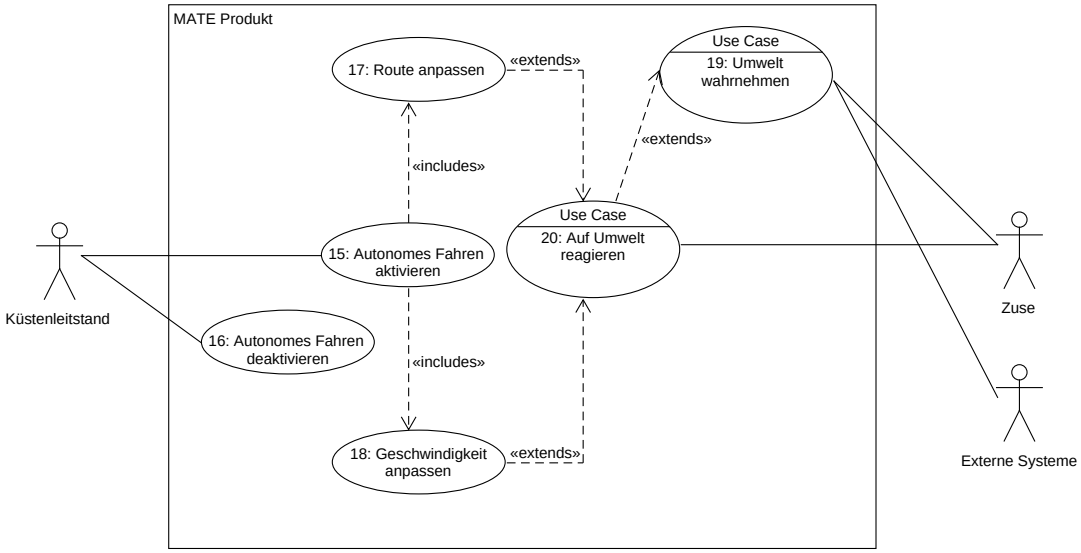


Abbildung 7: Anwendungsfälle des dritten Meilensteins

15. Autonomes Fahren aktivieren und 16. Autonomes Fahren deaktivieren: Der Küstenleitstand aktiviert die Funktionalität „autonomes Fahren“ über eine Benutzeroberfläche. Dies stellt ein Assistenzsystem für den Küstenleitstand dar, da dieser nicht mehr manuell Hindernisse erkennen und auf diese reagieren muss. Die Zuse kann nach Aktivierung der Funktionalität autonom die Umwelt wahrnehmen, Situationen erkennen, in denen Reaktionen notwendig sind, und entsprechend darauf reagieren. Ist das autonome Fahren nicht mehr erwünscht, kann der Küstenleitstand es deaktivieren.

Kurzbeschreibung	Der Küstenleitstand aktiviert das autonome Fahren, um die Zuse zu befähigen autonom auf die nähere Umwelt zu reagieren bzw. deaktiviert das autonome Fahren, falls es nicht mehr gewünscht ist.
Akteure	Küstenleitstand
Auslöser	Küstenleitstand
Vorbedingungen	
Ergebnis	a) Das autonome Fahren ist aktiviert. Die Zuse fährt autonom. b) Das autonome Fahren ist deaktiviert. Die Zuse fährt nicht mehr autonom.
Ablauf	(1a) Küstenleitstand aktiviert über eine Benutzeroberfläche das autonome Fahren der Zuse. (1b) Küstenleitstand deaktiviert über eine Benutzeroberfläche das autonome Fahren der Zuse. (2a) Zuse fährt ab diesem Zeitpunkt autonom. (2b) Zuse fährt ab diesem Zeitpunkt nicht mehr autonom.

Tabelle 18: Anwendungsfall 15 und 16: Autonomes Fahren aktivieren bzw. deaktivieren

17. Route anpassen: Erkennt die Zuse Hindernisse in ihrer näheren Umwelt oder kann Bereiche nicht befahren, muss die Route angepasst werden. Dies führt dazu, dass das Hindernis bzw. der Bereich umfahren wird.

Kurzbeschreibung	Wurde in der näheren Umwelt ein Hindernis erkannt, muss die Route angepasst werden, um das Hindernis zu umfahren.
Akteure	Zuse
Auslöser	Zuse
Vorbedingungen	Es wird ein Hindernis erkannt.
Ergebnis	Die Zuse hat die Route so angepasst, dass das Hindernis umfahren wird.
Ablauf	(1) Es besteht ein Hindernis. (2) Die Zuse passt die Route an. (3) Die Zuse umfährt das Hindernis ohne Schaden zu nehmen.

Tabelle 19: Anwendungsfall 17: Route anpassen

18. Geschwindigkeit anpassen: Die Zuse muss fähig sein, die Geschwindigkeit in unterschiedlichen Szenarien entsprechend anzupassen. Im Schiffsverkehr sind unter anderem vorgeschriebene Geschwindigkeiten einzuhalten, welche die Zuse autonom reguliert. Darüber hinaus muss die Geschwindigkeit angepasst werden, falls Gefahrensituationen eintreten, die einen Schaden der Zuse und dessen näherer Umwelt zur Folge hätten.

Kurzbeschreibung	Geschwindigkeit der Zuse muss in bestimmten unterschiedlichen Szenarien angepasst werden.
Akteure	Zuse
Auslöser	Zuse
Vorbedingungen	Es wird der Bedarf die Geschwindigkeit anzupassen erkannt.
Ergebnis	a) Geschwindigkeit ist auf einen Sollwert erhöht. b) Geschwindigkeit ist auf einen Sollwert verringert.
Ablauf	(1) Es besteht der Bedarf die Geschwindigkeit der Zuse anzupassen. (2a) Die Zuse erhöht die Geschwindigkeit. (2b) Die Zuse verringert die Geschwindigkeit. (3) Die Geschwindigkeit ist angepasst.

Tabelle 20: Anwendungsfall 18: Geschwindigkeit anpassen

19. Umwelt wahrnehmen: Ein Kernbestandteil des autonomen Fahrens ist die Wahrnehmung der näheren Umwelt des autonomen Systems bzw. externer Systeme, welche relevante Daten erfassen. Die Wahrnehmung der Umwelt erfolgt über installierte Sensorik, wie beispielsweise Radar-, Kamerasysteme und GPS.

Kurzbeschreibung	Die Erkennung der näheren Umwelt der Zuse erfolgt über die Erfassung von Daten über Sensorik.
Akteure	Küstenleitstand, Zuse, Externe Systeme
Auslöser	Zuse, Externe Systeme
Vorbedingungen	Sensorik ist installiert.
Ergebnis	Die Umwelt wird laufend wahrgenommen.
Ablauf	(1) Sensoren erkennen Teile der näheren Umwelt.

Tabelle 21: Anwendungsfall 19: Umwelt wahrnehmen

20. Auf Umwelt reagieren: Neben der beschriebenen Wahrnehmung der näheren Umwelt der Zuse, erfordert das autonome Fahren darüber hinaus, dass möglicher Handlungsbedarf erkannt wird und entsprechende Aktionen ausgelöst werden. Dies wird in diesem Anwendungsfall unter der Reaktion auf die Umwelt zusammengefasst.

Kurzbeschreibung	Wird ein Reaktionsbedarf auf Basis der näheren Umwelt festgestellt, so muss die Zuse die Route bzw. die Geschwindigkeit anpassen.
Akteure	Zuse
Auslöser	Zuse
Vorbedingungen	Umwelt wird laufend wahrgenommen. Es wird ein Reaktionsbedarf ermittelt.
Ergebnis	Zuse reagiert auf ein wahrgenommenes Ereignis.
Ablauf	(1) Umwelt wird laufend wahrgenommen und es wird ein Reaktionsbedarf festgestellt. (2) Falls Handlungsbedarf erkannt wurde, wird über die Anpassung der Geschwindigkeit bzw. Anpassung der Route auf den Bedarf reagiert.

Tabelle 22: Anwendungsfall 20: Auf Umwelt reagieren

Die beschriebenen Anwendungsfälle dienen als Basis für die Aufnahme von funktionalen und nichtfunktionalen Anforderungen. Auf die Erhebung, sowie die Gestaltung der Anforderungen wird in dem nachfolgenden Abschnitt eingegangen.

5.3 Anforderungserhebung

Anforderungen wirken unmittelbar auf den Systementwicklungsprozess ein und bilden eine Basis für viele weitere Schritte. Bei einem System handelt es sich „um eine Verbindung von Hardware, Software, Prozessen und Personen [...], die gemeinsam die Fähigkeit haben, ein bestimmtes Ziel zu erreichen oder bestimmte Eigenschaften auszuprägen.“ ([Ebe08], S. 30)

Das Produkt der PG MATE verfolgt das Ziel den Akteuren die in Abschnitt 5.2 definierten Funktionalitäten bereitzustellen. Dabei sind die erhobenen Anforderungen als Eigenschaften des Produktes umzusetzen. Anforderungen dienen laut Rupp „allen am Sys-

tementwicklungsprozess beteiligten als Kommunikations-, Diskussions- und Argumentationsgrundlage.“ ([RP15], S. 16)

Über Anforderungen wird ein gemeinsames Verständnis aller am Entwicklungsprozess beteiligten Personen hergestellt. Dabei beschränkt sich der Nutzen der Anforderungen nicht auf den Zeitraum von der Anforderungserhebung bis zur Systemabnahme, denn sie dienen ebenfalls als Grundlage für Systemänderungen, -erweiterungen sowie der Integration in andere Systeme und die Durchführung von Tests. Anforderungen haben darüber hinaus einen wesentlichen Einfluss auf die Architektur eines Systems. Eine genaue Anforderungsdefinition ist laut Rupp daher unumgänglich (vgl. [RP15], S. 16).

Anforderungen lassen sich laut Rupp nach ihrer Art unterscheiden. Einige für dieses Dokument relevante Anforderungsarten sind:

- Funktionale Anforderungen
- Technologische Anforderungen
- Qualitätsanforderungen
- Anforderungen an die Benutzeroberfläche

In diesem Abschnitt wird primär auf funktionale Anforderungen sowie Qualitätsanforderungen eingegangen. Diese werden in Abschnitt 5.3.2 und Abschnitt 5.3.3 aufgeführt. Technologische Anforderungen sowie Anforderungen an die Benutzeroberfläche werden in Abschnitt 6 und Abschnitt 7 aufgenommen.

Funktionale Anforderungen werden wie folgt definiert:

> „Eine funktionale Anforderung ist eine Anforderung bezüglich des Ergebnisses eines Verhaltens, das von einer Funktion des Systems (oder einer Komponente eines Services) bereitgestellt werden soll.“ ([RP15], S. 17)

Nichtfunktionale Anforderungen kapseln unter anderem Qualitätsanforderungen und stellen Anforderungen dar, die keine funktionalen Anforderungen sind. Ebert beschreibt nichtfunktionale Anforderungen als Eigenschaften und Einschränkungen eines Produkts (vgl. [Ebe08], S. 28).

Die Erhebung von Anforderungen kann über unterschiedliche Quellen erfolgen. Diese sind beispielsweise Dokumente, Systeme in Betrieb und Personen, die Stakeholder darstellen. Dokumente, aus denen wichtige Informationen gewonnen werden können, um Anforderungen aufzunehmen, sind beispielsweise branchenspezifische Normen, Gesetze

und Standards. Zu den branchenspezifischen Standards gehören, wie bereits in Abschnitt 3.1 erläutert, NMEA 0183, NMEA 2000, S-100 und RTZ. Des Weiteren können Anforderungen über Systeme, die sich bereits in Betrieb befinden, aufgenommen werden. Eine Analogie zur autonomen Schifffahrt kann beispielsweise in der Automobilbranche gefunden werden. Aus dieser sind Anforderungen auf die Schifffahrt ableitbar. Eine dritte mögliche Quelle für Anforderungen sind Stakeholder. Diese sind in Abschnitt 5.1 aufgeführt und stellen die primäre Quelle für Anforderungen im Rahmen des Projekts dar.

5.3.1 Anforderungsschablonen

Neben der Erhebung der Anforderungen, aus den beschriebenen Quellen, ist die korrekte und einheitliche Dokumentation dieser notwendig. Dazu werden Anforderungssätze formuliert. Diese haben die Eigenschaft durch ihren gleichartigen Aufbau mit geringem Aufwand von den Projektbeteiligten geschrieben und gelesen werden zu können. Darüber hinaus bieten die Anforderungssätze, die nach einer einheitlichen Schablone verfasst werden, eine höhere Qualität der Anforderungen, da stets alle relevanten Aspekte betrachtet werden (vgl. [RP15], S. 217).

Anforderungen können über eine Anforderungsschablone gebildet werden, die Rupp wie folgt definiert:

> „Eine Anforderungsschablone ist ein Bauplan, der die Struktur eines einzelnen Anforderungssatzes festlegt.“ ([RP15], S. 218).

Es können unterschiedliche Anforderungsschablonen verwendet werden. Die in diesem Projekt eingesetzten Schablonen können in Anforderungsschablonen mit Bedingung sowie Anforderungsschablonen ohne Bedingung unterteilt werden. Diese werden im weiteren Verlauf dieses Abschnitts beschrieben und jeweils mit einem Beispiel versehen.

Anforderungsschablone ohne Bedingung

In Abbildung 8 ist zunächst die Anforderungsschablone ohne Bedingung dargestellt.

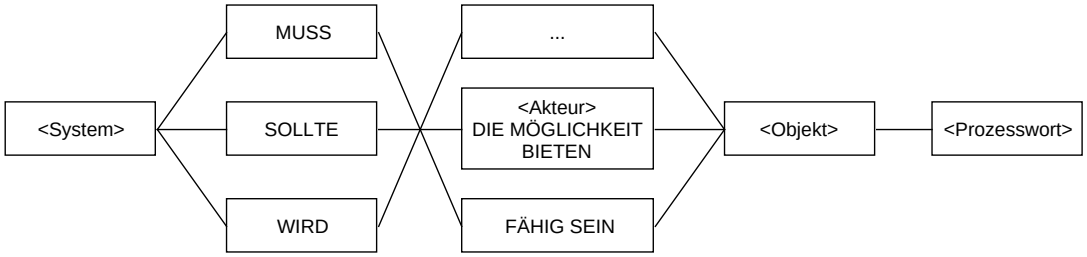


Abbildung 8: Anforderungsschablone ohne Bedingung

Nach der Schablone teilt sich der Anforderungssatz in fünf Teile. Der erste Teil „<System>“ beschreibt das System oder die Komponente, die aktiv oder passiv eine Aktion ausführt.

Auf das System folgt die rechtliche Verbindlichkeit, die über die Begriffe „muss“, „sollte“ oder „wird“ definiert wird. Die rechtliche Verbindlichkeit sorgt für eine klare Kennzeichnung in den Anforderungssätzen und kann laut Rupp im schlimmsten Fall juristisch eingeklagt werden (vgl.[RP15], S. 30). Der Begriff „muss“ wird verwendet, um rechtlich verpflichtende Anforderungen zu benennen. Anforderungen, die den Begriff „sollte“ beinhalten, sind demnach nicht verpflichtend, sondern finden Verwendung für die bessere Zusammenarbeit zwischen Stakeholdern und dem Projektteam. Diese dienen einer höheren Zufriedenheit der Stakeholder. Der Ausdruck „wird“ wird verwendet, um Anforderungen zu kennzeichnen, die in der Zukunft integriert werden. Diese sind als verpflichtend anzusehen und „helfen in der aktuellen Lösung, Vorbereitungen zu treffen, um Zukünftiges später optimal zu integrieren.“ ([RP15], S. 226)

Der dritte Teil der Anforderungsschablone beschreibt die Systemaktivität. Diese unterscheidet sich in drei Typen (vgl. [RP15], S. 222). Zum einen die selbstständige Systemaktivität, die beschreibt, dass ein System einen Prozess selbstständig startet und ausführt. Der Benutzer greift bei dieser Systemaktivität nicht aktiv ein. Ein einfaches Beispiel für einen Anforderungssatz mit einer selbstständigen Systemaktivität ist: „Ein Schiff muss fahren.“

Der zweite Typ beschreibt eine Benutzerinteraktion und wird in der Anforderung mit dem Ausdruck „<Akteur> die Möglichkeit bieten“ versehen. Ein Beispiel für diesen Typ ist: „Das MATE Produkt muss dem Küstenleitstand die Möglichkeit bieten, die Geschwindigkeit der Zuse manuell zu regulieren.“

Der dritte Typ beschreibt eine Schnittstellenanforderung. Diese beschreibt, dass das betrachtete System eine Aktion in Abhängigkeit von Dritten - ausgenommen Benutzern -

ausführt. Dies wird im Anforderungssatz über den Ausdruck „muss fähig sein“ gekennzeichnet. Ein Beispiel hierfür ist: „Das MATE Produkt muss fähig sein, Positionsdaten der Zuse darzustellen.“

Die letzten beiden Teile der Anforderungsschablone beinhalten das <Objekt> sowie ein <Prozesswort>. Über das Objekt wird gekennzeichnet, wofür eine Funktionalität durchgeführt wird. In dem vorherigen Beispiel ist das angegebene Objekt die „Positionsdaten der Zuse“, welches gefolgt ist von dem Prozesswort „darzustellen“. Das Prozesswort beschreibt die geforderte Funktionalität und wird über Verben definiert (vgl. [RP15], S. 220).

Über die Verwendung dieser Schablone ist es möglich, einen Großteil der Anforderungssätze, die aufgenommen werden, einheitlich und mit gleichbleibender Qualität zu beschreiben. Anforderungen, die nicht über die Anforderungsschablone ohne Bedingung beschrieben werden, können unter Umständen unvollständig sein. Daher ist in einigen Fällen eine Bedingung erforderlich. Diese wird im folgenden Abschnitt beschrieben.

Anforderungsschablone mit Bedingung

Rupp beschreibt in (vgl. [RP15], S. 240), dass es für eine Bedingung eine Vielzahl semantischer Belegungen geben kann. Sie beschränkt für Aufnahme von Anforderungen die drei einleitenden Konjunktionen „falls“, „sobald“ und „solange“. Diese sind in der Abbildung 10 dargestellt und erweitern die bereits vorgestellte Anforderungsschablone. Dies ist in Abbildung 9 ersichtlich.

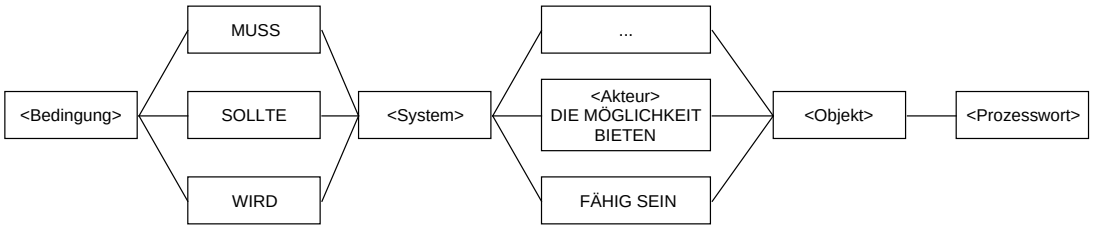


Abbildung 9: Anforderungsschablone mit Bedingung

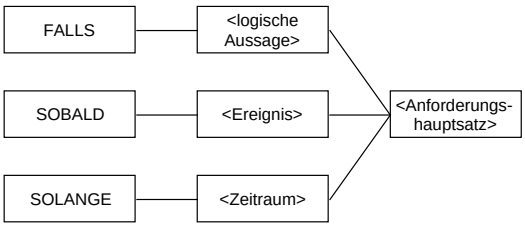


Abbildung 10: Bedingungen für Anforderungsschablonen

Die Konjunktion „falls“ leitet eine logische Aussage ein, die im Rahmen von Anforderungen verwendet wird, um Aktionen an logische Bedingungen zu knüpfen. Der Begriff „sobald“ bezieht sich auf ein Ereignis, welches eintreffen muss, bevor eine Aktion ausgeführt wird. Die dritte mögliche Bedingung wird mit dem Begriff „solange“ eingeleitet und beschreibt Zeiträume, in denen eine in der Anforderung definierte Aktion ausgeführt wird (vgl. [RP15], S. 241).

Die beschriebenen Schablonen werden im Rahmen dieses Dokuments verwendet, um Anforderungssätze auf Basis der Anwendungsfälle sowie der konzipierten Komponenten aufzunehmen.

5.3.2 Funktionale Anforderungen

In diesem Abschnitt werden Anforderungen zu den in Abschnitt 5.2 beschriebenen Anwendungsfällen in Form von Anforderungssätzen aufgenommen. Diese bestehen aus dem Präfix „MATE_REQ“, einer fortlaufenden Nummer, um die Anforderung eindeutig zu identifizieren sowie dem Anforderungssatz.

Anwendungsfall 1

MATE_REQ1: Das MATE Produkt muss dem Küstenleitstand die Möglichkeit bieten, über die Benutzeroberfläche Verbindungsdaten des Schiffs einzugeben.

MATE_REQ2: Das MATE Produkt muss dem Küstenleitstand die Möglichkeit bieten, über die Benutzeroberfläche eine Verbindung zum Schiff aufzubauen.

MATE_REQ3: Das MATE Produkt muss der Schiffsbesatzung die Möglichkeit bieten, über die Benutzeroberfläche Verbindungsdaten des Schiffs einzugeben.

MATE_REQ4: Das MATE Produkt muss der Schiffsbesatzung die Möglichkeit bieten, über die Benutzeroberfläche eine Verbindung zum Schiff aufzubauen.

MATE_REQ5: Falls das MATE Produkt keine aktive Verbindung zum Schiff hat, muss im Statusbereich der Benutzeroberfläche ein rotes Statussymbol dargestellt werden.

MATE_REQ6: Sobald das MATE Produkt eine Aktive Verbindung zum Schiff hat, muss im Statusbereich der Benutzeroberfläche ein grünes Statussymbol für die Verbindung dargestellt werden.

Anwendungsfall 2

MATE_REQ7: Das MATE Produkt muss dem Küstenleitstand die Möglichkeit bieten, über die Benutzeroberfläche Steuerdaten in Form von Motordrehzahl und Ruderwinkel einzugeben.

Anwendungsfall 3

MATE_REQ8: Die Notfallsteuerung muss dem Küstenleitstand die Möglichkeit bieten, über die Benutzeroberfläche eines sekundären Kommunikationskanals Verbindungsdaten des Schiffs einzugeben.

MATE_REQ9: Die Notfallsteuerung muss dem Küstenleitstand die Möglichkeit bieten, mit einem Gamepad über den sekundären Kommunikationskanal einen Verbindungsaufbau von der Notfallsteuerung zum Schiff zu initiieren.

MATE_REQ10: Die Notfallsteuerung muss dem Küstenleitstand die Möglichkeit bieten, mit einem Gamepad über den sekundären Kommunikationskanal eine bestehende Verbindung zum Schiff abzubauen.

MATE_REQ11: Die Notfallsteuerung muss dem Küstenleitstand die Möglichkeit bieten, mit einem Gamepad über den sekundären Kommunikationskanal die Motordrehzahl des Schiffs zu erhöhen bzw. verringern.

MATE_REQ12: Die Notfallsteuerung muss dem Küstenleitstand die Möglichkeit bieten, mit einem Gamepad über den sekundären Kommunikationskanal den Ruderwinkel des Schiffs zu steuern.

MATE_REQ13: Falls die Schaltflächen des Gamepads auf einer neutralen Position stehen, muss die Backup-Steuerung fähig sein, die Motordrehzahl des Schiffs in einen Leerlauf zu schalten und die Ruderstellung des Schiffs in eine neutrale Position zu stellen.

Anwendungsfall 4

MATE_REQ14: Das MATE Produkt muss fähig sein, vom Küstenleitstand definierte Steuerdaten an das Schiff zu senden.

Anwendungsfall 5

MATE_REQ15: Das MATE Produkt muss fähig sein, folgende Sensordaten in Form von Nachrichten des NMEA2000-Standards von dem Schiff abzurufen:

- Geschwindigkeit durch das Wasser
- Motordrehzahl
- Windwinkel
- Windgeschwindigkeit
- Ruderwinkel
- GPS-Koordinaten (Längengrad, Breitengrad)
- Wassertiefe
- Steuerkurs
- Drehgeschwindigkeit
- Radardaten
- AIS

MATE_REQ16: Das MATE Produkt muss fähig sein, AIS-Daten von externen Systemen abzurufen.

Anwendungsfall 6

MATE_REQ17: Das MATE Produkt muss dem Küstenleitstand die Möglichkeit bieten, über die Benutzeroberfläche folgende Sensordaten darzustellen:

- Geschwindigkeit durch das Wasser in Knoten
- Motordrehzahl
- Windwinkel
- Windgeschwindigkeit
- Ruderwinkel
- GPS-Koordinaten (Längengrad, Breitengrad)
- Echolot in Meter
- Steuerkurs
- Drehgeschwindigkeit
- AIS

Anwendungsfall 7

MATE_REQ18: Das MATE Produkt muss dem Küstenleitstand die Möglichkeit bieten, über die Benutzeroberfläche eine bestehende Verbindung zum Schiff abzubauen.

MATE_REQ19: Das MATE Produkt muss der Schiffsbesatzung die Möglichkeit bieten, über die Benutzeroberfläche eine bestehende Verbindung zum Schiff abzubauen.

MATE_REQ20: Sobald das MATE Produkt die bestehende Verbindung zum Schiff abgebaut hat, muss im Bereich der Benutzeroberfläche ein rotes Statussymbol für die Verbindung dargestellt werden.

Anwendungsfall 8

MATE_REQ21: Das MATE Produkt muss dem Küstenleitstand die Möglichkeit bieten, über die Benutzeroberfläche in einer Seekarte eine oder mehrere Routen, bestehend aus 2 bis n Wegpunkten zu erstellen, zu speichern, zu bearbeiten und zu löschen.

MATE_REQ22: Das MATE Produkt muss fähig sein, dem Küstenleitstand über die Benutzeroberfläche in einer Seekarte alle definierten Routen, die Wegpunkte einer Route sowie eine Verbindungslinie zwischen den Wegpunkten darzustellen.

Anwendungsfall 9

MATE_REQ23: Das MATE Produkt muss fähig sein eine zuvor definierte und vom Küstenleitstand aktivierte Route im RTZ-Format (siehe Abschnitt 3.1.4) an das Schiff zu senden.

Anwendungsfall 10

MATE_REQ24: Das MATE Produkt wird dem Küstenleitstand die Möglichkeit bieten, über die Benutzeroberfläche eine aktive Route zu bearbeiten, während das Schiff diese abfährt.

Anwendungsfall 11

MATE_REQ25: Das MATE Produkt muss dem Küstenleitstand die Möglichkeit bieten, über die Benutzeroberfläche einen Autopiloten zu einer zuvor definierten Route zu aktivieren.

MATE_REQ26: Sobald der Autopilot aktiviert ist, wird die aktive Route vom MATE Produkt an das Schiff gesendet.

MATE_REQ27: Sobald der Autopilot aktiviert ist, wird im Statusbereich der Benutzeroberfläche ein grünes Statussymbol für den Autopiloten dargestellt.

Anwendungsfall 12

MATE_REQ28: Das MATE Produkt muss dem Küstenleitstand die Möglichkeit bieten, über die Benutzeroberfläche einen aktiven Autopiloten zu deaktivieren.

MATE_REQ29: Sobald der Autopilot deaktiviert ist, wird die Route vom MATE Produkt auf den Status inaktiv gesetzt.

MATE_REQ30: Sobald der Autopilot deaktiviert ist, wird im Statusbereich der Benutzeroberfläche ein rotes Statussymbol für den Autopiloten dargestellt.

Anwendungsfall 13

MATE_REQ31: Das MATE Produkt muss fähig sein, dem Küstenleitstand über die Benutzeroberfläche die Position des eigenen Schiffes in einer Seekarte darzustellen und farblich von anderen Schiffen hervorzuheben.

MATE_REQ32: Das MATE Produkt muss fähig sein, der Schiffsbesatzung über die Benutzeroberfläche die Position des eigenen Schiffes in einer Seekarte darzustellen und farblich von anderen Schiffen hervorzuheben.

Anwendungsfall 14

MATE_REQ33: Das MATE Produkt muss fähig sein, die aktuelle Position des eigenen Schiffs und von möglichen anderen Schiffen vom eigenen Schiff abzurufen.

MATE_REQ34: Das MATE Produkt muss fähig sein, die aktuelle Position des eigenen Schiffs und von anderen Schiffen von einem externen System abzurufen.

Anwendungsfall 15

MATE_REQ35: Das MATE Produkt wird dem Küstenleitstand die Möglichkeit bieten, über die Benutzeroberfläche das autonome Fahren zu aktivieren.

MATE_REQ36: Sobald das autonome Fahren aktiviert ist, wird im Statusbereich der Benutzeroberfläche ein grünes Statussymbol für das autonome Fahren dargestellt.

Anwendungsfall 16

MATE_REQ37: Das MATE Produkt wird dem Küstenleitstand die Möglichkeit bieten, über die Benutzeroberfläche das autonome Fahren zu deaktivieren.

MATE_REQ38: Sobald das autonome Fahren deaktiviert ist, wird im Statusbereich der Benutzeroberfläche ein rotes Statussymbol für das autonome Fahren dargestellt.

Anwendungsfall 17

MATE_REQ39: Das MATE Produkt wird fähig sein, eine vorgegebene Route anzupassen, um Hindernisse bzw. nicht befahrbare Bereiche zu umfahren.

Anwendungsfall 18

MATE_REQ40: Das MATE Produkt wird fähig sein, die Geschwindigkeit des Schiffs zu regulieren.

Anwendungsfall 19

MATE_REQ41: Das MATE Produkt wird fähig sein, die Umwelt des Schiffs über folgende Sensoren wahrzunehmen:

- Kamera
- Radar
- Echolot
- DGPS Kompass
- AIS
- IMU
- Windsensor
- Lidar
- Ultraschallsensoren
- Infrarotkameras

Anwendungsfall 20

MATE_REQ42: Das MATE Produkt wird fähig sein, auf Basis der aufgezeichneten Sensordaten die Geschwindigkeit des Schiffs anzupassen.

MATE_REQ43: Das MATE Produkt wird fähig sein, auf Basis der aufgezeichneten Sensordaten die vorgegebene Route des Schiffs anzupassen.

5.3.3 Nichtfunktionale Anforderungen

Die nichtfunktionalen Anforderungen, die auch Qualitätsattribute genannt werden, stellen laut Ebert Eigenschaften und Einschränkungen eines Produktes dar (vgl. [Ebe08], S. 28). Spezifiziert werden diese über Verhaltensweisen und Qualitätseigenschaften der funktio-

nalen Anforderungen. Der ISO[1]-Standard 9126 strukturiert nichtfunktionale Anforderungen in Übertragbarkeit, Funktionalität, Wartbarkeit, Zuverlässigkeit und Benutzbarkeit (vgl. [ISO01]). In 2011 ist diese ISO-Norm durch die Norm ISO/IEC[2] 25010 ersetzt worden (vgl. [ISO11]). Im Rahmen dessen wurde die Struktur um Effizienz, Sicherheit und Kompatibilität erweitert.

Im Folgenden werden die jeweiligen Qualitätsattribute genannt, erläutert sowie um Anforderungen an das Produkt der PG MATE erweitert:

Übertragbarkeit: Die Übertragbarkeit, auch Portabilität genannt, stellt die Fähigkeit eines Softwaresystems dar, von einer Umgebung in eine andere übertragen zu werden. Dabei werden verschiedene Gesichtspunkte betrachtet, wie die Anpassbarkeit und die Installierbarkeit. Diese beschreiben, ob ein Softwareprodukt in einer spezifizierten Umgebung installiert werden kann, wobei nur festgelegte Mittel und Aktionen eingesetzt werden.

MATE_REQ44: Das MATE Produkt muss fähig sein, auf den Betriebssystemen Windows 7 64-bit und Linux 64-bit ausgeführt zu werden.

Funktionalität: Softwaresysteme stellen durch Systemkomponenten Funktionalität für andere Systemkomponenten zur Verfügung. Balzert beschreibt eine Funktion folgendermaßen:

> „Eine Funktion beschreibt eine Tätigkeit oder eine klar umrissene Aufgabe innerhalb eines größeren Zusammenhangs.“ ([Bal09], S. 127)

Aus den im Abschnitt 5.3.2 definierten funktionalen Anforderungen lassen sich unterschiedliche Funktionen ableiten, die nach Balzert (vgl. [Bal09], S. 468) festgelegte Bedürfnisse erfüllen müssen. Zu diesen gehört die Fähigkeit einer Komponente, für spezifizierte Aufgaben geeignete Funktionen bereitzustellen. Darüber hinaus müssen die richtigen Ergebnisse erzielt werden. Ein weiterer relevanter Punkt ist die Konformität der Funktionalität. Diese beschreibt, dass Standards, Konventionen oder gesetzliche Richtlinien, wie in folgenden Anforderungen aufgeführt:

MATE_REQ45: Das MATE Produkt muss den NMEA0183-Standard unterstützen.

MATE_REQ46: Das MATE Produkt muss den NMEA2000-Standard unterstützen.

[1] http://www.iso.org/iso/home.html [letzter Zugriff: 17.09.2017].
[2] http://www.iec.ch/ [letzter Zugriff 17.09.2017].

MATE_REQ47: Das MATE Produkt muss den RTZ-Standard unterstützen.

MATE_REQ48: Das MATE Produkt muss die Regularien des Bundesgesetzes über die Binnenschifffahrt (Schifffahrtsgesetz) einhalten.

MATE_REQ49: Das MATE Produkt muss die Regularien des Hafenverkehrs- und Schiffahrtsgesetz einhalten.

Wartbarkeit: Durch die Wartbarkeit eines Softwareprodukts wird die Änderungsfähigkeit dessen beschrieben. Die Änderungsfähigkeit beinhaltet Korrekturen, Verbesserungen oder Anpassungen der Software. Die Wartbarkeit zeichnet sich dabei dadurch aus, dass es mit geringem Aufwand möglich ist Ursachen für Mängel zu diagnostizieren. Des Weiteren muss sich die Software änderbar gestalten, was beinhaltet, dass Implementierungen vorgenommen werden können, ohne tiefe Eingriffe in die Architektur des bestehenden Systems vornehmen zu müssen. Ein Kernpunkt der Wartbarkeit stellt die Testbarkeit dar, welche die Fähigkeit beschreibt, modifizierte Software zu testen.

Eine hohe Wartbarkeit wird in dem Produkt der PG MATE beispielsweise durch geeignete Logging-Mechanismen, ausgiebige Kommentierung des Quellcodes, Dokumentationen und die Einhaltung von Entwicklungsrichtlinien erreicht.

MATE_REQ50: Das MATE Produkt muss eine modulare sowie testbare Softwarearchitektur aufzuweisen.

MATE_REQ51: Das MATE Produkt muss eine hohe Testabdeckung aufweisen.

Effizienz: Unter Effizienz wird die Fähigkeit eines Softwareprodukts verstanden, beim Einsatz von Hardwareressourcen ein angemessenes Leistungsniveau aufrechtzuerhalten. Dazu zählt eine angemessene Verarbeitungszeit der implementierten Funktionen.

MATE_REQ52: Das MATE Produkt muss fähig sein, zu Daten in einer akzeptablen Latenzzeit zu verarbeiten.

Zuverlässigkeit: Die Zuverlässigkeit beschreibt die Fähigkeit eines Softwareprodukts, ein bestimmtes Leistungsniveau aufrechtzuerhalten, während es unter zuvor festgelegten Bedingungen genutzt wird. Dies ist in dem Fall möglich, wenn das Softwareprodukt einen gewissen Reifegrad erreicht hat. Ab diesem ist es selbst nach einem Absturz möglich, ein gewisses Leistungsniveau wiederherzustellen und betroffene Daten wiederzugewinnen.

MATE_REQ53: Das MATE Produkt muss fähig sein, veraltete Daten, die keine Verwendung mehr finden, regelmäßig zu löschen.

Benutzbarkeit: Die Benutzbarkeit eines Softwareproduktes zeichnet sich dadurch aus, dass ein Benutzer dieses versteht und es verwenden kann. Das Softwareprodukt muss intuitiv bedienbar oder unter möglichst geringem Aufwand erlernbar sein.

MATE_REQ54: Das MATE Produkt muss sicherstellen, dass korrekte Benutzereingaben, valide Daten erzeugen die weiterverarbeitet werden können.

Kompatibilität: Die Kompatibilität beschriebt die Fähigkeit einer Softwarekomponente mit anderen Komponenten zusammenzuarbeiten, ohne dabei Einschränkungen zu unterliegen.

MATE_REQ55: Das MATE Produkt muss sicherstellen, dass Sensordaten, die zwischen den Komponenten des Produkts ausgetauscht werden, nicht verfälscht werden und valide sind.

MATE_REQ56: Das MATE Produkt muss sicherstellen, dass Steuerbefehle, die an die Zuse gesendet werden, nicht verfälscht werden und valide sind.

6 Systementwurf

Ausgehend von den Ergebnissen der Anforderungsanalyse erfolgt in diesem Kapitel die Beschreibung des Systementwurfs für das zu erstellende Produkt.

Dazu wird zuerst eine Gesamtarchitektur erstellt, welche aus mehreren Teilkomponenten besteht. Anschließend werden Konzepte für die Teilkomponenten entworfen. Auf Basis dieser Konzepte wird später die Implementierung erfolgen. Daher ist es wichtig, dass für jede Komponente die Aufgaben und der Datenfluss über Schnittstellen festgelegt werden.

6.1 Systemarchitektur

In diesem Abschnitt wird auf die Gesamtsystemarchitektur des zu konzipierenden und zu entwickelnden Systems eingegangen. Zunächst wird anhand eines Komponentendiagramms der allgemeine Aufbau erklärt. Im Anschluss folgt ein zweites Komponentendiagramm in dem der Aufbau des für die PG MATE relevanten Testbeds erläutert wird. Dieses Vorgehen soll einen einfachen Einstieg geben und somit ein besseres Verständnis des Systementwurfs ermöglichen.

6.1.1 Produkt-Systemarchitektur

Abbildung 11 zeigt auf hoher Abstraktionsebene die grundlegenden Komponenten des zu konzipierenden Produkts.

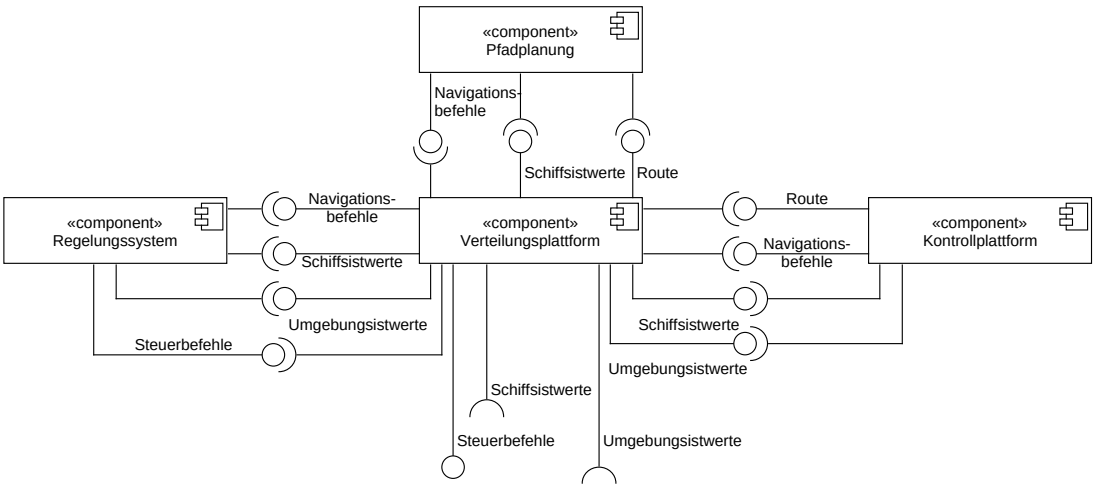


Abbildung 11: Systemarchitektur des PG Produkts

Im Folgenden werden die einzelnen Komponenten kurz beschrieben. Eine detaillierte Beschreibung der Komponenten und die Konzeption dieser erfolgt in den folgenden Abschnitten.

Verteilungsplattform

Die Verteilungsplattform fungiert als zentrales Kommunikationselement, das den Austausch von Informationen zwischen den einzelnen Komponenten des Produkts ermöglicht. Die einzelnen Komponenten kommunizieren voraussichtlich nicht mit den gleichen Nachrichtenformaten bzw. -standards. Dies ist unter anderem dem Umstand geschuldet, dass das zu entwickelnde Produkt in eine bereits existierende heterogene Systemlandschaft zu integrieren ist und mit anderen Software-Anwendungen kommunizieren muss, die verschiedene Nachrichtenstandards verwenden. Die Software-Anwendungen sollten dabei nicht umfangreich modifiziert werden.

Die Verteilungsplattform muss somit über die Möglichkeit verfügen, verschiedene Nachrichtenstandards zu verstehen und Nachrichten eines Standards in Nachrichten eines anderen Standards zu transformieren. Außerdem muss die Verteilungsplattform als zentrales Kommunikationselement über die Möglichkeit verfügen Nachrichten von definierten Quellen zu Empfangen (zum Beispiel der Pfad-Planungs-Komponente) und diese zu einem definierten Empfänger zu Senden (zum Beispiel dem Regelungssystem). Neben zahlreichen Schnittstellen mit anderen Komponenten des Produkts muss die Vermittlungs-

plattform auch über Schnittstellen zum Testbed verfügen, um zum einen Steuerbefehle an dieses senden zu können und zum anderen Schiffs- und Umgebungsistwerte von diesem zu empfangen.

Kontrollplattform

Die Kontrollplattform stellt ein, ähnlich wie bereits in der Schifffahrt eingesetztes ECDIS, elektronisches Navigationsinformationssystem dar, dass neben der Seekartendarstellung die Darstellung von Sensordaten wie Positionsdaten, AIS-Daten und schiffsbezogene Informationen wie den aktuellen Kurs ermöglicht. Zu diesem Zweck muss die Kontrollplattform Schifffs- und Umweltistwerte entgegennehmen können. Neben diesen Monitoring-Aufgaben soll die Kontrollplattform als Schnittstelle für Eingaben vom Anwender im Küstenleitstand dienen. Dabei müssen im Falle der manuellen Steuerung Steuerbefehle zu Geschwindigkeit und Ruderausrichtung entgegengenommen werden. Für den Autopiloten muss die Möglichkeit zum Erstellen und Speichern von Routen gegeben sein. Des Weiteren muss es dem Anwender möglich sein das Abfahren von Routen zu initiieren.

Pfadplanung

Die Pfadplanungskomponente hat die Aufgabe auf Basis einer vorgegebenen Route und der aktuellen Position des Schiffs zu ermitteln, welche beiden Wegpunkte das Schiff aktuell abfahren muss. Diese zwei Wegpunkte definieren die aktuelle Teilstrecke der Route. Die Pfadplanungskomponente muss diese Wegpunkte an das Regelungssystem und die Kontrollplattform übergeben, von der die Route stammt. Eine Route ist dabei ein Tupel aus mindestens zwei Wegpunkten, die in einer vorgegebenen Reihenfolge abgefahren werden müssen. Ein Wegpunkt bezeichnet eine Position auf der Erdoberfläche bestehend aus den geografischen Koordinaten Längen- und Breitengrad. Neben diesen Informationen kann eine Route weitere Parameter enthalten, wie beispielsweise die vorgegebene Geschwindigkeit für die Teilstrecke oder einen Radius, der festlegt, wann ein Wegpunkt als erreicht gilt. Die Pfadplanungskomponente kommt nur beim automatischen Abfahren von Routen zum Einsatz. Beim manuellen Abfahren einer Route bleibt sie inaktiv.

Regelungssystem

Aufgabe des Regelungssystems ist die Ermittlung von Steuerparametern (zum Beispiel Motordrehzahl und Ruderwinkel) für ein spezifisches Schiff auf dem das zu entwickeln-

de Produkt implementiert ist. Im Falle der manuellen Steuerung passiert dies auf Basis der übertragenen Navigationsbefehle von der Kontrollplattform. Bei aktiviertem Autopiloten müssen die aktuellen Wegpunkte verarbeitet werden können, um auf Basis dieser die korrekten Steuerbefehle zu ermitteln, damit das Schiff die vorgegebene Route sinnvoll abfahren kann. Dabei verhindert das Regelungssystem unter anderem ein Zick-Zack-Fahren des Schiffs.

Die vorgegebenen Wegpunkte erhält das Regelungssystem in diesem Fall von der Pfadplanungskomponente. Zur Ermittlung der korrekten Steuerbefehle benötigt das Regelungssystem weitere Informationen. Dies sind verschiedene Umwelt- und Schiffsistwerte, welche das Regelungssystem von der Verteilungsplattform erhält.

6.1.2 Testbed Systemarchitektur

In diesem Abschnitt wird die Systemarchitektur des Testbeds in dem das Produkt der PG MATE eingesetzt wird vorgestellt. Dies ist notwendig, da die spätere Implementierung des Produkts nah am Testbed geschieht. Zudem befindet sich mit der Notfall-Steuerung eine Komponente der PG MATE unabhängig vom übrigen Produkt im Testbed. Diese Komponente ist notwendig, da die Notfallsteuerung einen vom Produkt unabhängigen zweiten Kommunikationsweg nutzen soll. Abbildung 12 zeigt einen abstrakten und für die PG MATE relevanten Aufbau des Testbeds.

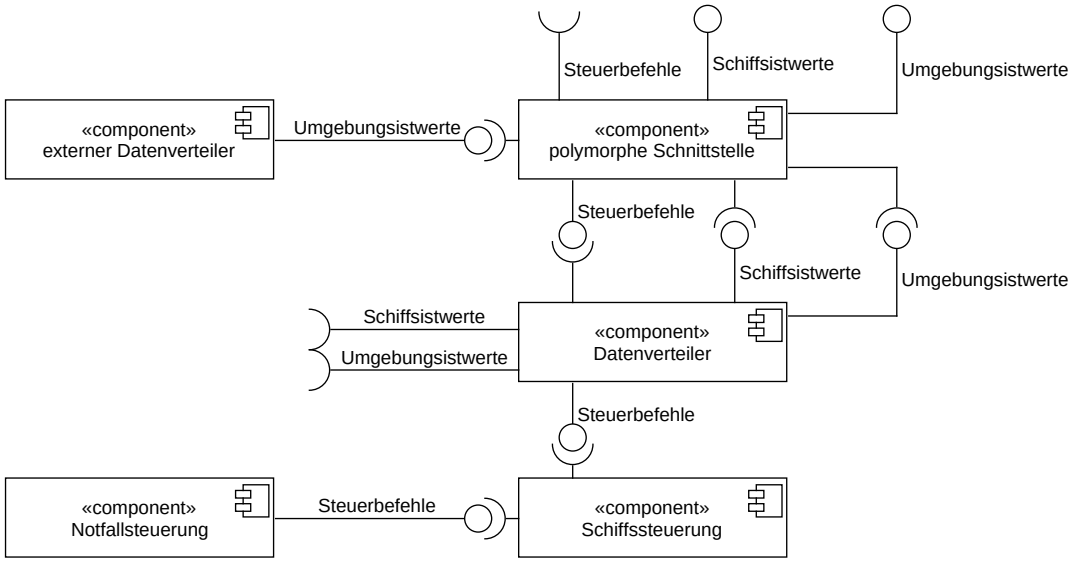


Abbildung 12: Systemarchitektur des Testbeds mit Notfallsteuerung

Im Folgenden werden die einzelnen Komponenten der Abbildung 12 kurz beschrieben:

Polymorphe Schnittstelle

Die polymorphe Schnittstelle dient, ähnlich wie die Vermittlungsplattform im PG MATE Produkt, als Komponente zum Nachrichtenaustausch zwischen anderen Komponenten. Im konkret vorliegenden Fall vermittelt es zwischen dem Datenverteiler und der Schnittstelle zum Produkt der PG MATE und ermöglicht eine Verbindung zwischen beiden Systemen. Dafür müssen Schiffsistwerte wie Sensordaten und Umgebungsistwerte wie die Windgeschwindigkeit vom Datenverteiler empfangen werden und dem MATE Produkt zur Verfügung gestellt werden. Vom MATE Produkt müssen parallel Steuerbefehle entgegengenommen und ans Datenverteiler weitergeleitet werden.

Datenverteiler

Aufgabe des Datenverteilers ist unter anderem die Aufbereitung und Bereitstellung von Sensordaten (Schiffs- und Umgebungsistwerte) des jeweiligen Schiffs, die der Vermittlungsplattform in einem definierten Nachrichtenformat zur Weiterverarbeitung zur Verfügung gestellt werden. Im Falle des Testbeds stellt das Schiff die Zuse dar. Außerdem muss der Datenverteiler Steuerbefehle für verschiedene Aktuatoren entgegennehmen und verarbeiten können.

Externer Datenverteiler

Der externe Datenverteiler übergibt der schiffsseitigen polymorphen Schnittstelle Umgebungsistwerte. Der externe Datenverteiler stellt dem Testbed zum Beispiel AIS-Daten aus externen Quellen zur Verfügung.

Notfallsteuerung

Die Notfallsteuerung setzt die Anforderung nach einer Möglichkeit zum Senden von Steuerbefehlen an das Schiff in einer Notfallsituation unabhängig von allen anderen Komponenten des zu entwickelnden Produkts um. Dabei werden über einen sekundären Kommunikationsweg Steuerbefehle über ein unabhängiges Eingabegerät im Küstenleitstand direkt an die Schiffssteuerung des Schiffs gesendet.

Sensoren und Aktuatoren

Auch die Sensoren und Aktuatoren sind, wie der Datenverteiler selbst, schiffsspezifisch zu implementierende plattformabhängige Komponenten. Diese kommunizieren (zumindest im Rahmen dieser Architektur) nur mit dem Datenverteiler und der Notfallsteuerung.

Schiffssteuerung Bei der Schiffssteuerung handelt es sich um die Steuereinheit der Aktuatoren Motor und Ruder. Im Falle des Testbeds wird über die Schiffssteuerung sowohl die Motordrehzahl als auch der Ruderwinkel gesteuert.

6.1.3 Entwurf der Softwarekomponenten

In den folgenden Abschnitten wird der Entwurf der verschiedenen Softwarekomponenten beschrieben. Nachdem in den vorherigen Abschnitten die unterschiedlichen Aufgaben und die grobe Zusammensetzung des Gesamtsystems geklärt wurden, wird jetzt näher auf den inneren Aufbau der einzelnen Komponenten eingegangen.

Es werden die zu Grunde liegenden Konzepte erläutert und auf Spezifikationen eingegangen, die bei der späteren Implementierung bedacht werden müssen. Des Weiteren werden pro Komponente die Schnittstellen nach außen definiert. Begleitet werden die Beschreibungen durch Komponenten- oder Klassendiagramme zur besseren Übersicht.

6.2 Polymorphe Schnittstelle

Das Testbed bzw. das Forschungsboot Zuse ist mit einem Datenverteiler ausgestattet (siehe Abschnitt 6.1.2). Der Datenverteiler ist eine Komponente, welche die Daten verschiedener Sensoren des Boots bündelt und diese in S-100 zur Verfügung stellt. Des Weiteren nimmt der Datenverteiler Sollwerte für die Steuerung des Boots in S-100 entgegen und leitet diese an die Schiffssteuerungskomponente weiter. Zum Senden und Empfangen der Daten nutzt der Datenverteiler das Advanced Message Queuing Protocol (AMQP).

Die Anforderung *MATE_REQ15* erfordert, dass das MATE Produkt Daten im Protokoll NMEA2000 über User Datagram Protocol (UDP) mit dem Testbed (und später mit anderen Schiffen) austauschen muss. Dadurch wird die Einsatzmöglichkeit des Produkts auf einer Vielzahl moderner Schiffstypen gewährleistet.

Aus dem eben beschriebenen Aufbau des Datenverteilers und der Anforderung an das MATE Produkt ergibt sich, dass eine Transformierung der Datenübertragungsart und der

Semantik der Daten stattfinden muss. Die Art der Datenübertragung wird zwischen UDP und AMQP transformiert. Auf semantischer Ebene muss S-100 in NMEA2000 transformiert werden. In beiden Fällen handelt es sich um eine bidirektionale Transformation, da Daten jeweils empfangen und gesendet werden.

Um Interoperabilität zwischen dem Testbed und dem MATE Produkt herzustellen, kommen verschiedene Lösungsansätze infrage:

- Entwicklung eines spezifischen Adapters (vgl. [MT04], S. 619). Dieser Adapter würde ausschließlich eine beidseitige Transformation zwischen NMEA2000 und S-100 sowie UDP und AMQP unterstützen.
- Entwicklung einer erweiterbaren, wiederverwendbaren Komponente zur Herstellung von Interoperabilität auf Basis einer domänenspezifischen Ontologie (vgl. [MT04], S. 619). Diese Lösung könnte beliebige Softwareanwendungen derselben Domäne kompatibel zueinander machen.

Die PG MATE hat den Auftrag den zweiten allgemeinen Ansatz umzusetzen. Zu diesem Zweck wird eine eigenständige Komponente entwickelt. Die polymorphe Schnittstelle ist eine Komponente der Systemarchitektur des Testbeds und gehört somit nicht direkt zu dem MATE Produkt.

Es sprechen einige Gründe für die Entwicklung der polymorphen Schnittstelle, auch wenn die Interoperabilität zwischen MATE Produkt und Testbed auch durch einen spezifischen Adapter hergestellt werden könnte. Die Entwicklung eines Adapters ist komplex und zeitintensiv. Bei Änderungen auf einer Seite des Systems ist eine Anpassung des Adapters nötig. Wenn weitere Systeme integriert werden müssen, die eine abweichende Form der Datenübertragung oder Syntax der Daten aufweisen, werden weitere Adapter benötigt (vgl. [MT04], S. 619). Diese Nachteile treten bei der Entwicklung einer polymorphen Schnittstelle und eines Referenzschemas nicht auf. Die initiale Entwicklung ist wie bei einem Adapter komplex und zeitintensiv. Der Aufwand für Erweiterungen und Anpassungen an der polymorphen Schnittstelle ist jedoch geringer, falls Änderungen in der Datenübertragung geschehen. Es müsste im einfachsten Fall ausschließlich eine Konfiguration der polymorphen Schnittstelle angepasst werden. Neben diesen Vorteilen kann die entwickelte polymorphe Schnittstelle wiederverwendet werden, da es sich um eine allgemeine Lösung zur Herstellung von Interoperabilität handelt.

6.2.1 Modell

Ein Modell ist eine vereinfachte Darstellung der Realität. Die Standards NMEA2000 und S-100 beschreiben beispielsweise jeweils eine Modellierung, mit der Informationen der maritimen Domäne dargestellt werden können. Eine Aufgabe der polymorphen Schnittstelle ist es, Instanzen solcher Modelle zu anderen Modellen zu transformieren.

Es ist sinnvoll, Informationen vor einer Transformation in eine standardisierte Form zu bringen, die sich gut weiterverarbeiten lässt. Extensible Markup Language (XML) ist eine solche Form, mit der sich hierarchisch strukturierte Informationen darstellen lassen. Um XML gibt es zudem ein mächtiges Ökosystem. Es gibt Möglichkeiten zur Validierung von XML-Dokumenten mithilfe von XML-Schemas. Transformationen zwischen verschiedenen XML-Schemas können mit Extensible Stylesheet Language Transformations (XSLT) durchgeführt werden. Es können Abfragen auf XML-Dokumente mit XQuery und XPath durchgeführt werden, um bestimmte Teilmengen herauszusuchen (vgl. [Mel11], S. 3). Des Weiteren gibt es eine Vielzahl an Softwarebibliotheken, die den Umgang mit XML und den dazugehörigen Technologien ermöglichen.

XML-Schemas bieten die Möglichkeit, Modelle formal zu spezifizieren. Bei einem XML-Schema handelt es sich wiederum um ein XML-Dokument. Es beschreibt die Struktur einer Instanz des entsprechenden Modells (vgl. [Mel11], S. 101). So kann zum Beispiel festgelegt werden in welchen Wertebereich sich ein Attribut eines Elements befinden muss. Die Beschreibung eines Modell mithilfe eines XML-Schemas hat den Vorteil, dass Instanzen des Modells validiert werden können (vgl. [Mel11], S. 100).

```
1  <?xml version="1.0" encoding="UTF−8" ?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3    <xs:element name="ships">
4      <xs:complexType>
5        <xs:sequence>
6          <xs:element name="ship" maxOccurs="unbounded">
7            <xs:complexType>
8              <xs:sequence>
9                <xs:element name="name" type="xs:string" />
10               <xs:element name="type">
11                 <xs:simpleType>
12                   <xs:restriction base="xs:string">
13                     <xs:enumeration value="FISHING_SHIP" />
14                     <xs:enumeration value="SAILING_SHIP" />
15                     <xs:enumeration value="RESEARCH_SHIP" />
```

```
16                          <xs:enumeration value="UNKNOWN" />
17                        </xs:restriction>
18                      </xs:simpleType>
19                    </xs:element>
20                  </xs:sequence>
21                </xs:complexType>
22              </xs:element>
23            </xs:sequence>
24          </xs:complexType>
25        </xs:element>
26    </xs:schema>
```

Auflistung 1: Beispiel eines Modells einer Liste von Schiffen als XML-Schema

In Auflistung 1 wird ein Beispiel eines XML-Schemas dargestellt. Es handelt sich um die Beschreibung eines Modells, mit dem eine Auflistung von Schiffen mit ihrem Namen und Typ dargestellt werden kann. In der Schiffsliste können beliebig viele Schiffe enthalten sein. Jedes Schiff hat einen Freitext als Namen. Des Weiteren muss der Schiffstyp gesetzt sein und einen der folgenden Werte enthalten: FISHING_SHIP, SAILING_SHIP, RESEARCH_SHIP oder UNKNOWN.

Für die Entwicklung der polymorphen Schnittstelle ist es essenziell zu wissen, wie die Modelle definiert sind, die transformiert werden sollen (vgl. [MT04], S. 630). Für jedes benötigte Modell muss demnach ein XML-Schema vorliegen. Das XML-Schema für S-100 kann aus eMIR mit dem Eclipse Modeling Framework (EMF) generiert werden. Für NMEA2000 muss eine geeignete XML-Repräsentation erstellt werden.

6.2.2 Modellinstanz

Als Modellinstanz wird im Kontext der polymorphen Schnittstelle eine konkrete Ausprägung eines Modells bezeichnet. Eine Modellinstanz wird durch ein XML-Dokument repräsentiert, welches dem XML-Schema des jeweiligen Modells entsprechen muss. Beispielsweise gibt es verschiedene Informationen, die mit NMEA2000 oder S-100 kommuniziert werden können. Die Position des Schiffes wird in einer eigenen Nachricht übermittelt. Um die Nachricht zu transformieren, wird sie in die entsprechende XML-Repräsentation gebracht. Bei dem entstandenen XML-Dokument handelt es sich um die Modellinstanz.

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <ships>
```

```
3   <ship>
4     <name>Zuse</name>
5     <type>RESEARCH_SHIP</type>
6   </ship>
7   <ship>
8     <name>Marie</name>
9     <type>UNKNOWN</type>
10  </ship>
11 </ships>
```

Auflistung 2: Beispiel einer Modellinstanz einer Liste von Schiffen als XML-Dokument

In Auflistung 2 wird ein Beispiel einer Modellinstanz einer Liste von Schiffen dargestellt, die dem XML-Schema aus Abschnitt 6.2.1 entspricht. Die Liste enthält zwei Schiffe. Das erste Schiff trägt den Namen „Zuse“ und ist vom Typ Forschungsschiff. Das zweite Schiff heißt „Marie“ und hat einen unbekannten Typ.

6.2.3 Modelltransformation

Eine Modelltransformation ist die Übersetzung einer Modellinstanz von einem Modell in ein anderes. Die Modelltransformation muss die semantischen Zusammenhänge beider Modelle berücksichtigen und die Informationen aus dem Ausgangsmodell in das Zielmodell übertragen. Eine Transformation kann nur verlustfrei geschehen, solange die Information aus dem Ausgangsmodell im Zielmodell dargestellt werden kann (vgl. [MT04], S. 623).

XSLT ist Teil des Ökosystems um XML und ist eine Sprache zur Transformierung von XML in andere Formate (vgl. [Mel11], S. 155). Die polymorphe Schnittstelle nutzt XML für die Repräsentation von Modellinstanzen. Somit bietet sich XSLT an, um eine XML-Modellinstanz des Ausgangsmodells zu einer XML-Modellinstanz des Zielmodells zu transformieren. XSLT-Programme sind wiederum XML-Dokumente und werden von einer Engine ausgeführt. Zum Auffinden und Abfragen von bestimmten Informationen aus dem Quelldokument werden XPath und XQuery verwendet (vgl. [Mel11], S. 155).

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
3    <xsl:template match="/">
4      <ships>
5        <xsl:apply-templates />
```

```
6     </ships>
7   </xsl:template>
8   <xsl:template match="vessel">
9     <ship>
10      <xsl:apply-templates />
11    </ship>
12  </xsl:template>
13  <xsl:template match="name">
14    <name><xsl:value-of select="." /></name>
15  </xsl:template>
16  <xsl:template match="type">
17    <type>
18      <xsl:choose>
19        <xsl:when test=". = 'FISHING_SMALL'">FISHING_SHIP</xsl:when>
20        <xsl:when test=". = 'FISHING_NORMAL'">FISHING_SHIP</xsl:when>
21        <xsl:when test=". = 'FISHING_LARGE'">FISHING_SHIP</xsl:when>
22        <xsl:when test=". = 'SAILING_SMALL'">SAILING_SHIP</xsl:when>
23        <xsl:when test=". = 'SAILING_NORMAL'">SAILING_SHIP</xsl:when>
24        <xsl:when test=". = 'SAILING_LARGE'">SAILING_SHIP</xsl:when>
25        <xsl:when test=". = 'RESEARCH_SMALL'">RESEARCH_SHIP</xsl:when>
26        <xsl:when test=". = 'RESEARCH_NORMAL'">RESEARCH_SHIP</xsl:when>
27        <xsl:when test=". = 'RESEARCH_LARGE'">RESEARCH_SHIP</xsl:when>
28        <xsl:otherwise>UNKNOWN</xsl:otherwise>
29      </xsl:choose>
30    </type>
31  </xsl:template>
32 </xsl:transform>
```

Auflistung 3: Beispiel einer Modelltransformation einer Liste von Schiffen als XSL-Transformation

Auflistung 3 zeigt ein Beispiel einer XSL-Transformation, welche eine Liste von Schiffen von einem Modell in ein anderes überführt. Die XSLT beinhaltet mehrere Templates, die aufeinander aufbauen und den Inhalt der Quellmodellinstanz durchsuchen und das Zieldokument mit den transformierten Informationen füllen. Im ersten Schritt wird das Hauptelement <ships> erstellt. Für jedes Element <vessel> wird danach ein Element <ship> angelegt. Der Name des Schiffes wird in derselben Form übernommen. Bei der Transformation des Schiffstyps kommt es zu einer nicht verlustfreien Umwandlung: Das Quellmodell sieht für jeden Schiffstypen drei verschiedene Größen vor, die in dem Zielmodell nicht vorhanden sind. Deshalb werden zum Beispiel die Typen

FISHING_SMALL, FISHING_NORMAL und FISHING_LARGE zu dem Typ FISHING_SHIP transformiert.

Die polymorphe Schnittstelle erlaubt durch die Nutzung von XSLT die Behandlung verschiedener Konflikte der zu transformierenden Modelle. Diese Konflikte können in folgende Typen eingeteilt werden (vgl. [MT04], S. 624):

- Verschiedene Formate oder Einheiten von Werten
- Verschiedene Datentypen
- Unterschiedliche Bezeichnungen derselben Information
- Abweichende Strukturierung derselben Information

6.2.4 Connector

In den vorherigen Abschnitten wurde beschrieben, wie die polymorphe Schnittstelle Modellinstanzen transformiert. Damit die polymorphe Schnittstelle innerhalb einer IT-Infrastruktur eingesetzt werden kann, muss diese die Modellinstanzen von anderen Systemen empfangen können. Die transformierten Modellinstanzen wiederum müssen an andere Systeme gesendet werden können.

Die Modellinstanzen werden in Form von XML-Dokumenten dargestellt (siehe Abschnitt 6.2.2). Die eingehenden Daten von anderen Systemen können in beliebigen Formaten vorliegen und müssen daher vor der Modelltransformation zunächst in Modellinstanzen übersetzt werden. Vor dem Versenden an andere Systeme müssen die Modellinstanzen wiederum in die gewünschten Datenformate übersetzt werden.

Die polymorphe Schnittstelle muss somit Daten empfangen, versenden und in Modellinstanzen übersetzen können. Damit die polymorphe Schnittstelle flexibel in einer IT-Infrastruktur eingesetzt werden kann, muss diese an die unterschiedlichsten Protokolle und Datenformate angebunden werden können. Für diese Einbindung werden die sogenannten Connectors verwendet. Connectors sind Erweiterungen für die polymorphe Schnittstelle, die für die Integration in das Systemumfeld implementiert werden können.

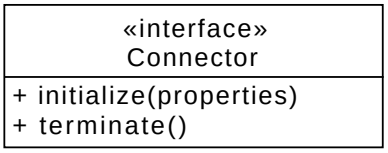


Abbildung 13: Schnittstelle eines Connectors in der polymorphen Schnittstelle

In der Abbildung 13 ist die Schnittstelle dargestellt, die jeder Connector implementieren muss. Ein Connector kann über Einstellungen konfiguriert werden. Auf diese Weise kann dieselbe Implementierung eines Connectors für verschiedene Zwecke eingesetzt werden. Beim Starten der polymorphen Schnittstelle wird jede Instanz eines Connectors mit seinen Einstellungen initialisiert. Hierfür wird die $\mathrm{initialize}(\mathrm{properties})$-Methode aufgerufen. Beim Beenden der polymorphen Schnittstelle wird jede Instanz eines Connectors terminiert. Hierfür wird die $\mathrm{terminate}()$-Methode aufgerufen.

Connector ist der allgemeine Oberbegriff für Erweiterungen der polymorphen Schnittstelle. Es gibt zwei konkrete Arten von Connectors: Adapter und Translator. Beide erweitern die Schnittstelle eines Connectors um weitere Methoden. In den folgenden Abschnitten werden diese näher erläutert.

6.2.5 Adapter

Adapter sind Connectors für die polymorphe Schnittstelle, die für die Kommunikation mit anderen Systemen zuständig sind. Über Adapter kann die polymorphe Schnittstelle Daten empfangen und versenden. Für das Empfangen von Daten werden $\mathrm{InputAdapter}$ verwendet. Für das Senden von Daten werden $\mathrm{OutputAdapter}$ verwendet.

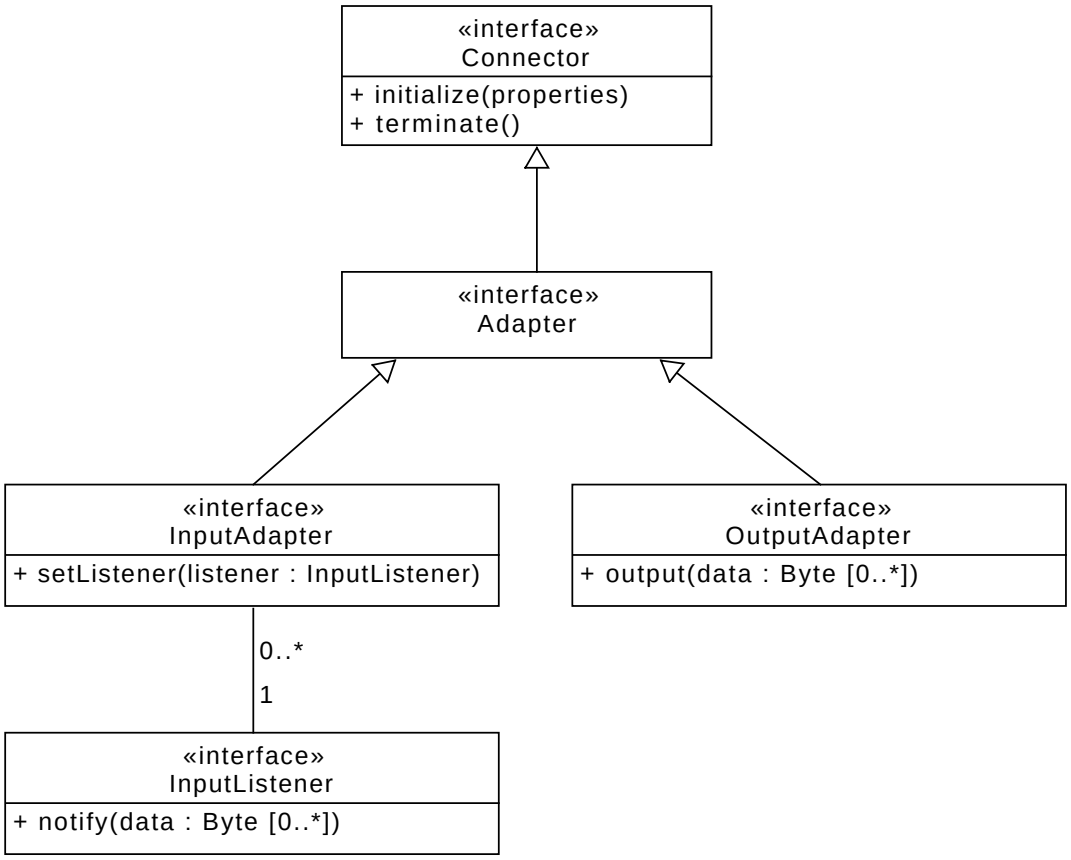


Abbildung 14: Schnittstelle eines Adapters in der polymorphen Schnittstelle

In Abbildung 14 sind die Schnittstellen dargestellt, die von einem $\mathrm{InputAdapter}$ oder einem $\mathrm{OutputAdapter}$ implementiert werden müssen. Jeder Adapter muss alle Methoden eines Connectors implementieren (siehe Abschnitt 6.2.4). Im Folgenden werden die zusätzlichen Methoden beschrieben, die ein $\mathrm{InputAdapter}$ bzw. ein $\mathrm{OutputAdapter}$ implementieren muss.

InputAdapter

Jeder $\mathrm{InputAdapter}$ muss die Methode $\mathrm{setListener(listener : InputListener)}$ implementieren. Nachdem die polymorphe Schnittstelle die Instanz eines $\mathrm{InputAdapters}$ initialisiert hat, wird der Instanz ein $\mathrm{InputListener}$ zugewiesen. Dafür wird die Methode $\mathrm{setListener(listener : InputListener)}$ des $\mathrm{InputAdapters}$ verwendet. Ein $\mathrm{InputAdapter}$ überträgt alle Daten, die er empfängt, an den ihm zugewiesenen $\mathrm{InputListener}$. Dafür wird die Methode $\mathrm{notify(data : Byte\ [0..*])}$ des $\mathrm{InputListeners}$ verwendet. Die empfangenen

Daten werden als Bytes an den InputListener übergeben. Die Instanzen der InputListener werden von der polymorphen Schnittstelle verwaltet, um die eingehenden Daten der InputAdapter verarbeiten zu können.

Die Methode notify(data : Byte [0..*]) eines InputListeners muss threadsicher implementiert sein. Dies ermöglicht einem InputAdapter, mehrere Threads zu verwenden, um Daten zu empfangen und an einen InputListener zu senden, ohne zusätzliche Synchronisierungslogik zu benötigen. Durch die garantierte Threadsicherheit kann ein InputAdapter einfacher und performanter implementiert werden.

OutputAdapter

Jeder OutputAdapter muss die Methode output(data : Byte [0..*]) implementieren. Die polymorphe Schnittstelle ruft diese Methode der Instanz eines OutputAdapters auf, um Daten an ein anderes System zu senden. Die Daten werden als Bytes an den OutputAdapter übergeben.

Die Methode output(data : Byte [0..*]) eines OutputAdapters muss threadsicher implementiert sein. Dies ermöglicht es der polymorphen Schnittstelle, mehrere Threads zu verwenden, um Daten über dieselbe Instanz eines OutputAdapters zu versenden, ohne zusätzliche Synchronisierungslogik zu benötigen.

6.2.6 Translator

Translator sind Connectors für die polymorphe Schnittstelle, die für das Übersetzen von Daten in Modellinstanzen zuständig sind. Ein InputTranslator übersetzt die Daten, die über einen InputAdapter empfangen wurden, in Modellinstanzen. Ein OutputTranslator übersetzt die Modellinstanzen in Daten, die über einen OutputAdapter versendet werden sollen. Die Daten der Adapter liegen als Bytes vor. Die Modellinstanzen werden in Form von XML-Dokumenten repräsentiert.

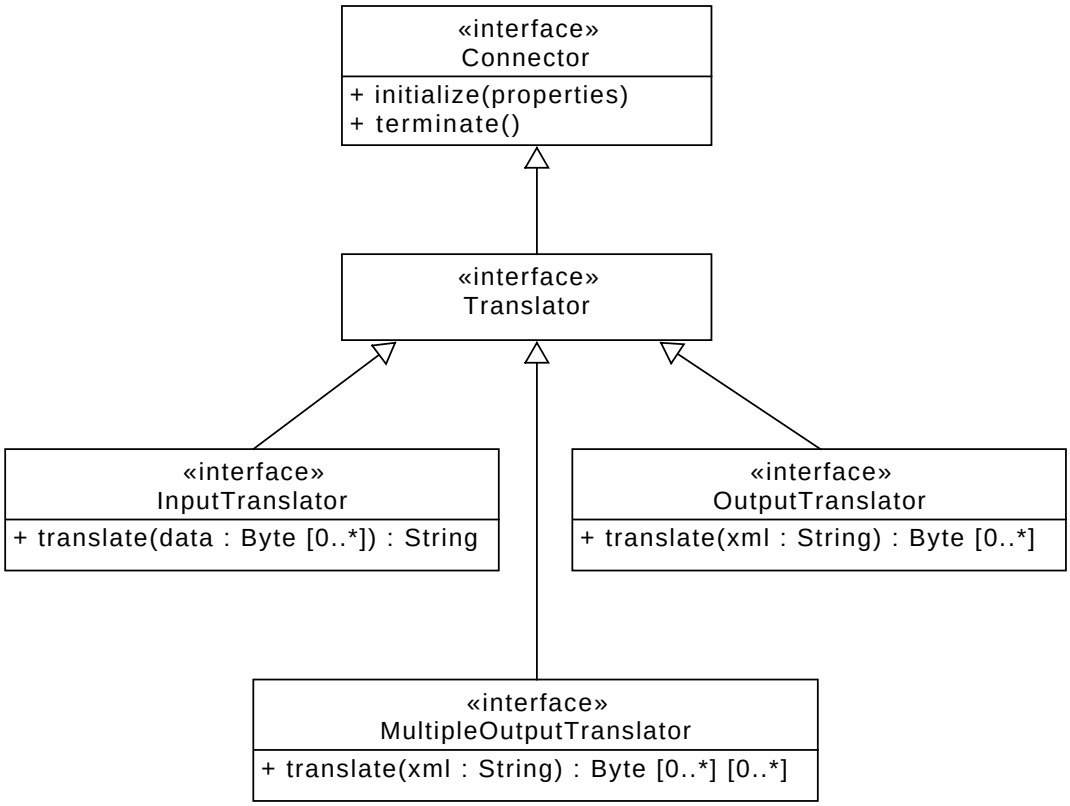


Abbildung 15: Schnittstelle eines Translators in der polymorphen Schnittstelle

In Abbildung 15 sind die Schnittstellen dargestellt, die von einem InputTranslator oder einem OutputTranslator implementiert werden müssen. Jeder Translator muss alle Methoden eines Connectors implementieren (siehe Abschnitt 6.2.4). Im Folgenden werden die zusätzlichen Methoden beschrieben, die ein InputTranslator bzw. ein OutputTranslator implementieren muss.

InputTranslator

Jeder InputTranslator muss die Methode translate(data : Byte [0..*] : XML) implementieren. Nachdem die polymorphe Schnittstelle über einen InputAdapter Daten empfangen hat, wird diese Methode eines InputTranslators aufgerufen, um aus den Bytes eine Modellinstanz zu erzeugen. Diese Modellinstanz wird in Form eines XML-Dokumentes zurückgegeben.

Die Methode translate(data : Byte [0..*] : XML) eines InputTranslators muss threadsicher implementiert sein. Dies ermöglicht es der polymorphen Schnittstelle, meh-

rere Threads zu verwenden, um Daten über dieselbe Instanz eines InputTranslators in Modellinstanzen zu übersetzen, ohne zusätzliche Synchronisierungslogik zu benötigen.

OutputTranslator

Jeder OutputTranslator muss die Methode translate(xml : String) : Byte [0..*] implementieren. Bevor die polymorphe Schnittstelle eine Modellinstanz über einen OutputAdapter versenden kann, wird diese Methode eines OutputTranslators aufgerufen, um aus der Modellinstanz Bytes zu erzeugen. Die erzeugten Bytes können dann über einen OutputAdapter versendet werden.

Die Methode translate(xml : String) : Byte [0..*] muss threadsicher implementiert sein. Dies ermöglicht es der polymorphen Schnittstelle, mehrere Threads zu verwenden, um Modellinstanzen über dieselbe Instanz eines OutputTranslators in Bytes zu übersetzen, ohne zusätzliche Synchronisierungslogik zu benötigen.

MultipleOutputTranslator

Ein MultipleOutputTranslator ist ein OutputTranslator, der aus einer einzelnen Modellinstanz mehrere Datenpakete aus Bytes erzeugen kann. Ein normaler OutputTranslator kann aus einer einzelnen Modellinstanz nur ein einzelnes Datenpaket aus Bytes erzeugen. Jedes Datenpaket wird einzeln über einen OutputAdapter versendet.

Jeder MultipleOutputTranslator muss die Methode translate(xml : String) : Byte [0..*] [0..*] implementieren. Diese Methode wird von der polymorphen Schnittstelle genauso wie bei einem OutputTranslator dazu verwendet, um Modellinstanzen vor dem Versenden über einen OutputAdapter in Bytes zu übersetzen.

Die Methode translate(xml : String) : Byte [0..*] [0..*] muss threadsicher implementiert sein. Dies ermöglicht es der polymorphen Schnittstelle, mehrere Threads zu verwenden, um Modellinstanzen über dieselbe Instanz eines MultipleOutputTranslator in Bytes zu übersetzen, ohne zusätzliche Synchronisierungslogik zu benötigen.

6.2.7 Pipe

Die polymorphe Schnittstelle empfängt Daten und übersetzt diese in Modellinstanzen. Die Modellinstanzen werden ineinander transformiert. Letztendlich werden die Modellinstanzen wieder in Daten übersetzt und versendet. All diese Schritte werden in einer Pipe

zusammengefasst. Innerhalb einer Pipe wird abgebildet, wie die polymorphe Schnittstelle eingehende Daten verarbeitet und das Ergebnis ausgibt. Die polymorphe Schnittstelle kann aus beliebig vielen Pipes bestehen.

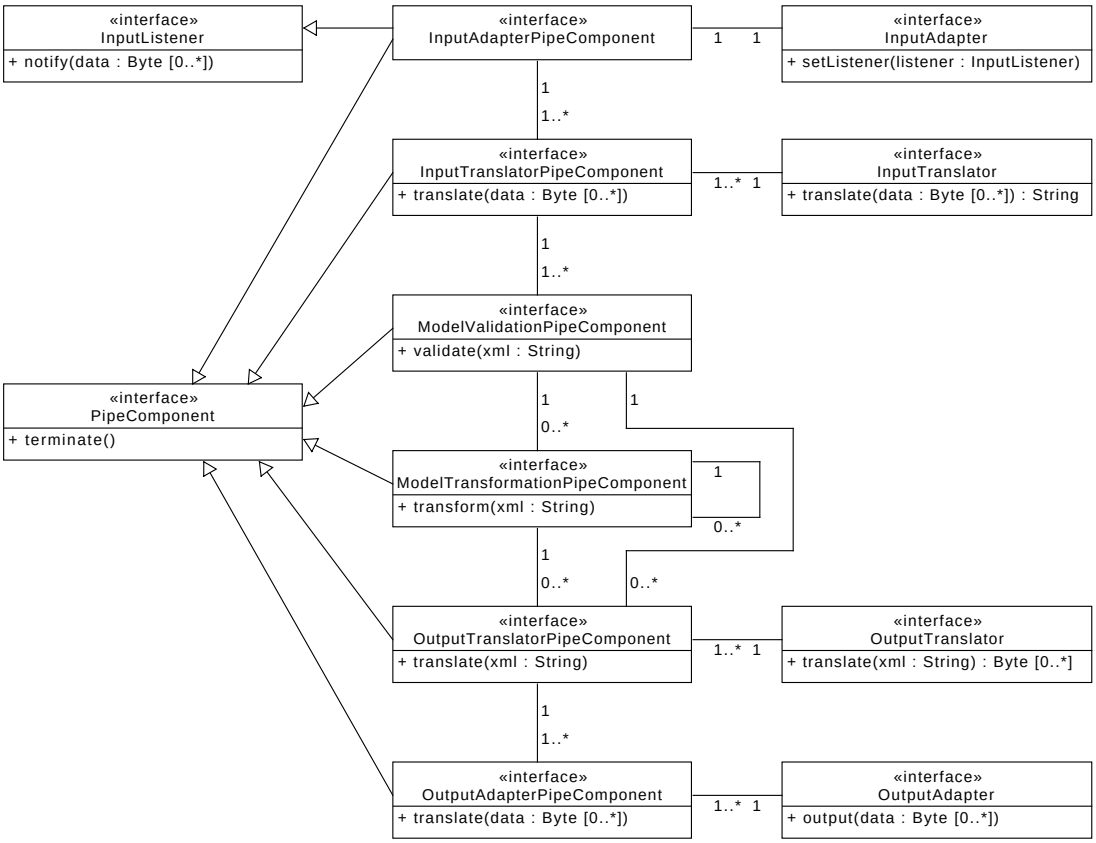


Abbildung 16: Aufbau einer Pipe in der polymorphen Schnittstelle

In Abbildung 16 ist der Aufbau einer Pipe dargestellt. Eine Pipe besteht aus mehreren PipeComponents, die im Folgenden näher erläutert werden. Jede PipeComponent muss die Methode terminate() implementieren, über welche die polymorphe Schnittstelle eine Pipe schrittweise beenden kann.

InputAdapterPipeComponent

Eine InputAdapterPipeComponent verarbeitet die Daten, die von einem InputAdapter empfangen werden. Dafür implementiert diese Komponente die Schnittstelle eines InputListeners. Einer InputAdapterPipeComponent ist genau eine Instanz eines Inpu-

tAdapters zugeordnet und andersherum. Die empfangenen Daten werden an eine oder mehrere InputTranslatorPipeComponents weitergegeben.

InputTranslatorPipeComponent

Eine InputTranslatorPipeComponent übersetzt die Daten einer InputAdapterPipeComponent mit Hilfe der Instanz eines InputTranslators in Modellinstanzen. Dafür wird die Methode translate(data : Byte [0..*]) : String des InputTranslators aufgerufen. Die Instanz eines InputTranslators kann von mehreren InputTranslatorPipeComponents verwendet werden. Die erzeugten Modellinstanzen werden an eine oder mehrere ModelValidationPipeComponents weitergegeben.

ModelValidationPipeComponent

Eine ModelValidationPipeComponent validiert die Modellinstanzen, die von einer InputTranslatorPipeComponent erzeugt wurden, gegen das dazugehörige Modell. Jede Modellinstanz muss gemäß dem dazugehörigen Modell aufgebaut sein (siehe Abschnitt 6.2.1 und Abschnitt 6.2.2). Jede ModelValidationPipeComponent ist genau einem Modell zugeordnet. Modellinstanzen, die nicht valide sind, werden von dieser Komponente verworfen. Alle validen Modellinstanzen werden an keine, eine oder mehrere ModelTransformationPipeComponents und an keine, eine oder an mehrere OutputTranslatorPipeComponents weitergegeben. Jeder ModelValidationPipeComponent muss mindestens eine ModelTransformationPipeComponent oder OutputTranslatorPipeComponent zugeordnet sein, an die valide Modellinstanzen weitergegeben werden.

ModelTransformationPipeComponent

Eine ModelTransformationPipeComponent transformiert Modellinstanzen in andere Modellinstanzen. Dieser Komponente sind immer genau zwei Modelle zugeordnet: Ein Ausgangsmodell und ein Zielmodell. Die eingehenden Modellinstanzen im Ausgangsmodell werden in Modellinstanzen im Zielmodell transformiert. Alle transformierten Modellinstanzen werden an keine, eine oder an mehrere weitere ModelTransformationPipeComponents und an keine, eine oder an mehrere OutputTranslatorPipeComponents weitergegeben. Jeder ModelTransformationPipeComponent muss mindestens eine ModelTransformationPipeComponent oder OutputTranslatorPipeComponent zugeordnet sein, an die transformierte Modellinstanzen weitergegeben werden.

OutputTranslatorPipeComponent

Eine OutputTranslatorPipeComponent übersetzt die Modellinstanzen einer ModelValidationPipeComponent oder einer ModelTransformationPipeComponent mit Hilfe der Instanz eines OutputTranslators in Daten, die von einem OutputAdapter versendet werden können. Dafür wird die Methode translate(xml : String) : Byte [0..*] des OutputTranslators aufgerufen. Die Instanz eines OutputTranslators kann von mehreren OutputTranslatorPipeComponents verwendet werden. Die erzeugten Daten werden an eine oder mehrere OutputAdapterPipeComponents weitergegeben.

OutputAdapterPipeComponent

Eine OutputAdapterPipeComponent nimmt die Daten einer OutputTranslatorPipeComponent entgegen und sendet diese mit Hilfe der Instanz eines OutputAdapters an andere Systeme. Dafür wird die Methode output(data : Byte [0..*]) des OutputAdapters aufgerufen. Die Instanz eines OutputAdapters kann von mehreren OutputAdapterPipeComponents verwendet werden. Eine OutputAdapterPipeComponent ist die letzte Komponente einer Pipe in der polymorphen Schnittstelle.

6.2.8 Laufzeitumgebung

Im vorherigen Abschnitt wurde beschrieben, wie die Datenverarbeitung innerhalb der polymorphen Schnittstelle mit Hilfe von Pipes definiert wird. Die Pipes der polymorphen Schnittstelle sollen flexibel konfiguriert werden können. Die polymorphe Schnittstelle soll daher in Form einer Laufzeitumgebung umgesetzt sein, in der die konfigurierten Pipes ausgeführt werden. Die Laufzeitumgebung ist für die Initialisierung, Verwaltung und Beendigung der Pipes und Connectors zuständig. Innerhalb der Pipes führt die Laufzeitumgebung die konfigurierten Modelltransformationen durch.

Konfiguration

Die Konfiguration der polymorphen Schnittstelle besteht aus drei Teilen: Die Modelle mit den Transformationsregeln, die Connectors und die Pipes. Im Folgenden werden diese drei Bestandteile der Konfiguration näher erläutert.

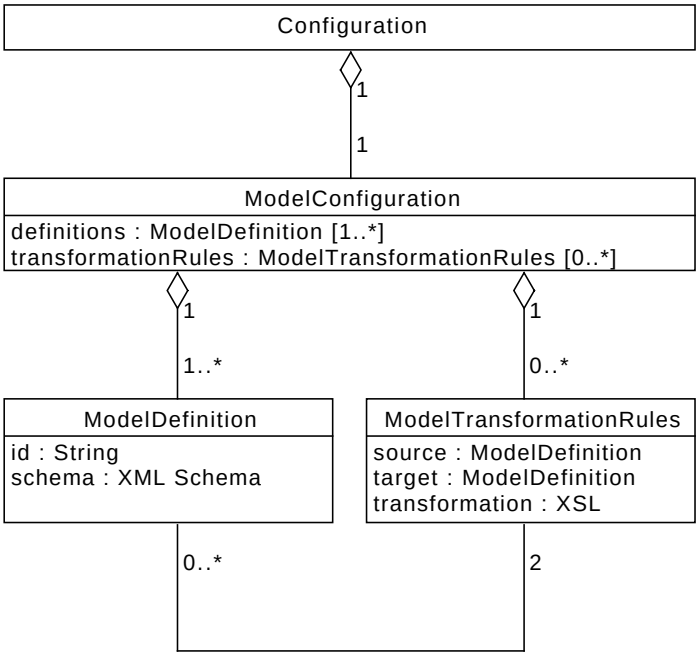


Abbildung 17: Konfiguration der Modelle der polymorphen Schnittstelle

In Abbildung 17 ist die Konfiguration der Modelle mit den Transformationsregeln dargestellt. Die ModelConfiguration besteht aus mindestens einer ModelDefinition und beliebig vielen ModelTransformationRules.

Jede ModelDefinition definiert ein Modell, das in der polymorphen Schnittstelle verwendet werden kann. Jedem Modell ist ein eindeutiger Bezeichner und ein XML Schema zugeordnet (siehe Abschnitt 6.2.1).

Allen ModelTransformationRules ist ein Ausgangsmodell und ein Zielmodell in Form einer ModelDefinition zugeordnet. Die ModelTransformationRules transformieren Modellinstanzen im Ausgangsmodell in Modellinstanzen im Zielmodell (siehe Abschnitt 6.2.3). Dafür ist allen ModelTransformationRules eine Menge an XSLT-Transformationsregeln zugeordnet. Diese Transformationsregeln werden von der polymorphen Schnittstelle interpretiert und für die Transformation von Modellinstanzen verwendet.

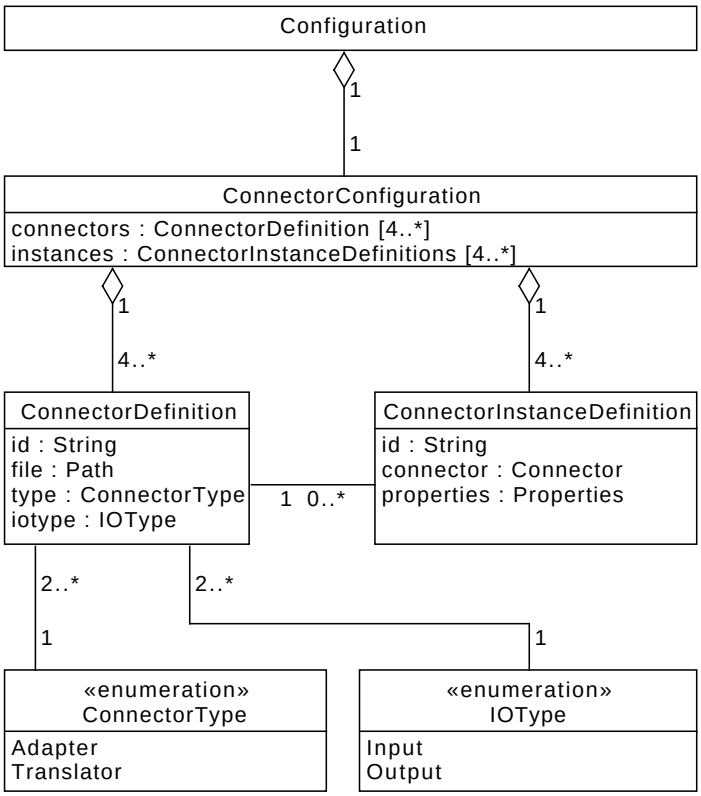


Abbildung 18: Konfiguration der Connectors der polymorphen Schnittstelle

In Abbildung 18 ist die Konfiguration der Connectors dargestellt. Die ConnectorConfiguration besteht aus mindestens vier ConnectorDefinitions und ConnectorInstanceDefinitions.

Jede ConnectorDefinition definiert einen Connector, der in der polymorphen Schnittstelle verwendet werden kann (siehe Abschnitt 6.2.4). Einer ConnectorDefinition ist ein eindeutiger Bezeichner und der Pfad zu der Datei zugeordnet, in der sich die Implementierung des Connectors befindet. Zusätzlich ist definiert, ob es sich um einen InputAdapter, OutputAdapter, InputTranslator oder um einen OutputTranslator handelt (siehe Abschnitt 6.2.5 und Abschnitt 6.2.6).

Jede ConnectorInstanceDefinition definiert die Instanz eines Connectors, die von der polymorphen Schnittstelle gestartet wird. Jeder ConnectorInstanceDefinition ist ein eindeutiger Bezeichner zugeordnet und basiert auf einer ConnectorDefinition. Zusätzlich können beliebig viele Einstellungen definiert werden, die bei der Initialisierung des Connectors an die Instanz übergeben werden (siehe Abschnitt 6.2.4).

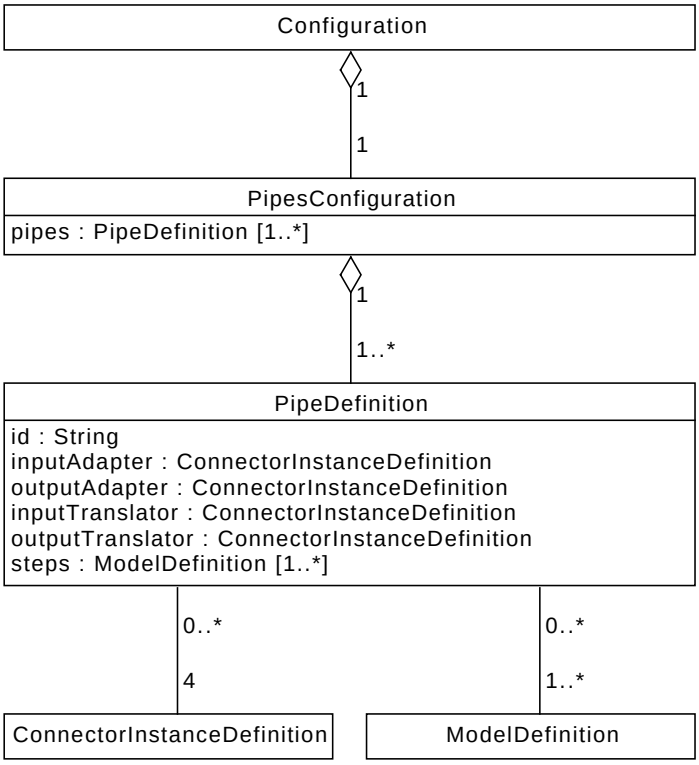


Abbildung 19: Konfiguration der Pipes der polymorphen Schnittstelle

In Abbildung 19 ist die Konfiguration der Pipes dargestellt. Die PipesConfiguration besteht aus mindestens einer PipeDefinition.

Jede PipeDefinition definiert eine Pipe, die von der Laufzeitumgebung der polymorphen Schnittstelle ausgeführt wird. Jeder PipeDefinition ist ein eindeutiger Bezeichner zugeordnet. Wie in Abschnitt 6.2.7 beschrieben, besteht jede Pipe aus einem InputAdapter, einem OutputAdapter, einem InputTranslator und einem OutputTranslator. Diese vier Connectors einer Pipe beziehen sich jeweils auf eine der zuvor konfigurierten ConnectorInstanceDefinitions.

Zusätzlich sind einer PipeDefinition eine oder mehrere ModelDefinitions in einer festen Reihenfolge zugeordnet. Dieses Tupel aus ModelDefinitions legt fest, in welcher Reihenfolge die Modellinstanzen innerhalb der Pipe in die verschiedenen Modelle transformiert werden sollen. Dabei werden die zuvor konfigurierten ModelTransformationRules verwendet.

Starten und Beenden

Beim Starten der polymorphen Schnittstelle wird zunächst die Konfiguration eingelesen und validiert. Dabei werden alle Modelle, Transformationsregeln und Connectors basierend auf der Konfiguration geladen und ebenfalls validiert.

Anschließend werden die konfigurierten Pipes soweit wie möglich zusammengeführt. Das bedeutet, dass Pipes, deren Anfänge übereinstimmen, zu einer Pipe kombiniert werden, die sich ab einer bestimmten Stelle aufspalten. Wenn beispielsweise zwei Pipes dieselben InputAdapter- und InputTranslator-Instanzen verwenden, aber unterschiedliche Modelltransformationen durchführen, werden diese zu einer Pipe zusammengeführt, die nach Ausführen des InputTranslators verschiedene Modelltransformationen ausführt. Diese Zusammenführung funktioniert auch bei übereinstimmenden Modelltransformationen und OutputTranslator-Instanzen. Wenn zwei Pipes exakt gleich sind, wird eine von beiden verworfen. Die Zusammenführung von Pipes verbessert die Performance der polymorphen Schnittstelle, da auf diese Weise identische Datenverarbeitungen nur einmal durchgeführt werden.

Nachdem die Pipe-Struktur erstellt worden ist, werden die Pipes und die enthaltenen Connector-Instanzen initialisiert. Dabei werden die Pipes von hinten nach vorne aufgebaut. Das bedeutet, dass zuerst die OutputAdapter und zuletzt die InputAdapter initialisiert werden. Sobald der InputAdapter einer Pipe initialisiert worden ist, ist die Pipe aktiv und kann eingehende Daten verarbeiten. Nachdem alle Pipes erfolgreich gestartet worden sind, gilt die polymorphe Schnittstelle als gestartet.

Wenn die polymorphe Schnittstelle beendet wird, werden zuerst die InputAdapter der Pipes terminiert. Dadurch können keine neuen Daten empfangen werden. Anschließend werden die Pipes von vorne nach hinten terminiert. Das bedeutet, dass zuerst die InputTranslator und zuletzt die OutputAdapter terminiert werden. Nachdem alle Pipes terminiert worden sind, gilt die polymorphe Schnittstelle als beendet.

6.2.9 Anbindung an das Testbed

In den vorherigen Abschnitten wurde der allgemeine Entwurf der polymorphen Schnittstelle beschrieben. Im Folgenden wird auf den Einsatz der Schnittstelle als Verbindungskomponente zwischen MATE Produkt und Testbed eingegangen.

Modelle und Transformationen

Für die polymorphe Schnittstelle werden die Modelle NMEA2000 und S-100 verwendet. Das MATE Produkt stellt eine Schnittstelle für NMEA2000 zur Verfügung. Das Testbed basiert auf S-100. Es sind zwei Transformationen nötig. Die erste Transformation wandelt die Schiffs- und Umgebungsistwerte von S-100 in NMEA2000 um. Die zweite Transformation wandelt die Steuerbefehle des MATE Produkts von NMEA2000 in S-100 um.

Adapter

Zwischen dem MATE Produkt und dem Testbed gibt es Unterschiede in der Formatierung der übertragenen Daten. Zusätzlich verwenden die Schnittstellen jeweils eine andere Art der Datenübertragung. Das MATE Produkt kommuniziert über UDP und das Testbed über das AMQP. Die polymorphe Schnittstelle muss demnach mit zwei Adapter-Paaren ausgestattet werden, welche die Protokolle unterstützen, um den Austausch von Daten zu ermöglichen. Es handelt sich um Adapter-Paare, da jeweils das Senden und Empfangen von Daten ermöglicht werden muss.

Translator

Zusätzlich zu den Adaptern werden für den Einsatz der polymorphen Schnittstelle zwei Translator-Paare benötigt. NMEA2000 und S-100 sind ursprünglich keine Modelle auf Basis von XML, sondern liegen als serialisierte Byte-Rohdaten vor. Die polymorphe Schnittstelle führt Transformationen jedoch mit Hilfe von XSLT durch, wobei die Modellinstanzen als XML-Dokumente vorliegen müssen. Aus diesem Grund werden NMEA2000 und S-100 Translator-Paare benötigt. Diese haben die Funktion aus Bytes des entsprechenden Standards die XML-Repräsentation zu erstellen und andersherum.

Pipes

Im Folgenden werden die konfigurierten Pipes mit den genutzten Modellen, Transformationen, Adapters und Translators beschrieben:

– Die erste Pipe nimmt S-100 Daten von dem RabbitMQ Server der Datenverteilungskomponente entgegen. Diese Daten werden mit dem S-100 Input Translator in ein XML-Dokument übersetzt. Es folgt die Modelltransformation von S-100 zu

NMEA2000. Die NMEA2000 Nachricht, die in XML vorliegt, wird in ihre Byte-Repräsentation durch den NMEA2000 Output Translator überführt. Zuletzt werden die Daten mit dem UDP Output Adapter an das MATE Produkt versendet.

- Die zweite Pipe ist der ersten Pipe ähnlich. Ausschließlich die Datenquelle ist eine andere: Es werden S-100 Nachrichten über das Internet von dem RabbitMQ des entfernten Datenverteilers entgegengenommen. Der weitere Ablauf ist wie bei der ersten Pipe.
- Die letzte benötigte Pipe ermöglicht die Übertragung der Steuerbefehle von dem MATE Produkt an den Datenverteiler. Dazu werden NMEA2000 Nachrichten von dem UDP Input Adapter entgegengenommen. Diese werden im nächsten Schritt mit dem NMEA2000 Input Adapter in ihre XML-Repräsentation übersetzt. Es folgt die Transformation von NMEA2000 zu S-100. Danach wird das entstandene S-100 XML-Dokument von dem S-100 Output Translator serialisiert. Zuletzt werden die Daten an eine Exchange-Schnittstelle des RabbitMQ Servers des Datenverteilers gesendet.

6.3 Verteilungsplattform

Bei der Verteilungsplattform handelt es sich um ein Modul, welches Daten der schiffsseitigen Komponenten entgegennimmt und an andere Komponenten verteilt. Umgesetzt werden soll dies durch eine zweite Instanz der Implementierung der polymorphen Schnittstelle aus dem vorherigen Abschnitt. Diese soll über mehrere Pipes und somit über mehrere Ausgänge und Eingänge verfügen. Die polymorphe Schnittstelle wird wiederverwendet, da ein Teil des Konzeptes übereinstimmt. Daten in unterschiedlichen Formaten sollen zwischen verschiedenen Komponenten ausgetauscht werden. Im Speziellen ist die Rede von folgenden Komponenten:

- Polymorphe Schnittstelle
- Pfadplanung
- Regelungssystem
- Kontrollplattform (mit vorgeschaltetem RabbitMQ-Server (siehe Abschnitt 6.3.5), zur Übertragung von der Schiffs- auf die Landseite)

6.3.1 Modelle und Transformationen

In der Verteilungsplattform werden die Modelle NMEA2000, NMEA0183 und RTZ verwendet. Die Verwendung der Modelle und die notwendigen Transformationsregeln werden in dem folgenden Abschnitt 6.3.4 näher beschrieben. Die Übersetzung der eingehenden Daten in die Modellinstanzen dieser Modelle wird in Abschnitt 6.3.2 und Abschnitt 6.3.3 beschrieben.

6.3.2 Adapter

Um die Kommunikation nach außen über mehrere Protokolle zu ermöglichen, benötigt die Verteilungsplattform mehrere Adapter. Diese regeln die Kommunikation mit dem Regelungssystem und dem Testbed über das UDP. Die Verbindung mit der Kontrollplattform und der Pfadplanung erfolgen über das Hypertext Transfer Protocol (HTTP). Der Verbindungstyp über HTTP soll über einen HTTP-Adapter umgesetzt werden. Mit diesem Adapter ist es möglich, neben einem verbindungslosen Datenaustausch mit UDP auch eine Art verbindungsorientierte Lösung über einen Request/Response (deutsch: Anfrage/Antwort) Ablauf nutzbar zu machen. Die anderen Adapter wurden bereits im Abschnitt 6.2 zur polymorphen Schnittstelle beschrieben, da sich die Verteilungsplattform mehrere Adapter mit dieser Komponente teilt. Insbesondere sind dies der RabbitMQ-Adapter und der UDP-Adapter.

6.3.3 Translator

Die Verteilungsplattform benötigt mehrere Übersetzer (englisch: Translator) für die Regelung der Kommunikation unter den verschiedenen Komponenten des PG MATE Produkts. Außerdem werden Übersetzer für der Kommunikation mit dem Testbed benötigt. Davon werden nachfolgend der NMEA0183 Translator und ein XML-String Translator beschrieben. Die anderen Translator wurden bereits im Abschnitt 6.2 zur polymorphen Schnittstelle beschrieben, da sich die Verteilungsplattform mehrere Translator mit dieser Komponente teilt.

XML-String Translator

In einigen Situationen kann es vorkommen, dass Daten bereits in einer XML Beschreibung vorliegen. Eine solche Situation ergibt sich beispielsweise bei Nachrichten im

RTZ-Format. Wie in Abschnitt 3.1.4 beschrieben, liegen diese Daten dann bereits im XML Format vor. Für solche Fälle ist kein weiteres Serialisieren oder Deserialisieren in oder von XML notwendig. Um diese Situationen ebenfalls in den in Abschnitt 6.3.4 beschriebenen Pipes einbinden zu können, soll ein XML-String Translator entwickelt werden. Dieser nimmt die im XML Format empfangenen Daten entgegen und leitet sie wieder zurück zum Server der Verteilungsplattform, welcher die Pipes handhabt.

NMEA0183-Translator

Aufgabe des NMEA0183-Translators soll es sein, NMEA0183-XML-Nachrichten in dem Standard entsprechende NMEA0183-Nachrichten zu konvertieren.

Folgende NMEA0183-Nachrichten sollen dabei konvertiert werden können:

NMEA0183 Nachrichten-ID	**Beschreibung (NMEA0183 Nachrichtenname)**
GLL	Geografische Position (Geographic Position)
VHW	Geschwindigkeit durchs Wasser und Peilung (Water speed and heading)
VTG	Kurs und Geschwindigkeit über Grund (Track made good and Ground speed)
RPM	Motordrehzahl (Revolutions)
RSA	Ruderwinkel (Rudder sensor angle)
HDT	Rechtweisender Steuerkurs (Heading - True)
ROT	Kursänderungsrate (Rate of turn)
MWV	Windgeschwindigkeit und -richtung (Wind Speed and Angle)
DPT	Wassertiefe (Depth)
VDM mit AIS Typ 1	AIS Report (AIS VHF Datalink (Own-vessel) Message)
VDM mit AIS Typ 5	AIS Report (AIS VHF Datalink (Own-vessel) Message)

Tabelle 23: Durch den NMEA0183-Translator zu konvertierende Nachrichten

Eine NMEA0183-Nachricht wird im ASCII-Format übertragen und besteht aus einer Bytefolge. NMEA0183-Nachrichten werden in Form von Sätzen übertragen.

Eine NMEA0183-XML-Nachricht besteht immer aus mindestens einem NMEA0183-Datensatz. Ein NMEA0183-Datensatz wiederum besteht aus mindestens einem Datenfeld. Dieses Datenfeld kann, muss aber nicht, einen dazugehörigen Datenwert enthalten. Im Folgenden wird eine beispielhafte NMEA0183-XML-Nachricht dargestellt:

```
1  <?xml version="1.0" encoding="UTF−8"?>
2  <NMEA0183>
3    <Message Sentence="M1">
4      <Field Name="F1" Value="V1"/>
5    </Message>
6    <Message Sentence="M2">
7      <Field Name="F2" Value="V2"/>
8      <Field Name="F3">
9        <Ais Type="1">
10         <Field Name="F4" Value="V4"/>
11         <Field Name="F5" Value="V5"/>
12       </Ais>
13     </Field>
14   </Message>
15 </NMEA0183>
```

Auflistung 4: Beispiel einer NMEA0183-XML-Nachricht

Zu Beginn bekommt der NMEA0183-Translator eine NMEA0183-XML-Nachricht übergeben. Um eine NMEA0183-XML-Nachricht in eine NMEA0183-Nachricht zu konvertieren benötigt der NMEA0183-Tranlator zunächst ein Datenmodell für die Abbildung der übergebenen Nachricht. Dieses Datenmodel muss die Elemente der zuvor beschriebenen NMEA0183-XML-Nachricht abbilden können. Die NMEA0183-XML-Nachricht muss im Anschluss von einem XML-Parser in eine Instanz dieses Datenmodells überführt werden.

Je nachdem welche Nachrichtensätze die NMEA0183-XML-Nachricht enthält, werden dann die jeweiligen Prozessoren zur den Nachrichtensätzen ausgeführt. Für jeden Nachrichtensatz der konvertiert werden soll, muss ein eigenständiger Prozessor zur Verfügung stehen. Alle Nachrichtensätze sollen nacheinander konvertiert und zu einer einzelnen NMEA0183-Nachricht zusammengefasst werden. Im Anschluss ist die Konvertierung abgeschlossen und die NMEA0183-Nachricht kann weiterverarbeitet werden.

6.3.4 Pipes

Im Folgenden werden die verschiedenen Pipes mitsamt der verwendeten Adapter und Translatoren beschrieben. Die Reihenfolge der Nennung in der Konfigurationsdatei spielt dabei keine Rolle:

- Die erste Pipe nimmt Sensordaten des Schiffs entgegen und leitet diese an die EPD weiter. Die Sensordaten werden von dem Testbed an diese Pipe gesendet. Die Sensordaten liegen im NMEA2000-Format vor. Die Pipe nimmt die Daten über einen UDP-Adapter entgegen und wandelt diese über den NMEA2000-Translator in Modellinstanzen um. Die NMEA2000-Modellinstanzen werden in NMEA0183-Modellinstanzen transformiert. Diese Modellinstanzen werden über den NMEA0183-Translator wieder in Daten umgewandelt und über den RabbitMQ-Adapter als AMQP-Paket an den RabbitMQ-Server gesendet. AMQP ist ein Protokoll zur Kommunikation zwischen Message-Brokern und wird von RabbitMQ verwendet.
- Die zweite Pipe nimmt manuelle Steuerdaten für das Schiff entgegen und leitet diese an das Regelungssystem weiter. Die Steuerdaten werden von der EPD über den RabbitMQ-Server an diese Pipe gesendet. Die Steuerdaten liegen im NMEA2000-Format als AMQP-Pakete vor. Die Pipe nimmt die Daten über einen RabbitMQ-Adapter entgegen und wandelt diese über den NMEA2000-Translator in Modellinstanzen um. Die NMEA2000-Modellinstanzen werden in diesem Fall nicht weiter transformiert. Dass die Daten in Modellinstanzen umgewandelt werden, obwohl diese nicht transformiert werden, ist für vollwertige Implementierung des Konzeptes notwendig. Das Konzept sieht einen modularen Aufbau vor, der ein beliebiges Format im Eingang in ein anderes beliebiges Format im Ausgang übersetzen kann. Die NMEA2000-Modellinstanzen werden über den NMEA2000-Translator wieder in Daten umgewandelt und über einen UDP-Adapter an das Regelungssystem gesendet.
- Die dritte Pipe nimmt Steuerdaten für das Schiff entgegen und leitet diese an die polymorphe Schnittstelle weiter. Es handelt sich um Steuerdaten sowohl für die manuelle Steuerung, als auch für das Abfahren einer Route. Die Steuerdaten werden vom Regelungssystem an diese Pipe gesendet. Die Sensordaten liegen im NMEA2000-Format vor. Die Pipe nimmt die Daten über einen UDP-Adapter entgegen und wandelt diese über den NMEA2000-Translator in Modellinstanzen um.

Die NMEA2000-Modellinstanzen werden in diesem Fall nicht weiter transformiert. Diese Modellinstanzen werden über den NMEA2000-Translator wieder in Daten umgewandelt und über einen UDP-Adapter an die polymorphe Schnittstelle gesendet.

– Die vierte Pipe nimmt Sensordaten des Schiffs entgegen und leitet diese an die Pfadplanungskomponente weiter. Die Sensordaten werden vom Testbed an diese Pipe gesendet. Die Sensordaten liegen im NMEA2000-Format vor. Die Pipe nimmt die Daten über einen UDP-Adapter entgegen und wandelt diese über den NMEA2000-Translator in Modellinstanzen um. Die NMEA2000-Modellinstanzen werden in NMEA0183-Modellinstanzen transformiert. Diese Modellinstanzen werden über den NMEA0183-Translator wieder in Daten umgewandelt und über einen HTTP-Adapter an die Pfadplanungskomponente gesendet.

– Die fünfte Pipe nimmt routenbezogene Daten, die für das teilautonome Abfahren einer Route benötigt werden, entgegen und leitet diese an das Regelungssystem weiter. Die routenbezogenen Daten werden von der Pfadplanungskomponente an diese Pipe gesendet. Die routenbezogenen Daten liegen im RTZ-Format als String vor. Die Pipe nimmt die Daten über einen HTTP-Adapter entgegen. Das RTZ Format ist XML daher wandelt ein XML-String-Translator die Daten in RTZ Modellinstanzen um. Die RTZ-Modellinstanzen werden in NMEA2000-Modellinstanzen transformiert. Diese Modellinstanzen werden über den NMEA2000-Translator wieder in Daten umgewandelt und über einen UDP-Adapter an das Regelungssystem gesendet.

– Die sechste Pipe nimmt routenbezogene Daten, die der Überwachung des teilautonomen Abfahrens einer Route dienen, entgegen und leitet diese an die EPD weiter. Die routenbezogenen Daten werden von der Pfadplanungskomponente an diese Pipe gesendet. Die routenbezogenen Daten liegen im RTZ-Format als String vor. Die Pipe nimmt die Daten über einen HTTP-Adapter entgegen und wandelt diese über den XML-String-Translator in Modellinstanzen um, da das RTZ Format auf XML basiert. Die RTZ-Modellinstanzen werden in diesem Fall nicht weiter transformiert. Diese Modellinstanzen werden über den XML-String-Translator wieder in Daten umgewandelt und über einen RabbitMQ-Adapter als AMQP-Paket an den RabbitMQ-Server gesendet.

– Die siebte Pipe nimmt Routendaten entgegen und leitet diese an die Pfadplanungkomponente weiter. Unter anderem wird diejenige Route übermittelt, die teilauto-

nom abgefahren werden soll. Aber auch inaktive Routen zum Beenden des teilautonomen Fahrens können hier übermittelt werden. Die Routendaten werden von der EPD über den RabbitMQ-Server an diese Pipe gesendet. Die Routendaten liegen im RTZ-Format als String vor. Die Pipe nimmt die Daten über einen RabbitMQ-Adapter entgegen und wandelt diese über den XML-String-Translator in Modellinstanzen um. Die daraus entstehenden RTZ-Modellinstanzen werden in diesem Fall nicht weiter transformiert. Diese Modellinstanzen werden über den XML-String-Translator wieder in Daten umgewandelt und über einen HTTP-Adapter an die Pfadplanungskomponente gesendet.

6.3.5 RabbitMQ-Server

Im Folgenden wird der Einsatz des Message Brokers „RabbitMQ“ näher beschrieben. Es wird darauf eingegangen, warum er verwendet wird und welche Möglichkeiten er bietet.

RabbitMQ ist die Implementierung eines Message Brokers (deutsch: Nachrichtenverteiler). Es ist eine Software, in der Warteschlangen (englisch: queues) definiert werden können, damit Anwendungen sich mit diesen verbinden und Nachrichten darüber austauschen können (vgl. [PSb]). Der RabbitMQ-Server wird benutzt, um Daten von schiffsseitigen Komponenten zur landseitigen EPD zu übertragen. Da RabbitMQ bereits im Testbed verwendet wird, hat sich die PG MATE dazu entschieden, eine zweite Instanz für diese Aufgabe einzusetzen.

Vorteile durch den Einsatz von RabbitMQ Der Inhalt einzelner Nachrichten spielt keine Rolle, jede beliebige Information kann in einer Nachricht übertragen werden. Dies ist optimal, da die Komponenten Nachrichten in verschiedenen Formaten austauschen. Die EPD empfängt zum Beispiel NMEA0183-Sensordaten und RTZ-Routenstatus bzw. sendet NMEA2000-Steuernachrichten und RTZ-Routeninformationen.

Die vorhandenen Rechenressourcen werden effektiv verwendet. Während eine Komponente neue Nachrichten erstellt, kann eine andere Komponente gleichzeitig Nachrichten verarbeiten.

Eine geringe Kopplung zwischen der Verteilungsplattform und der EPD ist gegeben, da die Komponenten nur über den Inhalt der Nachrichten kommunizieren. In welcher Programmiersprache die Komponenten die Nachrichten erstellen oder verarbeiten, ist für die jeweils andere Komponente unerheblich. Dadurch kann zum Beispiel der digitale Küsten-

leitstand in zukünftigen Szenarien ausgetauscht werden, ohne Änderungen am schiffsseitigen Produkt vornehmen zu müssen.

Ferner wäre es möglich, Nachrichten eines Produzenten an mehrere Empfänger zu verteilen. Dadurch könnten mehrere Schiffe von einem Küstenleitstand aus gesteuert werden. Dieses Szenario wurde im Rahmen dieser Projektgruppe allerdings nicht weiterverfolgt.

Funktionsweise Nachrichten werden nicht direkt auf einer Warteschlange (Queue) veröffentlicht, der Ersteller sendet sie zuerst an ein Vermittlungsregister (Exchange). Eine Exchange leitet die Nachrichten mit Hilfe von sogenannten „bindings“ und „routing keys“ zu verschiedenen Queues weiter. Ein binding ist dabei eine Verknüpfung, mit der eine Queue an eine Exchange gebunden wird. Ein routing key ist ein Nachrichtenattribut, anhand dessen die Exchange entscheidet, auf welche der verknüpften Queues die Nachricht veröffentlicht wird (vgl. [PSb]).

Während ihrer Generierung kann die Bestandszeit einer Exchange mit „durable“, „temporary“ oder „auto delete“ definiert werden. Durable bedeutet, dass die Exchange auch nach einem RabbitMQ-Serverneustart bestehen bleibt. Bei temporary ist dies nicht der Fall. Eine Exchange mit dem Parameterwert auto delete löscht sich von selbst, sobald alle bindings aufgehoben wurden (vgl. [PSb]).

In RabbitMQ gibt es die Möglichkeit zwischen vier Exchange-Typen zu wählen. Die Weiterleitung der Nachrichten unterscheidet sich je nach Art des bindings und der eingestellten Parameter (vgl. [PSb]).

- Direct Exchange wird verwendet, um Nachrichten auf der gleichen Exchange anhand eines einfachen String-Attributes (routing key) zu unterscheiden. Eine Nachricht wird auf der Queue veröffentlicht, deren binding exakt mit dem routing key der Nachricht übereinstimmt.
- Topic Exchange verwendet „routing pattern“, um Nachrichten auf einer oder mehreren Queues zu veröffentlichen. Das routing pattern wird beim binding angegeben und beinhaltet mehrere Begriffe oder Platzhalter, zum Beispiel „*.*.pg.mate“. Wird eine Nachricht auf die Exchange geladen, wird verglichen, ob der routing key dieser Nachricht mit Teilen des routing pattern übereinstimmt (zum Beispiel „uni.oldenburg.pg.mate“). Es können im Detail noch weitere Einstellungen vorgenommen werden, auf die an dieser Stelle nicht weiter eingegangen wird.

- Fanout Exchange kopiert eine Nachricht und leitet diese zu allen verknüpften Queues weiter, unabhängig von routing keys oder routing pattern. Verwendet wird dieser Typ zum Beispiel, wenn verschiedene Anwendungen, die jede auf eine eigene Queue hören, die gleiche Nachricht erhalten sollen.
- Headers Exchange ähnelt dem Topic Exchange, trifft die Entscheidung jedoch aufgrund von „header"-Werten anstatt von routing keys. Ein header besteht aus key und value. Nachricht und Exchange können mehrere header-Argumente beinhalten. In einer Exchange muss das Argument „x-match" mit den Werten „any" oder „all" definiert werden. Bei „all" müssen alle header-Argumente der Nachricht mit denen der Exchange übereinstimmen. Bei „any" genügt bereits eines. Auf weitere Einstellungsmöglichkeiten wird an dieser Stelle nicht im Detail eingegangen.
- Nachrichten die keiner Queue zugeordnet werden können, werden standardmäßig gelöscht. Alternativ kann die Nachricht auch zurück zum Sender geschickt werden.

Eine Möglichkeit den Empfang veralteter Daten zu beschränken ist eine Time-to-Live. Eine Nachricht bleibt maximal so lange in der Queue, wie im Parameter x-message-ttl angegeben (vgl. [PSd]).

Ferner gibt es in RabbitMQ die Möglichkeit, den Empfang von Nachrichten zu bestätigen. Hierfür wird zwischen acknowledged und unacknowledged messages unterschieden. Auf der Queue liegen erstmal nur unbestätigte Nachrichten. Sobald der Client jedoch eine Nachricht empfangen hat, sendet er eine Empfangsbestätigung für diese Nachricht auf die Queue. Mit dem Parameter basic_qos ist es möglich die Anzahl der unbestätigten Nachrichten zu begrenzen, die gleichzeitig von der Queue an den Client gesendet werden (vgl. [PSc]). Implementiert werden sollte dies in Kombination mit einer Time-to-Live als Performanz Optimierung. Werden zu viele Nachrichten gleichzeitig geschickt, kann es dazu kommen, dass die Nachrichten sich unbearbeitet im Client ansammeln. Bis dieser sie abgearbeitet hat, können einige Sekunden vergehen. Dadurch kommt es wiederum dazu, dass Daten veraltet sein könnten.

6.4 Kontrollplattform

Für die Darstellung von Schiffsdaten und anderen Informationen, sowie der Eingabe von Steuerbefehlen, wird eine digitale Kontrollplattform benötigt. Dieses Unterkapitel beschreibt den Entwurf, sprich das Aufbaukonzept einer solchen Software, mit der alle erhobenen User Stories (siehe Abschnitt 5.2) umgesetzt werden.

Die Kontrollplattform erfüllt insbesondere drei zentrale Aufgaben:

- Er muss die Überwachung eines Seebereiches ermöglichen, wozu die Anzeige der Position und Fahrtrichtung aller Schiffe gehört, die sich in diesem Areal befinden. Diese und einige erweiterte Informationen werden insbesondere aus AIS-Daten gewonnen.
- Weiter wird, ähnlich einem ECDIS, eine Anzeige zum Monitoring schiffsspezifischer Sensordaten eines bestimmten Schiffes benötigt.
- Die dritte wichtige Anforderung ist eine Eingabemöglichkeit zur Steuerung eines bestimmten Schiffes.

6.4.1 Verwendung der EPD

Von Mitarbeitern des OFFIS wurde die Verwendung von der e-Navigation Prototype Displays (EPD) als Ausgangspunkt für die eigenen Entwicklungen empfohlen. Die EPD steht als freie Software unter einer Apache-Lizenz auf der Software-Entwicklungsplattform GitHub (vgl. [PVDH+17]) zur Verfügung. Hier kann nicht nur der Quellcode des Projekts heruntergeladen werden, sondern es werden über die Weboberfläche auch eine Kurzbeschreibung als README.md und umfangreiche Statistiken über das Git-Repository angeboten.

Entstanden ist die EPD als Projekt der Dänischen Schifffahrtsbehörde (dän. Søfartsstyrelsen), wobei die ersten Git-Commits des öffentlichen Repositorys auf Februar 2013 zurückgehen. Zu der Software haben hauptsächlich P. O. Pedersen, Janus Varmarken, Adam Due Hansen, Jens Tuxen und David Camre beigetragen. Im Moment werden dort jedoch keine weiteren Änderungen am Quelltext vorgenommen, der letzte Commit stammt aus dem Juni 2015. Dieser Stand wird im folgenden Usprungs-EPD genannt.

Unabhängig vom öffentlichen GitHub-Repository ist die EPD aber nach wie vor die Grundlage verschiedener Forschungsprojekte zur computergestützten Navigation auf See und wird beispielsweise auch im Rahmen des eMIR-Projekts (vgl. [eMI17]) im OFFIS genutzt und intern weiterentwickelt. Das OFFIS kann der Projektgruppe einen entsprechenden Stand als Git-Repository bereit stellen.

Ein weiterer Vorteil der EPD ist, dass sie in der Programmiersprache Java geschrieben ist, die alle Projektgruppenteilnehmer beherrschen.

6.4.2 EPD-API

Die Ursprungs-EPD bietet dem Benutzer eine Oberfläche und kann Routeneingaben von ihm entgegennehmen. Um die Anforderungen der Projektgruppe zu erfüllen, muss die Ursprungs-EPD an verschiedenen Stellen um zusätzliche Funktionalitäten erweitert werden. Die EPD-API bildet dabei das Kernelement zwischen der Ursprungs-EPD und den PG-Erweiterungen. Die Abbildung 20 zeigt die Struktur der EPD, bestehend aus dem bisherigen Stand und der EPD-API.

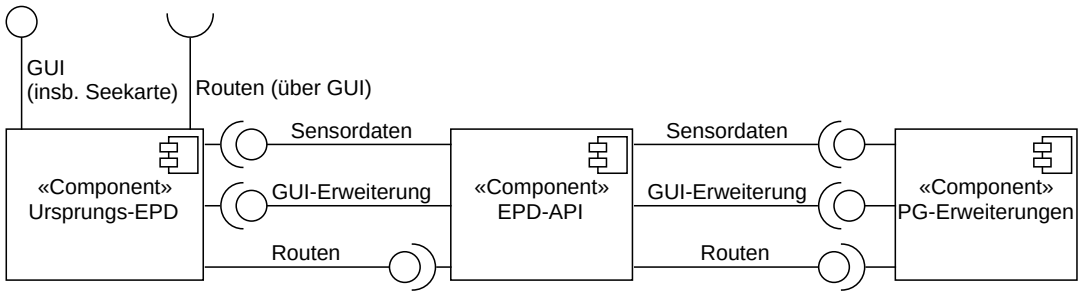


Abbildung 20: Die Struktur der EPD, bestehend aus dem bisherigen Stand und der EPD-API

In den beiden folgenden Abschnitten werden die PG-Erweiterungen näher beschrieben und erläutert, unter anderem auch welche Bedeutung die in Abbildung 20 dargestellten Schnittstellen „Sensordaten“, „GUI-Erweiterung“ und „Routen“ haben. Kernelement dieser Erweiterung ist die EPD-API. Diese soll ermöglichen, dass die EPD mit ihren bestehenden Funktionen erhalten bleibt und damit möglichst wenig Code verändert wird. Außerdem soll sie eine klare Abgrenzung des bestehenden Quelltextes zu neu hinzugefügtem Code ermöglichen und einzelne Funktionen voneinander kapseln. Durch diese Struktur können auch zukünftige Projektgruppen vereinfacht die EPD nutzen und erweitern.

Im Folgenden wird die Gesamtsoftware rund um die Kontrollplattform, sprich die ursprüngliche EPD mit unseren Erweiterungen, als EPD bezeichnet.

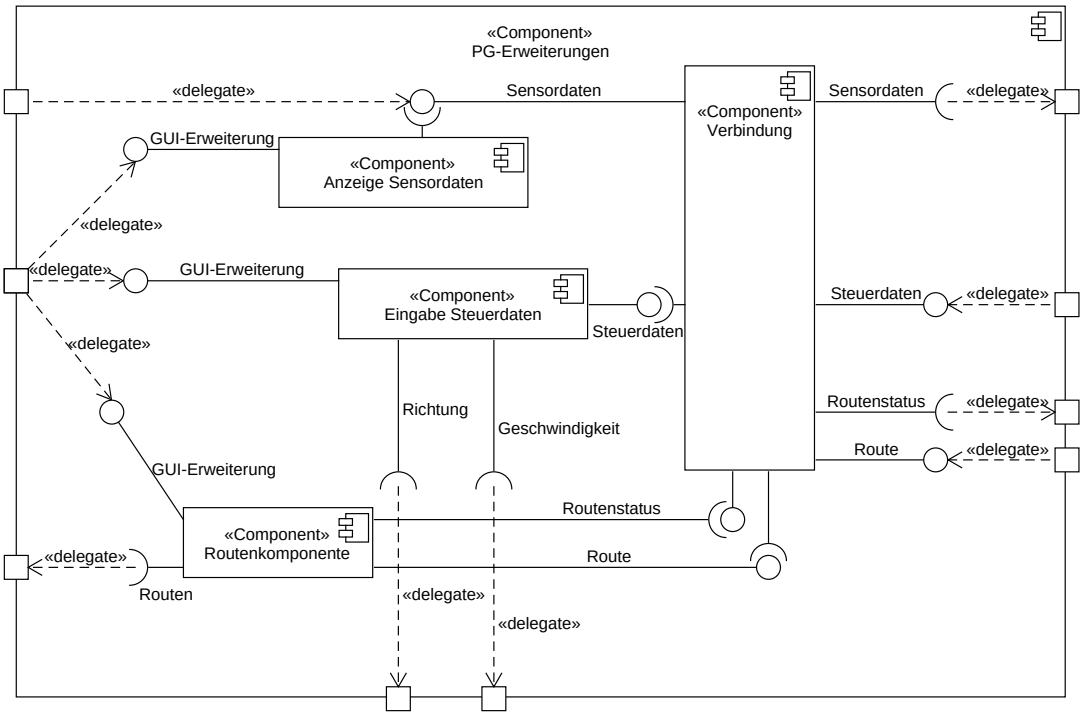


Abbildung 21: Struktur der Erweiterungskomponenten der EPD aus Meilenstein 1 und 2

6.4.3 Erweiterungen für den ersten Meilenstein

Für den ersten Meilenstein wird eine Schnittstelle benötigt, die schiffsspezifische und umweltbeschreibende Sensordaten von der Verteilungsplattform (siehe Abbildung 11) entgegennimmt und innerhalb der EPD an verschiedene Komponenten weiterleitet (siehe „Verbindung“ in Abbildung 21). Die ursprüngliche EPD kann bereits AIS-Daten verarbeiten und auf einer Karte darstellen. Dazu müssen die Daten im NMEA0183-Format vorliegen, was eine wichtige Anforderung an unsere Implementierung darstellt.

Zudem muss eine Komponente integriert werden, welche die schiffsspezifischen Sensordaten empfängt und geeignet darstellt (siehe „Anzeige Sensordaten“ in Abbildung 21), sowie eine weitere Komponente, welche die Eingabe von Steuerbefehlen ermöglicht und an die Verteilungsplattform weiterleitet (siehe „Eingabe Steuerdaten“ in Abbildung 21). Von OFFIS-Mitarbeitern hat die Projektgruppe Zugriff auf HTML- und JavaScript-Dateiern erhalten. Diese werden im Folgenden als „Virtual Handles“ (kurz VH) genannt. Mit den Virtual Handles ist es in der Implementierung möglich die beiden Komponenten „Anzeige Sensordaten“ und „Eingabe Steuerdaten“ auf einer Benutzeroberfläche zu vereinen.

6.4.4 Erweiterungen für den zweiten Meilenstein

Routen können bereits mit den vorhandenen Mitteln der EPD angelegt und bearbeitet werden. Für den zweiten Meilenstein wird jedoch eine neue Komponente benötigt, mit der ein Benutzer eine gespeicherte Route auswählen und an ein verbundenes Schiff senden kann. In diesem Zusammenhang wurden verschiedene Anforderungen aufgestellt, die beachtet werden sollten (siehe Tabelle 3). Darüber hinaus wird eine Schnittstelle benötigt, welche die aktive Route außerhalb der EPD für andere Komponenten des Produkts der PG MATE freigibt (siehe „Verteilungsplattform“ in Abbildung 11).

Eine weitere Schnittstelle muss vom Schiff kommende Informationen annehmen können. Informationen, die den aktuellen Status und aktive Wegpunkte beinhalten, müssen verarbeitet und dargestellt werden (siehe „Routen-Komponente“ in Abbildung 21). Dies impliziert auch das Stoppen des teilautonomen Fahrens.

Die Anzeige muss darüber hinaus dahingehend erweitert werden, dass das zu steuernde Schiff nicht mit anderen Schiffen verwechselt werden kann.

Die EPD soll dem Benutzer stets visuelles Feedback über die verwendeten Funktionen geben, damit schnell erfasst werden kann, ob die neuen Funktionen richtig konfiguriert wurden und problemlos verwendet werden können. Dieses visuelle Feedback soll dabei an das bestehende Look-and-Feel der EPD angelehnt werden.

In Abbildung 22 ist der gesamte Sollaufbau der EPD zu sehen. In dieser Abbildung sind die Inhalte aus Abbildung 20 und Abbildung 21 zusammengefasst.

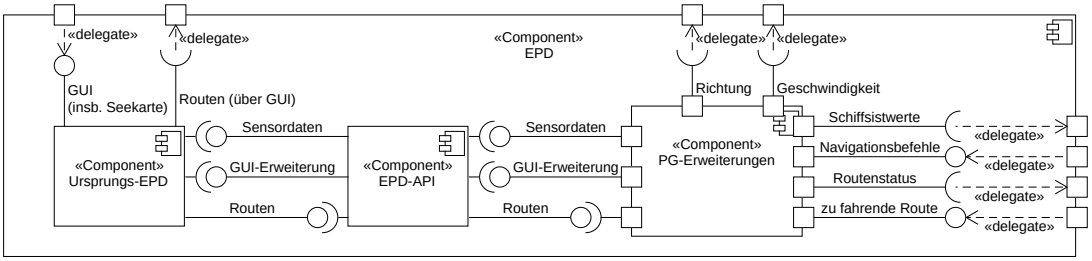


Abbildung 22: Gesamtstruktur der erweiterten EPD, bestehend aus bisherigem Stand, EPD-API und den PG-Erweiterungen

6.5 Pfadplanung

Die Pfadplanungskomponente kommt im Rahmen des automatischen Abfahrens von Routen durch einen Autopiloten, der in Meilenstein 2 umgesetzt wird, zum Einsatz. In An-

wendungsfall 11: „Autopilot aktivieren“ ist beschrieben, dass bei aktivierten Autopiloten das Schiff automatisch eine vorgegebene Route abfahren soll.

Aufgabe der Pfadplanungskomponente ist es, das automatische Abfahren von Routen im Bereich der Pfadplanung zu ermöglichen. Eine Route besteht aus zwei oder mehr Wegpunkten, die nacheinander abgefahren werden. Dabei sind stets zwei Wegpunkte der Route aktiv, die zusammen die aktuelle Strecke darstellen, die das Schiff abfahren soll. Auf Basis der aktuellen Route des Autopiloten und der aktuellen Position des Schiffes werden diese beiden aktiven Wegpunkte von der Pfadplanungskomponente berechnet. Die Route erhält die Pfadplanungskomponente von der Kontrollplattform. Die aktuelle Position stellen die schiffsseitigen Sensoren zur Verfügung. Die berechneten aktuellen Wegpunkte übertragt die Pfadplanungskomponente an das Regelungssystem. Auf diese Weise teilt die Pfadplanungskomponente dem Regelungssystem mit, welche Teilstrecke der Route aktuell abgefahren werden soll. Die dafür notwendigen Schnittstellen kommunizieren dabei allesamt über die Vermittlungsplattform mit den anderen Komponenten.

6.5.1 Definition von Route und Position

Eine Route ist ein Tupel von mindestens zwei Wegpunkten. Ein Wegpunkt besteht aus einer Position, die sich aus Längen- und Breitengrad zusammensetzt. Die aktuelle Position des Schiffes besteht aus Längen- und Breitengrad im selben Bezugssystem wie die Koordinaten der Wegpunkte. Außerdem enthält ein Wegpunkt die Angabe eines Radius um den Wegpunkt herum. Dieser Radius legt fest, ab wann ein Wegpunkt als erreicht gilt und der darauffolgende Wegpunkt angesteuert werden soll. Die Wegpunkte sollen in einer fest vorgegebenen Reihenfolge nacheinander abgefahren werden. Neben den Wegpunkten muss eine Route über ein eindeutiges Identifizierungsmerkmal wie eine ID oder einen Namen verfügen. Außerdem ist jeder Route ein Status zugewiesen. Dieser legt fest, ob die Route aktuell abgefahren werden soll oder inaktiv ist. Die Pfadplanungskomponente wird nur aktiv, wenn eine aktive Route vorhanden ist.

6.5.2 Schnittstellen der Pfadplanungskomponente

In Abbildung 23 sind die genutzten und bereitgestellten Schnittstellen der Pfadplanungskomponente abgebildet. Dies werden zusammen mit den zu nutzenden Nachrichtenstandards beschrieben.

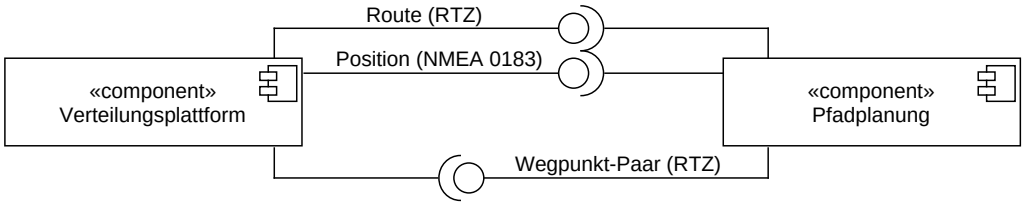


Abbildung 23: Schnittstellen der Pfadplanungskomponente

Die Ermittlung der aktuellen Wegpunkte soll dabei auf Basis einer Route und der aktuellen Position geschehen. Die Route erhält die Pfadplanungskomponente von der Kontrollplattform. Die Route wird mit Hilfe der Kontrollplattform durch einen Benutzer erstellt und aktiviert. Die Route gelangt über die Verteilungsplattform an die Pfadplanungskomponente. Die aktuelle Position erhält die Pfadplanungskomponente von den schiffsseitigen Sensoren. Diese übertragen die ermittelten Positionsdaten der Sensoren an den Datenverteiler des Schiffs. Dieses verarbeitet die Informationen und gibt diese an die schiffsseitige Instanz des Polymorphic-Interface weiter, welche die Daten wiederum mit Hilfe der Vermittlungsplattform an die Pfadplanungskomponente weitersendet (siehe Abschnitt 6.1.2).

Die Schnittstelle für das Empfangen einer Route muss ein Format vorgeben, in dem die Routen definiert werden. Dieses Format sollte einem bereits existierenden Standard entsprechen, der die oben beschriebenen Bestandteile einer Route unterstützt. Dieses Format muss auch von anderen Komponenten eingesetzt werden können. Die Verwendung eines vorhandenen Standards anstatt einer Eigenentwicklung erleichtert dies.

In anderen Komponenten werden bereits verschiedene maritime Standards verwendet. NMEA0183 und S-100 bieten keine entsprechenden Nachrichten zur Übertragung von Routen mit den definierten Anforderungen. NMEA2000 definiert einige Nachrichten mit denen Wegpunkte übertragen werden können. Jedoch können mit NMEA2000 keine Status für Routen festgelegt werden. Außerdem müssten die übrigen Informationen zu einer Route in mehrere NMEA2000 Nachrichten aufgeteilt werden.

Alternativ könnte ein eigenes Routen-Format in NMEA2000 oder S-100 definiert werden. Beide Standards ermöglichen die Erweiterung durch eigene Nachrichten. Dies führt jedoch zu dem Problem, dass die eigenen Nachrichten, da sie außerhalb des Standards definiert wurden, nicht mehr ohne Probleme von anderen Systemen verstanden werden

können. Damit eigenen sich diese Standards letztendlich nicht für das Übertragen von Routen.

Das Route plan Exchange Format hingegen ist ein maritimer Standard, der explizit für das Übertragen von Routen definiert worden ist (siehe Abschnitt 3.1.4). Das in diesem Standard beschriebene Format erfüllt alle oben beschriebenen Anforderungen an die Definition einer Route. Deshalb soll die Schnittstelle zum Austausch der Routen zwischen Kontrollplattform und Pfadplanung das RTZ-Format verwenden.

Auch die Schnittstelle für die aktuelle Position des Schiffs benötigt einen fest definierten Standard. Die Verteilungsplattform stellt der Kontrollplattform die aktuelle Position des Schiffs bereits im NMEA0183 Format zur Verfügung. Aus diesem Grund soll dieser Standard auch für die Pfadplanungskomponente verwendet werden.

Die errechneten aktuellen Wegpunkte werden zusammen mit dem Status der Route über die Vermittlungsplattform an die Regelungssystemkomponente zur Weiterverarbeitung übergeben. Das Regelungssystem ermittelt die Steuerbefehle für das Schiff, damit die Wegpunkte korrekt angesteuert werden können. Außerdem werden die aktiven Wegpunkte über die Vermittlungsplattform zurück an die Kontrollplattform übergeben, damit diese dem Benutzer angezeigt werden können. Die Übertragung der beiden Wegpunkte erfolgt zusammen mit dem Status der Route über die dafür vorgesehene Schnittstelle in einer RTZ-Nachricht.

6.5.3 Aufbau der Pfadplanungskomponente

Um auf Basis einer Route und der aktuellen Position die aktuellen Wegpunkte zu berechnen, muss die Pfadplanungskomponente die aktive Route sowie die aktuelle Position des Schiffs aufnehmen und speichern können.

Beim Start des Autopiloten werden die ersten beiden Wegpunkte einer Route als aktuelle Wegpunkte ausgewählt und zusammen mit dem Status der Route an die Schnittstelle der Vermittlungsplattform übergeben. Der aktuelle Status einer Route ist dabei in der übergebenen Route enthalten. Bei den aktuellen Wegpunkten handelt es sich immer um genau zwei Wegpunkte. Die beiden aktuellen Wegpunkte sind als Wegpunkte-Paar in der Pfadplanungskomponente zur Laufzeit zu speichern. Die Parameterstatussubkomponente muss damit die aktive Route, die aktuelle Position des Schiffs und das Wegpunkte-Paar speichern können. Abbildung 24 zeigt den Aufbau der Pfadplanungskomponente mit der Parameterstatussubkomponente und den weiteren Subkomponenten.

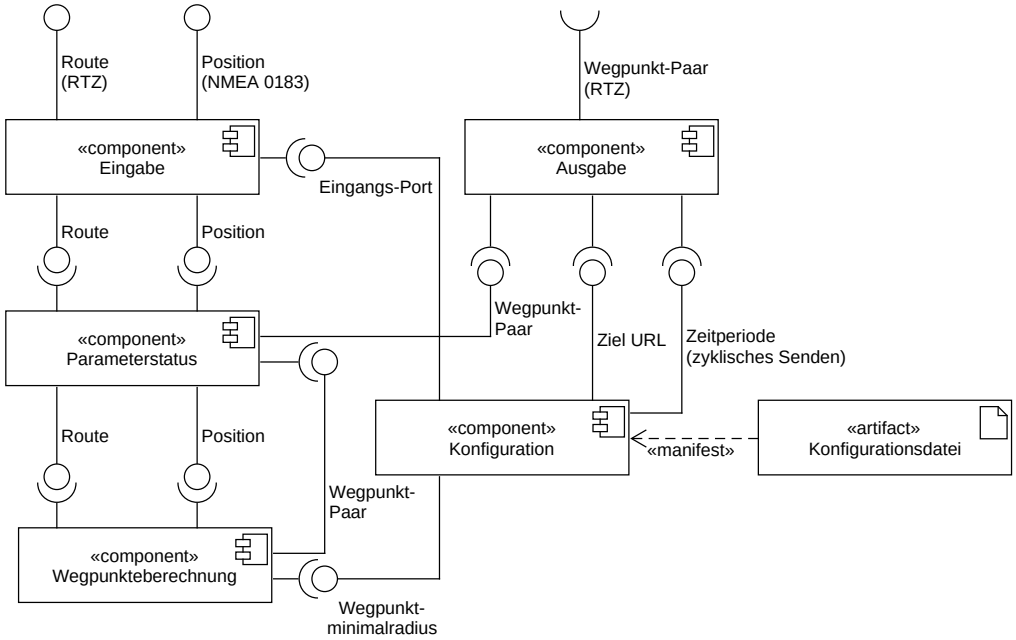


Abbildung 24: Aufbau der Pfadplanungskomponente

Während des Abfahrens einer Route ändert sich die aktuelle Position des Schiffes. Hat sich die Position des Schiffes verändert, wird eine Wegpunkteberechnungsmethode in der Pfadplanungskomponente aufgerufen. Diese prüft, ob der aktuell nächste Wegpunkt erreicht wurde. Dabei vergleicht die Wegpunkteberechnungsmethode die aktuelle Position des Schiffes mit den Koordinaten des aktuellen Wegpunkts. Befindet sich das Schiff innerhalb des in der Route vorgegebenen Radius um den Wegpunkt herum bzw. wenn dieser nicht gegeben ist innerhalb eines Minimalradius, gilt dieser Wegpunkt als erreicht. Der erreichte Wegpunkt des Wegpunktepaares wird zum ersten Wegpunkt des neuen Paares. Die Pfadplanungskomponente wählt aus der Route den nächsten Wegpunkt und setzt diesen als zweiten Wegpunkt des Wegpunktpaares. Das neue Wegpunktpaar wird zusammen mit dem aktuellen Status der Route an die Verteilungsplattform übergeben. Hat sich das Wegpunktpaar nicht geändert, wird das bisherige erneut übertragen.

Wird die aktive Route während des Abfahrens geändert, stoppt die Routenberechnung und das Abfahren der bisherigen Route. In diesem Fall wird der geänderte Status der nun inaktiven Route zusammen mit einem Wegpunktepaar an die bereitgestellte Schnittstelle übergeben, wobei das Wegpunktepaar nunmehr keine Bedeutung besitzt, da das Regelungssystem in diesem Fall nur den Status der Route berücksichtigt. Gleiches gilt, wenn

eine Route vollständig abgefahren ist. Eine Route gilt als abgefahren, wenn alle ihre Wegpunkte abgefahren wurden. Das Senden des aktuellen Wegpunktepaares und des Routenstatus erfolgt zyklisch anhand eines in einer Konfiguration zu setzenden Parameters. Diese Konfiguration bedarf einer Konfigurationsdatei. In dieser Konfiguration soll zudem festgelegt werden können, über welchen Port die Position und die Route empfangen werden sollen und an welche URL die Ergebnisse der Wegpunkteberechnungsmethode zu senden sind. Außerdem soll hier ein Minimalradius um einen Wegpunkt festgelegt werden können, ab wann dieser als erreicht gilt, falls dies nicht in dem jeweiligen Wegpunkt definiert wurde.

6.6 RTZ-Programmbibliothek

Im Systementwurf ist festgelegt, dass die Pfadplanungskomponente Informationen über Routen mit anderen Komponenten im Route plan Exchange Format austauschen soll (siehe Abschnitt 6.5). Wie in der Systemarchitektur beschrieben ist, werden Routen in der Kontrollplattform erstellt und über die Verteilungsplattform an die Pfadplanungskomponente gesendet (siehe Abbildung 11). Die Pfadplanungskomponente wiederum sendet Informationen über den Status der aktuellen Route zurück über die Verteilungsplattform an die Kontrollplattform. Das Regelungssystem erhält Navigationsbefehle, die von der Pfadplanungskomponente aus der aktuellen Route und Position des Schiffs abgeleitet werden (siehe Abschnitt 6.5.3). Somit verwenden die folgenden Komponenten das RTZ: Die Kontrollplattform, die Verteilungsplattform und die Pfadplanungskomponente. Das Regelungssystem erhält keine Informationen im RTZ (siehe Abschnitt 7.7.5). Daher müssen diese drei Komponenten Nachrichten im RTZ verstehen können. Zusätzlich müssen die Pfadplanungskomponente und die Verteilungsplattform Nachrichten im RTZ erzeugen können. Da die Logik zum Erzeugen und Verstehen von Nachrichten im RTZ in all diesen Komponenten identisch ist, ist die Verwendung einer gemeinsamen Programmbibliothek sinnvoll.

6.6.1 Umfang der Programmbibliothek

Die Programmbibliothek muss nicht das komplette RTZ unterstützen, sondern lediglich die Bestandteile, die im Systementwurf beschrieben worden sind (siehe Abschnitt 6.5.2). Neben der Definition von Routen und deren Wegpunkten können im RTZ auch Informationen über die Zeitplanung einer Route hinterlegt werden. Diese Möglichkeit wird zurzeit im Systementwurf nicht aufgegriffen und muss daher von einer RTZ-Programmbibliothek

nicht unterstützt werden. Auch bei der Definition von Etappen einer Route verwenden die Komponenten nur eine Teilmenge des RTZ. Im Folgenden werden die Informationen zusammengefasst, die von der Programmbibliothek unterstützt werden müssen.

Das RTZ beschreibt den Aufbau von XML-Dokumenten, in denen Routen und Wegpunkte definiert werden können. In der Tabelle 24 sind die Informationen aufgeführt, die von den oben beschriebenen Komponenten für Routen erfasst werden. Zu jeder Information ist das zugehörige XML-Element oder XML-Attribut in der XML Path Language adressiert.

Information	**XPath**
Name der Route	/route/routeInfo/@routeName
Status der Route	/route/routeInfo/@routeStatus
Name des Schiffs	/route/routeInfo/@vesselName
MMSI des Schiffs	/route/routeInfo/@vesselMMSI
Wegpunkte der Route	/route/waypoints/waypoint

Tabelle 24: Benötigte Informationen zu einer Route in der RTZ-Programmbibliothek

Der Name einer Route ist ein eindeutiger Bezeichner, über den eine Route identifiziert werden kann. Der Name kann dabei eine beliebig lange Zeichenkette sein. Auch der Status der Route kann laut dem RTZ eine beliebig lange Zeichenkette sein, über die der Zustand einer Route ausgedrückt wird. Die Semantik dieser Zeichenkette muss von den Anwendern des RTZ festgelegt werden und ist nicht auf bestimmte Werte eingeschränkt.

Zusätzlich kann zu einer Route der Name und die MMSI des Schiffs angegeben werden, für das diese Route bestimmt ist. Beide Informationen werden ebenfalls als beliebig lange Zeichenketten in einem RTZ-Dokument dargestellt.

Eine Route besteht aus zwei oder mehr Wegpunkten. Die Reihenfolge der XML-Elemente, die Wegpunkte repräsentieren, entspricht der Reihenfolge der Wegpunkte in der Route. In der Tabelle 25 sind die Informationen aufgeführt, die von den oben beschriebenen Komponenten für Wegpunkte erfasst werden. Zu jeder Information ist das zugehörige XML-Element oder XML-Attribut in der XML Path Language adressiert.

Information	XPath
ID des Wegpunktes	/route/waypoints/waypoint/@id
Name des Wegpunktes	/route/waypoints/waypoint/@name
Breitengrad des Wegpunktes	/route/waypoints/waypoint/position/@lat
Längengrad des Wegpunktes	/route/waypoints/waypoint/position/@lon
Maximalgeschwindigkeit	/route/waypoints/waypoint/leg/@speedMax
Radius des Wegpunktes	/route/waypoints/waypoint/@radius

Tabelle 25: Benötigte Informationen zu einem Wegpunkt in der RTZ-Programmbibliothek

Jeder Wegpunkt hat einen eindeutigen Bezeichner innerhalb einer Route, über den jeder Wegpunkt identifiziert werden kann. Die ID kann dabei eine beliebig lange Zeichenkette sein. Zusätzlich kann einem Wegpunkt ein Name zugeordnet sein, der ebenfalls als eine beliebig lange Zeichenkette definiert ist.

Jedem Wegpunkt muss eine geographische Position zugeordnet sein, die als Längen- und Breitengrad angegeben wird. Längen- und Breitengrade werden im RTZ als 64-Bit-Gleitkommazahlen nach dem IEEE-Standard 754 dargestellt (siehe [IEE08]).

Einem Wegpunkt kann ein Radius in nautischen Meilen zugewiesen sein. Dieser Radius beschreibt, ab wann ein Wegpunkt als erreicht gilt. Sobald ein Schiff sich in diesem Radius um die geographische Position eines Wegpunktes befindet, gilt dieser Wegpunkt als erreicht. Der Radius muss als 64-Bit-Gleitkommazahl nach dem IEEE-Standard 754 angegeben werden. Der Radius darf keine negative Zahl sein und muss kleiner als zehn nautische Meilen sein.

Zusätzlich kann einem Wegpunkt eine Maximalgeschwindigkeit zugewiesen werden, die in Knoten als 64-Bit-Gleitkommazahl nach dem IEEE-Standard 754 angegeben werden muss. Die Maximalgeschwindigkeit darf keine negative Zahl sein. Diese Geschwindigkeit bezieht sich auf die Teilstrecke zwischen dem vorherigen Wegpunkt und dem Wegpunkt, dem die Geschwindigkeit zugeordnet ist.

6.6.2 Aufbau der Programmbibliothek

Über die RTZ-Programmbibliothek können Objekte für Routen und Wegpunkte erzeugt werden. Diese Objekte sind unveränderlich. Das bedeutet, dass die Werte der Attribute der Objekte nach ihrer Erzeugung nicht mehr verändert werden können. Dies ermöglicht

es, die Objekte parallel in verschiedenen Threads zu verwenden, ohne zusätzliche Synchronisierungslogik zu benötigen.

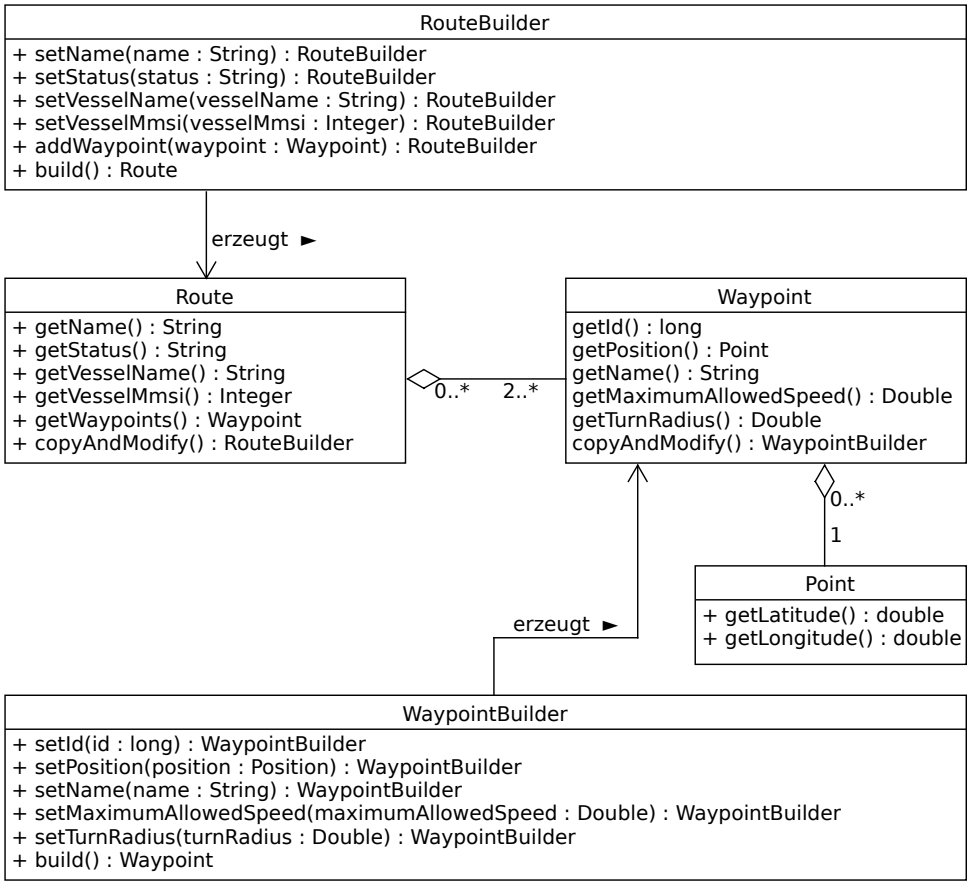


Abbildung 25: Routen und Wegpunkte in der RTZ-Programmbibliothek

In Abbildung 25 sind die Klassen für die Routen und Wegpunkte dargestellt. Einer Route sind immer mindestens zwei Waypoints zugeordnet. Jeder Waypoint wiederum hat genau einen Point. Ein Point ist eine geographische Position, die als Längen- und Breitengrad angegeben wird.

Ein neuer Waypoint kann über einen WaypointBuilder erzeugt werden. Über einen WaypointBuilder können die Attribute für einen Waypoint festgelegt werden. Über die Methode build() : Waypoint wird eine unveränderliche Instanz eines Waypoints erzeugt.

Eine neue Route kann über einen RouteBuilder erzeugt werden. Über einen RouteBuilder können die Attribute für eine Route festgelegt werden. Mit der Methode addWaypoint(waypoint : Waypoint) : RouteBuilder können die Waypoints der Route

festgelegt werden. Über die Methode $\mathrm{build}() : \mathrm{Route}$ wird eine unveränderliche Instanz einer Route erzeugt. Dies ist nur möglich, wenn mindestens zwei $\mathrm{Waypoints}$ an den $\mathrm{RouteBuilder}$ übergeben worden sind.

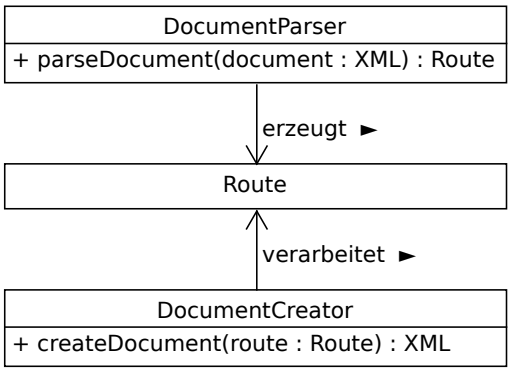


Abbildung 26: Einlesen und Erzeugen von XML-Dokumenten

In Abbildung 26 sind die Klassen dargestellt, mit denen XML-Dokumente im RTZ-Standard eingelesen und erzeugt werden können. Über den $\mathrm{DocumentParser}$ kann aus einem XML-Dokument eine Route erzeugt werden. Dafür wird die Methode $\mathrm{parseDocument}(\mathrm{document} : \mathrm{XML}) : \mathrm{Route}$ verwendet. Über den $\mathrm{DocumentCreator}$ wiederum kann aus einer Route ein XML-Dokument erzeugt werden. Dafür wird die Methode $\mathrm{createDocument}(\mathrm{route} : \mathrm{Route}) : \mathrm{XML}$ verwendet.

6.7 Regelungssystem

Das Regelungssystem ist für die Steuerbefehle des Schiffes zuständig. Dafür wird ein Controller implementiert, der diese Aufgabe erfüllen wird. Steuerbefehle werden mit Hilfe von Sensor- und Umgebungsdaten vom Schiff sowie Navigationsbefehlen berechnet und anschließend weiter verschickt. Die Steuerbefehle umfassen die Drehzahl des Motors und den Winkel des Ruders als wesentliche Parameter zur Steuerung des Schiffes. Die Parameter können elektronisch an die Schiffssteuerung übergeben werden, welche dementsprechend Motor und Ruder einstellt. Eine Steuerung über eine dritte Plattform war nicht möglich. Auch das Abfahren einer vorgegebenen Strecke war nicht möglich.

6.7.1 Aufgaben der Komponenten

Zur Erfüllung der Anforderungen muss das Regelungssystem daher eine Schnittstelle zur Verteilungsplattform zur Verfügung stellen, um Steuerbefehle nach dem NMEA2000-Standard entgegennehmen und senden zu können. Auch sollen zur Berechnung von Steuerbefehlen die folgenden Schiffsistwerte nach dem NMEA2000-Standard entgegengenommen werden:

- Geschwindigkeit über das Wasser
- Geschwindigkeit über den Grund
- Ruderwinkel
- Windgeschwindigkeit
- Steuerkurs
- Kurswinkel
- Drehzahl des Motors
- Position als Längen- und Breitengrad

Die Werte sind für eine automatische Abfahrt einer Route unerlässlich. Neben diesen Daten des Schiffes müssen für eine manuelle Steuerung eine vorgegebene Zieldrehzahl des Motors und Zielruderwinkel im NMEA2000-Standard entgegengenommen und weitergeleitet werden können. Für den Autopiloten werden zwei definierte Wegpunkte, bestehend aus Längen- und Breitengrad, im NMEA2000-Standard benötigt. Dann ist das Regelungssystem fähig auf Basis der Schiffsistwerte und den Wegpunkten die Steuerbefehle, bestehend aus Ruderwinkel und Motordrehzahl, zu berechnen. Die Steuerbefehle werden dann auf die gleiche Weise übertragen wie die manuellen Steuerbefehle. Das bedeutet vor allem, dass das Regelungssystem zwischen dem Weiterleiten von gegebenen Steuerbefehlen und der Berechnung von Steuerbefehlen unterscheiden muss. Der Versand und Empfang von Nachrichten erfolgt über die definierten Schnittstellen mit dem UDP Netzwerkprotokoll. Das Regelungssystem soll zudem als Echtzeitsystem entwickelt werden und Ruderwinkel sowie die Motordrehzahl innerhalb von $100\,ms$ berechnen und an die Verteilungsplattform senden. Auf die Gründe wird im nächsten Abschnitt genauer eingegangen.

Die Abbildung 27 zeigt die Verteilung der eben umrissenen Aufgaben als Komponentendiagramm. Die genaue Zuordnung dieser Aufgaben erfolgt mit erhöhtem Detail bei der

Beschreibung der Komponenten. Um das Datenformat klarer darzustellen, wird dieses an den Schnittstellen der Komponenten in Klammern ergänzt.

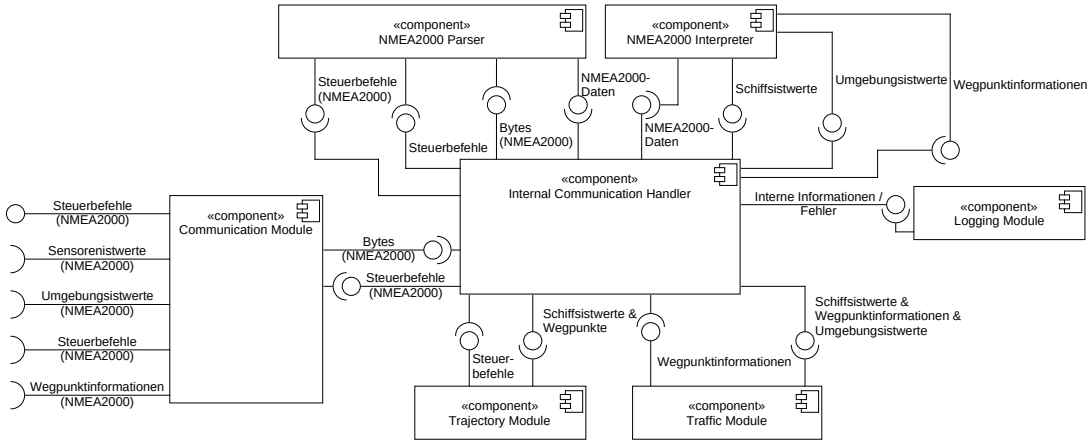


Abbildung 27: Architektur des Controllers mit den zur Erfüllung der Anforderungen nötigen Komponenten

Communication Module

Das Communication Module (deutsch: Kommunikationsmodul) ist für die externe Kommunikation zur Verteilungsplattform über UDP zuständig. Daher sind Einstellungsmöglichkeiten über IP-Adresse und Port umzusetzen. Es leitet ankommende NMEA2000-Nachrichten weiter zu dem Internal Communication Module, indem es die angenommenen Daten als Schnittstellen bereitstellt. Die ankommenden Daten beinhalten die Schiffsistwerte sowie die Umgebungsdaten. Zudem können auch manuelle Steuerbefehle oder Wegpunktinformationen entgegengenommen werden.

Internal Communication Module

Das Internal Communication Module (deutsch: Internes Kommunikationsmodul) ist für die interne Weiterleitung der Daten verantwortlich und hält alle aktuellen Daten vor. Durch das Vorhalten der Daten können auch spätestens alle $100\,ms$ neue Daten versendet werden. Es stellt zu jeder anderen Komponente Schnittstellen bereit und nimmt gleichzeitig über von diesen Komponenten bereitgestellte Daten entgegen. Damit nimmt das Internal Communication Module eine zentrale Rolle ein. Verbindungen nach außen oder zu anderen Komponenten sind nur über dieses Modul möglich. Über eine Schnittstelle zum Communication Module werden mit NMEA2000 Bytes über UDP entgegengenom-

men und manuelle Steuerbefehle im NMEA2000 Format versendet. Das Modul regelt auch die Reihenfolge der Ausführung der einzelnen Komponenten.

NMEA2000Parser

Zur Erfüllung seiner Aufgaben benötigt der NMEA2000Parser (deutsch: NMEA2000-Analysierer) die NMEA2000-Nachrichten vom Kommunikationsmodul. Diese bekommt er über das Internal Communication Module. Die NMEA2000-Nachrichten werden entpackt und analysiert. Entsprechend des Ergebnisses der Analyse wird der Datenanteil der NMEA2000-Nachricht an den nachfolgend beschriebenen NMEA2000Interpreter übergeben. Die Übergabe wird vom Internal Communication Module übernommen. Da nicht alle Informationen einer Nachricht relevant sind, werden über den Parser unnötige Daten herausgefiltert. Zu diesen Informationen zählen zum Beispiel die Header-Informationen aus dem NMEA2000-Standard. Der NMEA2000Parser übernimmt auch die Aufgabe errechnete oder manuelle Steuerbefehle in eine NMEA2000-Nachricht einzupacken und für die externe Kommunikation bereitzustellen.

NMEA2000Interpreter

Der NMEA2000Interpreter (deutsch: NMEA2000-Interpretierer) übernimmt die Aufgabe den Datenanteil einer NMEA2000-Nachricht zu analysieren und entsprechend der im NMEA2000-Standard angegebenen Werte weiter zu interpretieren. Zu dieser Interpretation gehört auch das korrekte Umwandeln der Werte mit dem scale eines NMEA2000-Feldes aus einer NMEA2000-Nachricht. Für das Weiterleiten von den interpretierten Daten an die weiterführenden Komponenten, wie Trajectory oder Traffic Module, ist das Internal Communication Module zuständig. Zur weiteren Aufgabe dieses Moduls gehört die beschriebene Unterscheidung zwischen dem Weiterleiten von Steuerbefehlen und der Berechnung von Steuerbefehlen aufgrund von Wegpunktinformationen. Das bedeutet der NMEA2000Interpreter regelt auch das Umschalten zwischen manueller Steuerung und der Berechnung der Steuerbefehle durch ein Trajectory Module.

Trajectory Module

Das Trajectory Module (deutsch: Trajektoriemodul) ist für die Berechnung und das Abfahren einer Trajektorie zuständig. Eine Trajektorie beschreibt im Deutschen eine vor-

gegebene Bahnkurve, die abgefahren wird. Die Bahnkurve wird durch die Gesamtheit der einzelnen Punkte, die ein Körper einnimmt, gebildet (vgl. [Dre12], S. 1). Für diese Berechnung nimmt das Trajectory Module die gesetzten Wegpunkte und die Schiffsistwerte entgegen. Da in den Anforderungen nicht weiter spezifiziert, wird zwischen den empfangenen Wegpunkten eine gerade Strecke gewählt. Die Trajektorie wird so gewählt, dass diese zum nächstgelegenen Wegpunkt berechnet wird. Aus dem Trajectory Module werden dann die Steuerbefehle an das Internal Communication Module gesendet und damit über das Communication Module auch an die Verteilungsplattform. Das Internal Communication Module regelt den weiteren Verlauf der Steuerbefehle.

Traffic Module

Das Traffic Module (deutsch: Verkehrsmodul) analysiert die Umgebung auf mögliche Gefahren hinsichtlich eines Zusammenstoßes. Für diese Berechnungen benötigt das Traffic Module zunächst die Umgebungsdaten des Schiffes und die Wegpunktinformationen. Dazu zählen etwa aufgenommene Bilder einer Schiffskamera oder von einem Radar aufgenommene Daten, die in diesem Modul weiterverarbeitet werden können. Erkannt werden sollen dann alle Objekte in unmittelbarer Nähe des Schiffes. Dazu zählen unter anderem andere Schiffe oder auf dem Wasser treibende Objekte, wie Container oder Bojen. Diese erkannten Gefahren sollen dem Trajectory Module als zusätzliche Umgebungsdaten übergeben werden, um dort gegebenenfalls eine neue Strecke zu errechnen.

Logging Module

Im Fehlerfall soll der Grund für den Fehler genauestens analysiert werden können. Daher wird der Controller um ein Logging Module (deutsch: Aufzeichnungsmodul) erweitert, das alle Fehler und sonstigen wichtigen Informationen aufzeichnet. Das hat den Vorteil, ein Problem nachträglich schnell nachbilden und entsprechend bereinigen zu können.

6.7.2 Regelungssystem

Die Berechnungen der Steuerbefehle finden in dem Trajectory Module statt. Das Problem in einem dynamischen System ist die mögliche Beeinträchtigung durch viele Störfaktoren. Nach [Fos11] lassen sich Schiffe mit Hilfe eines dynamischen Modells beschreiben. Unter den Störfaktoren fallen zum Beispiel Wind und Wellengang (vgl. [Fos11], S. 9).

Durch ein Regelungssystem lassen sich solche Störungen mit in die Berechnung einbeziehen.

Ein Regelungssystem besteht aus mehreren Teilsystemen und regelt zeitlich veränderliche Größen des Systems auf vorgegebene Werte (vgl. [Lit13], S. 25). Dabei können Störungen auftreten, die diesen Prozess beeinträchtigen und daher miteinbezogen werden müssen. Das Regelungssystem muss seinen Ausgabewert $y_R(t)$ zu jedem Zeitpunkt t trotz der Störungen ($d_A(t), d_P(t), d_S(t)$) halten können. In der Abbildung 28 ist ein Regelkreis eines Regelungssystems zu sehen.

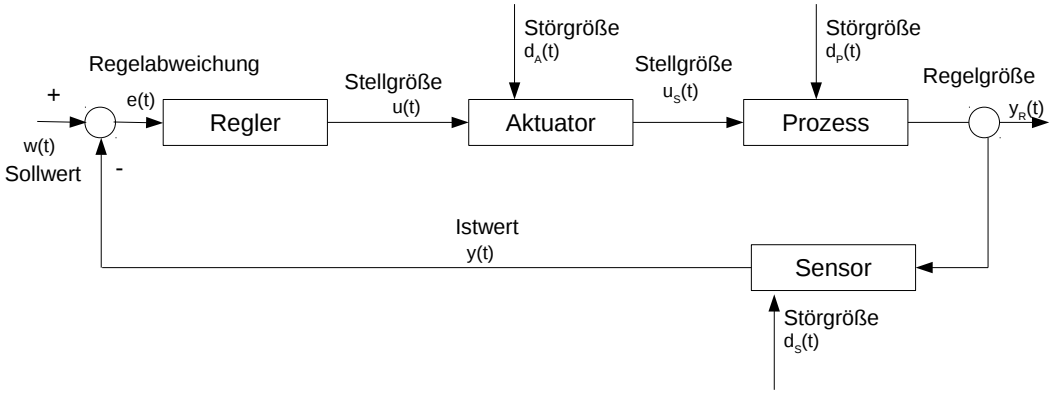


Abbildung 28: Einschleifiger Regelungskreis mit allen wichtigen Komponenten (vgl. [Lit13], S. 26)

Der Regelkreis bekommt einen Sollwert $w(t)$ gegeben. Der Istwert $y(t)$ soll dem Sollwert $w(t)$ gleichen und anschließend dort gehalten werden. Dafür wird als erstes die Regelabweichung $e(t)$ über $e(t) = w(t) - y(t)$ gebildet (vgl. [Lit13], S. 26). Diese wird dem Regelgesetz zugeführt, welcher eine Stellgröße $u(t)$ berechnet. Die Stellgröße wirkt auf den Aktuator korrigierend ein, sodass der Istwert später dem Sollwert entspricht.

Der Aktuator wird durch die Störung $d_A(t)$ beeinträchtigt. Es resultiert eine neue Stellgröße $u_S(t)$, die dem Prozess zugeführt wird. Die Regelgröße $y_R(t)$ wird durch den Prozess gebildet, aber ist durch $d_P(t)$ gestört. Die Komponente versucht die Stellgröße wieder dem Sollwert $w(w)$ anzugleichen. Die Regelgröße kann dann außerhalb des Regelkreises verwendet werden und wird zudem wieder in den Regelkreis zum Sensor gegeben. Mit Hilfe eines Sensors wird der tatsächliche Istwert $y(t)$ gemessen und zur Berechnung der

Regelabweichung zurückgeführt. Stört $d_S(t)$ den Istwert, dann ist $y(t)$ nur mit einer gewissen Abweichung genau. Im Idealfall, ohne eine einwirkende Störung, ist die Regelabweichung 0 und der Istwert entspricht dem Sollwert. In diesem Fall gilt $w(t) = y(t)$ und das System ist ausgeregelt (vgl. [Lit13], S. 26).

Eine Regelabweichung $e(t)$ tritt somit immer auf, wenn die Eingangsgröße $w(t)$ und der Istwert $y(t)$ nicht genau gleich sind. Wenn die Eingangsgröße gleichbleibt, muss der Istwert zu der Eingangsgröße ausgeregelt werden. Dies bedeutet, dass das Regelgesetz die Störungen ausregeln muss. Wenn die Regelabweichung 0 beträgt und es dann zu einer Änderung der Eingangsgröße $w(t)$ kommt, ist die Regelabweichung $e(t) \neq 0$. Das Regelgesetz muss daher so aufgebaut sein, dass die Regelabweichung auf $e(t) = 0$ geregelt wird (vgl. [Lit13], S. 26).

Ein Beispiel für ein Regelgesetz wäre der Proportional-Integral-Derivative (PID)-Regler. Dabei handelt es sich um ein Regler der in über 95 % der Fälle in der Industrie benutzt wird (vgl. [Lit13], S. 124). Die Bezeichnung kommt von einem proportionalen, einem integrierenden und einem differenzierenden Anteil im PID-Regler (vgl. [Lit13], S. 124). Der P-Anteil berücksichtigt die Gegenwart, der I-Anteil die Vergangenheit und der D-Anteil die Zukunft (vgl. [Lit13], S. 125). Alle Anteile können einzeln eingestellt werden und führen alle zusammen zu einer Stellgröße. Der PID-Controller kann zum Beispiel den benötigten Ruderwinkel berechnen (vgl. [Fos11], S. 352).

6.7.3 Echtzeitfähigkeit

Echtzeitsysteme lassen sich in unterschiedliche Kategorien einteilen. Es gibt „soft“, „firm“ und „hard“ Echtzeitsysteme, die sich in den Konsequenzen einer Nichteinhaltung von Fristen unterscheiden (vgl. [Roc14], S. 4). Dabei ist ein „hard“ Echtzeitsystem so definiert, dass das Nichteinhalten einer Frist ein Totalausfall des Systems bedeutet (vgl. [Roc14], S. 4 f.). Löst ein Airbag bei einem Autounfall nicht innerhalb einer vorgegebenen Frist aus, ist dies ein solcher Totalausfall, bei dem Menschenleben gefährdet sind (vgl. [Roc14], S. 5).

Bei einem „firm“ Echtzeitsystem wird die Nichteinhaltung der Frist nicht als Totalausfall bewertet (vgl. [Roc14], S. 5). Nach Ablauf der Frist berechnete oder ankommende Daten sind dann allerdings nutzlos. Die Daten werden dann verworfen und nicht mehr benötigt. In diesem Fall hat das Nichteinhalten von Fristen keinen gravierenden Einfluss auf die Umwelt (vgl. [Roc14], S. 5).

Bei einem „soft“ Echtzeitsystem sind die Daten nach der Frist noch nutzbar (vgl. [Roc14], S. 5). Ein Beispiel ist das Dekodieren von Videodaten. Werden die Bilder nicht innerhalb der gesetzten Frist dekodiert, kann das Bild ruckeln, aber das Video läuft weiter.

Damit der Controller seine Berechnungen für die Drehzahl des Motors und des Ruderwinkels immer in einer maximalen Zeit von 100*ms* durchführen kann, müssen die Anwendung des Controllers, das Betriebssystem und die Hardware echtzeitfähig sein. Das Regelungssystem als ein Echtzeitsystem darf eine festgelegte Zeit nicht überschreiten. Wird diese Zeit überschritten, muss dies als Totalausfall des Systems betrachtet werden. Der Grund lässt sich leicht an einem extremen Beispiel festmachen. Erkennt das Regelungssystem ein Hindernis und sendet nicht innerhalb der vorgegebenen Frist die neuen Steuerbefehle zum Ausweichen, so kann dies zu einer Kollision mit dem erkannten Hindernis führen. Daher ist das Regelungssystem als Echtzeitsystem mit der Eigenschaft „hard“ umzusetzen.

6.8 Notfallsteuerung

Bei der Steuerung eines Schiffs handelt es sich um ein komplexes Unterfangen. Es besteht die Möglichkeit, dass Gefahrensituationen entstehen, bei denen auch Menschen, Tiere, das Schiff oder andere Objekte wie eine Kaimauer zu Schaden kommen können. Neben den Komponenten des Produkts der PG MATE für das autonome Schiff soll deswegen eine weitere Komponente zur Verfügung stehen.

Diese Komponente soll das schnelle Eingreifen in Gefahrensituationen sicherstellen und möglichst unabhängig von allen anderen Komponenten sein. Dies ist vor allem in der frühen Entwicklungs- und Testphase von Bedeutung, da es hier vermehrt zu schwerwiegenden Fehlern kommen kann, die ein Eingreifen in die Steuerung erfordern. Die Entwicklung einer Notfallsteuerung stellt darüber hinaus eine explizite Anforderung der Projektgruppenbetreuer dar (siehe dazu: Tabelle 7).

Aufgabe der Notfallsteuerung ist es, einer Person die Möglichkeit zu geben die Kontrolle über das Schiff zu übernehmen. Diese Person kann sich an Land oder auf dem Schiff befinden. Dabei soll die Person die Steuerung des Schiffs so beeinflussen können, dass das Schiff aus der Gefahrensituation heraus manövriert werden kann. Für die Komponente ist deshalb ein möglichst stabiler und ausfallsicherer Kommunikationskanal zu wählen. Damit im Gefahrenfall die Steuerbefehle der Notfallsteuerung nicht mit den Steuerbefehlen des Produktes der PG MATE vermischt werden, müssen die Steuerbefehle der Notfallsteuerung höher priorisiert werden.

6.8.1 Schnittstellen der Notfallsteuerungskomponente

Die Notfallsteuerung muss über eine Benutzerschnittstelle verfügen, um Steuerbefehle über ein Eingabegerät vom Benutzer aufnehmen zu können. Außerdem müssen über eine Benutzerschnittstelle Verbindungsparameter eingegeben werden können. Die Notfallsteuerung verfügt über eine weitere Schnittstelle zur Schiffssteuerung im Testbed, um Steuerbefehle zur übertragen. Abbildung 12 zeigt die Einbindung der Notfallsteuerung in die Systemarchitektur des Testbeds.

6.8.2 Aufbau der Notfallsteuerungskomponente

Tritt eine Gefahrensituation ein, die ein Eingreifen erfordert, muss die Notfallsteuerung aktiviert werden. In einer Gefahrensituationen ist es erforderlich, dass ein Anwender schnell und intuitiv die Kontrolle über die Steuerung der Zuse übernehmen kann. Dies wird über ein Gamepad realisiert. Die Steuerung über ein Gamepad hat im Vergleich zur Steuerung über eine reine Softwarelösung den Vorteil, dass die Schaltflächen des Gamepads haptisch erfasst werden können. Der Anwender muss die Schaltflächen daher nicht genau erkennen. Dies ist beispielsweise bei schlechten Wetterverhältnissen der Fall. Darüber hinaus sitzt ein Gamepad bei der Steuerung sicher in der Hand des Anwenders und kann selbst bei schwerem Seegang eingesetzt werden.

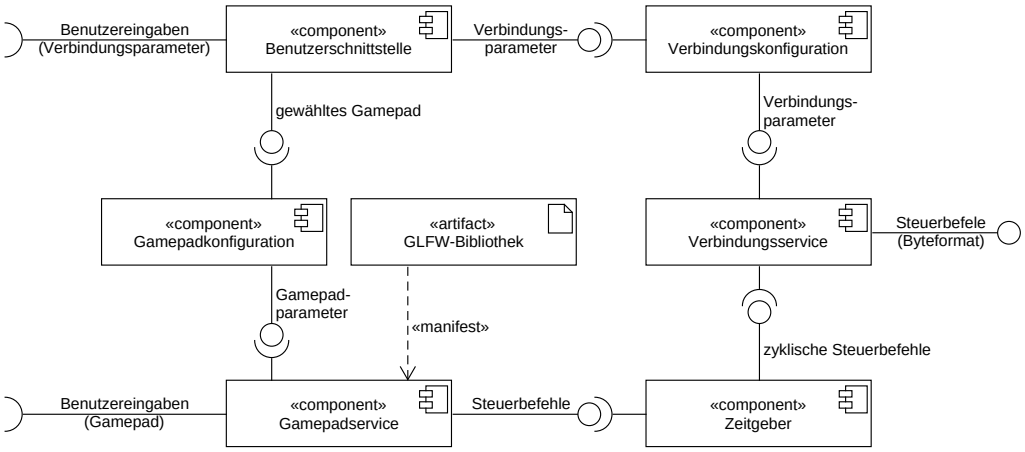


Abbildung 29: Aufbau der Notfallsteuerungskomponente

Zur Verarbeitung der Nutzereingaben über das Gamepad soll die „OpenGL FrameWork (GLFW)-Library“ verwendet werden. Bei dieser handelt es sich um eine plattform-

unabhängige, in Java umgesetzte Open-Source Lösung. Die GLFW-Library unterstützt viele verschiedene Gamepads. Als Gamepad wird ein „XBox 360 Controller“ verwendet. Dieser wird auch von der GLFW-Library unterstützt. Dieses Gamepad findet Einsatz, da dieser Gerätetyp mehrfach bei einzelnen Projektgruppenmitgliedern vorhanden ist, was die Entwicklung und das Testen vereinfacht. Außerdem ist dieser Gerätetyp weit verbreitet und kostengünstig, weshalb er auch von möglichen Folgeprojektgruppen verwendet werden kann.

Um in einer Gefahrensituation unmittelbar reagieren zu können, muss die Notfallsteuerung zu jeder Zeit einsatzbereit sein. Damit dies möglich ist, muss die Verbindung zwischen Notfallsteuerung und Schiffssteuerung im Vorfeld konfiguriert werden. Dafür müssen die notwendigen Verbindungsparameter in die Benutzeroberfläche der Notfallsteuerung eingegeben werden.

Die Benutzerschnittstelle wird über die Konsole bedient. Für die Kommunikation mit der Zuse stehen zwei Verbindungstypen zur Verfügung. Es kann eine Verbindung via RabbitMQ oder via UDP hergestellt werden. Bei RabbitMQ handelt es sich um eine Open Source Message Broker Software die das AMQP implementiert. UDP wurde als Protokoll gewählt, da zum einen die Schiffsteuerung ausschließlich das Netzwerkprotokoll UDP unterstützt.

Der Benutzer wird über die Konsole aufgefordert sich für einen Verbindungstypen zu entscheiden. Abhängig davon, welchen Verbindungstypen der Benutzer gewählt hat, muss der Benutzer im Anschluss die Verbindungsparameter eingeben.

Für RabbitMQ sind dies folgende:

Parameter	Beschreibung
Host	Name des Hosts des AMQP-Servers zu dem die Steuerbefehle gesendet werden sollen.
Port	Der Port auf dem AMQP-Server.
Benutzername	Benutzername für den AMQP-Server.
Passwort	Das Passwort zum Benutzernamen.
Exchange name	Name des AMQP Exchange zu dem die Steuerbefehle gesendet werden.
Exchange type	Typ des AMQP Exchange (Beispiel: „direct“)
Exchange routing key	Der Routing-Schlüssel für die Ziel-Warteschlange.
Durable	Der Exchange kann dauerhaft (true) oder vorübergehend (false) sein.
Big Endian	Die Byte-Reihenfolge, mit der die Steuerbefehle erzeugt werden sollen. Big Endian (true) oder Little Endian (false).

Tabelle 26: Verbindungsparameter für RabbitMQ (Notfallsteuerung)

Für UDP sind dies folgende:

Parameter	Beschreibung
Host	Name des Hosts des UDP-Servers zu dem die Steuerbefehle gesendet werden sollen.
Port	Der Port auf dem UDP-Server.
Little Endian	Die Byte-Reihenfolge, mit der die Steuerbefehle erzeugt werden sollen: Little Endian (true) oder Big Endian (false).

Tabelle 27: Verbindungsparameter für UDP (Notfallsteuerung)

Nachdem die Verbindungsparameter eingegeben wurden, erfolgt der Verbindungsaufbau zum Gamepad. Dabei wird geprüft, ob bereits ein Gamepad angeschlossen und erkannt wurde. Solange kein Gamepad angeschlossen und erkannt wurde, wartet die Benutzerschnittstelle bis dies erfolgt ist. Sobald ein Gamepad erkannt wurde, kann diese ausgewählt werden. Im Anschluss ist die Konfiguration abgeschlossen und die Steuerbefehle können an die Schiffssteuerung geschickt werden. Die Schiffssteuerung ist eine schiffsseitige Komponente, über welche die Motordrehzahl und der Ruderwinkel steuerbar sind. Da

die Schiffssteuerung direkt mit dem Zielformat angesprochen wird, müssen keine Daten transformiert werden. Bei dem Zielformat handelt es sich um eine vom OFFIS-Mitarbeiter Peter Tank vorgegebene Byte-Reihenfolge. Das Zielformat der Steuerbefehle wird in einem späteren Absatz näher beschrieben.

Im Anschluss müssen die Eingaben des Benutzers von der Notfallsteuerung entgegengenommen werden können, sobald die Gefahrensituation eintritt. Über die „Start“-Schaltfläche des Gamepads wird die Verbindung zum Schiff aufgebaut und die Notfallsteuerung aktiviert. Zum Verbindungsaufbau werden die zuvor eingegebenen Verbindungsparameter verwendet.

Die Notfallsteuerung muss die Benutzereingaben in Steuerbefehle umwandeln, die von der Schiffssteuerung entgegengenommen und interpretiert werden können. Diese Steuerbefehle müssen im fest definierten Format der Byte-Reihenfolge an die Schnittstelle der Schiffssteuerung übergeben werden.

Die durch den Benutzer eingegebenen Steuerbefehle für die Schiffssteuerung müssen im Anschluss an diese gesendet werden. Das Schiff kann mit dem rechten Analog-Stick kontrolliert werden. Die Motordrehzahl wird mit der vertikalen Y-Achse geändert. Der Rudderwinkel wird mit der horizontalen X-Achse geändert. Dabei müssen für die Steuerbefehle die folgenden Eingaben möglich sein:

- Die Notfallsteuerung muss dem Anwender die Möglichkeit bieten, über die „Hoch“-Schaltfläche der vertikalen Y-Achse des Gamepads die Motordrehzahl des Motors der Zuse auf bis zu 2000 RPM zu erhöhen.
- Die Notfallsteuerung muss dem Anwender die Möglichkeit bieten, über die „Runter“-Schaltfläche der vertikalen Y-Achse des Gamepads die Motordrehzahl des Motors der Zuse auf bis zu -2000 RPM zu reduzieren.
- Die Notfallsteuerung muss dem Anwender die Möglichkeit bieten, über die „Links“-Schaltfläche der horizontalen X-Achse des Gamepads den Ruderwinkel des Ruders der Zuse auf bis zu 20 Grad zu erhöhen.
- Die Notfallsteuerung muss dem Anwender die Möglichkeit bieten, über die „Rechts“-Schaltfläche der horizontalen X-Achse des Gamepads den Ruderwinkel des Ruders der Zuse auf bis zu -20 Grad zu reduzieren.
- Falls die Schaltfläche zum Steuern der Motordrehzahl sowie des Ruderwinkels auf Neutral steht, dürfen die zyklischen Steuerbefehle aufgrund der Ungenauigkeit des Gamepads den Wert 0 nur minimal über- bzw. unterschreiten.

Sobald die Verbindung zwischen der Notfallsteuerung und der Zuse hergestellt ist, soll die Notfallsteuerung zyklisch alle 250 Millisekunden Steuerbefehle an die Zuse senden. Sind die Steuerbefehle bei der Schiffssteuerung angekommen, werden die Befehle von dieser verarbeitet und das Schiff kann aus der Gefahrensituation herausmanövriert werden.

Ist die Gefahrensituation behoben, kann die Verbindung der Notfallsteuerung zum Schiff wieder abgebaut werden. Dies geschieht mit der „Zurück“-Schaltfläche des Gamepads. Dieser Verbindungsabbau ist notwendig, da die Steuerbefehle der Notfallsteuerung zyklisch gesendet werden, auch wenn keine neuen Eingaben mehr erfolgt sind. Außerdem werden die Steuerbefehle der Notfallsteuerung vom Schiffssystem höher priorisiert als die Steuerbefehle des Produktes der PG MATE. Sobald die Verbindung zwischen der Notfallsteuerung und der Zuse abgebaut ist, sendet die Notfallsteuerung nicht mehr zyklisch Steuerbefehle an die Zuse. Der letzte Abschnitt der Notfallsteuerungskomponente, die Kommunikation mit der Schiffsteuerung über die Byte-Reihenfolge, ist schiffsspezifisch. Diese muss für jedes Schiff auf dem die Notfallsteuerung eingesetzt werden soll, individuell implementiert werden.

6.8.3 Aufbau der Steuerbefehle

Im Folgenden wird der Aufbau der Steuerbefehle für die Notfallsteuerung beschrieben. Die Steuerbefehle liegen in einem Byte-Format vor. Es gibt zwei verschiedene Befehle: Einen für die Steuerung der Motordrehzahl und einen für die Steuerung des Ruderwinkels. Beide Befehle sind gleich aufgebaut und unterscheiden sich lediglich durch ein anders gesetztes Byte.

Index	**Länge**	**Inhalt**
0	1	Nachrichtentyp: Immer 2.
1	1	Nachrichtenlänge ohne Prüfsumme: Immer 12.
2	1	Validität der Nachricht: Immer 1.
3	1	Typ des Steuerbefehls: 1 (Motordrehzahl) oder 2 (Ruderwinkel).
4	4	Konstante: Immer 0.
8	4	Der neue Wert, der gesetzt werden soll.
12	1	Prüfsumme des Steuerbefehls.

Tabelle 28: Aufbau der Steuerbefehle für die Notfallsteuerung

In Tabelle 28 sind die Felder der Steuerbefehle dargestellt. In der ersten Spalte steht der Index, ab dem das Feld beginnt. In der zweiten Spalte steht die Länge des Feldes. In der dritten Spalte ist der Inhalt des Feldes beschrieben.

Jeder Befehl besteht immer aus genau 13 Bytes. Die ersten drei Felder sind für die Notfallsteuerung Konstanten. Über das vierte Feld werden die Befehle für die Steuerung der Motordrehzahl und des Ruderwinkels unterschieden. Das fünfte Feld ist ebenfalls eine Konstante. Über das sechste Feld wird der neue Wert für die Motordrehzahl oder den Ruderwinkel übermittelt. Im siebten Feld wird eine Prüfsumme für den Steuerbefehl übertragen. Die Prüfsumme wird über alle vorherigen Felder des Steuerbefehls ermittelt. Dabei werden die Bytes nacheinander mit einem XOR-Gatter verknüpft.

7 Umsetzung

Auf Basis der Konzepte aus dem Systementwurf wurde die Umsetzung durchgeführt. Die entsprechenden Implementierungen werden in diesem Kapitel beschrieben, analog zur Gliederung im Systementwurf.

7.1 NMEA2000

In diesem Abschnitt wird auf die XML Umsetzung des NMEA2000-Standards eingegangen. Die Gründe für die Wahl von XML werden genannt und anschließend der Aufbau des Dokuments erläutert.

7.1.1 XML-Wahl

Der NMEA2000-Standard muss für den Gebrauch in einem maschinenlesbaren Standard gebracht werden. Dabei fiel in Absprache mit den Betreuern die Wahl auf das XML-Format. Mit dem XML-Format lassen sich standardisierte Dokumente erstellen, die für Computersysteme lesbar sind (vgl. [Mel11], S. 3). Durch die Baumstruktur und passend benannte Elementnamen sind Daten im XML Format auch für den Menschen gut lesbar und verständlich. Dies hat direkte positive Auswirkungen auf die Wartbarkeit des Dokumentes.

In Absprache mit den Betreuern der Projektgruppe wurde nur ein Bruchteil der NMEA2000-Nachrichten aufgenommen. Dies hat den Grund, dass die im Projekt verwendeten Komponenten nur mit den ausgewählten Nachrichten kommunizieren. Das Projekt endet nicht nach der Beendigung dieser Projektgruppe. So können bei Bedarf über das XML-Dokument weitere NMEA2000-Nachrichten hinzugefügt werden. XML folgt einem schematischen Aufbau der die Erweiterbarkeit signifikant unterstützt. Dies vereinfacht die nachträgliche Einpflegung zusätzlicher Nachrichten in die XML-Datei.

Das XML-Dokument wird verwendet um später andere XML-Dateien zu serialisieren. Somit ist es wichtig, dass der Aufbau der XML-Datei nicht mehrdeutig ist. Der hierarchische Aufbau der XML-Datei ist für den Softwareentwickler somit von hohem Interesse.

7.1.2 Aufbau der XML

Der Aufbau der XML ist an den NMEA2000-Appendix angelehnt und beinhaltet viele wichtige Informationen aus diesem. Für mehr Informationen zum Standard siehe

[Com16]. Die Beschreibung des NMEA2000-Standards im NMEA2000-Appendix ist nicht frei zugänglich und darf somit an dieser Stelle nicht detailliert beschrieben werden. Im Folgenden wird somit auf den grundsätzlichen Aufbau der XML eingegangen, ohne den spezifischen Aufbau einer im Standard beschriebenen NMEA2000-Nachricht preiszugeben.

Da der Standard nicht alle Fälle der Kommunikation abdeckt, welche im Projekt benötigt wird, musste die Projektgruppe eine eigene PGN-Message standardkonform beschreiben. Diese Nachricht umfasst das Setzen von einem Revolutions per minute (RPM)-Wert. Die genaue Beschreibung dieser Nachricht wird nicht im NMEA2000-Appendix definiert und wird als Beispiel im Anhang (siehe Anhang A.6) gezeigt.

Die XML-Datei beginnt mit der Beschreibung der XML-Version und der Kodierung. In dem Fall der NMEA2000-XML wurde der Standard in der Version 1.0 und die Kodierung UTF8 genutzt. Im Folgenden werden die einzelnen Tags der XML-Datei und ihr Zusammenhang genauer beschrieben.

NMEA2000

Jede einzelne NMEA2000-Datei ist über einen einzigen NMEA2000-Tag definiert. Der NMEA2000-Tag besitzt keine weiteren Eigenschaften und kann nur Message-Tags beinhalten.

Message

Eine im Standard definierte Nachricht wird in einem Message-Tag widergespiegelt. Die eindeutige Nummer der Nachricht Parameter Group Number (PGN), ein selbst gewählter Typ und die Byte-Länge der Nachricht sind die Eigenschaften vom Message-Tag. Dieses kann wiederum nur mehrere Field-Tags beinhalten. Die Field-Tags sind in einer festen Reihenfolge als Tupel in einer Message eingebettet.

Field

Ein Field-Tag spiegelt die Beschreibung eines Wertes oder eine Menge von Werten im NMEA2000-Standard wieder. Dabei ist der Aufbau eines Fields immer exakt gleich und kann anhand der Eigenschaft Name eindeutig identifiziert werden. Ein bestimmtes Field-Tag kann in verschiedenen Messages zu finden sein, jedoch nur einmal. Eine Eigenschaft

des Fields enthält unter anderem den $\mathrm{BitOffset}$ in einer Nachricht sowie die $\mathrm{ByteLength}$. Des Weiteren werden noch Informationen über den Datentyp des beschriebenen Wertes und deren Skalierung über die Eigenschaften angegeben.

Ein Field besitzt eine weitere Eigenschaft, diese gibt an ob das Field-Tag eine Menge von Beschreibungen ist. Wenn dies der Fall ist, enthält das Field-Tag mehrere Value- oder Subfield-Tags. Ansonsten enthält das Field-Tag keine weiteren Tags.

Value

Die Beschreibung der Values befindet sich im übergeordneten Field-Tag. Die Value-Tags treten immer als Tupel mit eindeutigem Vorgänger oder Nachfolger und nie alleine auf. Sie geben in den Eigenschaften immer dessen Namen und den Wert für eine exakte Zuordnung an.

Subfield

Die Subfield-Tags beschreiben in Field-Tags weitere Standards, welche von dem NMEA2000-Standard benutzt werden. Diese Dokumente der Standards haben einen unterschiedlichen Aufbau wie der NMEA2000-Standard. Deshalb müssen diese entsprechend gesondert interpretiert werden und in den Subfields eingepflegt werden. Zudem treten die Subfield-Tags in der XML-Datei sehr selten auf.

7.2 Polymorphe Schnittstelle

Dieser Abschnitt befasst sich mit der Implementierung der polymorphen Schnittstelle. Der dazugehörige Entwurf ist in Abschnitt 6.2 zu finden. Jegliche Implementierungen werden in der Programmiersprache Java durchgeführt, mit der Ausnahme der Transformationen, die als XSLT umgesetzt werden.

7.2.1 Server-API

Die Server-API der polymorphen Schnittstelle umfasst die Java-Schnittstellen, die zum Implementieren und Ausführen von Connectors benötigt werden. Diese Programmierschnittstelle wird als eigenständige Komponente entwickelt, damit die Implementierung der Connectors und der Laufzeitumgebung auf einer gemeinsamen Basis aufsetzen können. Die Server-API wird als Maven Artefakt durch den Build-Prozess erstellt, das wie-

derum von anderen Komponenten mithilfe von Maven als Abhängigkeit referenziert werden kann.

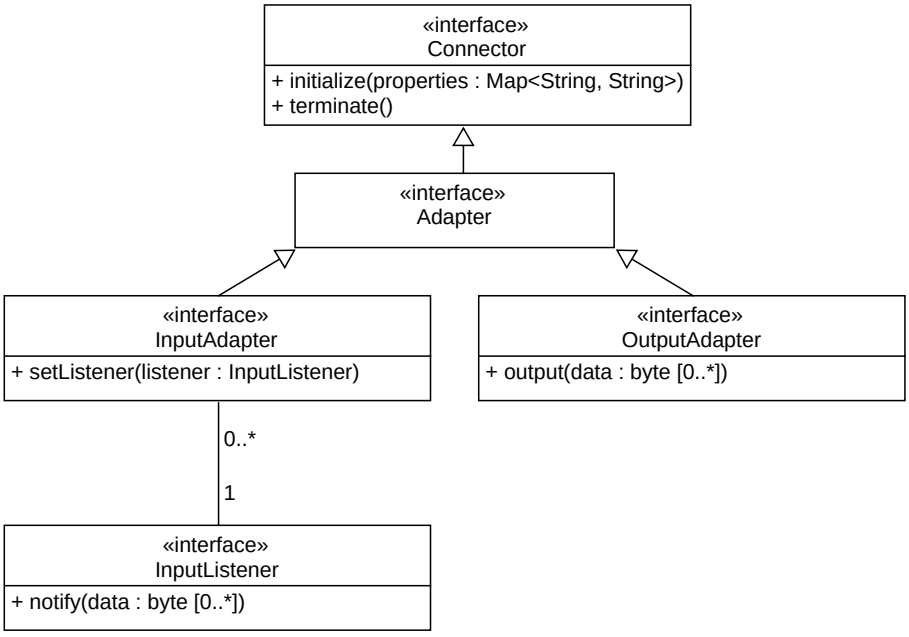


Abbildung 30: Schnittstelle eines Adapters in der polymorphen Schnittstelle

Adapter sind Connectors für die polymorphe Schnittstelle, die für die Kommunikation mit anderen Systemen zuständig sind. Über Adapter kann die polymorphe Schnittstelle Daten empfangen und versenden. In Abbildung 30 wird die Umsetzung der Schnittstellen in Java verdeutlicht. Siehe Abschnitt 6.2.5 für weitere Details zu den einzelnen Schnittstellen.

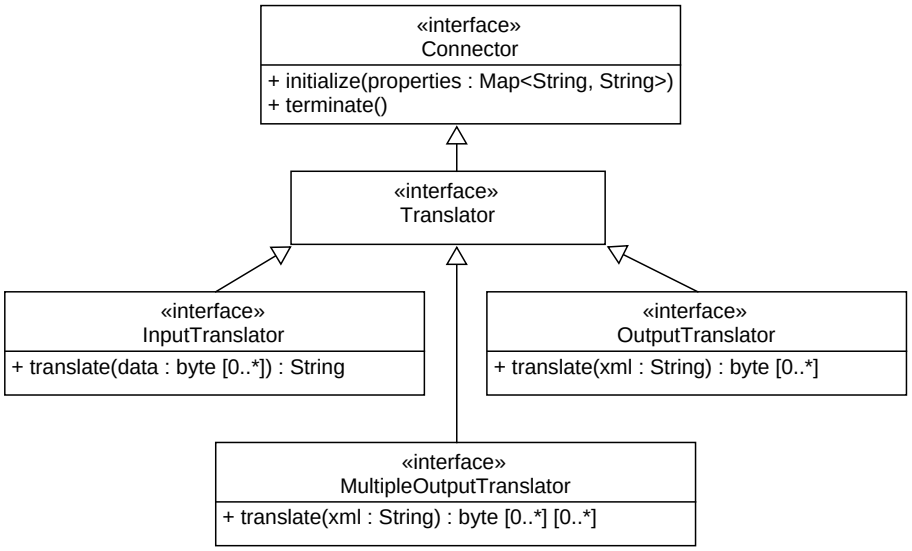


Abbildung 31: Schnittstelle eines Translators in der polymorphen Schnittstelle

Translator sind Connectors für die polymorphe Schnittstelle, die für das Übersetzen von Daten in Modellinstanzen zuständig sind. In Abbildung 31 wird die Umsetzung der Schnittstellen in Java verdeutlicht. Siehe Abschnitt 6.2.6 für weitere Details zu den einzelnen Schnittstellen.

Der Quellcode der Schnittstellen in der Server-API ist mit einer umfangreichen Javadoc-Beschreibung versehen, was die Implementierung von Connectors vereinfacht. Zudem kann eine Dokumentation in Form einer Webseite für die Server-API automatisch aus dem Javadoc generiert werden.

7.2.2 Server

Bei dem Server handelt es sich um die Implementierung der Komponente der polymorphen Schnittstelle, die für die eigentliche Ausführung der Transformationen zuständig ist. Dazu gehört das Laden der Konfiguration und der Connectors. Des Weiteren umfasst der Server das Aufbauen und Initialisieren der Pipes. Zuletzt führt der Server die Validierung und Transformation der Modellinstanzen aus. Die Umsetzung des Servers setzt sich dabei aus den in Abschnitt 6.2.7 und Abschnitt 6.2.8 beschriebenen Entwürfen zusammen.

Eine Besonderheit der Implementierung des Servers der polymorphen Schnittstelle stellt das Laden der Connectors dar. Wie in Abschnitt 6.2.8 beschrieben, enthält der Ent-

wurf der polymorphen Schnittstelle eine Laufzeitumgebung, die dynamisch alle notwendigen Connectors beim Starten lädt.

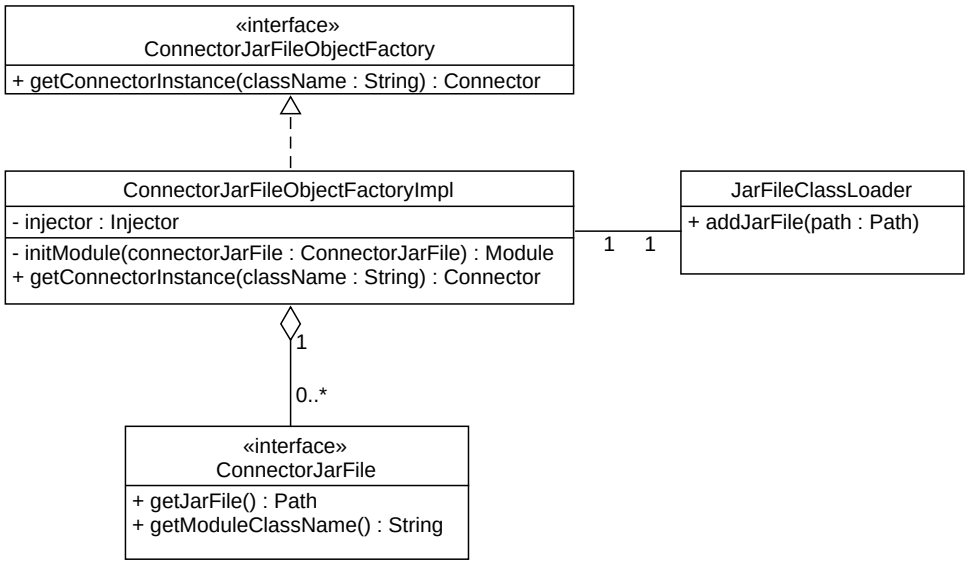


Abbildung 32: Komponenten für das Laden von Connectors in der polymorphen Schnittstelle

Abbildung 32 zeigt die Implementierung der benötigten Komponenten zum Laden der Connectors in Java. Jeder zu ladende Connector wird durch ein $\mathrm{ConnectorJarFile}$ repräsentiert. Diese Schnittstelle enthält Angaben zu dem Pfad der Jar-Datei des Connectors und den Klassennamen eines Guice-Module, dass in der Jar-Datei enthalten sein muss. Beide Angaben werden über die Konfigurationsdatei der polymorphen Schnittstelle für jeden Connector eingestellt. Guice ist ein leichtgewichtiges Framework für Dependency Injection für Java. Ein Module ist eine Konfiguration, die beschreibt welche Klassen mithilfe von Dependency Injection geladen werden können (vgl. [LM09]).

Das Laden eines Connectors läuft wie folgt ab:

1. Eine Instanz der Klasse $\mathrm{ConnectorJarFileObjectFactoryImpl}$ wird instanziiert. Das Objekt hat Zugriff auf die Konfiguration der polymorphen Schnittstelle und somit auch auf die konfigurierten $\mathrm{ConnectorJarFiles}$.

2. Für jedes $\mathrm{ConnectorJarFile}$ wird die private Methode $\mathrm{initModule(connectorJarFile : ConnectorJarFile)}$ aufgerufen. Diese Methode ruft wiederum die Methode $\mathrm{addJarFile(path : Path)}$ der Klasse $\mathrm{JarFileClassLoader}$

auf. Dabei werden die Jar-Dateien aller Connectors geladen. Danach werden die geladenen Klassen nach den konfigurierten Guice-Modules durchsucht, welche dann zu einem Injector zusammengefasst werden. Injector ist eine weitere Klasse von Guice, die das Instanziieren von Objekten aus Modules ermöglicht.

3. Im letzten Schritt wird die Methode getConnectorInstance(className : String) mit dem Klassennamen des zu ladenden Connectors aufgerufen. Diese Klasse wird von dem Injector aus einem der vorher geladenen Modules instanziiert und zurückgegeben.

Der geladene Connector kann nun initialisiert und zum Verarbeiten von Daten in einer Pipe genutzt werden.

7.2.3 Adapter

Zur Verbindung mit dem Produkt der PG MATE ist eine UDP Verbindung notwendig. Gleichzeitig müssen über eine RabbitMQ Schnittstelle Daten vom Datenverteiler entgegengenommen werden können. Für diese Verbindungseinrichtung und den Versand bzw. Empfang über diese Verbindungen sind die Adapter zuständig.

Für die beiden genannten Fälle erfolgt daher nachfolgend die detaillierte Beschreibung der konkreten Implementierung mit Unterstützung durch Klassendiagramme.

RabbitMQ

Die Verbindung der polymorphen Schnittstelle zum Datenverteiler ist durch diesen mit RabbitMQ vorgegeben. Zudem wird innerhalb des PG MATE Produkts mit der Verbindung von Verteilungsplattform und Kontrollplattform ebenfalls auf RabbitMQ gesetzt. Daher gilt die hier beschriebene Implementierung auch für die Verteilungsplattform und wird dort entsprechend referenziert.

RabbitMQ ist nach eigenen Angaben der meistgenutzte Message Broker (deutsch: Nachrichtenverteiler) (vgl. [PSa]). RabbitMQ basiert dabei auf dem AMQP Standard. Die derzeit eingesetzte Version dieses Standards ist 0-9-1. Andere Versionen oder zusätzliche Funktionen zur Spezifikation können über Plugins eingebunden werden. Eine der Hauptfunktion des Standards ist publish-subscribe (deutsch: veröffentliche und abonniere). Daten werden dabei auf einer Warteschlange (englisch: queue) veröffentlicht. Abonnenten der Warteschlange werden darüber dann informiert und die Daten zugestellt. Für mehr Informationen zu diesem Standard siehe [AAA+08].

RabbitMQ kann auf einfache Weise in die Programmiersprache Java durch die Bibliothek amqp-client eingebunden werden. Für eine detaillierte Installationsbeschreibung kann dem Handbuch in Anhang A.1.3 auf Seite 241 gefolgt werden. Um den Fokus auf die Arbeit von RabbitMQInputAdapter und RabbitMQOutputAdapter zu halten, werden die Einstellungsoptionen für RabbitMQ nicht genauer konkretisiert. Eine detailliertere Auflistung dieser Optionen kann ebenfalls dem Handbuch entnommen werden.

Das Veröffentlichen (englisch: publish) von Daten wird gemäß der Server-API von einer Implementierung des OutputAdapter übernommen. Diese Implementierung ist die Klasse RabbitMQOutputAdapter. Das Abonnieren (englisch: subscribe) von Daten wird gemäß der Server-API von einer Implementierung des InputAdapter übernommen. Das Abonnieren wird dabei nur implizit von der Implementierung RabbitMQInputAdapter übernommen. Implizit deshalb, weil diese nur die Initialisierung einer RabbitMQConsumer Klasseninstanz vornimmt. Die Klasse ist eine Erweiterung des DefaultConsumer aus der amqp-client Klassenbibliothek. Diese verbraucht bzw. nimmt Daten aus der abonnierten Warteschlange entgegen und ruft dann die Methode handleDelivery(final String consumerTag, final Envelope envelope, final AMQP.BasicProperties properties, final byte[] body) auf. Diese Methode wird vom RabbitMQConsumer implementiert und ruft einen InputListener auf, der die Übergabe der empfangenen Byte Daten zurück zum Server regelt. Der Listener wurde bei der Initialisierung des Adapters durch den Server mit der Methode setListener(final InputListener listener) gesetzt und wird dem RabbitMQConsumer übergeben.

Zur Einrichtung der Verbindung zu einem RabbitMQ-Server wird eine Implementierung der Schnittstelle RabbitMQConnectionFactory verwendet. Eine Implementierung der Schnittstelle ist die Klasse RabbitMQConnectionFactoryImpl. Neben verschiedenen Host-Informationen sowie Benutzername und Passwort, werden dort auch Sicherheitseinstellungen der Verbindung gesetzt. Dazu wird die Funktion create(final String username, final String password, final String host, final int port, boolean useSsl) aufgerufen, die eine Connection zurückgibt. Über dieses Klassenobjekt können dann Daten veröffentlicht oder abonniert werden. Abbildung 33 zeigt die Klassenstruktur der RabbitMQ Adapter mit den Schnittstellen der Server-API und den Klassen der amqp-client Bibliothek.

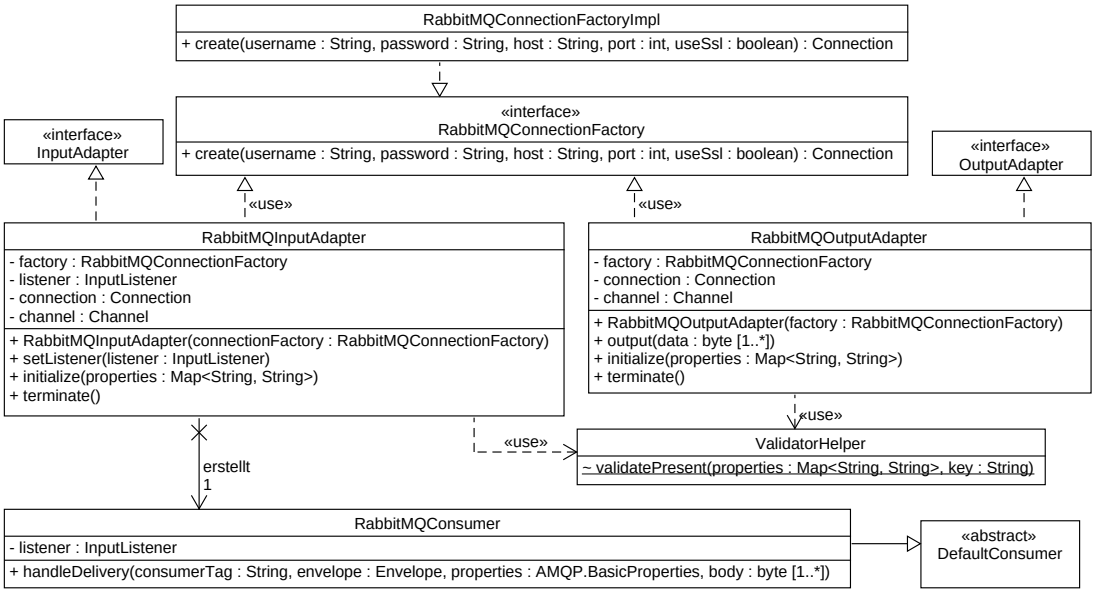


Abbildung 33: Klassendiagramm der RabbitMQ Adapter mit Bezug zur Server-API und den genutzten RabbitMQ-Bibliotheksklassen

Bei der Initialisierung der Adapter durch den Aufruf des Konstruktors wird eine RabbitMQConnectionFactory Schnittstelle mit übergeben. Im weiteren Schritt wird vom Server die Methode initialize(final Map<String, String> properties) aufgerufen. Dort wird zunächst die Existenz verschiedener Einstellungsoptionen aus der Map überprüft. Zu diesen Einstellungsoptionen zählen etwa der Benutzername, das Passwort oder der Host der gewünschten Verbindung. Die Überprüfung wird durch die Klasse ValidatorHelper unterstützt, indem diese eine statische Methode validatePresent(final Map<String, String> properties, final String key) zur Verfügung stellt. Die Methode löst eine Exception aus, falls der key sich nicht in den Einstellungen befinden sollte. Sind alle benötigten Einstellungsoptionen vorhanden, wird mit der RabbitMQConnectionFactory die Verbindung aufgebaut. Aus der Verbindung wird von beiden Adaptern ein Channel Objekt generiert, dass für das Veröffentlichen bzw. abonnieren der Daten genutzt wird.

Beim RabbitMQInputAdapter wird dann ein RabbitMQConsumer erstellt und die Warteschlange, die abonniert werden soll, auf dem Channel Objekt definiert. Dem RabbitMQConsumer wird dieses Objekt mit übergeben und er arbeitet dann wie oben beschrieben. Die Initialisierung für den RabbitMQInputAdapter ist damit beendet.

Beim RabbitMQOutputAdapter hingegen wird keine Warteschlange, sondern ein Exchange auf dem Channel Objekt definiert. Dieser bildet einen Eingangspunkt

beim Server, mit dem auf mehrere Warteschlangen veröffentlicht werden kann (vgl. [AAA+08]). Die Initialisierung für den RabbitMQOutputAdapter ist damit beendet. Sollen Daten auf veröffentlicht werden, wird vom Server die Methode output(final byte[] data) aufgerufen. Die Bytes werden dann über das Channel Objekt veröffentlicht.

UDP

Das MATE Produkt wird über UDP mit dem Testbed verbunden. Nach dem Systementwurf müssen dafür entsprechende Schnittstellen von der polymorphen Schnittstelle und der Verteilungsplattform bereitgestellt werden. Die Schnittstelle zum Empfangen von UDP Paketen wird gemäß der Server-API von einer Implementierung des OutputAdapter übernommen. Diese Implementierung ist die Klasse UdpInputAdapter. Die Schnittstelle zum Senden von UDP Paketen wird gemäß der Server-API von einer Implementierung des OutputAdapter übernommen. Die Implementierung der Schnittstelle wird durch die Klasse UdpOutputAdapter umgesetzt.

Beide Adapter erweitern zudem noch die abstrakte Klasse AbstractUdpAdapter. Dieser implementiert bereits die statische Funktion getPortProperty(final Map<String, String> properties). In dieser Funktion wird der gewünschte UDP Port ausgelesen, welcher von beiden Adaptern benötigt wird. Der UdpInputAdapter muss an einem Port nur auf ankommende Daten hören. Der UdpOutputAdapter muss zudem noch die IP-Adresse kennen, an dessen Port die UDP Pakete gesendet werden sollen. Daher implementiert dieser Adapter noch eine weitere private Methode getAddressProperty(final Map<String, String> properties). Die genauen Einstellungsmöglichkeiten für diese Adapter können dem Handbuch in Anhang A.1.3 auf Seite 243 entnommen werden.

Zur Erstellung einer Verbindung wird von beiden Adaptern die Klasse DatagramSocketFactory verwendet, welche bei der Initialisierung durch die Konstruktoren mit übergeben wird. Diese liefert bei Aufruf von createDatagramSocket(final SocketAddress socketAddress) ein Objekt der Klasse DatagramSocket zurück, welches zum Versenden bzw. Empfangen von UDP Paketen verwendet wird. Das Senden von Daten wird über den UdpOutputAdapter mit der Methode output(final byte[] data) ausgeführt. Um den vom Server gesetzten InputListener beim Empfang von Daten auszulösen, nutzt der UdpInputAdapter zudem einen ExecutorService. Dieser bekommt die private Methode receiveData(final int bufferSize) übergeben. Innerhalb der Methode wird auf eingehende UDP Pakete gehört. Der Datenanteil des Paketes wird beim Empfang dann dem Input-

Listener übergeben. Abbildung 34 zeigt den eben geschilderten Aufbau als Klassendiagramm.

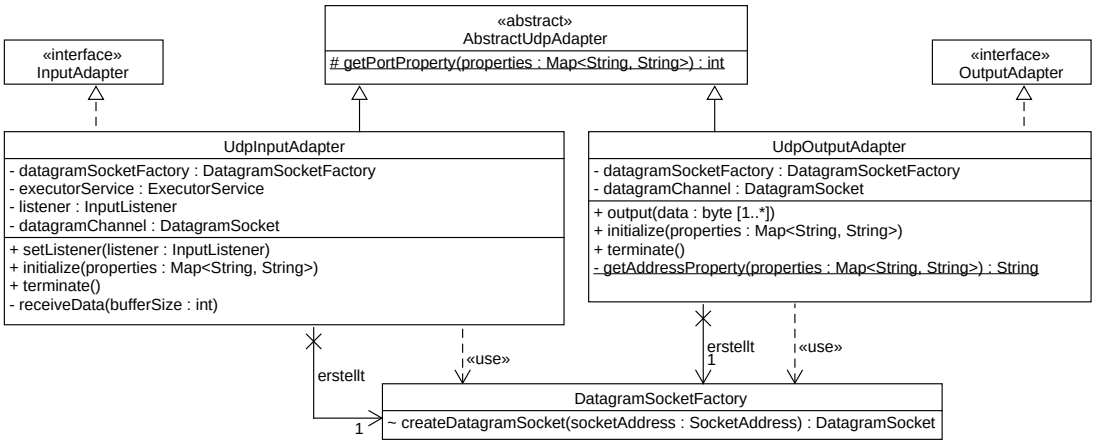


Abbildung 34: Klassendiagramm der UDP Adapter mit Bezug zur Server-API

7.2.4 Translator

Die polymorphe Schnittstelle liegt im Testbed und erhält vom Datenverteiler Nachrichten im S-100 Datenformat. Daher muss die polymorphe Schnittstelle Daten im S-100 Format entgegennehmen können. Das Produkt der PG MATE bietet hingegen nur eine Schnittstelle über NMEA2000 an. Die Schnittstelle dazu ist die Verteilungsplattform. Die polymorphe Schnittstelle muss daher Daten im NMEA2000 Format versenden können, um diese Schnittstelle zu bedienen. Entsprechend muss eine Übersetzung zwischen den einzelnen Datenformaten bzw. Protokollen durchgeführt werden. Zum Senden der Steuerbefehle vom MATE Produkt im NMEA2000 Format müssen diese wieder in das S-100 Format übersetzt und ausgegeben werden können.

Die polymoprhe Schnittstelle ist dazu so entwickelt, dass XML als zentrales Übersetzungsmedium zwischen den einzelnen Datenformaten bzw. Protokollen verwendet wird. Daher müssen Nachrichten im S-100 Format in eine äquivalente XML Darstellung umgewandelt werden können und vice versa. Außerdem müssen Nachrichten im NMEA2000 Format in eine äquivalente XML Darstellung umgewandelt werden können und vice versa.

Die genauen Konfigurationen der nachfolgend beschriebenen Übersetzer finden sich im Anhang A.1.3 auf Seite 247 und 249.

S-100 Translator Implementierung

Das Umwandeln zu einer S-100 XML Darstellung wird gemäß der Server-API von einer Implementierung des InputTranslator übernommen. Die Umwandlung der XML Darstellung in eine Nachricht im S-100 Datenformat wird gemäß der Server-API von einer Implementierung des OutputTranslator übernommen.

Nachrichten im S-100 Format werden als serialisierte EMF Modelle entgegengenommen. Für mehr Informationen zu EMF siehe [Fou17]. Der Grund für die Serialisierung der EMF Modelle ist, dass die Komponente Datenverteiler aus Abschnitt 6.1.2 die Sensor- und Umgebungsistwerte über ein EMF Ecore-Modell in ein oder mehrere S-100 Nachrichten bringt. Zum Versand dieser Nachrichten über ein Netzwerk werden dann die konkreten S-100 Ecore Modellinstanzen zunächst in ein reines Byte Format serialisiert. Im Umkehrschluss müssen die Daten beim Empfangen wieder deserialisert werden. Nach dem Deserialiseren der S-100 Nachrichten liegen Diese wieder in einer EMF Ecore Modellinstanz vor. Die Komponente Datenverteiler setzt für diese Vorgänge das eMIR Framework ein. Für mehr Informationen zu eMIR siehe [eMI17]. Die S-100 Translator Implementierung nutzt zur Erfüllung der Aufgaben ebenfalls eMIR. eMIR liefert dabei neben den Funktionen zum Serialisieren und Deserialisieren auch das EMF Ecore Modell für das S-100 Datenformat.

Das Umwandeln von bzw. zu einer äquivalenten XML Beschreibung einer S-100 Nachricht wird von einer Implementierung der XMLStringToEObject bzw. EObjectToXMLString Schnittstelle durchgeführt. Genauer wird dort unter Nutzung der Resource und GenericXMLResourceFactoryImpl Klassen aus der EMF Bibliothek entweder eine äquivalente XML Beschreibung aus einer S-100 Ecore Modellinstanz erstellt oder umgekehrt. Die S-100 Daten als XML können entsprechend der Transformationsregeln (siehe Abschnitt 6.2.9) auf andere Datenformate übersetzt werden. Andere Datenformaten können über entsprechende Transformationsregeln in eine S-100 XML Beschreibung übersetzt werden. Abbildung 35 zeigt den Zusammenhang der Klassen detaillierter im Klassendiagramm.

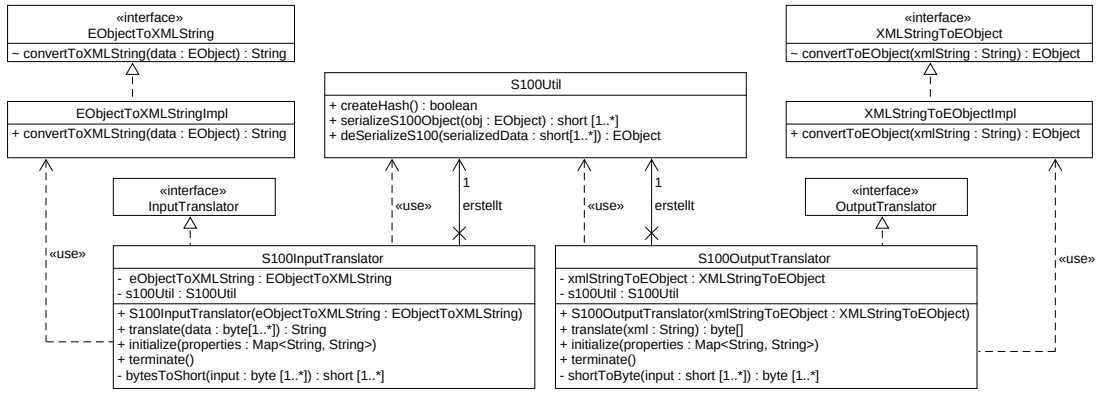


Abbildung 35: Klassendiagramm des S-100 Translators mit Bezug zur Server-API und den Schnittstellen zur eMIR Implementierung

Die Klasse S100InputTranslator ist eine Implementierung der InputTranslator Schnittstelle aus der Server-API. Ebenso ist die Klasse S100OutputTranslator eine Implementierung der OutputTranslator Schnittstelle aus der Server-API. Beide Klassen erstellen eine S100Util Instanz, welche die Aufgabe der Serialisierung und Deserialisierung wie oben beschrieben übernimmt. Dazu werden die öffentlichen Funktionen serializeS100Object(EObject obj) und deSerializeS100(short[] serializedData) aufgerufen. Die S100Util Instanz wird dabei zunächst durch einen Aufruf von createHash() beim Initialisieren der Übersetzer initialisiert.

Zum Übersetzen von serialisierten Bytes einer S-100 Nachricht wird bei der Klasse S100InputTranslator die Methode translate(final byte[] data) aufgerufen. Die serialisierten Bytes werden innerhalb der Funktion durch einen internen Aufruf von bytesToShort(final byte[] input) in ein Array des Datentyps short gebracht. Durch die Umwandlung in das Array ist ein Aufruf der deSerializeS100(short[] serializedData) Funktion zum Deserialisieren möglich. Das von der Funktion zurückgegebene EMF EObject enthält die S-100 Instanz des Ecore Modells. Diese kann unter Nutzung der convertToXMLString(EObject data) Funktion in eine äquivalente XML Beschreibung umgewandelt werden. Die genannte Funktion wird von einer Implementierung der EObjectToXMLString Schnittstelle bereitgestellt. Eine Implementierung dieser Schnittstelle ist die Klasse EObjectToXMLStringImpl, welche beim Aufruf des Konstruktors mit S100InputTranslator(final EObjectToXMLString eObjectToXMLString) übergeben wird. Die erzeugte XML Beschreibung wird als String zum ursprünglichen Aufruf zurückgegeben.

Zum Serialisieren einer S-100 XML Beschreibung wird bei der Klasse S100OutputTranslator die Methode translate(final String xml) aufgerufen. Die XML Beschreibung als String wird dann durch den Aufruf der Funktion convertToEObject(String xmlString) in eine äquivalente EMF Ecore Modellinstanz umgewandelt. Die genannte Funktion wird von einer Implementierung der XMLStringToEObject Schnittstelle bereitgestellt. Eine Implementierung dieser Schnittstelle ist die Klasse XMLStringToEObjectImpl, welche beim Aufruf des Konstruktors mit S100OutputTranslator(final XMLStringToEObject xmlStringToEObject) übergeben wird. Die S-100 Ecore Modellinstanz kann an die Funktion serializeS100Object(EObject obj) aus der Klasse S100Util zum Serialisieren übergeben werden. Von der Funktion wird ein Array des Datentyps short zurückgegeben. Zum Versand über ein Netzwerk wird dieses Array noch mit der Funktion shortToByte(short[] input) in ein Byte Array umgewandelt. Die Bytes werden zum ursprünglichen Aufruf zurückgegeben.

NMEA 2000 Translator Implementierung

Das Umwandeln von Nachrichten im NMEA2000 Format in eine äquivalente XML Darstellung wird unter Nutzung der XStream Bibliothek umgesetzt. Für den umgekehrten Fall wird diese Bibliothek ebenfalls verwendet. Diese ermöglicht es Instanzen der nachfolgend beschriebenen Klassen durch Nutzung von dessen Attributen in eine Äquivalente XML Beschreibung zu serialisieren. Im umgekehrten Fall können so auch Klasseninstanzen aus einer NMEA2000 XML Beschreibung generiert bzw. deserialisert werden (vgl. [XSt17]). Dabei werden nicht alle Attribute der nachfolgenden Klassen in die XML Beschreibung aufgenommen. In der nachfolgenden Beschreibung des NMEA2000 Models wird herausgestellt, welche Attribute in die Beschreibung mit aufgenommen werden. Zur Abbildung des NMEA2000 Standards sind die Klassen an die XML Beschreibung aus Abschnitt 7.1 angelehnt. Die Klassenstruktur des Models ist in Abbildung 36 dargestellt.

Da der NMEA 2000 Translator auch bei der Verteilungsplattform eingesetzt wird, wird an dieser Stelle der Translator als ganzes beschrieben und in Abschnitt 7.3 hierauf verwiesen.

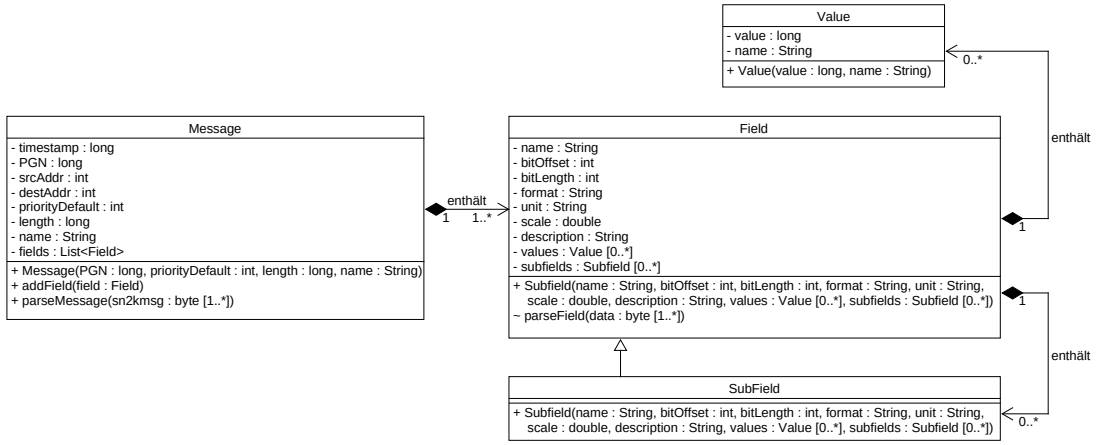


Abbildung 36: Klassendiagramm des NMEA2000 Models zur Abbildung des Standards auf Implementierungsebene

Die Klasse Message referenziert den Header Anteil einer NMEA2000 Nachricht. Neben dem Namen ist hier vor allem die PGN ein wichtiges Attribut zur späteren Übersetzung von und zu einer äquivalenten XML Beschreibung. Diese beiden Attribute werden auch unter Nutzung der XStream Bibliothek in die XML Beschreibung aufgenommen. Jede Message besteht aus einem oder mehreren Fields. Diese Klassen haben mit name ebenfalls wieder einen eindeutigen Namen zur Identifikation. Daneben sind hier die Eigenschaften scale und unit am wichtigsten. Die nicht genannten Attribute der Klassen sind nachrangig und werden daher nicht weiter erläutert. Mit dem scale werden alle Werte vor dem Übersetzen in eine äquivalente XML Beschreibung multipliziert, um den korrekten Wert des NMEA2000 Datums zu erhalten. Ebenso muss dann beim Umwandeln einer NMEA2000 XML Beschreibung in reine Bytes das Inverse des scale wieder multipliziert werden, um dem Standard zu genügen. Die unit gibt an, um welchen Datentyp es sich bei dem Wert im Field handelt.

Der Standard sieht mehr Datentypen vor, als die Programmiersprache Java vorsieht. Daher müssen durch geeignete shifting-Operationen (deutsch: Verschiebungen) der Bits die Datentypen der Programmiersprache Java händisch erweitert werden. Dieses Problem bezieht sich auch auf die Werte der Klasse Value. Ein Field kann ein oder mehrere Values haben. Diese beinhalten neben einem Attribut name auch ein Attribut value. Letzteres ist ein Enumeration (enum), das im NMEA2000 Standard höchstens 8 Bit besitzt – meist aber weniger. Der kleinste Datentyp short in der Programmiersprache Java besitzt 8 Bit. Daher müssen beim Umwandeln von oder zu einer XML Beschreibung meist wieder Verschiebungen der Bits durchgeführt werden, um den Wert korrekt darzustellen. Der Wert

des Fields oder des Values wird in der XML als Value-Attribut neben dem Namen des Field-Elements mit aufgenommen.

Ein Field kann auch aus mehreren SubFields bestehen. Diese erben alle Eigenschaften eines Fields, ohne diese zu erweitern. Im NMEA2000 Standard sind diese Felder als Erweiterung in einigen Nachrichten vorgesehen, um so mehr Informationen durch eine weitere Ebene übertragen zu können. Für eine detaillierte Beschreibung dazu siehe Abschnitt 7.1. Existiert ein solches SubField, wird dieses als Kindelement des Field-Elements in der XML Beschreibung der NMEA2000 Nachricht hinzugefügt. Bei der Initialisierung der Übersetzer werden die Eigenschaften für das XML Dokument in einer namensgleichen XStream Klasseninstanz gesetzt. Dazu werden die Konstruktor der Übersetzer aufgerufen.

Zu erkennen sind in den Klassen Message bzw. Field auch die Funktion parseMessage(final byte[] sn2kmsg) bzw. parseField(byte[] data). Diese Methoden werden zur Laufzeit zum Umwandeln von NMEA2000 Bytes in die value Attribute in den Modellklassen aufgerufen. Dabei wird innerhalb von parseMessage(final byte[] sn2kmsg) zunächst der Header-Anteil der Bytes umgewnadelt und die Attribute der Message damit gefüllt. Darin enthalten sind unter anderem die Länge der Nachricht in Bytes und die PGN. Für den Datenanteil der NMEA2000 Nachricht wird für jedes Field einer Message die Methode parseMessage(final byte[] sn2kmsg) aufgerufen. Auf diese Weise werden für jedes Field mit den dort angegebenen bitOffset die richtigen Werte aus dem Byte Array eingelesen. Nach diesem Prozess existiert eine NMEA2000 Message, welche zur NMEA2000 Byte Nachricht äquivalent ist.

Der oben geschilderte Ablauf wird zur Laufzeit von der Klasse NMEA2000Parser angestoßen. Diese stellt für den geschilderten Ablauf ebenfalls eine Funktion parseMessage(final byte[] sn2kmsg) bereit. Die Funktion gibt eine Message zurück, die die umgewandelten Werte aus dem Byte Array sn2kmsg enthält. Von der Klasse wird ebenfalls eine Funktion parseMessage(final Message message) implementiert. Diese wandelt die in der Modell-Repräsentation enthaltenen Werte der Fields in Bytes um und gibt die Bytes als Array zurück. Dazu werden von der Funktion zunächst die Header-Informationen aus der Message in Bytes umgewandelt und in ein Byte Array gelegt. Dieses Byte Array wird der privaten Methode parseField(final int fieldOffset, final int bitLength, final int bitOffset, final int byteOffset, final Field field, final byte[] sn2kmsg) übergeben. Diese wandelt mit Hilfe der Field Informationen das value-Attribut aus der Klasse Field in Bytes um und legt es an der korrekten Stelle im Byte Array sn2kmsg ab. Auf diese Weise wird eine NMEA2000 Message Klasseninstanz in ein äquivalentes Byte Array serialisiert.

Der NMEA2000Parser kapselt die Umwandlung des internen NMEA2000 Models in Bytes und vice versa. Dabei erstellen NMEA2000InputTranslator und NMEA2000OutputTranslator bei ihrer Initialisierung jeweils eine Instanz dieser Klasse. Die Klasse NMEA2000InputTranslator ist eine Implementierung der Schnittstelle InputTranslator aus der Server-API. Der NMEA2000OutputTranslator ist eine Implementierung der MultipleOutputTranslator Schnittstelle aus der Server-API. Damit stellt der NMEA2000OutputTranslator eine Besonderheit dar, da mehrere Nachrichten über die translate(String xml) Funktion zurückgegeben werden können. Der Grund für diese Besonderheit ist, dass es bei einigen Übersetzungen von zum Beispiel einer S-100 Nachricht in NMEA2000 semantisch notwendig ist, diese in mehrere NMEA2000 Nachrichten zu teilen. Abbildung 37 verdeutlicht die Struktur von NMEA2000InputTranslator und NMEA2000OutputTranslator mit den Hilfsklassen.

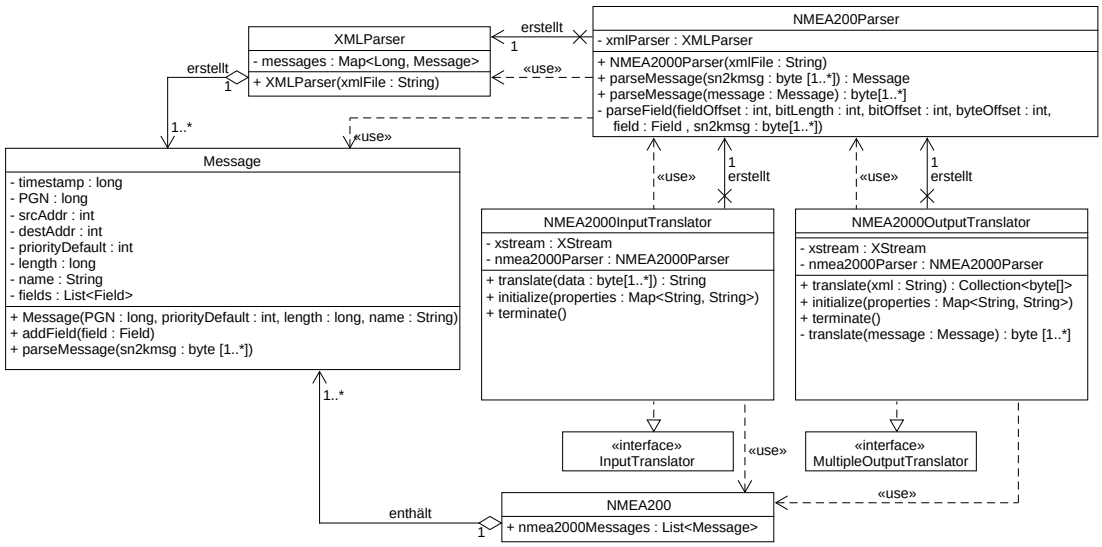


Abbildung 37: Hauptklassen des NMEA2000 Translators

Zum Übersetzen von einer NMEA2000 Byte Nachricht wird bei der Klasse NMEA2000InputTranslator die Methode translate(final byte[] data) aufgerufen. Die Bytes werden dann durch Aufruf der Funktion parseMessage(final byte[] sn2kmsg) über die NMEA2000Parser Instanz in eine Message Instanz gebracht. Die Message Instanz wird an den NMEA2000InputTranslator zurückgegeben und in das nmea2000Messages Listenattribut einer NMEA2000 Klasseninstanz angehängt. Diese Klasse ermöglicht es mehrere NMEA2000 Nachrichten in eine XML einzubauen. Dazu wird vom NMEA2000InputTranslator im nächsten Schritt die bei der Initialisierung erstellte

XStream Instanz mit der Funktion toXML(Object obj) aufgerufen und die NMEA2000 Klasseninstanz als obj übergeben. Von dieser Funktion wird die zur umgewandelten NMEA2000 Byte Nachricht äquivalente XML Beschreibung als String zurückgegeben. Die erzeugte XML Beschreibung wird zum ursprünglichen Aufruf zurückgegeben.

Soll eine NMEA2000 XML Beschreibung in Bytes umgewandelt werden, wird die Funktion translate(final String xml) im NMEA2000OutputTranslator aufgerufen. Dort wird zunächst durch dem Aufruf von fromXML(String xml) aus der bei der Initialisierung erstellten XStream Klasseninstanz eine NMEA2000 Klasseninstanz generiert. Diese Klasseninstanz enthält eine oder mehrere Message Objekte mit den entsprechenden Werten in den Field Klasseninstanzen. Für jede dieser Message Instanzen wird dann die private Funktion translate(final Message message) im NMEA2000OutputTranslator aufgerufen. Diese wandelt die übergebene Message durch Aufruf von parseMessage(final Message message) im NMEA2000Parser und den weiter oben beschriebenen Ablauf in ein äquivalentes Byte Array um. Dieses Array wird zurückgegeben und einer Collection<byte[]> Variable hinzugefügt. Nach Umwandlung jeder Message in der NMEA2000 Klasseninstanz wird die Kollektion zum ursprünglichen Aufruf zurückgegeben.

In Abbildung 37 ist ebenfalls eine Klasse XMLParser zu sehen. Bisher wurde zum besseren Verständnis außer acht gelassen, wie die zusätzlichen Informationen wie etwa die Byte Länge eines Fields oder dessen unit als Datentyp in das interne NMEA2000 Modell zu jeder Message gelangen. Zu Anfang dieses Unterabschnittes wurde bereits erklärt, dass die Modellklassen mit Message, Field, SubField und Value an die XML Beschreibung des NMEA2000 Standards aus Abschnitt 7.1 angelehnt sind. Mit Hilfe des XMLParsers wird die XML Beschreibung des Standards sukzessive eingelesen. Das bedeutet für jedes Nachrichtenelement in dieser XML wird eine Message erstellt und die entsprechenden Informationen in die Klasse überführt. Dazu zählen neben den Attributen des Nachrichtenelements auch die Kind-Elemente, die die Field Klassen abbilden. Dieser Vorgang wird beim Aufruf des Konstruktors mit XMLParser(final String xmlFile) gestartet. Das übergebene xmlFile stellt den Pfad zur genannten XML Beschreibung des Standards dar. Über getMessages() ist nach Durchführung des Konstruktors der Abruf der Nachrichten über eine Map<Long, Message> möglich. Als Schlüssel dieser Map dient die PGN.

Der XMLParser wird vom NMEA2000Parser bei dessen Initialisierung mit initialisiert. Der gesamte Initialisierungsprozess wird sowohl vom NMEA2000InputTranslator

als auch vom NMEA2000OutputTranslator durch die Methode initialize(final Map<String, String> properties) in Gang gesetzt.

7.3 Verteilungsplattform

In diesem Abschnitt wird die Umsetzung des Entwurfs der Verteilungsplattform aus Abschnitt 6.3 beschrieben. Dabei bleiben das Konzept und die Basisimplementierung zur polymorphen Schnittstelle gleich. Die Umsetzung der polymorphen Schnittstelle wurde bereits in Abschnitt 7.2 beschrieben. Die dazugehörige Konfigurationsdatei muss allerdings abgeändert werden, damit die Pipes wie in Abschnitt 6.3.4 umgesetzt werden. Die genaue Definition der Pipes kann der entsprechenden Konfigurationsdatei in Anhang A.7 zur Verteilungsplattform entnommen werden.

Aufgrund der Ähnlichkeit zur Implementierung der polymorphen Schnittstelle werden im Folgenden nur die zusätzlichen Adapter sowie Translator (deutsch: Übersetzer) beschrieben. Da zudem für die Verteilungsplattform noch eine Konfiguration eines RabbitMQ-Servers notwendig ist, wird diese zum Abschluss dieses Abschnitts noch einmal beschrieben.

7.3.1 Adapter

In Abschnitt 6.3 wurden bereits die notwendigen Adapter für die Komponente Verteilungsplattform beschrieben. Mit diesen Adaptern ist eine Kommunikation mit den unterschiedlichen Komponenten innerhalb des PG MATE Produkts und dem Testbed möglich. Speziell werden für die Kommunikation innerhalb des PG MATE Produkts ein HTTP Adapter, ein UDP Adapter und ein RabbitMQ Adapter benötigt. In Abschnitt 7.2.3 wurden bereits die letzten beiden Adapter beschrieben. Daher wird nachfolgend nur die Implementierung des HTTP Adapters beschrieben.

HTTP

Zur Verbindung der Verteilungsplattform mit der Pfadplanungskomponente aus Abschnitt 6.5 soll HTTP als verbindungsorientierter Datenaustausch verwendet werden. Diese Lösung garantiert die Zustellung von Daten mittels des Request/Response (deutsch: Frage/Antwort) Ablaufs bei einer solchen Verbindung. Dieser Ablauf wird durch die HTTP POST Methode mit Nutzung von Servlets ermöglicht.

Die Schnittstelle zum Entgegennehmen eines HTTP POST Request wird gemäß der Server-API von einer Implementierung des OutputAdapter übernommen. Diese Implementierung ist die Klasse HttpInputAdapter. Die Schnittstelle zum Ausführen eines HTTP POST Request wird gemäß der Server-API von einer Implementierung des OutputAdapter übernommen. Die Implementierung der Schnittstelle wird durch die Klasse HttpOutputAdapter umgesetzt.

Bei beiden Adaptern müssen mehrere Einstellungen vorgenommen werden. Dazu zählt unter anderem der Port auf dem Requests entgegengenommen oder abgesetzt werden sollen. Die weiteren Einstellungen können dem Handbuch im Anhang A.1.3 auf Seite 239 entnommen werden. Zur Verifikation der Einstellungen nutzen beide Adapter die Klasse ValidatorHelper, um auf die Existenz verschiedenster Einstellungsoptionen zu prüfen. Die Methoden dieser Klasse können dem Klassendiagramm in Abbildung 38 entnommen werden.

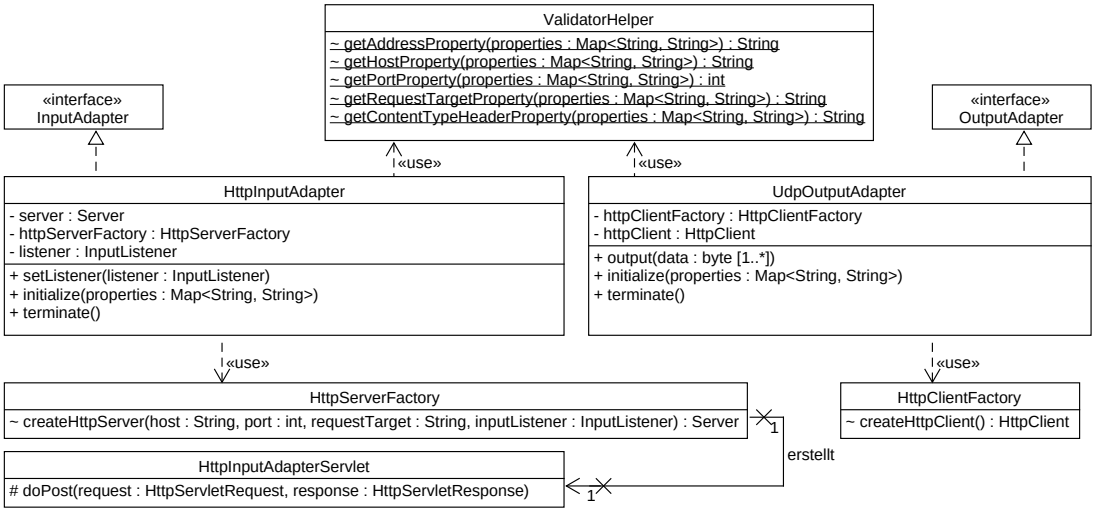


Abbildung 38: Klassendiagramm der HTTP Adapter mit Bezug zur Server-API

Die Verifikation der Einstellungsoptionen wird bei der Initialisierung der Adapter mit der Methode initialize(final Map<String, String> properties) vorgenommen. Dort werden die entsprechend notwendigen Überprüfungen durch die statischen Methoden der Klasse ValidatorHelper durchgeführt.

Für den HttpInputAdapter wird dem Konstruktor eine HttpServerFactory Klasseninstanz übergeben. Diese Klasseninstanz wird dann bei der Initialisierung für die Erstellung eines Server Objektes durch Aufruf der Funktion createHttpServer(final String

host, final int port, final String requestTarget, final InputListener inputListener) genutzt. Innerhalb dieser Funktion wird der InputListener beim erstellten Server über eine HttpInputAdapterServlet Klasseninstanz registriert. Die Klasseninstanz ist einer Erweiterung des HttpServlet aus der Java-Klassenbibliothek. In der Klasse HttpInputAdapterServlet wird die Methode doPost(final HttpServletRequest request, final HttpServletResponse response) überschrieben, welche die Daten eines HTTP POST Requests entgegennimmt. Innerhalb dieser Methode werden dann die Daten aus dem request dem InputListener zur weiteren Verarbeitung übergeben. Durch die Methode setListener(final InputListener listener) wird der InputListener von der Server-Implementierung der Verteilungsplattform bei der Initialisierung des HttpInputAdapter gesetzt. Zum Abschluss der Initialisierung wird der HTTP Server mit der Methode start() auf dem Server Objekt gestartet. Wird die Methode terminate() aufgerufen, wird der HTTP Server mit der Methode stop() auf dem Server Objekt wieder beendet.

Dem HttpOutputAdapter wird über den Konstruktor eine HttpClientFactory Klasseninstanz übergeben. Diese Klasseninstanz wird dann bei der Initialisierung dazu genutzt mit createHttpClient() ein HttpClient Objekt zu erzeugen. Damit ist die Initialisierung des HttpOutputAdapter abgeschlossen. Durch Aufruf der Methode output(final byte[] data) wird dann über das erzeugte HttpClient Objekt ein HTTP POST Request an die gewünschte Adresse abgesetzt. Wird die Methode terminate() aufgerufen, werden alle initialisierten Objekte zurückgenommen.

7.3.2 Translator

In Abschnitt 6.3 wurden bereits die notwendigen Translator (deutsch: Übersetzer) für die Komponente Verteilungsplattform beschrieben. Mit diesen Übersetzern und der Nutzung von XML als zentrales Übersetzungsmedium kann leicht in andere Protokolle bzw. Datenformate übersetzt werden. Für die Verteilungsplattform werden zur Verständigung unter den einzelnen Komponenten mehrere Übersetzer benötigt. Speziell handelt es sich dabei um einen NMEA0183 Übersetzer, einen XML-String Übersetzer und einen NMEA2000 Übersetzer. Die Implementierung des letzten Übersetzers wurde bereits in Absatz 7.2.4 beschrieben. Für die noch nicht beschriebenen Übersetzer erfolgt daher im Folgenden die Beschreibung auf Implementierungsebene.

NMEA0183-Translator Implementierung

Der NMEA0183-Translator wird mit der Klasse NMEA0183OutputTranslator umgesetzt. Diese implementiert das Interface MultipleOutputTranslator. Der MultipleOutputTranslator ermöglicht es eine XML-Nachricht in Form eines Strings in Bytes zu transformieren. Diese Bytes können von einem OutputAdapter an ein anderes System übergeben werden.

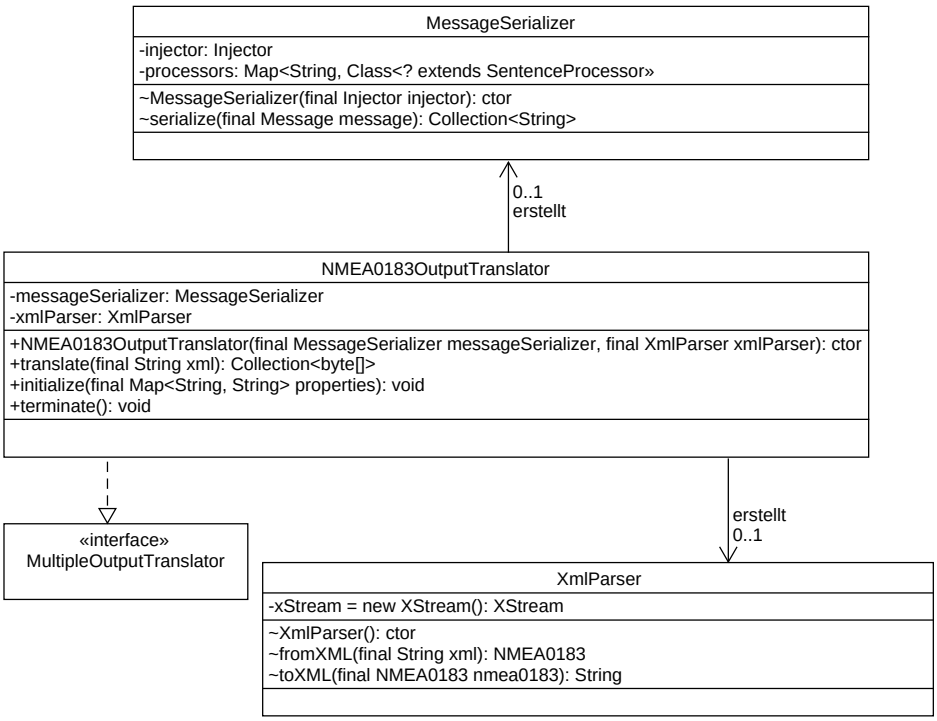


Abbildung 39: Klassendiagramm des NMEA0183OutputTranslators

Der NMEA0183OutputTranslator nutzt die Methode translate(final String xml), um eine übergebene NMEA0183-XML-Nachricht in eine NMEA0183-Nachricht umzuwandeln.

Dabei wird zunächst aus der NMEA0183-XML-Nachricht ein Objekt der Klasse NMEA0183 erzeugt. Die Klasse NMEA0183 bildet die Oberklasse des abgebildeten NMEA0183-Datenmodells. Das Umwandeln von NMEA0183-XML-Nachrichten in Objekte der Klasse NMEA0183 wird unter Nutzung der XStream-Bibliothek umgesetzt. Diese ermöglicht es Klasseninstanzen aus einer NMEA0183-XML-Beschreibung zu generieren bzw. zu deserialisieren.

Das NMEA0183-Datenmodell wird mit Hilfe verschiedener Klassen implementiert. Die Klassenstruktur des NMEA0183-Datenmodell ist in Abbildung 40 dargestellt.

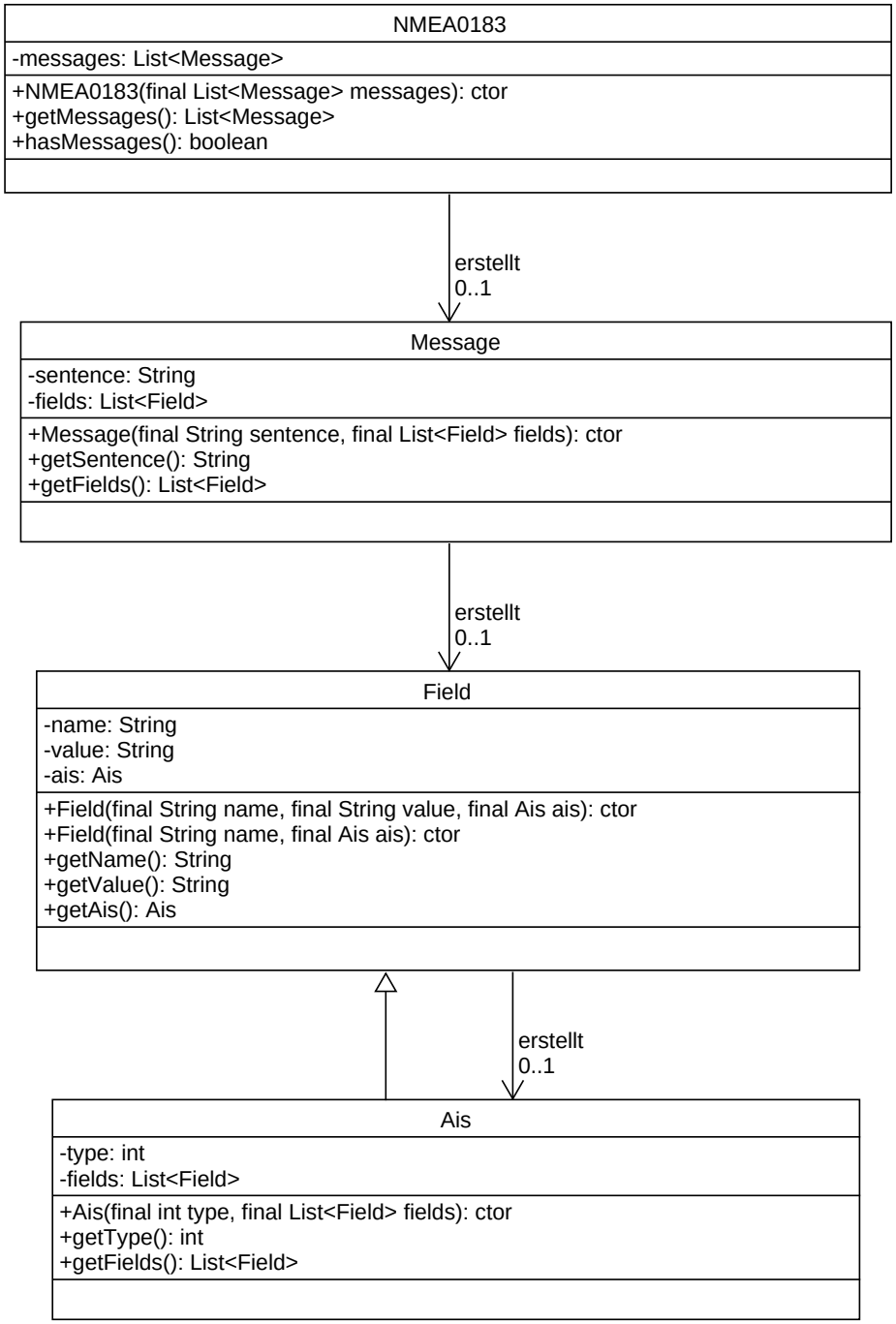


Abbildung 40: Klassendiagramm des NMEA0183-Datenmodells

Dabei beinhaltet ein Objekt der Klasse NMEA0183 eine List mit einem oder mehreren Objekten der Klasse Message. In einer Message können wiederum ein oder mehrere Objekte der Klasse Field in einer List enthalten sein. Zudem verfügt eine Message über das Attribut sentence. In diesem Attribut wird auf den jeweiligen NMEA0183-Datensatz verwiesen (Beispiel: VDM). Eine Hierarchieebene unter der Message Klasse gibt es noch die Klasse AIS für die AIS-Datensätze VDM und VDO. Die Objektinstanzen der Klasse Field enthalten die einzelnen Parameter der NMEA0183-Datensätze in Form von Paaren der Attribute Name und Value, wobei das Attribut Value dabei nicht immer belegt sein muss.

Die letztendliche Übersetzung und Erstellung der NMEA0183-Nachricht findet mit Hilfe des MessageSerializer statt. Zunächst wird mit der Methode hasMessages() der NMEA0183-Klasse geprüft, ob eine oder mehrere Message vorhanden sind. Wenn diese zutrifft, werden diese in einer Schleife abgerarbeitet. Dabei wird jede Message vom MessageSerializer mit der Methode serialize verarbeitet. Abhängig von dem im Attribut sentence gesetzten Wert wird dann der für den NMEA0183-Datensatz spezifische SentenceProcessor aufgerufen. Abbildung 41 zeigt am Beispiel des VDMSentenceProcessors wie ein SentenceProcessor für die verschiedenen NMEA0183-Nachrichten implementiert wurde.

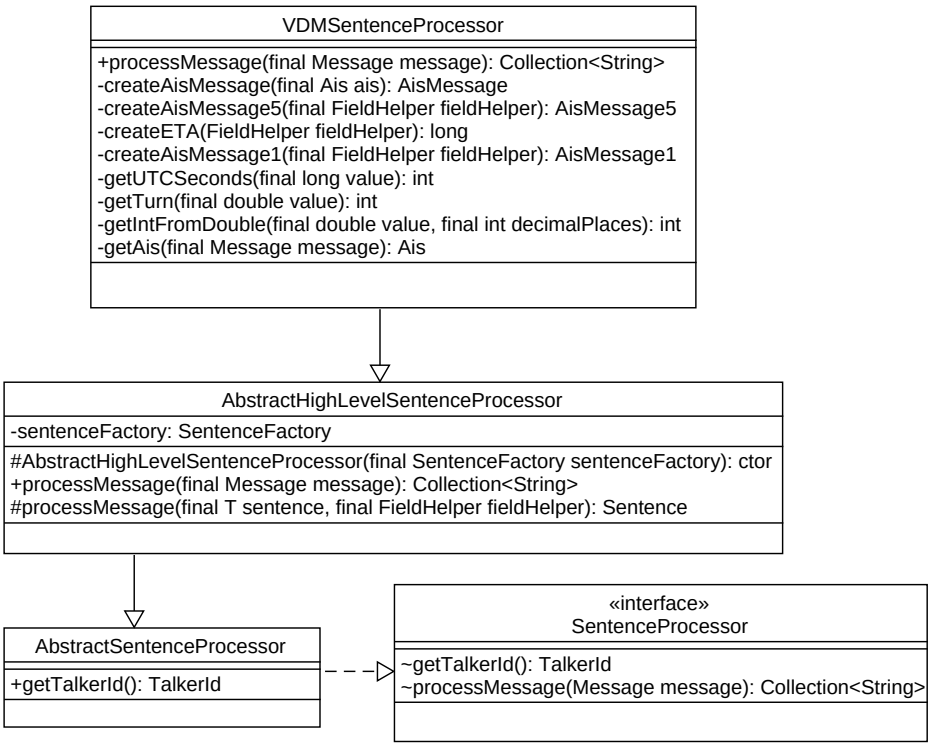


Abbildung 41: Klassendiagramm des NMEA0183-Processors

Der $\mathrm{VDMSentenceProcessors}$ baut aus der übergebenen $\mathrm{Message}$ Schritt für Schritt den NMEA0183-Datensatz zusammen. Dieser wird im letzten Schritt in Byte-Code umgewandelt.

XML-String Translator Implementierung

In Absatz 6.3.3 wurde bereits die Aufgabe des XML-String Translators (deutsch: Übersetzer) beschrieben. Da keine Übersetzungen notwendig sind, nutzt dieser Übersetzer nur die von der Server-API vorgegebenen Strukturen zur Erfüllung seiner Aufgabe. Das bedeutet für die Implementierung der $\mathrm{InputTranslator}$ Schnittstelle, dass dem von der Server-API vorgegebenen $\mathrm{InputAdapter}$ nur die bereits in XML vorliegenden Daten übergeben werden. Für die Implementierung der $\mathrm{OutputTranslator}$ Schnittstelle aus der Server-API, wird dann entsprechend nur die Funktion $\mathrm{translate(final\ String\ xml)}$ aufgerufen, die die Bytes des Strings zurückgibt. Entsprechend wird an dieser Stelle auf eine weitere Beschreibung mittels Diagrammen verzichtet und auf das Handbuch im Anhang A.1.3 auf Seite 250 verwiesen.

7.3.3 RabbitMQ Implementierung

Die PG MATE verwendet eine RabbitMQ-Serverinstanz auf einem Server der Universität Oldenburg der unter der URL „https://duemmer.informatik.uni-oldenburg.de:15672/“ zu erreichen ist.

Es werden insgesamt vier Exchangen und vier Queues definiert, alle sind als durable deklariert. Für alle Exchangen wird der Typ direct verwendet, da er den Anforderungen der PG MATE entspricht. Im Folgenden werden die Aufgaben und Parameter der Exchangen und Queues aufgelistet:

- „route-current-exchange“ nimmt Nachrichten von der Verteilungsplattform entgegen und leitet sie an die Queue „route-current-state“ mit dem gleichnamigen routing key „route-current-state“ weiter. Inhalt der Nachrichten sind der Routenstatus und die aktuellen Wegpunkte, die von der Pfadplanungskomponente berechnet werden. Die EPD empfängt die Nachrichten und verarbeitet diese. Die Time-to-Live dieser Exchange beträgt 300.000 Millisekunden.
- „route-target-exchange“ nimmt Nachrichten von der EPD entgegen und leitet sie an die Queue „route-target-state“ mit dem gleichnamigen routing key „route-target-state“ weiter. Inhalt der Nachrichten sind diejenigen Routen, die in der EPD ausgewählt und für die das teilautonome Fahren aktiviert wurde. Die Verteilungsplattform empfängt die Nachrichten und leitet sie wiederum an die Pfadplanungskomponente weiter, welche sie verarbeitet.
- „zuse-current-exchange“ nimmt Nachrichten von der Verteilungsplattform entgegen und leitet sie an die Queue „zuse-current-state“ mit dem gleichnamigen routing key „zuse-current-state“ weiter. Inhalt der Nachrichten sind alle Daten aus dem Testbed, welche die polymorphe Schnittstelle gebündelt an die Verteilungsplattform gesendet hat. Die EPD empfängt die Nachrichten und verarbeitet diese. Die Time-to-Live der Daten auf dieser Exchange beträgt 1.000 Millisekunden.
- „zuse-target-exchange“ nimmt Nachrichten von der EPD entgegen und leitet sie an die Queue „zuse-target-state“ mit dem gleichnamigen routing key „zuse-target-state“ weiter. Inhalt der Nachrichten sind die Steuerdaten für Richtung und Geschwindigkeit, die in den Virtual Handles eingegeben wurden. Die Verteilungsplattform empfängt die Nachrichten und leitet sie wiederum an die Controller-Komponente weiter, welche diese verarbeitet. Zur Veranschaulichung des Nachrichtenflusses der manuellen Steuerdaten siehe Abbildung 42.

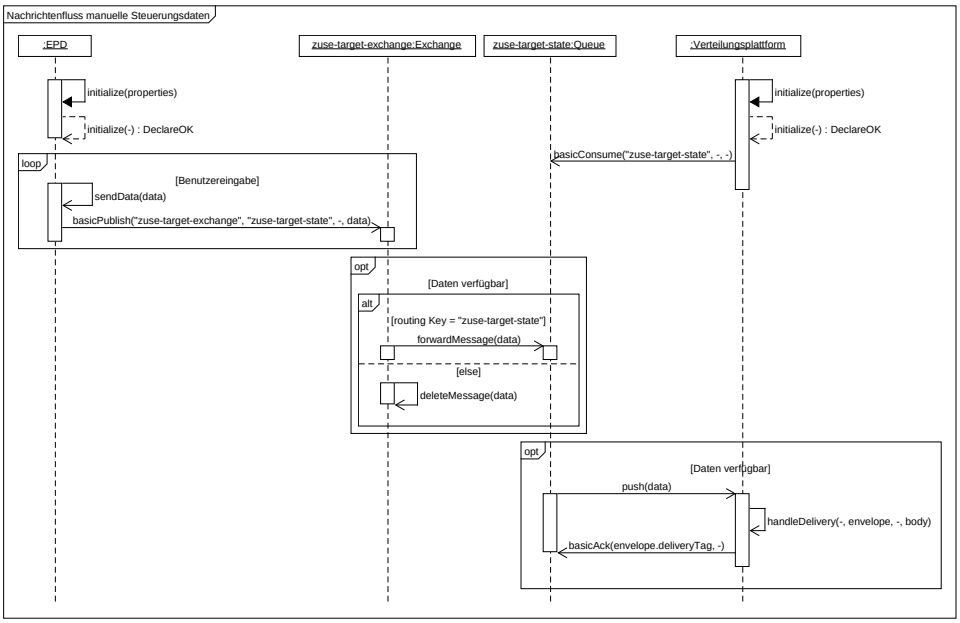


Abbildung 42: Sequenzdiagramm zur Veranschaulichung des Nachrichtenflusses von manuellen Steuerungsdaten über den RabbitMQ-Server

7.4 Kontrollplattform

In diesem Kapitel wird die Erweiterung der EPD erläutert. Die ersten beiden Unterkapitel befassen sich mit dem Ausgangszustand und Umgang mit neuem Code. In den folgenden Unterkapiteln werden die Implementierungen der Submodule in der EPD beschrieben. Diese setzen sich aus der Api, Vessel-Connection, Virtual Handles und RoutePilot zusammen. In Abbildung 44, Abbildung 46, Abbildung 48 und Abbildung 49 sind Screenshots der fertigen Implementierung einiger Elemente zu sehen.

7.4.1 Mate-Master

Für die Weiterentwicklung der EPD ersuchte die PG einen funktionierenden und stabilen Stand. Das OFFIS stellte der Gruppe alle Branches zur Verfügung und empfohl die Verwendung des Branches „enavunderway“. Aus diesem erstellte die PG einen neuen eigenen Master, den „mate-master“. In diesen werden alle Änderungen und Erweiterungen der resultierenden Branches gemerged, sodass der Mate-Master-Branch immer einen stabilen Stand liefert.

7.4.2 Eingriffe im bestehenden Quelltext - @ApiHook

In Abschnitt 6.4.2 wurde beschrieben, dass alle Änderungen in bestehenden Klassen so gering wie möglich gehalten werden. Konzeptionell wurde die Api über eine Schnittstelle erweitert, die eine Abkapselung externer Module ermöglicht und in dem folgenden Unterkapitel beschrieben wird. Dennoch erfordern auch die erstellten Interfaces an bestimmten Quelltextstellen einen Eingriff. Um diese Eingriffe so transparent wie möglich zu halten, erstellte die PG eine Annotation $@\mathrm{ApiHook}$. Jede Änderung wurde in Methoden ausgelagert und mit der Annotation gekennzeichnet.

7.4.3 Api

Das Submodul EPD-Api stellt nur Interfaces zur Verfügung, welche in den jeweiligen EPD-Modulen oder in den Plugins implementiert werden. Diese Implementierungen befinden sich in der Ordnerstruktur $\mathrm{de.uni\_oldenburg.mate.epd.common/shore}$. Die Initialisierung erfolgt über den $\mathrm{EPDApiShoreConnector}$, welcher mit Hilfe von Guice die Module sowie externe Plugins lädt (siehe Abbildung 43).

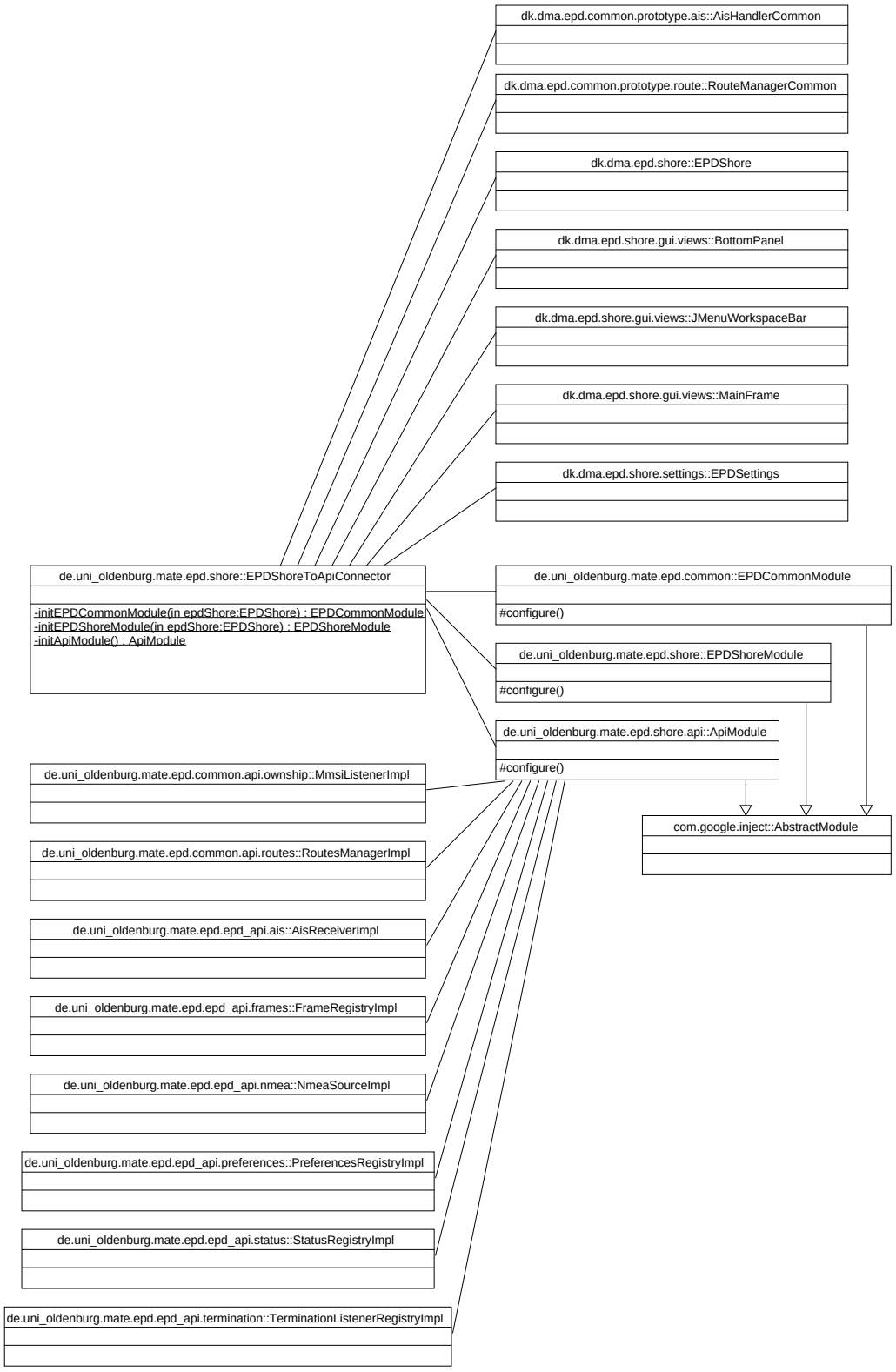


Abbildung 43: Klassendiagramm zur Verwendung des EPDApiShoreConnector

Innerhalb dieser Module werden die Abhängigkeiten der Instanzen definiert, sodass alle Injection-Klassen die benötigten EPD-Parameter und Klassen erhalten. Zusätzlich werden im Connector alle Plugins geladen, die in der Properties-Datei resources.epd-api-plugins.properties als aktiv markiert sind. Für die Aktivierung müssen die Submodule mit dem Pfad als true gesetzt werden, wie zum Beispiel: de.uni_oldenburg.mate.epd.virtual_handles. VirtualHandlesModule=true

7.4.4 Vessel-Connection

Das Submodul vessel-connection ist die Implementierung der Komponente „Verbindung“ aus dem Entwurf (siehe Abschnitt 6.4.3).

Mithilfe dieses Moduls werden über zwei Input-Adapter Daten empfangen und über zwei Output-Adapter versendet, nachdem Verbindungen zum RabbitMQ-Server aufgebaut wurden.

- Der RabbitMQInputAdapter empfängt Sensordaten vom Testbed. Ausserdem ändert diese Klasse den Status des RabbitMQStatus, um Anzuzeigen, ob die Konfiguration zur Übertragung von Sensordaten stimmt.
- Der RabbitMQInputAdapterRTZ empfängt Routendaten von der Pfadplanungskomponente. Ausserdem ändert diese Klasse den Status des RtzRabbitMqStatus, um Anzuzeigen, ob die Konfiguration zur Übertragung von Routendaten stimmt.
- Der RabbitMQOutputAdapter sendet manuelle Steuerdaten an das Regelungssystem.
- Der RabbitMQOutputAdapterRTZ sendet Routendaten an die Pfadplanungskomponente.

Für die Einrichtung einer solchen Verbindung werden die Preferences-Interfaces benötigt. Die PreferenceCategory wird dabei auf der Seite der Vessel-Connection implementiert und über die Schnittstelle PreferencesRegistry registriert. Durch diese Registrierung erscheint ein weiterer Tab in dem Preferences-Fenster der EPD- Shore. Innerhalb des Tabs können die Daten für Benutzername, Passwort und Host zur Anmeldung am RabbitMQ-Server eingegeben werden. Ein Screenshot mit beispielhaften Anmeldedaten der Projektgruppe (ohne Passwort) ist in Abbildung 44 zu sehen. Außerdem besteht mit dem Button „Load MATE Default“ die Möglichkeit, vordefinierte Einstellungen zu laden. Mit diesen wird eine Verbindung zum RabbitMQ-Server der Projektgruppe hergestellt. Die

Konfigurationsdaten für die einzelnen Exchanges und Queues sind in den Constants fest definiert (siehe Anhang A.8). Die Verbindung kann im Tab auch getestet werden.

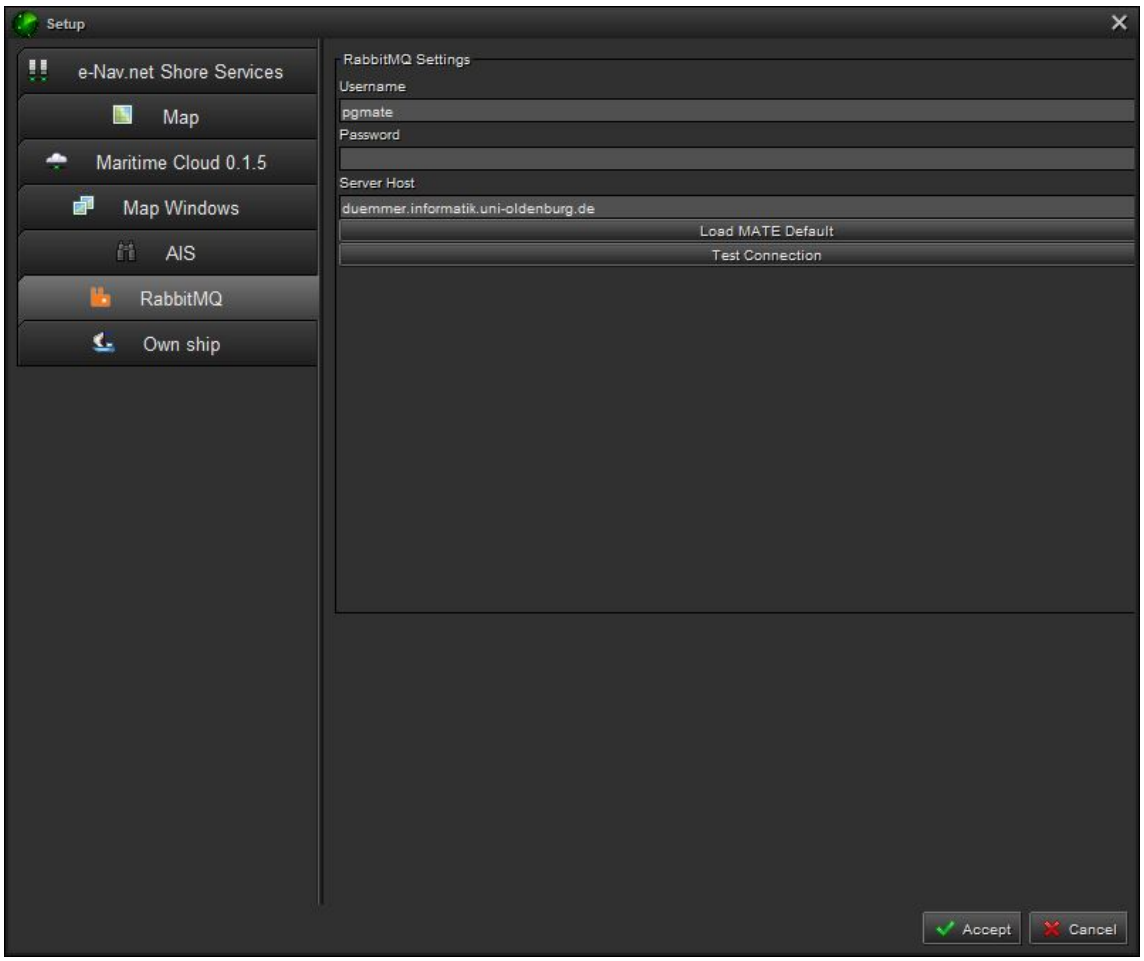


Abbildung 44: Screenshot des Preferences-Tab zur Eingabe von Daten zur Anmeldung am RabbitMQ-Server

Ein grafisches Feedback zur Verbindung mit dem RabbitMQ-Server erhält der Nutzer außerdem über mehrere Bottompanels unten rechts, die mit Hilfe der StatusRegistry registriert und verändert werden können (siehe Abbildung 45). Die Status RabbitMQ RTZ, RoutePilot (siehe späteren Abschnitt) und RabbitMQ Zuse wurden von der Projektgruppe implementiert. Folgende Farben können angenommen werden:

– Grau: Erster Start der EPD, es wurden noch keine Anmeldedaten eingetragen und nicht versucht, eine Verbindung herzustellen.

– Gelb: Anmeldung am Host OK, Probleme mit den Einstellungsdaten bezüglich Exchange oder Queue.

– Grün: Anmeldung am Host und Verbindung zum Exchange oder zur Queue OK. Daten können ausgetauscht werden.

– Rot: Keine Anmeldung am Host möglich oder Fehler während des Datenaustauschs.

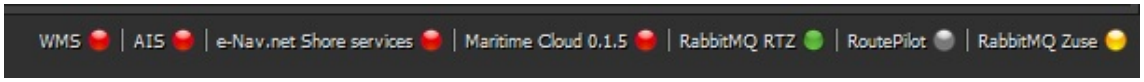


Abbildung 45: Screenshot der vorhandenen Status in verschiedenen Zuständen

7.4.5 Virtual Handles

Im Folgenden wird die Implementierung der Virtual Handles (kurz VH) beschrieben. Wie schon im Systementwurf beschrieben (siehe Abschnitt 6.4.3, können in den Virtual Handles sowohl Sensordaten angezeigt, als auch manuelle Steuerdaten entgegengenommen werden. Die Abbildung 46 zeigt einen Screenshot der Virtual Handles. Die Eingabemöglichkeit für die Richtung und die Motordrehzahl ist gelb umrahmt. Drum herum befindet sich die visuelle bzw. textuelle Darstellung von Sensordaten. Oben rechts in der Abbildung ist der RoutePilot zu sehen. Die Bedienung des Bow-Thruster wurde nicht implementiert.

Die Einbindung der index.html erfolgt in dem VirtualHandlesFrame via „javafx“. An dieser Stelle wird das JFXPanel und die WebEngine initialisiert. Gleichzeitig implementiert diese Klasse das Frame-Interface der EPD-Api, welches über die VirtualHandlesFrameFactory an die EPD übergeben wird. Diese Frames können über die Menübar im Reiter „Plugins“ geöffnet werden.

Um Änderungen in den VH anzuzeigen, wird der LocalWebSocketHandler verwendet, welcher eine WebSocketMessage in JSON parsed und an den WebSocket schickt. Der LocalWebSocketHandler besitzt außerdem eine Funktion zum Hinzufügen von Change-Listenern, sodass der VirtualHandlesChangeListener Eingaben vom Nutzer entgegennimmt. Ändert der Nutzer den Ruderwinkel, die Motordrehzahl, die Route oder den Status des Autopiloten entscheidet der Listener in der Methode onChange(WebSocketMessage message) über die nachfolgenden Aktionen.

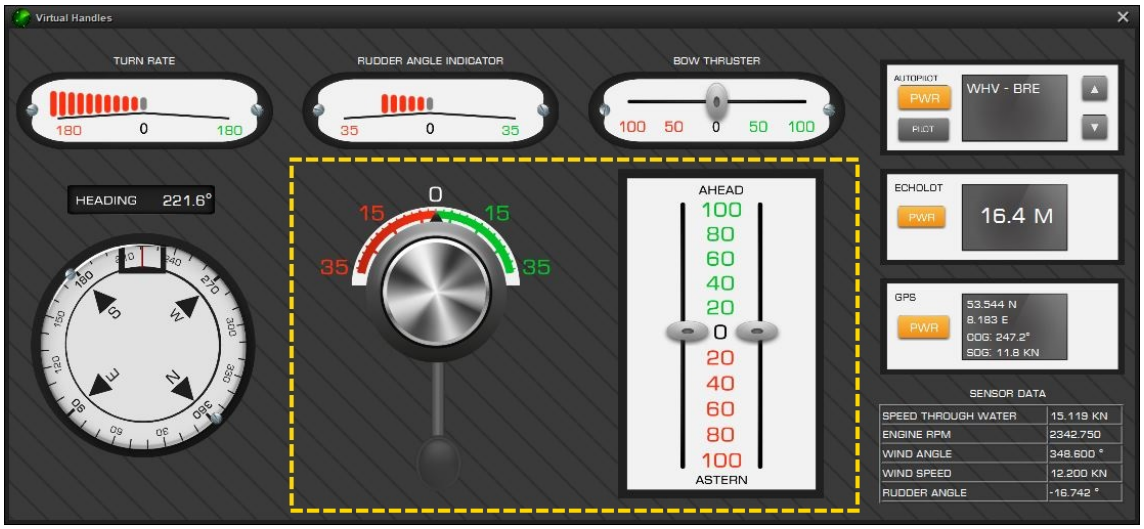


Abbildung 46: Screenshot der Virtual Handles (gelb umrahmt die Eingabemöglichkeit für Richtung und Motordrehzahl)

7.4.6 RoutePilot

Das Starten und Stoppen des teilautonomen Abfahrens einer Route wird im Modul routepilot implementiert. Bevor beschrieben wird, was passiert, wenn ein User den RoutePilot startet, wird aufgezeigt, wo die Routen herkommen.

Die Hauptklassen sind an dieser Stelle der RoutesManager und RoutesChangeListener, welche jeweils eine Liste von Routen zurückliefern. Die Route-Klasse beinhaltet wiederum WayPoints und Positionen. Die Implementierung des RoutesChangeListeners erfolgt im RoutesListener und schickt bei Änderungen an der Route eine WebSocketMessage an die Virtual Handles. Der RoutesManager ist in der EPD-Common implementiert und liefert über die Methode getRoutes() die Liste zurück. Außerdem werden bei Änderungen an der Route alle registrierten RoutesChangeListener mit der veränderten Liste informiert.

Bei den Virtual Handles beginnt die eigentliche Implementierung des RoutePilots. Die Auswahl und die Aktivierung einer Route erfolgt mit der selben GUI, in der die Sensordaten dargestellt werden und das Schiff manuell gesteuert werden kann (siehe Abbildung 46 oben rechts). Das Modul VirtualHandles bindet durch seine pom.xml das Modul routepilot ein.

Im Modul routepilot dreht sich alles um die Klasse RoutePilot. Hier werden die Hauptmethoden startRoutePilot(Route route) und stopRoutePilot() definiert, die vom VirtualHandlesChangeListenerImpl aufgerufen werden können. Beim Aufruf von start-

RoutePilot wird die EPD-Route mit Hilfe der RTZ-Library (siehe Abschnitt 7.6) ins RTZ-Format konvertiert und durch die Methode sendViaRabbitMq(String rtzRoute) Richtung Pfadplanungskomponente versendet. Dafür wird der RabbitMQOutputAdapterRTZ aus der VesselConnection verwendet.

Die Methode receiveDelivery(String message) erhält Nachrichten, die beim RabbitMQInputAdapterRTZ von der Pfadplanungskomponente ankommen und verarbeitet sie weiter. Die beiden Hauptaufgaben sind:

- Falls die Nachricht eine „active“ RTZ-Route enthält, stelle die beiden aktiven Wegpunkte auf der Karte dar.
- Falls die Nachricht eine „disabled“ RTZ-Route enthält, stoppe den RoutePilot und informiere die Virtual Handles.

Der Zustand des RoutePilot wird in einem Status unten rechts in der EPD angezeigt (siehe Abbildung 45)). Folgende Farben können angenommen werden:

- Grau: Start der EPD, der RoutePilot wurde in dieser Sitzung noch nicht bedient.
- Gelb: Eine Route wurde vom Benutzer aktiviert, warte auf Feedback der Pfadplanungskomponente.
- Grün: Eine Route wurde vom Benutzer aktiviert und aktive Wegpunkte wurden von der Pfadplanungskomponente empfangen.
- Rot: Die Route wurde vollständig abgefahren, oder der RoutePilot manuell beendet, oder ein Fehler ist aufgetreten.

7.4.7 Own Ship

Im Entwurf wurde vorgesehen das zu steuernde Schiff von anderen Schiffen auf der Karte hervorzuheben. Für die Implementierung wurde ein eigenes Package erstellt, damit Funktionen strikter getrennt werden konnten.

Es gibt einen eigenen Preferences-Tab, der durch die Klasse PreferenceCategoryOwnShip implementiert wurde. Hier kann die MMSI (siehe Abschnitt 3.1.5) des zu steuernden Schiffes angegeben werden. Über den MmsiListener ist es möglich dem bereits existierenden AisHandlerCommon die MMSI zu übergeben. Der AisHandler sucht in allen ihm vorliegenden Ais-Daten nach der MMSI und markiert das entsprechende VesselTarget als isOwnShip. Von hier aus ist der Datenfluss zwischen den Klassen der Gleiche, wie in der Ursprungs-EPD (siehe obere Hälfte in Abbildung 47). Die Ursprungs-EPD

konnte bereits Schiffe darstellen. Der Datenfluss zwischen den Klassen bleibt nahezu unverändert. Allerdings waren noch etliche Anpassungen und Methodenergänzungen nötig, um das eigene Schiff farblich (rot) von anderen Schiffen abzuheben (Screenshot vom Ergebnis siehe Abbildung 48 und Abbildung 49). In Abbildung 47 sind ab der Klasse VesselGraphicComponentSelector abwärts die Attribute und Methoden zu erkennen, welche von der Projektgruppe ergänzt wurden.

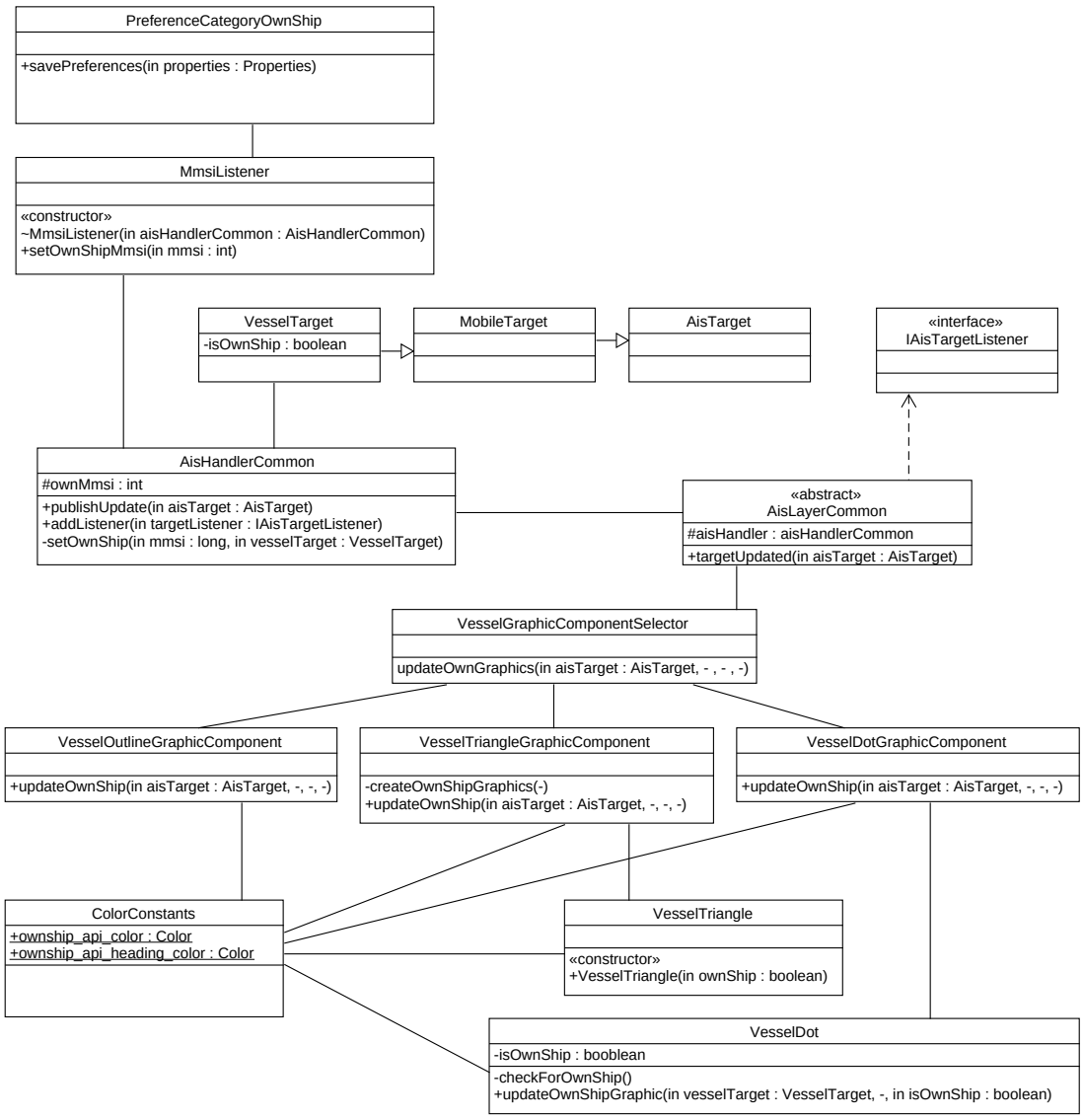


Abbildung 47: Klassendiagramm zur Hervorhebung des zu steuernden Schiffes gegenüber den Schiffen in der Umgebung

Nach der Implementierung dieser Schritte hat die Projektgruppe erkannt, dass die Positionsaktualisierung durch die bisher gelieferten AIS-Daten nur in relativ großen Zeitabständen stattfindet.

Sensordaten, und damit Positionsdaten, des zu steuernden Schiffes werden für die Umsetzung des ersten Meilensteins bereits in den VirtualHandles angezeigt. Es wurde beschlossen, die Position auf der Karte auch durch diese Sensordaten zu aktualisieren, da diese deutlich häufiger vorliegen. Insbesondere die GPS-Position, true heading, course over ground und speed over ground werden durch die Klasse OwnShipStateImpl in Ais-Nachrichten umgewandelt und dem schon existierenden AisReceiver übergeben. Durch die Ursprungs-EPD war bereits implementiert, dass der AisReceiver bei eingehender Ais-Nachricht den AisHandler benachrichtigt, damit die Änderung auf der Karte dargestellt wird.

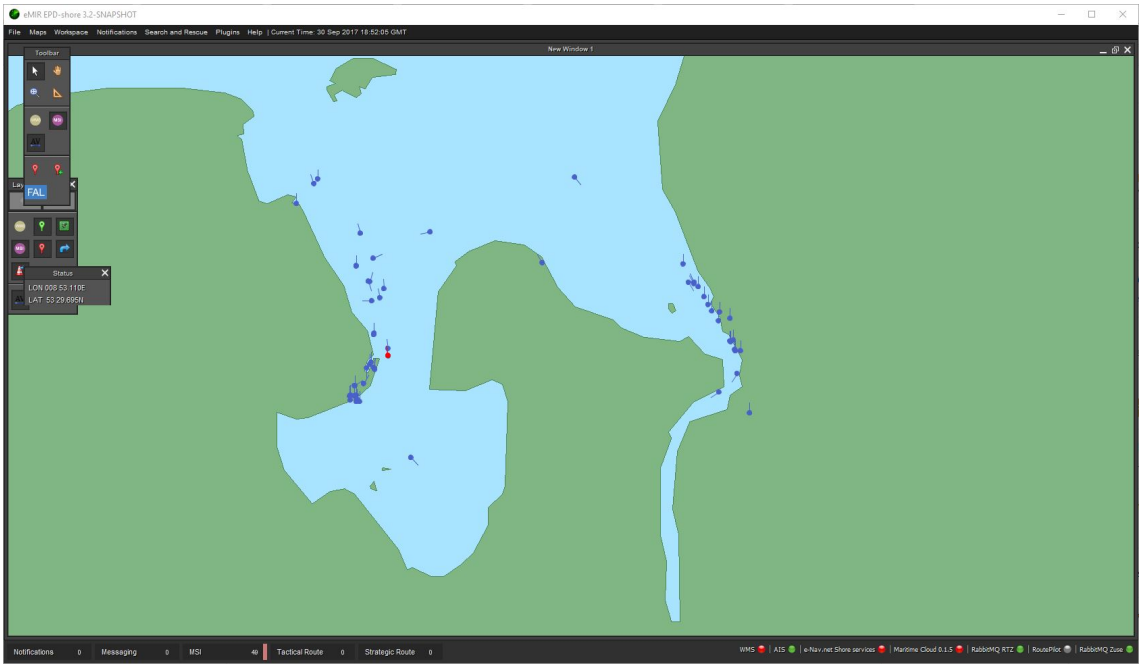


Abbildung 48: Screenshot der EPD, insbesondere der Seekarte (herausgezoomt)

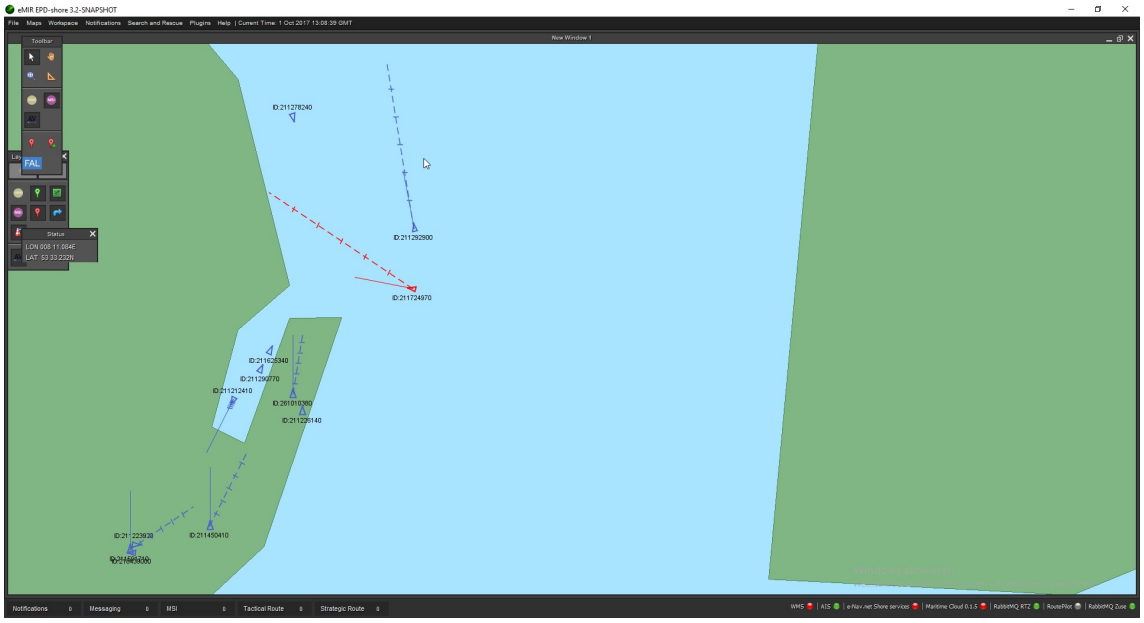


Abbildung 49: Screenshot der EPD, insbesondere der Seekarte (hineingezoomt)

7.5 Pfadplanung

Die Pfadplanungskomponente ist in der Programmiersprache Java implementiert und verwendet das Build-Management-Tool Maven. Der Aufbau entspricht der beschriebenen Architektur im Rahmen des Systementwurfs (siehe Abschnitt 6.5.3). Bei der Umsetzung der Schnittstellen wurde die RTZ-Programmbibliothek dazu verwendet, um eingehende Routen zu verarbeiten. Die Programmbibliothek ist als Maven-Abhängigkeit in das Projekt eingebunden (siehe Abschnitt 7.6).

Die Pfadplanungskomponente wird über die Kommandozeile gestartet. Beim Starten der Komponente muss der Pfad zu einer Konfigurationsdatei übergeben werden. Der Inhalt der Konfigurationsdatei entspricht den Vorgaben aus dem Systementwurf. Weitere Informationen zum Starten der Pfadplanungskomponente befinden sich im Anhang A.1.6.

```
1 input_port=4242
2 output_url=http://localhost:4243/waypoints
3 output_period=1000
4 minimal_turn_radius=0.01
```

Auflistung 5: Konfigurationsdatei der Pfadplanungskomponente

In Auflistung 5 ist ein beispielhafter Inhalt für die Konfigurationsdatei aufgeführt. Jede Einstellung steht in einer eigenen Zeile. Der Name und der Wert jeder Einstellung sind durch ein Gleichheitszeichen voneinander getrennt.

Über die Einstellung input_port wird der Port angegeben, über den die Pfadplanungskomponente Positions- und Routendaten empfangen soll. Die Einstellung output_url legt die URL fest, an die per HTTP die aktuellen Wegpunkte gesendet werden sollen. Dies geschieht periodisch. Über die Einstellung output_period wird diese Periode festgelegt. Die Zeit wird in Millisekunden angegeben. Die Einstellung minimal_turn_radius legt den Minimalradius fest, in dem ein Wegpunkt als erreicht gilt. Der Minimalradius wird in nautischen Meilen angegeben.

7.6 RTZ-Programmbibliothek

Alle Komponenten, welche die RTZ-Programmbibliothek benötigen, sind in der Programmiersprache Java implementiert und verwenden das Build-Management-Tool Maven. Aus diesem Grund ist die Programmbibliothek ebenfalls in der Programmiersprache Java implementiert und als Abhängigkeit für Maven-Projekte zur Verfügung gestellt worden. Die Klassen der Programmbibliothek wurden gemäß dem Systementwurf umgesetzt (siehe Abschnitt 6.6).

```
1 <dependency>
2     <groupId>de.uni_oldenburg.mate</groupId>
3     <artifactId>rtz</artifactId>
4     <version>1.2.0</version>
5 </dependency>
```

Auflistung 6: RTZ-Programmbibliothek als Abhängigkeit in einem Maven-Projekt

In Auflistung 6 ist die Abhängigkeit dargestellt, mit der die Programmbibliothek in ein Maven-Projekt eingebunden werden kann. Die Programmbibliothek wurde in dem Maven-Repository der Projektgruppe veröffentlicht. Die Verwendung der Programmbibliothek ist anhand von Beispielen im Anhang näher beschrieben (siehe Anhang A.1.7).

7.7 Regelungssystem

Das Regelungssystem aus Abschnitt 6.7 ist eine wesentliche Komponente des PG-MATE Produkts. Die Umsetzung trägt im Folgenden den Namen Controller, um den Begriff

Regelungssystem stärker von der zusätzlichen Nutzung von Kommunikationsmodulen, Übersetzungs- und Interpretationsmodulen abzugrenzen. Mit eingehenden Sensordaten und Wegpunktinformationen regelt der Controller die Trajektorie (siehe Absatz 6.7.1) im Autopiloten oder sorgt für die Weiterleitung von manuellen Steuerbefehlen. Damit nimmt der Controller insbesondere Einfluss auf jegliche Befehle, die an das Testbed und damit außerhalb des PG-MATE Produkts gesendet werden.

Bei der Umsetzung des Entwurfs zum Umwandeln und Interpretieren von Nachrichten traten einige Probleme auf. Im Folgenden wird daher eine multiple Lösung des Entwurfs von $\mathrm{NMEA2000Parser}$ und $\mathrm{NMEA2000Interpreter}$ beschrieben. Die finale Lösung dieser Module sowie die der anderen im Abschnitt 6.7 beschriebenen und erforderlichen Komponenten wird nachfolgend beschrieben – für nicht umgesetzte Module erfolgt eine Begründung. Zur Beschreibung der endgültigen Lösung des Controllers wird zusätzlich ein typischer Programmablauf mit Wechsel zwischen manuellem Modus und Autopiloten als Sequenzdiagramm dargestellt.

7.7.1 Erste Implementierung mit C++

Eine erste Umsetzung des Controllers wurde über die Programmiersprache C++ vorgenommen. Die Implementierungen hinsichtlich $\mathrm{NMEA2000Parser}$ und $\mathrm{NMEA2000Interpreter}$ waren sehr weit fortgeschritten, als klar wurde, dass mit diesen Implementierungen keine Echtzeitfähigkeit erreicht werden kann – eine zusätzliche Spezifikation des Controllers. Der Grund dafür war eine Vielzahl an dynamischen Allokationen mit dem new Operator oder dynamischen Umwandeln (englisch: cast) von Bytes in Datentypen oder anderweitige Datenstrukturen. In einem Echtzeit-Regelungssystem wie dem Controller zerstören dynamische Operationen die Berechenbarkeit der Anwendung und damit die Garantie der Echtzeit (vgl. [Iye03]).

Eine Lösung des Problems liegt in der Nutzung eines Puffer-Pools zum Umwandeln oder Allozieren. Dieser Pool muss bei der Initialisierung der Anwendung statisch in hinreichender Größe alloziert werden. Zur Laufzeit wird sich nur noch aus diesem Pool bedient, wenn zusätzlicher Speicher benötigt wird (vgl. [Iye03]). Eine Umsetzung dieser Aufgabe wurde im Vergleich zu der nachfolgend vorgestellten Lösung allerdings als zu aufwändig und langfristig schwerer wartbar angesehen. Die Abbildung 50 zeigt den letzten Stand der alten Controller Implementierung als Komponentendiagramm.

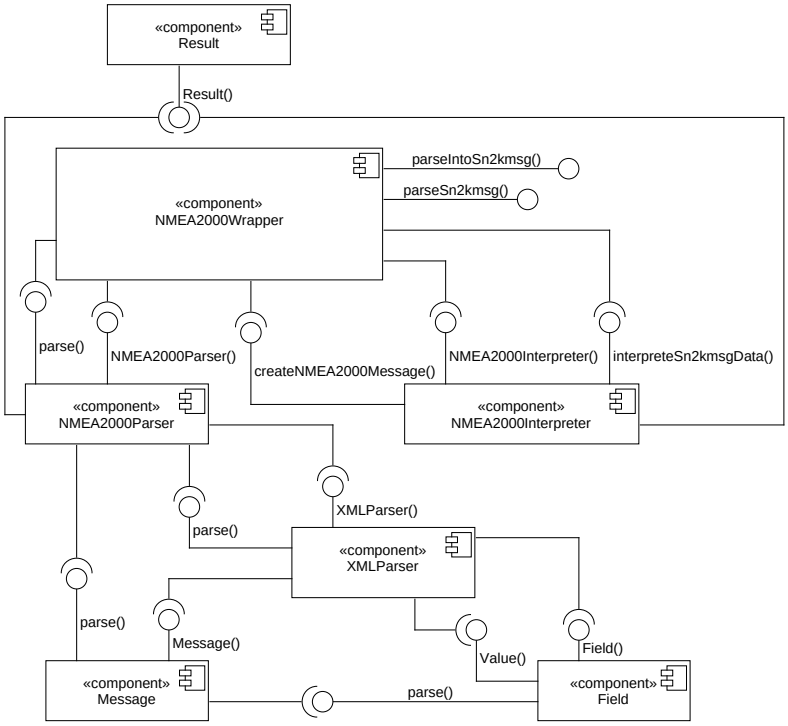


Abbildung 50: Letzter Stand der C++ Implementierungen des Controllers mit NMEA2000Parser und NMEA2000Interpreter

Der NMEA2000 Standard wurde in ein XML Dokument eingepflegt und wiederum über den $\mathrm{XMLParser}$ zur Laufzeit eingelesen. Dieser nutzt dafür zunächst die $\mathrm{Message}$ und Field Komponenten, die die Struktur der XML Datei widerspiegeln. Die eingelesenen $\mathrm{Messages}$ enthalten alle nötigen Informationen, die zu einer PGN vorliegen. Die Informationen werden zum Umwandeln von erhaltenen NMEA2000 Byte Nachrichten vom $\mathrm{NMEA2000Parser}$ verwendet. Das Ergebnis des Umwandelns wird in der Komponente Result hinterlegt, die wiederum von dem $\mathrm{NMEA2000Interpreter}$ zur weiteren Verarbeitung genutzt wird. Das große Problem dynamischer Operationen lag dann in den $\mathrm{parse}()$ Schnittstellen, die jeweils zur Verfügung gestellt wurden. Dort wurden zahlreiche dynamische Allokationen zum Beispiel durch das Einlesen von der XML Standardbeschreibung und Umwandlungen der Bytes in Datentypen vorgenommen. Das führte letztlich zum Widerspruch der Echtzeitfähigkeit.

7.7.2 Matlab-Simulink-Entwicklung

Die große Aufgabe des Controllers liegt mit dem Regelungssystem in mathematischen Berechnungen für die Trajektorie. Das Softwarehaus The MathWorks, Inc. spezialisiert sich vor allem auf mathematische Produkte und erleichtert damit die Implementierung der mathematischen Anteile des Controllers (vgl. [TMa]). Simulink ist ein Produkt von The MathWorks, Inc., dessen Stärken vor allem in einer modellbasierten Erstellung von Softwaresystemen und in der Simulation mit Blockdiagrammen liegen. Simulink bietet dabei eine Bibliothek aus Blöcken, die bereits implementierte und verifizierte Funktionalität bieten – vergleichbar mit der Java Klassenbibliothek. Der Nutzer kann diese Blöcke in eigenen Modellen einbinden und so das Softwaresystem mittels Blockdiagrammen modellbasiert entwickeln. Die Modelle lassen sich ebenfalls über eine Block-Bibliothek in andere Modelle einbinden (vgl. [TMc]).

Auch bietet Simulink mit einem zusätzlichen Paket die Möglichkeit der Echtzeitfähigkeit. Die integrierbare Echtzeitfähigkeit, die breite Anzahl an mathematischen Berechnungen sowie die modellbasierte Entwicklung mit Blockdiagrammen waren die Gründe für die finale Lösung des Controllers mit Simulink.

Realtime Fähigkeit

Im Abschnitt 6.7 wurde bereits die Echtzeitfähigkeit als zusätzliche Spezifikation begründet. Mit Simulink Real-Time lassen sich unter Nutzung spezieller Simulink-Blöcke echtzeitfähige Programme entwickeln. Die einzelnen Blöcke werden nachfolgend im jeweils zugehörigen Unterabschnitt erklärt. Die $\mathrm{Sample\ Time}$ in Simulink kann für einzelne Blöcke oder ganze Modelle eingestellt werden. Dies ermöglicht es in einem diskreten Zeitintervall $x \in \mathbb{R}$ Ausgabewerte zu produzieren (vgl. [TMe]). Die Garantie für die Produktion dieser Ausgabewerte innerhalb der $\mathrm{Sample\ Time}$ übernimmt das Simulink Real-Time Paket (vgl. [TMd]). Für die finale Lösung wurde die $\mathrm{Sample\ Time}$ auf $100\,ms$ gesetzt. Dadurch wird zyklisch alle $100\,ms$ eine Nachricht von Controller geliefert.

Internal Communication Handler

Der interne Kommunikationsvermittler aus Abschnitt 6.7 sollte die Kommunikation unter den einzelnen Subkomponenten des Controllers steuern und überwachen. Dadurch sollte gegensätzliche Ausführung vermieden und eine vorgegebene Reihenfolge, in der die Subkomponenten arbeiten sollten, geregelt werden. Mit den Signalen, Blockdiagrammen

und Modellen, mit denen Simulink funktioniert, regelt Simulink die Kommunikation der Komponenten, auch unter Nutzung der Sample Time, selbst. Daher wird diese zentrale Subkomponente bereits durch die Nutzung von Simulink umgesetzt.

7.7.3 Communication Modul

Der Controller benötigt zum Abfahren der Trajektorie stets aktuelle Informationen zur Lage des Schiffes. Dabei handelt es sich um Sensordaten, Wegpunktinformationen und um manuelle Steuerbefehle. Die Herkunft dieser Informationen wird jeweils im nächsten Unterabschnitt aufgelistet und beschrieben. Danach erfolgt eine genauere Beschreibung über Format und Verbindungstyp zur Annahme der Daten. Das Senden von Steuerbefehlen über das Kommunikationsmodul wird dazu ebenfalls beschrieben. Das Communication Modul als solches wurde direkt in die Controller-Komponente integriert, das heißt, es befindet sich in keinem gesonderten Simulink Modell, sondern wird über die Nutzung von UDP Send und UDP Receive aus dem Real-Time Paket abgebildet.

Verbindungsherstellung

Der Controller nimmt Informationen über eine UDP Schnittstelle entgegen und sendet verarbeitete Informationen ebenfalls wieder über eine UDP Schnittstelle aus. Dabei kann jeweils der Port für das Senden sowie Empfangen von Informationen angegeben werden. Für das Senden ist zudem die Eingabe einer IP-Adresse (Internet Protokoll Version 4 (IPv4)) möglich. Die Eingabe der IP-Adresse muss händisch über die Simulink-Blöcke erfolgen. Die Echtzeitfähigkeit der Annahme und des Sendens wird durch die Nutzung der UDP Send und UDP Receive Blöcke aus dem Simulink Real-Time Paket garantiert (vgl. [TMd]).

Ein- und Ausgabedaten für den Controller

Die über die UDP Verbindung entgegengenommenen Informationen müssen dem NMEA2000 Standard entsprechen, da diese entsprechend des nächsten Unterabschnittes nur so analysiert und interpretiert werden können. Für das Empfangen der Ausgabedaten des Controllers gilt entsprechend das gleiche. In Tabelle 29 sind die Eingabe- und Ausgabedaten mit Verbindungsrichtung, Datentyp und der das Datum enthaltenen PGN angegeben.

Datum	Verbindungsrichtung	Datentyp	PGN
Windstärke	Empfangen	uint16	130306
Windrichtung	Empfangen	uint16	130306
Wind-Referenz	Empfangen	uint8	130306
Geographische Breite	Empfangen	int32	129025
Geographische Länge	Empfangen	int32	129025
Geschwindigkeit über Wasser	Empfangen	uint16	128259
Geschwindigkeit über Grund	Empfangen	uint16	129026
Drehzahl	Empfangen	uint16	127488
Steuerkurs	Empfangen	uint16	127250
Kurswinkel (Course Angle)	Empfangen	uint16	129026
Route Status	Empfangen	uint8	130066
Wegpunkt ID 1	Empfangen	uint16	130067
Wegpunkt 1 Breite	Empfangen	int32	130067
Wegpunkt 1 Länge	Empfangen	int32	130067
Wegpunkt ID 2	Empfangen	uint16	130067
Wegpunkt 2 Breite	Empfangen	int32	130067
Wegpunkt 2 Länge	Empfangen	int32	130067
Ruderposition	Empfangen	int16	127245
Zieldrehzahl	Senden	int32	000001
Zielruderwinkel	Senden	int16	127245

Tabelle 29: Definition der Ein- und Ausgabedaten des Controllers mit PGN und Datentyp aus dem NMEA2000 Standard

Die Zieldrehzahl mit PGN 000001 wurde der Standarddefintion von NMEA2000 selbstständig hinzugefügt. Die PGN 000001 ist zur Wahrung des Standards an die Geschwindigkeit über Wasser (PGN 128259) und die Geschwindigkeit über Grund angelehnt (PGN 129026). Beide Daten weisen mit uint16 nur positive Geschwindigkeiten auf. Um das Rückwärtsfahren mit negativen Werten zu ermöglichen, wurde der Datentyp von uint16 auf int32 erweitert, um den negativen Wertebereich in gleicher Menge abzudecken. Da hier RPM als Befehl gesendet werden sollen, handelt es sich dabei um Drehzahlbereiche. Nach Angaben des NMEA2000 Komitees ist es möglich standardkonforme PGNs zu der Standardnutzung hinzuzufügen. Dies wurde entsprechend umgesetzt (vgl. [Com16]).

Die PGN befindet sich im Anhang A.6 als XML-Beschreibung. Die Beschreibung der anderen PGNs ist vom Grundaufbau ähnlich, kann aber abweichende Elemente besitzen.

Eine Besonderheit ergibt sich noch hinsichtlich der Größe der empfangenen Daten. Diese dürfen nicht größer als 1800 Bytes sein. Zudem werden die gesendeten Daten entsprechend der Sample Time aus dem vorherigen Abschnitt in Echtzeit zyklisch gesendet. Der Controller ist zudem so entwickelt, dass er zyklisch Daten empfangen kann und auch erwartet. Der Grund dafür liegt in der Nutzung von UDP, bei dem keine Garantie über die Zustellung von Paketen erfolgt. Um den Verlust von UDP auszugleichen, wird zyklisch gesendet und empfangen.

7.7.4 Parser und Interpreter

In Abschnitt 6.7 war sowohl ein $\mathrm{NMEA2000Parser}$ Modul, als auch ein $\mathrm{NMEA2000Interpreter}$ Modul vorgesehen. Der $\mathrm{NMEA2000Parser}$ übernimmt das Umwandeln der Bytes aus den empfangenen UDP Paketen bzw. das Einpacken von Befehlen in NMEA2000 Nachrichten. Beide Module können nicht getrennt voneinander arbeiten, sondern müssen wechselseitig ihre Aufgabe erfüllen. Einer der Gründe dafür ist, dass bei jeder empfangenen Nachricht zunächst der Header der NMEA2000 Nachricht analysiert werden muss, um die darin enthaltene PGN interpretieren zu können. Nach der PGN richtet sich dann die weitere Verarbeitung der NMEA2000 Nachricht, das heißt, die Interpretation der PGN entscheidet den weiteren Programmablauf. Durch die wechselseitige Trennung wurden die beiden genannten Module zu einem neuem $\mathrm{NMEA2000ParserInterpreter}$-Modul zusammengefasst. Die Abbildung 51 verdeutlicht noch einmal als Sequenzdiagramm den grob dargestellten Programmablauf für das Parsen und Interpretieren einer NMEA2000 Nachricht mit PGN 000001.

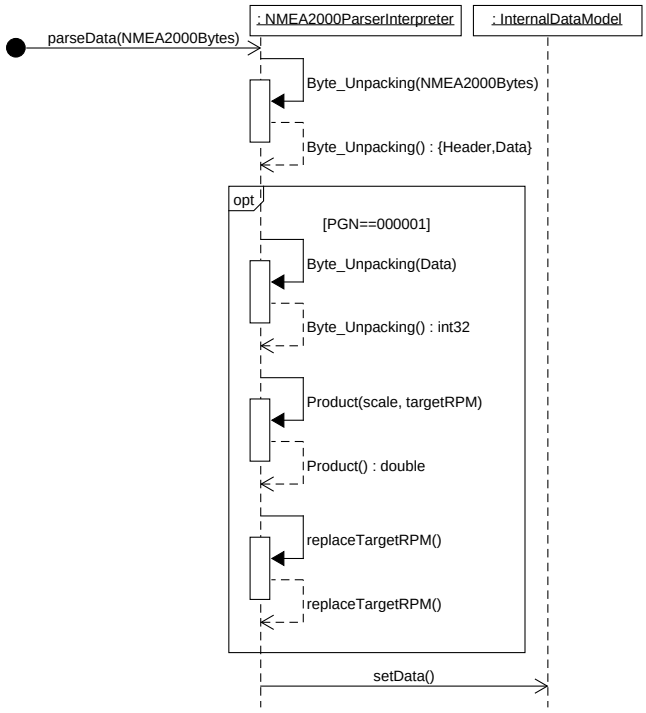


Abbildung 51: Programmablauf des NMEA2000ParserInterpreter-Moduls beim Parsen und Interpretieren einer NMEA2000 Nachricht mit PGN 000001

Bei Simulink wird immer der komplette NMEA2000ParserInterpreter als Block aufgerufen, aber nicht neu initialisiert. Damit ist die Nutzung des NMEA2000ParserInterpreters ähnlich eines Methodenaufrufes und wird durch die Methode parseData dargestellt. Diese Methode stößt den Umwandlungs- und Interpretationsprozess durch Übergabe der zu verarbeitenden Bytes an. Der erste Aufruf von Byte_Unpacking trennt den Header Teil vom Datenteil der NMEA2000 Nachricht. Entsprechend wird beides von der Funktion zurückgegeben. Aus dem Header Teil wird die PGN interpretiert und der Datenteil nochmals analysiert – genau dann, wenn die PGN 000001 ist. Der so gewonnene RPM Wert als Zieldrehzahl im int32 Datenformat wird mit dem im NMEA2000 Standard angegebenen Scale (siehe [Com16]) multipliziert. Der dann aktuelle targetRPM Wert ersetzt zunächst den letzten Zieldrehzahlwert und wird anschließend an das InternalDataModel weitergegeben.

Wird der opt Block nicht betreten, wird die zuletzt bekannte Zieldrehzahl oder ein initialer Wert gesendet. Das bedeutet, dass im NMEA2000ParserInterpreter alle Daten vorgehalten und beim Empfangen aktuellerer Daten ersetzt werden. Der Grund für das Vorhalten und Ersetzen von alten Werten ist das bereits beschriebene zyklische Senden

und Empfangen. Das If-Action-Subsystem aus Simulink hält stets den zuletzt bekannten oder initialen Wert vor und ändert diesen nur, wenn eine entsprechende NMEA2000 Nachricht analysiert und interpretiert wurde – dargestellt durch den opt Block. Würde der Wert immer ersetzt werden, würde bedingt durch das zyklische Empfangen jeder andere Wert automatisch wieder auf 0 gesetzt werden und damit zu falschen Sensordaten im internen Datenmodell führen.

An den Ausgängen der If-Action-Subsystem liegt stets der zuletzt bekannte Wert an, das heißt, dem internen Datenmodell wird immer ein Wert übergeben – dargestellt durch setData. Das interne Datenmodell ähnelt der Tabelle 29 und ist mit Initialwerten in der Tabelle 30 zu sehen. Die geänderten Datentypen ergeben sich aus der Multiplikation mit dem Scale.

Datum	**Initialwert**	**Datentyp**
Windstärke	0	double
Windrichtung	0	double
Wind-Referenz	0	double
Geographische Breite	0	double
Geographische Länge	0	double
Geschwindigkeit über Wasser	0	double
Geschwindigkeit über Grund	0	double
Drehzahl	0	double
Steuerkurs	0	double
Kurswinkel (Course Angle)	0	double
Route Status	1	uint8
Wegpunkt ID 1	0	double
Wegpunkt 1 Breite	0	double
Wegpunkt 1 Länge	0	double
Wegpunkt ID 2	0	double
Wegpunkt 2 Breite	0	double
Wegpunkt 2 Länge	0	double
Ruderposition	0	double
Zieldrehzahl	0	double
Zielruderwinkel	0	double

Tabelle 30: Internes Datenmodel des Controllers mit Initialwerten je Datum

Vom Controller werden Zieldrehzahl und Zielruderwinkel ausgesendet. Diese Daten werden in ihre entsprechenden NMEA2000 Nachrichten analysiert. Dies wird ebenfalls vom NMEA2000ParserInterpreter-Modul übernommen. Dabei wird beispielsweise die Zieldrehzahl zunächst mit dem inversen des im NMEA2000 Standards angegebenen Scales multipliziert und dann mit entsprechender PGN in eine NMEA2000 Nachricht übersetzt. Das Kommunikationsmodul übernimmt mit dem UDP Send die Ausgabe. Die genannten Operationen sind in Abbildung 52 im nächsten Abschnitt integriert.

7.7.5 Manueller Modus und Autopilot

Der Controller unterstützt zwei Modi: Manueller Modus und Autopilot. Der Modus wird anhand des Route Status gewechselt. Der Route Status wird zyklisch empfangen. Nach der Tabelle 30 ist der Initialwert vom Route Status 1, was bei der entsprechenden NMEA2000 PGN 130066 als inactive (deutsch: inaktiv) angegeben ist. Das bedeutet, dass der Controller initial im manuellem Modus startet und erst beim Empfangen eines active (deutsch: aktiven) Route Status mit 0 in den Autopiloten wechselt. Der Controller bleibt so lange in seinem derzeitigen Zustand, bis zum ersten Mal ein anderer Route Status gesendet wird.

Im manuellem Modus werden die Zieldrehzahl sowie der Zielruderwinkel jeweils nur zwischengespeichert und weitergeleitet. Das bedeutet, dass falls die Route inaktiv ist, lediglich die Zielnachrichten wieder mit dem Inverse-Scale multipliziert und in eine NMEA2000 Nachricht übersetzt werden. Ist die Route aktiv wird der PID Controller aus dem nächsten Abschnitt als Autopilot aufgerufen. Die Abbildung 52 verdeutlicht, wie der Wechsel zwischen Autopiloten und manuellem Modus implementiert wurde.

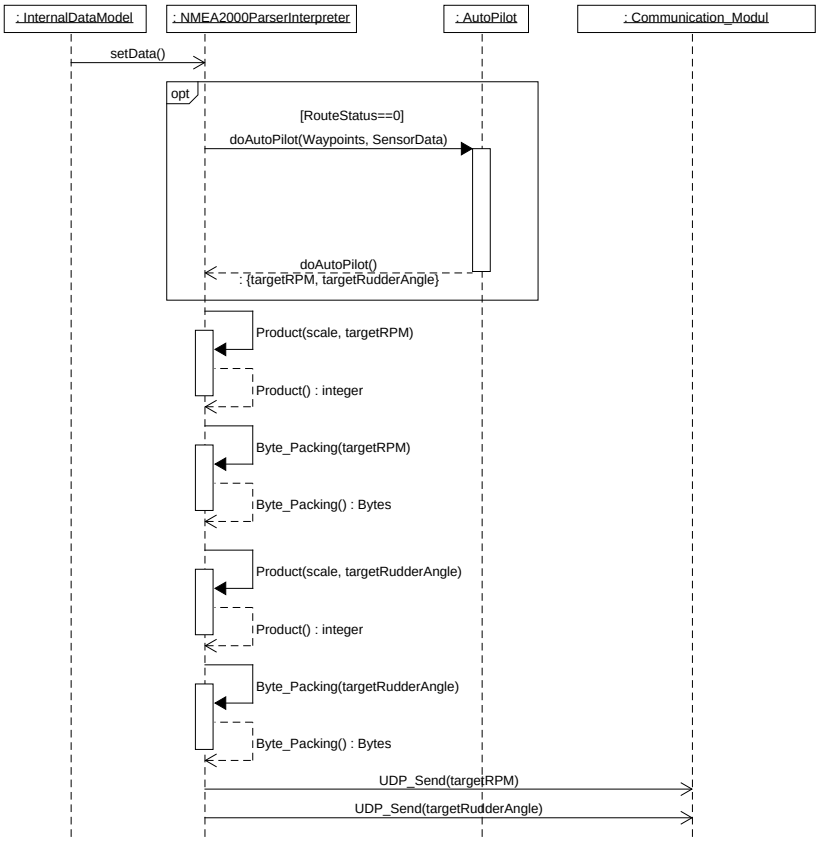


Abbildung 52: Wechsel zwischen manuellem Modus und Autopiloten unter Interpretation vom Route Status

In Simulink liegen stets alle Daten des internen Datenmodells an den Ausgängen an und werden entsprechend der Sample Time aktualisiert und neu bereitgestellt. Da die Interpretation direkt an das interne Datenmodell angeschlossen ist, können die Daten zu jeder Zeit direkt abgegriffen werden. Mit setData soll dieser Prozess in Simulink dargestellt werden. Der Aufruf von doAutoPilot beim Autopiloten stellt einen ähnlichen Fall dar. Dort liegen auch zu jeder Zeit benötigte Daten des Autopiloten an, allerdings werden diese nur verarbeitet, wenn der Autopilot über mehrere If-Action-Subsystem Verbindungen betreten werden darf. Das genaue Aufschlüsseln dieser Strukturen für den Autopiloten würde hier eher verwirren, daher kapselt doAutoPilot dieses Verhalten.

Ist wie beschrieben der Route Status 0, dann berechnet der Autopilot anhand von Wegpunkten und anderen Sensordaten die Zieldrehzahl (targetRPM) sowie den Zielruderwinkel (targetRudderAngle) getrennt voneinander und gibt diese Werte zurück. Die neuen Werte ersetzen dann temporär die im internen Datenmodell erhaltenen Zieldaten.

Temporär deshalb, weil der Autopilot niemals die interne Datenspeicherung beeinflusst. Sobald in den manuellen Modus zurückgeschaltet wird, wird der opt Block nicht mehr durchlaufen und entsprechend die im internen Datenmodell vorgehaltene Zieldrehzahl bzw. Zielruderwinkel erneut gesendet. Ist die Route aktiv, werden von der vorgelagerten Transformation von RTZ zu NMEA2000 immer auch die Zieldrehzahl und Zielruderwinkel als NMEA2000 Nachricht mit Wert 0 erstellt und mitgesendet (siehe Abschnitt 6.2.9). Der Grund dafür ist, dass es aufgrund von Schleifenrestriktionen mit $\mathrm{If\text{-}Action\text{-}Subsystem}$ in der Subsystem Semantik nicht möglich war die manuellen Steuerbefehle zurückzusetzen, ohne beim zyklischen Empfangen ungewollt die Zieldaten mit 0 zu überschreiben. Mehr zu diesem Problem findet sich unter [Rou13].

Alle Ergebnisse einer Operation werden in die beiden Variablen $\mathrm{targetRPM}$ und $\mathrm{targetRudderAngle}$ geschrieben, das heißt, am Ende wird dem Kommunikationsmodul eine Zieldrehzahl und ein Zielruderwinkel als NMEA2000 Nachricht in Bytes übergeben und über UDP ausgesandt. Bevor beide Nachrichten in die NMEA2000 Byte Nachricht eingepackt werden, werden diese noch mit ihrem Inverse-Scale aus der Standarddefinition multipliziert und wieder das Ergebnis in der Variable gespeichert. Der von der Operation $\mathrm{Product}$ zurückgeworfene Wert ist ein ganzzahliger Wert $z \in \mathbb{Z}$ und daher nicht auf den positiven Zahlenbereich begrenzt, wie zum Beispiel uint16.

7.7.6 Einbindung PID Controller

Ein PID Regler wird sehr häufig in der Regelungstechnik eingesetzt und besteht aus den drei Teilen: Dem Proportionalglied (P-Glied), dem Integrierglied (I-Glied) und dem Differenzierglied (D-Glied). Mit dem PID-Regler ist es unter anderem möglich die Trajektorie beizubehalten und Geschwindigkeiten nicht ruckartig, sondern sukzessive proportional bis zum gewünschten Wert anzupassen. Insbesondere wird es dadurch möglich frühzeitig einen angemessenen Winkel zur nächsten Strecke zwischen zwei Wegpunkten einzuschlagen, um damit nicht ruckartig bei Erreichen des Wegpunktes den Kurs ändern zu müssen (vgl. [Ver17], S. 151).

Die Implementierung einer solchen PID Regelung als Komponente im Controller wurde aus zeitlichen Gründen von Mohamed Abdelaal, einer der Betreuer dieser Projektgruppe, übernommen. Daher wird nachfolgend mit Abbildung 53 die Integration dieser Komponente mit einem sehr hohen Abstraktionsgrad beschrieben. Gleichzeitig stellt der Autopilot als PID Regelung zugleich die Erfüllung des Trajectory Module aus dem Systementwurf dar.

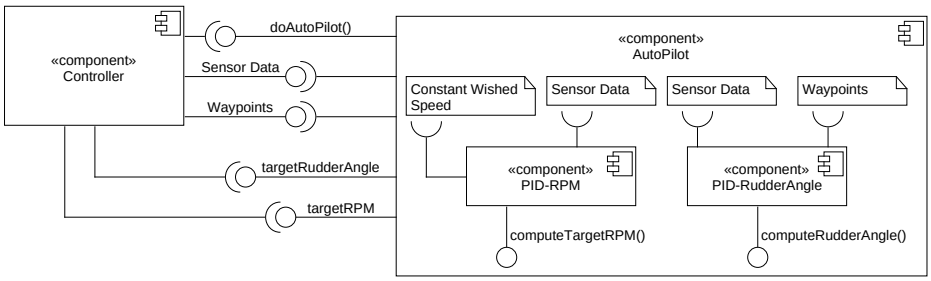


Abbildung 53: Integration der PID-Controller Implementierungen als Autopilot in die eigentliche Controller Komponente

Einen kurzen Einblick über die Nutzung des Autopiloten wurde bereits mit Abbildung 52 und dem Wechsel zwischen manuellem Modus und Autopiloten gegeben. Die Abbildung 53 zeigt nun noch zusätzlich die angesprochenen PID Regelungen mit der $\mathrm{PID\text{-}RPM}$ und dem $\mathrm{PID\text{-}RudderAngle}$ als zwei Subkomponenten des Autopiloten. Diese bieten jeweils eine Schnittstelle an, um die Zieldrehzahl bzw. den Zielruderwinkel mittels der oben angerissenen Glieder zu berechnen. Der Autopilot bietet nach Beendigung seiner Aufgabe die Zieldrehzahl mit $\mathrm{targetRPM}$ und den Zielruderwinkel mit $\mathrm{targetRudderAngle}$ als Schnittstelle an. Beide Subkomponenten benötigen für ihre Aufgabe neben den Sensordaten, zusätzlich noch Wegpunktinformationen oder eine gewünschte Geschwindigkeit. In der Simulink Implementierung werden diese Daten bereitgestellt, sofern der Autopilot aktiv ist.

Die gewünschte Geschwindigkeit stellt einen besonderen Aspekt dar. Aus zeitlichen Gründen war es nicht mehr möglich eine PGN zu entwerfen oder bestehende PGNs so zu verwenden, dass diese eine Zielgeschwindigkeit mitliefern können. Daher wird in der Implementierung der Wert 10 als gewünschte Geschwindigkeit in Knoten konstant an die $\mathrm{PID\text{-}RPM}$ Regelung übergeben. Auf gleiche Weise sind die PID Regler so entworfen worden, dass sie für beliebige Schiffe arbeiten können. Das bedeutet, dass zunächst von der $\mathrm{PID\text{-}RPM}$ Komponente ein Wert zwischen 0 und 255 sowie von der $\mathrm{PID\text{-}RudderAngle}$ Komponente ein Wert zwischen -10 und 10 zurückgegeben wird. Dieser wurde angepasst auf das Testbed mit der Zuse, für die Drehzahl Regelung mit 10 multipliziert und für die Ruderwinkel Regelung ebenfalls mit 10 multipliziert, um die gewünschten Wertebereiche des Testbeds einzuhalten.

7.7.7 Endianness

Die Endianness bezeichnet die Byte-Reihenfolge, in der die Bytes vom Speicher eingelesen bzw. im Speicher abgelegt oder allgemein interpretiert werden. Die beiden Hauptformen von Little-Endian und Big-Endian unterscheiden sich dabei stark. Bei Little-Endian wird das höchstwertige Byte immer ans Ende der Byte-Reihe geschrieben und bei Big-Endian an den Anfang. Die Endianness ist stark vom Prozessor der Hardware-Architektur abhängig (vgl. [Asc16], S. 72).

Um Probleme bei der Kommunikation mit anderen Komponenten im PG-MATE Produkt mit unterschiedlicher Endianness zu vermeiden, wurde der Controller daher in zwei Versionen entworfen. Bei der einfachen Version wird keine Endianness-Transformation vorgenommen, das heißt, der Ablauf ist wie in den vorherigen Abbildungen zum Controller dargestellt. Ein allgemeiner Programmablauf des Controllers ohne Endianness-Transformation ist in Abbildung 55 abgebildet. Arbeitet eine der anderen Komponenten des PG-MATE Produkts mit einer anderes Endiannes, wird beim Umwandeln vor jedem Byte Unpacking bzw. Byte Packing Block in Simulink ein Change endianness Block vorangestellt. Dadurch können die Bytes entsprechend aufgearbeitet und anschließend korrekt interpretiert bzw. korrekt vom Kommunikationsmodul ausgesandt werden.

7.7.8 Nicht umgesetzte Module

In Abschnitt 6.7 war ein Logging Modul vorgesehen. Aus zeitlichen Gründen wurde diese Komponente nicht mehr mit umgesetzt. Der Zeitverlust durch den Verwurf der ersten Lösung war zu groß, um rechtzeitig einen funktionierenden Logging-Mechanismus zu entwerfen, der auch noch bei einem Einsatz ohne persistente Datenspeicherung wie mit dem Simulink Real-Time funktioniert. Für weitere Informationen siehe [TMb].

Zudem wurde im Systementwurf ein Traffic Modul beschrieben, um Hindernisse auf dem Wasser oder andere Schiffe valide erkennen zu können. Diese Komponente wurde aus Zeitgründen ebenfalls nicht mehr umgesetzt, da dafür weitreichendere Sensordaten empfangen und verarbeitet hätten werden müssen.

7.7.9 Gesamtarchitektur

Bisher wurden alle Komponenten des Controllers weitestgehend einzeln betrachtet und beschrieben. Die Abbildung 54 verbindet die bisherigen Informationen aus den vorherigen Sequenz- und Komponentendiagrammen zu einer Gesamtarchitektur des Controllers.

Die Gesamtarchitektur stellt zugleich das implementierte Gegenstück zu dem in Abschnitt 6.7 entworfenen Komponentendiagramm dar und ermöglicht einen einfacheren Soll-Ist-Vergleich.

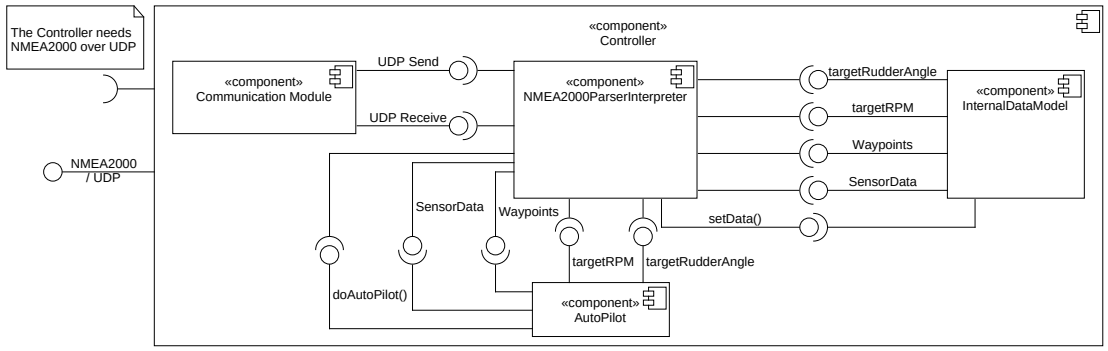


Abbildung 54: Komponentenbasierte Darstellung der gesamten Controller Ist-Architektur mit inneren und äußeren Schnittstellen sowie inneren Komponenten

Da die Aufgabe der internen Kommunikationsvermittlung von Simulink übernommen wird, zeigt obiges Komponentendiagramm nur die Kernkomponenten des Controllers. Die Schnittstellen des Controllers sind nach außen mit NMEA2000 über UDP gleichgeblieben. Auch das Kommunikationsmodul sowie der Autopilot als Trajectory Module sind ähnlich zum Entwurf. Das interne Datenmodell sowie der $\mathrm{NMEA2000ParserInterpreter}$ sind zwei neue Komponenten – beide aus einer Notwendigkeit heraus entstanden. Der Grund für die Integration von $\mathrm{NMEA2000Parser}$ und $\mathrm{NMEA2000Interpreter}$ in dem $\mathrm{NMEA2000ParserInterpreter}$ wurde bereits erklärt. Das interne Datenmodell wurde geschaffen, um die Auswahl an empfangenen Sensordaten, Wegpunktinformationen und manuellen Steuerbefehlen übersichtlich in einem Simulink Modell zu verwalten.

Die verwendeten bzw. zur Verfügung gestellten Schnittstellen dieser Komponenten sind detaillierter dargestellt. Mit der $\mathrm{setData}$ Schnittstelle vom $\mathrm{NMEA2000ParserInterpreter}$ werden neue Sensordaten, Wegpunktinformationen oder manuelle Steuerbefehle im internen Datenmodell registriert. Letzteres stellt zugleich die benötigen Informationen wieder zur Verfügung, wenn diese über $\mathrm{UDP\ Send}$ im manuellem Modus ausgeliefert werden sollen oder zu weiteren Verarbeitung mit $\mathrm{doAutoPilot}$ an den Autopiloten übergeben werden. Zentrale Komponente ist dabei der $\mathrm{NMEA2000ParserInterpreter}$, der die Einordnung der empfangenen Daten in das interne Datenmodell durch vorheriges Umwandeln ermöglicht, Daten aus diesem

wieder bereitstellt oder Steuerbefehle wieder in NMEA2000 einpackt und an das Kommunikationsmodul übergibt.

7.7.10 Allgemeiner Programmablauf

In Abbildung 55 wird ein typischer Programmablauf des Controllers nach dem Starten dargestellt. Dabei werden direkt nach dem Start Steuerbefehle vom Controller ausgesendet. Anschließend wird der Vorgang entsprechend der $\mathrm{Sample\ Time}$ in einer Endlosschleife wiederholt. Das bedeutet, dass der dort gezeigte Programmablauf solange zyklisch wiederholt wird, bis der Controller abgeschaltet wird.

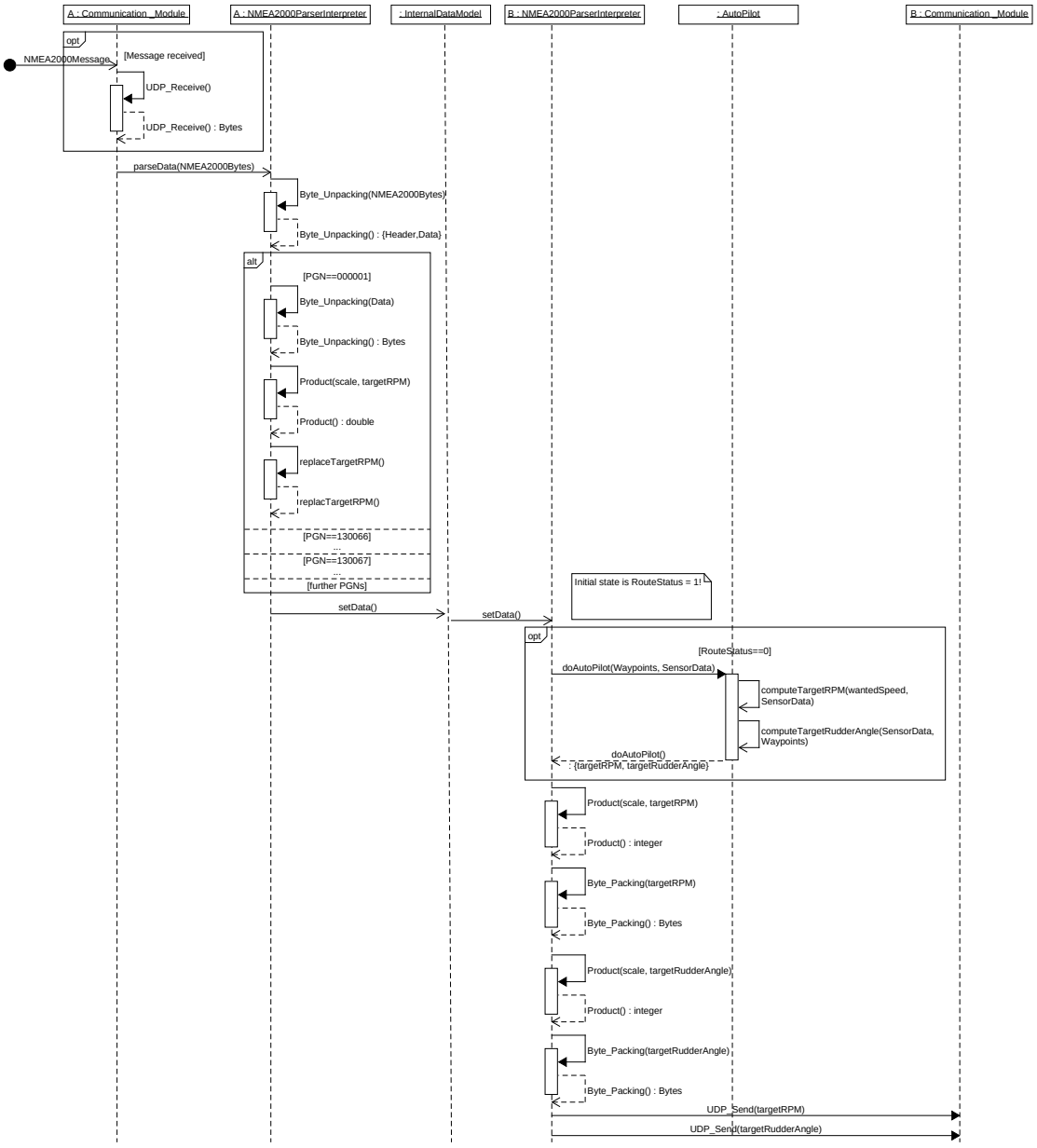


Abbildung 55: Standardmäßiger Ablauf des Controllers mit optionalem Wechsel zwischen Autopilot und manueller Steuerung

Eine weitere Information ergibt sich hinsichtlich der alt Blöcke für das Umwandeln des Datenanteils einer NMEA2000 Nachricht. Beispielhaft wird dort erneut das Umwandeln der PGN 000001 gezeigt. In den Alternativen für die anderen PGNs nach Tabelle 30 sind oft mehr erforderliche Daten innerhalb einer NMEA2000 Nachricht. Allerdings ist der Vorgang dabei mit dem Auspacken, Multiplizieren und Ersetzen der Daten immer gleich. Um das Sequenzdiagramm übersichtlich zu gestalten, wurden die anderen Alter-

nativen mit ... abgekürzt. Besonders ist darauf zu achten, dass zwischen den einzelnen Lebenslinien stets nur asynchrone Aufrufe liegen, da in der Simulink Controller Implementierung jede Komponente eigenständig und damit unabhängig von der Arbeit einer anderen operiert. Einzige Ausnahme bildet der Autopilot, der allerdings nur betreten wird, falls der $\mathrm{Route\ Status}$ 0 ist.

Der Rest des Programmablaufs ist entsprechend Abbildung 52 gleich und wird daher nicht nochmal erklärt. Ergänzt wurde hier noch der interne und asynchrone Aufruf von $\mathrm{computeTargetRPM}$ und $\mathrm{computeTargetRudderAngle}$ im Autopiloten. Diese Subkomponenten des Autopiloten wurden bereits mit Abbildung 53 erläutert.

7.8 Notfallsteuerung

Die Notfallsteuerung wurde als Java-Programm umgesetzt und kann über die Kommandozeile gestartet werden. Im Folgenden werden die wichtigsten Bestandteile der Notfallsteuerung beschrieben. Der Aufbau entspricht dem Systementwurf (siehe Abschnitt 6.8).

7.8.1 Eingabe von Steuerbefehlen

Die Eingaben werden, wie im Systementwurf beschrieben, über GLFW von einem Gamepad empfangen. In der Abbildung 56 sind die Komponenten dargestellt, die für die Verarbeitung der eingebenden Steuerbefehle zuständig sind.

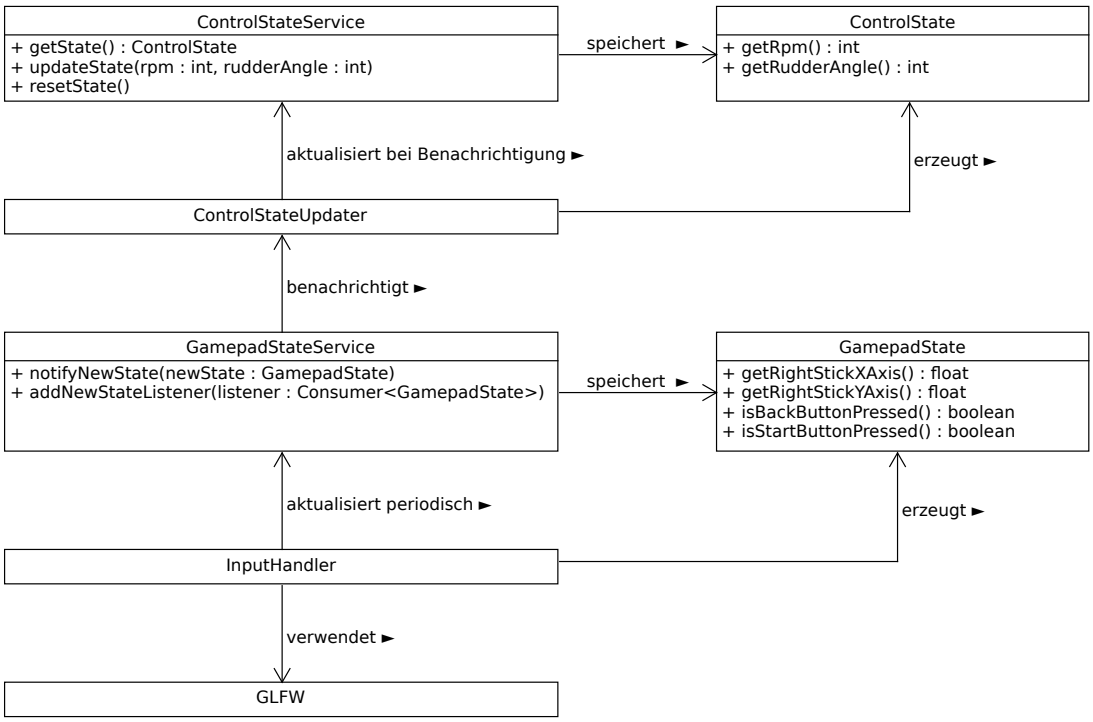


Abbildung 56: Eingabe von Steuerbefehlen in der Notfallsteuerung

Der InputHandler verwendet GLFW, um periodisch den Zustand des Gamepads abzurufen. Aus diesen Informationen wird ein GamepadState erzeugt und an den GamepadStateService übergeben. Dafür wird die Methode notifyNewState(newState : GamepadState) verwendet. Der GamepadState enthält den aktuellen Zustand des Steuerknüppels und der Start- und Stoppknöpfe.

Der GamepadStateService benachrichtigt den ControlStateUpdater, sobald ein neuer GamepadState verfügbar ist. Dieser berechnet auf Basis des GamepadStates die gewünschte Motordrehzahl und den gewünschten Ruderwinkel. Beide Informationen werden an den ControlStateService übergeben. Dafür wird die Methode updateState(rpm : int, rudderAngle : int) verwendet.

Der ControlStateService speichert diese Informationen im ControlState. Über die Methode getState() : ControlState können andere Komponenten die gewünschte Motordrehzahl und den gewünschten Ruderwinkel auslesen, ohne den Zustand des Gamepads auswerten zu müssen.

7.8.2 Senden von Steuerbefehlen

Aus den Eingaben werden, wie im Systementwurf beschrieben, Steuerbefehle erzeugt, die an das Schiff gesendet werden. In Abbildung 57 sind die Komponenten dargestellt, die für das Senden der Steuerbefehle zuständig sind.

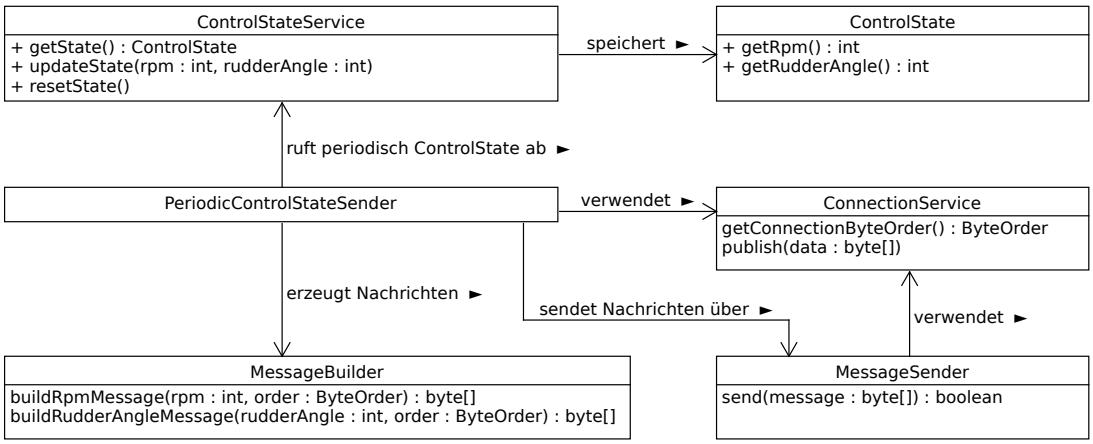


Abbildung 57: Senden von Steuerbefehlen in der Notfallsteuerung

Der PeriodicControlStateSender ruft periodisch den ControlState über den ControlStateService ab. Dafür wird die Methode getState() : ControlState verwendet.

Mit Hilfe des MessageBuilders werden die Steuerbefehle im Byte-Format erzeugt. Die Befehle sind gemäß dem Systementwurf aufgebaut (siehe Tabelle 28). Die Befehle für die Motordrehzahl werden über die Methode buildRpmMessage(rpm : int, order : ByteOrder) : byte[] erzeugt. Die Befehle für den Ruderwinkel werden über die Methode buildRudderAngleMessage(rudderAngle : int, order : ByteOrder) : byte[] erzeugt. Dabei wird die Byte-Reihenfolge verwendet, die der ConnectionService über die Methode getConnectionByteOrder() : ByteOrder vorgibt.

Die erzeugten Befehle werden über den MessageSender an das Schiff gesendet. Dafür wird die Methode send(message : byte[]) : boolean verwendet. Der MessageSender verwendet beim Senden der Befehle die Methode publish(data : byte[]) des ConnectionService.

8 Test

Die Qualität des Softwaresystems ist ein entscheidender Faktor für den reibungslosen und sicheren Betrieb des Produkts (vgl. [SL04], S. 1). Sie wird häufig an der Funktionalität und Zuverlässigkeit gemessen. Merkmale wie die Effizienz, Benutzbarkeit und Erweiterbarkeit sind jedoch ebenso entscheidend (vgl. [SL04], S. 10).

Ein Mangel oder Fehler in einem solchen Softwaresystem ist eine Abweichung von dem geforderten Sollzustand der Software im Vergleich mit dem Istzustand (vgl. [SL04], S. 7). So liegt zum Beispiel ein Fehler vor, wenn das Schiff sich nicht durch die Kontrollplattform steuern lässt, dies aber in den Anforderungen an das Produkt definiert wurde. Klarheit über das Sollverhalten des Softwaresystems wird über die Erstellung von Anforderungen und Spezifikationen gewährleistet. Erstrebenswert ist eine möglichst minimale Abweichung des Istzustands einer Software von ihrem Sollzustand (vgl. [SL04], S. 7). Dies ist folglich gleichbedeutend mit einer geringen Anzahl von Softwaremängeln.

Das Testen einer Software ist ein Prozess, der auf systematische Weise Abweichungen von dem Sollkonzept aufdecken kann. Dieser Prozess beinhaltet das Planen, Durchführen und Auswerten von Tests. Für jeden Testfall werden Randbedingungen, Eingabedaten und erwartete Ausgabedaten definiert. Mehrere Testfälle können in einem Testlauf ausgeführt und deren Ergebnisse danach untersucht werden (vgl. [SL04], S. 8). Die Erkenntnisse der Testläufe können danach genutzt werden, um Fehler des Systems zu beheben. Das Testen trägt demnach dazu bei, die Qualität einer Software zu erhöhen. Ein fehlerfreies Softwaresystem zu entwickeln, gilt jedoch ab einer geringen Komplexität als unmöglich und kann auch durch Tests nicht gewährleistet werden. Tests tragen lediglich zu einer Minimierung der Anzahl von Fehlern bei (vgl. [SL04], S. 9).

Die Tests müssen geplant, spezifiziert, durchgeführt, protokolliert und ausgewertet werden. Der dadurch entstehende Arbeitsaufwand steht dem positiven Aspekt entgegen, dass Fehler durch Tests aufgezeigt werden können (vgl. [SL04], S. 18). Die Testintensität beschreibt als Arbeitsaufwand das Verhältnis zwischen dem Test- und Entwicklungsaufwand und wird bei der Planung sowie Entwicklung eines Softwaresystems festgelegt. Die Festlegung der Testintensität ist abhängig von den Anforderungen an das Softwaresystem (vgl. [SL04], S. 14). Zum Beispiel ist der Testaufwand bei der Entwicklung einer Steuerung einer Rakete höher anzusetzen als der Testaufwand für eine Webseite eines Sportvereins, da signifikante Unterschiede in den Sicherheitsanforderungen bestehen. Im

Allgemeinen sind Testintensitäten von 50% des Gesamtentwicklungsaufwands einer Software ein realistischer Richtwert (vgl. [SL04], S. 14).

8.1 Testprozess

Bei dem Produkt der PG MATE handelt es sich um ein Softwaresystem zur Steuerung eines Schiffs. Nach Abschluss des 3. Meilensteins soll das autonome Fahren ermöglicht werden (siehe Abschnitt 2.1.2). Es handelt sich demnach um ein System bei dem Fehler zu schwerwiegenden Schäden an Personen und der Umwelt führen können. Aus diesem Grund müssen im Besonderen die Anforderungen in Bezug auf die Zuverlässigkeit und Sicherheit korrekt umgesetzt werden. Um dies grundlegend sicherzustellen setzt die PG MATE auf eine hohe Testintensität. Zudem ist der Testprozess fest im Vorgehensmodell verankert.

Das strukturierte Vorgehen der PG MATE während der Entwicklung basiert auf dem V-Modell. In diesem Modell korrespondieren Testarbeiten mit den jeweiligen Entwicklungsarbeiten. Abbildung 1 aus Abschnitt 2.1.1 verdeutlicht diesen Zusammenhang. Im linken Ast des „V“ werden die Entwurfs- und Spezifikationsarbeiten mit jedem Schritt detaillierter. Nach der Entwicklung der einzelnen Komponenten beschreibt der rechte Ast des „V“ die immer gröber werdenden Integrationsschritte und die dazu nötigen Testarbeiten (vgl. [SL04], S. 37). Die Spezifikation einzelner Komponenten wird durch Modultests verifiziert. Ob Gruppen von Komponenten ihrem Systementwurf entsprechen, wird in Integrationstests geprüft. Die Anforderungen an das Softwaresystem werden im Systemtest mit dem Istzustand verglichen. Zuletzt wird in einem Abnahmetest validiert, ob das entstandene Produkt die gestellte Aufgabe erfüllt (vgl. [SL04], S. 39).

8.2 Modultest

Modultests oder auch Komponententests sind Softwaretests, die Fehler in den kleinsten Bausteinen eines Softwaresystems aufdecken sollen. Ein Modultest isoliert dabei ein Softwaremodul und vergleicht dessen Spezifikation mit dem Istzustand. Je nach Programmiersprache gibt es unterschiedliche Definitionen für die getesteten minimalen Softwarebausteine. In Java und anderen objektorientierten Programmiersprachen werden zum Beispiel Klassen mithilfe von Modultests überprüft (vgl. [SL04], S. 40).

Bei einem Modultest handelt es sich um Quellcode, der als eigenständiges Programm ausgeführt werden kann. Das Ergebnis eines Modultests ist binär: Er schlägt entweder fehl

oder ist erfolgreich. Sind alle Modultests erfolgreich, ist die hinreichende Bedingung erfüllt, dass auch das gesamte Softwaresystem funktionsfähig ist. Eine Fehlerfreiheit kann durch Modultests jedoch nicht nachgewiesen werden. Sind einzelne Modultests fehlgeschlagen, liegen Fehler in einzelnen Komponenten vor. Die Fehler in den Modulen führen dazu, dass auch das Softwaresystem nicht den Anforderungen entspricht (vgl. [Hal14], S. 125).

Der Einsatz von Modultests verfolgt verschiedene Ziele. Die wichtigste Aufgabe ist die Überprüfung der Funktionalität des getesteten Softwarebausteins. Zusätzlich wird Robustheit der Komponente überprüft, indem unzulässige Eingabeparameter während eines Testlaufs genutzt werden. In bestimmten Fällen ist es auch sinnvoll die Effizienz einer Komponente zu überprüfen. Dies gilt insbesondere dann, wenn die Komponente auf externe Systeme zugreift, wie zum Beispiel die Festplatte oder das Netzwerk (vgl. [SL04], S. 45).

Modultests sind nach dem folgenden Prinzip aufgebaut (vgl. [Hal14], S. 126):

1. **Arrangieren** des Testumfelds: Das Testobjekt und dessen Zustand muss so erzeugt werden, dass das zu testende Szenario in einem realistischen Umfeld durchgeführt werden kann (vgl. [Hal14], S. 126).

2. **Ausführen** der zu testenden Aktion: Das Szenario wird mithilfe des Testobjekts ausgeführt. In der Praxis besteht dieser Schritt aus wenigen Methodenaufrufen mit vorher festgelegten Eingabeparametern (vgl. [Hal14], S. 127).

3. **Verifizieren** des Ergebnisses: Die Rückgabewerte der aufgerufenen Methoden oder der neue Zustand des Testobjekts werden mit den erwarteten Sollwerten verglichen. Bei Gleichheit ist der Modultest erfolgreich und bei Ungleichheit gilt der Modultest als fehlgeschlagen (vgl. [Hal14], S. 128).

Das Erstellen von Modultests wird in der Softwareentwicklung oft als verpflichtend angesehen. Dies liegt daran, dass die Modultests einen Hinweis auf die Fehlerfreiheit des Gesamtsystems liefern und sich automatisch in kurzer Zeit ausführen lassen (vgl. [Hal14], S. 125). Die PG MATE hat sich zu Beginn geeinigt, diesem Leitsatz zu folgen und für jede Komponente Modultests zu erstellen.

Zusätzlich verfolgt die PG MATE das Vorgehen der testgetriebenen Entwicklung. Bei diesem Vorgehen werden Modultests erstellt bevor die Implementierung der Komponente erfolgt (siehe Abschnitt 2.2.1).

8.2.1 Beispiel der Erstellung von Modultests

Anhand des folgenden Beispiels soll die Vorgehensweise bei der Erstellung von Modultests der PG MATE erläutert werden. Die Softwarekomponente Pfadplanung (siehe Abschnitt 6.5) benötigt einen Speicher für die aktuelle Position des Schiffs, um eine Berechnung des zu fahrenden Pfads durchzuführen. Das Vorgehen der testgetriebenen Programmierung besagt, dass die Tests einer Klasse vor der Implementierung entstehen sollen. So wird sichergestellt, dass die zu entwickelnde Klasse möglichst exakt dem Sollkonzept entspricht und keine fehlende oder überflüssige Funktionalität aufweist.

```
1  public interface PositionState {
2      Optional<Point> getPosition();
3      void setPosition(Point position);
4  }
```

Auflistung 7: Definition der PositionState Schnittstelle in der Pfadplanung

In Auflistung 7 wird die Schnittstelle $\mathrm{PositionState}$ dargestellt. Es ist sinnvoll diese noch vor den Tests zu erstellen, da die Schnittstelle in den Tests genutzt werden kann und der Quellcode übersetzbar bleibt. Mit der Schnittstelle werden die Methoden der zu implementierenden Klasse $\mathrm{PositionStateImpl}$ festgelegt. Es werden zwei Methoden definiert: Eine zum Setzen der aktuellen Position des Schiffs ($\mathrm{setPosition}$) und eine zum Auslesen der gespeicherten Position ($\mathrm{getPosition}$).

```
1  @Test
2  public void setPosition_withPosition_setsPosition() {
3      // arrange
4      PositionState state = new PositionStateImpl();
5      Point expectedPosition = Point.at(1, 2);
6      // act
7      state.setPosition(expectedPosition);
8      // assert
9      assertEquals(expectedPosition, state.getPosition().get());
10 }
```

Auflistung 8: Modultest zum Überprüfen des Setzens einer Position in der Pfadplanung

In Auflistung 8 wird ein Modultest dargestellt, der das spezifizierte Verhalten der Klasse $\mathrm{PositionStateImpl}$ verifiziert. Zuerst wird das Testumfeld arrangiert. Dazu wird eine Instanz der zu testenden Klasse erzeugt. Zudem wird ein Punkt initiiert, der die zu setzende Position beinhaltet, die in diesem Test $(1;2)$ beträgt. Der zweite Schritt ist das

Ausführen der Methode $\mathrm{setPosition}$, welche die Position $(1;2)$ in dem Objekt der Klasse $\mathrm{PositionStateImpl}$ speichert. Zuletzt folgt die Verifizierung, dass die gespeicherte Position dem entspricht, was im vorherigen Schritt gesetzt wurde. Dies wird überprüft indem die aktuelle Position mithilfe der Methode $\mathrm{getPosition}$ abgerufen und mit der Position $(1;2)$ verglichen wird.

Jeder Modultest besteht in Java aus einer eigenen Methode. Die PG MATE setzt ein einheitliches Schema ein, Testmethoden zu benennen. Die Namen setzen sich wie folgt zusammen: $<\mathrm{getestete\ Methode}>\_<\mathrm{Bedingung}>\_<\mathrm{erwartetes\ Ergebnis}>$. Dies hat den Vorteil, dass die Entwickler nur den Namen der Testmethode lesen müssen, um den Zweck des Modultests nachzuvollziehen.

```
1  @Test
2  public void setPosition_withoutPosition_setsPositionToEmpty() {
3      // arrange
4      PositionState state = new PositionStateImpl();
5      // act
6      state.setPosition(null);
7      // assert
8      assertFalse(state.getPosition().isPresent());
9  }
```

Auflistung 9: Modultest zum Überprüfen des Zurücksetzens der Position in der Pfadplanung

Im ersten Test (siehe Auflistung 8) wurde die Position auf einen validen Punkt gesetzt. Die Position kann jedoch auch nicht gesetzt sein bzw. den Wert null annehmen. Dieser Aspekt muss durch einen Modultest abgedeckt werden, der in Auflistung 9 dargestellt wird. Die Testmethode ist der vorherigen sehr ähnlich mit dem Unterschied, dass keine konkrete Position gesetzt wird, sondern der Wert null. Die Verifizierung beschränkt sich darauf zu prüfen, ob eine Position in dem Objekt state gesetzt ist.

Es sind weitere Modultests für die Klasse $\mathrm{PositionStateImpl}$ denkbar. Zum Beispiel könnte die initiale Position spezifiziert und in einem Testfall verifiziert werden.

Wurden alle nötigen Modultests erstellt, kann mit der eigentlichen Implementierung der Klasse begonnen werden. Zu Beginn sollten die Tests fehlschlagen. Wurde die Implementierung anhand der Spezifikation vervollständigt, sind alle Tests der Klasse erfolgreich.

8.2.2 Richtlinien für die Modultesterstellung der PG MATE

Anhand des zuvor beschriebenen Beispiels wurden einige Richtlinien für die Modultesterstellung der PG MATE erläutert:

- Jede Softwarekomponente wird durch Modultests überprüft.
- Eine Testmethode verifiziert jeweils nur eine Klasse.
- Jedes spezifizierte Verhalten wird in einem Modultest verifiziert.
- Testmethoden bestehen aus drei Schritten: Arrangieren, Ausführen und Verifizieren.
- Die Namen der Testmethoden folgen einem definierten Schema.

8.2.3 Testwerkzeuge

Es gibt einige Werkzeuge, die das Erstellen und das Ausführen von Modultests für Entwickler ermöglichen oder erleichtern.

Bei Modultests handelt es sich in Java um Methoden, die von einem Test-Runner ausgeführt werden müssen. JUnit ist ein Test-Framework und beinhaltet einen solchen Test-Runner sowie die benötigten Softwarebibliotheken zur Erstellung von Modultests. Es ist das de facto Standard-Framework bei der Testerstellung in Java, weshalb auch die PG MATE JUnit einsetzt.

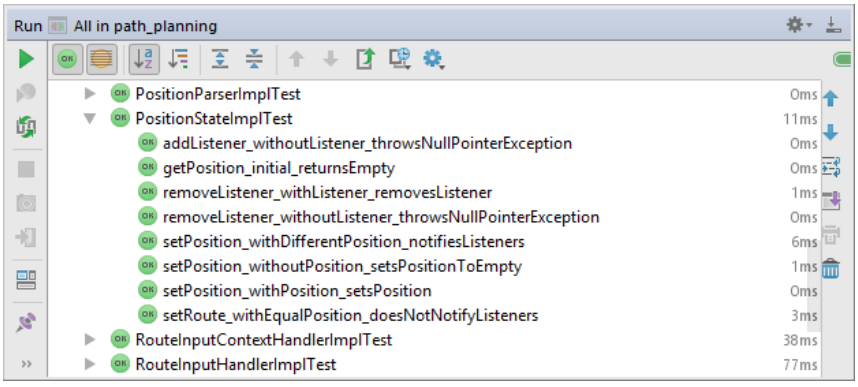


Abbildung 58: Screenshot der Ergebnisse der Modultests der Pfadplanung in IntelliJ IDEA

JUnit ist in vielen Entwicklungsumgebungen integriert. Abbildung 58 zeigt die Ausführung der Modultests der Pfadplanung in der Entwicklungsumgebung IntelliJ IDEA. Es wurden alle vorhandenen Tests erfolgreich ausgeführt. Zu sehen sind insbesondere die

Modultests der Klasse PositionStateImpl, welche in dem Beispiel zur Testerstellung beschrieben wurden. Das Testwerkzeug ermöglicht unter anderem eine schnelle Ausführung der Tests, eine Übersicht über die erfolgreichen und fehlgeschlagenen Modultests sowie die Angabe von Laufzeiten der einzelnen Testmethoden.

8.3 Integrationstest

Modultests sind ein wichtiger Faktor für die Sicherstellung der Funktionalität eines Softwaresystems. Da sie jedoch nur einzelne Komponenten bzw. Klassen betrachten, können und sollen sie deren Zusammenspiel nicht überprüfen. Im V-Modell folgen die Integrationstests auf die Modultests. Ein Integrationstest bezieht sich, anders als ein Modultest, auf eine Gruppe von Komponenten. Er setzt voraus, dass die Funktionalität der zusammenhängenden Komponenten im Einzelnen durch Modultests verifiziert wurden. Integrationstests nutzen den Feinentwurf des Softwaresystems als Basis, um den Istzustand mit dem Sollzustand zu vergleichen (vgl. [SL04], S. 52). Sie sind auf die Schnittstellen und Interaktionen zwischen einzelnen Softwarekomponenten fokussiert (vgl. [SL04], S. 53).

Integrationstests werden in verschiedene Kategorien unterteilt, die von der Auswahl der zu testenden Komponenten abhängen. Diese Testkategorien werden auch Integrationsstufen bezeichnet. Ein Integrationstest von internen Schnittstellen zwischen einzelnen Klassen wird Komponentenintegrationstest genannt. Im Kontrast dazu steht der Systemintegrationstest, der den Zusammenhang zwischen Subsystemen des Softwaresystems betrachtet (vgl. [SL04], S. 54).

Das Ziel der Integrationstests ist es, Schnittstellenfehler zu finden. Grob lassen sich die Fehler in folgende Gruppen unterteilen (vgl. [SL04], S. 56):

- Fehlende Möglichkeit der Integration schon vor der Ausführung des Integrationstests. Dies ist zum Beispiel der Fall, wenn sich Komponenten nicht zusammenfügen lassen oder wenn eine Komponente fehlt.
- Fehlender oder syntaktisch inkorrekter Datenaustausch zwischen Komponenten. Dies lässt auf funktionale Fehler in den Komponenten oder Protokollfehler vermuten.
- Missinterpretation der ausgetauschten Daten. Zum Beispiel tritt ein solcher Fehler auf, wenn die verbundenen Komponenten mit unterschiedlichen Einheiten rechnen. Semantische Fehler sind häufig auf zu ungenaue Spezifikationen zurückzuführen.
- Senden von Daten zu einem falschen Zeitpunkt oder in einem falschen Zeitintervall.

Keine dieser Fehlerkategorien kann durch Modultests aufgedeckt werden. Im Kontrast dazu könnten Integrationstests die Modultests ersetzen, da sie auch funktionale Fehler in den einzelnen Komponenten finden können. Von diesem Vorgehen wird jedoch abgeraten, da der spätere Fehlerdiagnoseaufwand meist höher ist als der Aufwand die Modultests zu erstellen (vgl. [SL04], S. 57). Aus diesem Grund werden von der PG MATE sowohl Modul- als auch Integrationstests eingesetzt.

Für die Erstellung und Ausführungen werden Werkzeuge benötigt. Diese übergeben den zu testenden Subsystemen Eingabedaten, führen Aktionen aus und betrachten im Anschluss die Ausgabedaten. Da dieser Ablauf der gleiche ist wie bei den Modultests, können dieselben Testwerkzeuge eingesetzt werden (vgl. [SL04], S. 55). Folglich wird für jeden Integrationstest eine Methode in Java implementiert. Die Methode folgt denselben Richtlinien wie bei den Modultests, außer dass sich der Test nun auf eine Gruppe von Komponenten bezieht und nicht auf eine einzelne.

Die Integrationstests werden von der PG MATE mithilfe des Test-Frameworks JUnit erstellt und ausgeführt. Jedes größere Subsoftwaresystem wird durch einige Integrationstests auf Schnittstellenfehler überprüft. Eine Ausnahme bilden dabei Komponenten mit einer grafischen Benutzeroberfläche. Dies liegt daran, dass es einen hohen Entwicklungsaufwand bedeutet diese mit automatisierten Integrationstests zu testen. Komponenten mit grafische Benutzeroberflächen werden stattdessen in manuellen Tests durch die Entwickler oder in Systemtests überprüft.

8.4 Systemtest

Die dritte Teststufe im V-Modell nach dem Integrationstest ist der Systemtest. Der Systemtest ist auf das integrierte System fokussiert. Es wird demnach möglichst das gesamte Softwaresystem betrachtet. Der Hauptunterschied zu den Integrationstests ist jedoch, dass nicht ausschließlich anhand der Softwarespezifikation getestet wird, sondern auch auf die Anforderungen Bezug genommen wird (vgl. [SL04], S. 60). Beim Systemtest handelt es sich folglich um die erste Teststufe im V-Modell, die validiert, ob das entwickelte Softwaresystem die Anforderungen der Stakeholder erfüllt. Als Basis für einen Systemtest dienen zudem die Spezifikationen auf der Systemebene (vgl. [SL04], S. 61). Im Fall der PG MATE sind dies vor allem die Akzeptanzkriterien, die für jede User-Story im Entwurf des Softwaresystems festgelegt werden.

Mithilfe des Systemtests sollen primär Abweichungen von den Anforderungen in dem Istzustand des Softwaresystems aufgedeckt werden. Des Weiteren sollen fehlende oder

widersprüchlich beschriebene Anforderungen identifiziert werden. Anders als im Modul- oder Integrationstest ist dieser Schritt im Systemtest möglich, da das gesamte Softwaresystem betrachtet wird (vgl. [SL04], S. 63).

Ein Systemtest sollte in einer Testumgebung ausgeführt werden, die der Umgebung für den späteren Produktiveinsatz sehr nahe kommt. Es sollten demnach möglichst die gleiche Hard- und Software genutzt werden (vgl. [SL04], S. 61). Vermieden werden sollte allerdings das Durchführen des Systemtest in der Produktivumgebung selbst, da Fehler des Softwaresystems zum Beispiel zu Defekten oder anderen Beeinträchtigungen führen können (vgl. [SL04], S. 62).

Der Aufwand der für die Durchführung von Systemtest benötigt wird, darf nicht unterschätzt werden. Es müssen zum Beispiel spezielle Komponenten entwickelt werden, die den Systemtest ermöglichen. Des Weiteren erfordert die manuelle Durchführung und Auswertung des Tests einen gewissen Zeitaufwand (vgl. [SL04], S. 62).

Die PG MATE hat wiederholt Systemtests des entwickelten Softwaresystems durchgeführt. Insbesondere bevor Akzeptanztests auf der Zuse (siehe Abschnitt 8.5) durchgeführt wurden, wurde durch einen Systemtest die grundlegende Funktionalität des MATE Produkts untersucht. Ein Systemtest des MATE Produkts kann von einer einzelnen Person an einem beliebigen Ort durchgeführt werden. Die Zuse hingegen ist eine Ressource, die nicht ständig zur Verfügung steht, weshalb die Zeiten für die Nutzung möglichst effizient gestaltet werden sollten.

Die Durchführung des Systemtests des MATE Produkts folgt keinem vorgegeben Protokoll. Vielmehr handelt es sich um einen explorativen Prozess, der je nach gewähltem Fokus angepasst werden muss. Soll beispielsweise die Effizienz des Systems geprüft werden, ist ein anderes Vorgehen zu wählen als beim Testen der Benutzerfreundlichkeit.

Es wurden einige Softwarekomponenten von der PG MATE entwickelt, die den Systemtest erleichtern oder ermöglichen. Dies ist notwendig, da die Testumgebung nicht der Produktivumgebung (einem Schiff) entspricht. Es müssen demnach einige Komponenten der Systemarchitektur simuliert werden (siehe Abschnitt 6.1.2).

- **File Adapters:** Bei den File Adapters handelt es sich um zwei Adapter. Diese ermöglichen es der polymorphen Schnittstelle Nachrichten von Dateien zu lesen, bzw. Nachrichten als Dateien zu speichern. Diese Funktion wird für den Einsatz der polymorphen Schnittstelle im Produktivbetrieb nicht benötigt. Zu Testzwecken ist dies aber sinnvoll. Zum Beispiel kann die polymorphe Schnittstelle so konfiguriert werden, dass alle Nachrichten, die von dem Datenverteiler empfangen werden, in Da-

teien gespeichert werden. Danach kann mit der Zuse ein Manöver gefahren werden. So werden alle Nachrichten der Schiffssensoren während einer realistischen Fahrt abgespeichert. Diese Dateien können wiederverwendet werden, um ohne den Datenverteiler des Testbeds Systemtests durchzuführen.

– **Navibox Simulator:** Bei dem Navibox Simulator handelt es sich um eine in Java programmierte Konsolenanwendung. Die Komponente hat die Funktion das Testbed während eines Systemtests zu simulieren, indem sie Nachrichten aus Dateien lädt und auf einen Exchange eines RabbitMQ Servers schreibt. Dies simuliert die Datenverteilungskomponente des Testbeds (siehe Abschnitt 6.1.2).

  Die Nachrichten in den Dateien können manuell erstellte Nachrichten sein, um Grenzfälle wie zum Beispiel invalide Daten zu überprüfen. Eine andere Möglichkeit ist, vorher mit dem File Adapter aufgenommen Daten zu nutzen. Ein Systemtest mit solchen realistischen Daten würde der Produktiveinsatz des MATE Produkts ähneln.

8.5 Akzeptanztest

Bei den zuvor in dem Abschnitt 8 beschriebenen Teststufen handelt es sich um Tests, die von der entwickelnden Projektgruppe durchgeführt werden. Diese werden durchgeführt, bevor die Software an einen möglichen Kunden übergeben wird. Bevor ein Softwareprodukt in Betrieb genommen werden kann, muss gemäß V-Modell ein abschließender Test, ein sogenannter Abnahmetest durch den Kunden erfolgen. Ein Abnahmetest ist ein „formales Testen hinsichtlich der Benutzeranforderungen und -bedürfnisse [...], das durchgeführt wird, um einen Auftraggeber oder eine bevollmächtigte Instanz in die Lage zu versetzen, entscheiden zu können, ob ein System anzunehmen ist oder nicht.“ ([SL04]. S. 243)

Bei diesem stehen primär die Sicht des Kunden bzw. des Anwenders im Vordergrund. Der Abnahmetest stellt laut Spillner und Linz den einzigen Test dar, den der Kunde nachvollziehen kann, an dem er beteiligt ist oder für den der Kunde verantwortlich ist (vgl. [SL04], S. 64).

Bevor der beschriebene Abnahmetest erfolgt, ist es möglich Akzeptanztests durchzuführen. Diese werden, anders als der Abnahmetest, nicht vom Kunden durchgeführt, sondern von der Projektgruppe. Dabei wird der Test aus Sicht des Kunden durchgeführt. Die Akzeptanztests können vom Entwicklungsteam im Rahmen vorheriger Teststufen oder

sogar über mehrere Teststufen verteilt erfolgen. Bei der Entwicklung von Individualsoftware besteht laut Spillner und Linz ein hohes Risiko, welches die Durchführung von Akzeptanztests erfordert (vgl. [SL04], S. 64).

Als Basis für umfangreiche Akzeptanztests können unterschiedlichste Dokumente oder Informationen dienen, die das Produkt aus Anwendersicht beschreiben. Im Rahmen der Entwicklung des Produktes der PG MATE sind für diesen Zweck in Abschnitt 5.2 Anwendungsfälle definiert. Darüber hinaus sind in Abschnitt 5.3 funktionale sowie nichtfunktionale Anforderungen erhoben, welche als Basis für Akzeptanztests verwendet werden können.

Aus den definierten Anwendungsfällen, die aus Kundensicht beschrieben sind, werden in diesem Abschnitt Testfälle abgeleitet. Testfälle umfassen die für die Ausführung notwendigen Vorbedingungen, die notwendigen Eingabewerte und die voraussagbaren Sollwerte. Darüber hinaus sind mögliche Nachbedingungen an das zu testende System zu definieren [SL04], S. 264).

Die Testfälle werden in sogenannten Testdrehbüchern festgehalten. Diese umfassen alle Testfälle, die sich an eine oder mehrere Funktionalitäten einer zu betrachtenden Software richten. Für die Durchführung der Akzeptanztests im Rahmen der Entwicklung des Produktes der PG MATE sind die folgenden zwei Testdrehbücher entstanden:

- TDB_001_Nofallsteuerung und
- TDB_002_Schiffssteuerung_EPD_Shore

Diese beinhalten primär Testfälle, die sich zum einen an die Notfallsteuerung richten und zum anderen an die Schiffssteuerung über die Benutzerschnittstelle EPD Shore. Die Testdrehbücher wurden im Rahmen des Projektes inkrementell mit Fortschreiten des Projektes erweitert. Das zeichnet sich dadurch aus, dass die für die Akzeptanztests des 1. Meilensteins erstellten Testfälle innerhalb der Testdrehbücher kontinuierlich verbessert, konkretisiert und um Testfälle für den 2. Meilenstein erweitert wurden. Dies erfolge auf Basis der gewonnenen Erfahrung bei der Ausführung der Akzeptanztests sowie der Weiterentwicklung des Produktes.

Im weiteren Verlauf dieses Abschnitts wird auf den Aufbau der Testdrehbücher sowie auf einzelne Testfälle und Testschritte eingegangen. Darüber hinaus werden Vorbedingungen an das Testbed erhoben, welche für einen erfolgreichen Test einzuhalten sind. Die detaillierte Zeitplanung des Tests ist ebenfalls Inhalt dieses Abschnitts.

8.5.1 Testdrehbücher

Die erstellen Testdrehbücher sind nach einem festen Format aufgebaut. Diese können in Anhang A.2 eingesehen werden. Zunächst besitzen diese eine Übersichtsseite, welche allgemeine Informationen zur Testdurchführung beinhaltet. Diese bestehen aus Informationen zum Test, wie dem Software-Build, dem Namen des Testers und dem Testdatum. Darüber hinaus werden Informationen festgehalten, die das Testsystem betreffen. Die Informationen zum Testsystem beinhalten relevante Informationen zu der verwendeten Hard- und Software und bestehen aus dem verwendeten Prozessor, Arbeitsspeicher, Systemtyp (32-bit oder 64-bit) sowie dem verwendeten Betriebssystem.

Die genannten Informationen werden erhoben, um hard- bzw. softwarespezifische Fehler eingrenzen zu können. Darüber hinaus kann somit nachhaltig festgehalten werden, welche Eigenschaften das verwendete Testsystem hatte.

Neben der Übersichtsseite existiert eine Seite im Testdrehbuch, welche eine Auflistung aller Testfälle beinhaltet. Jeder Testfall besitzt eine Testfall-ID, die aus dem Präfix TF, der fortlaufenden Nummer des Testdrehbuchs sowie einer fortlaufenden Nummer der Testfälle innerhalb des Testdrehbuchs besteht. Ein Beispiel für die Testfall-ID kann wie folgt aussehen: TF_001_001.

Neben der Testfall-ID beinhaltet der Testfall eine Testfallbezeichnung, eine Kurzbeschreibung, eine Priorität sowie Vorbedingungen, die bestehen müssen, bevor ein Testfall durchgeführt werden kann.

Im Folgenden ist eine Übersicht der Testfälle des Testdrehbuchs TDB_001_Notfallsteuerung gegeben:

Testfall 1: Um das Schiff von Land aus zu steuern, ist es notwendig eine Verbindung zum Schiff herzustellen. Eine sichere Kommunikation zum Schiff wird über den Aufbau eines VPN-Tunnels gewährleistet.

Testfall-ID	TF_001_001
Bezeichnung	VPN Verbindung herstellen
Vorbedingungen	(1) Router auf des Schiffs muss konfiguriert sein. (2) IP-Adresse des Routers muss bekannt sein. (3) Das Schiff muss mit dem Internet verbunden sein.
Priorität	Niedrig

Tabelle 31: Testfall 1: VPN-Verbindung herstellen

Testfall 2: Es wird über das Gamepad eine Verbindung zum Schiff hergestellt.

Testfall-ID	TF_001_002
Bezeichnung	Verbindung zum Schiff aufbauen
Vorbedingungen	(1) Notfallsteuerung muss gestartet sein. (2) Gamepad muss verbunden sein.
Priorität	Hoch

Tabelle 32: Testfall 2: Verbindung über das Gamepad aufbauen

Testfall 3: Über die Schaltflächen am Gamepad kann die Motordrehzahl sowie der Ruderwinkel des Schiffs erhöht oder verringert werden.

Testfall-ID	TF_001_003
Bezeichnung	Schiff steuern
Vorbedingungen	(1) Notfallsteuerung muss gestartet sein. (2) Gamepad muss verbunden sein. (3) Verbindung zum Schiff muss vorhanden sein.
Priorität	Hoch

Tabelle 33: Testfall 3: Motordrehzahl und Ruderwinkel über Gamepad steuern

Testfall 4: Es wird über das Gamepad eine bestehende Verbindung zum Schiff abgebaut.

Testfall-ID	TF_001_004
Bezeichnung	Verbindung zum Schiff abbauen
Vorbedingungen	(1) Notfallsteuerung muss gestartet sein. (2) Gamepad muss verbunden sein. (3) Verbindung zum Schiff muss vorhanden sein.
Priorität	Hoch

Tabelle 34: Testfall 4: Verbindung zum Schiff wird über das Gamepad abgebaut

Im Folgenden ist eine Übersicht der Testfälle des zweiten Testdrehbuchs TDB_002_Schiffssteuerung_EPD_Shore gegeben:

Testfall 5: Um eine Verbindung zum Schiff aufbauen zu können, ist es notwendig Verbindungsparameter anzugeben.

Testfall-ID	TF_002_001
Bezeichnung	Verbindung zum Schiff aufbauen
Vorbedingungen	(1) Es besteht eine aktive Internetverbindung. (2) Der RabbitMQ-Server auf dem Duemmer-Server muss gestartet und korrekt konfiguriert sein;
Priorität	Hoch

Tabelle 35: Testfall 5: Verbindung zum Schiff über die EPD aufbauen

Testfall 6: Die Geschwindigkeit sowie Ruderstellung des Schiffes kann über die Virtual Handels der EPD gesteuert werden.

Testfall-ID	TF_002_002
Bezeichnung	Schiff über Virtual Handels steuern
Vorbedingungen	(1) Die EPD muss gestartet sein. (2) Es muss eine Verbindung zum Schiff aufgebaut sein.
Priorität	Hoch

Tabelle 36: Testfall 6: Schiff über Virtual Handles steuern

Testfall 7: In der EPD können diverse schiffsspezifische Daten von beispielsweise Sensordaten dargestellt werden. Diese gilt es im Rahmen dieses Testfalls zu kontrollieren. Dies beinhaltet die syntaktische und semantisch korrekte Darstellung sowie die Verwendung der korrekten Einheit.

Testfall-ID	TF_002_003
Bezeichnung	Schiffsdaten darstellen
Vorbedingungen	(1) Die EPD muss gestartet sein. (2) Es muss eine Verbindung zum Schiff aufgebaut sein.
Priorität	Hoch

Tabelle 37: Testfall 7: Darstellung von Schiffsdaten

Testfall 8: In der EPD wird eine Route erstellt und an das Schiff gesendet.

Testfall-ID	TF_002_004
Bezeichnung	Schiff eine Route vorgeben
Vorbedingungen	(1) Die EPD muss gestartet sein. (2) Es muss eine Verbindung zum Schiff aufgebaut sein.
Priorität	Hoch

Tabelle 38: Testfall 8: Route über EPD erstellen und an Schiff senden

Die aufgeführten Testfälle bestehen aus mehreren Testschritten, die ein Tester durchführen muss, um einen Testfall als erfolgreich bzw. als nicht erfolgreich deklarieren zu können. Jeder Testschritt kann über eine Testschritt-ID identifiziert werden. Diese ist anhand des Präfixes „TS“ zu erkennen. Ein Beispiel für eine Testschritt-ID kann wie folgt aussehen: TS_001_1.

Jeder Testschritt beinhaltet neben der Testschritt-ID eine Testfallbeschreibung sowie eine Vorbedingung, die vorliegen muss, damit der Testschritt durchgeführt werden kann. Des Weiteren ist die genaue Aktion beschrieben, die durch den Tester verrichtet werden muss. Um den Test möglichst fehlerfrei und wiederholbar zu gestalten, werden darüber hinaus Eingabe- bzw. Testdaten angegeben. Es wird ein erwartetes Ergebnis beschrieben, welches bei erfolgreichem Testergebnis erreicht ist. Das festgestellte Testergebnis wird ebenfalls festgehalten. Das Testergebnis des jeweiligen Testfalls kann folgende Werte annehmen: erfolgreich, fehlerhaft, nicht ausführbar, in Ausführung und offen. Die beschriebenen Attribute der Testschritte sind in Anhang A.2 dargestellt.

8.5.2 Testausführung

In diesem Abschnitt wird auf die Testausführung der Akzeptanztests am 19.09.2017 eingegangen. Dieser fand in Wilhelmshaven in Verbindung des MATE Produkts mit dem Forschungsschiff Zuse statt. Im Rahmen der Testausführung wurden die Testschritte der Testfälle aus den Testdrehbüchern TDB_001_Nofallsteuerung und TDB_002_Schiffssteuerung_EPD_Shore ausgeführt.

Der Test, bei dem die Zuse sich auf dem Wasser befindet, ist an explizite Vorbedingungen geknüpft, damit der Test ausgeführt werden kann. Die Vorbedingungen beziehen sich auf die Umwelt, Hard- und Software sowie an Personen, die sich auf der Zuse befinden müssen, damit ein Test reibungslos durchgeführt werden kann.

Im Folgenden werden Vorbedingungen an das Testbed aufgeführt, die verpflichtend für die Ausführung aller Testschritte anzusehen sind:

Umwelt

TB_REQ1: Die Windstärke muss an der Lokation, wo der Test durchgeführt wird, unter 4 Bft liegen.

TB_REQ2: Der Wellengang muss einen zuverlässigen und ungefährlichen Test ermöglichen.

Hardware

TB_REQ3: Die Zuse muss in Wilhelmshaven im Wasser liegen.

TB_REQ4: Die Zuse muss vollgetankt sein, um Fahrten im Hafenbecken und auf dem Meer zu ermöglichen.

TB_REQ5: Die Zuse muss eine vollständig geladene Batterie haben.

TB_REQ6: In der Nähe der Testlokation sollte eine freie Steckdose (230V) für das Aufladen von Notebooks zum Testen zur Verfügung stehen.

TB_REQ7: Im Testbed müssen folgende Hardware-Komponenten zur Verfügung stehen:

- Navibox
- Raspberry PI
- Industrie-PC

TB_REQ8: Folgende Sensoren müssen gestartet und mit der Navibox verbunden sein (Eine Übersicht der Sensoren ist in Anhang A.4 gegeben):

- GPS Location
- Speed through Water
- Speed/Course over Ground
- Engine RPM
- Rudder Angle
- IMU
- Heading
- Rate of Turn

- Windspeed an direction
- Water Depth
- AIS Position Report
- AIS Stativ Voyage
- Report
- Radar Track

Software

TB_REQ9: Das Testbed muss ein Netzwerk mit aktiver Internetverbindung bereitstellen.

TB_REQ10: Die dynamische IP-Adresse und der Port der Komponente Schiffssteuerung muss vorliegen.

TB_REQ11: Das Testbed muss fähig sein einen VPN-Zugang zum lokalen Netzwerk des Schiffs zur Verfügung zu stellen.

TB_REQ12: Die Skripte der Komponente Schiffssteuerung müssen gestartet sein.

TB_REQ13: Die Navibox muss im Testbed gestartet sein.

TB_REQ14: Die lokalen RabbitMQ Queues und Exchanges der Zuse müssen gestartet und die Zugangsdaten bekannt sein.

TB_REQ15: Die externen RabbitMQ Queues und Exchanges der Zuse müssen gestartet und die Zugangsdaten bekannt sein.

TB_REQ16: Die Navibox in Brunsbüttel und die Navibox in Cuxhaven müssen gestartet sein und die AIS Receiver müssen aktiv sein.

TB_REQ17: Alle Daten vom Festland auf dem Wilkinson-Server müssen per Exchange_Routing_Keys auf die RabbitMQ Queue pgMate verweisen.

Personen

TB_REQ18: Für den Test muss ein Bootsführer mit einem Sportbootführerschein sowie einem Funkschein nach Short Range Certificate zur Verfügung stehen.

Für die Ausführung des Tests in Verbindung mit dem Testbed müssen die erhobenen Bedingungen erfüllt sein. Darüber hinaus müssen eine Reihe von Komponenten korrekt konfiguriert sein. Eine Übersicht über die Parameter der jeweiligen Komponenten ist in den jeweiligen Handbüchern in Anhang A.1 gegeben.

Neben den definierten Bedingungen an das Testbed für die Durchführung von Tests ist ein genauer Testablauf notwendig. Dieser ist in Anhang A.5 vorzufinden. Darin ist der genaue zeitliche Ablauf der An- und Abfahrt, der Tests sowie eventuelle Pausen protokolliert.

Die protokollierten Ergebnisse der Tests befinden sich in Anhang A.2. Darin sind alle Testschritte einsehbar, die durchgeführt wurden.

9 Evaluation

In diesem Kapitel wird die Evaluation der Ergebnisse der Projektgruppe durchgeführt. Zuerst wird anhand der erhobenen Anforderungen das Produkt evaluiert. Anschließend wird die Projektarbeit anhand des Projektvorgehens reflektiert.

9.1 Anforderungen

Die in diesem Abschnitt aufgeführte Evaluation wird auf Basis der in Abschnitt 5.2 sowie der in Abschnitt 5.3 erhobenen funktionalen und nichtfunktionalen Anforderungen durchgeführt. Die beschriebenen Anwendungsfälle stellen eine Übersicht der gewünschten Funktionalitäten des Produktes der PG MATE aus Sicht der Stakeholder dar. Diese sind in Abschnitt 5.1 aufgeführt.

Auf Basis der Anwendungsfälle sind funktionale und nichtfunktionale Anforderungen abgeleitet worden. In den nachfolgenden Abschnitten werden die jeweiligen Anforderungen aufgegriffen, der Erfüllungsgrad untersucht und beschrieben.

9.1.1 Erfüllte Anforderungen

Im Folgenden wird eine Übersicht der erfüllten Anforderungen aufgeführt, die zuvor in Abschnitt 5.3 erhoben wurden. Darüber hinaus wird im Einzelnen kurz beschrieben, wie die Lösung zu der definierten Anforderung umgesetzt ist.

Anforderungs-ID	**Beschreibung**
MATE_REQ1	Das MATE Produkt besitzt ein PreferencesPanel in der EPD-Shore. Über dieses kann der Küstenleitstand Daten zum Herstellen einer Verbindung eingeben, mit welcher Daten zwischen Kontrollplattform und Schiff übertragen werden können.
MATE_REQ2	Das MATE Produkt kann über das Speichern der Einstellungen eine Verbindung zum Schiff aufbauen.
MATE_REQ3	Das MATE Produkt besitzt ein PreferencesPanel in der EPD-Shore. Über dieses kann die Schiffsbesatzung Daten zum Herstellen einer Verbindung eingeben, mit welcher Daten zwischen Kontrollplattform und Schiff übertragen werden können.
MATE_REQ4	Das MATE Produkt kann über das Speichern der Einstellungen in der EPD-Shore eine Verbindung zum Schiff aufbauen.

MATE_REQ5	Kann das MATE Produkt keine aktive Verbindung zum Schiff aufbauen, wird im Statusbereich der EPD-Shore ein rotes Statussymbol dargestellt.
MATE_REQ6	Sobald das MATE Produkt eine aktive Verbindung zum Schiff hat, wird im Statusbereich der Benutzeroberfläche ein grünes Statussymbol für die Verbindung dargestellt.
MATE_REQ7	Das MATE Produkt besitzt in den Virtual Handles Regler, über die Steuerdaten in Form von Motordrehzahl und Ruderwinkel an das Schiff gesendet werden.
MATE_REQ8	Über eine vom Produkt der PG MATE unabhängige Applikation ist es dem Küstenleitstand möglich, über eine Benutzeroberfläche Verbindungsdaten des Schiffs einzugeben.
MATE_REQ9	Die Notfallsteuerung ermöglicht dem Küstenleitstand über ein angeschlossenes Gamepad einen Verbindungsaufbau zum Schiff zu initiieren.
MATE_REQ10	Die Notfallsteuerung ermöglicht dem Küstenleitstand über ein angeschlossenes Gamepad eine bestehende Verbindung zum Schiff zu trennen.
MATE_REQ11	Es ist dem Küstenleitstand durch die Notfallsteuerung möglich, über ein angeschlossenes Gamepad die Motordrehzahl zu erhöhen bzw. zu verringern.
MATE_REQ12	Es ist dem Küstenleitstand durch die Notfallsteuerung möglich, über ein angeschlossenes Gamepad die Ruderstellung nach links bzw. nach rechts zu bewegen.
MATE_REQ13	Wenn die Schaltflächen des Gamepads auf einer neutralen Position stehen, sendet die Backup-Steuerung Daten, sodass die Motordrehzahl des Schiffs in einem Leerlauf und die Ruderstellung des Schiffs in einer neutralen Position sind.
MATE_REQ14	Das MATE Produkt ist fähig, vom Küstenleitstand definierte Steuerdaten an das Schiff zu senden.
MATE_REQ15	Das MATE Produkt ist fähig, alle definierten Sensordaten in *MATE_REQ15* in Form von Nachrichten des NMEA2000-Standards abzurufen.

MATE_REQ16	Das MATE Produkt ist fähig AIS-Daten vom Testbed abzurufen. Über die polymorphe Schnittstelle können bei Bedarf weitere externe Systeme angeschlossen werden.
MATE_REQ17	Das MATE Produkt stellt alle definierten Sensordaten in *MATE_REQ17* in den Virtual Handles dar.
MATE_REQ20	Sobald das MATE Produkt die bestehende Verbindung zum Schiff abgebaut hat, wird in der EPD-Shore ein rotes Statussymbol für die Verbindung dargestellt.
MATE_REQ21	Das MATE Produkt bietet dem Küstenleitstand die Möglichkeit, mithilfe des RoutesManager in der Seekarte von der EPD-Shore eine oder mehrere Routen, bestehend aus 2 bis n Wegpunkten zu erstellen, zu speichern, zu bearbeiten und zu löschen.
MATE_REQ22	Das MATE Produkt stellt dem Küstenleitstand in der EPD-Shore alle definierten Routen, Wegpunkte sowie deren Verbindungslinien auf der Seekarte dar.
MATE_REQ23	Das MATE Produkt kann eine zuvor definierte und vom Küstenleitstand aktivierte Route im RTZ-Format an das Schiff senden.
MATE_REQ25	Das MATE Produkt kann in den Virtual Handles vom Küstenleitstand einen Autopiloten mit einer zuvor definierten Route aktivieren.
MATE_REQ26	Sobald der Autopilot aktiviert ist, wird die aktive Route vom MATE Produkt an das Schiff gesendet.
MATE_REQ27	Sobald der Autopilot aktiviert ist, wird im Statusbereich der EPD-Shore ein grünes Statussymbol für den Autopiloten dargestellt.
MATE_REQ28	Das MATE Produkt kann in den Virtual Handles vom Küstenleitstand einen aktiven Autopiloten deaktivieren.
MATE_REQ29	Sobald der Autopilot deaktiviert ist, wird die Route vom MATE Produkt auf den Status inaktiv gesetzt.
MATE_REQ30	Sobald der Autopilot deaktiviert ist, wird im Statusbereich der EPD-Shore ein rotes Statussymbol für den Autopiloten dargestellt.

MATE_REQ31	Das MATE Produkt kann dem Küstenleitstand über die EPD-Shore die Position des eigenen Schiffes in einer Seekarte darstellen und farblich (rot) von anderen Schiffen hervorheben.
MATE_REQ32	Das MATE Produkt kann der Schiffsbesatzung über die EPD-Shore die Position des eigenen Schiffes in einer Seekarte darstellen und farblich (rot) von anderen Schiffen hervorzuheben.
MATE_REQ33	Das MATE Produkt kann die aktuelle Position des eigenen Schiffs und von möglichen anderen Schiffen vom eigenen Schiff abrufen.
MATE_REQ44	Das MATE Produkt ist fähig, auf den Betriebssystemen Windows 7 64-bit sowie Linux 64-bit ausgeführt zu werden.
MATE_REQ45	Das MATE Produkt unterstützt den NMEA0183-Standard.
MATE_REQ46	Das MATE Produkt unterstützt den NMEA2000-Standard.
MATE_REQ47	Das MATE Produkt unterstützt den RTZ-Standard.
MATE_REQ48	Das MATE Produkt unterliegt zum aktuellen Entwicklungsstand keinen Regularien des Bundesgesetzes über die Binnenschifffart. Die Anforderung gilt als erfüllt, da keine Regularien verletzt werden.
MATE_REQ49	Das MATE Produkt unterliegt zum aktuellen Entwicklungsstand keinen Regularien des Hafenverkehrs- und Schifffahrtsgesetzes. Die Anforderung gilt als erfüllt, da keine Regularien verletzt werden.
MATE_REQ50	Das MATE Produkt weist eine modulare sowie testbare Softwarearchitektur auf. (Siehe Abschnitt 6).
MATE_REQ51	Das MATE Produkt weist eine hohe Testabdeckung von 97 Prozent bis 100 Prozent der jeweiligen Komponenten auf.
MATE_REQ52	Das MATE Produkt ist fähig, in einer akzeptablen Latenzzeit Daten zu verarbeiten. Eine Erfüllung dieser Anforderung ist dadurch gegeben, dass keine spürbaren Wartezeiten bei der Verarbeitung von Befehlen gegeben sind.
MATE_REQ53	Das MATE Produkt hält keine Daten vor, die keine begründete Anwendung finden.

MATE_REQ54	Die referenzierte Anforderung stellt eine komplexe Anforderung dar, die nicht vollumfänglich getestet und somit gewährleistet werden kann. Es existieren eine hohe Anzahl von möglichen Parametern in diversen Kombinationen. Auf Basis der durchgeführten Akzeptanztests können ausreichend valide Daten gewährleistet werden.
MATE_REQ55	Die Komponenten des MATE Produkts verfälschen bei der Bearbeitung sowie Weiterleitung keine Sensordaten. Dies wurde im Rahmen von System- sowie Akzeptanztests überprüft.
MATE_REQ56	Das MATE Produkt stellt sicher, dass Steuerbefehle korrekt an die Zuse gesendet werden und valide sind. Dies wurde im Rahmen von System- und Akzeptanztests überprüft.

Tabelle 39: Erfüllte Anforderungen

9.1.2 Nicht erfüllte Anforderungen

Dieses Unterkapitel listet die nicht erfüllten Anforderungen in den jeweiligen Anwendungsfällen auf. Anschließend wird evaluiert wieweit diese erfüllt sind und was der Grund zur Nichterfüllung ist.

Anwendungsfall 7

MATE_REQ18	Das MATE Produkt muss dem Küstenleitstand die Möglichkeit bieten, über die Benutzeroberfläche eine bestehende Verbindung zum Schiff abzubauen.
MATE_REQ19	Das MATE Produkt muss der Schiffsbesatzung die Möglichkeit bieten, über die Benutzeroberfläche eine bestehende Verbindung zum Schiff abzubauen.

Tabelle 40: Nicht erfüllte Anforderungen - Anwendungsfall 7

Die EPD-Shore besitzt keine direkte Möglichkeit die Verbindung abzubauen. Sie baut die Verbindung ab, sobald sie geschlossen wird oder Einstellungen geändert und gespeichert werden. Will der Nutzer die Verbindung abbauen, muss er somit eine falsche oder nicht

vorhandene Verbindungseinstellung speichern. Kann die EPD-Shore keine Verbindung aufbauen oder wird beendet, wird das Statussymbol für die Verbindung rot dargestellt. *MATE_REQ20* ist damit erfüllt.

Anwendungsfall 10

MATE_REQ24	Das MATE Produkt wird dem Küstenleitstand die Möglichkeit bieten, über die Benutzeroberfläche eine aktive Route zu bearbeiten, während das Schiff diese abfährt.

Tabelle 41: Nicht erfüllte Anforderungen - Anwendungsfall 10

Die Anforderung *MATE_REQ24* wurde in Absprache mit den Stakeholdern als nicht dringlich aufgenommen und hat eine niedrige Priorität erhalten. Es existiert für die Bearbeitung einer aktiven Route der Workaround, eine aktive Route zu deaktivieren und im Anschluss eine neue aktive Route einzuspielen. Aufgrund dieser Tatsache wurde die Umsetzung der Funktionalität nicht höher priorisiert und aufgrund anderer höher priorisierter Funktionen nicht umgesetzt.

Anwendungsfälle 15 bis 20

Die Umsetzung eines autonom fahrenden Schiffes stellt ein komplexes und zeitaufwändiges Unterfangen dar. Aufgrund eines befristeten Zeitraums des Projekts auf 12 Monate sowie einer Organisationsphase zu Beginn des Projektes, konnte der dritte Meilenstein nicht umgesetzt werden. Aufgrund dessen sind die Anforderungen *MATE_REQ35* bis *MATE_REQ43* der Anwendungsfälle 15 bis 20 als nicht erfüllt einzustufen.

9.2 Projektvorgehen

Während der Projektarbeit zeigten sich einige Schwächen und Probleme beim Projektvorgehen. Über die aufgekommenen Probleme wurde innerhalb der Projektgruppe gesprochen, um so die Ursache der Probleme zu identifizieren und konkrete Maßnahmen zur Lösung dieser Probleme zu erarbeiten. Diese jeweiligen Probleme, ihre Ursachen und die daraus resultierten Maßnahmen werden im Folgenden vorgestellt. Der Aufbau und die untersuchten Elemente richten sich dabei nach dem Aufbau des Kapitels zum Projektvorgehen (siehe Abschnitt 2).

9.2.1 Projektorganisation

In diesem Abschnitt soll die Projektorganisation reflektiert werden. Dies geschieht anhand des gewählten Vorgehensmodells sowie der allgemeinen Projektplanung. Die allgemeine Projektplanung umfasst dabei unter anderem die Aufteilung in Zyklen und Meilensteine sowie das Einhalten von Fertigstellungsterminen.

Vorgehensmodell

Während der Arbeit in den Zyklen fiel auf, dass innerhalb eines Zyklus oftmals nicht am erarbeiteten V-Modell festgehalten wurde, das in einem jeden Zyklus durchlaufen werden sollte. Dies wurde vor allem durch fehlende Anforderungen deutlich. Die in den User-Stories definierten Akzeptanzkriterien wurden in der Konstruktion und in der Entwicklung vernachlässigt. Erst im Nachhinein wurden Defizite in der Umsetzung und Implementierung deutlich, deren Korrektur zu diesem Zeitpunkt mit einem erhöhten Aufwand verbunden war. Dieses Problem wurde in den späteren Zyklen durch ein konsequenteres Festhalten am definierten V-Modell abgemildert.

In der Theorie sah das erarbeitete Vorgehensmodell vor, einen Zyklus erst zu beginnen, wenn der vorherige abgeschlossen wurde. Der vorherige Zyklus sollte nach dem Abschluss in der Retrospektive reflektiert werden, um so Erkenntnisse für Verbesserungen am Vorgehen für den Folgezyklus zu sammeln. In der Praxis zeigte sich jedoch, dass einzelne Aufgaben deutlich länger andauerten als geplant. Dies waren vor allem Aufgaben, die mit externen Stakeholdern abgestimmt werden mussten. Damit nicht die übrige Projektgruppe, deren Arbeiten am Zyklus bereits abgeschlossen waren, auf den Abschluss dieser Aufgaben warten musste, wurde bereits mit den Arbeiten am nächsten Zyklus begonnen. Dies führte jedoch zu einem unsauberen Übergang in den nächsten Zyklus, wodurch unter anderem die Retrospektive nicht wie geplant durchgeführt werden konnte. Dadurch gingen mögliche Erkenntnisse verloren und die potentiell verbesserbare Arbeitsmethodik wurde unverändert weitergeführt. Auch dieses Problem wurde in den späteren Zyklen durch ein Festhalten am definierten V-Modell behoben. Dies wurde erreicht, indem auch die Projektgruppenmitglieder, die an noch nicht abgeschlossenen Aufgaben des vorherigen Zyklus arbeiteten, in den Folgezyklus eingebunden wurden.

Die Arbeit in Teilgruppen ermöglichte ein paralleles Arbeiten an verschiedenen größeren Teilaufgaben und Komponenten. Während der Projektarbeit zeigten sich jedoch Probleme in der Abstimmung der Teilgruppen untereinander. Deren Ergebnisse mussten

am Ende im PG Produkt nahtlos ineinander übergreifen. Durch Abstimmungsschwierigkeiten war dies nicht immer auf Anhieb möglich. Diese Problematik zeigte sich schon früh in der Projektarbeit. Das Einführen der Statusberichte und zusätzliche Treffen, an denen die Teilgruppen ihre (Zwischen-)Ergebnisse vorstellten, konnten diesem Problem im späteren Projektverlauf vorbeugen. Durch die wöchentlichen Statusberichte fand unter anderem durch vermehrte Rückfragen ein verbesserter Informationsaustausch zwischen den Teilgruppen statt. Die Zusatztermine zur Präsentation von (Zwischen-)Ergebnissen machten die Arbeit der Teilgruppenmitglieder allen Projektgruppenmitgliedern vertraut.

Insgesamt zeigte sich, dass die Erarbeitung des Vorgehensmodells nicht vollständig und methodisch korrekt durchgeführt wurde. Die Tailoring-Maßnahmen am V-Modell waren in ihrer Grundannahme zwar korrekt, denn das Standard V-Modell wäre für die Arbeit in mehreren Zyklen, die notwendig waren, nicht geeignet gewesen. Auch die Unerfahrenheit der Projektgruppenmitglieder im Projektkontext hätte beim Vorgehen nach dem allgemeinen V-Modell zu Planungsfehlern in den frühen Entwurfsphasen geführt, die in der Implementierung nicht mehr hätten behoben werden können. Allerdings waren die Anpassungen am V-Modell mangelhaft. Dies hat dazu geführt, dass die Abgrenzung zwischen Entwurf und Implementierung in den Phasen nicht genau definiert war. Dies führte zu vielen Problemen während der Projektarbeit und zeigte sich zuletzt vor allem bei der Erarbeitung der Abschlussdokumentation deutlich. Die Trennung der Abschnitte 6 und 7 bereitete einige Probleme, da nicht genau spezifiziert war, was in welchem der beiden Projektphasen durchgeführt wurde.

Abschließend lässt sich hier sagen, dass es besser gewesen wäre, sich beim Vorgehensmodell näher an etablierten Lösungen zu orientieren. Das allgemeine Vorgehensmodell wurde in der tatsächlichen Arbeitsweise der PG MATE kaum angewandt. Die Arbeit, zum Beispiel nach Scrum, hätte möglicherweise zu einem besseren Arbeitsergebnis führen können.

Projektplanung

Bei der Projektplanung zeigte sich, dass die Arbeit in Zyklen sich als richtig erwies. Jedoch waren die Zyklen vor allem zum Beginn des ersten Zyklus zu umfangreich und hätten granularer aufgebaut werden müssen. Durch diese zusätzlichen Iterationen wäre es möglich gewesen, schneller Erkenntnisse zu sammeln und zu evaluieren. Außerdem hätten so die im vorherigen Abschnitt erwähnten Probleme mit der Einhaltung des V-Modells umgangen werden können. Dieses Problem wurde schon während der Projektarbeit

erkannt und dadurch gelöst, dass die letzten beiden Zyklen deutlich kürzer waren als der erste Zyklus.

Generell zeigte sich, dass geplante Fertigstellungstermine, wie sie zu Beginn eines Zyklus in der Konzeption bzw. im Gantt-Diagramm festgelegt wurden, nicht eingehalten werden konnten. Um dieses Problem zu beheben, wurden verschiedene Methoden und Techniken angewandt. Zunächst war dies das Gantt-Diagramm, was jedoch zu keiner nennenswerten Verbesserung geführt hat. Die später eingeführten Statusberichte konnten diesem Problem besser entgegenwirken. Identifizierbare Risiken, die nicht direkt zu beheben waren, wurden in die ebenfalls eingeführte Projektrisikoliste aufgenommen. Dadurch war es möglich diese Risiken laufend zu überwachen und frühzeitig vorbeugende Maßnahmen ergreifen zu können. Durch diese Maßnahmen zum Risikomanagement konnten Fertigstellungstermine in den späteren Zyklen besser eingehalten werden. Dies und das konsequente Setzen und Einhalten von Deadlines führte zu einer besseren Terminsicherheit.

9.2.2 Rollenverteilung

Die Rollenverteilung aus Abschnitt 2.1.3 zeigt die einzelnen Rollen, die nachfolgend reflektiert werden.

- **Projektleitung**: Die Projektleitung ist ein sehr wichtiger Bestandteil der Projektgruppe gewesen. Besonders hat die Projektleitung das gesamte Projekt vor Augen und kann somit schnell Probleme sehen. Auch als Problemlöser wurde die Projektleitung eingesetzt und hat oft an Terminen aktuelle Probleme angesprochen. Dies umfasste zum Beispiel anbahnende zwischenmenschliche Problemen oder Projektrisiken.
- **Projektplanung**: Die Projektplanung ist als Rolle oft eingesetzt worden. So wurden als Beispiel die Gantt-Diagramme erstellt und mit den Stakeholdern besprochen. Die Projektplanung war auch für die Einhaltung des Vorgehensmodells zuständig.
- **Teilprojektleiter**: Ein Teilprojektleiter ist anfangs nicht bestimmt worden. Im späteren Projektverlauf sind mehrere Teilprojektleiter für sich ständig wechselnde Untergruppen ernannt worden. Diese sind für die unterschiedlichen internen Teilprojekte zuständig gewesen. Sie haben die Statusberichte ausgefüllt und der Gruppe vorgestellt, sowie die zeitliche Einordnung der Teilprojekte innerhalb des Projekt-

verlaufs im Blick gehabt. Dies hat sich als eine nützliche Methode für die Gruppe herausgestellt.

- **Admin**: Ein Administrator ist ein wichtiger Bestandteil gewesen. Der Admin betreute mehrere externe Software-Systeme, mit denen im Projekt gearbeitet wurde. Ein Ausfall von externen Systemen kam häufiger vor, sodass der Admin hier eine schnelle Lösung bieten musste.
- **Dokumentationsbeauftragter**: Der Dokumentationsbeauftragte wurde nicht in dem Maße eingesetzt, wie es am Anfang vorgesehen wurde. Während des Projektverlaufs wurden zwar Protokolle und Notizen zu den einzelnen Projektelementen angefertigt, allerdings nicht im komplexem und zusammenhängendem Ausmaß. Die Ausarbeitung zu einer richtigen Dokumentation wurde erst spät von der gesamten Projektgruppe bearbeitet. Durch Verzögerungen in der Implementierung und beim Testen kam es auch in der vorher definierten Dokumentationsphase zu Verzögerungen.
- **Inventarbeauftragter**: Ein Inventarbeauftragter ist nicht notwendig gewesen, da sich nur ein einziger Gegenstand im Inventar der Projektgruppe befand. Dies war zu Beginn der Projektgruppe allerdings nicht vorauszusehen und wurde daher trotzdem als wichtig erachtet.
- **Qualitätsbeauftragter**: Der Qualitätsbeauftragte ist in dieser Projektgruppe vor allem zu Beginn der Implementierungsphase notwendig gewesen. Die einzelnen Projektgruppenmitglieder hatten unterschiedlich großes Wissen darüber, wie qualitativ hochwertiger Code geschrieben wird. Der Qualitätsbeauftragte hatte stets ein Auge darauf, dass alle sich an die wichtigsten Konventionen halten. Durch Merge Requests und testgetriebener Entwicklung ist die Softwarequalität auf einem hohen Stand.
- **Testbeauftragter**: Für die Testplanung und -durchführung ist ein Testbeauftragter sehr wichtig, da er für die Tests auf der Zuse zuständig war. Dadurch konnte eine sehr genaue Testdurchführung der einzelnen Komponenten sichergestellt werden.
- **Öffentlichkeitsbeauftragter**: Die Aufgabe des Öffentlichkeitsbeauftragten war es eine Webseite mit den Ergebnissen der Projektgruppe zu pflegen. Von den Betreuern benötigte Poster wurden kurz vor der Fertigstellung hinfällig, da der Vorstellungstermin mit der Zwischenpräsentation leider abgesagt werden mussten. Da die

Website eine Pflichtleistung für die Projektgruppe ist, ist es definitiv notwendig einen Ansprechpartner für die Öffentlichkeitsarbeit zu haben.

– **Raum-/Transportbeauftragter**: Der Raum-/Transportbeauftragter hat die Projektgruppe und die Stakeholder über jede Terminänderung in Kenntnis gesetzt. Auch als Ansprechpartner für die Räumlichkeiten war diese Person zuständig. Daher ist ein Raum-/Transportbeauftragter für die Projektgruppe ein wichtiger Bestandteil gewesen.

9.2.3 Projektmethoden

Im Folgenden wird der Einsatz der „testgetriebenen Entwicklung“ , der „Retrospektive“ und der „Brainstorming“-Methode reflektiert.

Arbeit mit der „testgetriebenen Entwicklung“

In der Reflexion des Einsatzes der Methode zeigt sich, dass die theoretischen Nachteile auch in der Praxis eintreten. Nur bei dem konsequenten Vorgehen nach der testgetriebenen Entwicklung, wenn Tests tatsächlich als Erstes geschrieben wurden, zeigten sich die Vorteile, ansonsten traten Probleme auf. Auch der Einarbeitungsaufwand war höher als erwartet und hat den Projektfortschritt zu Beginn gehemmt. Allerdings ist die Software-Qualität des Endprodukts hoch und die Projektgruppenmitglieder haben während der Arbeit mit der Methode viel gelernt. Zudem zeigte sich, dass durch die Arbeit mit der Methode schnell auf geänderte Anforderungen reagiert werden konnte, da durch die automatischen Modultests „Seiteneffekte“ bei Änderungen besser bemerkt werden konnten.

„Retrospektive“-Methode

In der Reflexion der Methode zeigt sich, dass der Einsatz sinnvoll und konstruktiv war. Allerdings hätte die Retrospektive durchaus öfter durchgeführt werden können. Dafür wären kürzere Zyklen sinnvoll gewesen.

„Brainstorming“-Methode

Ziel der Brainstorming-Methode war es zum Beginn eines jeden Zyklus, die anstehenden Aufgaben, die von der Projektgruppe im nächsten Zyklus umgesetzt werden sollten, zu

sammeln und zu strukturieren. Für diesen Einsatz war die Brainstorming-Methode geeignet und lieferte gute Ergebnisse.

9.2.4 Kommunikationswerkzeuge

GitLab

Das GitLab-Projekt bietet alle Funktionen welche von einer Versionsverwaltungslösung benötigt werden. Außerdem bietet es die Möglichkeit zur Dokumentation des Projekts und Verwaltung der Aufgaben in Kanban-Boards (siehe Absatz 2.3.4).

Des Öfteren kam es zu Fehlern der Hardware-Infrastruktur der IT-Dienste. Aus diesem Grund war kein Zugriff auf die serverseitigen Repositories möglich. Dies ist für ein eng getaktetes Projekt ein Problem. Es ist während der Ausfallzeit nicht möglich serverseitige Backups zu erzeugen. Außerdem kann der Quellcode nicht automatisiert zusammengeführt werden. Dies behindert im schlimmsten Fall den Projektfortschritt. Alle auftretenden Fehler ließen sich schnell beheben, sodass die Infrastruktur kurzzeitig gefährdet schien, jedoch nicht auf einen anderen GitLab-Anbieter umgestellt werden musste.

Die Einarbeitung und Standardisierung von GitLab war aufwändiger als angenommen. In den internen Terminen wurden die Schwierigkeiten genannt und die Arbeitsweise mit GitLab wurde vereinheitlicht. Nachdem die anfänglichen Probleme gelöst waren, profitierten alle Beteiligten und insbesondere das Projektmanagement von den Funktionen GitLabs. Durch die starke Verbindung in GitLab zwischen Projektmanagementwerkzeug und den Repositories profitierte die Rückverfolgbarkeit im besonderen Maße.

Telegram

Die Telegram-Gruppen waren geplant für den Einsatz als schneller Kommunikationskanal. Für diesen Zweck hat sie der Projektgruppe rückblickend einen guten Dienst geleistet. Hinzu bot Telegram die Möglichkeit sicherheitsunkritische Dateien und Dokumente zu verteilen. Mit mehr als zehn verschiedenen Teilgruppen-Chats war es sehr einfach, die interne direkte Kommunikation in themenbezogene Kanäle zu unterteilen. So war es möglich verschiedenste Probleme gleichzeitig zu diskutieren ohne den Gesprächsfluss anderer Teilgruppen zu stören oder zu unterbrechen.

Einzelne Teilnehmer der Projektgruppe hatten nun nicht mehr vollen Zugriff auf alle Informationen jeder Teilgruppe. Dieser Aspekt war allen Teilnehmern bewusst und wurde

im Austausch für eine bessere Übersichtlichkeit in Kauf genommen. Telegram ist ein unkomplizierter und schneller Kommunikationskanal. Daraus resultiert, dass Diskussionen über Telegram geführt wurden, die an einer anderen Stelle hätten geführt werden müssen. Diskussionen die zu einem Git-Issue gehörten wurden aus Bequemlichkeit in Telegram geführt. Diese hätten aus Gründen der Dokumentation und Rückverfolgbarkeit im jeweiligen Diskussionsbereich des Issues geführt werden müssen. Des Weiteren gab es Themen, welche besser während der Präsenzzeit der Termine hätten besprochen werden sollen, als über den Telegram-Chat.

Mail-Verteiler

Durch den Einsatz des Mail-Verteilers ergaben sich Komplikationen. Durch die Verwendung der „@informatik.uni-oldenburg.de“-Subdomain wurde der Einrichtungs- und Verwaltungsaufwand an die ARBI ausgelagert. Da die Adressen der Stakeholder jedoch des Öfteren geändert werden mussten, wurde die Auslagerung an einen Dienstleister, zu einem zuvor nicht kalkulierten Kommunikationsaufwand und Risiko. Schnelle Änderungen waren nicht möglich und haben den Kommunikationsweg beeinträchtigt. Ein kostenloser Mail-Verteiler kann mit übersichtlichem Aufwand bei Diensten wie Google oder Yahoo erstellt werden. Schließlich hat sich die Projektgruppe für die Maßnahme entschieden die bisherige Adresse durch einen „@googlegroups.com“-Verteiler zu ersetzen. Der „@googlegroups.com“-Verteiler kann direkt durch die Projektgruppenteilnehmer administriert werden.

Im ersten Zyklus des Projektes ist aufgefallen, dass zu wenig über den Verteiler kommuniziert wurde. Viele Angelegenheiten wurden am Verteiler vorbei zwischen einzelnen Teilnehmern und Stakeholdern kommuniziert. Dies führte dazu, dass wichtige Informationen nicht für alle Teilnehmer zugänglich waren. Somit wurden aufgrund fehlender Informationen schlechte oder gar falsche Entscheidungen getroffen.

Als Maßnahme wurde durch Peter Tank angeregt, dass der Mail-Verteiler, bei jeder Kommunikation mit den Stakeholdern, in „CC“ genommen werden sollte. Dieser Vorschlag gefiel der Projektgruppe sehr gut, sodass nun ein reger Austausch über den Verteiler stattgefunden hat und alle Informationen an jeden Stakeholder übermittelt wurden. Als Nachteil kann eingeordnet werden, dass jeder Teilnehmer auch Informationen bekam, die er nicht benötigte. Andererseits entstanden an vielen Stellen konstruktive Anmerkungen von Stakeholdern, welche nicht direkt angesprochen wurden, sodass ein ungeplanter Mehrwert generiert wurde.

GanttProject

Im Nachhinein betrachtet hat Ganttproject seinen Zweck nicht ausreichend erfüllt. Die Diagramme sollten bei den internen und externen Treffen (siehe Absatz 2.3.2) den Projektfortschritt darstellen. Jedoch erzeugte das Programm sehr unübersichtliche Diagramme, deren Inhalte der Übersicht nicht dienlich waren. Aus Gründen der Transparenz (siehe Absatz 9.2.5) hat sich die Projektgruppe gegen die weitere Verwendung der Gantt-Diagramme und GanttProject entschieden.

Als Resultat dieser Maßnahme wurde der kritische Pfad aus GanttProject in die kritischen Aufgaben der Statusberichte (siehe Absatz 2.3.3) übertragen.

Slack

Slack wurde nach der Einführungen von einigen Teilnehmern der Projektgruppe genutzt. Anderen Teilnehmern der Projektgruppe war der Mehrwert von Slack nicht klar und somit haben diese das Tool nicht verwendet. Daraus resultierte, dass die Slack-Funktionen nicht genutzt werden konnten. Slack hatte somit keinen nennenswerten Mehrwert für die Projektgruppe.

9.2.5 Transparenz des Projektstatus

Der Projektstatus sollte gegenüber den Stakeholdern und Projektgruppenmitgliedern stets transparent gehalten werden. Dadurch konnten sie jederzeit den Projektfortschritt beobachten. Eventuelle Unstimmigkeiten sollten dabei erkannt und wenn möglich schnell gelöst werden. Im Folgenden werden mehrere Artefakte (siehe Abschnitt 2.3) evaluiert, welche die Transparenz fördern.

Kanban-Board

Innerhalb von GitLab gibt es die Möglichkeit die einzelnen Issues über ein Kanban-Board darzustellen. In Issues können zum Beispiel Anforderungen, Bugs oder konkrete Implementierungsaufgaben beschrieben werden. Für den Fortschritt eines Issues wurden Labels erstellt, welche den Stand des Issues darstellen. Als Beispiele seien folgende Labels genannt:

- Specifying

- Doing
- Testing
- Closed

Über diese konnten die Issues kategorisiert und der Fortschritt erkannt werden. Jedes Issue wurde je nach Fortschritt mit einem Label versehen. Dies war für jeden Stakeholder offen in GitLab einsehbar.

Im Kanban-Board können Spalten zu den Labels angefertigt werden. Dabei zeigt das Kanban-Board die Issues in der jeweiligen Spalten an. Damit konnte der Projektstatus innerhalb eines Teilprojektes leicht eingesehen werden.

Das Kanban-Board ist zu Anfang des Projekts nicht ausreichend gepflegt worden. Issues sind erst zu einem späteren Zeitpunkt im Projekt regelmäßig mit Labels versehen worden. Daher war am Anfang des Projekts keine genaue Aussage des Projektstandes zu sehen. Des Weiteren kam das Problem auf, dass die Gruppenmitglieder zu Beginn der Projektgruppe in der Einarbeitszeit noch nicht mit den Funktionalitäten der Labels, Issues und des Kanban-Boards vertraut waren. Dies führte dazu, dass zwei Teilnehmer parallel an einem Issue gearbeitet haben, ohne das eine Zuweisung des Issues stattgefunden hat. Als Konsequenz dieser Probleme wurden die Labels und Issues besser gepflegt und die Projektgruppe erkannte den Mehrwert der GitLab-Funktionen und einer sauberen Vorgehensweise.

Das Kanban-Board war ein gutes Mittel für die Transparenz des Projektfortschritts, auch wenn es anfänglich nicht von allen Teilnehmern ausreichend genutzt wurde.

Weekly-Update

Beim Weekly-Update konnte die Gruppe gezielt auf Aufgaben eingehen und Feedback geben. Außerdem konnten andere Projektgruppenmitglieder eventuelle Hilfestellungen und Anregungen über aktuelle und kommende Aufgaben geben. Insgesamt zeigte sich das Weekly-Update als eine gute Methode, um jedes Projektgruppenmitglied und die Stakeholder über den derzeitigen Projektstand in Kenntnis zu setzen. Außerdem konnten Aufgaben mit potenziellen Konflikten schnell erkannt werden.

Statusberichte

Die Statusberichte sind als Maßnahme nach der ersten Retrospektive eingeführt worden, um die Transparenz des Fortschritts der Teilgruppen zu erhöhen. Vorher wurde der Projektfortschritt lediglich mithilfe des Weekly-Updates überwacht. Ein Statusbericht fasst die einzelnen Fortschritte des Teilprojektes der vorherigen Woche zusammen. Alle Statusberichte sind innerhalb der Gruppe besprochen worden, um Probleme aufzuzeigen und gemeinsam Lösungsvorschläge zu sammeln. Die Statusberichte sind wöchentlich zweimal besprochen worden, im Detail beim internen und als Zusammenfassung beim externen Treffen. Durch die Statusberichte konnte die Transparenz des aktuellen Status der Teilgruppen erhöht werden.

Projektrisiken

Des Weiteren hat sich die Projektgruppe dafür entschieden ein Dokument zu pflegen, welches die Projektrisiken dokumentiert und greifbar macht. Die Projektrisiken sind bei jedem internen und externen Treffen angesprochen worden, um diese stetig präsent zu halten.

Dabei sind externe und interne Projektrisiken gleichermaßen berücksichtigt worden. Allerdings sind einige Projektrisiken zu lange nicht von der Liste abgehakt worden, obwohl diese gelöst wurden. Außerdem haben sich durch die schlechte Formulierung einiger Projektrisiken einzelne Personen angegriffen gefühlt. Im Rückblick ist zu sagen, dass sich die regelmäßige Betrachtung um eine transparente Lösung für Risiken handelt. Jedoch sollte die Formulierung gut gewählt sein, da es sich bei Risiken ohnehin um einen negativ behafteten Text handelt.

Gantt-Diagramm

Das Gantt-Diagramm ist in der Regel eine gute Methode, um die Übersicht über die zu erledigenden Aufgaben und den zugeordneten „Human Resources“ zu behalten. Es sollte in den Terminen der Veranschaulichung des Projektfortschritts dienen. Jede Woche mussten ungewöhnlich viele Veränderungen vorgenommen werden, welche einen immensen Aufwand für die Aktualisierung des Diagramms erzeugten. Viele Projektrisiken sorgten dafür, dass eine detaillierte Planung schlichtweg nicht möglich gewesen ist, bzw. einen nicht vertretbar hohen Wartungsaufwand zur Folge hatte. Somit hat sich die Gruppe dafür

entschieden eine gröbere Projektplanung vorzunehmen, um den Verwaltungsaufwand zu reduzieren. Eine grobe Projektplanung, wie sie vonnöten, war unterstützte die Methode des Gantt-Diagramms jedoch nicht. Somit wurde das Gantt-Diagramm durch einen ausführlicheren Statusbericht ersetzt.

10 Zusammenfassung und Ausblick

10.1 Zusammenfassung und Zielerreichung

Ausgehend von der zentralen Fragestellung der Projektgruppe wurde die Grundlage für ein autonom fahrendes Schiff entwickelt und getestet.

Das Ergebnis ist ein Produkt, das dazu in der Lage ist, ein Schiff fernzusteuern und in weiten Teilen das automatische Abfahren von Routen unterstützt. Das Produkt verfügt über eine komplexe Systemarchitektur mit küsten- und schiffsseitigen Komponenten.

Das Produkt verfügt über eine Kontrollplattform, die auf der EPD basiert. Diese ermöglicht unter anderem die Darstellung verschiedener Sensordaten sowie des eigenen Schiffs. Außerdem ermöglicht die Kontrollplattform die Eingabe von Steuerbefehlen für die Fernsteuerung des Schiffs sowie die Eingabe und Speicherung von Routen. Von der Kontrollplattform kann zudem der Autopilot für das automatische Abfahren von Routen aktiviert und überwacht werden.

Ein Regelungssystem schafft die Voraussetzung zur Berechnung von Steuerbefehlen basierend auf Schiffsist- und Schiffssollwerten unter Berücksichtigung eines dynamischen Schiffsmodells.

Die Pfadplanungskomponente ermöglicht als erste Ausbaustufe der Komponenten zum autonomen Fahren die Ermittlung der aktuellen Wegpunkte einer Route, basierend auf der Route und der aktuellen Position des Schiffs.

Eine Notfallsteuerung ermöglicht es in einer Gefahrensituation durch Eingaben in ein Gamepad die Steuerung über das Schiff zu übernehmen. Diese Komponente soll unabhängig von den restlichen Komponenten des Produkts sein. Die Steuerbefehle der Notfallsteuerung gehen daher direkt an die Schiffssteuerung.

Eine komplexe Verteilungsplattform bewerkstelligt den Nachrichtenaustausch zwischen allen anderen Komponenten. Die Verteilungsplattform kann über diverse Adapter und Translator verschiedene Nachrichtentypen von unterschiedlichen Quellen aufnehmen und diese mithilfe von XSLT in andere Nachrichtentypen umwandeln. Die transformierten Nachrichten können wiederum an verschiedene Nachrichtenempfänger übergeben werden. Zu den unterstützten Nachrichtentypen zählen NMEA0183, NMEA2000, S-100 und RTZ. Die Verteilungsplattform fungiert zudem als zentrale Schnittstelle zum Schiff bzw. zur Außenwelt allgemein.

Die Plattform wurde in eine vorhandene Systemarchitektur eingebettet. Diese besteht unter anderem aus dem Forschungsschiff Zuse und dem virtuellen Testbed LABSKAUS im Rahmen der eMIR-Initiative

Das Produkt soll im Rahmen weiterer Forschungsprojekte im maritimen Bereich an der Universität Oldenburg und im OFFIS genutzt werden. Die Entwicklung und die Tests fanden mithilfe des Forschungsschiffs Zuse statt. Aufgrund der NMEA2000-Schnittstelle könnte die Plattform jedoch auch auf anderen Schiffen eingesetzt werden.

Geplant war die Schaffung eines autonom fahrenden Schiffs in drei aufeinander aufbauenden Schritten. Der erste Schritt befasst sich mit der Fernsteuerung eines Schiffs. Diese wurde erfolgreich umgesetzt. Mit Hilfe der sogenannten „Virtual Handles“ der Kontrollplattform kann ein Schiff sowohl vom Schiff als auch von Land ferngesteuert werden.

Der zweite Schritt sah das automatische Abfahren einer vorgegebenen Route durch das Schiff vor. Dieser wurde weitestgehend umgesetzt. Aufgrund eines Fehlers in der Regelungssystemkomponente konnten in einem abschließenden Feldtest nicht alle Anforderungen vollständig getestet werden. Die Regelungssystemkomponente sollte basierend auf den übergebenen aktuellen Wegpunkten die Steuerbefehle für das Schiff ermitteln, damit diese Wegpunkte vom Schiff abgefahren werden können. Die Regelungssystemkomponente konnte diese Steuerbefehle nicht korrekt liefern. Die übrigen am zweiten Schritt beteiligten Komponenten wurden in einem Trockentest erfolgreich getestet.

Der dritte und letzte Schritt hatte das vollständig autonom fahrende Schiff zum Ziel, das dynamisch auf diverse Umwelteinflüsse, wie Hindernisse im Wasser oder andere Schiffe auf Kollisionskurs, reagieren kann. Dieser Schritt wurde nicht umgesetzt.

Das Ergebnis der PG MATE soll von anderen Projektgruppen im maritimen Bereich bzw. von möglichen Folgeprojektgruppen genutzt und weiterentwickelt werden. Deshalb wurde auf eine umfassende Dokumentation der Entwicklung und der Ergebnisse geachtet. Womit sich eine Folgeprojektgruppe in diesem Kontext beschäftigen kann, wird im folgenden Abschnitt beschrieben.

10.2 Ausblick und Fazit

Für eine Folgeprojektgruppe ist zunächst der Abschluss des von den Projektgruppenbetreuern vorgesehenen zweiten Schrittes, das automatische Abfahren von Routen durch das Schiff, anzustreben. Dafür ist eine detaillierte Beschäftigung mit dieser Abschlussdokumentation notwendig. Aufbauend darauf kann mit der Fertigstellung der Regelungssys-

temkomponente begonnen werden. Sind die Arbeiten an dieser abgeschlossen, können weitere Feldtests mit der Zuse durchgeführt werden. Außerdem kann in diesem Zusammenhang die Notfallsteuerung dahingehend erweitert werden, dass diese auch von Land aus einsetzbar ist.

Im Anschluss kann mit der Bearbeitung des dritten Schritts, dem vollständig autonom fahrenden Schiff, begonnen werden. In diesem Kontext muss das Produkt dynamisch auf eine sich verändernde Umwelt reagieren und die Fahrweise des Schiffs an diese anpassen. Dafür ist zum Beispiel die Berücksichtigung der Verkehrsregeln im Rahmen der Seeschifffahrtsstraßenordnung notwendig. Außerdem muss das Produkt Objekte erkennen und auf diese reagieren können. Mit dieser Problemstellung haben sich bereits die Projektgruppen MOPS I - IV beschäftigt. Daher ist eine Einsicht in die Ergebnisse dieser Projektgruppen zu empfehlen.

Über die Fragestellung dieser Projektgruppe hinaus haben sich einige weitere interessante Themengebiete ergeben, deren Untersuchung in Betracht gezogen werden kann. Wie verhält sich das Produkt beispielsweise auf anderen Schiffen und welche Sensoren müssen mindestens auf einem Schiff vorhanden sein, damit es mit Hilfe des Produkts autonom fahren kann.

Abschließend lässt sich sagen, dass die Schaffung eines autonom fahrenden Schiffes ein komplexes und umfangreiches, aber auch sehr interessantes Projekt ist. Ähnlich wie im Automobilsektor haben die Entwicklungen in der Schifffahrt großes Potenzial im Hinblick auf die Verkehrssicherheit sowie ökologische und ökonomische Vorteile. Auch wenn am Ende der PG MATE kein vollständig autonom fahrendes Schiff steht, wurde eine solide Grundlage für weitere Forschungstätigkeiten in diesem Bereich an der Universität Oldenburg und am OFFIS geschaffen.

A Anhang

A.1 Handbücher

A.1.1 Notfallsteuerung

Backup control

The backup control allows you to control the vessel Zuse.
The engine speed and the rudder angle can be controlled via a gamepad.
The control commands of the backup control overwrite all control commands from other software.
The backup control sends the control commands over the internet in the AMQP protocol
or via UDP packets to the NaviBox on the Zuse.

Build

Maven 3 and Java 8 are required to build the backup control.
The software is packed in a jar file.
The software can only be run on the platform on which it was built (Windows, Linux or Mac OS).
The dependency to the GLFW library is responsible for this platform dependency.

```
1 git clone git@gitlab.uni-oldenburg.de:PG-MATE/Backup-Steuerung.git
2 cd Backup-Steuerung
3 mvn clean install
```

The following section describes how to use the backup control.
Please read the documentation before using the backup control.

Usage

Java 8 is required to use the backup control.
To start the backup control, start the generated jar file:

```
1 java -jar target/backup_control-1.1.0.jar
```

After the backup control has been started, the connection to the Zuse and the gamepad must be configured.
The following sections describe the configuration.

Connection configuration

When the backup control is started, the user is directly prompted to configure the connection to the Zuse.
Two types of communication are available. The user can connect to the vessel either via RabbitMQ or via UDP.

RabbitMQ

To connect via RabbitMQ, the following properties must be set.

Name	Type	Description
Host	InetAddress	The host name of the AMQP server to which the control commands are to be sent.
Port	Integer	The port number of the AMQP server.
Username	String	The name of the user for the AMQP server.
Password	String	The password for the user.
Exchange name	String	The name of the AMQP exchange to which the control commands are to be sent.
Exchange type	String	The type of the AMQP exchange (for example: direct).

1

Name	Type	Description
Exchange routing key	String	The routing key for the target queue.
Durable	Boolean	Exchanges can be durable (true) or transient (false).
Big Endian	Boolean	The byte order with which the control commands are to be generated. Big-endian (true) or little-endian (false).

UDP

To connect via UDP, the following properties must be set.

Name	Type	Description
Host	InetAddress	The host name of the UDP server to which the control commands are to be sent.
Port	Integer	The port number of the UDP server.
Little Endian	Boolean	The byte order with which the control commands are to be generated. Little-endian (true) or big-endian (false).

Gamepad configuration

Once the connection has been configured, the user must select the gamepad that he will use to control the Zuse.
Theoretically all gamepads supported by the GLFW library can be used.
But only the following gamepads were tested with the backup control and are officially supported:

- XBox 360 Controller

For other gamepads the backup control may need to be adjusted.
If you have successfully tested another gamepad, please add the name of the gamepad to this list.

Control of the vessel

After a gamepad has been selected, no control commands are sent yet.
To start sending control commands, press the start button on the gamepad.
To cancel sending control commands, press the back button on the gamepad.
If the sending of control commands is activated, the Zuse can be controlled with the gamepad.

The Zuse is controlled by the right analog stick.
The engine spped is set via the vertical y axis.
The engine speed can be adjusted between -2000 and 2000 rpm.
A negative rpm value activates the reverse.
The rudder angle is set via the horizontal x axis.
The angle can be adjusted between -20 and 20 degrees.

URL zum Repository: https://gitlab.uni-oldenburg.de/PG-MATE/Backup-Steuerung

A.1.2 Regelungssystem

Controller

This repository contains the Controller implementation of the MATE project.
The Controller does some calculations to determine the best trajectory between two waypoints and some calculations in a discret interval to stay on the choosen path.
To get those calculations done the Controller needs an UDP-Connection Modul for communication with external systems.
To move a ship the Controller can send manual commands by bypassing received RPM and RudderAngle commands or can calculate those values by itself based on Sensor data and given waypoints.

Prerequisites

The Controller runs with *MATLAB Real-Time Simulink* but has some C++ implementations imported, too. Therefore, the following prerequisites has to be met:

1. Download and install MATLAB. You may have to login or provide an Activation Key. If you are a member of the **Carl of Ossietzky University Oldenburg** you can use this Installation Guide, if not just follow the installtion instructions of MATLAB. Daemon Tools can be used to mount the .iso. MATLAB for Linux does not contain the *Simulink Real-Time* and *Simulink Desktop Real-Time* features. It should therefore not be used for this project and further notes and steps will omit Linux OS. The Installtion may take a while. After the installtion process succeeded just follow the activation instructions.
2. Download and install Visual Studio C++. Beware of the ca. 12 GB needed for that operation. This may take a while to be done.
3. .NET framework for creation of .NET assemblies and deployable archives with Excel integration. You can select this package as part of the *Visual Studio C++* installation.
4. A supported compiler for creation of COM components, C and C++ Shared libraries. If you installed *Visual Studio C++* you have one already. Another may be mingw.
5. A Java JDK for creation of Java packages.
6. Download and install git or any other git-Tool you prefer.

Installation

At the beginning clone the Controller repository:

```
1 git clone https://gitlab.uni-oldenburg.de/PG-MATE/Controller.git
2 cd Controller
```

Now you can import the Controller project into MATLAB. This can be done with the following steps

1. Start MATLAB
2. Press `Open` and select the `Controller.prj` file inside the `Controller` subproject of the Controller repository.
3. If MATLAB does not switch itself, you can switch to the newly Simulink project within the main tab menu at the top of the MATLAB interface.
4. For compilation and runnable notes see the next topic.
5. Press `New` in the navigation bar and either select `Simulink Model` or `Simulink Project` to create new components for the Controller.

Usage of Simulink Realtime

- You find this Simulink Real-Time guide usefull to setup and configure Simulink Real-Time:
- Use `slrtexplr` to start Simulink Real-Time Explorer to edit configuration
- If you have *Visual Studio C++* installed, run:
- `mex -setup` which sets up C compilers
- `mex -setup cpp` which sets up C++ compilers
- Open controller model and build it:
- Because the Controller is a real time application further developers should start with this Simulink Real-Time Application Guide to learn how Simulink works and how a Simulink Model can be build and transformed into a Real-Time Application.

1

- The model can now be run on the target machine

Controller usage

The Controller uses an own internal data representation to encapsulate the NMEA2000 messages into a more lightly structure shortened to the very needs of the Controller.
The connection to and from the Controller is always UDP with NMEA2000 byte packages. Thus you need some kind of a NMEA2000 parsing module when connecting to the Controller to ensure mutual understanding.
The Controller only sends the `Target RPM` and `Target rudder angle` comands using the PGNs displays below.

Definition of the internal data format

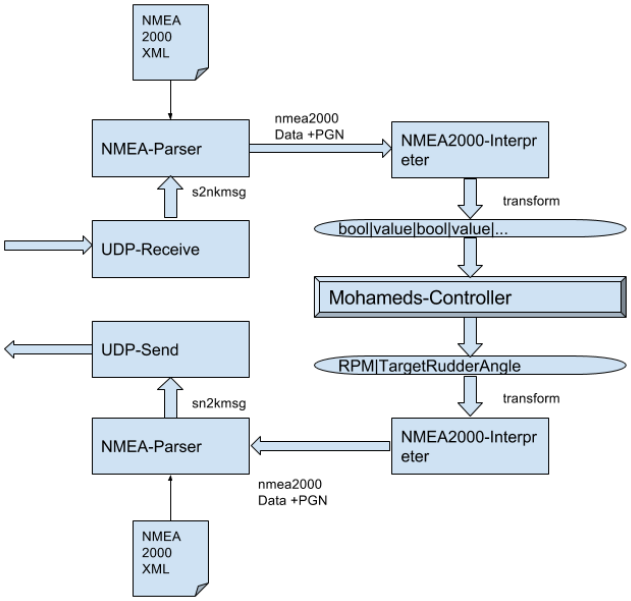
This paragraph will define how the internal data format will look like (See also Interface definition).
The internal representation of the data referes to the NMEA2000-Standard. Thus, there is a related PGN given for each entry within our data model.
The bold PGNs are the ones currently parsed and interpreted at the Controller.

Internal data encapsulation of received sensor data:

Data	Data type	ByteLength	Related PGN(s)
Target RPM	'double'	8	(**1**), (127488), (127498)
Target rudder angle	'double'	8	(**127245**), (127237)
Wind speed	'double'	8	(**130306**), (130323), (130324)
Wind direction	'double'	8	(**130306**), (130323), (130324)
Wind reference	'uint8'	1	(**130306**), (130323)
Latitude	'double'	8	(**129025**), (129027), (**129029**), AIS: (129038), (129039), (129040), (129041), (129044), (129284), (129545), (129792), (129793), (129798), etc.
Longitude	'double'	8	(**129025**), (129027), (**129029**), AIS: (129038), (129039), (129040), (129041), (129044), (129284), (129545), (129792), (129793), (129798), etc.
Speed over water (sow)	'double'	8	(**128259**),
Speed over ground (sog)	'double'	8	(127233), (**129026**), (129813),
RPM	'double'	8	(**127488**), (**127498**)
Heading	'double'	8	(**127250**) , , AIS: . . .
Course angle	'double'	8	(**129026**)„ AIS: . . .
Route Status	'uint8'	1	(**130066**)
WPID 1	'uint16'	2	(**130067**)
WP 1 Latitude	'double'	8	(**130067**)
WP 1 Longitude	'double'	8	(**130067**)
WPID 2	'uint16'	2	(**130067**)
WP 2 Latitude	'double'	8	(**130067**)
WP 2 Longitude	'double'	8	(**130067**)
RudderAnglePosition	'double'	8	(**127245**), (127237)

2

FlowDiagram



Developer notes

If you want to further develop the `NMEA2000_MessageCreator` you may be interested in the steps to recompile the S-Function.
From within the Matlab Simulink command terminal type:

```
cd models\test\NMEA2000Messages\
mex -v -IByteReader NMEA2000_MessageCreator.cpp NMEA2000_MessageCreator_Wrapper.cpp NMEA2000Helper.
    cpp ByteReader\ByteReader.cpp
```

Testing

The tests need to run manually, meaning running in `normal` mode and each test has to be executed alone.
The project structure (inside the `models` dir) is ordered by:

- integrationtests: Containing one integration test for the CoreController modelwith UDP connections. You need to setup the `Polymorphic Interface` to run this integration test. TODO installation guide reference to the poly
- tests: Containing some kind of unit tests for Simlink models. The S-Function code is acutally much wider tested within the cpp implementation. Contains test helper models which are also tested, too. At the end of one of those tests the boolean `display` displays whether the test succeeded (display: 1) or failed (display 0).

How to run

Start one of the `Controller_*.slx` (WithEndianess if you are on a Little-Endian machine). Run the model in `normal` mode with `inf` time set. Start one of the `Send*.slx` example models within the `test` folder and see the displays within the model or fetch the returning UDP commands for RPM and RudderAngle to see the result.

Receiving data

The Controller Receive Block needs to be configured with an IP-Address. If you want to test it local set it to 127.0.0.1.
The Polymorphic interface can be used to send NMEA2000 messages to the Controller.
With the display blocks the received values can be shown.
There also is a monoitoring Simulink model to display outgoing NMEA2000 messages after they have either forwarded or calculated.
The UDP Send blocks have to set to broadcast for that purpose.

4

URL zum Repository: https://gitlab.uni-oldenburg.de/PG-MATE/Controller

A.1.3 Verteilungsplattform / Polymorphe Schnittstelle

Polymorphic Interface

Transforms data formats into other formats using XML, XSD and XSLT.
Various data sources are supported by different adapters.
Several translators generate XML from different protocols.
Maven 3 and Java 8 are required to build the Polymorphic Interface.

Server

Main classes of the Polymorphic Interface that are responsible for executing the transformations.
While starting the configuration, connectors and models are loaded by the server.
The Server-Api can be used to write custom adapters and translators.

- Server
- Server-Api

Adapters

Adapters enable the Polymorphic Interface to interact with various data sources and sinks.

- File Adapters
- Http Adapters
- RabbitMQ Adapters
- UDP Adapters

Translators

Translators are used to convert various data formats to XML.

- Base64 Translators
- NMEA0183 Translators
- NMEA2000 Translators
- NMEA2000 Zuse Commands Translators
- S-100 Translators
- XML String Translators

Tools

Collection of tools that are not directly related to the Polymorphic Interface.

- NaviBox Simulator
- S-100 Sample Sender
- S-100 Xml Analyser

Experiments

Collection of projects that were created while developing the Polymorphic Interface.

- Server Example Instance
- Ship XSLT Example
- XSLT Example

1

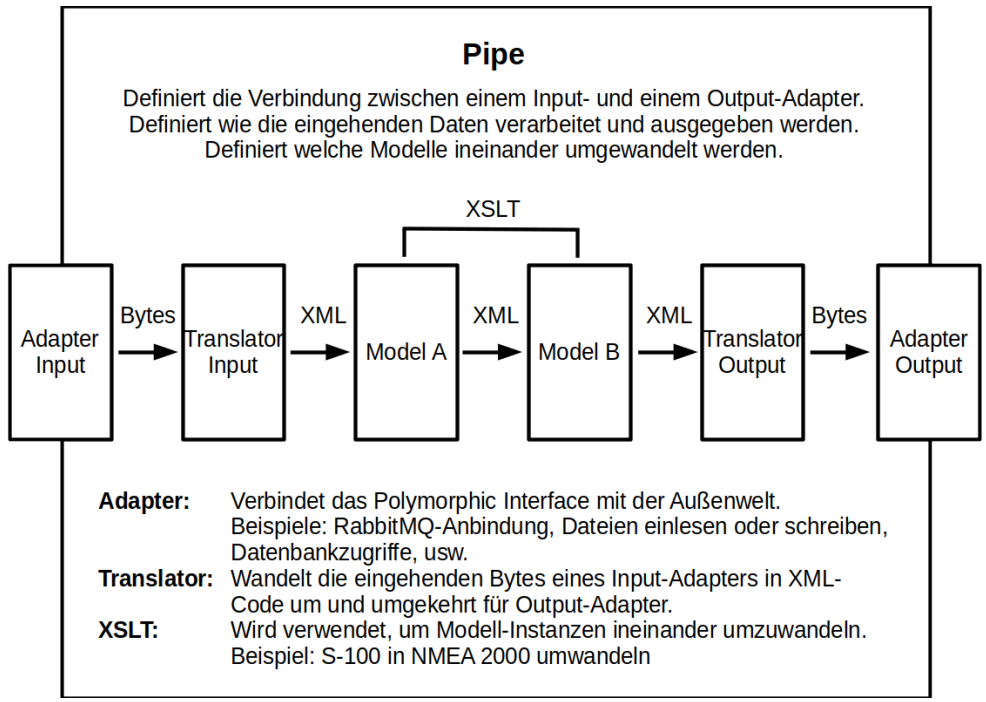
Useful Information

- Requirements
- eMir Library

Class Diagrams

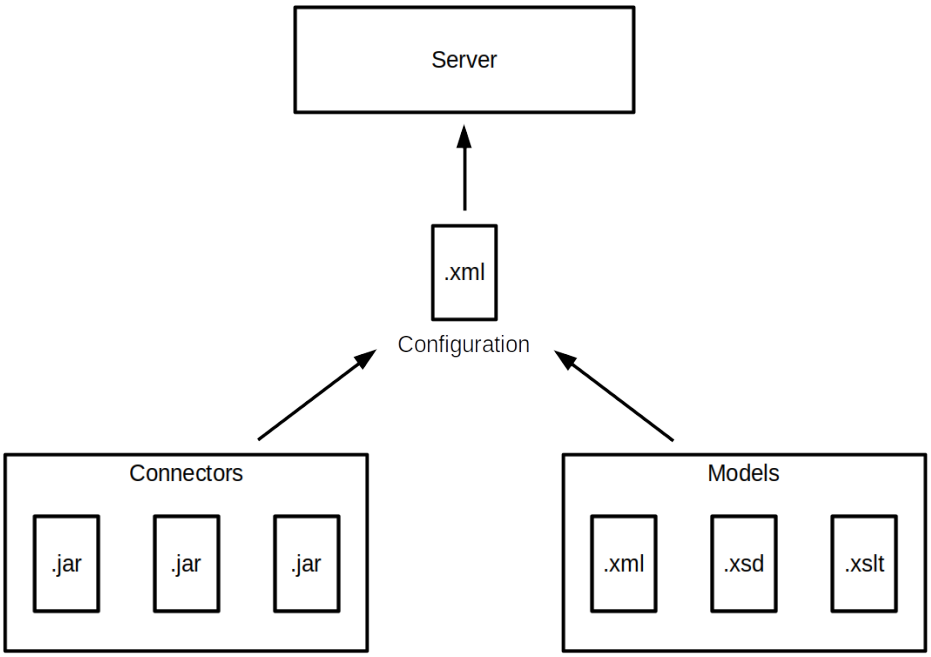
- Server-API
- Adapter
- Server
- Configuration
- Connectors
- Models
- Pipes

Pipes



2

Configuration



Example Configuration

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration
    xmlns="http://polymorphic-interface.mate.uni-oldenburg.de/configuration"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://polymorphic-interface.mate.uni-oldenburg.de/configuration
        /de/uni_oldenburg/mate/polymorphic_interface/server/configuration/configuration.xsd">
    <models directory=".">
        <definitions>
            <definition id="LISE" schemaFile="../schema/mate/position.xsd" />
            <definition id="S-100" schemaFile="../schema/S100/s100.xsd" />
            <definition id="base64" schemaFile="models/base64-wrapper.xsd" />
        </definitions>
        <transformations>
            <transformation source="S-100" target="LISE" xsltFile="../schema/mate/s100-to-position.
                xsl" />
        </transformations>
    </models>
    <connectors>
        <resources directory="./connectors">
            <jar file="rabbitmq_adapters-1.0.1.jar" module="de.uni_oldenburg.mate.
                polymorphic_interface.rabbitmq_adapters.Module" />
```

3

```
18            <jar file="file_adapters-1.0.jar" module="de.uni_oldenburg.mate.polymorphic_interface.
                  file_adapters.Module" />
19            <jar file="xml_string_translators-1.1.jar" module="de.uni_oldenburg.mate.
                  polymorphic_interface.xml_string_translators.Module" />
20            <jar file="lise_adapters-1.1.jar" module="de.uni_oldenburg.mate.polymorphic_interface.
                  lise_adapters.Module" />
21            <jar file="base64_translators-1.0.jar" module="de.uni_oldenburg.mate.
                  polymorphic_interface.base64_translators.Module" />
22            <jar file="s100translator-2.0.jar" module="de.uni_oldenburg.mate.polymorphic_interface.
                  s100translator.Module" />
23            <jar file="udp_adapters-1.0.jar" module="de.uni_oldenburg.mate.polymorphic_interface.
                  udp_adapters.Module" />
24        </resources>
25        <instances>
26            <adapters>
27                <input id="File-Input-Adapter" class="de.uni_oldenburg.mate.polymorphic_interface.
                      file_adapters.FileInputAdapter">
28                    <property key="input_directory" value="./tmp/input" />
29                </input>
30                <input id="Rabbitmq-Input-Adapter-Simulation" class="de.uni_oldenburg.mate.
                      polymorphic_interface.rabbitmq_adapters.RabbitMQInputAdapter">
31                    <property key="username" value="guest" />
32                    <property key="password" value="helloworld" />
33                    <property key="host" value="localhost" />
34                    <property key="port" value="5672" />
35                    <property key="queue_name" value="PG-System-Test" />
36                    <property key="args_durability" value="100" />
37                    <property key="durable" value="false" />
38                </input>
39                <input id="Rabbitmq-Input-Adapter-NaviBox" class="de.uni_oldenburg.mate.
                      polymorphic_interface.rabbitmq_adapters.RabbitMQInputAdapter">
40                    <property key="username" value="PGMate" />
41                    <property key="password" value="mate" />
42                    <property key="host" value="wilkinson.informatik.uni-oldenburg.de" />
43                    <property key="port" value="5672" />
44                    <property key="queue_name" value="PGMate" />
45                    <property key="args_durability" value="100" />
46                    <property key="durable" value="true" />
47                </input>
48                <input id="Udp-Input-Adapter" class="de.uni_oldenburg.mate.polymorphic_interface.
                      udp_adapters.UdpInputAdapter">
49                    <property key="port" value="13377" />
50                    <property key="buffer_size" value="2048" />
51                </input>
52                <output id="Xml-File-Output-Adapter" class="de.uni_oldenburg.mate.
                      polymorphic_interface.file_adapters.FileOutputAdapter">
53                    <property key="output_directory" value="./tmp/output/xml" />
54                    <property key="output_file_suffix" value=".xml" />
55                </output>
56                <output id="Byte-File-Output-Adapter" class="de.uni_oldenburg.mate.
                      polymorphic_interface.file_adapters.FileOutputAdapter">
57                    <property key="output_directory" value="./tmp/output/byte" />
58                    <property key="output_file_prefix" value="unchanged_input_" />
59                </output>
60                <output id="Lise-Output-Adapter" class="de.uni_oldenburg.mate.polymorphic_interface.
                      lise_adapters.LiseOutputAdapter">
61                    <property key="url" value="http://duemmer.informatik.uni-oldenburg.
                          de:1534/chat/"/>
```

```
62                      <property key="vessel_name" value="Zuse"/>
63                  </output>
64                  <output id="Rabbitmq-Output-Adapter-NaviBox" class="de.uni_oldenburg.mate.
                        polymorphic_interface.rabbitmq_adapters.RabbitMQOutputAdapter">
65                      <property key="username" value="PGMate" />
66                      <property key="password" value="mate" />
67                      <property key="host" value="wilkinson.informatik.uni-oldenburg.de" />
68                      <property key="port" value="5672" />
69                      <property key="exchange_name" value="ServiceBusData" />
70                      <property key="exchange_type" value="direct" />
71                      <property key="exchange_routing_key" value="Zuse" />
72                  </output>
73              </adapters>
74              <translators>
75                  <input id="Bytes-to-XML" class="de.uni_oldenburg.mate.polymorphic_interface.
                        xml_string_translators.XmlStringInputTranslator">
76                      <property key="charset" value="UTF-8" />
77                  </input>
78                  <input id="Bytes-to-Base64" class="de.uni_oldenburg.mate.polymorphic_interface.
                        base64_translators.BytesToBase64InputTranslator" />
79                  <input id="Bytes-to-S-100" class="de.uni_oldenburg.mate.polymorphic_interface.
                        s100translator.S100InputTranslator">
80                      <property key="ERROR_DIR" value="tmp/errors/" />
81                  </input>
82                  <output id="XML-to-Bytes" class="de.uni_oldenburg.mate.polymorphic_interface.
                        xml_string_translators.XmlStringOutputTranslator">
83                      <property key="charset" value="UTF-8" />
84                  </output>
85                  <output id="Base64-to-Bytes" class="de.uni_oldenburg.mate.polymorphic_interface.
                        base64_translators.Base64ToBytesOutputTranslator" />
86                  <output id="S-100-to-Bytes" class="de.uni_oldenburg.mate.polymorphic_interface.
                        s100translator.S100OutputTranslator" />
87              </translators>
88          </instances>
89      </connectors>
90      <pipes>
91          <pipe id="NaviBox-to-Xml-File">
92              <adapters>
93                  <input id="Rabbitmq-Input-Adapter-NaviBox" />
94                  <output id="Xml-File-Output-Adapter" />
95              </adapters>
96              <translators>
97                  <input id="Bytes-to-S-100" />
98                  <output id="XML-to-Bytes" />
99              </translators>
100             <steps>
101                 <step model="S-100" />
102             </steps>
103         </pipe>
104         <pipe id="NaviBox-to-Byte-File">
105             <adapters>
106                 <input id="Rabbitmq-Input-Adapter-NaviBox" />
107                 <output id="Byte-File-Output-Adapter" />
108             </adapters>
109             <translators>
110                 <input id="Bytes-to-Base64" />
111                 <output id="Base64-to-Bytes" />
112             </translators>
```

5

```
113             <steps>
114                 <step model="base64" />
115             </steps>
116         </pipe>
117         <pipe id="Udp-to-NaviBox">
118             <adapters>
119                 <input id="Udp-Input-Adapter" />
120                 <output id="Rabbitmq-Output-Adapter-NaviBox" />
121             </adapters>
122             <translators>
123                 <input id="Bytes-to-XML" />
124                 <output id="S-100-to-Bytes" />
125             </translators>
126             <steps>
127                 <step model="S-100" />
128             </steps>
129         </pipe>
130     </pipes>
131 </configuration>
```

6

URL zum Repository: https://gitlab.uni-oldenburg.de/PG-MATE/Polymorphic-Interface bzw. https://gitlab.uni-oldenburg.de/PG-MATE/Polymorphic-Interface-Instance

Adapter

File Adapters

This project includes two adapters that can be used to process data from
files through the server and to write data to files through the server.

FileOutputAdapter

```
1 de.uni_oldenburg.mate.polymorphic_interface.file_adapters.FileOutputAdapter
```

This OutputAdapter writes the passed data to files.
Each call to the output method creates a new file.

The files are numbered in ascending order, starting at 0.
If a file with this name already exists, the respective number is skipped.

The behavior of this Adapter can be set using the following properties:

Key	Description	Example
output_directory	The path to the directory where the files are created.	/tmp/output
output_file_prefix	Optional: A prefix for the names of the files that are created.	data_
output_file_suffix	Optional: A suffix for the names of the files that are created.	.csv

FileInputAdapter

```
1 de.uni_oldenburg.mate.polymorphic_interface.file_adapters.FileInputAdapter
```

This InputAdapter monitors a directory for new files and sends the file contents to the server.

The behavior of this Adapter can be set using the following properties:

Key	Description	Example
input_directory	The path to the directory that is being monitored for new files.	/tmp/input

1

HTTP Adapters

This project includes two adapters that can be used to send and receive data via the Hypertext Transfer Protocol (HTTP).

The Adapters will be used in a pipe of the Polymorphic Interface to communicate with the Path Planning.

HttpOutputAdapter

```
de.uni_oldenburg.mate.polymorphic_interface.http_adapters.HttpOutputAdapter
```

This OutputAdapter writes the passed data to a HTTP socket.
Each call to the output method creates a new package.

The behavior of this Adapter can be set using the following properties:

Key	Description	Example
address	IP address or hostname of the remote host.	192.168.1.10
port	Port number on the remote host.	1337
request_target	request-target derived from the target URI number on the remote host.	/world
content_type_header	The content-type entity header used to indicate the media type of the resource.	application/xml; charset=utf-8

HttpInputAdapter

```
de.uni_oldenburg.mate.polymorphic_interface.http_adapters.HttpInputAdapter
```

This InputAdapter receives data from a HTTP socket and sends the contents to the server.

The behavior of this Adapter can be set using the following properties:

Key	Description	Example
host	The hostname representing the interface to which this connector will bind, or 0.0.0.0 for all interfaces.	localhost
port	Local port number the socket is bound to.	1337
request_target	The request target in origin-form the adapter will accept requests on.	/world

Configuration Example

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration xmlns="http://polymorphic-interface.mate.uni-oldenburg.de/configuration">
    <models directory="./models">
        <!-- ... -->
    </models>
    <connectors>
        <resources directory="./connectors">
            <jar file="http_adapters-1.0.jar" module="de.uni_oldenburg.mate.polymorphic_interface.
                http_adapters.Module" />
        </resources>
        <instances>
            <adapters>
                <input id="Http-Input-Adapter" class="de.uni_oldenburg.mate.polymorphic_interface.
                    http_adapters.HttpInputAdapter">
```

```
13                      <property key="host" value="127.0.0.1" />
14                      <property key="port" value="1337" />
15                      <property key="request_target" value="/world" />
16                  </input>
17                  <output id="Http-Output-Adapter" class="de.uni_oldenburg.mate.polymorphic_interface.
                        http_adapters.HttpOutputAdapter">
18                      <property key="address" value="127.0.0.1" />
19                      <property key="port" value="1337" />
20                      <property key="request_target" value="/world" />
21                      <property key="content_type_header" value="application/xml; charset=utf-8" />
22                  </output>
23              </adapters>
24              <translators>
25                  <!-- ... -->
26              </translators>
27          </instances>
28      </connectors>
29      <pipes>
30          <!-- -->
31      </pipes>
32  </configuration>
```

2

Rabbit-MQ-Adapter

This project includes two rabbitMQ adapters that can be used to broadcast or receive data from/to a rabbitMQ-server.

MVN Build + Integration-Tests Requirement

Installation

- Installation & Starting RabbitMQ-Server
- Optional: Install Management Plugin
  - http://localhost:15672

Changing default password

For the Build Process and the Integration-Tests a running RabbitMQ-Server on `localhost` on standard port `5672` is required.
In addition, the password of the user `guest` must be changed to `helloworld`:

Linux:

```
sudo rabbitmq-server                                # starts the server
sudo rabbitmqctl change_password guest helloworld # changes the password
```

Windows (inside sbin directory e.g. `C:\Program Files\RabbitMQ Server\rabbitmq_server-3.6.9\sbin`):

```
./rabbitmqctl.bat change_password guest helloworld # changes the password
```

Enable TLS Support

Furthermore, a secure connection must be enabled on port `5671` to successfully run the integration tests.
All needed test certificates and keys are located in this repository under `src/integration-test/resources`.
The private key and certificates are intended to be only used for testing purposes.

Edit the RabbitMQ configuration file to enable TLS support (location of rabbitmq.config):

```
[
  {rabbit,
    [
      {ssl_listeners, [5671]},
      {ssl_options, [{cacertfile, "C:/Git/Polymorphic-Interface/adapters/rabbitmq-adapters/src/
          integration-test/resources/cacert.pem"},
                     {certfile,   "C:/Git/Polymorphic-Interface/adapters/rabbitmq-adapters/src/
                         integration-test/resources/cert.pem"},
                     {keyfile,    "C:/Git/Polymorphic-Interface/adapters/rabbitmq-adapters/src/
                         integration-test/resources/key.pem"},
                     {verify, verify_peer},
                     {fail_if_no_peer_cert, false}]}
    ]}
].
```

Change the paths to fit your environment.

1

RabbitMQOutputAdapter

```
1 de.uni_oldenburg.mate.polymorphic_interface.rabbitmq_adapters.RabbitMQOutputAdapter
```

This OutputAdapter broadcasts the passed data to the correspondend server. Each call to the output method creates a new message.
The behavior of this Adapter can be set using the following properties:

Key	Description	Example
username	The username for the Server-Connection.	PGMate
password	The password for the given username	Secret
host	The host for the Server-Connection	localhost
port	The Port for the Server-Connection	5672
use_tls	Whether or not the tls protocol should be used.	true, false
exchange_name	The Exchange_Name to declare where the Data will be send	PGMate
exchange_type	The Exchange_Type how the Data will be send	direct, fanout
exchange_routing_key	A Routing Key to describe what kind of Data will be send.	warnings, error, commands
exchange_durable	Whether the exchange is durable or not. If durable the exchange survives a broker restart.	true

RabbitMQInputAdapter

```
1 de.uni_oldenburg.mate.polymorphic_interface.rabbitmq_adapters.RabbitMQInputAdapter
```

This InputAdapter receives the data from the correspondend server. Each new Message will notify the given InputListener.
The behavior of this Adapter can be set using the following properties:

Key	Description	Example
username	The username for the Server-Connection.	PGMate
password	The password for the given username	Secret
host	The host for the Server-Connection	localhost
port	The Port for the Server-Connection	5672
use_tls	Whether or not the tls protocol should be used.	true, false
queue_name	The Queue_Name to declare which queue will get subscribed	PGMate
x-message-ttl	The args_durability specifying the time in milliseconds to preserve data in the queue.	100
durable	The durable tag to determine whether the queue has any durability enabled or not.	true, false
filter_empty_messages	Whether or not empty messages should be filtered.	true, false

UDP Adapters

This project includes two adapters that can be used to send and receive data via the User Datagram Protocol (UDP).

The Adapters will be used in a pipe of the Polymorphic Interface to communicate with the Controller.

UdpOutputAdapter

```
de.uni_oldenburg.mate.polymorphic_interface.udp_adapters.UdpOutputAdapter
```

This OutputAdapter writes the passed data to a UDP socket.
Each call to the output method creates a new package.

The behavior of this Adapter can be set using the following properties:

Key	Description	Example
port	Port number on the remote host.	1337
address	IP address or hostname of the remote host.	192.168.1.10

UdpInputAdapter

```
de.uni_oldenburg.mate.polymorphic_interface.udp_adapters.UdpInputAdapter
```

This InputAdapter receives data from a UDB socket and sends the contents to the server.

The behavior of this Adapter can be set using the following properties:

Key	Description	Example
port	Local port number the socket is bound to.	1338
buffer_size	Optional: Length of the buffer for receiving packages. Should be set equal to the size of the longest expected packet (length in byte). Default: 1024.	128

Configuration Example

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration xmlns="http://polymorphic-interface.mate.uni-oldenburg.de/configuration">
    <models directory="./models">
        <!-- ... -->
    </models>
    <connectors>
        <resources directory="./connectors">
            <jar file="udp_adapters-1.0.jar" module="de.uni_oldenburg.mate.polymorphic_interface.udp_adapters.Module" />
        </resources>
        <instances>
            <adapters>
                <input id="Udp-Input-Adapter" class="de.uni_oldenburg.mate.polymorphic_interface.udp_adapters.UdpInputAdapter">
                    <property key="port" value="13377" />
                    <property key="buffer_size" value="128" />
                </input>
                <output id="Udp-Output-Adapter" class="de.uni_oldenburg.mate.polymorphic_interface.udp_adapters.UdpOutputAdapter">
```

```
                    <property key="address" value="127.0.0.1" />
                    <property key="port" value="13377" />
                </output>
            </adapters>
            <translators>
                <!-- ... -->
            </translators>
        </instances>
    </connectors>
    <pipes>
        <!-- -->
    </pipes>
</configuration>
```

2

Translator

Base64 Translators

This project includes two translators that can be used to convert bytes to Base64 String wrapped in XML code and vice versa.
The XML schema of the generated XML code is defined in the file base64-wrapper.xsd.

BytesToBase64InputTranslator

```
1 de.uni_oldenburg.mate.polymorphic_interface.base64_translators.BytesToBase64InputTranslator
```

This InputTranslator converts bytes into a Base64 String wrapped in XML code.

Base64ToBytesOutputTranslator

```
1 de.uni_oldenburg.mate.polymorphic_interface.base64_translators.Base64ToBytesOutputTranslator
```

This OutputTranslator converts a Base64 String wrapped in XML code back into bytes.

1

NMEA0183 Translators

This project includes a translator that can be used to convert NMEA0183 XML to NMEA0183 messages.

See Schemas Repository for NMEA0183 XML representation and examples.

Building and Running Tests

Make sure to init and update the schemas git submodule.
Some integration tests require test files from that repository.

```
git submodule update --init
mvn install
```

Running Tests with IntelliJ

When editing the project with IntelliJ errors might occur when executing the integration tests.

Make sure to declare the `src/integration-test/resources` directory as *Test Resources*.
Go to the settings: `File -> Project Structure... -> Modules`.

NMEA0183OutputTranslator

```
de.uni_oldenburg.mate.polymorphic_interface.nmea0183_translators.NMEA0183OutputTranslator
```

Configuration Example

```
<jar file="nmea0183_translators-1.0.0.jar" module="de.uni_oldenburg.mate.polymorphic_interface.
    nmea0183_translators.Module" />
```

```
<output id="XML-to-NMEA0183" class="de.uni_oldenburg.mate.polymorphic_interface.
    nmea0183_translators.NMEA0183OutputTranslator" />
```

Supported Messages

- GLL
- VHW
- VTG
- RPM
- RSA
- HDT
- ROT
- MWV
- DPT
- VDM mit AIS Typ 1
- VDM mit AIS Typ 5

1

NMEA2000-Translator

This project includes two translators that can be used to convert a NMEA2000 XML message as String into a NMEA2000 byte array message and vice versa.

The produced or received XML is based on the NMEA2000 standard description.
The description is referenced by the NMEA2000 XML description within this project in the schemas git submodule.

Installation guide

Building the jar needs an additional `nmea2000-serializer` library that is used to create NMEA2000 message. The library is used in the tests only.

Furthermore the schemas git submodule must be initialized.

```
git clone git@gitlab.uni-oldenburg.de:PG-MATE/EPD.git
cd EPD/nmea2000-serializer
mvn clean install

cd ../../
git clone git@gitlab.uni-oldenburg.de:PG-MATE/Polymorphic-Interface.git
git submodule update --init
cd Polymorphic-Interface/translators/nmea2000_translator
mvn clean install
java -jar target/nmea2000_translator-1.0.jar
```

In case of any errors follow the stacktrace or check your maven or java installation.

NMEA2000InputTranslator

```
de.uni_oldenburg.mate.polymorphic_interface.nmea2000_translator.NMEA2000InputTranslator
```

The NMEA2000InputTranslator parsed NMEA2000 message bytes into the internal models representation of the NMEA2000 standard description. The internal models representation is then used to create a XML String representation of the NMEA2000 message bytes. Each call to the `translate` method creates a new NMEA2000 XML message.
The InputTranslator needs a NMEA2000 standard description reference in XML. The referencing XML file can be set using the following properties:

Key	Description	Example
XMLFile	The full path to the XML file that references the NMEA2000 standard.	src/test/resources/nmea2000v1.1.xml

NMEA2000OutputTranslator

```
de.uni_oldenburg.mate.polymorphic_interface.nmea2000_translator.NMEA2000OutputTranslator
```

The NMEA2000OutputTranslator parsed a XML String representation of a NMEA2000 message into the internal models representation of the NMEA2000 standard description. The internal models representation is then used to create valid NMEA2000 message bytes of the XML String representation. Each call to the `translate` method creates a new NMEA2000 byte message.
The OutputTranslator needs a NMEA2000 standard description reference in XML. The referencing XML file can be set using the following properties:

Key	Description	Example
XMLFile	The full path to the XML file that references the NMEA2000 standard.	src/test/resources/nmea2000v1.1.xml

NMEA2000 Zuse Commands Translators

This project includes a translator that can be used to convert NMEA2000 XML to Zuse command messages.
The translator only converts messages with the PGN 000001 (RPM) or
127245 (Rudder angle).
Other messages are ignored.

```
1 de.uni_oldenburg.mate.polymorphic_interface.nmea2000_zuse_commands_translator.
    NMEA2000ZuseCommandsOutputTranslator
```

The behavior of this translator can be configured using the following property:

Key	Description	Example
byte_order	The byte order to be used when creating the command messages.	LITTLE_ENDIAN or BIG_ENDIAN

1

S100 Translator

This Java application creates XML String representations by parsing passed EMF Ecore models with its properties and attributes - and vice versa. In addition, the de- and serialization of EMF Ecore object in the S100 Model context is provided. In comparison to with the described parsers this application is a translator for the S100 data model.

Preconditions

- Java JDK 8
- Maven 3
- Server-API
- Base64 translator

Project related subfolders

- The **eMIRDevs_libraries** subfolder contains and encapsulates the third party repositories needed by the S100Util classes of the eMIR project.
  You need to copy the `de` subfolder within `eMIRDevs_libraries` into your local `.m2/repository`:

```
cp -rf eMIRDevs_libraries/* ~/.m2/repository
```

Implementation specifics

If a S100 object is not initialized in an appropriate way the translator once threw an error and stored the incorrect bytes. Due to project related dependencies the following lines inside the `EObjectToXMLStringImpl` has been changed to deactivate the described behavior for one specific error:

```
try {
    resource.save(byteArrayOutputStream, null);
} catch (Resource.IOWrappedException e) {
    LOGGER.warn("Resource.IOWrappedException occurred while processing some bytes.", e);
    // NOTE: the byte array is actually filled with the correct remaining bytes. All references not
        found has been discarded.
}
```

Normally, this is not what the translator should do but with the error not thrown the upper process within the *Polymorphic Interface Server* can go on. You may want to re-activate this by replacing the lines with:

```
resource.save(byteArrayOutputStream, null);
```

Keep in mind to change the tests afterwards.

Clone the project and run the maven-build process

```
git clone git@gitlab.uni-oldenburg.de:PG-MATE/Polymorphic-Interface.git
cd Polymorphic-Interface/s100_translator
mvn clean install
```

Run the translator

```
java -jar target/s100translator-2.0.jar
```

1

Create example xml data

Use the class `de.uni_oldenburg.mate.polymorphic_interface.s100translator.example_builder.ExampleBuilder` to build S-100 xml example data.
Run the main method with IntelliJ or any other ide.

XML String Translators

This project includes two translators that can be used to convert XML code as String into bytes and vice versa.

XmlStringInputTranslator

```
1 de.uni_oldenburg.mate.polymorphic_interface.xml_string_translators.XmlStringInputTranslator
```

This InputTranslator converts XML code as bytes into XML code as String.

The behavior of this Translator can be set using the following properties:

Key	Description	Example
charset	The character set of the data to be converted.	UTF-8

XmlStringOutputTranslator

```
1 de.uni_oldenburg.mate.polymorphic_interface.xml_string_translators.XmlStringOutputTranslator
```

This OutputTranslator converts XML code as String into XML code as bytes.

The behavior of this Translator can be set using the following properties:

Key	Description	Example
charset	The character set of the data to be converted.	UTF-8

1

Server

Server

Running Tests with IntelliJ

When editing the project with IntelliJ errors might occur when executing the integration tests.

Make sure to declare the `src/integration-test/resources` directory as *Test Resources*.
Go to the settings: `File -> Project Structure... -> Modules`.

1

Server-API

Server-API

The Server API can be used to implement Connectors for the server.
First the Server API must be compiled and installed locally.

```
git clone git@gitlab.uni-oldenburg.de:PG-MATE/Polymorphic-Interface.git
cd Polymorphic-Interface/server-api
mvn clean install
```

The Server API can then be used as a dependency in a Maven project:

```
<dependency>
    <groupId>de.uni_oldenburg.mate.polymorphic_interface</groupId>
    <artifactId>server_api</artifactId>
    <version>2.2.0</version>
</dependency>
```

The JavaDoc documentation describes how to implement a Connector for the server.

```
target/apidocs/de/uni_oldenburg/mate/polymorphic_interface/server_api/package-summary.html
```

1

Navibox Simulator

NaviBox Simulator

This project contains a basic abstraction of the NaviBox for testing purposes.
The NaviBox Simulator is able to send (produce) messages via RabbitMQ from files.
These files should be recorded messages from the real NaviBox.

Preparation

Use the Polymorphic Interface to record messages from the NaviBox.

These adapters and translators should be used for that purpose:

- RabbitMQ input adapter
- Base64 input and output translators
- File output adapter

A pipe must be defined in the configuration file of the Polymorphic Interface.

This process produces a directory containing recorded messages in separate files.

Using the simulator

After recording messages from the original NaviBox the Simulator can be used to replay those messages.

```
usage: java -jar navibox_simulator-1.0.jar -d <arg> [--delay <arg>]
 -d,--directory <arg>   Directory containing the files that should be send with the RabbitMQ queue.
    --delay <arg>       Delay between the sending of files. Default: 100 ms.
```

The simulator will publish each message to a queue called `PG-System-Test` of the default exchange.
It uses the user `guest` and password `helloworld` just like the integration test of the RabbitMQ Adapters.

Fast Mode

To enable the fast mode the `--delay` option has to be provided.
The simulator will publish the messages sequentially with a fixed delay.

Realtime Mode

This is the default mode that is active when no `--delay` option is provided.
In this mode the messages are send in the same order and time difference as they were recorded.
The modification date of the message files is used for this mode.

1

S100-Sample-Sender

s100-sample-sender

A simple JavaScript tool that sends S-100 rpm and rudder angle commands every second as datagram package.

Usage

- Install NodeJS (LTS version will be fine)
- Configure *pg-mate-system-test* to accept S-100 command on port 13377 (see *Server configuration example* below)
- Run script with `node index.js`

Server configuration example

Adapter

```
<input id="Udp-Input-Adapter" class="de.uni_oldenburg.mate.polymorphic_interface.udp_adapters.
    UdpInputAdapter">
    <property key="port" value="13377" />
    <property key="buffer_size" value="2048" />
</input>
```

Pipe

```
<pipe id="Udp-to-NaviBox">
    <adapters>
        <input id="Udp-Input-Adapter" />
        <output id="Rabbitmq-Output-Adapter-NaviBox" />
    </adapters>
    <translators>
        <input id="Bytes-to-XML" />
        <output id="S-100-to-Bytes" />
    </translators>
    <steps>
        <step model="S-100" />
    </steps>
</pipe>
```

1

S100 XML Analyser

s100-xml-analyser

Tool helping analysing many s100 sample messages.

Usage

- Extract sample data
  - Sample data can be found in directory `pg-mate-system-test/example_data/S-100_bytes` in multiple zip files
- Open this project with any IDE
- Edit `Main.java` to point to the directory where the sample files have been extracted to
- Optional: Edit `XmlFileAnalyser.java` to get the needed information
- Run the main method with the IDE

1

A.1.4 Schemas

Schemas

Contains all schema and transformation files used by PG-MATE to parse and convert various data formats.

Available transformation rules

- NMEA2000 to NMEA0183
- NMEA2000 to S100
- S100 to NMEA2000
- RTZ to NMEA2000

Available transformable Messages

The transformation rules are implemented and available only from right to left in the read direction of the following tables.

S100 to NMEA2000 and NMEA2000 to NMEA0183

S100 Description	S100 referencedIdentifier	NMEA 2000 PGN name	NMEA 2000 PGN number	NMEA 0183 message name	NMEA 0183 message id
GPS Location	00040100	Position Rapid Update	129025	Geographic Position	GLL
Speed through Water	00080300	Speed Through Water	128259	Water speed and heading	VHW
Speed/Course over Ground	00040300	SoG / CoG Rapid Update	129026	Track made good and Ground speed	VTG
Engine RPM	00070400	Engine Paramters, Rapid Update (RPM)	127488	Revolutions	RPM
Rudder Angle	00070200	Rudder	127245	Rudder sensor angle	RSA
Heading	00040200	Vessel Heading	127250	Heading - True	HDT
Rate of Turn	00040400	Rate of Turn	127251	Rate of turn	ROT
Windspeed and direction	00030300	Wind Speed and Direction	130306	Wind Speed and Angle	MWV
Water Depth	00080600	Water Depth	128267	Depth	DPT
AIS Position Report	02010700, 01010700, 05010700, 06010700, 00010700, 00020700, 01020700, 02020700	AIS Class A Position Report	129038	AIS VHF Datalink (Own-vessel) Message	VDM, VDO
AIS Static Voyage Report	02010701, 01010701, 05010701, 06010701, 00010701	AIS Class A Static and Voyage Related Data	129794	AIS VHF Datalink (Own-vessel) Message	VDM, VDO

NMEA2000 to S100

1

NMEA 2000 PGN name	NMEA 2000 PGN number	S100 Description	S100 referencedIdentifier
Rudder	127245	CommandedRudderAngle	NavTor
NaviBox RPM Command	000001	CommandedEngineRpm	NavTor

RTZ to NMEA2000

RTZ Message	NMEA 2000 PGN name	NMEA 2000 PGN number
valid RTZ	Route and WP Service - Route/WP-List Attributes	130066
valid RTZ (active Route only)	Route and WP Service - Route - WP Name and Position	130067
valid RTZ (active Route only)	NaviBox RPM Command	000001
valid RTZ (active Route only)	Rudder	127245

Messages (and message sources) that have been used to test the transformation rules

- Stream of different active NaviBoxes
- Recorded Data from the Zuse
- Each transformed message has an example in the directory of it's XSD-Schema

XSDs and XML-Descriptions

- An XSD schema can be found in each Message-Standard-type-directory for the validation of the individual messages. Some directories contain an XML-description of the Message-Standard.

xslt-translator-test-tool

- The xslt-translator-test-tool contains tests for each transformation rule.

Message Fields and Content which are currently not transformed

Transformation	Not transformed fields/Content	Message Number
S100 to NMEA2000	angularVelocity	06010700
S100 to NMEA2000	"vessel:NavigationInformation"-status	05010701, 06010701
S100 to NMEA2000	estimatedTimeOfArrival	05010701, 06010701
S100 to NMEA2000	ObjectSurfaceInformation	01020700
S100 to NMEA2000	AIS object type	00010701, 01010701, 02010701, 05010701, 06010701
NMEA2000 to NMEA0183	AIS timestamp	VDM, VDO

2

URL zum Repository: https://gitlab.uni-oldenburg.de/PG-MATE/Schemas

A.1.5 EPD

e-Navigation Prototype Displays

Introduction

eMIR EPD (eNavigation Prototype Displays) consists of two applications for demonstrating potential e-navigation solutions. An ECDIS-like shipside application and a shoreside application.

The applications are in Java and use OpenMap(tm) for presenting geospatial information, and JavaBeans(tm) as a component framework.

Prerequisites

- Java 8
- Maven 3

Building

```
mvn clean install
```

Running

Runnable jar files are located here

```
distribution/EPD-Ship-Singlejar/target/epd-ship-dist-X.Y-SNAPSHOT.jar
distribution/EPD-Shore-Singlejar/target/epd-shore-dist-X.Y-SNAPSHOT.jar
```

To run from command line

```
java -jar distribution/EPD-Ship-Singlejar/target/epd-ship-dist-X.Y-SNAPSHOT.jar
```

Folders with settings files are created in the home folder

```
<home folder>/.epd-ship
<home folder>/.epd-shore
```

Creating Windows EXE files

Windows executables can be created by using the following Maven profile

```
mvn clean install -Pexe
```

The executables will be located here

```
distribution/EPD-Ship-Singlejar/target/epd-ship.exe
distribution/EPD-Shore-Singlejar/target/epd-shore.exe
```

Eclipse development

Use M2 Eclipse plugin or use Maven eclipse target

```
mvn eclipse:eclipse
```

1

Virtual AIS transponder

The virtual transponder can provide a live anonymized AIS feed. The following settings can be used for the transponder:

Host:port: `ais.e-navigation.net:8002`
Username/password: `anon/anon`

Quick start

To be able to see AIS targets and possible own ship, the sensor should be configured. In the main window press the Setup button in the top and go to the Sensor tab. Choose either TCP or serial connection type for AIS and configure TCP host/port or serial port.

If a separate sensor is used for GPS this can be configured the same way.

If the AIS source is not a transponder providing own ship information, an own ship can be simulated by choosing a vessel target present in the AIS stream. On the bottom of the sensor tab enable Simulated GPS and enter MMSI. If the AIS source provides lots of targets, the targets shown can be limited by selecting a sensor range so targets farther than this distance away not will be shown.

Press OK and restart application.

Design

EPD uses a component based design to facilitate an event driven architecture. Functionality is encapsulated in components. Events come from sensor and user input.

Using components allows easy collaborative development as components can be developed independently. Components automatically identify each others in the BeanContext, so no hard wiring is necessary. Components can also be moved between different projects using the same component framework. E.g. OpenMap.

Contribution

Fork the project and make pull requests.

Try to use the component architecture as much as possible. Implement components and hook up to other components with the `findAndInit` method rather than hard-wiring. Try to follow the coding standards already used in the project and document within the code with Javadoc comments. For more extensive documentation use the Wiki.

ENC layer

EPD does not come with an ENC layer but with the possibility to add one as a plugin. Currently the only known OpenMap ENC layer is a commercial one from the danish company Navicon.

To use Navicon ENC layer with EPD-ship please follow the steps below

1. Contact Navicon sales regarding a purchase of their ENC/S52 rendering engine. Mention the following:

- To be used with DMA E-navigation Prototype Display
- Version for use with OpenMap 5
- If you are using a 64-bit machine, ask for 64-bit dongle drivers

1. After purchasing you will receive a SDK and a USB dongle.
   Unpack the SDK in a directory. The SDK should at least contain the following folders
   `lib`, `native` and `data`.
2. Run installer in `native`.
3. Place the `lib` folder in `<home folder>/.epd-ship`
4. Create the folder: `<home folder>/.epd-ship/navion`
5. Place the folders `native` and `data` in the newly created folder.
6. In on a 32-bit system. Remove `native/keylock.dll` and rename `native/keylock32.dll` -> `native/keylock.dll`
7. Copy `<home folder>/.epd-ship/enc_navicon.properties` to `<home folder>/.epd-ship/enc.properties`.
8. Edit `<home folder>/.epd-ship/enc.properties` and set charts location in `enc.s57PathLocation`.
9. Run application
10. Enter Setup -> Map and enable ENC. Restart.

Utilities

The module `epd-util` contains some utilities to use with EPD.

Route from AIS generation

A route file can be generated given an AIS file given MMSI number. Example:

```
1 java -jar epd-util/target/epd-util-0.1-SNAPSHOT.jar aistoroute -in ~/tmp/aisdump.txt -out
    ~/tmp/route.txt -mmsi 304913000
```

Attribution

Some icons by Yusuke Kamiyamane. Licensed under a Creative Commons Attribution 3.0 License.

3

URL zum Repository: https://gitlab.uni-oldenburg.de/PG-MATE/EPD/tree/mate-master

EPD-API

EPD API

Introduction

To add additional behavior to the EPD project, the project group MATE created the EPD API. It has several registries to register new Frames, Statuses and Preference Panels.
The main reason for this implementation is to clearly separate external functionality from the EPD core.
The integration of the registries inside the EPD Core is annotated with the `@ApiHook` interface.

Integration

To integrate additional functionality into the EPD, follow these steps:
First, add the maven-dependency of the new project to the EPD-Shore-pom.xml and add the module to in the EPD-pom.xml.
Add and activate the new module (needs to extend from AbstractModule) in the epd-api-plugins.properties. To deactivate any plugin simply change the value from true to false. Examples:

`de.uni_oldenburg.mate.epd.epd_api_examples.ExamplesModule=false de.uni_oldenburg.mate.epd.gamepad.GamepadModule=true`

Then add the API as a maven dependency to your plugin:

```
<groupId>de.uni_oldenburg.mate.epd</groupId>
<artifactId>epd-api</artifactId>
<version>2.0</version>
```

The given Module will be injected into the needed registries. Only use these registries to extend the EPD functionality.

Registries

The following Registries are currently implemented:

- The FrameRegistry registers new frames in the menu bar. The method addFrame needs a String as the menu title (will be displayed in the plugins tab), a singleton/multipleOpen flag which decides if the user can open the frame more than once and the callable to get the Frame(s).
- Examples
- SingleFrameExample
- MultipleFramesExample
- The PreferencesRegistry registers a tab in the EPD preferences. The tab with the belonging panel can save and load properties at the applications start. addPreferenceCategory needs the PreferenceCategory.
- Example
- The StatusRegistry registers a Status in the bottom right corner of the EPD. It may display the status of a remote (server or client) connection. It needs a Consumer so the Status can show changed at runtime.
- Example
- The TerminationListenerRegistry registers a Listener that will get invoked when the EPD is about to get closed. This is needed if some cleanups are necessary before the application gets closed.
- Example

```
package de.uni_oldenburg.mate.epd.epd_api_examples;

import com.google.inject.Inject;
import de.uni_oldenburg.mate.epd.epd_api.termination.TerminationListener;
import de.uni_oldenburg.mate.epd.epd_api.termination.TerminationListenerRegistry;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * The TerminationListener responds to the termination of the application
```

```
 */
public class TerminationListenerExample implements TerminationListener {

    private static Logger LOG = LoggerFactory.getLogger(TerminationListenerExample.class);

    @Inject
    TerminationListenerExample(TerminationListenerRegistry terminationListenerRegistry) {
        terminationListenerRegistry.addTerminationListener(this);
        LOG.info("TerminationListener connected");
    }

    @Override
    public void applicationWillTerminate() {
        LOG.info("EPD is about to terminate");
    }
}
```

NmeaSource

- The NmeaSource is a special interface. It accepts NMEA 0183 messages and parses it. If there are AIS messages or other Messages which can be displayed in the EPD already, the EPD will do so. It can also register AbstractSentenceListener to get the parsed data back in various Sentences.
- Example

```
package de.uni_oldenburg.mate.epd.epd_api_examples;

import com.google.inject.Inject;
import de.uni_oldenburg.mate.epd.epd_api.nmea.NmeaSource;

public class NmeaSourceExample {

    private final NmeaSource nmeaSource;

    @Inject
    NmeaSourceExample(final NmeaSource nmeaSource) {

        this.nmeaSource = nmeaSource;
        drawExampleVessels();
    }

    private void drawExampleVessels() {
        nmeaSource.receiveNmeaMessage("!AIVDM,1,1,,A,14eG;o@034o8sd<L9i:a;WF>062D,0*7D"); // AIS
            example from https://en.wikipedia.org/wiki/NMEA_0183
        //nmeaSource.receiveNmeaMessage("!AIVDM,1,1,0,,16e:U6h001PjlhjO1F=9Gqbh28>C,0*16"); // Real
            example from the e nav cloud
    }
}
```

NMEA2000-Serializer

NMEA2000 message creator

This module creates NMEA2000 command messages. In detail the API provides interfaces to create `NaviBox RPM Command` (PGN=1) and `Rudder` (PGN=127245) messages.

Installation guide

```
git clone git@gitlab.uni-oldenburg.de:PG-MATE/EPD.git
cd EPD/nmea2000_messageCreator
mvn clean install
java -jar target/nmea2000_messageCreator-1.0.jar
```

In case of any errors follow the stacktrace or check your maven or java installation.

Data type restrictions

The *NMEA2000 Message Creator* has some restrictions regarding the parameters. Those are (enumerated by the messages):

- `NaviBox RPM Command` (PGN=1): The scale used to represent the RPM is 0.25. Meaning the actual passed RPM value is multiplied by the inverse of this scale before using this API. Expects a long integer in the range of -65535 to 65535 (not range of signed integer 32 bit).
- `Rudder` (PGN=127245): The scale used to represent the RPM is 0.0004. Meaning the actual Angle Order value is multiplied by the inverse of this scale before using this API. Expects a long integer in the range of +- 31415 (not the whole range of signed integer 16 bit).

The restrictions refer to the NMEA2000 tstandard description.

Examples:

- `NaviBox RPM Command` (PGN=1): The inverse scale is 4. Let us assume the actual RPM value is 100. The passed value into this API is then: 100*4=400.
- `Rudder` (PGN=127245): The inverse scale is 10^4. Let us assume the actual Angle Order value is -3. The passed value into this API is then: -3*10^4=-30000.

1

EPD-Vessel-Connection

EPD Vessel Connection

EPD Vessel Connection is a module which handles the data exchange to a remote RabbitMQ server which provides and receives vessel or route data.
There are two connections on the same server. One provides and receives vessel data, the other exchanges RTZ-routes for the EPD Route Pilot.

Prerequisites

- Java 8
- Maven 3

Building

‘`mvn clean install`

Maven dependency for this module:

```
<groupId>de.uni_oldenburg.mate.epd</groupId>
<artifactId>vessel-connection</artifactId>
<name>Vessel Connection</name>
<version>1.0</version>
```

Running

This module starts as a plugin within the EPD. By default the VesselConnectionModule is activated in the epd-api-plugins.

Quick Start

Set the configuration for a RabbitMQ server inside the EPD preferences (File -> Preferences -> RabbitMQ). You can test the configuration before connecting.
A BottomStatus indicates the status of the connection for vessel data (RabbitMQ Zuse) and route data (RabbitMQ RTZ).

If activated, the vessel-data will be shown in the VirtualHandles Module. The RoutePilot for RTZ-Route-Exchange can be started in the Virtual Handles Module also.

RabbitMQ Connection

The RabbitMQConnection creates a connection to the server and authenticates with the username and password specified in the EPD preferences.

Vessel data

The RabbitMQInputAdapter and RabbitMQOutputAdapter each create a channel to receive and send data on these default queues:

RabbitMQ-Value	MATE default
HOST_PORT	5672
EXCHANGE_NAME	zuse-target-exchange
EXCHANGE_TYPE	direct
EXCHANGE_ROUTING_KEY	zuse-target-state
QUEUE_NAME	zuse-current-state
ARGS_DURABILITY	100

1

RabbitMQ-Value	MATE default
BASIC_QOS	20
DURABLE	true

RTZ Route data

The RabbitMQInputAdapterRTZ and RabbitMQOutputAdapterRTZ each create a channel to receive and send data on these default queues:

RabbitMQ-Value	MATE default
HOST_PORT	5672
EXCHANGE_NAME	route-target-exchange
EXCHANGE_TYPE	direct
EXCHANGE_ROUTING_KEY	route-target-state
QUEUE_NAME	route-current-state
ARGS_DURABILITY	300000
BASIC_QOS	20
DURABLE	true

2

EPD-Own-Ship

EPD Own Ship

EPD Own Ship is a module which is a module that displays and highlights your own ship on the map.
The module uses sensor data and AIS messages to display your own ship on the map.
The MMSI of your own ship can be set via the settings.

Prerequisites

- Java
- Maven 3

Build

The module is built together with the EPD.
The module can be built manually via `mvn clean install`.

Running

This module starts as a plugin within the EPD.
By default the module is activated in the epd-api-plugins.

1

EPD-Virtual-Handles

EPD Virtual Handles

Introduction

The Virtual Handles module is an EPD plugin for the visualization and the control of the state of a connected vessel.
It imitates the instruments on the vessels bridge. Some elements present incoming vessel data, other handle user input.
The Virtual Handles module works in conjunction with the Vessel Connection module.

Prerequisites

- Java 8
- Maven 3

Building

mvn clean install

The Maven dependency for this module:

```
<groupId>de.uni_oldenburg.mate.epd</groupId>
<artifactId>virtual-handles</artifactId>
<name>Virtual Handles</name>
<version>2.0</version>
```

Running

The Virtual Handles start within the EPD. By default the VirtualHandlesModule is activated in the epd-api-plugins.

Quick Start

Open the Virtual Handles panel inside the EPD (Plugins -> Virtual Handles).
If the connection to the vessel is established via the the Vessel Connection module, the data is transmitted automatically.

Virtual Handles Elements

These are the various elements within the Virtual Handles:

Visual Elements:

- Turn rate scale (output)
- Rudder angle indicator (output)
- Bow thruster slider (input)
- Machine Telegraph (input)
- Rudder (input)
- Heading (output)
- Compass (output)
- GPS with course and speed over ground (output)
- Echolot (output)
- Autopilot (input and output)

Data Table (output only)

- Engine RPM
- Speed through water
- Rudder angle

- Wind angle
- Wind speed

Credits

The Virtual Handles originate from the MATJES group at Offis (www.offis.de) and were adapted by the PG MATE (pg-mate.uni-oldenburg.de) for the needs of the EPD.

MATJES Authors: Uwe Wilko Grünefeld, Tim Claudius Stratmann, Julia Gerdes

EPD-Route-Pilot

EPD Route Pilot

EPD Route Pilot is a module which handles routes for autonomous driving. It transforms an EPD-route into a RTZ-Route and sends it out via the EPD Vessel Connection.
Also it analyzes incoming RTZ-Routes to mark current waypoints or stopping the Route Pilot.

Prerequisites

- Java 8
- Maven 3
- EPD Vessel Connection
- Route Exchange Format Library

```
<repository>
    <id>mate</id>
    <name>MATE</name>
    <url>http://duemmer.informatik.uni-oldenburg.de:11003/nexus/content/repositories/releases</url>
    <releases>
        <updatePolicy>always</updatePolicy>
    </releases>
</repository>
<dependency>
    <groupId>de.uni_oldenburg.mate</groupId>
    <artifactId>rtz</artifactId>
    <version>1.2.0</version>
</dependency>
```

Building

```
mvn clean install
```

Maven dependency for this module:

```
<groupId>de.uni_oldenburg.mate.epd</groupId>
<artifactId>routepilot</artifactId>
<name>RoutePilot</name>
<version>1.0</version>
```

Running

This module starts as a plugin within the EPD. By default the RoutePilotModule is activated in the epd-api-plugins.

Quick Start

Set up a RabbitMQ-connection in the VesselConnection Module. Choose a route in the VirtualHandles Module and click "AutoPilot"-Button.
The EPD-Route will be transformed into RTZ-Format and sent out to the MATE Path-Planning-Component.

If the RoutePilot receives a RTZ-Route from the Path-Planning-Component with active waypoints, they will be marked on the EPD-Map.
If a disabled RTZ-Route is received, the Autopilot in the VirtualHandles-Module will be stopped.

1

A.1.6 Pfadplanung

Path planning

This application is configured via a properties file.
The configuration file must contain the following three properties.

- **input_port:**
  The port at which the HTTP server is started. The server receives new route and position data.
- **output_url:**
  The URL to which the current waypoints and the status of the route are sent.
- **output_period:**
  The time in milliseconds between the cyclical sending of the current waypoints and the status of the route.
- **minimal_turn_radius:**
  The smallest radius in nautical miles at which a waypoint is considered to be reached.

For example, a configuration file might look like this:

```
input_port=4242
output_url=http://localhost:4243/waypoints
output_period=1000
minimal_turn_radius=0.01
```

This application is executed via the Java Runtime Environment 8.
The path to the configuration file must be passed as an argument:

```
java -jar path_planning-%version%.jar configuration.properties
```

1

URL zum Repository: https://gitlab.uni-oldenburg.de/PG-MATE/Path-planning

A.1.7 RTZ-Bibliothek

Route Exchange Format Library

Library for parsing, modifying and writing XML documents in the Route Exchange Format (RTZ).
The library requires Java 8.

Installation

This library can be integrated as a Maven dependency into a project.

```
<dependency>
    <groupId>de.uni_oldenburg.mate</groupId>
    <artifactId>rtz</artifactId>
    <version>1.2.0</version>
</dependency>
```

Alternatively, the library can be built locally and added manually as a Jar file to the classpath
of a project. Maven 3 is required for the build process.

```
git clone git@gitlab.uni-oldenburg.de:PG-MATE/Route-Exchange-Format-Library.git
cd Route-Exchange-Format-Library
mvn clean install
```

Usage

A route according to the Route Exchange format (RTZ) consists of at least two waypoints.
The following sections describe how routes and waypoints can be defined and
how RTZ documents can be created and parsed.

Waypoints

Waypoints can be created using a `WaypointBuilder`, as shown in the following example.
See the JavaDoc documentation for more information about the attributes of a waypoint.

```
import de.uni_oldenburg.mate.rtz.Point;
import de.uni_oldenburg.mate.rtz.Waypoint;
import de.uni_oldenburg.mate.rtz.WaypointBuilder;

Point position = Point.at(60, 120);

Waypoint waypoint = new WaypointBuilder(123, position).
                    setName("First waypoint").
                    setMaximumAllowedSpeed(30.0).
                    setTurnRadius(5.0).
                    build();
```

Waypoint objects are immutable. To change an existing waypoint, a new object must be created.
For this purpose, a modified copy based on an existing waypoint can be created,
as shown in the following example.

```
import de.uni_oldenburg.mate.rtz.Point;
import de.uni_oldenburg.mate.rtz.Waypoint;
import de.uni_oldenburg.mate.rtz.WaypointBuilder;

Waypoint w1 = new WaypointBuilder(1, Point.at(60, 120)).
              setMaximumAllowedSpeed(30.0).
              build();

```

1

```
Waypoint w2 = w1.copyAndModify().
                setId(2).
                setPosition(Point.at(0, 0)).
                build();
```

Routes

Routes can be created using a `RouteBuilder`, as shown in the following example.
See the JavaDoc documentation for more information about the attributes of a route.

```
import de.uni_oldenburg.mate.rtz.Point;
import de.uni_oldenburg.mate.rtz.Waypoint;
import de.uni_oldenburg.mate.rtz.WaypointBuilder;
import de.uni_oldenburg.mate.rtz.Route;
import de.uni_oldenburg.mate.rtz.RouteBuilder;

Waypoint w1 = new WaypointBuilder(1, Point.at(60, 120)).build();
Waypoint w2 = new WaypointBuilder(2, Point.at(70, 130)).build();

Route route = new RouteBuilder("Awesome route").
                setStatus("active").
                setVesselName("Zuse").
                setVesselMmsi(235762000).
                addWaypoint(w1).
                addWaypoint(w2).
                build();
```

Route objects are immutable. To change an existing route, a new object must be created.
For this purpose, a modified copy based on an existing route can be created,
as shown in the following example.

```
import de.uni_oldenburg.mate.rtz.Point;
import de.uni_oldenburg.mate.rtz.Waypoint;
import de.uni_oldenburg.mate.rtz.WaypointBuilder;
import de.uni_oldenburg.mate.rtz.Route;
import de.uni_oldenburg.mate.rtz.RouteBuilder;

Waypoint w1 = new WaypointBuilder(1, Point.at(60, 120)).build();
Waypoint w2 = new WaypointBuilder(2, Point.at(70, 130)).build();

Route route = new RouteBuilder("Awesome route").
                addWaypoint(w1).
                addWaypoint(w2).
                build();

Waypoint w3 = new WaypointBuilder(3, Point.at(80, 140)).build();
Route other = route.copyAndModify().
                addWaypoint(w3).
                build();
```

Create RTZ documents

An RTZ document is an XML document that contains all information about a route and its waypoints.
An RTZ document can be created using the `DocumentCreator`, as shown in the following example.

```
import de.uni_oldenburg.mate.rtz.Point;
import de.uni_oldenburg.mate.rtz.Waypoint;
```

```
import de.uni_oldenburg.mate.rtz.WaypointBuilder;
import de.uni_oldenburg.mate.rtz.Route;
import de.uni_oldenburg.mate.rtz.RouteBuilder;
import de.uni_oldenburg.mate.rtz.DocumentCreator;
import org.w3c.dom.Document;

Waypoint w1 = new WaypointBuilder(1, Point.at(60, 120)).build();
Waypoint w2 = new WaypointBuilder(2, Point.at(70, 130)).build();

Route route = new RouteBuilder("Awesome route").
                addWaypoint(w1).
                addWaypoint(w2).
                build();

Document document = DocumentCreator.createDocument(route);
String documentAsString = DocumentCreator.createString(route);
```

The RTZ document created in this example looks as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<route xmlns="http://www.cirm.org/RTZ/1/0"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.cirm.org/RTZ/1/0 RTZ-Schema-version-1-0.xsd"
       version="1.0">
    <routeInfo routeName="Awesome route"/>
    <waypoints>
        <waypoint id="1">
            <position lat="60.0" lon="120.0"/>
        </waypoint>
        <waypoint id="2">
            <position lat="70.0" lon="130.0"/>
        </waypoint>
    </waypoints>
</route>
```

Parse RTZ documents

From an RTZ document, the contained route and its waypoints can be created as objects.
An RTZ document can be parsed using the `DocumentParser`, as shown in the following example.

```
import de.uni_oldenburg.mate.rtz.Waypoint;
import de.uni_oldenburg.mate.rtz.Route;
import de.uni_oldenburg.mate.rtz.DocumentParser;
import org.w3c.dom.Document;

// The previous section describes how RTZ documents can be created.
Document document = ...
String documentAsString = ...

Route routeFromDocument = DocumentParser.parseDocument(document);
Route routeFromString = DocumentParser.parseString(documentAsString);
```

If an invalid RTZ document is passed to the parser, an `InvalidDocumentException` is thrown.

This library does not support the full RTZ format.
Unsupported parts of an RTZ document are ignored by the parser.

3

URL zum Repository: https://gitlab.uni-oldenburg.de/PG-MATE/Route-Exchange-Format-Library

A.2 Testdrehbuch TDB_001_Notfallsteuerung

Testdrehbuch TDB_001_Notfallsteuerung_UDP

Informationen zum Test
Version: 1.1.0
Tester: Tim Wiechmann, Philipp Mudra
Testdatum: 15.09.2017

Übersicht Testfälle
Mögliche Werte für die Attribute der Testfälle:
Priorität: Hoch, Mittel, Niedrig, Offen
Testergebnis: Fehlerhaft, Erfolgreich, Nicht ausführbar, In Ausführung, Offen
Komponenten: EPD Shore (MATE:Shore), EPD Ship (MATE:Ship), RabbitMQ (MATE:Server), RabbitMQ (Testbed), Controller, Polymorphic Interface (MATE:Ship), Polymorphic Interface (Testbed), Navibox, Backup-Steuerung

Testsystem
Prozessor: Intel Core i5-6300U CPU 2.40GHz
Arbeitsspeicher: 8GB
Systemtyp (32 bit oder 64 bit): 64 bit
Betriebssystem: Windows 7

Übersicht Testfälle

Testfall-ID	Testfallbezeichnung	Testfall Kurzbeschreibung	Vorbedingungen	Priorität
TF_001_001	VPN Verbindung herstellen	Um das Schiff von Land aus zu steuern, ist es notwendig eine Verbindung zum Schiff herzustellen. Dies erfolgt über VPN.	Router auf des Schiffs muss konfiguriert sein. IP-Adresse des Routers muss bekannt sein. Das Schiff muss mit dem Internet verbunden sein.	Hoch
TF_001_002	Verbindung zum Schiff aufbauen	Es wird über das Gamepad eine Verbindung zum Schiff hergestellt.	Notfallsteuerung muss gestartet sein. Gamepad muss verbunden sein.	Hoch
TF_001_003	Schiff steuern	Über die Buttons am Gamepad kann die RPM-Anzahl sowie der RudderAngle des Schiffs erhöht oder verringert werden.	Notfallsteuerung muss gestartet sein. Gamepad muss verbunden sein. Verbindung zum Schiff muss vorhanden sein.	Hoch
TF_001_004	Verbindung zum Schiff abbauen	Es wird über das Gamepad eine bestehende Verbindung zum Schiff abgebaut.	Notfallsteuerung muss gestartet sein. Gamepad muss verbunden sein. Verbindung zum Schiff muss vorhanden sein.	Hoch

Testfall TF_001_001
VPN Verbindung herstellen

Testschritt	Vorbedingung	Aktion	Eingabedaten / Testdaten	Testschrittbeschreibung	Erwartetes Ergebnis	Testergebnis
TS_001_1	IP-Adresse, Benutzername, PW (und ggf. VPN-Protokoll) des Routers des Schiffes ist bekannt, das Schiff ist mit dem Internet verbunden, der Router des Schiffes ist konfiguriert.	VPN Verbindung herstellen	IP-Adresse des Schiffs, am Router konfigurierter Benutzername und Passwort, ggf. Domäne	Über Windows/Linux-Boardmittel eine neue VPN-Verbindung erstellen. Dabei Angaben zu der externen IP-Adresse des Routers des Schiffs machen. Des Weiteren ist der am Router konfigurierte Benutzername und Passwort zu pflegen.	VPN Verbindung wurde erfolgreich hergestellt. Zu prüfen über einen Abgleich der IP-Adressen.	Nicht ausführbar

Testfall TF_001_002
Verbindung zum Schiff aufbauen

Testschritt	Vorbedingung	Aktion	Eingabedaten / Testdaten	Testschrittbeschreibung	Erwartetes Ergebnis	Testergebnis
TS_002_1	Gamepad Treiber müssen ggf. installiert sein.	Gamepad anschließen		Gamepad wird am PC/Notebook angeschlossen.	Gamepad ist angeschlossen und kann verwendet werden.	Erfolgreich
TS_002_2		Notfallsteuerung Applikation starten		Notfallsteuerung wird gestartet.	Notfallsteuerung ist gestartet.	Erfolgreich
TS_002_3	Notfallsteuerung ist korrekt gestartet; Gamepad wurde erkannt; Verbindungsdaten wurden korrekt angegeben;	Verbindung aufbauen	Siehe Werte für "Parameter zur Konfiguration"	Über den Button "START" wird die Verbindung zum Gamepad aufgebaut.	Verbindung zum Raspberry Pi im Testbed ist erfolgreich hergestellt.	Erfolgreich
TS_002_4	Verbindung besteht	Zyklische Steuerdaten senden.	Zeitintervall: 0.25 s	Notfallsteuerung sendet kontinuierlich Steuerdaten an das Schiff, damit die Verbindung nicht unterbrochen wird.	Notfallsteuerung sendet kontinuierlich Steuerdaten an das Schiff. Verbindung wird nicht unterbrochen.	Erfolgreich

Testfall TF_001_003
Schiff steuern

Testschritt	Vorbedingung	Aktion	Eingabedaten / Testdaten	Testschrittbeschreibung	Erwartetes Ergebnis	Testergebnis
TS_003_1	Notfallsteuerung muss gestartet sein. Gamepad muss verbunden sein. Verbindung zum Schiff muss vorhanden sein.	RPM steuern	+ 2.000 RPM	RPM wird über den rechten JoyStick des Gamepads auf der Y-Achse gesteuert.	RPM des Schiffs erhöht sich auf + 2.000 RPM	Erfolgreich
TS_003_2	Notfallsteuerung muss gestartet sein. Gamepad muss verbunden sein. Verbindung zum Schiff muss vorhanden sein.	RPM steuern	- 2.000 RPM	RPM wird über den rechten JoyStick des Gamepads auf der Y-Achse gesteuert.	RPM des Schiffs verringert sich auf - 2.000 RPM	Erfolgreich
TS_003_3	Notfallsteuerung muss gestartet sein. Gamepad muss verbunden sein. Verbindung zum Schiff muss vorhanden sein.	RPM steuern	0 RPM	Rechter JoyStick wird in die neutrale Stellung der X- und Y-Achse belassen.	RPM des Schiffs bleibt konstant bei 0 RPM	Erfolgreich
TS_003_4	Notfallsteuerung muss gestartet sein. Gamepad muss verbunden sein. Verbindung zum Schiff muss vorhanden sein.	RudderAngle steuern	2.000	RudderAngle wird über den rechten JoyStick des Gamepads auf der X-Achse gesteuert.	RudderAngle des Schiffs erhöht sich auf + 20°	Erfolgreich
TS_003_5	Notfallsteuerung muss gestartet sein. Gamepad muss verbunden sein. Verbindung zum Schiff muss vorhanden sein.	RudderAngle steuern	-2.000	RudderAngle wird über den rechten JoyStick des Gamepads auf der X-Achse gesteuert.	RudderAngle des Schiffs verringert sich auf - 20°	Erfolgreich
TS_003_6	Notfallsteuerung muss gestartet sein. Gamepad muss verbunden sein. Verbindung zum Schiff muss vorhanden sein.	RudderAngle steuern		Rechter JoyStick wird in die neutrale Stellung der X- und Y-Achse belassen.	RudderAngle des Schiffs bleibt konstant bei 0. Schiff befindet sich im Leerlauf.	Erfolgreich
TS_003_7	Notfallsteuerung muss gestartet sein. Gamepad muss verbunden sein. Verbindung zum Schiff muss vorhanden sein.	Exploratives Steuern des Schiffs	Variierung des RudderAngles und des RPM-Wertes	Rechter JoyStick wird für die Einstellung der X- und Y-Achse verwendet	Das Schiff übernimmt sowohl RudderAngle, als auch RPM.	Erfolgreich

Testfall TF_001_004
Verbindung zum Schiff abbauen

Testschritt	Vorbedingung	Aktion	Eingabedaten / Testdaten	Testschrittbeschreibung	Erwartetes Ergebnis	Testergebnis
TS_004_1	Notfallsteuerung muss gestartet sein. Gamepad muss verbunden sein. Verbindung zum Schiff muss vorhanden sein.	Verbindung abbauen		Über den Button "BACK" wird die Verbindung zum Gamepad abgebaut.	Verbindung zu dem Schiff ist abgebaut. Die Notfallsteuerung sendet keine zyklischen Steuerdaten mehr.	Erfolgreich

Parameter zur Konfiguration

VPN-Verbindung zum Schiff		
Parameter	**Werte**	**Bemerkung**
IP-Adresse	Nicht verfügbar	Öffentliche IP-Adresse der Schiffsrouters
Benutzername	Nicht verfügbar	Konfigurierter Benutzername im Schiffsrouter
Passwort	Nicht verfügbar	Konfiguriertes Passwort im Schiffsrouter

Verbindung zum Raspberry PI		
Parameter	**Werte**	**Bemerkung**
UDP-Port	54345	Der Port wird vom Raspberry PI auf eingehende Nachrichten überwacht.
IP	192.168.1.x	Es handelt sich um eine dynamische Klasse C IP-Adresse

Testmanagement
Auswertung

	Anzahl
Gesamtanzahl gültiger Testfälle:	13
Nicht durchgeführt (Offen/In Arbeit):	0
Nicht durchgeführt (Nicht ausführbar):	1
Durchgeführt (Ohne Fehler):	12
Durchgeführt (Mit Fehlern):	0
Bemerkungen:	Der Testfall TF_001_001 und Testschritt TS_001_1 konnte aufgrund fehlender VPN-Konfigurationsdaten nicht ausgeführt werden.

A.3 Testdrehbuch TDB_002_EPD_Shore

Testdrehbuch TDB_002_Schiffssteuerung_EPD_Shore

Informationen zum Test
Version: 3.2-SNAPSHOT
Tester: Maximilian Wappler
Testdatum: 19.09.2017

Testfalllegende
Mögliche Werte für die Attribute der Testfälle:
Priorität: Hoch, Mittel, Niedrig, Offen
Testergebnis: Fehlerhaft, Erfolgreich, Nicht ausführbar, In Ausführung, Offen, Gestrichen
Komponenten: EPD Shore (MATE:Shore), EPD Ship (MATE:Ship), RabbitMQ (MATE:Server), RabbitMQ (Testbed), Controller, Polymorphic Interface (MATE:Ship), Polymorphic Interface (Testbed), Navibox, Backup-Steuerung

Testsystem
Prozessor: Intel Core i5-6300U CPU 2.40GHz
Arbeitsspeicher: 8GB
Systemtyp (32 bit oder 64 bit): 64 bit
Betriebssystem: Windows 7

Übersicht Testfälle

Testfall-ID	Testfallbezeichnung	Testfall Kurzbeschreibung	Vorbedingungen	Priorität	Komponente
TF_002_001	Verbindung zum Schiff aufbauen	Um eine Verbindung zum Schiff aufbauen zu können, ist es notwendig einige Parameter anzugeben.	Internetverbindung; RabbitMQ-Server auf dem Duemmer-Server muss gestartet sein;	Hoch	EPD Shore (MATE:Shore)
TF_002_002	Schiff über Virtual Handels steuern	Die Geschwindigkeit sowie Ruderstellung des Schiffes kann über die Virtual Handels gesteuert werden.	Die EPD muss gestartet sein; Es muss eine Verbindung zum Schiff aufgebaut sein;	Hoch	EPD Shore (MATE:Shore)
TF_002_003	Schiffsdaten darstellen	In der EPD können diverse Schiffsspezifische Daten von bspw. Sensoren dargestellt werden.	Die EPD muss gestartet sein; Es muss eine Verbindung zum Schiff aufgebaut sein;	Hoch	EPD Shore (MATE:Shore)
TF_002_004	Schiff eine Route vorgeben	In der EPD wird eine Route erstellt und an das Schiff gesendet.	Die EPD muss gestartet sein; Es muss eine Verbindung zum Schiff aufgebaut sein;	Hoch	EPD Shore (MATE:Shore)

TF_002_001
Verbindung zum Schiff aufbauen

Testschritt	Vorbedingung	Aktion	Eingabedaten / Testdaten	Testschrittbeschreibung	Erwartetes Ergebnis	Testergebnis
TS_001_1		EPD-Shore starten			EPD-Shore wird fehlerfrei gestartet;	Erfolgreich
TS_001_2	EPD-Shore muss korrekt gestartet sein	Vorkonfiguration vornehmen		Um eine Vorkonfiguration vorzunehmen, muss über die EPD-Shore Benutzeroberfläche der Menüpunkt "Files/Preferences" aufgerufen werden.	Es öffnet sich das Setup-Fenster der EPD-Shore;	Erfolgreich
TS_001_3	EPD Shore muss gestartet sein; RabbitMQ-Server auf dem Duemmer-Server muss gestartet sein;	Verbindungsdaten angeben		Es müssen alle Verbindungsdaten für die Verbindung der EPD-Shore zum RabbitMQ-Server angegeben werden und mit Accept bestätigt werden.	Es besteht eine Verbindung zum duemmer-Server. Status-LED im Informationsbereich der EPD ist auf "grün".	Erfolgreich

TF_002_002
Schiff über Virtual Handels steuern

Testschritt	Vorbedingung	Aktion	Eingabedaten / Testdaten	Testschrittbeschreibung	Erwartetes Ergebnis	Testergebnis
TS_002_1	EPD-Shore ist gestartet; Verbindung zum duemmer-Server ist aufgebaut;	Virtual Handels starten		Um die Virtual Handels über Mauseingaben steuern zu können, müssen zunächst die Virtual Handels über den Menüpunkt "Plugins/Virtual Handels" aufgerufen werden.	Es öffnet sich das Virtual Handels Fenster;	Erfolgreich
TS_002_2		RPM steuern	100%	RPM wird über das RPM-Control bis auf 100% gestellt.	Innerhalb der Virtual Handels erhöht sich der Wert "Engine RPM" auf bis zu 2.000 RPM. Der RPM-Kontrollbereich erhöht sich entsprechend der am Virtual Handels durchgeführten Eingaben.	Erfolgreich
TS_002_3		RPM steuern	-100%	RPM wird über das RPM-Control bis auf -100% gestellt.	Innerhalb der Virtual Handels verringert sich der Wert "Engine RPM" auf bis zu -2.000 RPM. Der RPM-Kontrollbereich verringert sich entsprechend der am Virtual Handels durchgeführten Eingaben.	Erfolgreich
TS_002_4		RPM steuern	0%	RPM wird über das RPM-Control bis auf 0% gestellt.	Innerhalb der Virtual Handels verringert/erhöht sich der Wert "Engine RPM" auf bis zu 0 RPM. Der RPM-Kontrollbereich verringert/erhöht sich entsprechend der am Virtual Handels durchgeführten Eingaben.	Erfolgreich
TS_002_5		RPM steuern	explorativ	RPM wird über das RPM-Control auf unterschiedliche Werte innerhalb des maximalen positiven und negativen Bereichs gestellt.	Innerhalb der Virtual Handels verringert/erhöht sich der Wert "Engine RPM" je nach getätigter Eingabe. Der RPM-Kontrollbereich verringert/erhöht sich entsprechend der am Virtual Handels durchgeführten Eingaben.	Erfolgreich
TS_002_6		RudderAngle steuern	+ 20°	RudderAngle wird über den "Right"-Button des GamePads bis auf 20% gestellt.	Innerhalb der Virtual Handels erhöht sich der Wert "Rudder Angle Indicator" auf bis zu 20°. Der Wert Heading verändert sich je nach eingegebenem RudderAngle.	Nicht ausführbar
TS_002_7		RudderAngle steuern	-20°	RudderAngle wird über das RudderAngle-Control bis auf -20% gestellt.	Innerhalb der Virtual Handels erhöht sich der Wert "Rudder Angle Indicator" auf bis zu -20°. Der Wert Heading verändert sich je nach eingegebenem RudderAngle.	Erfolgreich
TS_002_8		RudderAngle steuern	0°	RudderAngle wird über das RudderAngle-Control bis auf 0% gestellt.	Innerhalb der Virtual Handels erhöht sich der Wert "Rudder Angle Indicator" auf 0°. Der Wert Heading verändert sich je nach eingegebenem RudderAngle.	Erfolgreich
TS_002_9		RudderAngle steuern	explorativ	RudderAngle wird über das RudderAngle-Control innerhalb des maximalen positiven und negativen Bereichs gestellt.	Innerhalb der Virtual Handels erhöht/verringert sich der Wert "Rudder Angle Indicator" entsprechend der Eingaben. Der Wert Heading verändert sich je nach eingegebenem RudderAngle.	Erfolgreich

TF_002_003
Schiffsdaten darstellen

Testschritt	Vorbedingung	Aktion	Eingabedaten / Testdaten	Testschrittbeschreibung	Erwartetes Ergebnis	Testergebnis
TS_003_1	EPD-Shore ist gestartet; Verbindung zum duemmer-Server ist hergestellt; Virtual Handels ist gestartet;	Sensordaten aktualisieren	explorativ	RPM und RuddeAngle werden erhöht/verringert, um zu prüfen, wie sich die Sensordaten verhalten.	Sensordaten verhalten sich wie erwartet. Eigenes Schiff wird dargestellt.	Erfolgreich

TF_002_004
Schiff eine Route vorgeben

Testschritt	Vorbedingung	Aktion	Eingabedaten / Testdaten	Testschrittbeschreibung	Erwartetes Ergebnis	Testergebnis
TS_004_1	EPD-Shore ist gestartet; Verbindung zum duemmer-Server ist aufgebaut;	Wegpunkte setzen		In der EPD Shore muss in der Toolbar der Button "Add Route" gewählt werden. Anschließend müssen mehrere Wegpunkte auf der Karte gesetzt werden.	Es werden fortlaufend nummerierte Wegpunkte gesetzt.	Erfolgreich
TS_004_2	Wegpunkte sind gesetzt.	Route einsehen	Diverse Wegpunkte in der EPD.	Exploratiiver Test: Die Route muss im Route Manager betrachtet werden und die Werte müssen mit denen der angelegten Route/Wegpunkte übereinstimmen.	Angaben im Route Manager und die zuvor angelegte Route/Wegpunkte müssen übereinstimmen.	Erfolgreich
TS_004_3	Wegpunkte sind gesetzt.	Wegpunkte löschen		Mit einem Rechtsklick auf einzelne Wegpunkte auswählen und "Delete Waypoint" auswählen.	Erster Wegpunkt: Der erste Wegpunkt ist gelöscht. Der darauffolgende ist anschließend als erster Wegpunkt markiert. Letzter Wegpunkt: Der letzte Wegpunkt ist gelöscht. Mittendrin: wird ein in der Route liegender Wegpunkt gelöscht, muss eine Verbindung zwischen vorherigen und nachfolgenden Wegpunkt dargestellt werden.	Erfolgreich
TS_004_4	Wegpunkte sind gesetzt.	Route löschen	Diverse Wegpunkte in der EPD.	Jeden Wegpunkt auf der Karte über Rechtsklick "Delete Waypoint" löschen.	Mit dem Löschen des vorletzten Wegpunkts, muss die Route komplett gelöscht werden.	Erfolgreich
TS_004_5	Wegpunkte sind gesetzt.	Route löschen	Diverse Wegpunkte in der EPD.	Angelegte Route im Route Manager auswählen und über die Schaltfläche "Delete" löschen.	Route ist nicht mehr im Route Manager bzw. in der Karte vorhanden.	Erfolgreich
TS_004_6	Route ist angelegt.	Autopilot aktivieren	Route "Neue Route" ist vorhanden.	Autopilot wird in den virtual Handels aktiviert.	Schiff fährt selbstständig an der Trajektorie des Schiffs zum nächsten Wegpunkt entlang, bis der letzte Wegpunkt erreicht ist.	Erfolgreich
TS_004_7	Autopilot ist aktiviert.	Autopilot deaktivieren		Autopilot wird in den virtual Handels deaktiviert.	Schiff bleibt stehen.	Fehlgeschlagen

Testmanagement
Auswertung

	Anzahl
Gesamtanzahl gültiger Testfälle:	20
Nicht durchgeführt (Offen/In Arbeit):	0
Nicht durchgeführt (Nicht ausführbar):	1
Durchgeführt (Ohne Fehler):	18
Durchgeführt (Mit Fehlern):	1
Bemerkungen:	Der Testfall TF_002_004 und Testschritt TS_004_7 ist fehlgeschlagen. Der Testfall TF_002_002 und Testschritt TS_004_6 konnte nicht durchgeführt werden.

A.4 Sensoren im Testbed

Sensor Typ	**Nachricht**	**Bezeichnung**	**Seriennummer**
AIS	AIS Position Report	NAIS-400	421000203300 58
AIS	AIS Static Voyage Report	NAIS-400	421000203300 58
DGPS Compass	Speed/Course over Ground	NNN-21	KF34332
DGPS Compass	Heading	NNN-21	KF34332
DGPS Compass	Rate of Turn	NNN-21	KF34332
GPS	GPS Location	NNN-21	KF34332
IMU	IMU	Anlagennr. 833567	1600272
Radar	Radar Track	Navico broadband 4G Radar	000-10417-001
ShipEchoSpeedTemp	Speed through Water	Zuse	Nicht bekannt
ShipEchoSpeedTemp	Water Depth	Zuse	Nicht bekannt
ShipEngine	Engine RPM	D4-225	92200190749
ShipEngine	Rudder Angle	D4-225	92200190749
Windsensor	Windspeed and direction	advanSea S400 wind masthead	90-80-006-000

Tabelle 42: Sensoren im Testbed

A.5 Testablauf

Die folgende Tabelle zeigt den zeitlichen Ablauf sowie die notwendigen Schritte für die Ausführung des Akzeptanztests in Wilhelmshaven am 15.09.2017.

Zeit	**Testfall und -schritt**	**Beschreibung**
8:00 Uhr		Abfahrt in Oldenburg am OFFIS
9:00 Uhr		Ankunft in Wilhelmshaven, Vorbesprechung
9:30 Uhr	TF1	Beginn Testfall 1 - EPD
9:30 Uhr	TF1 Schritt 0.	Aufbau / Vorbereitung
9:50 Uhr	TF1 Schritt 1.	Daten empfangen in Fahrt über Grund
10:20 Uhr	TF1 Schritt 2.	Daten senden - manuelle Steuerung
10:50 Uhr	TF1 Schritt 3.	Daten senden - Autopilot
11:40 Uhr		Mittagspause
12:30 Uhr	TF2	Beginn Testfall 2 - Backup-Steuerung
12:30 Uhr	TF2 Schritt 0.	Aufbau / Vorbereitung
12:40 Uhr	TF2 Schritt 1.	Verbindung zum Schiff aufbauen I
12:50 Uhr	TF2 Schritt 2.	Schiff steuern
13:10 Uhr	TF2 Schritt 3.	Verbindung zum Schiff abbauen
13:20 Uhr	TF2 Schritt 4.	Verbindung zum Schiff aufbauen II
13:30 Uhr		Kurze Nachbesprechung des Systemtest vor Ort
14:00 Uhr		Abbau in WHV und Abfahrt nach Oldenburg

Übersicht Testdrehbuch 1 - EPD Shore

Dauer	Testschritt	Beschreibung
20 min	TF 0.	Aufbau/ Vorbereitung
	TS 0.1	Internetverbindung über WLAN der NaviBox herstellen
	TS 0.2	EPD starten
	TS 0.3	Polymorphe Schnittstelle auf einem anderen PC starten
	TS 0.4	RabbitMQ-Verbindung testen mit inaktiver Motorsteuerung)
	TS 0.4.1	Werden beliebige AIS-Daten auf der Karte angezeigt?
	TS 0.4.2	Werden alle relevanten Sensordaten dargestellt?
	TS 0.4.3	Wenn beliebige Daten von den VH gesendet werden, kommen diese bei der NaviBox an?

Dauer	Testschritt	Beschreibung
30 min	TF 1.	Daten empfangen in Fahrt über Grund
	TS 1.1	RabbitMQ-Verbindung steht, Status "grün"
	TS 1.2	AIS-Daten werden auf der EPD-Karte angezeigt
	TS 1.3	Die Sensor-Daten der NaviBox werden in den VirtualHandles angezeigt und verändern sich gegebenenfalls
	TS 1.3.1	GPS Location
	TS 1.3.2	Speed through Water
	TS 1.3.3	Speed/Course over Ground
	TS 1.3.4	Engine RPM
	TS 1.3.5	Rudder Angle
	TS 1.3.6	Angular Velocity Vector 3D (IMU)
	TS 1.3.7	Heading
	TS 1.3.8	Rate of Turn
	TS 1.3.9	Windspeed and direction
	TS 1.3.10	Water Depth
	TS 1.3.11	AIS Position Report
	TS 1.3.12	AIS Static Voyage Report
	TS 1.3.13	Radar Track

Dauer	Testschritt	Beschreibung
30 min	TF 2.	Daten senden - manuelle Steuerung
	TS 2.1	RabbitMQ-Verbindung steht, Status "grün"
	TS 2.2	Geschwindigkeit(in Prozent) lässt sich in den VirtualHandles einstellen
	TS 2.3	Rudereinstellung(in Grad) lässt sich in den VirtualHandles einstellen
	TS 2.4	Die Daten kommen auf den RabbitMQ-Instanzen an
	TS 2.4.1	Duemmer
	TS 2.4.2	Wilkinson
	TS 2.4.3	Navibox
	TS 2.5	Boot steuert in die eingestellte Richtung
	TS 2.5.1	In einem angemessenem Zeitrahmen
	TS 2.5.2	[optional] Zeitmessung
	TS 2.6	Boot fährt mit richtiger Geschwindigkeit
	TS 2.6.1	In einem angemessenem Zeitrahmen
	TS 2.6.2	[optional] Zeitmessung

Dauer	Testschritt	Beschreibung
50 min	TF 3.	Daten senden - Autopilot
	TS 3.1	Eine neue Route kann in die EPD eingegeben werden
	TS 3.2	Die neue Route kann in den VirtualHandles ausgewählt werden
	TS 3.3	Eine Richtgeschwindigkeit für die Route kann angegeben werden
	TS 3.4	RabbitMQ-Verbindung steht, Status "grün"
	TS 3.5	Der Autopilot wird gestartet
	TS 3.5.1	Die Route wird in der EPD visuell hervorgehoben
	TS 3.5.2	Die Route wird über RabbitMQ-Duemmer-Instanz verschickt (Kontrolle per Webview)
	TS 3.5.3	Die Pfadplanungskomponente gibt Feedback, ob die Route per RTZ ankommt
	TS 3.5.4	Die Pfadplanungskomponente berechnet via Position des Schiffes & Route welche Wegpunkte aktiv sein sollen
	TS 3.5.5	Die Pfadplanungksomponente gibt Feedback, welche Wegpunkte an das Polymorphe Schnittstelle geschickt werden
	TS 3.5.6	Die aktiven Wegpunkte werden über die RabbitMQ-Duemmer-Instanz verschickt (Kontrolle per Webview)
	TS 3.5.7	Die aktiven Wegpunkte werden in der EPD visuell hervorgehoben
	TS 3.5.8	Die EPD erhält die aktiven Wegpunkte über die RabbitMQ-Duemmer-Instanz
	TS 3.5.9	Die Regelungssystemkomponente erhält die aktiven Wegpunkte
	TS 3.5.10	Die Regelungssystemkomponente berechnet die RPM & Rudder-Angle
	TS 3.5.11	Die Regelungssystemkomponente verschickt die berechneten Daten via Polymorphe Schnittstelle an die RabbitMQ-Wilkinson-Instanz
	TS 3.5.12	Das Boot fährt die Route gemäß eingestellter Richtung ab
	TS 3.5.13	Das Boot fährt die Route gemäß eingestellter Geschwindigkeit ab

Übersicht Testdrehbuch 2 - Notfallsteuerung

Dauer	Schritt	Beschreibung
10 min	TS 0.	Aufbau / Vorbereitung
	TS 0.1	VPN Verbindung einrichten
10 min	TS 1	Verbindung zum Schiff aufbauen I (ohne aktiv sendende EPD)
	TS 1.1	GamePad anschließen
	TS 1.2	Notfallsteuerung Applikation starten
	TS 1.3	Verbindung aufbauen
	TS 1.4	Zyklische Steuerdaten senden
20 min	TS 2	Schiff steuern
	TS 2.1	RPM steuern: + 2.000 RPM, den Motor des Schiffes ansteuern (beschleunigen)
	TS 2.2	RPM steuern: - 2.000 RPM, den Motor des Schiffes ansteuern (abbremsen/ rückwärts beschleunigen)
	TS 2.3	RPM steuern: 0 RPM, den Motor des Schiffes ansteuern (Stillstand befehlen)
	TS 2.4	Rudder steuern: +2.000, das Rudder des Schiffes ansteuern (+20 Grad Winkel)
	TS 2.5	Rudder steuern: -2.000, das Rudder des Schiffes ansteuern (-20 Grad Winkel)
	TS 2.6	Rudder steuern: 0 , das Rudder des Schiffes ansteuern (gerade aus Fahren)
	TS 2.7	Variierung des RudderAngles und des RPM-Wertes, verschiedene nicht genau bestimmte Werte testen
10 min	TS 3	Verbindung zum Schiff abbauen
	TS 3.1	Über den Button "BACK"wird die Verbindung zum GamePad abgebaut.
10 min	TS 4	Verbindung zum Schiff aufbauen II (mit aktiv sendender EPD)
	TS 4.1	GamePad anschließen
	TS 4.2	Backup-Steuerung Applikation starten
	TS 4.3	Verbindung aufbauen
	TS 4.4	Zyklische Steuerdaten senden.

A.6 NMEA2000 XML Beschreibung

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <NMEA2000>
3     <Message PGN="000001" Name="NaviBox RPM Command" Length="4"
          Priority_Default="2">
4         <Field Name="RPM" BitOffset="0" BitLength="32" Format="Rate of rotation"
              Unit="int32_t" Scale="0.25"
5                 Description="Rate of Rotation, from -16,383 to 16,383 RPM"/>
6     </Message>
7 </NMEA2000>
```

Auflistung 10: Selbst erstellte, standardkonforme NMEA2000 PGN zum Versenden einer gewünschten RPM

A.7 Konfigurationsdatei der Verteilungsplattform

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration
    xmlns="http://polymorphic-interface.mate.uni-oldenburg.de/configuration"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://polymorphic-interface.mate.uni-oldenburg.de/configuration
        /de/uni_oldenburg/mate/polymorphic_interface/server/configuration/configuration.xsd">
    <models directory="./schemas">
        <definitions>
            <definition id="NMEA0183" schemaFile="nmea0183/nmea0183.xsd" />
            <definition id="NMEA2000" schemaFile="nmea2000/nmea2000.xsd" />
            <definition id="RTZ" schemaFile="rtz/RTZ-Schema-version-1-0.xsd" />
        </definitions>
        <transformations>
            <transformation source="NMEA2000" target="NMEA0183"
                xsltFile="transformations/nmea2000-to-nmea0183.xsl" />
            <transformation source="RTZ" target="NMEA2000"
                xsltFile="transformations/rtz-to-nmea2000.xsl" />
        </transformations>
    </models>
    <connectors>
        <resources directory="./connectors">
            <jar file="http_adapters-1.0.0.jar" module="de.uni_oldenburg.mate.
                polymorphic_interface.http_adapters.Module" />
            <jar file="rabbitmq_adapters-1.5.0.jar" module="de.uni_oldenburg.mate.
                polymorphic_interface.rabbitmq_adapters.Module" />
            <jar file="xml_string_translators-1.1.jar" module="de.uni_oldenburg.mate.
                polymorphic_interface.xml_string_translators.Module" />
            <jar file="udp_adapters-1.2.1.jar" module="de.uni_oldenburg.mate.
                polymorphic_interface.udp_adapters.Module" />
            <jar file="nmea0183_translators-2.0.0.jar" module="de.uni_oldenburg.mate.
                polymorphic_interface.nmea0183_translators.Module" />
            <jar file="nmea2000_translator-2.6.0.jar" module="de.uni_oldenburg.mate.
                polymorphic_interface.nmea2000_translator.Module" />
        </resources>
        <instances>
            <adapters>
                <input id="Rabbitmq-Input-Adapter-Mate-NMEA" class="de.uni_oldenburg.
                    mate.polymorphic_interface.rabbitmq_adapters.RabbitMQInputAdapter">
                    <property key="username" value="pgmate" />
                    <property key="password" value="JQW9hwZzgu7KPHu" />
                    <property key="host" value="duemmer.informatik.uni-oldenburg.de" />
                    <property key="port" value="5671" />
```

```
32                  <property key="queue_name" value="zuse-target-state" />
33                  <property key="x-message-ttl" value="100" />
34                  <property key="durable" value="true" />
35                  <property key="use_tls" value="true" />
36                  <property key="filter_empty_messages" value="true" />
37              </input>
38              <input id="Rabbitmq-Input-Adapter-Mate-RTZ" class="de.uni_oldenburg.
                    mate.polymorphic_interface.rabbitmq_adapters.RabbitMQInputAdapter">
39                  <property key="username" value="pgmate" />
40                  <property key="password" value="JQW9hwZzgu7KPHu" />
41                  <property key="host" value="duemmer.informatik.uni-oldenburg.de" />
42                  <property key="port" value="5671" />
43                  <property key="queue_name" value="route-target-state" />
44                  <property key="x-message-ttl" value="300000" />
45                  <property key="durable" value="true" />
46                  <property key="use_tls" value="true" />
47                  <property key="filter_empty_messages" value="true" />
48              </input>
49              <input id="Udp-Input-Adapter-Controller" class="de.uni_oldenburg.mate.
                    polymorphic_interface.udp_adapters.UdpInputAdapter">
50                  <property key="port" value="8002" />
51                  <property key="buffer_size" value="2048" />
52              </input>
53              <input id="Udp-Input-Adapter-Polymorphic-Interface" class="de.
                    uni_oldenburg.mate.polymorphic_interface.udp_adapters.
                    UdpInputAdapter">
54                  <property key="port" value="2323" />
55                  <property key="buffer_size" value="2048" />
56              </input>
57              <input id="Http-Input-Adapter" class="de.uni_oldenburg.mate.
                    polymorphic_interface.http_adapters.HttpInputAdapter">
58                  <property key="host" value="127.0.0.1" />
59                  <property key="port" value="4243" />
60                  <property key="request_target" value="/waypoints" />
61              </input>
62              <output id="Http-Output-Adapter-Route" class="de.uni_oldenburg.mate.
                    polymorphic_interface.http_adapters.HttpOutputAdapter">
63                  <property key="address" value="127.0.0.1" />
64                  <property key="port" value="4242" />
65                  <property key="request_target" value="/route" />
66                  <property key="content_type_header" value="application/xml;
                        charset=utf-8" />
67              </output>
68              <output id="Http-Output-Adapter-Position" class="de.uni_oldenburg.mate.
                    polymorphic_interface.http_adapters.HttpOutputAdapter">
```

```
69                <property key="address" value="127.0.0.1" />
70                <property key="port" value="4242" />
71                <property key="request_target" value="/position" />
72                <property key="content_type_header" value="text/plain; charset=utf-8" />
73            </output>
74            <output id="Xml-File-Output-Adapter" class="de.uni_oldenburg.mate.
                  polymorphic_interface.file_adapters.FileOutputAdapter">
75                <property key="output_directory" value="./tmp/output/xml" />
76                <property key="output_file_suffix" value=".xml" />
77            </output>
78            <output id="Byte-File-Output-Adapter" class="de.uni_oldenburg.mate.
                  polymorphic_interface.file_adapters.FileOutputAdapter">
79                <property key="output_directory" value="./tmp/output/byte" />
80                <property key="output_file_prefix" value="unchanged_input_" />
81            </output>
82            <output id="Rabbitmq-Output-Adapter-Mate-NMEA" class="de.
                  uni_oldenburg.mate.polymorphic_interface.rabbitmq_adapters.
                  RabbitMQOutputAdapter">
83                <property key="username" value="pgmate" />
84                <property key="password" value="JQW9hwZzgu7KPHu" />
85                <property key="host" value="duemmer.informatik.uni-oldenburg.de" />
86                <property key="port" value="5671" />
87                <property key="exchange_name" value="zuse-current-exchange" />
88                <property key="exchange_type" value="direct" />
89                <property key="exchange_routing_key" value="zuse-current-state" />
90                <property key="exchange_durable" value="true" />
91                <property key="use_tls" value="true" />
92            </output>
93            <output id="Rabbitmq-Output-Adapter-Mate-RTZ" class="de.uni_oldenburg.
                  mate.polymorphic_interface.rabbitmq_adapters.RabbitMQOutputAdapter">
94                <property key="username" value="pgmate" />
95                <property key="password" value="JQW9hwZzgu7KPHu" />
96                <property key="host" value="duemmer.informatik.uni-oldenburg.de" />
97                <property key="port" value="5671" />
98                <property key="exchange_name" value="route-current-exchange" />
99                <property key="exchange_type" value="direct" />
100               <property key="exchange_routing_key" value="route-current-state" />
101               <property key="exchange_durable" value="true" />
102               <property key="use_tls" value="true" />
103           </output>
104           <output id="Udp-Output-Adapter-Controller" class="de.uni_oldenburg.mate.
                  polymorphic_interface.udp_adapters.UdpOutputAdapter">
105               <property key="port" value="8001" />
106               <property key="address" value="10.18.13.192" />
107           </output>
```

```
108                 <output id="Udp-Output-Adapter-Polymorphic-Interface" class="de.
                        uni_oldenburg.mate.polymorphic_interface.udp_adapters.
                        UdpOutputAdapter">
109                     <property key="port" value="2324" />
110                     <property key="address" value="127.0.0.1" />
111                 </output>
112             </adapters>
113             <translators>
114                 <input id="Bytes-to-XML" class="de.uni_oldenburg.mate.
                        polymorphic_interface.xml_string_translators.XmlStringInputTranslator">
115                     <property key="charset" value="UTF-8" />
116                 </input>
117                 <input id="NMEA2000-to-XML" class="de.uni_oldenburg.mate.
                        polymorphic_interface.nmea2000_translator.NMEA2000InputTranslator">
118                     <property key="XMLFile"
                            value="schemas/nmea2000/nmea2000-description.xml" />
119                 </input>
120                 <output id="XML-to-Bytes" class="de.uni_oldenburg.mate.
                        polymorphic_interface.xml_string_translators.XmlStringOutputTranslator">
121                     <property key="charset" value="UTF-8" />
122                 </output>
123                 <output id="XML-to-NMEA0183" class="de.uni_oldenburg.mate.
                        polymorphic_interface.nmea0183_translators.NMEA0183OutputTranslator"
                        />
124                 <output id="XML-to-NMEA2000" class="de.uni_oldenburg.mate.
                        polymorphic_interface.nmea2000_translator.NMEA2000OutputTranslator">
125                     <property key="XMLFile"
                            value="schemas/nmea2000/nmea2000-description.xml" />
126                 </output>
127             </translators>
128         </instances>
129     </connectors>
130     <pipes>
131         <pipe id="Polymorphic-Interface-to-RabbitMQ-Mate-NMEA">
132             <adapters>
133                 <input id="Udp-Input-Adapter-Polymorphic-Interface" />
134                 <output id="Rabbitmq-Output-Adapter-Mate-NMEA" />
135             </adapters>
136             <translators>
137                 <input id="NMEA2000-to-XML" />
138                 <output id="XML-to-NMEA0183" />
139             </translators>
140             <steps>
141                 <step model="NMEA2000" />
142                 <step model="NMEA0183" />
```

```
143             </steps>
144         </pipe>
145         <pipe id="RabbitMQ-Mate-NMEA-to-Controller">
146             <adapters>
147                 <input id="Rabbitmq-Input-Adapter-Mate-NMEA" />
148                 <output id="Udp-Output-Adapter-Controller" />
149             </adapters>
150             <translators>
151                 <input id="NMEA2000-to-XML" />
152                 <output id="XML-to-NMEA2000" />
153             </translators>
154             <steps>
155                 <step model="NMEA2000" />
156             </steps>
157         </pipe>
158         <pipe id="Controller-to-Polymorphic-Interface">
159             <adapters>
160                 <input id="Udp-Input-Adapter-Controller" />
161                 <output id="Udp-Output-Adapter-Polymorphic-Interface" />
162             </adapters>
163             <translators>
164                 <input id="NMEA2000-to-XML" />
165                 <output id="XML-to-NMEA2000" />
166             </translators>
167             <steps>
168                 <step model="NMEA2000" />
169             </steps>
170         </pipe>
171     <!-- Direkte Verbindung von EPD zur NaviBox statt ueber den Controller: -->
172         <!--
173         <pipe id="RabbitMQ-Mate-NMEA-to-Polymorphic-Interface">
174             <adapters>
175                 <input id="Rabbitmq-Input-Adapter-Mate-NMEA" />
176                 <output id="Udp-Output-Adapter-Polymorphic-Interface" />
177             </adapters>
178             <translators>
179                 <input id="NMEA2000-to-XML" />
180                 <output id="XML-to-NMEA2000" />
181             </translators>
182             <steps>
183                 <step model="NMEA2000" />
184             </steps>
```

```
185         </pipe>
186         −−>
187         <pipe id="Polymorphic−Interface−to−Path−Planning">
188             <adapters>
189                 <input id="Udp−Input−Adapter−Polymorphic−Interface" />
190                 <output id="Http−Output−Adapter−Position" />
191             </adapters>
192             <translators>
193                 <input id="NMEA2000−to−XML" />
194                 <output id="XML−to−NMEA0183" />
195             </translators>
196             <steps>
197                 <step model="NMEA2000" />
198                 <step model="NMEA0183" />
199             </steps>
200         </pipe>
201         <pipe id="Path−Planning−to−Controller">
202             <adapters>
203                 <input id="Http−Input−Adapter" />
204                 <output id="Udp−Output−Adapter−Controller" />
205             </adapters>
206             <translators>
207                 <input id="Bytes−to−XML" />
208                 <output id="XML−to−NMEA2000" />
209             </translators>
210             <steps>
211                 <step model="RTZ" />
212                 <step model="NMEA2000" />
213             </steps>
214         </pipe>
215         <pipe id="Path−Planning−to−RabbitMQ−Mate−RTZ">
216             <adapters>
217                 <input id="Http−Input−Adapter" />
218                 <output id="Rabbitmq−Output−Adapter−Mate−RTZ" />
219             </adapters>
220             <translators>
221                 <input id="Bytes−to−XML" />
222                 <output id="XML−to−Bytes" />
223             </translators>
224             <steps>
225                 <step model="RTZ" />
226             </steps>
```

```
227             </pipe>
228             <pipe id="RabbitMQ-Mate-RTZ-to-Path-Planning">
229                 <adapters>
230                     <input id="Rabbitmq-Input-Adapter-Mate-RTZ" />
231                     <output id="Http-Output-Adapter-Route" />
232                 </adapters>
233                 <translators>
234                     <input id="Bytes-to-XML" />
235                     <output id="XML-to-Bytes" />
236                 </translators>
237                 <steps>
238                     <step model="RTZ" />
239                 </steps>
240             </pipe>
241         </pipes>
242 </configuration>
```

Auflistung 11: Konfigurationsdatei der Verteilungsplattform

A.8 EPD VesselConnection Constants.java

```
1  package de.uni_oldenburg.mate.epd.vessel_connection.common;
2
3  public class Constants {
4
5      /**
6       * The username to authenticate with.
7       */
8      public static final String USERNAME = "username";
9
10     /**
11      * The password to authenticate with.
12      */
13     public static final String PASSWORD = "password";
14
15     /**
16      * The host where to subscribe at.
17      */
18     public static final String HOST = "host";
19
20     /**
21      * Constants for vessel-data-exchange with Zuse over RabbitMQ
22      */
23     public static final int DEFAULT_PORT = 5672;
24     public static final String DEFAULT_EXCHANGE_NAME = "zuse-target-exchange";
25     public static final String DEFAULT_EXCHANGE_TYPE = "direct";
26     public static final String DEFAULT_EXCHANGE_ROUTING_KEY = "zuse-target-state";
27     public static final String DEFAULT_QUEUE_NAME = "zuse-current-state";
28     public static final int DEFAULT_ARGS_TTL = 10;
29     public static final int DEFAULT_BASIC_QOS = 20;
30     public static final boolean DEFAULT_DURABLE = true;
31
32     /**
33      * Constants for route-exchange with Path-Planning over RabbitMQ
34      */
35     public static final int DEFAULT_PORT_RTZ = 5672;
36     public static final String DEFAULT_EXCHANGE_NAME_RTZ = "route-target-exchange";
37     public static final String DEFAULT_EXCHANGE_TYPE_RTZ = "direct";
38     public static final String DEFAULT_EXCHANGE_ROUTING_KEY_RTZ =
           "route-target-state";
39     public static final String DEFAULT_QUEUE_NAME_RTZ = "route-current-state";
```

```
40      public static final int DEFAULT_ARGS_TTL_RTZ = 300000;
41      public static final int DEFAULT_BASIC_QOS_RTZ = 20;
42      public static final boolean DEFAULT_DURABLE_RTZ = true;
43  }
```

Auflistung 12: EPD VesselConnection Constants.java

Glossar

Nachfolgend sind noch einmal wesentliche Begriffe dieser Arbeit zusammengefasst und erläutert. Eine ausführliche Erklärung findet sich jeweils in den einführenden Abschnitten sowie der jeweils darin angegebenen Literatur. Das im Rahmen des Glossars verwendete Symbol „∼ “ bezieht sich jeweils auf den im Einzelnen vorgestellten Begriffe und das Symbol „↑“ verweist auf einen ebenfalls innerhalb dieses Glossars erklärten Begriffs.

Abnahmetest Ein ∼ ist ein formaler Test hinsichtlich der Benutzeranforderungen und -bedürfnisse und wird durch den Kunden durchgeführt.

AIS ∼ steht für Automatic Identification System und bezeichnet ein Funksystem zum Austausch von Navigations- und Schiffsdaten in der Schifffahrt. Die Verbesserung der Sicherheit in der Schifffahrt steht dabei im Vordergrund.

Akteure ∼ stellen eine Rolle dar, die einen Benutzer, Benutzergruppen oder andere Systeme widerspiegeln und mit einem Subjekt interagieren.

Akzeptanztest Der ∼ ist ein vom Entwicklungsteam durchgeführter Test, der aus Sicht des Kunden erfolgt. Es handelt sich dabei um eine Vorstufe zum ↑Abnahmetest.

Anforderungsschablone Eine ∼ ist ein Bauplan, der die Struktur eines einzelnen Anforderungssatzes festlegt.

DGPS Bei ∼ handelt es sich um Verfahren zur Positionsbestimmung mit dem Ziel, die Genauigkeit von Systemen wie GPS zu erhöhen.

ECDIS Das ∼ ist ein elektronisches Navigationsinformationssystem und kombiniert dabei Daten aus unterschiedlichen Quellen.

eMaritime, eMir ∼ ist eine Initiative der deutschen Schifffahrtsindustrie, um die Sicherheit und Effizienz im maritimen Transportwesen zu erhöhen, sowie Schifffahrtsunfälle zu verhindern.

EPD-Ship Die ∼ ist eine schiffsseitige Software zur computergestützten Navigation auf See.

EPD-Shore Anders als die ↑EPD-Ship ist die ∼ für den Einsatz an Land und damit für die Überwachung bzw. Steuerung mehrerer Schiffe konzipiert.

e-Navigation Prototype Display, EPD Die ∼ ist die Grundlage verschiedener Forschungsprojekte zur computergestützten Navigation auf See.

Funktionale Anforderungen Eine ∼ ist eine Anforderung bezüglich des Ergebnisses eines Verhaltens, das von einer Funktion des Systems (oder einer Komponente eines Services) bereitgestellt werden soll.

Gantt-Diagramm Ein ∼ ist ein Werkzeug zur Gliederung von Aufgaben in einem Projekt.

Git ∼ ist ein Versionsverwaltungsprogramm und ermöglicht es, verschiedene Versionen einer Datei über eine Versionierung eines Verzeichnisbaums zu erhalten.

GitHub, GitLab ∼ sind serverseitige webbasierte Anwendungen zur Versionsverwaltung und zum Management von Projekten.

Hägis ∼ stellt ein virtuelles Testbed dar, welches eingesetzt wird, um die Sicherheit und Effizienz von Transportsystemen über Simulationen zu verifizieren.

Integrationstest Ein ∼ bezieht sich, anders als ein ↑Modultest, auf eine Gruppe von Komponenten und hat zum Ziel, Schnittstellenfehler zu finden.

LABSKAUS ∼ stellt ein physikalisches Testbed für neue Lösungen sowie Technologien im eNavigation und ↑eMaritime Umfeld dar.

Meilenstein Ein ∼ umfasst eine Projektphase, an deren Ende ein fest definiertes Ergebnis steht.

MMSI Die ∼ (deutsch: Rufnummer des mobilen Seefunkdienstes) ist eine neunstellige Nummer zur Identifikation von See- und Küstenfunkstellen.

Modultest Ein ∼ oder auch Komponententest ist ein Softwaretest, der Fehler in den kleinsten Bausteinen eines Softwaresystems aufdecken soll.

Nichtfunktionale Anforderungen ∼ stellen Eigenschaften und Einschränkungen eines Produkts dar.

NMEA0183 Der ∼ Interface Standard definiert Anforderungen an elektrische Signale, an das Datenübertragungsprotokoll und an die Datenübertragungszeit für verschiedene Sensoren.

NMEA2000 Der ∼ -Standard ist ein serielles Datennetzwerk und erlaubt es mehreren Elektrogeräten, sich auf einem einzigen Kanal miteinander zu verbinden.

OFFIS Das ∼ ist ein AN-Institut der Carl von Ossietzky Universität Oldenburg und stellt in Kooperation mit der Universität die Betreuer für die PG MATE.

PID Der ∼ -Regler besitzt einen proportionalen, eine integrierenden und eine differenzierenden Anteil.

RabbitMQ Bei ∼ handelt es sich um eine Open-Source-Message-Broker-Software, welche das Protokoll AMQP implementiert.

Radar Unter ∼ werden verschiedene Erkennungs- und Ortungsverfahren verstanden.

RTZ ∼ ist ein XML basierter maritimer Standard zum Austausch von Routen.

Stakeholder Ein ∼ eines Systems ist eine Person oder Organisation, welche (direkt oder indirekt) Einfluss auf die Anforderungen des betrachteten Systems hat.

System Bei einem ∼ handelt es sich um eine Verbindung von Hardware, Software, Prozessen und Personen, die gemeinsam die Fähigkeit haben, ein bestimmtes Ziel zu erreichen oder bestimmte Eigenschaften auszuprägen.

Systemtest Der ∼ testet Softwarespezifikationen und nimmt die Anforderungen dabei in Bezug.

S-100 ∼ stellt einen modernen hydrografischen Datenstandard dar, welcher ein großes Spektrum an hydrografischen und digitalen Datenquellen unterstützt.

Testgetriebene Entwicklung Kernelement der ∼ ist das konsequente Entwerfen von Softwaretests vor der Implementierung der zu testenden Software.

User-Story Eine ∼ ist eine in Alltagssprache formulierte Softwareanforderung mit einem bestimmten Aufbau.

V-Modell Das ∼ ist ein Vorgehensmodell, welches auf dem ↑Wasserfallmodell basiert. Der linke Pfad beschreibt die aufeinander folgenden Phasen, der rechte Pfad ergänzt diese um passende Tests.

V-Modell XT Das ∼ erlaubt umfangreiche Tailoring-Maßnahmen, um das ↑V-Modell auf die jeweilige Projektsituation anpassen zu können.

Wasserfallmodell Das ∼ ist ein streng lineares Vorgehensmodell zur Softwareentwicklung.

Weekly-Update Im ∼ stellen die Projektgruppenmitglieder kurz in wenigen Sätzen vor, welche Tätigkeiten sie seit dem letzten Treffen im Zusammenhang mit der Projektgruppenarbeit geschafft haben und woran sie als Nächstes arbeiten.

XML ∼ ist eine Auszeichnungssprache zur Erstellung hierarchischer Datenmodelle.

Zuse Die ∼ ist ein Forschungsschiff des ∼ OFFIS und Teil von ↑LABSKAUS.

Abkürzungen

AIS	Automatic Identification System
AMQP	Advanced Message Queuing Protocol
ARPA	Automatic Radar Plotting Aid
CAN	Controller Area Network
DGPS	Differential Global Positioning System
EBL	Electronic Bearing Line
ECDIS	Electronic Chart Display and Information System
EMF	Eclipse Modeling Framework
eMIR	eMaritime Integrated Reference Platform
ENC	Electronic Nautical Chart
enum	Enumeration
EPD	e-Navigation Prototype Displays
GLFW	OpenGL FrameWork
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IHO	International Hydropgraphic Organization
IPv4	Internet Protokoll Version 4
MMSI	Maritime Mobile Service Identity
NMEA0183	National Marine Electronics Association 0183
NMEA2000	National Marine Electronics Association 2000
PGN	Parameter Group Number
PID	Proportional-Integral-Derivative
Radar	Radio detection and ranging
RPM	Revolutions per minute

RTZ	Route plan Exchange Format
SOA	Systemanalyse und -optimierung
UDP	User Datagram Protocol
VRM	Variable Range Marker
XML	Extensible Markup Language
XPath	XML Path Language
XSD	XML Schema
XSLT	Extensible Stylesheet Language Transformations

Abbildungen

Tabellenverzeichnis

Quellcode

Literaturverzeichnis

[AAA+08] Sanjay Aiyagari, Matthew Arrott, Mark Atwell, et al. *AMQP - Advanced Message Queuing Protocol- Protocol Specification*, November 2008. (zuletzt aufgerufen am 03.10.2017).

[Asc16] Rüdiger R. Asche. *Embedded Controller: Grundlagen und praktische Umsetzung für industrielle Anwendungen*. Springer Fachmedien Wiesbaden, 2016. ISBN: 978-3-6581-4850-8.

[Bal09] Helmut Balzert. *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*. Spektrum Akademischer Verlag, Heidelberg, 2009. ISBN: 978-3-8274-2247-7.

[Bal14] Paul Balzer. cbcity.de - Fahrzeugumfeldsensorik: Überblick und Vergleich zwischen Lidar, Radar, Video. (zuletzt aufgerufen am 03.10.2017), July 2014. URL: http://www.cbcity.de/fahrzeugumfeldsensorik-ueberblick-und-vergleich-zwischen-lidar-radar-video.

[BPK15] Hans Brandt-Pook and Rainer Kollmeier. *Softwareentwicklung kompakt und verständlich: wie Softwaresysteme entstehen*. Springer Vieweg, Wiesbaden, 2. auflage edition, 2015. ISBN: 978-3-658-10876-2.

[Bus16] Matthias Busse. Das NMEA2000 Marine Netzwerk, etwas zum Einstieg. (zuletzt aufgerufen am 03.10.2017), November 2016. URL: http://shelvin.de/das-nmea2000-netzwerk-etwas-zum-einstieg.

[BWN14] A. G. Bole, Alan Wall, and Andy Norris. *Radar and ARPA manual: radar, AIS and target tracking for marine radar users*. Butterworth-Heinemann, Oxford, 3. auflage edition, 2014. ISBN: 978-0-08-097752-2.

[Cha14] Ben Chacon, Scott ; Straub. *Pro git*. aPress, 2014.

[CIR] CIRM. Route plan exchange format - RTZ. (zuletzt aufgerufen am 03.10.2017). URL: http://cirm.org/rtz/index.html.

[Com16] NMEA2000 Standards Committee. Nmea2000. appendices a & b - parameter groups (pgns). nmea network messages. (zuletzt aufgerufen am 03.10.2017), May 2016. URL: https://www.nmea.org/content/nmea_standards/nmea_2000_ed3_10.asp.

[Deu15] Bundesnetzagentur Deutschland. Nummernplan für rufzeichen, maritime mobile service identities (mmsi) und automatic transmitter identification system-nummern (atis-nummern) für besondere anwendungen im see- und binnenschifffahrtsfunk. (zuletzt aufgerufen am 03.10.2017), July 2015. URL: https://www.bundesnetzagentur.de/SharedDocs/Downloads/DE/Sachgebiete/Telekommunikation/Unternehmen_Institutionen/Nummerierung/TechnischeNummern/SeeBinnenschifffahrtsfunk/2015_35_NP_besodereAnwendungen.pdf?__blob=publicationFile&v=3.

[Dre12] Hans-Joachim Dreyer. *Technische Mechanik Kinematik und Kinetik*. Springer, 11. auflage edition, 2012. ISBN: 978-3-8348-2239-0.

[Ebe08] Christof Ebert. *Systematisches Requirements-Engineering und Management - Anforderungen ermitteln, spezifizieren, analysieren und verwalten*. Dpunkt-Verlag, Köln, 2008. ISBN: 978-3-898-64546-1.

[eMI17] eMIR. emaritime.de - Overview. (zuletzt aufgerufen am 03.10.2017), September 2017. URL: http://www.emaritime.de.

[FGG+12] Ralf Fahney, Thomas Gartung, Jörg Glunde, Anne Hoffmann, and Uwe Valentini. *Requirements Engineering und Projektmanagement* -. Springer-Verlag, Berlin Heidelberg New York, 2012. ISBN: 978-3-642-29432-7.

[Fos11] Thor I. Fossen. *Handbook of marine craft hydrodynamics and motion control = Vademecum de Navium Motu Contra Aquas et de Motu Gubernando*. Chichester, West Sussex : Wiley, Chichester, West Sussex, 2011.

[Fou17] The Eclipse Foundation. Eclipse modeling framework (emf). (zuletzt aufgerufen am 03.10.2017), September 2017. URL: http://www.eclipse.org/modeling/emf/.

[Gas12] Tom M. Gasser. Rechtsfolgen zunehmender Fahrzeugautomatisierung. Technical report, Bundesanstalt für Straßenwesen, 2012. URL: http://www.bast.de/DE/Publikationen/Foko/Downloads/2012-11.pdf?__blob=publicationFile.

[Glo16] Boris Gloger. *Scrum: Produkte zuverlässig und schnell entwickeln*. Carl Hanser Verlag München, 5. auflage edition, 2016. ISBN: 978-3-446-44836-0.

[Gol15] Daniel Goll, Joachim ; Hommel. *Mit Scrum zum gewünschten System*. Springer Vieweg, 2015.

[Hal14] Gary McLean Hall. *Adaptive code via C#: Agile coding with design patterns and SOLID principles*. Microsoft Press, 2014. ISBN: 978-0-735-68320-4.

[Hei07] Gert Heinrich. *Allgemeine Systemanalyse*. Oldenbourg, 2007. ISBN: 978-3-486-58365-6.

[Hil05] Dietmar Hiller. *Project-Management with Gantt-Charts*. Apache Software Foundation, 2005. (zuletzt aufgerufen am 03.10.2017). URL: https://www.openoffice.org/documentation/HOW_TO/spreadsheet/gantt_pm.pdf.

[Hou07] Davis House. *Ship Handling*. Taylor and Francis, 2007. ISBN: 9781136366574.

[HWLW08] Bernhard Hofmann-Wellenhof, Herbert Lichtenegger, and Elmar Wasle. *GNSS–global navigation satellite systems: GPS, GLONASS, Galileo, and more*. Springer Science & Business Media, 2008. ISBN: 978-3-211-73017-1.

[ICB17] ICBM. Das Leitbild des ICBM. (zuletzt aufgerufen am 03.10.2017), August 2017. URL: https://www.icbm.de/leitbild.

[IEE08] IEEE. IEEE standard for floating-point arithmetic. *IEEE Std. 754-2008*, Seiten 1–70, August 2008. ISBN: 978-0-7381-5752-8.

[Ise10] Rolf Isermann. *Elektronisches Management motorischer Fahrzeugantriebe Elektronik, Modellbildung, Regelung und Diagnose für Verbrennungsmotoren, Getriebe und Elektroantriebe*. Vieweg+Teubner, Wiesbaden, 2010. ISBN: 978-3-8348-0855-4.

[ISO01] ISO International Organization for Standardization. *ISO/IEC 9126 - Software Engineering - Product Quality*, 2001.

[ISO11] ISO International Organization for Standardization. *ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*, 2011.

[Iye03] Pankaj Iyer, Sriram V. & Gupta. *Embedded Realtime Systems Programming*. McGraw-Hill Education (India) Company Limited, 2003. ISBN: 9780070482845.

[Jak15] Walter Jakoby. *Projektmanagement für Ingenieure: ein praxisnahes Lehrbuch für den systematischen Projekterfolg*. Springer Vieweg, 3. auflage edition, 2015. ISBN: 978-3-658-02608-0.

[Kö17] Andreas Kölbl. ECDIS. (zuletzt aufgerufen am 03.10.2017), 2017. URL: http://www.schiffsspotter.de/Sachwortregister/ECDIS.

[Lit13] L. Litz. *Grundlagen der Automatisierungstechnik: Regelungssysteme - Steuerungssysteme - Hybride Systeme*. De Gruyter, 2013. ISBN: 978-3-4867-1981-9.

[LM09] Tobias Lütticke and Christian Meder. Google guice: Framework für dependency injection. (zuletzt aufgerufen am 03.10.2017), January 2009. URL: https://www.heise.de/-227144.

[Mel11] Jim Melton. *Querying XML XQuery, XPath, and SQL/XML in context*. Elsevier Science, 2011. ISBN: 978-1-55860-711-8.

[Mik10] Jerzy Mikulski. *Transport systems telematics: 10th Conference, TST 2010, Katowice - Ustroń, Poland, October 20-23, 2010, selected papers*. Springer Berlin, 2010. ISBN: 978-3-642-16471-2.

[MT04] Michele Missikoff and Francesco Taglino. *An Ontology-based Platform for Semantic Interoperability*, Seiten 617–633. Springer Berlin Heidelberg, 2004. ISBN: 978-3-540-24750-0.

[OFF] OFFIS. offis.de - Kooperierende Mobile Systeme. (zuletzt aufgerufen am 03.10.2017). URL: https://www.offis.de/f-e-bereiche/verkehr/kooperierende-mobile-systeme.html.

[PP16] Daewon Park and Suhyun Park. *Multiple-domain marine data utilization structure for e-navigation*. Springer, 2016.

[PSa] Inc. Pivotal Software. Amqp - Advanced Message Queuing Protocol- Protocol Specification. (zuletzt aufgerufen am 03.10.2017). URL: https://www.rabbitmq.com.

[PSb] Inc. Pivotal Software. Rabbitmq - amqp 0-9-1 model explained. (zuletzt aufgerufen am 03.10.2017). URL: https://www.rabbitmq.com/tutorials/amqp-concepts.html.

[PSc] Inc. Pivotal Software. Rabbitmq - consumer prefetch. (zuletzt aufgerufen am 03.10.2017). URL: https://www.rabbitmq.com/consumer-prefetch.html.

[PSd] Inc. Pivotal Software. Rabbitmq - time-to-live extensions. (zuletzt aufgerufen am 03.10.2017). URL: https://www.rabbitmq.com/ttl.html.

[PVDH+17] P. O. Pedersen, Janus Varmarken, Adam Due Hansen, Jens Tuxen, and David Camre. e-navigation prototype displays. (zuletzt aufgerufen am 03.10.2017), September 2017. URL: https://github.com/dma-enav/EPD.

[Rei16] Konrad Reif. *Sensoren im Kraftfahrzeug*. Springer Vieweg, 3. auflage edition, 2016. ISBN: 978-3-658-11211-0.

[Roc14] Christine Rochange. *Time-Predictable Architectures*. Hoboken : Wiley, Hoboken, 2014.

[Roj09] Mariano E. Rojas. Iho s-100: The new iho hydrographic geospatial standard for marine data and information. Technical report, International Hydrographic Organization (IHO) and the Hydrographic and Oceanographic Service of the Chilean Navy (SHOA), 2009. (zuletzt aufgerufen am 03.10.2017). URL: http://icaci.org/files/documents/ICC_proceedings/ICC2009/html/nonref/3_26.pdf.

[Rou13] Guy Rouleau. Data dependency violation errors and subsystem semantics. (zuletzt aufgerufen am 03.10.2017), February 2013. URL: https://blogs.mathworks.com/simulink/2013/02/04/data-dependency-violation-errors-and-subsystem-semantics.

[RP15] Chris Rupp and Klaus Pohl. *Basiswissen Requirements Engineering - Aus- und Weiterbildung nach IREB-Standard zum Certified Professional for Requirements Engineering Foundation Level*. dpunkt.verlag, Heidelberg, 2015. ISBN: 978-3-864-91674-8.

[RQ12] Chris Rupp and Stefan Queins. *UML 2 glasklar: Praxiswissen für die UML-Modellierung*. Carl Hanser Verlag München, 4. auflage edition, 2012. 978-3-446-43197-3.

[Ryd12] Anders Rydlinger. STM Voyage exchange format and architecture. Technical report, Swedish Maritime Administration, 2012. MONALISA 2.0 project.

[SDW+10] Alexander Schatten, Markus Demolsky, Dietmar Winkler, Stefan Biffl, Erik Gostischa-Franta, and Thomas Östreicher. *Best practice software-engineering: Eine praxiserprobte Zusammenstellung von komponentenorientierten Konzepten, Methoden und Werkzeugen*. Spektrum Akademischer Verlag : Springer e-books, 2010. ISBN: 978-3-8274-2487-7.

[SL04] Andreas Spillner and Tilo Linz. *Basiswissen Softwaretest: Aus- und Weiterbildung zum Certified Tester ; Foundation Level nach ASQF- und ISTQB-Standard*. punkt Verl. Heidelberg, 2. auflage edition, 2004. ISBN: 978-3-898-64256-9.

[SOA17] SOA. Systemanalyse und -optimierung: Willkommen. (zuletzt aufgerufen am 03.10.2017), August 2017. URL: https://www.uni-oldenburg.de/sao.

[TMa] Inc. The MathWorks. Über mathworks. (zuletzt aufgerufen am 03.10.2017). URL: https://de.mathworks.com/company.html?s_tid=hp_ff_a_company1.

[TMb] Inc. The MathWorks. Signal logging with simulink real-time explorer. (zuletzt aufgerufen am 03.10.2017). URL: https://de.mathworks.com/help/xpc/signal-logging-start-here.html.

[TMc] Inc. The MathWorks. Simulation and model-based design. (zuletzt aufgerufen am 03.10.2017). URL: https://de.mathworks.com/products/simulink.html.

[TMd] Inc. The MathWorks. Simulink real-time. build, run, and test real-time applications. (zuletzt aufgerufen am 03.10.2017). URL: https://de.mathworks.com/products/simulink-real-time.html.

[TMe] Inc. The MathWorks. Specify sample time. (zuletzt aufgerufen am 03.10.2017). URL: https://de.mathworks.com/help/simulink/ug/how-to-specify-the-sample-time.html.

[Ver17] Berthold Heinrich VerfasserIn. *Grundlagen Automatisierung : Sensorik, Regelung, Steuerung*. Wiesbaden s.l. : Springer Fachmedien Wiesbaden, 2017.

[WB16] Andrea Windolph and Alexander Blumenau. *Brainstorming Alles, was du für ein perfektes Brainstorming wissen musst*. Books on Demand, 2016. ISBN: 978-3-8423-3773-2.

[Wei13] Adam Weintrit. *Marine navigation and safety of sea transportation: Navigational problems*. CRC Press, 2013. ISBN: 978-1-138-00107-7.

[Wes06] Frank Westphal. *Testgetriebene Entwicklung - mit JUnit & FIT: wie Software änderbar bleibt*. dpunkt.Verl. Heidelberg, 1. auflage edition, 2006. ISBN: 978-3-89864-220-0.

[WG11] Robert Ward and Barrie Greenslade. IHO S-100 - The Universal Hydrographic Data Model. Technical report, IHO Transfer Standards Maintenance and Applications Development Working Group, January 2011. (zuletzt aufgerufen am 03.10.2017). URL: https://www.iho.int/mtg_docs/com_wg/TSMAD/TSMAD_Misc/S-100InfoPaper_FinalJan2011.pdf.

[WM17] Ralf Wirdemann and Johannes Mainusch. *Scrum mit User Stories*. Carl Hanser Verlag München, 3. auflage edition, 2017. ISBN: 978-3-446-45052-3.

[XSt17] XStream. x-stream.github.io - About XStream- Features Typical Uses Known Limitations Getting Started Latest News. (zuletzt aufgerufen am 03.10.2017), September 2017. URL: http://x-stream.github.io.

[ZZ12] Z. Wang Z. Zhang, Y. Li. *Software engineering and knowledge engineering: theory and practice : proceedings of 2009 International Conference on Knowledge Engineering and Software Engineering (KESE 2009)*. Springer Verlag Berlin, 2012. ISBN: 978-3-642-25348-5.