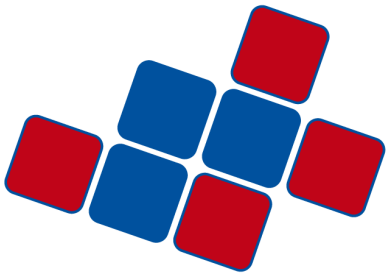


SYSTEMANALYSE UND -OPTIMIERUNG

Carl von Ossietzky Universität Oldenburg

Abschlussdokumentation

Status: 31. März 2016

Betreuer: Prof. Dr.-Ing. Axel Hahn
Dipl. Inform. Sören Schweigert
M.Sc. Volker Gollücke
M.Sc. Carsten Buschmann

Gruppenmitglieder: Ali Akyol
Hauke Bremer
Philipp Gallandt
Tobias Günther
Neele Karbe
Florian Klein
Florian Kuper
Arne Lamm
Sabrina Noster
Matthias Steidel
Shudong Sun
Jonas von Reeken

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Das erhöhte Verkehrsaufkommen auf Schifffahrtswegen durch die zunehmende Anzahl von Passagier- und Containerschiffen hat zur Folge, dass die Verkehrsdichte ansteigt und Seeunfälle zunehmen. Um die Sicherheit zu erhöhen und die Effizienz zu steigern, ist es sinnvoll, das Schiffsverhalten vorab zu analysieren. Eine Möglichkeit den Seeunfällen vorzubeugen, ist die Verbesserung der Kollisionsvermeidung. Fernab entwickelt sich die Technologie des autonomen Fahrens der Schiffe weiter, die die Mensch-Maschine-Interaktion unterstützen soll. Systeme wie diese müssen ebenfalls vor Einsatz auf ihre Funktionalität getestet werden.

Die Projektgruppe Maritime Transportation System Simulation (MTSS) der Carl-von-Ossietzky Universität Oldenburg beschäftgit sich ein Jahr mit dieser Thematik und entwickelt auf Grundlage dieser Kenntnisse eine Simulationsumgebung. Unterstützt wird die Arbeit durch die Abteilung Systemanalyse und -optimierung unter der Leitung von Prof. Dr.-Ing. Axel Hahn.

1.2 Zielsetzung

Die Aufgabe der Projektgruppe MTSS ist es eine Plattform zu entwickeln, die es ermöglicht, den Schiffsverkehr zu analysieren. Es soll eine Simulationsumgebung implementiert werden, um eigene oder bereits vorhandene Verhaltensszenarien der Schiffe zu testen. Das Ziel ist es, Effizienz- und Risikountersuchungen durchzuführen und neue Assistenz- und Sensorsysteme auf ihre Performance zu überprüfen. Zudem sollen Planungs- und Optimierungsalgorithmen getestet werden. Ergänzt wird die Umgebung durch einen Umwelt-Simulator, der die Umwelteinflüsse bereitstellt.

2 Anforderungen

Um sicherzustellen, dass die Erwartungen des Product Owners im Rahmen der Softwareentwicklung erfüllt werden, wurden während der Analyse-Phase Anforderungen in Absprache zwischen der Projektgruppe MTSS und dem Product Owner festgelegt. Die Anforderungen beschreiben die genauen Leistungen, die die Software am Ende des Projekts zu erfüllen hat. Es wird unterschieden zwischen Muss-, Soll- und Kann-Anforderungen.

Anforderungsart	**Beschreibung**
Muss	Diese Anforderungen müssen vollständig umgesetzt werden, um den Entwicklungsauftrag erfolgreich abzuschließen.
Soll	Diese Anforderungen sollten umgesetzt werden, sofern freie Ressourcen zur Verfügung stehen. Sie sind für einen erfolgreichen Abschluss der Entwicklung nicht zwingend notwendig.
Kann	Diese Anforderungen erweitern das Produkt um "Nice-To-Have-Features, die das Entwicklungsteam nach belieben umsetzen kann, sofern Muss- und Soll-Anforderungen dadurch nicht beeinträchtigt werden.

2.1 Allgemeine Anforderungen

Die folgenden grundsätzlichen Anforderungen sollten bei der Implementierung der MTSS berücksichtigt werden:

1. **Erweiterbarkeit**
2. **Framework-Charakter**
3. **Anpassbarkeit**
4. **Wartbarkeit**
5. **Usability**
6. **Korrekte Ergebnisse**
7. **S-100 Konformität**

Auf einzelne der Allgemeinen Anforderungen wird im Folgenden näher eingegangen.

2.1.1 Erweiterbarkeit

Um die allgemeine Anforderung der Erweiterbarkeit zu erfüllen, basiert die Systemarchitektur der MTSS auf einzelnen Komponenten, die über die HLA miteinander kommunizieren und interagieren. Die Erweiterbarkeit ist ein Aspekt, der den Framework-Charakter der Software unterstreicht.

2.1.2 Usability

Die Usability wird insbesondere durch die Editoren der einzelnen Komponenten bewerkstelligt. Sie ermöglichen es, die Konfigurationsdateien komfortabel anzupassen. Generell stellt Usability im Bereich der Softwareentwicklung eine Anforderung hoher Priorität dar, weil sie eine intuitive und reibungslose Bedienbarkeit der Software bewerkstelligt.

2.1.3 S-100 Konformität

Damit die MTSS in die Forschungsprojekte LABSKAUS und HAGGIS integrierbar ist, muss das verwendete Datenmodell auf S-100 Standards basieren.

2.1.4 Korrekte Ergebnisse

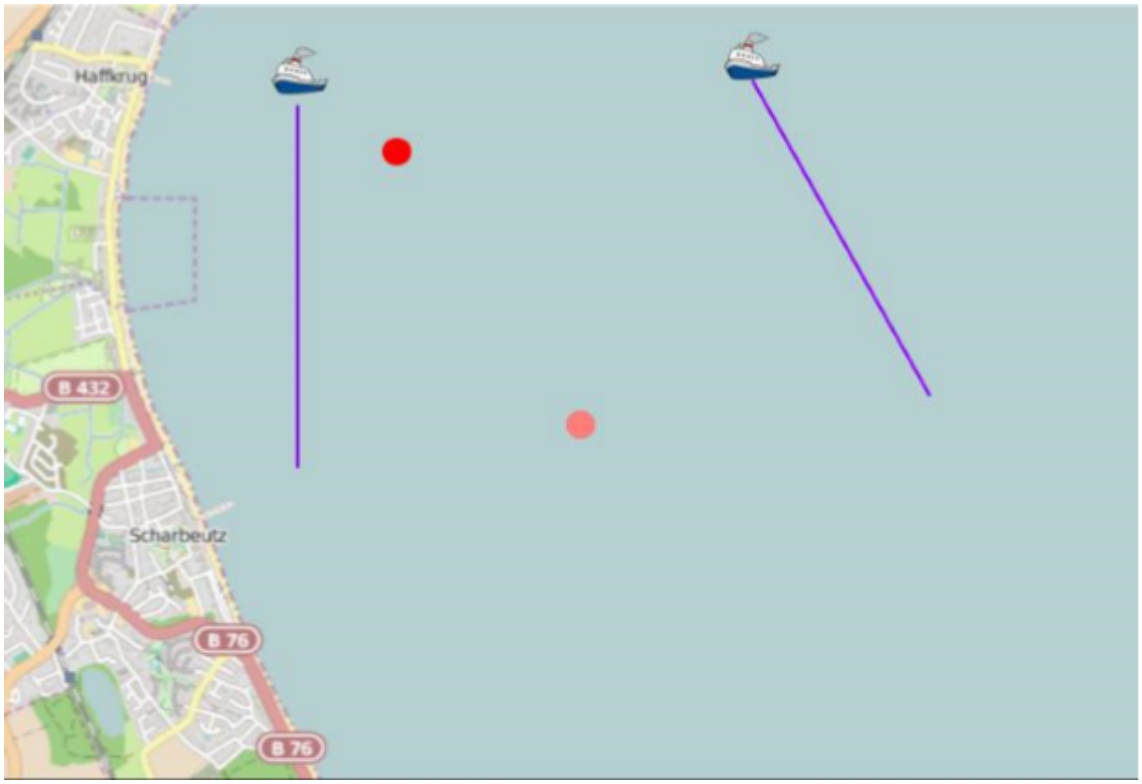
Um die Reproduzierbarkeit einer Simulation zu gewährleisten, müssen verlässlich korrekte Ergebnisse von der Software geliefert werden. Durch Testfälle kann die Korrektheit der Ergebnisse überprüft werden.

2.2 Szenarien

Um die Erfüllung einzelner Anforderungen in Verknüpfung miteinander zu überprüfen, wurden Szenarien entwickelt.

2.2.1 Mini Szenario

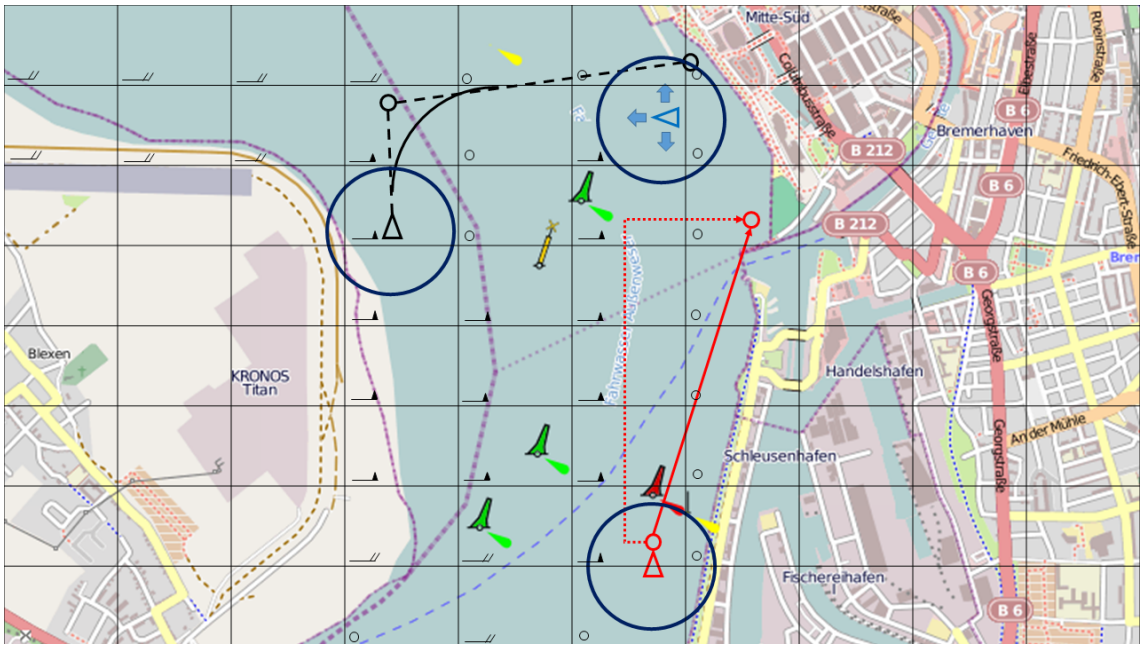
In der Design-Phase wurde basierend auf den Anforderungen ein Mini Szenario entwickelt, das am Ende der Phase mithilfe der Software ausführbar sein sollte. Das Mini Szenario gilt für den Protoypen der Software.



Zwei Schiffe folgen ihren Routen, welche durch einen Start- und einen Endpunkt definiert sind. Eine global einheitliche Windrichtung, die durch das Environment Federate generiert wurde, verändert sich stetig, hat jedoch keinen Einfluss auf die Schiffe. Das EPD Federate zeigt die Bitmap einer Seekarte, die sich bewegenden Schiffe und die Windrichtung und -kraft. Außerdem werden Tonnen visuell dargestellt und hervorgehoben, wenn sie sich im Feld des Schiffskurses befinden. Die Information darüber, welche Tonnen her-

vorgehoben werden müssen, stammt vom Sensor Service des World Federates. Um dies zu erreichen, muss die World Komponente die zum Kartenmaterial zugehörige GML-Datei sowie die über HLA veröffentlichten Schiffsinformationen auslesen, interpretieren und im World Model speichern. Die GML-Datei und Bitmaps der Seekarten werden durch die Chart Komponente und den Mapserver zur Verfügung gestellt. Die Datensätze sind S-57 konform.

2.2.2 Initiales Szenario



Das als Initial Scenario bezeichnete Szenario dient zur Überprüfung, ob die Anforderungen an jede Komponente erfüllt werden. Es müssen mindestens drei verschiedene Schiffstypen, zum Beispiel ein Motorboot, Segelschiff und ein Tanker, in das Szenario involviert sein. Die Schiffe werden unterschiedlich gesteuert. Mindestens eines der Schiffe sollte manuell mithilfe eines separaten Eingabegerätes gesteuert werden. Das Schiff hat lediglich ein Dynamikmodell und kein Behaviour, weil dieses durch den Nutzer repräsentiert wird, der das Schiff manuell steuert. Außerdem kann ein Schiff autonom gesteuert sein. An dieser Stelle wird zwischen zwei Möglichkeiten der autonomen Steuerung differenziert. Die erste Option ist das Folgen einer vordefinierten Route mit diversen Wegpunkten. Bei der zweiten Option sollte das autonom gesteuerte Schiff in der Lage sein, seine eigene Route mithilfe eines gesetzten Start- und Endpunktes (und möglicherweise diversen Häfen, die mithilfe

des A*-Algorithmus während der Fahrt zum Endpunkt angesteuert werden) zu finden. Dies ermöglicht dem Nutzer die Simulation einer Rundfahrt.

Während der Fahrt berücksichtigen die Schiffe die Umwelteinflüsse. Die nötigen Informationen werden durch die Environment Komponente bereitgestellt. Die angewendeten Umwelteinflüsse sind Wind mit einer Richtung und Geschwindigkeit, Strömung mit einer Richtung und Geschwindigkeit und Tide. Die Umwelteinflüsse verändern sich im Rahmen des Szenarios, um zu zeigen, dass das Dynamikmodell und die ausgewählten Behaviours die Einflüsse berücksichtigen. Resultierend aus der Dynamikberechnung fahren die Schiffe auf Trajektorien.

Die Routen der Schiffe kreuzen sich und es kommt zusätzlich zur frontalen Begegnung zweier Schiffe während des Szenarios. Außerdem soll es zu einer Kollision kommen. Diese Situationen sollen zum Einen die Funktionstüchtigkeit der Kollisionsdetektion in der World Komponente demonstrieren und zum Anderen testen, ob die Kollisionsvermeidung der einzelnen Schiffe funktioniert. Dieser Fall ist hilfreich zur Überprüfung, ob das in den COLREGs festgehaltene Verhalten eingehalten wird. Außerdem soll ein Schiff mit der Küste kollidieren, um die physische Integrität in der World Komponente zu überprüfen. Zusätzlich soll mithilfe der Sensordaten aus der World Komponente die Sichtbarkeit der Vessel manipuliert werden.

Autonom gesteuerte Schiffe mit einem Routenfindungsalgorithmus nutzen Informationen zur World aus der Charts Komponente, um eine Route zwischen ihrem Start- und Endpunkt zu finden. Darüber hinaus bekommt jede Komponente Informationen zur World von der Charts Komponente. Hierauf basierend setzt beispielsweise die Environment Komponente die Positionen einzelner Umwelteinflüsse oder die World Komponente überprüft auf Kollisionen mit Boyen oder der Küste.

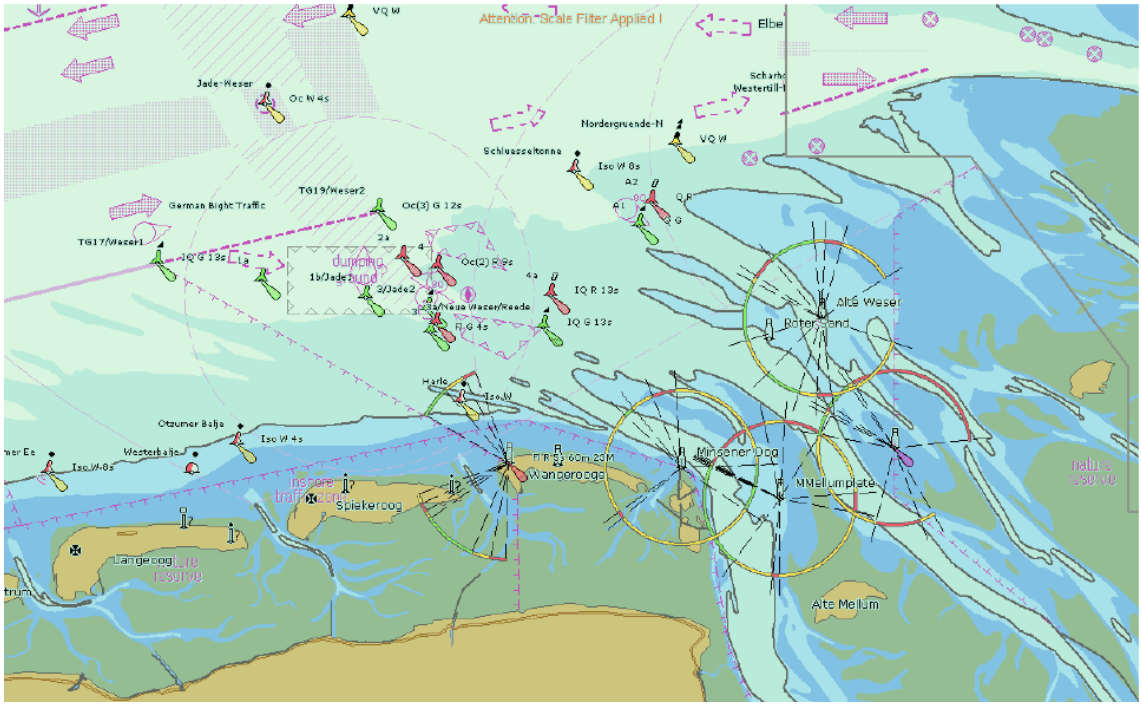
Innerhalb des Initial Scenarios visualisiert die EPD die Schiffe sowie dazugehörige Routen und Trajektorien, den Sichtbereich jedes Schiffs, die Objekte aus der Seekarte und die Windgeschwindigkeit und -richtung aus der Environment Komponente. Außerdem wird jegliche Verletzung der COLREGs oder Kollision durch die EPD visualisiert. Die meisten dieser Informationen werden als verschiedene Layer der S-57 Seekarte dargestellt.

Die Konfiguration des Szenarios erfolgt durch die Nutzung der Editoren für die Ships, Environment, Charts und World Komponente. Im Ships Editor ist es möglich, die Anzahl

der Ships-Federates sowie der Schiffe und ihre Typen zu konfigurieren. Jedem Schiff wird außerdem ein Dynamikmodell, ein Behaviour, eine Route und ein ControlType zugewiesen. Außerdem erfolgt die Eingabe von Schiffsspezifikationen wie der Masse, Länge oder der Anzahl der Ruder. Die Dynamik und das Behaviour bestehen aus diversen Subdynamiken/-behaviours, die auf verschiedene Art und Weise miteinander kombiniert werden müssen. Daraus resultieren sich unterschiedlich verhaltende Schiffe. So ist die Möglichkeit der Anpassung der Schiffe für verschiedene Anwendungsfälle gewährleistet.

Im Editor der Environment Komponente wählt der Nutzer aus, welche Umwelteinflüsse wann und wo erscheinen sollen. Welche Sensordaten während der Simulation geliefert werden sollen, die Einstellungen der physischen Integrität sowie die Kollisionsdetektion können im World Editor konfiguriert werden. Außerdem kann der Nutzer den Detaillierungsgrad der Seekarten mithilfe des Charts Editors festlegen. Diese Einstellungen müssen zu Beginn der Simulation vorgenommen werden. Hierbei werden verschiedene Konfigurationsdateien erzeugt und von der jeweiligen Komponente ausgelesen.

2.2.3 Abschlussszenario



Gegen Ende der Projektgruppe MTSS wurde ein abschließendes Szenario konzipiert, das

die Erfüllung aller Anforderungen überprüfen und demonstrieren sollte.

Die Simulation soll erstellt und gestartet werden, ohne dass eine Ziele Code angefasst werden muss. Durch die Editoren soll es jedoch möglich sein, die Simulation erst zur Vorführung zu erstellen.

Hierfür muss zusätzlich beim Chartserver die Möglichkeit bestehen, S-57 und/oder S-63 Karten einlesen zu können, um diese für die Simulation zu nutzen. Das Kartenmaterial soll im GML-Format für die Komponenten World, EPD und Ships bereitgestellt werden. Dies gilt als erfolgreich, wenn nach dem Start der Simulation die Schiffe Landflächen erkennen können und mithilfe der unten beschriebenen Behaviors darauf reagieren.

Beim Environment Editor soll es möglich sein, verschiedene Polygone mit unterschiedlichen Umwelteinflüssen zu erstellen und ein Grib-File einzulesen. Es soll mindestens ein Polygon mit dem Umwelteinfluss Wind und eines mit dem Umwelteinfluss Current (Strömung) erstellt werden. Hierbei soll eines der beiden Polygone über die Zeit konstant bleiben, ein anderes soll über die Zeit hinweg interpoliert werden.

Das Szenario beinhaltet 20 Schiffe, welche sich in drei unterschiedliche Dynamiken einteilen lassen. Hierbei soll mindestens ein Schiff das SimpleKinematic, eines das Kinematic und ein anderes das FixedPositionTurn Dynamikmodell verwenden. Es sollen Reaktionen auf die Umwelteinflüsse (Environment-Capability / Wind und Current) zu beobachten sein. Die Ruder und Motoren der Schiffe sollen in diversen Komplexitätsstufen einstellbar sein und mindestens zwei unterschiedliche Schiffstypen mit mehreren Rudern / Motoren fahren.

Mindestens eine Kollision soll provoziert werden, wodurch sich beobachten lässt, dass die Schiffe ihre Geschwindigkeit auf 0 setzen und sich der neue Status des Schiffes auf *grounded* ändert.

Für die Schiffe sind vier Behaviors vorgesehen, die allesamt zum Einsatz kommen sollen. Ein Schiff soll einer Route folgen, die sich ausschließlich auf Wasser befindet. Ein anderes Schiff soll einer Route folgen und dabei das Rechtsfahrgebot beachten. Ein weiteres Schiff soll die Route mithilfe des A*-Algorithmus, ein anderes mit dem Dijkstra-Algorithmus planen. Es soll bei einem der beiden Schiffe ein Landhindernis im Weg sein, sodass die Bahnplanung greift und das Schiff um die Landfläche navigiert wird. Eines der Schiffe soll

mithilfe des Schiffcontrollers oder des Keyboards steuerbar sein.

Die EPD soll während der Simulation die Karteninformationen des ChartServers als Visualisierung nutzen. Es soll möglich sein, während der Simulation die Schiffe, ihre Routen sowie ihre Bounding (als Ellipse) darzustellen. Die Bounding soll visuell dargestellt werden, sobald die CollisionDetection aktiviert ist. Die Position der Schiffe soll sich während der Simulation aktualisieren. Die Distanz der Schiffe soll dargestellt werden und Kollisionen sollen durch die Erkennung der World Komponente dargestellt werden. Auch die Environment-Polygone sowie deren Informationen (METOC-Forecastpoints) sollen angezeigt werden. Informationen über die GML sollen als PickUpReport im Dockable Panel dargestellt werden.

Weitere Services der World wie der RadarService und der ViewSensorService sollen auf der EPD ein- und ausgeblendet werden können.

Die Simulation soll mithilfe eines Konfigurators pausiert, sowie die Simulationsgeschwindigkeit verändert werden können. Anschließend soll die Simulation gespeichert und zum gleichen Zeitpunkt wieder geladen werden.

3 Systemarchitektur

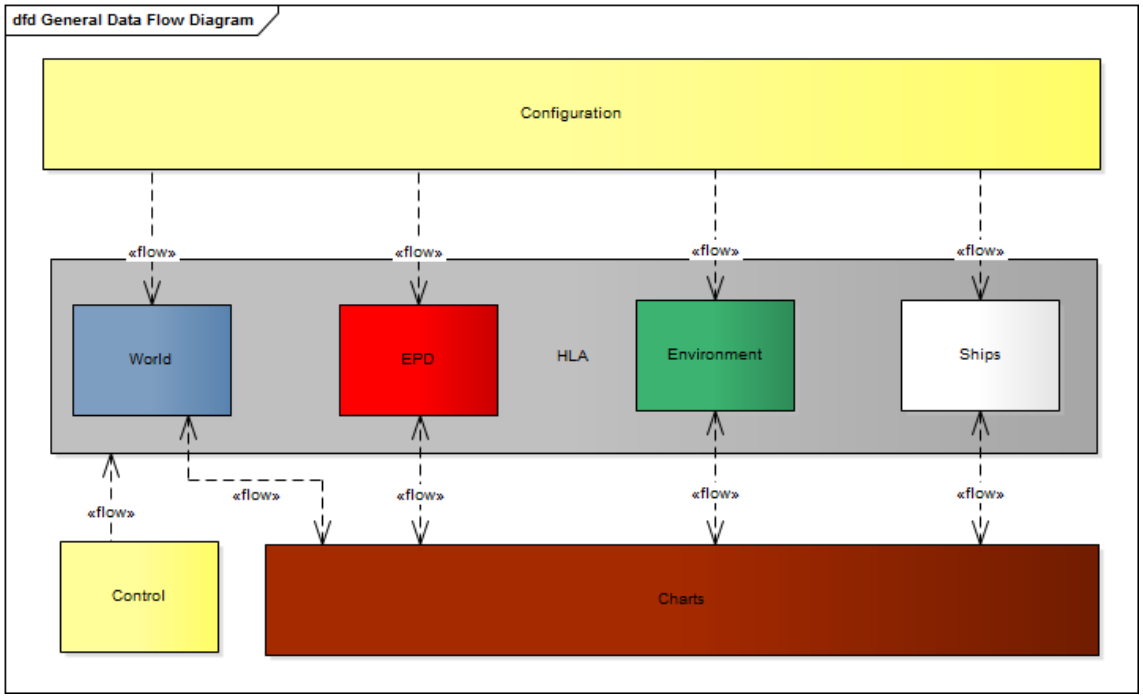


Abbildung 1: Grober Aufbau der MTSS

Die Systemarchitektur wurde basierend auf den Anforderungen abgeleitet. Die *Charts* Komponente, stellt die Karteninformationen bereit. Diese werden den anderen Komponenten entweder als GML oder als Bitmap zur Verfügung gestellt. Über das Hypertext Transfer Protocol (HTTP) ist es möglich mit der *Charts* Komponente zu kommunizieren. Da diese als MapServer entwickelt wurde unterstützt die Komponente sowohl WMS als auch WFS Anfragen. Die Komponenten *Ships, World* und *Environment* besitzen Editoren, durch welche es möglich ist, die für die Simulation notwendigen Konfigurationen vorzunehmen. Diese Konfigurationen werden als Dateien (Config-Files) abgespeichert und können für jede beliebige Simulation verwendet werden. Hierfür sollte für jeder Komponente ein Config-File existieren.

Die *EPD* ermöglicht die Visualisierung der Simulation und kommuniziert mit den Komponenten *World, Ships* und *Environment* über den High-Level-Architecture (HLA) Standard. im Kontext der HLA werden die Komponenten als Federates gestartet wodurch sich der Vorteil bietet eine Komponente in mehrere Federates aufzuteilen und diese auf unterschiedlichen Rechnern laufen lassen zu können, wodurch ein enormer Performancegewinn

während der Simulation erzielt werden kann.

Die Simulation lässt sich durch das *Control* Federate starten, stoppen und pausieren. Desweiteren ist es mit dem *Control* Federate möglich die Simulationszeit zu manipulieren, sodass die Simulation 100x schneller und 8x langsamer als Echtzeit laufen gelassen werden kann.

Die Datenmodelle der Komponenten wurden in den S-100 Standard der IHO überführt. Hierdurch ist es möglich die Komponenten im eMIR Kontext zu verwenden und auch in anderen Projekten die diesen Standard unterstützen. Für eine übersichtliche Darstellung der notwendigen Informationsflüsse für eine Simulation kann die Abbildung 1 genutzt werden.

Für das Starten der Simulation müssen .district_simplan Dateien angelegt werden, welche das korrekte Anmelden der Federates an die HLA übernehmen. Der sequenzielle Ablauf dieses Vorgangs ist in Abbildung 2 abgebildet.

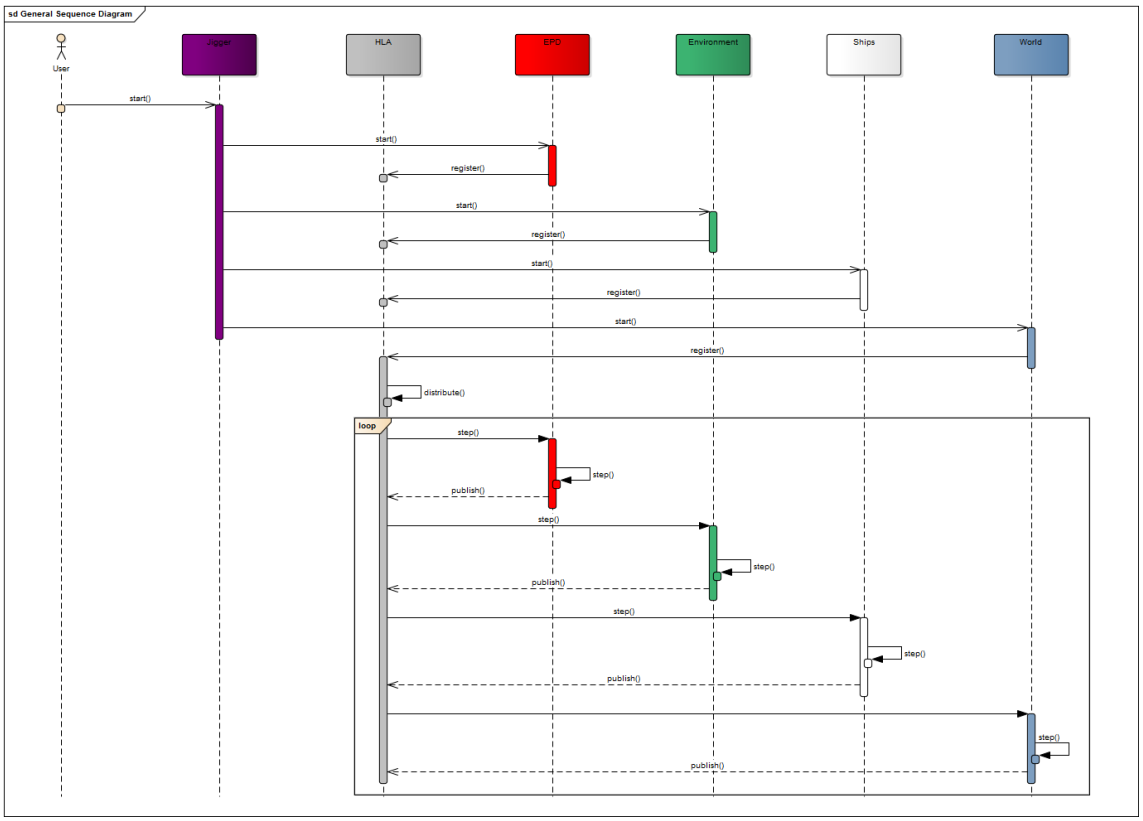


Abbildung 2: Sequenzdiagramm zum Verständnis von HLA

4 Komponenten

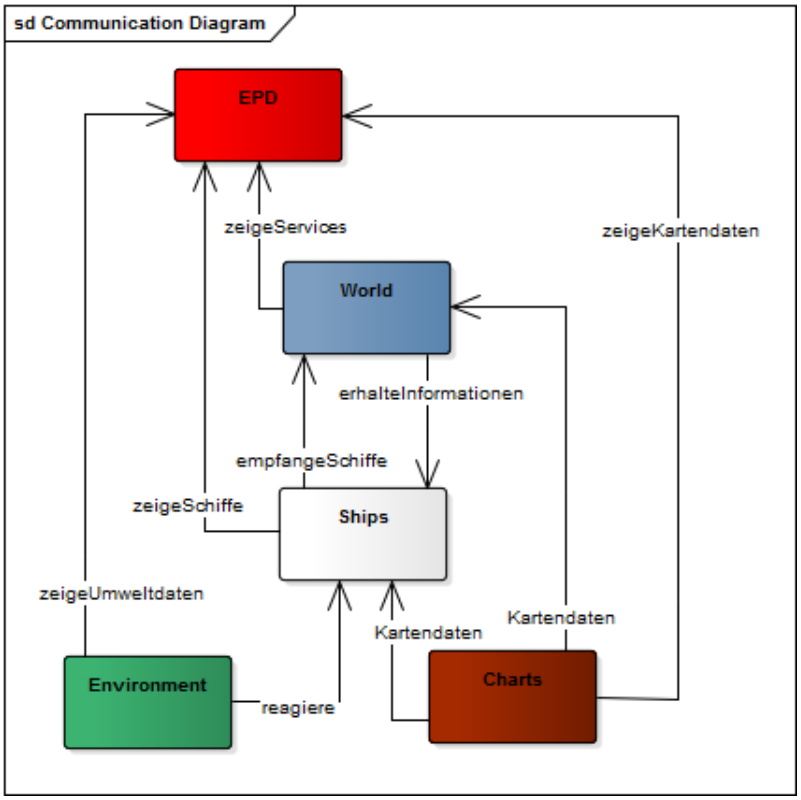


Abbildung 3: Kommunikationsdiagramm der MTS

Die entwickelte MTS besteht aus insgesamt fünf Komponenten. Diese lassen sich in die Bereiche Ships, Environment, World, Charts und EPD unterteilen. Bei der Entwicklung lag der Fokus auf einer unabhängigen Nutzung dieser Komponenten. Dennoch sollten die Möglichkeit bestehen, die Komponenten miteinander interagieren zu lassen um so realitätsgetreue Verhalten der Schiffe zu erhalten, um diese im Nachhinein für Forschungszwecke nutzbar zu machen. Für die Kommunikation der Komponenten wurde auf den High Level Architecture (HLA) Standard zurückgegriffen. Dieser im eigentlich für militärische Zwecke entwickelte Standard eignet sich vor allem für die Synchronisierung der Simulationskomponenten bei der Verteilung auf unterschiedliche Ressourcen.

Um eine hohe Performance und geringere Rechenzeiten zu gewährleisten, wurde die Möglichkeit geschaffen die Komponenten in unterschiedliche Federates zu unterteilen um diese auf unterschiedlichen Ressourcen laufen lassen zu können. Die Komponenten haben die

Möglichkeiten die Änderungen zu "publishen" damit die anderen Komponenten, welche den Status der Komponente "subscribed" haben, diese Daten auch empfangen können. Die Abhängigkeiten können in Abbildung 1 genauer eingesehen werden.

Des weiteren wurde bei der Entwicklung der Komponenten darauf geachtet, dass die Möglichkeit der Erweiterung der Komponenten gegeben ist. Die Gruppe hat sich deshalb bei der Implementierung daran gehalten einen Framework-Charakter zu waren um die Erweiterung möglichst einfach zu halten.

Die Funktionalitäten sowie die Ideen hinter der Implementierung werden im Folgenden näher erläutert. Es werden die Ansätze die die Komponenten verfolgt haben näher beschrieben. Eine Übersicht über die grundlegenden Interaktionen der fünf entwickelten Komponenten lässt sich Abbildung 3 entnehmen.

4.1 Charts

Die Komponente *Charts* ist für die Bereitstellung des Kartenmaterials zuständig. Um eine einheitliche Datenbasis zu gewährleisten erfolgt die Bereitstellung der Karten beziehungsweise Implementierung der Komponente als Web-Server für Karten, einen sogenannten MapServer.

Für die Anforderung von Kartenmaterial wurden zwei Möglichkeiten implementiert. Dies liegt der Tatsache zu Grunde, dass Seekarten, sogenannte Electronic Navigational Chart (ENC), im deutschen Gewässer nur in verschlüsselter Form zur Verfügung stehen. ENCs lassen sich in S-57 und S-63 Karten unterteilen. Die freien, für jeden zugänglichen Karten liegen im S-57 Format vorher wohingegen die deutschen Seekarten nur als S-63 Format zur Verfügung stehen. Um S-63 Karten nutzen zu können muss ein Decoder oder ein Lizenzschlüssel erworben werden. Dieser wird nur von einer handvoll Unternehmen in Deutschland angeboten.

Der Universität Oldenburg stehen Lizenzschlüssel der SevenCs GmbH zur Verfügung. Zu diesen Lizenzschlüsseln besitzt die Universität Oldenburg ebenfalls eine Version des von der Firma SevenCs entwickelten ECDIS Kernels mitsamt eines Dongles um den Kernel nutzen zu können. Da es sein kann, dass der Dongle nicht permanent zur Verfügung steht, wurde eine Alternative zum SevenCs Kernel entwickelt um für Testzwecke auch ohne S-63 auszukommen. Hierfür wurde der MapServer von OSGeo verwendet um eine Nutzung der MTS auch ohne SevenCs möglich zu machen.

4.1.1 MapServer von OSGeo

Bei dem MapServer handelt es sich um einen WebServer für Kartenmaterial welcher auf Map-Files basiert. Die Map-Files funktionieren ähnlich wie Scripte über welche der MapServer seine Informationen bezieht, welches Material bereitgestellt werden soll und in welcher Form. Die für die MTS relevanten Services sind der Web-Map-Service (WMS) sowie der Web-Feature-Service (WFS).

Da der MapServer wie ein WebServer funktioniert, erhält dieser seine Anfragen über einen HTTP-Request und beantwortet diesen mit einem Bildes bei WMS Anfragen und mit einem GML-File bei WFS Anfragen. Über eine Bounding-Box lässt sich der Kartenausschnitt angeben, aber auch diverse Coordinate-Reference-Systeme (CRS) stehen zur Verfügung. Die MapFiles bauen das Kartenmaterial in Schichten (Layer) auf, wodurch

auch nur bestimmte Layer angefragt werden können, wie eine Art Filterfunktion.

Da der MapServer ausschließlich für die Nutzung von ShapeFiles entwickelt wurde, wird für die Nutzung von S-57 Karten der Driver *OGR* verwendet. Da für die Zwecke die die Chart-Komponente erfüllen soll entsprechend viel Kartenmaterial genutzt werden soll, wurde eine MapFile-Konfigurator entwickelt durch welchen es möglich ist, die verschiedensten MapFiles zu erzeugen. Diese müssen anschließend nur noch auf den Server geladen werden.

Da der MapServer nur für einfache Shapefiles entwickelt wurde um einfache Seekarten anzeigen zu können, stößt dieser bei der Einbindung von einer größeren Anzahl von Karten aus Performance Sicht an seine Grenzen. Das Anfragen von Bild und GML kann daher sehr lange dauern. Daher ist der MapServer hauptsächlich für Testzwecke geeignet und sollte bei einer großen Anzahl von Kartenmaterial nicht verwendet werden. Einzelne Karten lassen sich jedoch sehr gut darstellen.

Es ist zu beachten das der MapServer keine S-52 Konforme Darstellung unterstützt, weshalb hier die Visualisierung selbst gewählt werden kann. Hierbei lassen sich einfach Bilder auf den Server laden um bspw. Boyen darzustellen oder die Farbe verschiedener Flächen (Areas) kann im Mapfile oder über den Konfigurator angepasst werden.

4.1.2 ECDIS Kernel von SevenCs

Der ECDIS Kernel liegt ausschließlich in einer C++ Version vor. Daher wurde dieser nicht wie der Rest der MTS in Java sondern in C++ implementiert. Der Kernel bietet jedoch schon einige Beispielprogramme welche für die Implementierung genutzt werden konnten. Der Kernel wurde ebenfalls als WebServer implementiert und kann Anfragen sowohl mit einem Bild als auch einer GML beantworten. Bei dem ECDIS Kernel handelt es sich zwar nicht um einen 'offiziellen' MapServer er kann jedoch auch auf WMS und WFS Anfragen reagieren da die Anfrage auf die bestimmten Schlüsselwörter geprüft wird. Mithilfe einer GUI auf dem Server lässt sich bequem S-57 aber auch S-63 Kartenmaterial einlesen und auch auf dem Server darstellen.

Da der Kernel für die Implementierung von ECDIS-Anlagen gedacht ist, bietet dieser von Haus aus die Visualisierung der Karten in einem S-52 Format an. Dieser MapServer kann auch Anfragen von großen Kartenausschnitten mit viel Material ohne Probleme beantworten.

Da der Kernel als solcher weder die Möglichkeit bietet als WebServer genutzt zu werden noch GMLs bereitzustellen wurde hierfür auf Third-Party-Tools zurückgegriffen. TUFAO

bildet die Grundlage für die Nutzung des Kernels als WebServer da es sich hier um ein einfachen WebServer für C++ Anwendungen handelt.

Für die Umwandlung des Kartenmaterial in GML wurde ebenfalls OGR als Bibliothek verwendet. Eine übersichtliche Darstellung des Daten- und Informationsflusses der Nutzung von SevenCs als auch OSGeo lässt sich Abbildung 4 entnehmen.

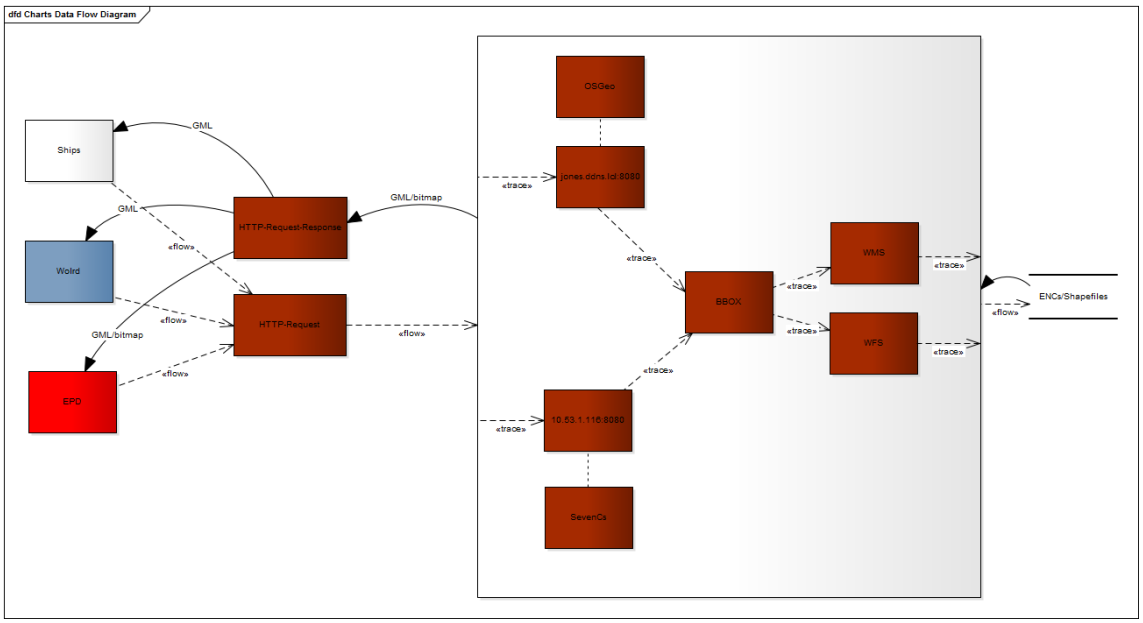


Abbildung 4: DataFlow der Charts-Komponente

4.2 Environment

Die *Environment*-Komponente ist zuständig für die Bereitstellung der Umwelteinflüsse (*EnvironmentalImpacts*). Die Einflüsse *Wind*, *Current* und *Tide* werden in der Simulation berücksichtigt. Diese nehmen Einfluss auf das Verhalten der Schiffe und werden der *EPD* zur Visualisierung bereitgestellt.

4.2.1 Modell

In der *Environment*-Komponente wird ein Umweltszenario festgelegt, das sowohl zeitliche als auch geografische Informationen berücksichtigt. Es ist möglich, ein statisches oder dynamisches Szenario zu wählen. Bei einem statischen Szenario ändern sich die *EnvironmentalImpacts* über den zeitlichen Verlauf nicht. In dem dynamischen Szenario ändern sich die Umweltbedingungen über den Zeitverlauf. Für die geografische Bestimmung werden Polygon festgelegt. Um ein Umweltszenario zu erstellen, das auch realen Daten beruht, gibt es die Möglichkeit, eine Datei mit Wetter-Daten zu importieren. Hierbei handelt es sich um *GRIB*-Dateien, die u. a. von dem Bundesamt für Seeschifffahrt und Hydrographie bereitgestellt werden. *GRIB* steht für *gridded binary* und ist von der Weltorganisation für Meteorologie (WMO) eingeführt worden. Die Konfiguration des Umweltszenarios findet in dem *Environment*-Editor statt, der eine grafische Oberfläche zur Erstellung einer Konfigurationsdatei bietet.˜

Das Modell basiert auf dem S100-Datenmodell, das ebenfalls von dem *eMir-Projekt* verwendet wird. In den folgenden Abschnitten wird das Datenmodell anhand des Klassendiagramms erläutert.

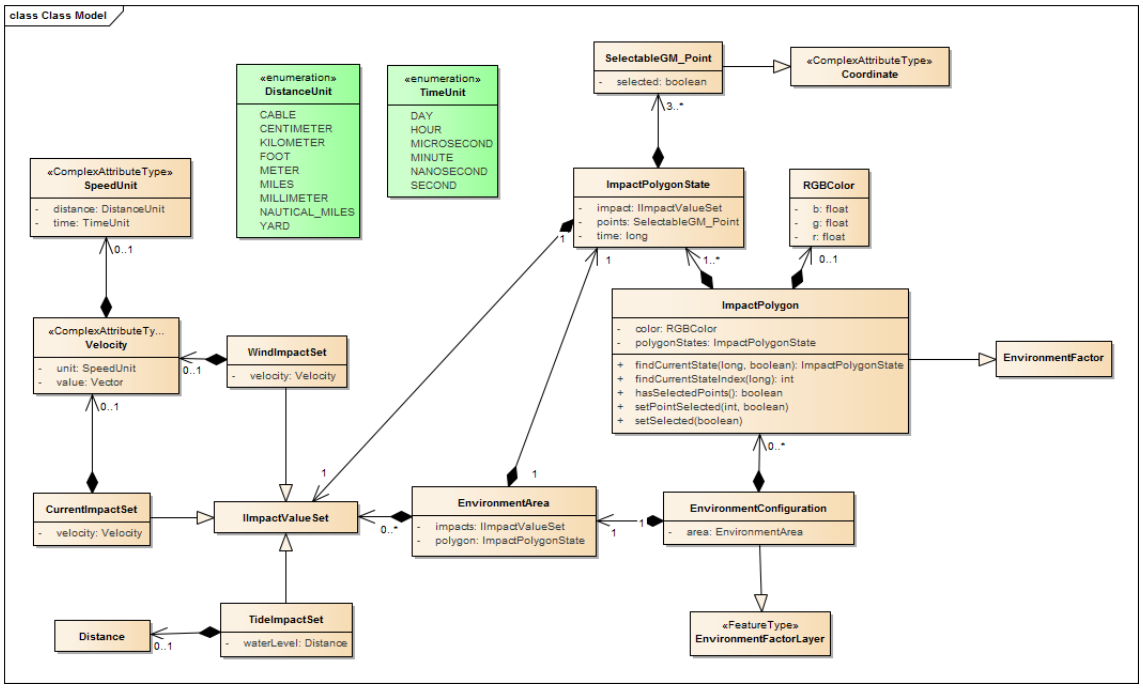


Abbildung 5: Environment Klassendiagramm

Das Datenmodell enthält Klassen aus dem S100-Datenmodell, wurde jedoch auch um weitere Klassen, die in dem MTSS-Projekt verwendet werden, erweitert.
Ein *Environment*-Objekt besteht aus einer *EnvironmentConfiguration*, die von der FeatureType-Klasse *EnvironmentFactorLayer* des S100-Datenmodells erbt. Der *EnvironmentConfiguration* wird eine *EnvironmentArea* zugewiesen, die für den gesamten Bereich einen Umwelteinfluss festlegt. Eine *EnvironmentConfiguration* kann mehrere *ImpactPolygone* enthalten, die die visuell sichtbaren Polygone darstellen. Ein *ImpactPolygon* erbt von der S100-Klasse *EnvironmentFactor*. Jedem Polygon wird eine Farbe und mindestens ein *ImpactPolygonState* zugewiesen. Ein *ImpactPolygonState* suggeriert einen bestimmten zeitlichen und geographischen Zustand des Polygons. Das Polygon muss aus mindestens drei GM-Points bestehen. Zu jedem Status wird ein *EnvironmentalImpact* in Form eines *ImpactValue* festgelegt. Unterstützt werden in der MTSS die drei Umwelteinflüsse *Wind*, *Current* und *Tide*. Sowohl einem *WindImpactSet* als auch einem *CurrentImpactSet* wird jeweils eine Geschwindigkeit in Form einer *Velocity* zugewiesen. Die ComplexAttributeType-Klasse *Velocity* basiert auf dem S100-Datenmodell und setzt sich zusammen aus einer Einheit (Unit) und einem Vektor. Der Umwelteinfluss *Tide* besteht aus einem *WaterLevel*, das in einer *Distance* angegeben wird.

4.2.2 Datenfluss

Die *Environment*-Komponente kommuniziert über die HLA mit den Komponenten *EPD*, *Ships* und *World*. Sie veröffentlicht jedoch nur Objekte und empfängt keine Daten. Der HLA-Standard lässt allerdings die Möglichkeit offen, auch Objekte empfangen zu können. Diese Komponente ist allerdings nur dafür zuständig, Umweltdaten bereitzustellen. Sie benötigt bspw. keine Informationen über die Schiffsposition, sodass eine einseitige Kommunikation mit der *Ships*-Komponente ausreichend ist. Die *EPD* hat die Aufgabe, die Umweltdaten auf der grafischen Oberfläche zu visualisieren. Daher gilt auch in diesem Zusammenhang, dass ausschließlich die *EPD* Informationen von der *Environment*-Komponente benötigt.

4.2.3 Federate

In dem *Environment*-Federate wird aus der zu übergebenden Konfigurationsdatei das *Environment*-Szenario erstellt. Zur Übertragung an die *Ships*-Komponente, *World*-Komponente und *EPD* werden aus der Konfigurationsdatei die einzelnen Polygone mit ihren Zustandseinträgen ausgelesen. Jeder Zustandseintrag enthält geografische und zeitliche Informationen. Geografische Informationen werden als GM-Points mit jeweils einem Längengrad-Wert (*longitude*) und Breitengrad-Wert (*latitude*) veröffentlicht.

4.3 EPD

Die *EPD* stellt im Kontext der Projektgruppe MTSS die Visualisierungskomponente dar.

Die Open-Source-Anwendung ist von der **Danish Maritime Organization** entwickelt worden. Sie beinhaltet eine landseitige (EPD-Shore) sowie schiffsseitige (EPD-Ship) Anwendung und basiert auf JavaBeans™ sowie OpenMap™. In dieser Projektgruppe wird die landseitige Anwendung als ausgelagerte Visualisierungskomponente umgesetzt. Das liegt daran, dass in der EPD-Shore Informationen zu mehreren Schiffen angezeigt werden und nicht nur aus der Sicht eines Schiffes.

Die GUI der Anwendung besitzt ein zentrales Fenster zur Darstellung von Seekarten samt geographischer Daten. Wichtige dynamische und statische Daten wie Name, Geschwindigkeit, Tiefgang oder Richtung eines Schiffes werden bei einem Mausklick auf das jeweilige Schiff innerhalb des rechten Panels angezeigt. Weiterhin existieren Funktionen, wie etwa die Radarsuche. Auf diese Weise lassen sich auf Basis einer bestimmten Position benachbarte Schiffe anzeigen. Die Seekarten können beliebig vergrößert oder verkleinert werden. Die Details der Seekarten sind abhängig von der eingestellten Detaillierungsstufe.

Die darzustellenden Seekarten bekommt die EPD von der Komponente *Charts*. Diese stellt die Seekarten als Bitmap zur Verfügung. Des Weiteren kommuniziert *EPD* über *HLA* mit den Komponenten *Environment Ships* und *World*.

Für die Kommunikation von S-100 Daten über *HLA*, bekommt die *EPD* einen S-100 Datenempfänger", dieser kann S-100 konforme Daten über HLA empfangen. Zusätzlich wird in der Planung auch eine Schnittstelle zum Empfangen von Daten über RabbitMQ berücksichtigt. Diese Schnittstelle wird von LABSKAUS verwendet und wird auch dort umgesetzt.

Im folgenden wird nun auf den Datenfluss der *EPD* eingegangen. Dabei werden verschiedene Diagramme vorgestellt. Für das weitere Verständnis ist es notwendig zu wissen, dass die *EPD* ein (fast) ausschließliches passives Federate ist. Einzige Ausnahme ist das Senden eines Requests zu der *Charts*-Komponente. Die nachfolgende Tabelle 1 zeigt, welche Informationen die *EPD* von den anderen Komponenten empfängt.

Objekte	Schiffsinformationen	Umwelteinflüsse
Schiffahrtszeichen	Positionen aller Schiffe (Longitude und Latitude)	Tidedaten
Wassertiefe	Geschwindigkeit über Grund (speed over ground, sog)	Strömungsdaten
Häfen	Kurs über Grund (course over ground, cog	Winddaten
Wind- und Ölparks	Kurs (heading)	
Schiffe vor Anker	Routen	
	Name	

Tabelle 1: Kommunizierte Daten an die EPD

4.3.1 Datenfluss

Über die *HLA* bekommt die *EPD* Objekte von allen anderen Federates kommuniziert.Diese Objekte sind in der eben vorgestellten Tabelle 1 zu sehen. Diese Kommunikation der Daten ist in Abbildung 6 dargestellt. Dabei werden von der Komponente *Ships* Infomationen über Schiffe, wie z.B. Name, MMSI und Geschwindigkeit, übermittelt. Von der *Environment* bekommt die *EPD* Informationen über die Umweltbedingungen innerhalb der Simulation, die auf der GUI dargestellt werden. Die Komponente *World* sendet der *EPD* Informationen von allen SensorServices zu. Diese können z.B. ein Radar- oder AIS-Service sein.
Über einen MapServer bekommt die *EPD* die zu visualisierende Seekarte. Diese wird von der Komponente *Charts* als WMS-Link bereitgestellt, der innerhalt der *EPD* im Setup-Menu nur noch eingetragen werde muss. Diese Seekarte enthält alle wichtigen Informationen einer Seekarte und bildet dabei die komplette Welt ab. Zudem lässt sich diese Seekarte Stufenlos, per Maus, hinein- und herauszoomen.

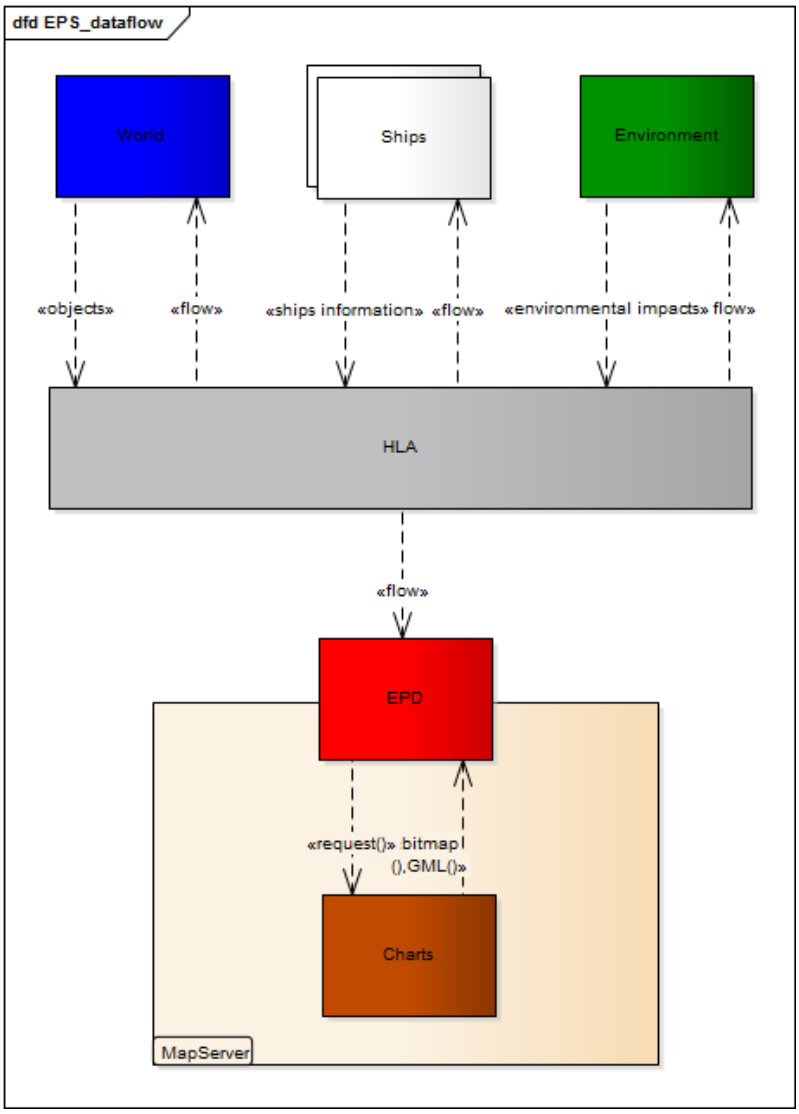


Abbildung 6: Datenflussdiagramm der EPD

4.3.2 S-100 Schnittstelle

LABSKAUS [1] sendet die RabbitMQ [2] XML serialisiert Instanzen der S-100-Datenstrukturen. Die LABSKAUS-Kommunikationskomponente muss diese XML-Daten akzeptieren und wandelt sie in die S-100-Datenstrukturen. Die Darstellung von Informationen geschieht im Anschluss mit dem S-100-Visualizer. Jedoch muss diese Schnittstelle nicht von der

[1] Laboratory for safte critical experiements at sea
[2] RabbitMQ ist ein auf Basis des OTP-Frameworks implementierter Message Broker.

Projektgruppe durchgeführt werden, sondern soll nur bei der Planung berücksichtigt werden.

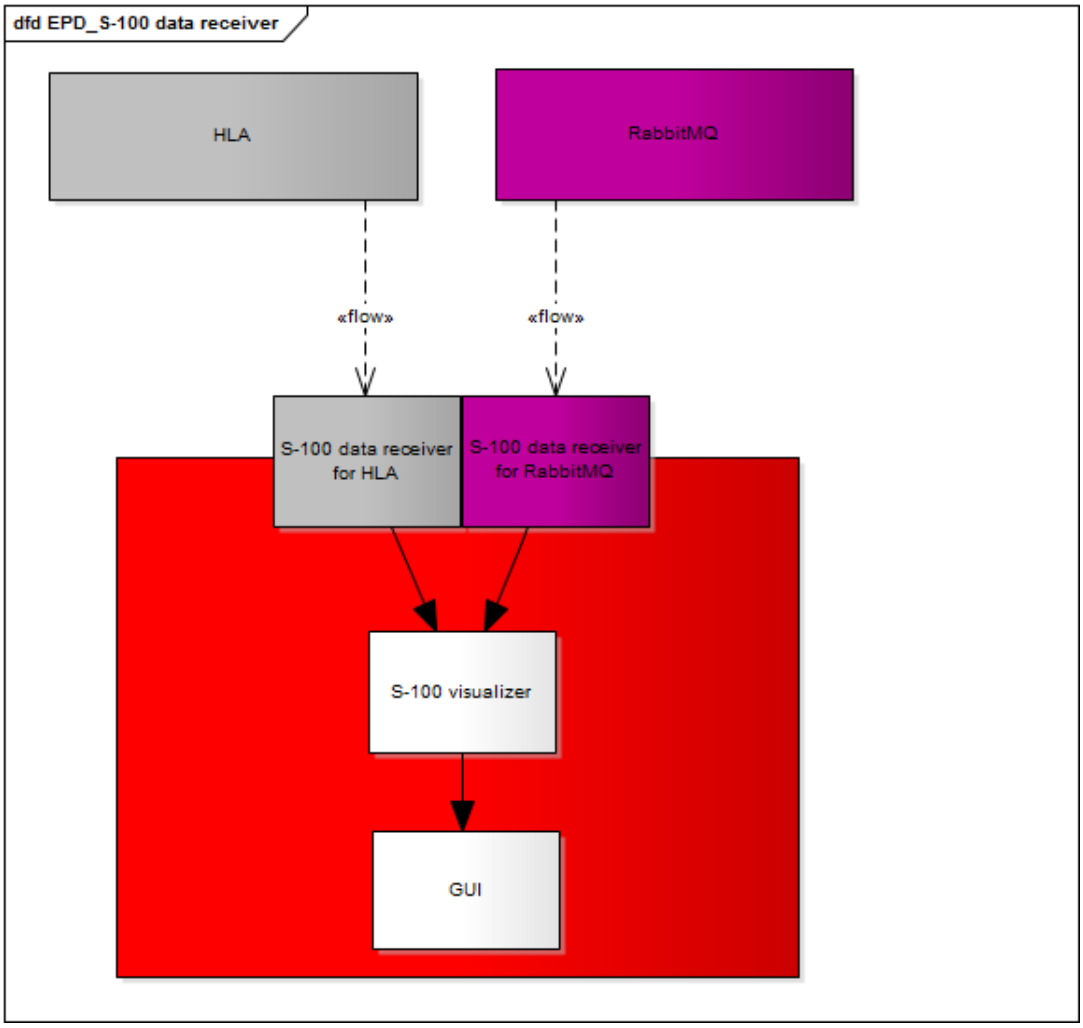


Abbildung 7: S-100 Schnittstelle der EPD

4.3.3 Sequenzdiagramm

Sequenzdiagramme beschreiben den Austausch von Nachrichten zwischen Instanzen mit Lebenslinien. Bei der Verwendung eines MapServer sendet die *Chart*-Komponente eine Bitmap an der *EPD*. Dann kann die *EPD* die gesendete Seekarte visualisieren. Wenn die *EPD* Informationen, wie zum Beispiel Objekte, Schiffsdaten und Umweltauswirkungen über *HLA* kommuniziert bekommt, visualisiert dies die *EPD* über die GUI. Diese

Informationen werden in einer Schleife verwendet, da die kommunizierten Daten ständig aktualisiert werden müssen, um Veränderungen dieser (wie zum Beispiel Änderung der Windrichtung) dargestellt werden müssen. Es ist wichtig zu betonen, dass die *EPD* wie oben bereits beschrieben ein fast ausschließliches passives Federat ist.

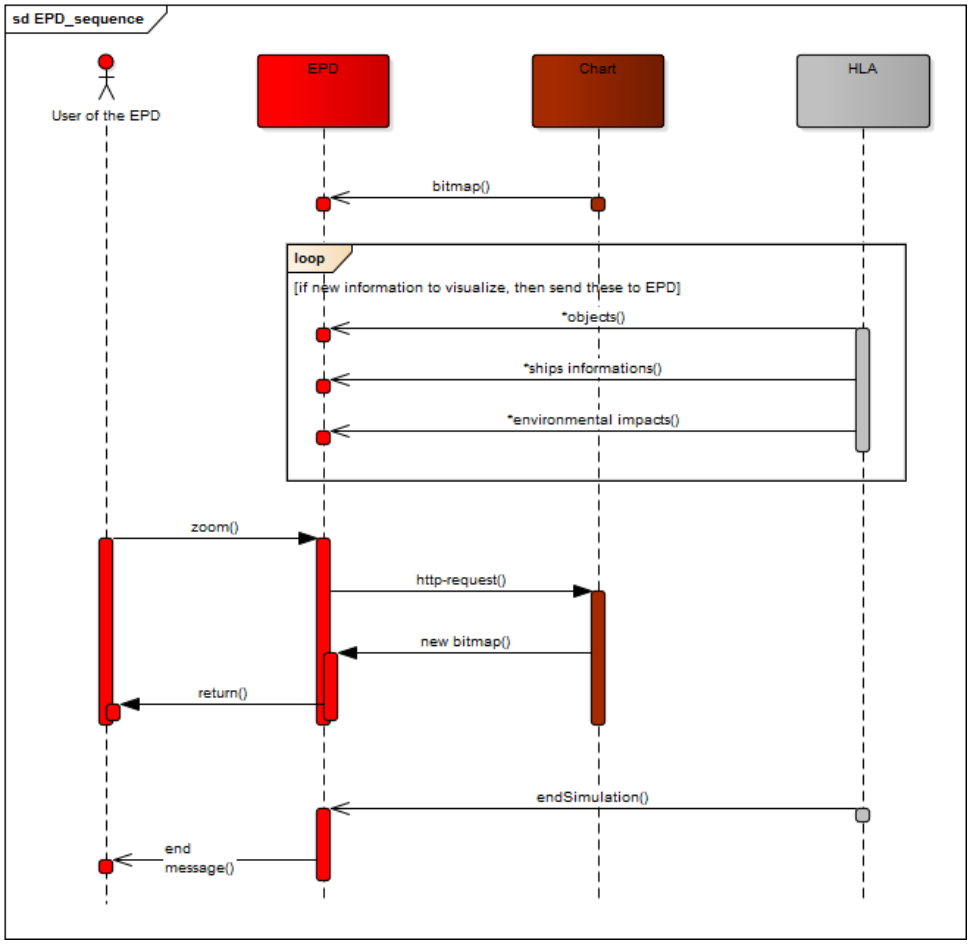


Abbildung 8: Sequenzdiagramm der EPD

Einzige Ausnahme ist der HTTP-request für eine neue BitMap. Wenn ein Benutzer hinein oder heraus zoomt, sendet die *EPD* einen request an den MapServer an die Chart-Komponente. Die Charts sendet den neuen Ausschnitt der Seekarte. Wenn die Simulation beendet wird, kommuniziert die *HLA* der *EPD* dies mit und die Visualisierung wird beendet.

4.3.4 Use-case Diagramm

Das Use-case Diagramm enthält den Benutzer und das System. Das System sollte durch den Benutzer verwendet werden. Dieses System wird hier durch die *EPD* dargestellt. Kurze Beschreibungen werden in den Ellipsen aufgelistet. Jede Ellipse enthält eine Funktion. Die Ellipsen sind also die Use-Cases.

Beispiel für Funktionen, die durchgeführt werden können sind: hinein- und herauszoomen, Objekte auf der Karte anklicken (Pick-up report), aktivieren und deaktivieren von Layern und Routen.

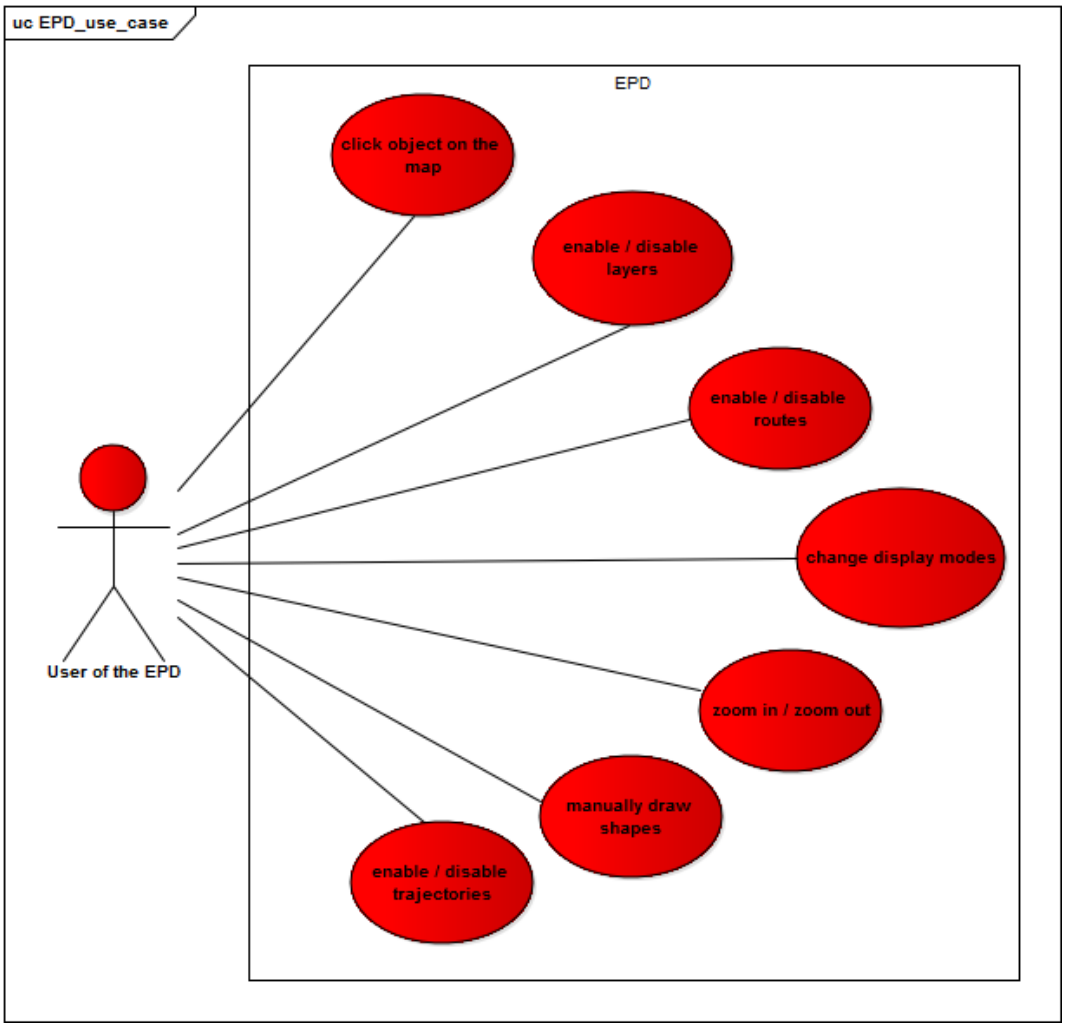


Abbildung 9: Use-case Diagramm der EPD

4.3.5 Aktivitätsdiagramm

Ein Aktivitätsdiagramm eignet sich für die Modellierung aller Aktivitäten innerhalb eines Systems. Beispielsweise wird in Abbildung 10 die Aktivität den Anzeigemodus ändern angezeigt. Hierzu klickt der Benutzer auf "Preferences" und dann auf "Maps". Dort wählt er "Colourprofile". Hier hat der Benutzer die Wahl zwischen drei Anzeigemodi: 1. "Night", 2. "Dusk" und 3. "Day". Diese drei Modi sind standardmäßig in der *EPD* integriert. Anschließend muss der Benutzer ein Informationsprofil auswählen. Hier sind seine folgenden Optionen: 1. "BASIC", 2. "NORMAL" und 3. "VOLL".

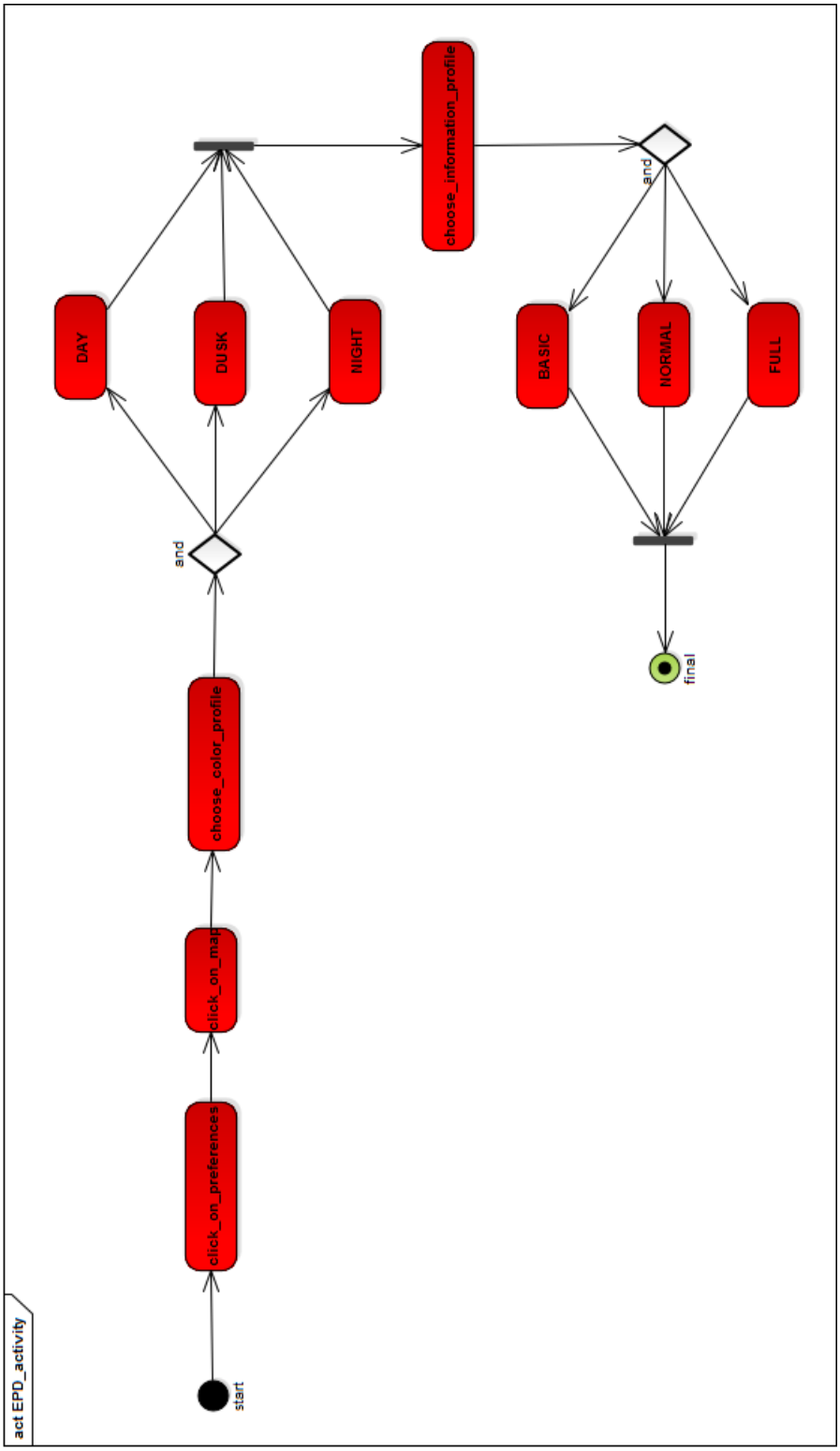


Abbildung 10: Aktivitätsdiagrammder EPD

4.4 Ships

In dieser Komponente wird jegliches Verhalten eines oder mehrerer Schiffe berechnet. Dabei werden die Berechnungen in den Schiffen in zwei Kategorien eingeteilt: die Dynamik- und Verhaltensberechnungen (Behavior). Durch ein S-100 konformes Modell soll die Aufteilung in *Behavior* und *Dynamic* beschrieben werden.
Neben der Datenstruktur, die im nachfolgenden Abschnitt behandelt wird, ist der funktionelle Aufbau der Schiffe interessant. Die untenstehende Abbildung 11 bildet den Aufbau eines Schiffes in der *MTSS* ab. Wie bereits erwähnt besteht ein Schiff aus einem Dynamikmodell und einem Verhaltensmodell. Bei dem Dynamikmodell kann es sich um die aufgeführten Implementierungen handeln. Wichtig zu erwähnen ist hierbei die Tatsache, dass ein Schiff exakt ein Dynamikmodell besitzt. Im Gegesatz dazu kann ein Schiff mehrere Verhaltensmodell besitzen, die miteinander kombiniert werden können. Dabei muss jedes Schiff einen *ControlType* besitzen. Alle Implementierungen, die dieser Ebene zuzuordnen sind, ermöglichen das Steuern und navigieren der Schiffe. Des Weiteren kann ein Schiff die Möglichkeit besitzen sich Routen selbstständig zu berechnen. Ein Schiff muss nicht zwangsweise ein solches Verhalten besitzen. Dieses Subverhalten eignet sich beispielsweise nicht für Schiffe, die manuell gesteuert werden, da diese Schiffe keine Informationen über eine Route benötigen und demnach auch keine besitzen. Falls ein Schiff für die Logik eines bestimmten Subverhaltens Informationen aus der Seekarte benötigt, so stellt das Suvbverhalten *ChartInformation* diese Informationen bereit. Das gesamte Konzept dahinter lässt sich des Weiteren noch erweitern, sodass beispielsweise noch Subverhalten zur Kollisionsvermeidung einfach hinzugefügt werden können. Im weiteren Verlauf dieses Kapitels sollen die implementierten Subverhalten und das Dynamikmodell erläutert werden.

4.4.1 Modell

Das Datenmodell, das jedem Schiff zugrunde liegt, basiert auf dem S-100 Datenmodell, das im *eMIR*-Projekt verwendet wird. Dieses Datenmodell wurde entsprechend der Klassen und Attribute erweitert, die für die Simulation der MTSS-Schiffe notwendig sind. Eine genaue Betrachtung dieser Klassen findet in den folgenden Abschnitten statt. Jeder *Ships*-Federate beinhaltet mindestens ein Schiff, wobei jedes Schiff aus genau einer Dynamikmodell (*Dynamic*) und einem Verhaltensmodell (*Behavior*) besteht. Dem *Behavior* sind jegliche Kontrollmöglichkeiten für die Ruder und Schrauben zugeordnet. Die teilen sich jeweils in *RudderControl* und *EngineControl* auf. Um das Schiff navigieren zu können habt das Verhalten eine Assoziation zum *ControlType*, dass sich wiederum in *ControlTypeWi-*

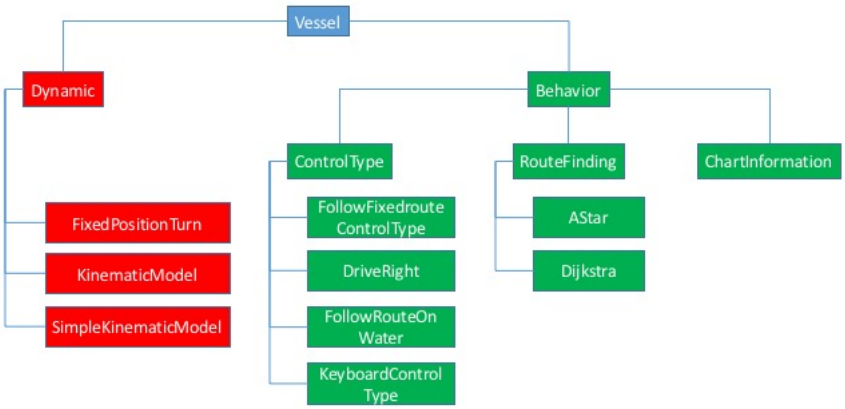


Abbildung 11: Hierachie der unterschiedlichen Implementierungen

thRoute aufteilen kann. Hier sind jegliche Verhalten einzuordnen, die eine festvorgegebene Route zur Navigation benötigen. Verhalten, die das nicht benötigen, können direkt dem *ControlType* zugeordnet werden.
Zusätzlich sind dem Verhalten jegliche Logiken zuzuordnen, die sich mit der Routenfindung beschäftigen und Informationen aus elektronischen Seekarten bereitstellen und verarbeiten.

Der *Dynamic* sind alle Logiken zuzuordnen, die für die Berechnung der Schiffsbewegung notwendig sind. Hierzu zählt neben dem eigentlich Dynamikmodell die Umwelteinflüsse (*EnvironmentalImpacts*), Maschinen (*Engine*) und Ruder (*Rudder*).

Dem Schiff an sich (*MTSSVessel*-Objekt) sind zusätzlich zu dem Dynamik- und Verhaltensmodell noch Ruder- und Maschinenemodelle zuzuordnen.

An diesem Modell ist besonders hervorzuheben, dass es modular aufgebaut ist, wodurch unterschiedliche Komplexitätsstufen ermöglicht werden. Zusätzlich enthält das Modell auch die Speicherung der schiffseigenen Attribute wie Gewicht, Länge, Drehgeschwindigkeit, Motor und Ruder. Dies alles erlaubt es schließlich unterschiedliche Schiffstypen nachzubilden, die sich auf Basis dieser Modularität auch entsprechend ihrer physikalischen Einschränkungen verhalten. Dadurch ist des Weiteren eine eindeutige Aufteilung des Schiffes in Antriebsorgane, Stellsignalgeber und Beschreibung der Schiffscharakteristik möglich. Eine klare Trennung zwischen Dynamik und Verhalten, sowie die eindeutige Zuordnung

von Klassen zum jeweiligen Typ sind die Folge.

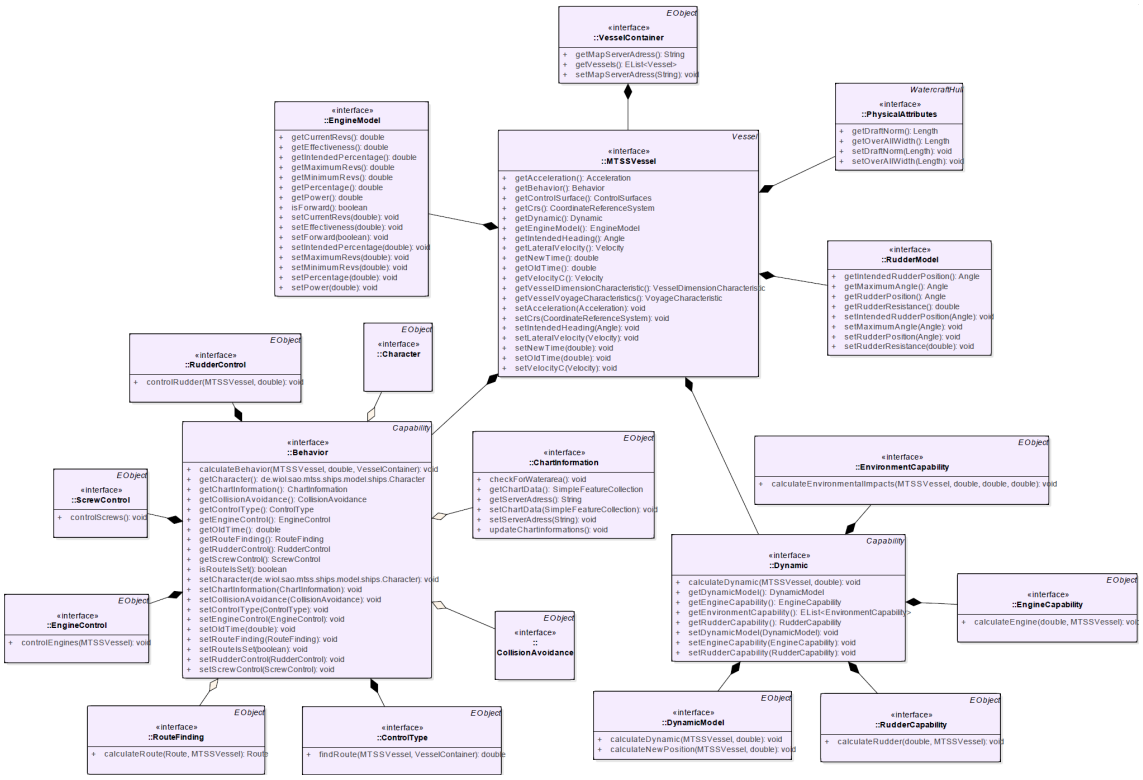


Abbildung 12: ShipsFederate-Model nach S-100 Umstellung

4.4.2 Federate

Im *Ships*-Federate wird aus der zu übergebenden Konfigurationsdatei das Modell für jedes Schiff erstellt, sofern es nicht bereits vorhanden ist. Anschließend wird wie bei jedem anderen Federate auch die *step-Methode* für die übergebene Zeitspanne ausgeführt. Hier werden für jedes Schiff zuerst die Verhaltens- und anschließend die Dynamikberechnungen ausgeführt. Zuvor wird geprüft, ob das Schiff mit einem anderen Schiff oder Objekt kollidiert oder es auf Grund gelaufen ist. Ist dies der Fall, werden die Berechnungen von Verhalten und Dynamik nicht ausgeführt und der Status der (*NavigationInformation*) des jeweiligen Schiffs auf *grounded* gesetzt. Zusätzlich werden die Umwelteinflüsse, die von der *Environment*-Komponente veröffentlicht werden, auf Überschneidungen mit der Schiffsposition geprüft und bei positivem Ergebnis an die dementsprechenden Klassen - sofern vorhanden - zur Umwelteinflussberechnung des Schiffs weitergegeben.

4.4.3 Behavior

Das Schiffsverhalten (*Behavior*) bildet in der MTSS sämtliche menschlichen und technischen Fähigkeiten eines Schiffes nach. Die technischen Fähigkeiten zielen auf die physikalische Beeinflussung des Schiffes ab, wie zum Beispiel die Position des Ruders zu verändern. Menschliche Fähigkeiten bilden die Fähigkeiten ab, die die Schiffsbesatzung während der Steuerung eines Schiffes benötigt. Somit verleiht das Verhalten den Schiffen eine gewisse Intelligenz, die sich je nach verwendeten und implementierten Verhalten in ihrer Komplexität unterscheidet. Im Folgenden werden die in der MTSS zur Verfügung stehenden Verhalten also in technische und menschliche Fähigkeiten unterteilt und als solche beschrieben. Als Übersicht über die Klassenhierarchie des *Behaviors* dient Abbildung 13.

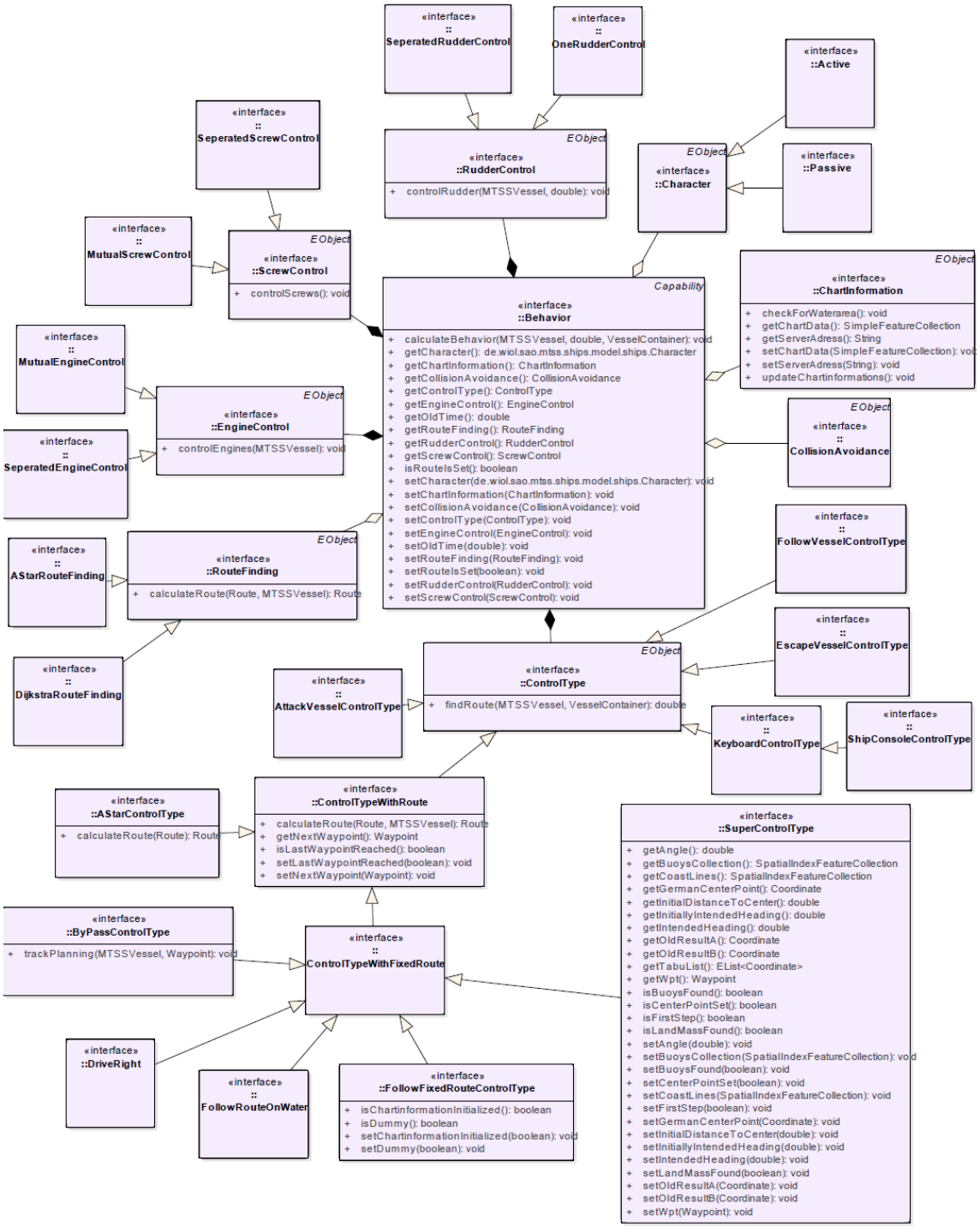


Abbildung 13: Behavior

Menschliche Fähigkeiten Wie bereits einführend oben beschrieben, bietet die Verwendung eines Verhaltens die Möglichkeit intelligentes, menschliches Handeln im Rahmen der Schiffe nachzubilden. Anwendungsgebiete wären hier unter anderem die Navigation oder Kollisionsvermeidung. Es ist des Weiteren möglich mehrere solcher Verhalten miteinander zu kombinieren, um so möglichst intelligente und realitätsnahe Schiffe zu simulieren.

DriveRight Im Allgemeinen sind Schiffe laut den *Traffic Serperation Schemes* dazu verpflichtet auf ihrer Trajektorie so weit rechts wie möglich zu fahren. Dieses Verfahren ist eine einfache Möglichkeit Kollisionen zu vermeiden. In der Navigation werden hierfür auf hoch frequentierten Seestraßen oder Zufahrten zu Häfen rote und grüne schwimmende Seezeichen in Form von Bojen eingesetzt. In diesem Zusammenhang gilt dann die Regel, dass Schiffe, die von der See kommen, sich so halten, dass sich die roten Bojen an der Steuerbordseite des Schiffes befinden. Entsprechend müssen sich grünen Bojen auf der Steuerbordseite befinden, wenn die Schiffe in Richtung See fahren.

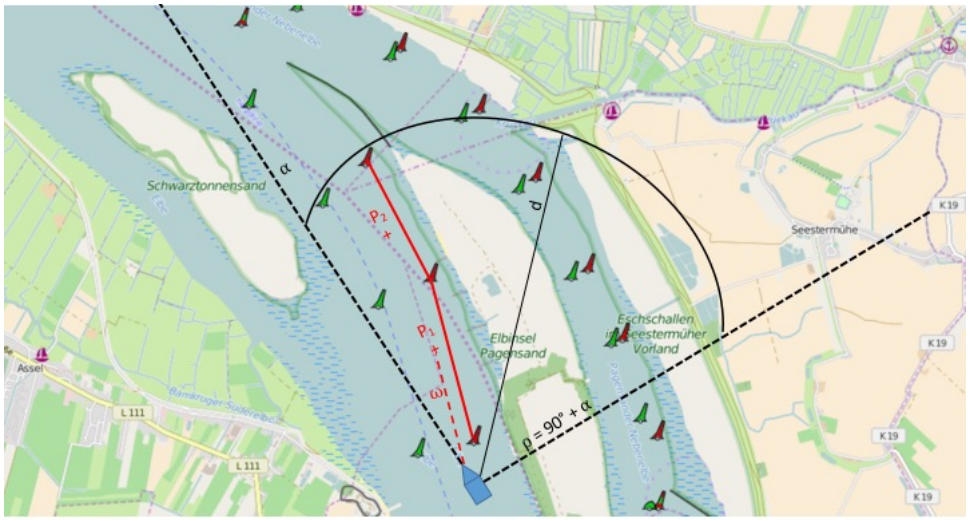


Abbildung 14: Schematische Darstellung der Logik für das Rechtsfahrgebot mit Bojen

Abbildung 14 zeigt schematisch die Funktionsweise für das Rechtsfahrgebot, wenn Bojen vorhanden sind. α bezeichnet hierbei den aktuellen Kurswinkel des blauen Eigenschiffs. Um nun die nächstgelegene Boje zu finden, an die sich das Schiff zu halten hat, wird in einem Halbkreis zwischen dem aktuellen Kurswinkel α und dem Winkel ρ gesucht, der sich aus der Summe von dem aktuellen Kurswinkel α und 90 zusammensetzt. Die Suchdistanz d wird durch den Radius des Halbkreises beschrieben.

Die Funktionsweise dieses Verhaltens lässt sich in zwei Schritte unterteilen.

1. **Suche nach der nächstgelegenen Boje und der Nachbarboje:** In dem oben beschriebenen Halbkreis werden die entsprechenden Bojen gesucht.
2. **Bestimmung einer neuen Bahn:** Hierfür wird zwischen den beiden gefundenen Bahnen eine Senkrechte gezogen. Dies geschieht mit Hilfe einer Methode, die dann einen Punkt entweder links oder rechts von der Senkrechten errechnet. In dem Beispiel auf Abbildung 14 ist der gefundene Punkt P_1 und P_2 das jeweilige Resultat. Die entstandenen Punkte beschreiben dann die Wegpunkte der Bahn, die das Schiff abfahren muss um sich an das Rechtsfahrgebot entlang der entsprechenden Bojen zu halten. ω ist der neue Kurswinkel zum Bahnpunkt P_1.

Für den Fall, dass keine Bojen innerhalb des Suchradius gefunden werden, gilt die Prämisse, dass keine Boje vorhanden ist. Da das Rechtsfahrgebot dennoch gilt, nutzen die Schiffe in diesem Fall das Land als Orientierungshilfe. Abbildung 15 zeigt schematisch die Funktionsweise dieses Ansatzes.

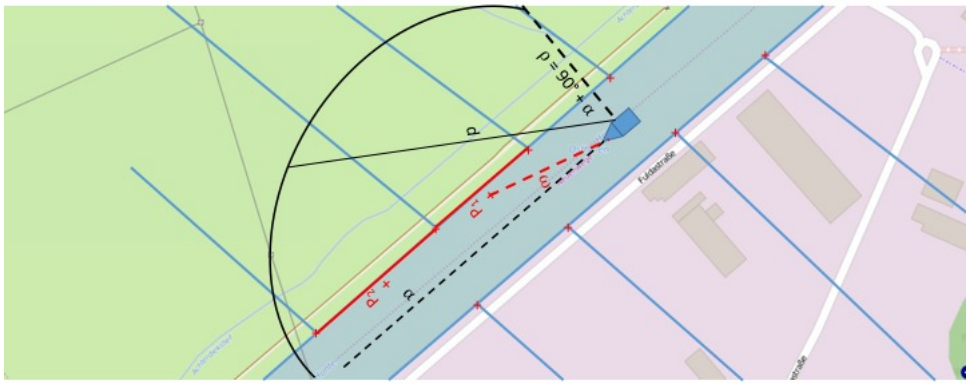


Abbildung 15: Schematische Darstellung der Logik für das Rechtsfahrgebot ohne Bojen

Für dieses Verfahren gelten die gleichen Bedingungen für den Halbkreis wie oben beschrieben. Der Unterschied liegt darin, dass die Landmasse als Orientierung zum Rechtsfahren genutzt wird. Die Landmasse wird in der GML als Polygon beschrieben, dessen Punkte Koordinatenpaare sind. Hierbei wird dann in Analogie zu dem Verfahren mit Bojen der Polygonpunkt gesucht, der dem Schiff am nächsten ist. Somit sind die oben beschriebenen

zwei Schritte auch auf dieses Verfahren anzuwenden.

FollowFixedRouteControlType *FollowFixedRouteControlType* ist das simpelste Verhalten, welches ein Schiff besitzen kann und ist auf Abbildung 16 zu sehen. Bei diesem Verhalten fährt das Schiff eine vorher definierte Route, die aus den Wegpunkten *WP1* und *WP2* besteht, auf direktem Wege ab. Hierfür kalkuliert das Verhalten bei jedem Aufruf den Kurswinkel von der zu dem Zeitpunkt des Aufrufes aktuelle Position des Schiffes zu der Position des nächsten Wegpunktes. Im Rahmen dieser Berechnung werden jegliche Informationen über die Landmasse oder die Positionen anderer Schiffe ignoriert. Folglich kann es unter der Verwendung von diesem Verhalten zu Kollisionen mit anderen Schiffen oder einer anderen Landmasse kommen.

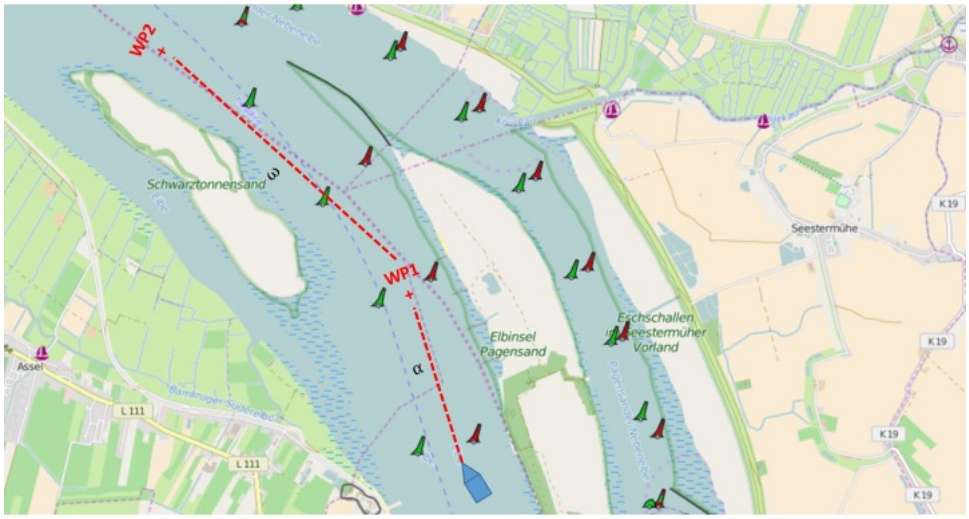


Abbildung 16: Grundlegende Problemstellung

FollowRouteOnWater Im Gegensatz zu *FollowFixedRouteControlType* ermöglicht das Subverhalten *FollowRouteOnWater* Routen zu folgen und dabei nur entlang der Wasserflächen zu navigieren.

Das Problem, welchem diesem Verhalten zu Grunde liegt, wird in Abbildung 17 illustriert. Das Schiff soll von seiner aktuellen Position zu einem Wegpunkt am Ende der Weser fahren und dabei nur den Wasserflächen folgen. Die bisher beschriebenen Verhalten ermöglichen diese Art der Navihation nicht. Bei der Verwendung des *FollowFixedRouteControlType* würde das Schiff beispielsweise auf direktem Weg zu dem Wegpunkt navigieren und keine

Abbildung 17: Grundlegende Problemstellung

Rücksicht auf etwaige Landflächen auf diesem Weg nehmen.
Um dies zu garantieren behinhaltet das Verhalten *FollowRouteOnWater* eine sogannte Vorschau (*lookahead*). Die untenstehenden Abbildung 18 veranschaulicht die Funktionsweise der Vorschau. Es wird überprüft, ob ausgehend von der aktuellen Position und in Richtung des Kurses in einer bestimmten Entfernung, die der dreifachen Länge des Schiffes entspricht, Landfläche vorzufinden ist. Die Berücksichtigung der Schiffslänge soll gerade bei der Simulation besonders großer Schiffe ein Ändern des Kurses ermöglichen.

Für den Fall, dass eine Kollision mit Land erwartet werden kann, sucht der Algorithmus von der Position, an der die Kollision erwartet wurde, in zehn Grad Schritten rechts und links davon nach einem Ausweichpunkt. Die Grundregel für die Auswahl der Punkte ist, dass derjenige Punkt zuerst gewählt wird, welcher auch zuerst gefunden wird. Wird jeweils rechts und links ein passender Ausweichpunkt gefunden, so soll der gewählt werden, welcher sich am nächsten an dem ursprünglichen Wegpunkt befindet. Sollten beide Punkte dann auch noch die gleiche Distanz zu dem Wegpunkt haben, so wird ein Punkt zufällig ausgewählt.

KeyboardControlType und ShipConsoleControlType Beide dieser *ControlTypes* sind dafür gedacht, dass ein Nutzer ein einzelnes Schiff manuell steuern kann, indem er die Rolle eines *Behaviors* übernimmt. Dabei gibt es die Möglichkeit die Position des Ruders und die Geschwindigkeit zu erhöhen bzw. zu verringern, vorwärts oder rückwärts zu fahren oder

Abbildung 18: Grundlegende Problemstellung

auf Maximal- bzw. Minimalgeschwindigkeit zu gehen. Die *KeyboardControl* funktioniert über eine GUI auf der Knöpfe für die oben genanten Möglichkeiten, sowie eine Anzeige für die aktuelle Ausrichtung, Geschwindigkeit und Ruderposition des Schiffs zur Verfügung gestellt werden. Zu sehen ist diese GUI in Abbildung 19. Die Steuerung über die Tastatur ist über die unter dem Tab *Config* angegebenen Tasten möglich.

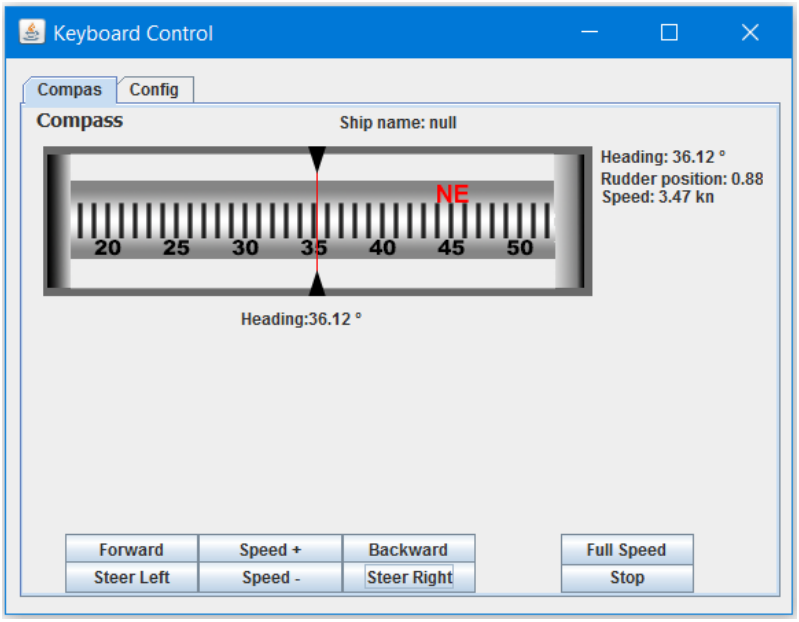


Abbildung 19: KeyboardControl Benutzeroberfläche

Bei dem *ShipsConsoleControlType* besteht zusätzlich die Möglichkeit ein Schiff über die Nachbildung eines Schiffsteuerpults zu steuern, das an einen Computer, auf dem die Simulation ausgeführt wird, angeschlossen ist. Eine Abbildung dieses Steuerpults kann Abbildung 20 entnommen werden.

Abbildung 20: ShipsConsole Steuerpult (vgl. `http://www.wilcopub.com/ship-console.html`)

Von den vielen Funktionen dieses Steuerpults werden im *ShipsConsoleControlType* die Schubkontrolle und die X-Rotation (zur Ruderlegung) genutzt.

RouteFinding Es gibt unterschiedliche Möglichkeiten, die ein Schiff haben kann um die Route zu berechnen. Dabei wurden zwei bekannte Suchalgorithmen verwendet die den Schiffen eine Route berechnen. In der Konfigurationsdatei gibt der Benutzer die Start und Zielposition ein. Anhand dieser Koordinaten wird die Route berechnet. Der Benutzer hat die Möglichkeit in der Konfigurationsdatei anzugeben, wo die sich das Schiff am Anfang befinden soll. Anschließend hat er die Möglichkeit die Routenpunkte festzulegen. Der erste Routenpunkt ist der Startpunkt und der letzte ist der Endpunkt der Route. Alle dazwischen liegenden Punkte sind Zwischenpunkte, die nacheinander abgefahren werden.
Bevor das Schiff anfängt zu fahren, wird die gesamte Route, die aus dem Startpunkt, den Zwischenpunkten und dem Endpunkt besteht, berechnet. Die Route ist dann abhängig

davon, ob die Schiffsposition mit der Routenstartposition übereinstimmt oder nicht.

Falls die Schiffsposition mit dem Startpunkt der Route übereinstimmt, wird eine Route von der Schiffsposition über die Zwischenpunkte zum Routenendpunkt berechnet.
Falls die Schiffsposition in der Konfigurationsdatei nicht mit dem Startpunkt der Route übereinstimmt fährt das Schiff von seiner Position zu der Routenstartposition, die in der Konfigurationsdatei festgelegt wurde. Dabei wird eine Route von der Schiffsposition zur Routenstartposition mittels des festgelegten Suchalgorithmuses berechnet.

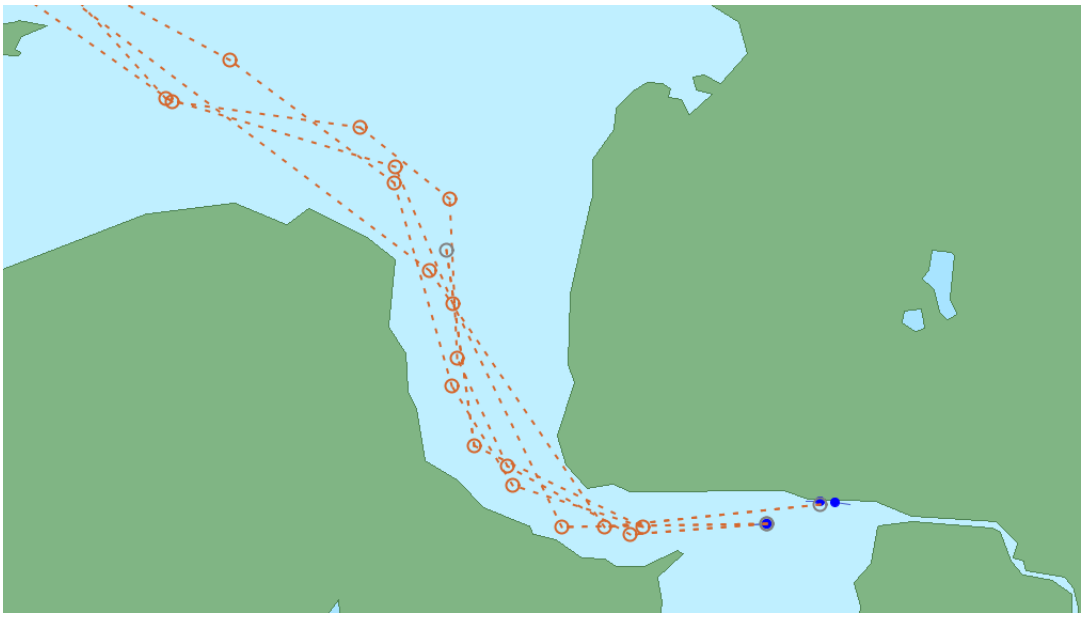


Abbildung 21: Beispiel einer Route mit dem Astern

Dijkstra-Algorithmus Der Dijkstra-Algorithmus ist ein Klassiker unter den bekannten Suchalgorithmen. Mit diesem Algorithmus ist es möglich anhand eines Graphen die kürzeste Route zwischen zwei Positionen zu berechnen. Bei der MTSS wird der Dijkstra-Algorithmus ebenfalls verwendet. Der Benutzer kann im Editor den Dijkstra-Algorithmus als Routenfindung auswählen. Sollte dieser Algorithmus ausgewählt worden sein, wird eine Route anhand dieses Algorithmuses erstellt, die das Schiff abfahren soll.
Die für die Umsetzung der kürzesten Route wird die Klasse DijkstraShortestPathFinder [3] der Javabiliothek (org.geotools.graph.path) verwendet, da dieser in ihrer Implementierung und Effizienz hoch ist. Um eine Route von der Schiffsposition über die Startposition der Route, den Zwischenpunkten und zum dem Endpunkt zu erhalten, wird dieser Algorith-

[3] http://docs.geotools.org/latest/javadocs/org/geotools/graph/path/DijkstraShortestPathFinder.html

mus innerhalb der Zwischenpunkte verwendet und abschließend als eine Route gespeichert, die das Schiff dann abfahren kann.

A*Algorithmus Der Unterschied zwischen dem Astern und dem Dijkstra-Algorithmus liegt daran, dass dieser nicht beliebig alle Nachbarpunkte abarbeitet sondern die Nachbarpunkte nach der Distanz zum Zielpunkt in eine Warteschlange speichert, daher spricht man von einem informierten Suchverfahren. Dadurch ist dieser Algorithmus effizienter und hat eine besser Laufzeit als der Dijkstra-Algorithmus. Bei der Verwendung diesen Algorithmuses wird ebenfalls die Java-Bibliothek verwendet. So ist sichergestellt, dass die Implementierung dieses Algorithmuses korrekt und effizient ist.
Ähnlich wie beim Dijkstra-wird dafür eine Route erstellt, die das Schiff abfahren kann. Abhängig von der Schiffs -und Startposition wird die Route berechnet, die das Schiff abfahren soll.

Verwendung und Aufbau der Route Die Routenfindung ist ein festgelegtes Verhalten (Behavior) eines Schiffes. Die Sie kann für unterschiedliche Zwecke zum Einsatz kommen. Der Benutzer legt die Startpunkt, die Zwischenpunkte und die Endpunkt fest. Anschließend kann das Schiff über eine vom Benutzer festgelegten Dynamik (Simple Kinematic Model, Kinematic Model, Fixed Cornering usw.) die Route abfahren. Diese Art von Routenfindung kann für unterschiedliche Szenarien verwendet werden. Aufgrund der Effizienz, eignet es sich, dass die Routenfindungen einzusetzen, wenn die Strecke Route eine hohe Distanz aufweist. Da der Algorithmus auch Wasserstraßen und Kanäle abfahren kann, ist es möglich die kürzeste Route zum Ziel zu finden. Besonders einfach zu verwenden ist es, da der Benutzer im "Konfigurator"nur die Stationen als Routenpunkte einfügen brauch, die das Schiff abfahren soll.

Setzen der Waypoints Die Waypoints werden nicht nach bestimmten Abständen gesetzt sondern richten sich an Kurven. Der Vorteil besteht darin, dass die Route aus nur den "nötigsten"Waypoints besteht. So lange das Schiff das Ruder nicht ändern muss, wird kein neues Waypoint gesetzt. So würde eine z.B. vier Kilometer lange Strecke aus nur einem Start- und einem Endpunkt bestehen. Dabei wird vorausgesetzt, dass sich auf der Strecke keine Hindernisse wie Land befinden.

Technische Fähigkeiten Die technisch orientierten Verhalten ermöglichen es, Steuersignale an das Schiff zu geben. Die entsprechenden Werte werden von der bereits oben

beschriebenen Verhalten errechnet und im weiteren Verlauf der Aufrufiteration von der Dynamik (siehe unten) weiterverarbeitet.

RudderControl Die Fähigkeiten *RudderControl* und *SeperatedRudderControl* ermöglichen die Beeinflussung der Ruderpositionen der jeweiligen Schiffe. Im Gegensatz zum *RudderControl*, wo die jeweiligen Ruder nicht einzeln angesteuert werden können, ermöglicht *SeperatedRudderControl* die separate Ansteuerung jedes einzelnen Ruders der jeweiligen Schiffe. Dies ermöglicht in der Simulation Schiffe mit unterschiedlichen Manövrierfähigkeiten nachzubilden.

EngineControl In Analogie zu dem oben beschriebenen Verhalten *RudderControl* ist es möglich, die Motoren eines jeden Schiffes entweder separat oder gemeinsam zu steuern. Die hierfür zur Verfügung stehenden Verhalten werden als *EngineControl* und *SeperatedEngineControl* beschrieben.

4.4.4 Dynamik

Die Dynamik eines Schiffs ist für die Berechnung der Schiffsbewegung aufgrund der vom Verhalten eingestellten Parameter zuständig. Die Methode *calculateDynamic()* bildet hierbei den Einstiegspunkt. Sie koordiniert alle weiteren Klassen der Dynamikberechnung, wie *DynamicModel, Engine, Rudder* und *EnvironmentalImpacts*.

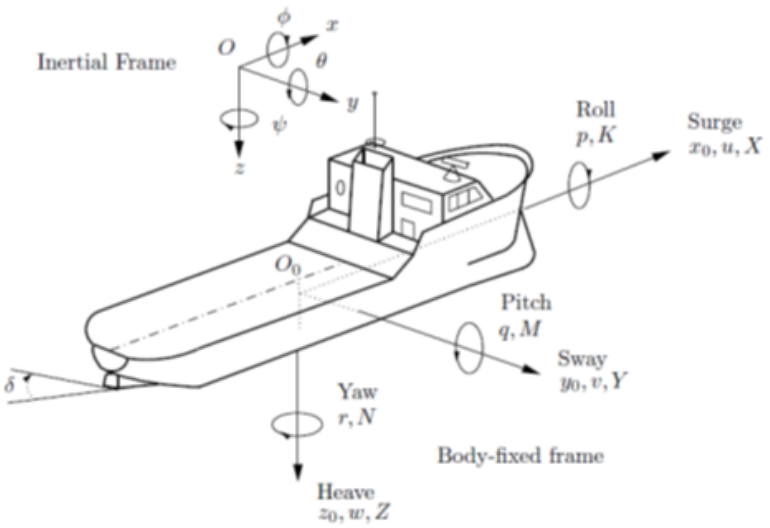


Abbildung 22: Bewegung in sechs Freiheitsgraden (DOF) [PB02]

Für die Simulation von Verkehrsflüssen genügt es, sich auf die drei Freiheitsgrade auf der flachen Ebene, also *Surge, Sway* und *Yaw*, zu beschränken, die Abbildung 22 entnommen werden können. Das Kippen um die X- bzw. Y-Achse des Schiffs, sowie die Auf- und Abwärtsbewegung können aufgrund ihrer geringen Auswirkung auf die generelle Schiffsbewegungsrichtung vernachlässigt werden. Somit sind realitätsnahe Verkehrsflusssimulationen möglich, die weniger Rechenleistung und Zeit benötigen als die Simulation von allen sechs Freiheitsgraden. Abbildung 23 gibt einen Überblick über die Klassen, die zur Dynamik gehören und im Anschluss näher beschrieben werden.

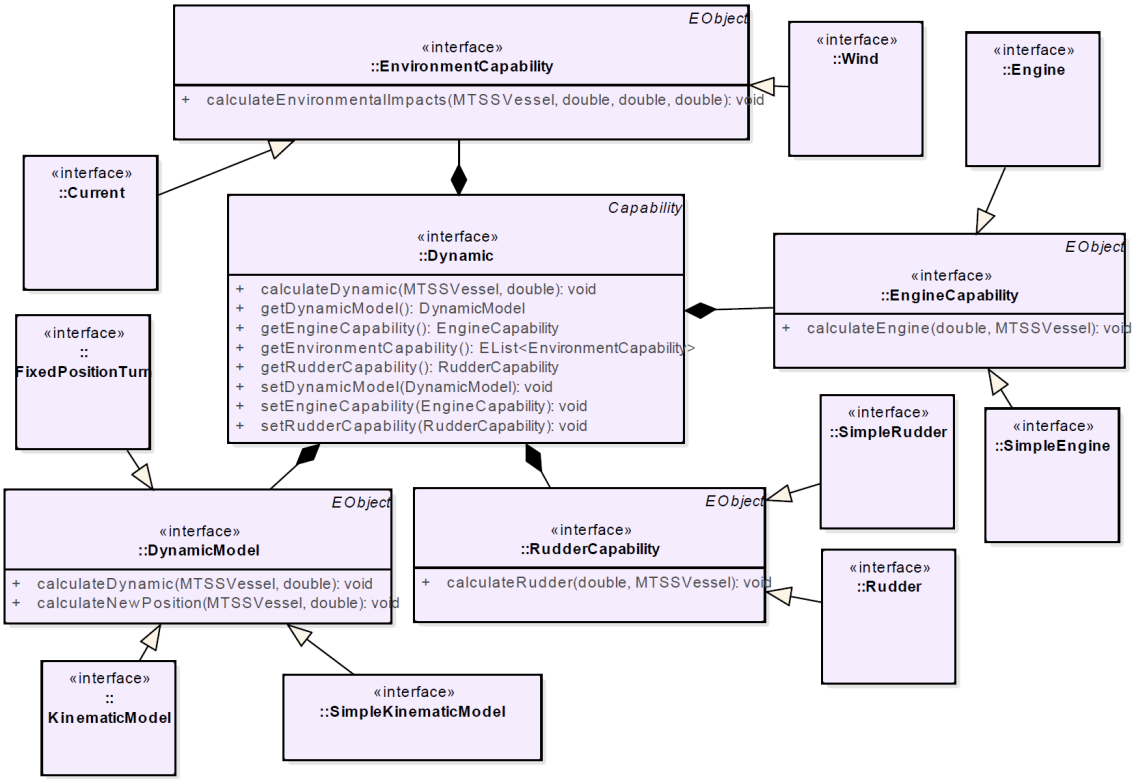


Abbildung 23: Dynamik

DynamicModel Das *DynamicModel* ist innerhalb der Dynamik für die Berechnung der Ausrichtung des Schiffs zuständig, dessen Geschwindigkeit, der zurückgelegten Strecke und der neuen Position des Schiffs im globalen Koordinatensystem. Um diese Berechnungen in unterschiedlichen Komplexitätsgraden zu ermöglichen und damit entweder mehr oder weniger realistische Schiffsbewegungen zu realisieren, gibt des drei verschiedene Dynamikmodelle zur Auswahl. So kann für Simulationen mit vielen Schiffen ein weniger komplexes

Dynamik-Modell ausgewählt werden, das die Rechenzeit verkürzt, jedoch weniger exakte Ergebnisse liefert. Für einen groben Überblick über Bewegungen von vielen Schiffen und deren Verhalten einander gegenüber eignet sich ein weniger komplexes Dynamikmodell jedoch gut.

Das grundlegende und damit am wenigsten komplexe Dynamikmodell ist *FixedPosition-Turn*. Hier wird die Geschwindigkeit ohne die Berücksichtigung einer Beschleunigung mittels Multiplikation von Maximalgeschwindigkeit und der aktuellen prozentualen Maschinenleistung berechnet. Die Ausrichtung des Schiffs (das Heading) wird so berechnet, dass sich das Schiff auch ohne Vortrieb auf der Stelle drehen kann. Dazu wird lediglich der aktuell anliegende Ruderwinkel zu dem aktuell vorliegenden Heading addiert. Die Ergebnisse dieser Berechnungen werden im *Vessel*-Objekt bzw. dessen *Capabilities* abgespeichert. Schlussendlich wird die Methode zur Berechnung der neuen Schiffsposition aufgerufen. Diese führt die Berechnung der vom Schiff im *HLA-Step* zurückgelegten Wegstrecke auf Basis der Geschwindigkeit durch. Anschließend wird mit Hilfe des *GeodeticCalculator* von GeoTools die neue Schiffsposition errechnet. Dazu benötigt die Funktion *getDestination-GeographicPoint()* die Position des Schiffs, die zurückgelegte Distanz und die Ausrichtung des Schiffs. Nun wird die alte Schiffsposition mit der neuen Position überschrieben und die Dynamikberechnungen sind abgeschlossen.

Das *SimpleKinemticModel* ist das nächst-komplexere Dynamikmodell. Es berechnet die Schiffsgeschwindigkeit realistischer als *FixedPositionTurn*, da eine Beschleunigungskomponente mit in die Berechnung einbezogen wird. Somit ist gewährleistet, dass das Schiff erst nach einiger Zeit seine Maximalgeschwindigkeit erreichen kann indem gleichmäßig beschleunigt wird. Die Beschleunigung ist in diesem Falle ein fester Wert, der vor Beginn der Simulation festgelegt wurde und sich im Verlauf dieser nicht ändert. Für die Berechnung des Headings wird im Gegensatz zu *FixedPositionTurn* noch die maximale Richtungsänderung des Schiffs innerhalb einer Zeitspanne (*Rate Of Turn*) mit einbezogen. Dieser Wert wird ebenfalls vor Beginn der Simulation einmalig festgelegt und bleibt anschließend unverändert. Durch diese Art der Berechnung wird die Bewegung des Schiffs etwas natürlicher und realistischer, da es sich in einem festgelegten Zeitraum nur eine begrenzte Anzahl an Grad um seine eigene Achse drehen kann. Alle übrigen Aktionen dieses Dynamik-Modells sind deckungsgleich mit denen des *FixedPositionTurn*.

Das dritte und komplexeste Dynamikmodell ist das *KinematicModel*. Es wurde auf Basis der Ergebnisse von *Jörn Beschnidt* in seiner Arbeit *Virtual Waterway* erstellt [Bes10]. Dieses Dynamikmodell erweitert die beiden vorherigen, indem neben der reinen Vorwärtsbewegung

des Schiffs auch das seitliche Abdriften vom vorgesehenen Kurs durch Umwelteinflüsse und den/die Antriebe mit einbezogen wird. Zusätzlich werden für die Berechnung des Headings sowie der Vorwärts- und Seitwärtsgeschwindigkeit wesentlich genauere Formeln verwendet. Deren Ergebnis sind Kräfte, die auf eine der Schiffsachsen wirken. Neben diesen Kräften werden zusätzlich z.B. vom Schiffsrumpf oder Ruder entstehende Widerstände mit einberechnet, wodurch realitätsnähere Ergebnisse zu erwarten sind. Die grundlegenden Formeln können in *Jörn Beschnidts Virtual Waterway* nachgeschlagen werden.

Engine Ähnlich wie beim *DynamicModel* gibt es auch bei den *Engines* verschiedene Komplexitätsstufen für die Berechnung des aktuellen Zustandes der Maschinen. Beiden gemeinsam ist die Berechnung des aktuellen Zustandes aufgrund eines Prozentsatzes, der von einem *Bahavior* gesetzt wird und die gewünschte Maschinendrehzahl angibt. Erreicht wird diese anschließend, indem mittels einer *Engine*-Beschleunigung errechnet wird, um wie viel die Umdrehungen der Maschine im angegebenen Zeitintervall erhöht oder verringert werden können und diese Werte im Anschluss hinzugefügt bzw. abgezogen werden. Die komplexere Maschine (*Engine*) unterscheidet sich von der weniger komplexen (*SimpleEngine*) dadurch, dass sie neben der Vorausfahrt auch eine Rückwärtsfahrt des Schiffs ermöglicht. Dabei werden auf Befehl des *Behaviors* bei Vorausfahrt die Umdrehungen bis auf null verringert und anschließend bei Erhöhung der Drehzahl bei Rückwärtsfahrt mit negativen Vorzeichen versehen. Beim Wechsel von Rückwärts- in Vorwärtsfahrt funktioniert dies auf umgekehrtem Wege.

Rudder Genau wie bei der *Engine* gibt es auch hier zwei verschiedene Ruder mit unterschiedlichem Umfang. Das *SimpleRudder* erlaubt es dem Schiff seine Ruder direkt auf den gewünschten Winkel anzulegen. Dabei wird der vom *Behavior* gewünschte Ruderwinkel (intendedRudderPosition) direkt angelegt. Im Gegensatz dazu wird im zweiten Ruder (*Rudder*) ein Attribut (*rateOfTurn*) verwendet, welches angibt, um wie viel sich das Ruder im gewünschten Zeitintervall ändern kann.

Environemt Um den Einfluss der*EnvironmentalImpacts* auf das Schiff berechnen und darstellen zu können gibt es die Klassen *Wind* und *Current*, die für die Berechnung der Auswirkungen von Wind bzw. Strömung auf das Schiff zuständig sind. Dafür werden mit Hilfe eines Kräfteparallelogramms und der dafür benötigten Winkel und Kräfte bzw. Geschwindigkeiten von Schiff und Umwelteinfluss das resultierende Heading und die sich

ergebende neue Geschwindigkeit des Schiffs berechnet und die Pose und Geschwindigkeit des Schiffs dementsprechend angepasst.

4.4.5 Eigenschaften des Schiffs

Abbildung 24 zeigt die Eigenschaften eines Schiffs, die vor Beginn der Simulation gesetzt werden müssen. Dazu zählen die Datenrepräsentation der Maschinen und Ruder, sowie die *PhysicalAttributes*, die für Berechnungen des *DynamicModels* benötigte Attribute zur Verfügung stellen und das Schiff charakterisieren.

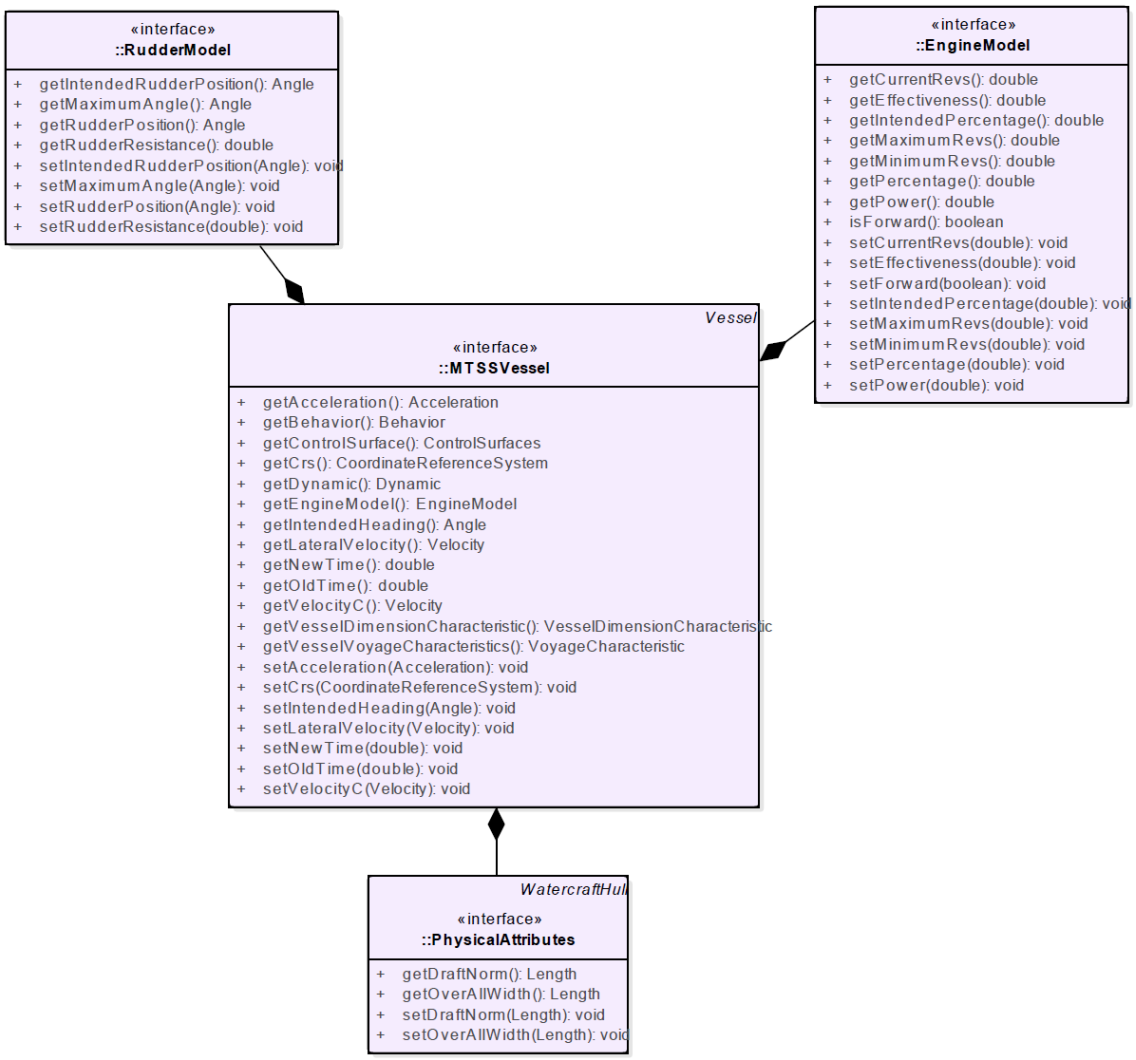


Abbildung 24: Repräsentation von Ruder, Maschine und einiger Attribute des Schiffs im Datenmodell

4.4.6 UtilCalculations

Dieses Paket stellt Methoden zur Kalkulation unterschiedlicher Werte bereit, die von Verhalten klassischerweise genutzt werden.

calculateAzimuth Im Allgemeinen bezeichnet der Azimuth einen Horizontalwinkel in eine der vier Himmelsrichtungen. Im Kontext der Nautik und damit auch der *MTSS* wird der Azimuth in der Navigation auf Norden bezogen. Die Methode *calculateAzimuth*

errechnet somit den Horizontalwinkel zwischen einer Start- und Endposition in Grad auf Norden bezogen. Hierzu werden Methoden der Java-Bibliothek *GeoTools* verwendet.

calculateDistance Die Distanz wird im Rahmen der *MTSS* zu unterschiedlichen Zwecken genutzt. Ein Beispiel hierfür ist die Überprüfung, wie weit das Schiff von dem aktuellen Wegpunkt entfernt ist und ob es sich schon mit dessen Radius schneidet und somit der nachfolgende Wegpunkt angefahren werden muss. Im Allgemeinen errechnet die Methode *calculateDistance* die Entfernung zwischen zwei gegebenen Koordinatenpunkten. Das Ergebnis dieser Methode wird in nautischen Meilen (nm) angegeben.

isPositionOnLand *isPositionOnLand* bietet für Schiffe die Möglichkeit zu prüfen, ob sich ein angegebener Koordinatenpunkt an Land befindet. Ein auf das jeweilige Schiff bezogene relatives Koordinatensystem beschreibt die Ausmaße eines jeden Schiffes, also die Länge und Breite. Nach einer Umwandlung der relativen Koordinaten in das verwendete globale Koordinatensystem ist es möglich aus dem Schiff eine Geometrie zu generieren. Die jeweilige Position ist genau dann auf Land, wenn eine Schnittmenge der Schiffsgemeotrie mit der Geometrie der Landmasse existiert.

calculateNewPosition Unter der Verwendung der aktuellen Position, einer zurückgelegten Distanz und dem Kurswinkel, in welche Richtung die Distanz zurückgelegt wurde, errechnet die Methode *calculateNewPosition* die neue Position im globalen Koordinatensystem, die sich aus den genutzten Variablen ergibt.

initializeChartInformations Von dieser Methode wird das Einlesen der Karteninformationen aus GML-Dateien übernommen, wenn eben solche Informationen für das Schiff verfügbar sind. Die Informationen werden, falls vorhanden, in dem Subverhalten *ChartInformation* gespeichert und sind für das jeweilige Schiff nutzbar.

calcPerpendicular Während der Berechnung zieht diese Methode eine Senkrechte zwischen zwei angegebenen Punkten. Unter der Angabe einer Distanz und Richtung wird dann abschließend ein Koordinatenpunkt entweder rechts oder links von der Senkrechten ausgegeben. Diese Methode wird bspw. genutzt um ein Schiff entlang von Bojen navigieren zu lassen.

checkForCollision Das Ergebnis dieser Methode ist ob sich zwei Geometrien, genauer gesagt *LineStrings*, überschneiden. Dies kann dazu verwendet werden um beispielsweise eine Kollision zwischen einem Schiff und einer Küstenlinie vorherzusagen.

transformLineString In der *GML* sind üblicherweise Küstenlinien als *LineString*-Geometrien vermerkt. Das Format dieser Geometrien entspricht nicht dem Format, das in der **MTSS** verwendet wird. Aus diesem Grund ist eine Transformation notwendig, welche durch diese Methode ermöglicht wird. Aus einer übergebenen *LineString*-Geometrie in dem Format aus der vorliegenden *GML* wird eine *LineString*-Geometrie in dem passenden *MTSS*-Format.

generateLineString Die Methode ermöglicht es eine einfache *LineString*-Geometrie zwischen zwei gegebenen Punkten zu generieren. Dies kann beispielsweise genutzt werden um festzustellen ob die direkte Kursverbindung von einem Schiff zu einem angestrebten Wegpunkt mit einer Küstenlinie kollidiert.

Circle Für das Rechtsfahrtgebot muss ein Kreis um die Wegpunkte gezeichnet werden, um die Gegebenheiten um den Wegpunkt zu bestimmen. Befindet sich der Wegpunkt auf "offenem Meer" (Innerhalb des übergebenen Radius' befindet sich kein Land) fährt das Schiff direkt zum nächsten Wegpunkt ohne sich rechts am Land zu halten.

4.5 World

Die Komponente World beinhaltet den gesamten Zustand des globalen World-Modells und stellt diesen oder Ausschnitte davon anderen Komponenten bereit. Dafür besitzt sie Schnittstellen, über die Services zur Selektion der durch andere Komponenten angefragten Daten angekoppelt werden. Das Federate "World" enthält dazu das *World-Model*. Außerdem liefert das *World Federate* Sensor Daten, die von einzelnen Services für die Schiffe in der Simulation berechnet werden. Diese Sensor Daten werden von den anderen Federates zur Laufzeit der Simulation zur weiteren Berechnung benutzt.

4.5.1 Datenflussdiagramm

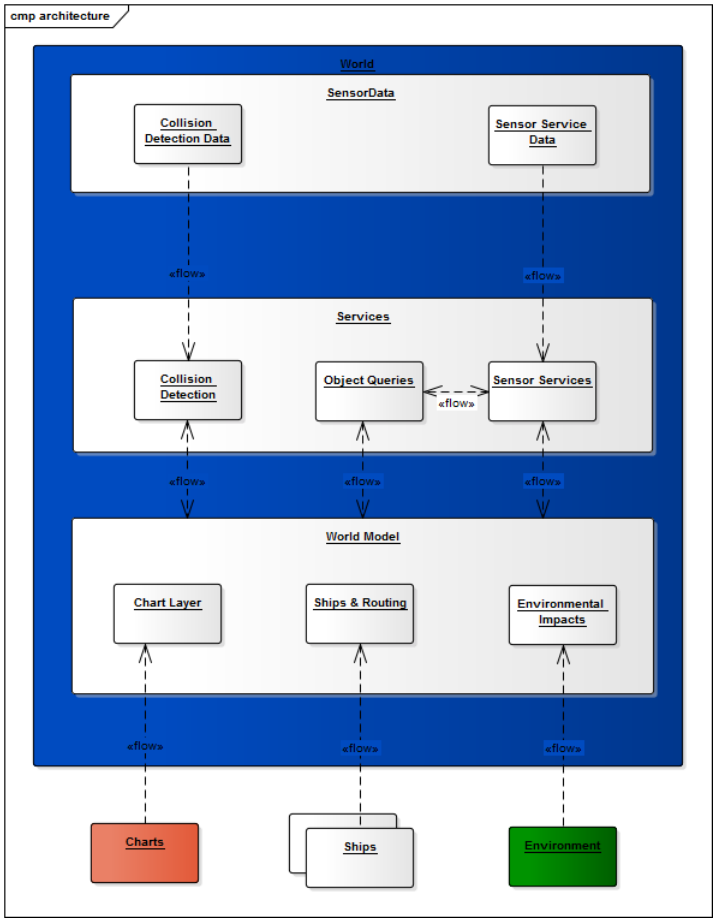


Abbildung 25: Datenflussdiagramm der World- Komponente

Wie in der Grafik Abbildung 25 dargestellt erhält der World Federate World-Daten, Daten von der Charts Komponente, Schiffsdaten von dem Ship Federate, sowie Umweltdaten von dem Environment Federate. Diese Informationen dienen für weitere Berechnungen in den verschiedenen Services des World Federates, wie die Kollisionsberechnung oder dem Radar Sensor Service. Diese Service-Daten werden für die Förderation veröffentlicht.

4.5.2 Sequenzdiagramm

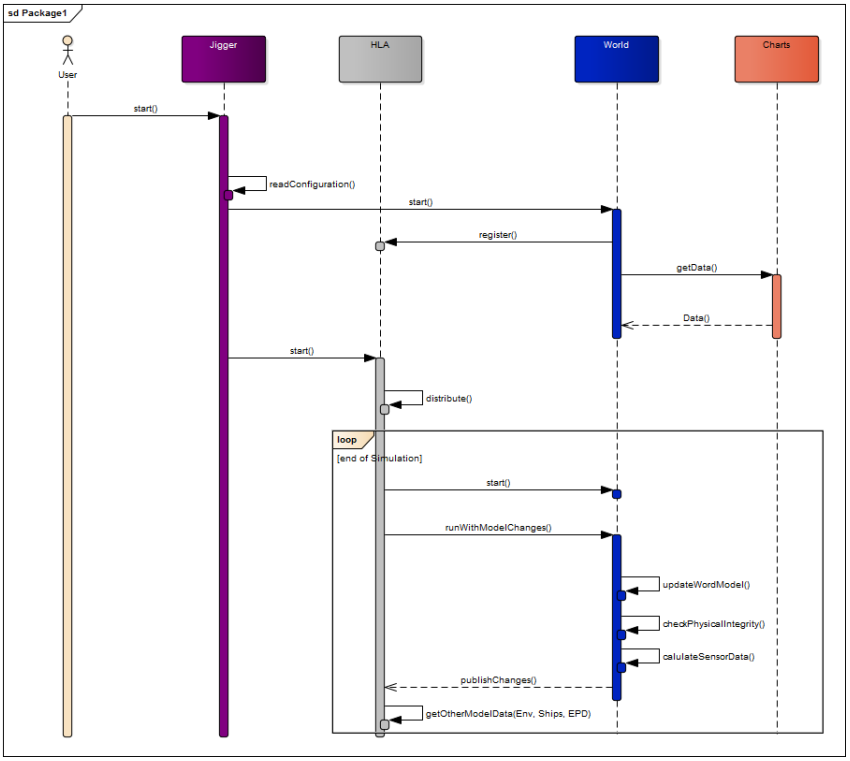


Abbildung 26: Sequenzdiagramm der World- Komponente

In der Abbildung 26 wird das Sequenzdiagramm für die World Komponente dargestellt. Der erste Schritt ist der Start von Jigger durch den Benutzer und das Auslesen der Konfigurations-Datei, durch die beschrieben wird, welche Services für die Simulation ausgeführt werden sollen. Danach werden die Chart-Daten von der Charts Komponente angefragt. Daraufhin beginnen die Simulationsschritte, indem die Services gestartet werden und diese Daten erzeugen sowie diese dann an die anderen Federates veröffentlichen.

4.5.3 Sensor Services

Im folgendem Abschnitt werden die einzelnen Services und ihre Funktion genauer erläutert.

AisSensorService Der *Ais Sensor Service* berechnet für jedes einzelne Schiff ein kreisförmiges Polygon, welches den Sensor Service darstellen soll. Der Radius des Polygons kann vor der

Laufzeit in dem Konfigurator des *World Federates* festgelegt werden. In jedem Simulationsschritt überprüft der Service daraufhin, ob sich andere Schiffe innerhalb des Polygons befinden. Ist dies der Fall, wird ein Datensatz mit dem Schiff, welches mit dem aktiven Polygon und den in dem Polygon befindlichen Schiffen, angelegt. Am Ende des Simulationsschritt wird der Datensatz "AisSensorServiceData" über *HLA* an den *EPD* und *Ships* Federate veröffentlicht. Die *EPD* kann dann den Ais Sensor Service anzeigen.

Collision Detection Service Der *Collision Detection Service* ist für die Erkennung von Schiffskollisionen zuständig. Diese Überprüfung findet für jeden Simulationsschritt statt. Der Service berechnet dafür ein Polygon aus den *Bounds* (Grenzen) eines Schiffes. Dies wird für alle in der Simulation befindlichen Schiffe durchgeführt und daraufhin überprüft, ob sich jeweils zwei Polygone überschneiden. Ist dies der Fall, sind die beiden Schiffe miteinander kollidiert und ein Datensatz mit den jeweiligen *Vessel-Objekten* wird erstellt. Dieser wird über *HLA* an den *EPD* und *Ship* Federate veröffentlicht. Die Kollision kann dann auf der *EPD* angezeigt werden, sowie die Status der Schiffe in der Schiffs Komponente angepasst werden, zum Beispiel auf "ist kollidiert".

CompassService Der *Compass Service* stellt den Kompass eines Schiffes dar. Der Service liest dafür in jedem Simulationsschritt für jedes Schiff das *Heading* (Steuerkurs) aus. Das *Heading* wird daraufhin an die anderen Federates über HLA veröffentlicht. Der Service kann dafür genutzt werden, dass *Heading* eines Schiffes während der Laufzeit zu ändern und somit eine Störung des Kompass zu simulieren.

PhysicalIntegrityService Der *Physical Integrity Service* ist für die Überprüfung von Kollisionen zwischen Schiffe und Land implementiert. Die von dem *Chart-Server* übergebene *GML-Datei* wird zum Start des *World Federates* geparsed und kann somit zur Laufzeit auf Land Daten überprüft werden. Aus den Geometrien der Daten werden Polygone erstellt und in jedem Simulationsschritt wird die Kollision zwischen den Schiffspolygonen und den Landpolygonen überprüft. Findet eine Kollision mit Land statt, wird der Datensatz mit dem kollidiertem Schiff an den *EPD* und *Ships Federate*, mittels der *HLA* übertragen. Die Kollision kann dann auf der *EPD* angezeigt werden, sowie der Status des Schiffes in der Schiffs Komponente geändert werden, zum Beispiel auf "ist kollidiert".

RadarService Der *Radar Service* stellt das Radar eines Schiffes dar. Der Radar eines Schiffes erkennt andere Schiffe und Bojen und wird als kreisförmiges Polygon berechnet.

Die Bojen werden aus einer "GML Datei", bereit gestellt von der *Charts Komponente*, geparsed und ausgelesen. Die Größe des Radars lässt sich im Konfigurator des *World Federates* anpassen. Für jeden Simulationsschritt wird daraufhin überprüft, ob sich andere Objekte in dem Radius des Radars befinden. Ist dies der Fall wird ein Datensatz *RadarSensorServiceData* erstellt und die Objekte, wie Schiffe und Bojen werden dann über *HLA* an den *EPD* und *Ships Federate* übertragen. Die *EPD* kann dann den "Radar Service" als Polygon und die darin befindlichen Objekte anzeigen.

SonarSensorService Der *Sonar Sensor Service* stellt das Sonar eines Schiffes dar und erkennt in einem berechneten Polygon Objekte, wie Steinformationen und Schiffwracks, die aus einer "GML Datei" geparsed und ausgelesen werden. Die "GML Datei" wird von der *Charts Komponente* bereit gestellt. Befinden sich Steine oder Wracks in dem Radius des Polygons, wird ein Datensatz *SonarSensorServiceData* erstellt, welcher die Objekte enthält. Die Daten werden über *HLA* an den *EPD* und *Ships Federate* übertragen. Die *EPD* kann das "Sonar Polygon" mit den darin befindlichen Objekten anzeigen. Des Weiteren kann der *Ships Federate* die Daten zum Beispiel für eine Routenberechnungen eines Schiffes berücksichtigen.

ViewSensorService Der *View Sensor Service* stellt die Sicht eines Menschen, zum Beispiel des Kapitäns, auf einem Schiff dar. Dabei kann die Reichweite in dem Konfigurator des *World Federates* festgelegt werden. Der Service berechnet dafür ein kreisförmiges Polygon und überprüft ob sich andere Objekte, wie Schiffe, Land oder Bojen in dessen Radius befinden. Im Falle einer Überschneidung mit dem Polygon wird ein *ViewSensorServiceData* Datensatz erstellt, welcher alle in dem Raidus befindlichen Objekte enthält. Dieser wird über *HLA* an den *EPD* und *Ships Federate* übertragen. Die *EPD* kann daraufhin die Sicht eines Schiffes darstellen, mit dem Polygon und die darin enthaltenen Objekte anzeigen. Außerdem kann der *Ships Federate* die Daten zum Beispiel für die Berechnung von Routenberechnungen berücksichtigen.

4.5.4 Klassendiagramme

Im folgenden werden die wichtisten Klassendiagramme des World-Modells sowie das Sensor-Data-Modell beschrieben und dargestellt. Als erstes folgen die Interface- und daraufhin die implementierten Klassen des World-Modells. Die Abbildung 27 visualisiert das Interface *IService*. Es veranschaulicht, dass jeder Sensor von der Klasse *IService* erbt. Diese hat

die notwendigen Methoden implementiert, in der die Ausführung des Services umgesetzt werden soll.

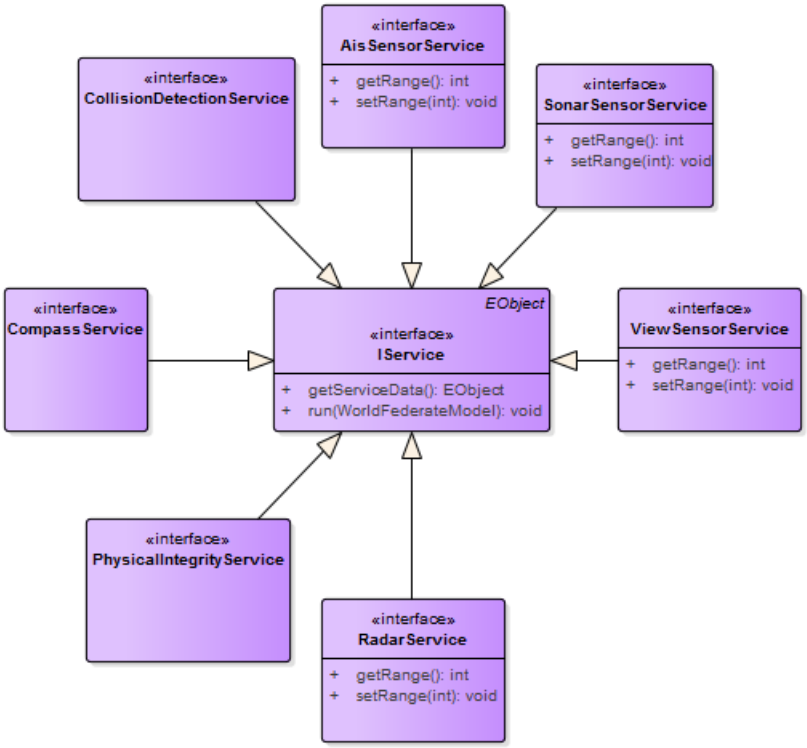


Abbildung 27: World-Interface: IService

Folgend werden die jeweiligen implementierten Service-Klassen der World abgebildet. Funktionen die von verschiedenen Services genutzt werden, sind im Paket *util*, in eigene Klassen, ausgelagert. Ein Beispiel dafür wäre die Klasse mit ihrer Methode *CreateCircle*. Diese wird von verschiedenen Services, wie dem *Radar Sensor Service* oder *View Sensor Service*, benötigt. Am Besiepiel des *Radar Sensor Service* wird die Methode in *CheckRadar* aufgerufen.

Die folgende Abbildung 28 visualisiert die Klasse des *Radar Service* da. Mittels der Funktion *checkRadar* wird überprüft welche Objekte sich innerhalb des Radius, des aktuellen Schiffes, befinden. Dabei handelt es sich zum Beispiel um andere Schiffe oder Bojen.

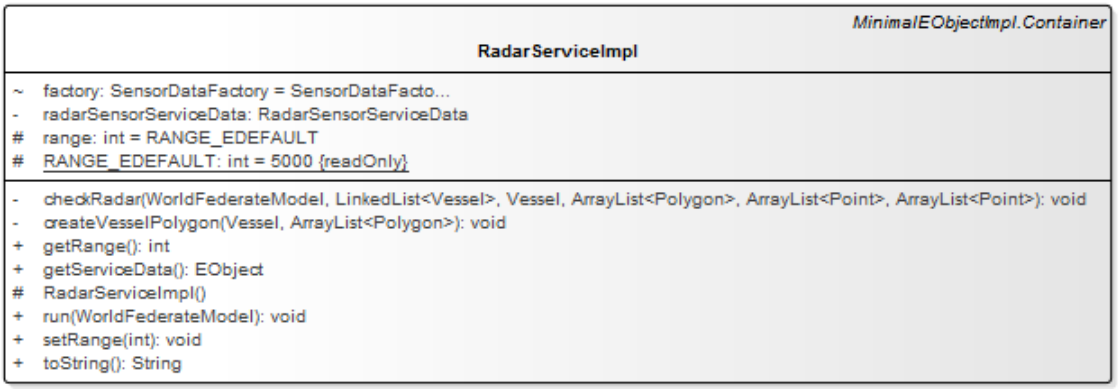


Abbildung 28: RadarServiceImpl

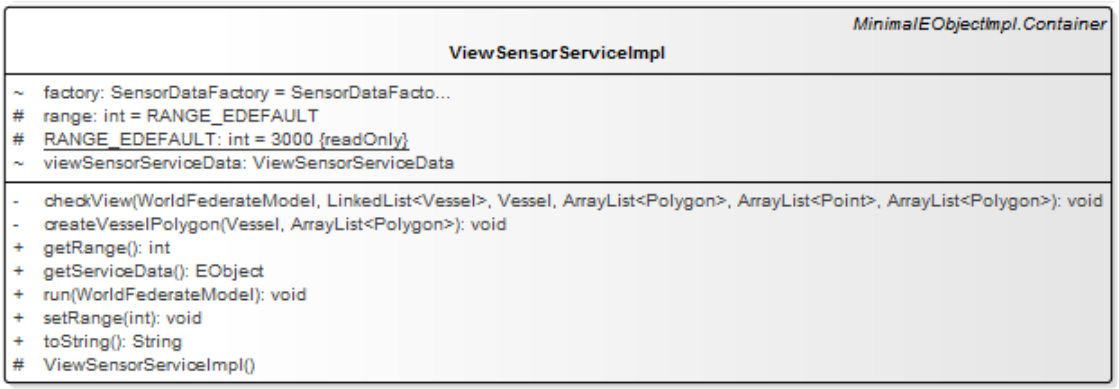


Abbildung 29: ViewServiceImpl

Der *ViewService*, Abbildung 29 ähnelt dem *Radar Service*. Dieser entdeckt mittels der Methode *checkView* andere Schiffe, Bojen oder aber auch Land. Ansonsten entspricht der Service dem gleichen Aufbau wie der *Radar Service*.

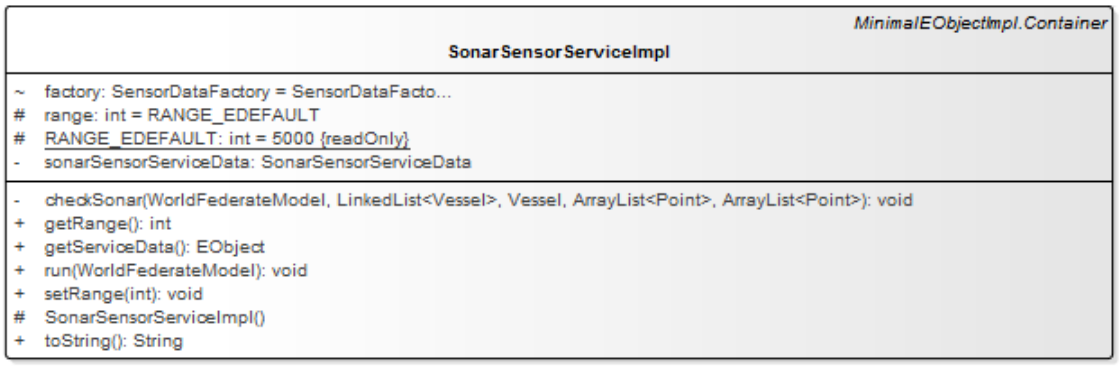


Abbildung 30: SonarServiceImpl

Wie schon die anderen beiden Services, überprüft der *Sonar Service* ob sich Objekte innerhalb eines Radius befinden. Dabei handelt es sich aber um Unterwasserobjekte. Anhand der Methode *checkSonar* werden in dem Service Wracks und Unterwasserfelsen gefunden, die in der *GML* hinterlegt sind.

Des Weiteren gibt es noch den *Ais Sensor Service*. Dieser ist eine einfachere Form des *Radar Service*. In der Methode *checkAis* werden ausschließlich Schiffe innerhalb des Suchradius entdeckt. Dieser Service wurde implementiert um bei Filterungen von Schiffsdaten in anderen Services, alle Informationen der Schiffe zu übertragen. Zum Beispiel könnte die View nur die Größe und Position des Schiffes übermitteln, die *Ships-Komponente* benötigt aber mehr Informationen und kann diese dann über den *Ais Sensor Service* abfragen. Die Filterung der kam aber wegen dem *Postico Bug* (siehe Entwicklerhandbuch) nicht zustande.

Innerhalb der Methode *checkCollision* wird zunächst die sogenannte *Bounds* des Schiffes bestimmt. Anschließend wird bei jedem Simulationsschritt überprüft, ob zwei oder mehrere Schiffe miteinander kollidiert sind. Das geschieht durch die Überprüfung, ob sich die *Bounds* verschiedener Schiffe überschneiden.

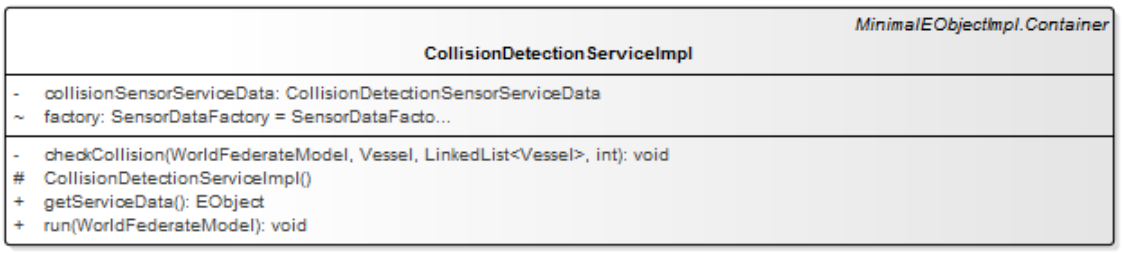


Abbildung 31: CollisionServiceImpl

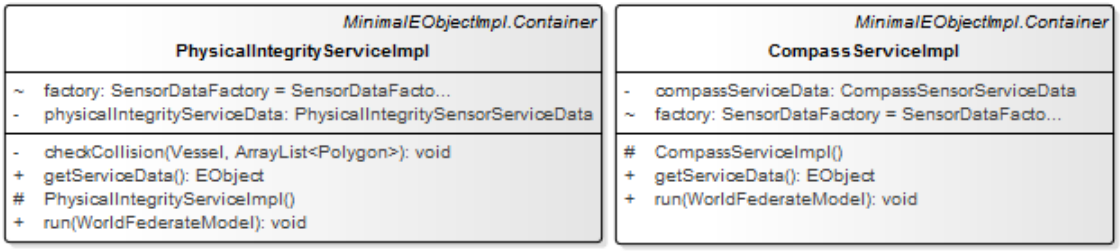


Abbildung 32: PhysicalCompassServiceImpl

Die Abbildung 32 veranschaulicht zwei Klassen. Zum einen den *Physical Integrity Service* und zum anderen die des *Compass Service*. Der *Physical Integrity Service* hat auch eine Methode namens *checkCollision* implementiert. Diese überprüft in jedem Simulationsschritt, ob ein Schiff auf Land aufgefahren ist.
Der *Compass Service* hat in der *run*-Methode seine derzeitige Funktion implementiert. Dieser holt sich die aktuelle Ausrichtung und übergibt es wieder an die *HLA*.

Die Abbildung 33 und veranschaulicht die verschiedenen *Sensor Service Data*, die jeweils eine *EList* enthalten vom Typ ihres entsprechenden *SensorDataObjekts*. Diese enthält die übergebenen Objekte der *World-Services*. Jeder einzelne *Service* benötigt ein *Sensor Service Data* sowie das entsprechende *Sensor Data Objekt*. Dieses Listenobjekt wird dann später in die *HLA* gepusht.
Abbildung 34 visualisiert, welche Objekte die jeweiligen Klassen der *Sensor Service Data* implementieren.

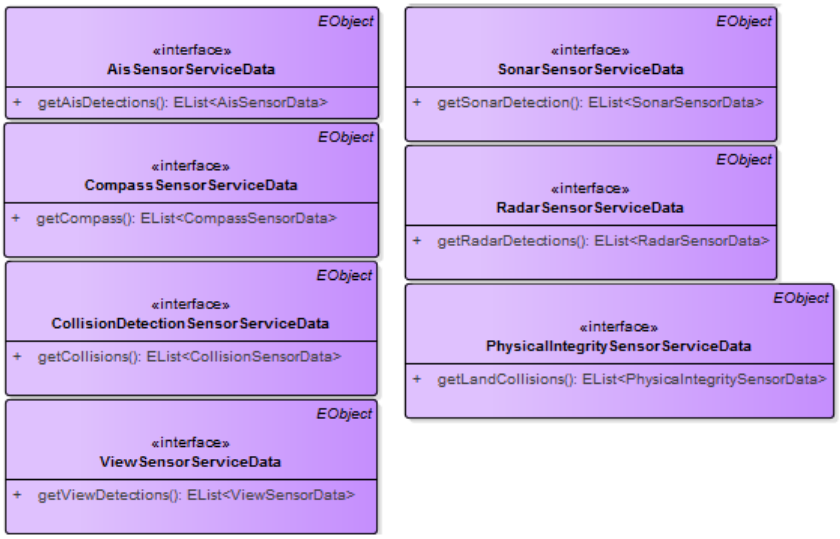


Abbildung 33: SensorData-Interface 1

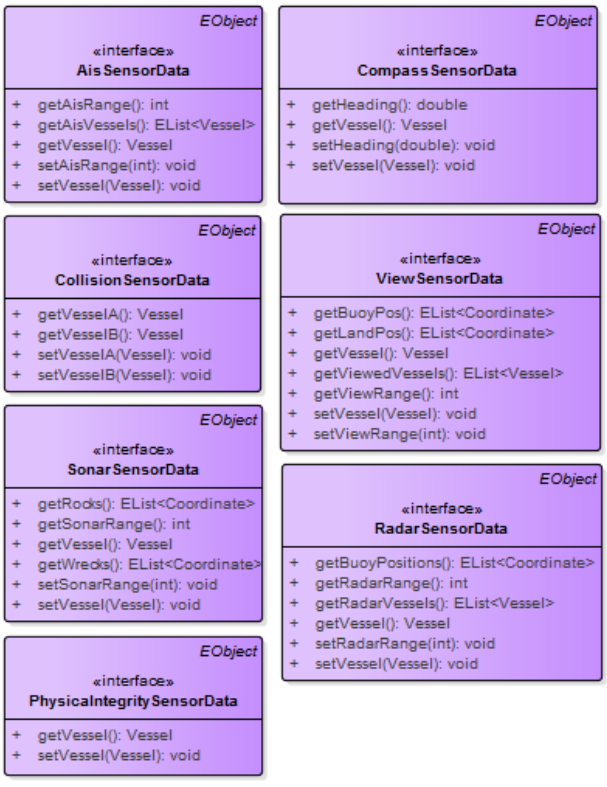


Abbildung 34: SensorData-Interface 2

5 Technologien

In diesem Abschnitt werden die Softwaretechnologien vorgestellt, die im Verlaufe des Projektes Anwendung fanden. Zu jeder Software folgt eine kurze Begründung für die Verwendung der Software.

5.1 Atlassian Confluence

Confluence ist ein webbasiertes, von Atlassian entwickeltes und kostenpflichtiges Dokumentationssystem. Mit Hilfe von Confluence können Informationen gesammelt, verteilt und eingesehen werden. Es handelt sich also um ein kollaboratives Informationsmanagementtool. Die Nutzung von Confluence wurde der Projektgruppe durch den Product Owner ermöglicht. Durch die weitgehend intuitive Bedienung erweist sich die Nutzung von Confluence nach einer gewissen Einarbeitungszeit in das System als leicht. Relevante Informationen wurden über den gesamten Projektverlauf im Confluence gesammelt und organisiert. Hierunter fallen Szenarien, Protokolle, Diskussionen über das weitere Vorgehen im Projekt, Meinungsaustausch zum aktuellen Stand der Implementierung (Bugs, allgemeine Verbesserungsvorschläge, usw.) und die Terminplanung für Gruppentreffen.

5.2 Atlassian Jira

Jira ist eine von Atlassian entwickeltes, webbasiertes Projektmanagement Tool. Jira erlaubt es auf Grundlage des agilen Projektmanagements Scrum Aufgaben zu planen, zu organisieren und zu verwalten. Jira ist mit diversen Plugins erweiterbar, sodass z.B. eine Anbindung zu Confluence möglich ist und Jira-Task mit Inhalten aus Confluence verknüpft werden können.

5.3 Ecore / EMF

Ecore ist ein Metamodell, welches die Objektmodelle im EMF beschreibt. Ecore ist die Modellierungssprache, die für das Datenmodell verwendet wird. Das EMF (Eclipse Modeling Framework) ist ein Java-basiertes Framework, das der automatisierten Erzeugung von Quellcode aus Modellen dient.

5.4 Eclipse Mars

Eclipse ist ein open-source Framework, das auf Java basiert. Ursprünglich wurde Eclipse als integrierte Entwicklungsumgebung (IDE) für die Programmiersprache Java genutzt, inzwischen jedoch wird es wegen seiner Erweiterbarkeit auch für viele andere Entwicklungsaufgaben eingesetzt. Es gibt eine Menge von kommerziellen aber auch opensource Plug-Ins für Eclipse. Innerhalb der Projektgruppe wurde die Version Mars.2 verwendet. Begründung: –¿ Aus Gründen!

5.5 HLA

Die verteilten Simulationskomponenten verwenden ein gemeinsame genutztes Speicher-Modell, um die Interoperabilität auf Basis des *High Level Architecture*(HLA) Standard für verteilte Simulations-Systeme zu gewährleisten. HLA ist eine vom Verteidigungsministerium der Vereinigten Staaten von Amerika entwickelte Softwarearchitektur zur Übermittlung und Integration von Daten in verteilten Simulationen. Diese Architektur wird verwendet, da sie sich als Standard international durchgesetzt hat und auch vom Institute of Electrical and Electronics Engineers (IEEE) im Standard 1516 festgelegt wurde. HLA sieht vor, dass die gesamte Simulation (Fedeation) in mehrere kleinere Teilsimulationen (Federates) aufgespalten wird. Um die Kommunikation zwischen den einzelnen Federates zu ermöglichen, sind diese über eine Laufzeitinfrastruktur (Runtime Infrastructure, RTI) mit einander verbunden.

Die Umsetzung der HLA/RTI ist mitthilfe von portico erfolgt. Bei portico handelt es sich um eine Open-Source Implementierung für die vollkommene Unterstützung von HLA/RTI Implementierungen und ist plattformunabhängig.

5.6 SVN

Subversion ist eine Open-Source-Software zur Versionsverwaltung von Dateien und Verzeichnissen. Mit diesem Source-Code-Management-System ist Zusammenarbeit mehrerer Entwickler an einer Datei aus verschiedenen Standorten gewährleistet. Durch die Möglichkeit, vorherige Quell-Code-Versionen aufzurufen, kann der Entwicklungsfortschritt verbessert und verglichen werden.

5.7 Tycho

Die MTS wurde als Tychoprojekt implementiert. Dies bedeutet, dass es sich bei den Projekten sowohl um ein Maven als auch um ein Eclipse-Plugin-Projekt handelt. Dies bedeutet das Abhängigkeiten die bei Mavenprojekten in der Regel in den POM-Dateien angegeben werden, in einere extra Metadatei verwaltet werden. Der Vorteil ist, das sich die Projekte als Plugin einbinden lassen um auf verschiedene Klassen über die Einbindung als Plugin zugegriffen werden kann, das Projekt aber auch als Maven-Projekt gebuildet und als eigenständiges Projekt ausgeführt werden kann.

5.8 S-100

Bei dem S-100 handelt es sich um einen zeitgenössischen hydrographischen Daten Standard, welcher ein großes Spektrum an hydrographischen, digitalen Datenquellen unterstützt. Er ist vollkommen an die internationale ISO 19000 Serie angepasst. Der S-100 erweitert somit den Umfang des existierenden S-57. Der S-100 ist wesentlich flexibler, als der S-57 und stellt unter anderem Komponenten für die Nutzung der Symbolik und Rasterdaten bereit. Außerdem verbessert und erweitert der S-100 die Metadaten und verwendet diverse Codierungsformate [Pow11].

6 Akzeptanztests

In diesem Kapitel folgen die Akzeptanztests der einzelnen Komponenten, um sicherzustellen, dass die Software für den Benutzer die gewünschten Ergebnisse liefert.

6.1 Charts

Testfall	erfüllt
In den Einstellungen der EPD wird der WMS-Link zum Chartserver eingefügt. Leuchtet die Lampe für den WMS grün und es erscheint eine Chartlayer in S-52 konformer Darstellung über der Standardkarte der *EPD* gilt der Testfall als erfüllt.	ja
Wenn beim Starten des *Ships* Federates mit Konsole die Fehlermeldung "Can't load gml" **nicht** erscheint, oder wenn die Schiffe mit Behavior DriveRight am Land entlangfahren, oder wenn Schiffe die mit dem KollisionSensorService der *World* ausgestattet sind gegen eine Landfläche fahren und direkt anhalten, gilt dieser Testfall als erfüllt.	ja

6.2 EPD

Testfall	erfüllt
Die EPD kann manuell, zu jeder Zeit gestartet werden.	ja
Sobald die Simulation gestartet wird, verbindet sich die EPD mit der HLA.	ja
Wenn die EPD mit HLA verbunden ist, werden folgende Informationen auf der EPD angezeigt:	
- Pick-up Report / Mouse-over	ja
- Simulierte Schiffe	ja
- Routen	ja
- Kollidierte Schiffe	ja
- Navigationszeichen	ja
- Wind Layer, mit METOC-Forcastpoint /Mouse-over	ja
- Current Layer, mit METOC-Forcastpoint /Mouse-over	ja
- Tide Layer, mit METOC-Forcastpoint /Mouse-over	ja
Sollte die EPD nicht starten, so wird eine Pop-up Menü den Benutzer darauf hinweisen.	ja

6.3 World

Testfall	erfüllt
Die World kann zu jeder Zeit gestartet werden.	ja
Sobald die Simulation gestartet wird, verbindet sich die World mit der HLA.	ja
Jeder Service funktioniert eigenständig.	ja
Alle Objekte der Welt können durch die verschiedenen Services bereitgestellt werden.	ja
Folgende Services und ihre Funktion stehen zur Verfügung:	
AISSensorSerivce: ermittelt alle Vessel im definierten Radius und übergibt diese an die HLA.	ja
RadarSensorService: ermittelt alle Vessel und Boyen im definierten Radius, die in der GML hinterlegt sind und übergibt diese an die HLA.	ja
SonarSensorService: ermittelt alle Wracks und Unterwasserfelsen im definierten Radius, die in der GML hinterlegt sind und übergibt diese an die HLA.	ja
ViewSensorService: ermittelt alle Vessel, Land und Boyen im definierten Radius, die in der GML hinterlegt sind und übergibt diese an die HLA.	ja
CollisionSensorService: ermittelt jede Kollision zwischen verschiedenen Vessel und übergibt genau diese an die HLA.	ja
CompassSensorService: ermittelt die Ausrichtung des übergebenen Vessel und übergibt dieses wieder an die HLA.	ja
PhysicalIntegritySensorSerive: ermittelt jedes Vessel was auf Land gestoßen ist und übergibt jedes an die HLA.	ja

6.4 Environment

Testfall	erfüllt
Es ist möglich, ein Grib-File zu importieren, um realitätsnahe Umwelteinflüsse für die Simulation verwenden zu können. Kann ein Grib-File im Environment-Editor importiert werden und Erscheinen mehrere Polygone auf der Karte des Environment-Editors, gilt der Testfall als erfüllt.	ja
Es ist möglich, einem Polygon **ein** Umwelteinfluss zuzuweisen. Kann beim Erstellen eines Polygons ausschließlich ein Umwelteinfluss selektiert werden, gilt der Testfall als erfüllt.	ja
Es ist möglich, ein Polygon mit dem Umwelteinfluss Wind zu erstellen. Es erscheint beim Erstellen ein Pop-Up-Fenster zur Auswahl des Umwelteinflusses. Der Testfall gilt als erfüllt, wenn in der Listenansicht des Editors ein Listeneintrag mit einem Wind-Polygon erscheint.	ja
Es ist möglich, ein Polygon mit dem Umwelteinfluss Strömung zu erstellen. Es erscheint beim Erstellen ein Pop-Up-Fenster zur Auswahl des Umwelteinflusses. Der Testfall gilt als erfüllt, wenn in der Listenansicht des Editors ein Listeneintrag mit einem Strömung-Polygon erscheint.	ja
Es ist möglich, ein Polygon mit dem Umwelteinfluss Tide zu erstellen. Es erscheint beim Erstellen ein Pop-Up-Fenster zur Auswahl des Umwelteinflusses. Der Testfall gilt als erfüllt, wenn in der Listenansicht des Editors ein Listeneintrag mit einem Tide-Polygon erscheint.	ja
Es ist möglich, ein dynamisches Szenario zu erstellen. Der Umwelteinfluss eines Polygons ändert sich über die Zeit. Es müssen zwei unterschiedliche Werte zu verschiedenen Zeitpunkten für den Umwelteinfluss bestimmt werden. Werden die Werte über die Zeit interpoliert, gilt der Testfall als erfüllt.	ja
Es ist möglich, ein statisches Szenario zu erstellen. Der Umwelteinfluss eines Polygons ändert sich nicht über die Zeit. Es müssen zwei identische Werte zu verschiedenen Zeitpunkten für den Umwelteinfluss bestimmt werden. Bleiben die Werte im Zeitverlauf gleich, gilt der Testfall als erfüllt.	ja
Es ist möglich, ein Polygon auf die Karte zu zeichnen. Dafür müssen mindestens drei beliebige Punkte auf der Karte selektiert werden. Der letzte Punkt wird auf den ersten Punkt gesetzt, um die Zeichnung des Polygons abzuschließen. Erscheint auf der Karte ein farbiges Polygon mit den gesetzten Punkten, gilt der Testfall als erfüllt.	ja

6.5 Behavior

Testfall	erfüllt
Wenn *FollowFixedRouteControlType* als *ControlType* ausgewählt ist, folgt das Schiff einer Route auf direktem Weg. Dabei ignoriert es Landflächen oder andere Schiffe als Hindernisse. Auf der EPD ist dieses Verhalten sichtbar. Beim Erreichen des letzten Wegpunkts reduziert das Verhalten die Maschinenleistung und hält an. Dies ist ebenfalls auf der EPD sichtbar.	ja
Wenn *DriveRight* als *ControlType* ausgewählt ist, muss das jeweilige Schiff das Subverhalten *ChartInformation* ausgewählt sein. Falls dies nicht so ist, ist in der Konsole die Ausgabe zu finden, dass das Schiff das entsprechende Verhalten nicht besitzt und somit die *GML* parsen kann.	ja
Wenn *DriveRight* als *ControlType* und das Schiff das Subverhalten *ChartInformation* besitzt, fährt das Schiff seine vorgegebene Route ab und hält sich dabei entweder an der rechten Landseite oder an die entsprechenden farblich markierten Bojen, was auf der EPD unter der Verwendung des passenden *WMS*-Links sichtbar ist.	ja
Wenn *FollowRouteOnWater* als *ControlType* ausgewählt wurde, so ist auf der EPD sichtbar, wie das Schiff seiner vorgegebenen Route folgt und dabei nur auf Wasserflächen navigiert.	ja
Wenn *FollowRouteOnWater* als *ControlType* ausgewählt wurde, so muss das jeweilige Schiff das Subverhalten *ChartInformation* ausgewählt sein. Falls dies nicht so ist, ist in der Konsole die Ausgabe zu finden, dass das Schiff das entsprechende Verhalten nicht besitzt und somit die *GML* parsen kann.	ja
Wenn dem Schiff ein *ControlTypeWithFixedRoute* zugewiesen wurde, kann das Schiff das Subverhalten *RouteFinding* besitzen und eine Route zwischen gegebenen Wegpunkten berechnen. Neben einer Konsolenausgabe ist auf der EPD die errechnete Route sichtbar.	ja
Wenn *KeyboardControlType* als *ControlType* ausgewählt wurde, kann der Nutzer über Steuerangaben das Schiff manövrieren. Die Reaktion des Schiffes auf die Eingabe der Steuersignale ist auf der EPD sichtbar.	ja
Wenn das Subverhalten *ChartInformation* ausgewählt wurde und das Parsen der *GML* nicht erfolgreich war, so ist eine entsprechende Meldung auf der Konsole sichtbar.	ja
Wenn *OneRudderControl* als Ruderkontrollverhalten ausgewählt wurde, so wird zu dem errechneten angestrebten Kurswinkel der entsprechende Ruderstellwinkel errechnet. Der errechnete Wert wird in dem jeweiligen *Vessel*-Objekt unter *intendedRudderAngle* abgespeichert.	ja
Wenn *MutualEngineControl* ausgewählt wurde, wird der von dem jeweiligen *ControlType*-Verhalten errechnete prozentuale Wert der Maschinenleistung bei den Maschinen gesetzt.	ja

6.6 Dynamik

Testfall	erfüllt
Wenn *FixedPositionTurn* als Dynamik ausgewählt ist, nimmt das Schiff ohne Zwischenschritte die vom Verhalten gewünschte Ausrichtung an. Dies ist auf der EPD zu sehen, wenn sich die Schiffsausrichtung sprunghaft ändert.	ja
Wenn *FixedPositionTurn* als Dynamik ausgewählt ist, nimmt das Schiff ohne Berücksichtigung einer Beschleunigung die vom Verhalten gewünschte Geschwindigkeit an. Dies ist auf der KeyboardControl zu sehen, wenn sich die Schiffsgeschwindigkeit aus dem Stand heraus innerhalb kürzester Zeit auf die Maximalgeschwindigkeit erhöht.	ja
Wenn *SimpleKinematicModel* als Dynamik ausgewählt ist, nimmt das Schiff die vom Verhalten gewünschte Ausrichtung nur schrittweise nach und nach an. Dies ist auf der EPD zu sehen, wenn sich die Schiffsausrichtung langsam und gleichmäßig ändert.	ja
Wenn *SimpleKinematicModel* als Dynamik ausgewählt ist, nimmt das Schiff mit Berücksichtigung einer Beschleunigung die vom Verhalten gewünschte Geschwindigkeit an. Dies ist auf der KeyboardControl zu sehen, wenn sich die Schiffsgeschwindigkeit aus dem Stand heraus erst langsam auf die Maximalgeschwindigkeit erhöht.	ja
Wenn *KinematicModel* als Dynamik ausgewählt ist, nimmt das Schiff die vom Verhalten gewünschte Ausrichtung abhängig von der Größe des Ruderwinkels und der aktuellen Geschwindigkeit schneller oder langsamer an. Dies ist auf der KeyboardControl zu sehen, wenn sich die Schiffsausrichtung anfangs langsam und mit steigender Geschwindigkeit und steigendem Ruderwinkel schneller ändert.	ja
Wenn *KinematicModel* als Dynamik ausgewählt ist, nimmt das Schiff die vom Verhalten gewünschte Geschwindigkeit unter Berücksichtigung verschiedene Schiffswiderstände, dem Vortrieb durch die Maschine und der im KinematicModel errechneten Beschleunigung an. Dies ist auf der KeyboardControl zu sehen, wenn sich die Schiffsgeschwindigkeit aus dem Stand heraus beim Anfahren zuerst langsam und später, wenn sich das Schiff bereits fortbewegt, schneller erhöht.	ja
Wenn *KinematicModel* als Dynamik und Engine als Maschine ausgewählt sind, besitzt das Schiff die Fähigkeit, sich rückwärts zu bewegen. Dies ist auf der KeyboardControl zu sehen, wenn die Geschwindigkeit mit einem negativen Vorzeichen versehen ist.	ja

Testfall	erfüllt
Wenn *SimpleRudder* als Ruder ausgewählt ist, wird der vom Verhalten gewünschte Ruderwinkel ohne Zwischenschritte sofort gelegt. Dies ist auf der KeyboardControl zu sehen, wenn sich nach Klicken des Steer Right/Steer Left Buttons der aktuelle Ruderwinkel direkt um eins erhöht bzw. verringert.	ja
Wenn Rudder als Ruder ausgewählt ist, wird der vom Verhalten gewünschte Ruderwinkel mit Zwischenschritten und unter Berücksichtigung der RateOfTurn des Ruders schrittweise gelegt. Dies ist auf der KeyboardControl zu sehen, wenn sich nach klicken des Steer Right/Steer Left Buttons der aktuelle Ruderwinkel nicht sofort um eins erhöht bzw. verringert, sondern zwischenberechnete Ruderwinkel als aktueller Ruderwinkel angezeigt werden.	ja
Wenn *SimpleEngine* als Maschine ausgewählt ist, findet die Berechnung der aktuellen Maschinendrehzahl aufgrund der Maschinenbeschleunigung statt. Dies ist daran zu erkennen, dass selbst die einfachste Dynamik nicht sofort aus dem Stand auf Maximalgeschwindigkeit beschleunigen kann (s.o.).	ja

Literatur

[Bes10] Jörn Beschnidt. *"Virtual Waterway", eine Simulationsumgebung für Verkehrsabläufe auf Binnenwasserstraßen.* PhD thesis, Universität Stuttgart, Holzgartenstr. 16, 70174 Stuttgart, 2010.

[PB02] Tristan Perez and Mogens Blanke. *Mathematical ship modelling for control applications.* Ørsted-DTU, Automation, 2002.

[Pow11] Julia Powell. The New Electronic Chart Product Specification S-101: An Overview. *International Recent Issues about ECDIS, e-Navigation and Safety at Sea: Marine Navigation and Safety of Sea Transportation*, page 69, 2011.