

Institut für Mathematik und Informatik

Fernuniversität Hagen

Vergleichende Implementierung und Evaluierung einer Echtzeitvisualisierung von geographischen Schiffsbewegungsdaten in HTML5

Bachelorarbeit

Barbara Drücke

Matrikel-Nummer 7397860

Abgabedatum: 23.05.2013

Betreuer	Dr. Jörg Brunsmann
Erstprüfer	Prof. Dr. Hemmje
Zweitprüfer	Dr. Jörg Brunsmann

Inhaltsverzeichnis

Abbildungsverzeichnis	IV
Tabellenverzeichnis	V
Listings	VI
1 Einleitung	1
1.1 Motivation für diese Arbeit	2
1.2 Aufbau der Arbeit	3
2 Real-Time-Schiffsverfolgung per AIS-Daten-Strom	4
2.1 Anwendungsfälle	4
2.2 Beschreibung der Anforderungen	5
2.2.1 Funktionale Anforderungen	5
2.2.2 Nicht funktionale Anforderungen	5
2.3 Grobentwurf der Anwendung	6
3 Grundlagen	7
3.1 Automatisches Informationssystem	7
3.2 OpenStreetMap	9
3.3 Leaflet	9
3.4 Bidirektionale Kommunikation über HTML5 Websockets	9
3.5 Node.js	10
3.6 Google Dart	11
4 Implementierungen	13
4.1 Strategie bei der Vorgehensweise	13
4.2 Notwendige Strategie-Korrekturen	13
4.3 Das Problem der Vergleichbarkeit	14
4.4 Beschreibung der ausgeführten Implementierungen	15
4.4.1 socket.io-Server	15
4.4.2 socket.io-Client	19
4.4.3 HTML5-Server	22
4.4.4 HTML5-Client in Javascript	22
4.4.5 HTML5-Client in Google Dart	23
5 Ergebnisse	27
5.1 Evaluation der Anwendung	27
5.1.1 Funktionale Anforderungen	27

5.1.2	Nicht funktionale Anforderungen	28
5.2	Vergleichende Evaluation der Javascript- und der Dart-Anwendung	29
5.2.1	Vergleich des socket.io-Servers mit dem HTML5-Server	30
5.2.2	Vergleich des Javascript-Clients mit dem Dart-Client	33
6	Fazit und Ausblick	38
6.1	Die Real-Time-Anwendung	38
6.2	Vergleich Javascript und Google Dart	38

Abbildungsverzeichnis

1.1	Architektur der Web-Anwendung der Vesseltracker.com-GmbH	1
1.2	Ansicht der Elbe hinter Hamburg in der 'Cockpit'-Anwendung	2
2.1	Architektur-Entwurf der Real-Time Web-Anwendung	6
4.1	Übersicht über ausgeführte Server-und Clientimplementierungen	14
4.2	Übersicht über die implementierten Javascript-Dateien und -Objekte	15
4.3	Übersicht über implementierte Dart-Dateien und -Klassen	23
5.1	Anzeige aller Schiffe im Hamburger Hafen	27
5.2	Auf schnell fahrende Schiffe reduzierte Anzeige am Beispiel der Nordsee . .	28
5.3	Schleppmanöver	28
5.4	Anwortzeiten des socket.io-Websocket-Servers in Abhängigkeit von der Anzahl verbundener Clients	29
5.5	socket.io-Websocket-Server: Latenzzeit der Positionsmeldungen und Anzahl empfangener Schiffe	31
5.6	HTML5-Websocket-Server: Latenzzeit der Positionsmeldungen und Anzahl empfangener Schiffe	31
5.7	Verarbeitungsdauer in Dartium (HP 530 Notebook PC)	35
5.8	Verarbeitungsdauer in Chrome (HP 530 Notebook PC)	36
5.9	Verarbeitungsdauer in Firefox (HP 530 Notebook PC)	36
5.10	Verarbeitungsdauer in Dartium (Apple MacBook Pro)	36
5.11	Verarbeitungsdauer in Chrome (Apple MacBook Pro)	37
5.12	Verarbeitungsdauer in Firefox (Apple MacBook Pro)	37

Tabellenverzeichnis

3.1	Intervalle, in denen Schiffe AIS-Nachrichten aussenden	8
5.1	Unterstützung von Browsern durch die socket.io- und HTML5-Websocket-Clients	32
5.2	Anzahl Zeilen Code im Vergleich der Clients	33

Listings

4.1	Generierung von Kindprozessen in master.js	16
4.2	Schreiben in die mongoDB in aisData-client.js	16
4.3	Aufbau des Geo-Indexes in aisData-client.js	17
4.4	Vessel-in-Bounds-query in worker.js	17
4.5	Speichern der übermittelten Client-Daten in worker.js	18
4.6	Weiterleitung von Positions-Updates an Websocket-Clients in worker.js	18
4.7	‘Revealing Module Pattern’ in LeafletMap.js	19
4.8	Client-Seite der socket.io-Websocket-Verbindung in ais-socket.io.js	20
4.9	Aufruf der public function paintToMap des Vessel-Objekts in ais-socket.io.js	20
4.10	vom socket.io-Server gesendete Nachricht	22
4.11	vom HTML5-Websocket-Server gesendete Nachricht	22
4.12	Konstruktor des LeafletMap-Objektes mit Zugriff auf den Javascript-Kontext	24
4.13	Konstruktor des Dart-Objekts Polyline	25
4.14	EventHandling mithilfe von Callback-Funktionen	26
4.15	Zugriff auf das L.Map-Objekt (im Javascript-Kontext) über das LeafletMap-Objekt (im Dart-Kontext) in LeafletMap.dart	26
5.1	socket.io Client-Request in Internet Explorer 9	32
5.2	LeafletMap.dart mit ursprünglicher Abfrage des Bounds-Objektes	33
5.3	LeafletMap.dart mit workaround über BBoxString	34

1 Einleitung

Die Vesseltracker.com GmbH ist ein Schiffsmonitoring und -reporting-Dienstleister. Der kostenpflichtige Dienst stellt den Kunden umfangreiche Informationen zu Schiffen weltweit zur Verfügung. Dabei handelt es sich einerseits um Schiffs-Stammdaten und andererseits um Schiffs-Positionsdaten. Die Positionsdaten sind AIS (Automatic Identification System) -Daten, wie sie von allen Schiffen über Funk regelmäßig zu senden sind.

Vesseltracker.com unterhält ein Netzwerk von ca. 800 terrestrischen AIS-Antennen, mit denen küstennahe AIS-Meldungen empfangen und via Internet an einen zentralen Rohdatenserver geschickt werden. Der Rohdatenserver verarbeitet die Meldungen und gibt sie umgewandelt und gefiltert an die Anwendungen des Unternehmens weiter. Zusätzlich erhält das Unternehmen AIS-Daten via Satellit über einen Kooperationspartner. Damit werden die küstenfernen Meeresgebiete und Gegenden, in denen Vesseltracker.com keine AIS-Antenne betreibt, abgedeckt. Die Kernanwendung des

Unternehmens ist eine Webanwendung, die die terrestrischen AIS-Daten in einer Geo-Datenbank speichert und sie mit den Schiffs-Stammdaten und Satelliten-AIS-Daten in Beziehung setzt.

Für eine geographische Visualisierung der Schiffspositionen existiert das sogenannte 'Cockpit', wo die Schiffe als Icons auf Openstreetmap-Karten dargestellt werden. Diese Karte zeigt jeweils alle Schiffe an, die sich in dem frei wählbaren Kartenausschnitt zu der Zeit befinden. Aktualisiert werden die Positionsinformationen jeweils bei Änderung des betrachteten Bereichs oder einmal pro Minute. Detailinformationen erhält der Nutzer durch ein Click-Popup über das Icon des Schiffes. Darüber kann er sich auch die gefahrene Route der letzten 24 Stunden anzeigen lassen.

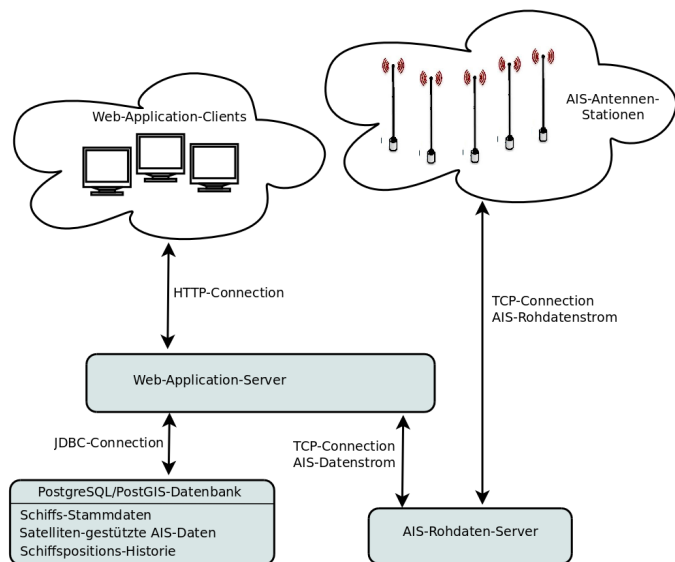


Abbildung 1.1: Architektur der Web-Anwendung der Vesseltracker.com-GmbH

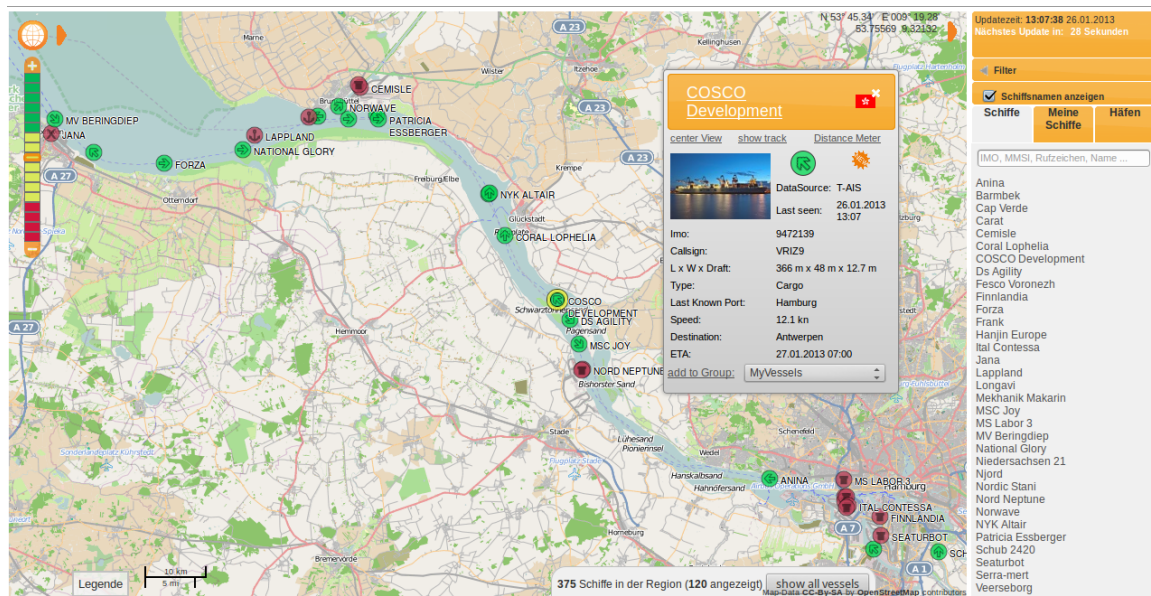


Abbildung 1.2: Ansicht der Elbe hinter Hamburg in der 'Cockpit'-Anwendung

1.1 Motivation für diese Arbeit

Aus mehreren Gründen entstand der Plan, dem Portfolio des Unternehmens neben der existierenden Cockpit-Anwendung eine Real-Time-Darstellung der geographischen Schiffspositionen hinzuzufügen.

- Aufgrund der herausragenden Qualität des vesseltracker.com Antennen-Netzwerks sind die verfügbaren terrestrischen AIS-Daten höchst aktuell, aktualisieren sich kontinuierlich und erreichen eine hohe weltweite Abdeckung des Schiffsverkehrs. Damit ist es möglich, die Situation des Schiffsverkehrs in beliebigen Häfen, Wasserstraßen und Küstengebiete weltweit und sekundengenau zu präsentieren. Diese Genauigkeit wird in der Cockpit-Anwendung nicht vollständig genutzt.
- Real-Time-Anwendungen gewinnen laufend an Bedeutung. Ihre Verbreitung wird durch den Fortschritt der verfügbaren Webtechnologien auf breiter Basis unterstützt. Mitbewerber auf dem Markt für AIS-Daten (z.B. Fleetmon.com) bieten bereits Echtzeit-Darstellungen ihrer AIS-Daten an. Um in diesem Geschäftsfeld weiterhin eine Spitzenposition beanspruchen zu können, sollte auch Vesseltracker eine Real-Time-Anwendung zur Schiffsverfolgung für Desktop-Computer zur Verfügung stellen und in einem nächsten Schritt auch für Mobile Devices.
- Ein Phänomen in der menschlichen Wahrnehmung liefert ein weiteres Argument, die Cockpit-Anwendung durch eine Real-Time-Anwendung zu ergänzen oder sogar abzulösen: Aufgrund der sogenannten Veränderungsblindheit oder "Change Blindness" werden Veränderungen an einem Objekt (in diesem Fall die Position eines Schiffs-Icons auf der Karte) in der Wahrnehmung überdeckt, wenn im selben Augenblick Veränderungen an der Gesamtsicht vonstatten gehen [Dor04]. Genau dies geschieht im Cockpit, wenn nach dem Laden neuer Positionsdaten alle Schiffs-Icons neu gerendert und Namens-Fähnchen gelöscht oder hinzugefügt werden. Dieses kurze "Flackern"

macht es dem Betrachter fast unmöglich, die Positionsänderung eines Schiffes auf der Karte mit dem Auge zu verfolgen.

1.2 Aufbau der Arbeit

Im Kapitel 2 werden mögliche Anwendungs-Szenarien genauer beleuchtet und die funktionalen und nicht funktionalen Anforderungen an die geplante Anwendung herausgestellt. Anschließend wird die Systemarchitektur der geplanten Anwendung grob entworfen. In Kapitel 3 wird kurz auf die technischen Grundlagen eingegangen: die AIS-Technologie, die bei der Gewinnung des Datenmaterials verwendet wird und damit für Art und Format der Daten verantwortlich ist; das Javascript-Framework node.js sowie Google Dart, die bei der Programmierung der Anwendung zum Einsatz kommen; HTML5-Websockets, weil sie für die Verteilung der Daten eingesetzt werden; das OpenStreetMap-Projekt als Lieferant des Kartenmaterials wird kurz vorgestellt, sowie die Leaflet-Bibliotheken, mit deren Hilfe die Schiffsobjekte auf die Karte gerendert werden.

In Kapitel 4 werden zunächst die Gründe für die spezifische Auswahl an Implementierungen dargelegt. Anschließend wird die Vorgehensweise bei der Implementierung erläutert und zwar zunächst ausführlich für die jeweils erste Server- bzw. Client-Implementierung (socket.io-Server 4.4.1 und socket.io-Client 4.4.2). Anschließend werden für die alternativen Implementierungen nur jeweils die Unterschiede herausgestellt. Die fertigen Programme werden in Kapitel 5 getestet und nach verschiedenen Aspekten verglichen. Kapitel 6 fasst die Ergebnisse zusammen und gibt einen Ausblick auf mögliche Weiterentwicklungen.

2 Real-Time-Schiffsverfolgung per AIS-Daten-Strom

2.1 Anwendungsfälle

Hafendienstleister wie Schlepper, Lotsen oder Festmacher verschaffen sich über einen Monitor einen Überblick über die Arbeitsvorgänge in ihrem jeweiligen Heimathafen, z.B. welche Schlepper schleppen welches Schiff, wo gehen Lotsen an oder von Bord, welche Tanker betanken welche Schiffe, usw. Sie kontrollieren die Ausführung der eigenen Aufträge oder auch die der Mitbewerber. Die Anwendung läuft hierbei eigenständig, das heißt, Zoomstufe und Kartenausschnitt ändern sich nicht oder nur selten. Es ist also notwendig, dass die Anwendung unabhängig von Benutzer-Interaktionen immer die aktuellsten verfügbaren Daten anzeigt.

Ein verwandter Anwendungsfall betrifft Nutzer, für die die Beobachtung, bzw. Überwachung bestimmter Wasserverkehrswege oder Häfen von besonderem Interesse ist. Dies trifft auf Sicherheitsorgane (z.B. die Wasserschutzpolizei), Schiffsfotografen und Nutzer der Passagierschiffahrt zu.

Reedereien beobachten das Einlaufen, Anlegen, Festmachen oder Ablegen und Auslaufen ihrer Schiffe in entfernten Häfen, wo es keine Unternehmensniederlassung gibt. Zum Beispiel kontrollieren sie, wann und an welchen Liegeplätzen ein Schiff wie lange festmacht. Dazu ist es zum einen notwendig, auf eine geringe Zoomstufe heraus- und auf einen anderen Hafen wieder hineinzoomen zu können. Zum anderen soll die Anwendung Schnittstellen bieten, um zusätzliche Informationen aus dem vesseltracker.com Datenpool (z.B. Liegeplatzinformationen) anzufordern.

Die vesseltracker.com GmbH nutzt die Real-Time-Anwendung, um die vom Unternehmen angebotenen Daten zu präsentieren und zu bewerben. Dabei ist es wichtig, dass die Anwendung gesendete AIS-Signale im Schnitt in weniger als einer Sekunde auf dem Monitor als Position oder Positionsänderung darstellen kann und dass die Schiffsbewegungen fließend ohne "Flackern" dargestellt werden. Damit kann vesseltracker.com die größere Genauigkeit und Aktualität der eigenen Daten gegenüber denen anderer Anbieter herausstellen.

Die Anwendungsfälle verdeutlichen noch einmal, dass der zusätzliche Nutzen der Real-Time-Anwendung gegenüber der Cockpit-Anwendung nicht ausschließlich in der höheren Aktualität liegt, denn die Daten im Cockpit sind ja ebenfalls im Minutenbereich aktuell. Ein wichtiger Vorteil liegt vielmehr in der Lebendigkeit der Darstellung. Bewegte Objekte binden stärker die Aufmerksamkeit des Betrachters. Sie sind ohne Anstrengung mit dem Auge zu verfolgen und geben der Anwendung einen gewissen Unterhaltungswert.

2.2 Beschreibung der Anforderungen

2.2.1 Funktionale Anforderungen

- als Datenquelle sollen ausschließlich die vom Rohdatenserver als JSON-Datenstrom zur Verfügung gestellten AIS-Informationen dienen
- Schiffe sollen an ihrer aktuellen (Real-Time-) Position auf einer Karte im Browser dargestellt werden
- Positionsänderungen einzelner Schiffe sollen ad hoc sichtbar gemacht werden
- die Schiffsbewegungen auf der Karten sollen nicht sprunghaft, sondern fließend erscheinen (Animation der Schiffsbewegungen in dem Zeitraum zwischen zwei Positionsmeldungen)
- die Karte soll in 16 Zoomstufen die Maßstäbe von 1:2000 bis 1: 200 Mio abdecken
- Schiffe sollen auf der Karte als Symbole dargestellt werden, die den Navigationsstatus und gegebenenfalls den Kurs widerspiegeln
- bei hoher Auflösung / großem Zoom-Level und ausreichend statischen AIS-Informationen soll ein Schiff als maßstabsgetreues Polygon (genauer: Fünfeck) in die Karte gezeichnet werden.
- bei geringer Auflösung / niedrigem Zoom-Level soll ein Eindruck über die Verteilung der empfangenen Schiffe vermittelt werden, ohne jedoch jedes Schiff tatsächlich darzustellen
- Detail-Informationen zu jedem Schiff sollen als Popups über das Symbol auf der Karte abrufbar sein

2.2.2 Nicht funktionale Anforderungen

- die von den Antennen empfangenen AIS-Daten sind mit minimaler Verzögerung (unter 500 msec) auf der Karte darzustellen
- die Anwendung soll ca. 300 gleichzeitige Client-Verbindungen erlauben und skalierbar sein
- als Clients der Anwendung sollen die gängigen Browser in den am meisten verbreiteten Versionen unterstützt werden (Microsoft Internet Explorer, Google Chrome, Mozilla Firefox, Apple Safari)
- der Programm-Code wird auf Github als private repository gehostet
- verwendete Software-Module sollen frei zugänglich (open source) sein
- als Kartenmaterial sind die von vesseltracker gehosteten OpenstreetMap-Karten zu verwenden

- ein Prototyp der Anwendung soll zeitnah zur Verfügung stehen, um Mitarbeitern und Kunden zu ermöglichen, ihre Anforderungen genauer zu spezifizieren oder weitere Anforderungen zu formulieren.

2.3 Grobentwurf der Anwendung

Die eingehende Schnittstelle der zu erstellenden Anwendung ist die Verbindung zum Rohdatenserver, die als TCP-Verbindung ausgeführt ist und einen JSON-Datenstrom liefert. Die ausgehende Schnittstelle ist der HTTP-Client (Browser). Zu erstellen ist also eine Client-Server-Anwendung, in der der Server zweierlei zu leisten hat, nämlich

1. eine TCP-Socket-Verbindung zum Rohdatenserver zur Abfrage des AIS-Daten-Stroms zu unterhalten und
2. eine bidirektionale Verbindungen zu vielen HTTP-Clients herzustellen, in der die Clients Änderungen des betrachteten Kartenausschnittes an den Server senden können und der Server jederzeit den Client über relevante, aus dem JSON-Datenstrom ausgelesene, Schiffsbewegungen im betrachteten Kartenausschnitt informieren kann.

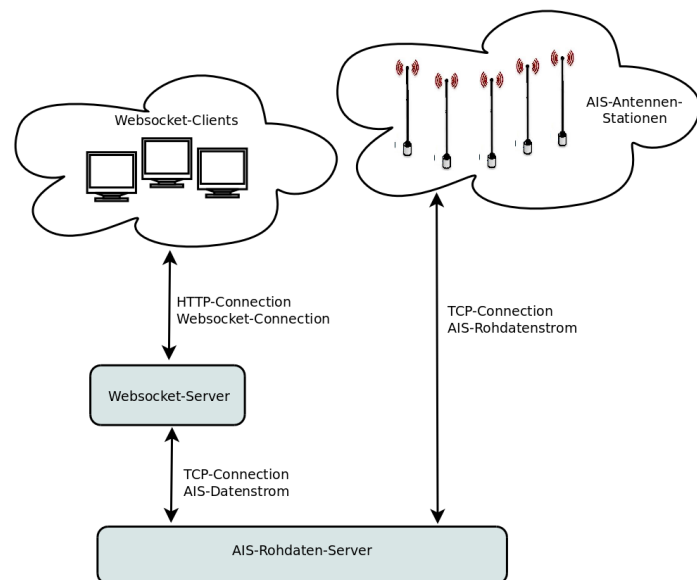


Abbildung 2.1: Architektur-Entwurf der Real-Time Web-Anwendung

3 Grundlagen

3.1 Automatisches Informationssystem

Das Automatic Identification System (AIS) ist ein UKW-Funksystem im Schiffsverkehr, das seit 2004 für alle Berufsschiffe über 300 BRZ (Bruttoreaumzahl) in internationaler Fahrt und seit 2008 auch für solche über 500 BRZ in nationaler Fahrt verpflichtend eingeführt worden ist. Es soll dabei helfen, Kollisionen zwischen Schiffen zu verhüten und die landseitige Überwachung und Lenkung des Schiffsverkehrs zu erleichtern. Außerdem verbessert AIS die Planung an Bord, weil nicht nur Position, Kurs und Geschwindigkeit der umgebenen Schiffe übertragen werden, sondern auch Schiffsdaten (Schiffsname, MMSI-Nummer, Funkrufzeichen, etc.). AIS ist mit UKW-Signalen unabhängig von optischer Sicht und Radarwellenausbreitung [Wik13].

Für die Nutzung von AIS ist ein aktives, technisch funktionsfähiges Gerät an Bord Voraussetzung, das sowohl Daten empfängt als auch Daten sendet. Für Schiffe der Berufsschiffahrt sind Klasse-A-Transceiver an Bord vorgesehen, für nicht ausrüstungspflichtige Schiffe genügen Klasse-B-Transceiver, die mit niedriger VHF-Signalstärke und weniger häufig senden. Die dynamischen Schiffsdaten (s.u.) erhält der AIS-Transceiver vom integrierten GPS-Empfänger, bei Klasse A auch von der Navigationsanlage des Schiffes. Die Vorausrichtung (Heading) kann über eine NMEA-183-Schnittstelle vom Kompass eingespeist werden.

Die AIS-Einheit sendet schiffsspezifische Daten, die von jedem AIS-Empfangsgerät in Reichweite empfangen und ausgewertet werden können:

Statische Schiffsdaten

- IMO-Nummer¹
- Schiffsname
- Rufzeichen²
- MMSI-Nummer³
- Schiffstyp (Frachter, Tanker, Schlepper, Passagierschiff, SAR, Sportboot u. a.)
- Abmessungen des Schiffes (Abstand der GPS-Antenne von Bug, Heck, Backbord- und Steuerbordseite)

1 <http://de.wikipedia.org/wiki/Schiffsnummer#IMO-Nummer>

2 <http://de.wikipedia.org/wiki/Schiffsnummer#Unterscheidungssignal>

3 <http://de.wikipedia.org/wiki/Schiffsnummer#MMSI>

Dynamische Schiffsdaten

- Navigationsstatus (unter Maschine, unter Segeln, vor Anker, festgemacht, manövrierunfähig u. a.)
- Schiffsposition (LAT, LON = Latitude, Longitude in WGS 84)
- Zeit der Schiffsposition (nur Sekunden)
- Kurs über Grund (COG = Course over Ground)
- Geschwindigkeit über Grund (SOG = Speed over Ground)
- Vorausrichtung (HDG = Heading)
- Kursänderungsrate (ROT = Rate of Turn)

Reisedaten

- aktueller maximaler statischer Tiefgang in dm
- Gefahrgutklasse der Ladung
- Reiseziel (UN/LOCODE)⁴
- geschätzte Ankunftszeit (estimated Time of Arrival = ETA)
- Personen an Bord

Der Navigationsstatus und die Reisedaten müssen vom Wachoffizier manuell aktualisiert werden. Gesendet werden die AIS-Signale auf zwei UKW-Seefunkkanälen (Frequenzen 161,975 MHz und 162,025 MHz), wobei die Sendeintervalle abhängig sind von der Klasse, dem Manöverstatus und der Geschwindigkeit⁵.

Klasse	Manöver-Status	Geschwindigkeit	Sendeintervall
Class A	geankert/festgemacht	<3kn	3 min
Class A	geankert/festgemacht	>3kn	10 sec
Class A	in Fahrt	0-14kn	10 sec
Class A	in Fahrt, Kursänderung	0-14	3 1/3 sec
Class A	in Fahrt	14-23kn	6 sec
Class A	in Fahrt, Kursänderung	14-23	2 sec
Class A	in Fahrt	>23kn	2 sec
Class B		<2 kn	3 min
Class B		>2 kn	30 sec

Tabelle 3.1: Intervalle, in denen Schiffe AIS-Nachrichten aussenden

Für AIS-Daten sind 22 standardisierte Nachrichtentypen bzw. Telegramme festgelegt. Für diese Arbeit interessieren nur die regulären Positionsmeldungen (dynamische Schiffsdaten) der Klasse-A-Transceiver (Typ 1-, 2- und 3-Messages) und die regulären Meldungen von (statischen) Schiffs- und Reisedaten der Klasse-A-Transceiver (Typ 5-Messages).

⁴ <http://www.unece.org/cefact/locode/service/location.html>

⁵ http://de.wikipedia.org/wiki/Automatic_Identification_System#Senden_der_AIS-Daten

3.2 OpenStreetMap

OpenStreetMap⁶ ist eine freie, editierbare Karte der gesamten Welt auf der Basis von Daten, die von einer breiten Nutzergemeinde zusammengetragen werden. Inzwischen kann die Qualität der Karten mit denen proprietärer Angebote mithalten und übertrifft sie sogar in manchen Bereichen. Die Daten können gemäß der entsprechenden Freien Lizenz frei heruntergeladen und genutzt werden unter der Bedingung, dass sie nur unter der "Creative Commons Attribution-Share-Alike" (CC-BY-SA) -Lizenz weitergegeben werden.

3.3 Leaflet

Leaflet ist eine open source JavaScript-Bibliothek⁷ für interaktive Karten, die von einer Gruppe um Vladimir Agafonkin geschrieben wurde. Die Bibliothek zeichnet sich durch ein klares, schlankes Design aus und überzeugt durch gute Performance. Leaflet unterstützt alle Plattformen auch im mobilen Bereich mithilfe von HTML5 und CSS3. Es ist hinreichend dokumentiert und verfügt über gut lesbaren Quellcode.

3.4 Bidirektionale Kommunikation über HTML5 Websockets

In der Entwicklung der Kommunikationstechnologien im Internet galt lange Zeit das request/response Paradigma, nach dem Anfragen eines Clients von einem Server beantwortet werden. Dieses Paradigma wird Stück für Stück aufgebrochen durch kontinuierliche Weiterentwicklungen in Richtung einer bidirektionalen Kommunikation zwischen Server und Client.

Schon seit HTTP Long Polling, HTTP Streaming und Ajax on demand ist es für Serveranwendungen möglich, nach einem initialen Verbindungsaufbau durch den Client, beim serverseitigen Eintreffen neuer Daten scheinbar selbständig einen Datenaustausch zum Client zu initiieren. Dabei handelt es sich eigentlich nur um die aufgeschobene Beantwortung einer zuvor gestellten Client-Anfrage.

Der Nachteil dieser Technologien liegt darin, dass sie, weil sie Nachrichten über das HTTP-Protokoll austauschen, einen großen Überhang an Header-Informationen mitzusenden gezwungen sind, der sich in Summe negativ auf die Latenzzeit auswirkt [Var11]. Damit sind diese Technologien für zeitkritische (Real-Time-) Anwendungen nicht unbedingt geeignet. Das 2011 eingeführte WebSocket-Protokoll dagegen spezifiziert eine API (HTML5-WebSocket API-Spezifikation⁸), die eine echte bidirektionale Socket-Verbindung zwischen Server und Client ermöglicht, in der beide Seiten jederzeit Daten schicken können. Dieser Socket wird im Anschluss an einen initialen HTTP-handshake aufgebaut, indem Server und Client einen Upgrade der Verbindung auf das WebSocket-Protokoll aushandeln [MU12].

6 <http://www.openstreetmap.org/>

7 <http://leafletjs.com/reference.html>

8 <http://www.w3.org/TR/2011/WD-websockets-20110419/>

3.5 Node.js

Node.js⁹ ist ein Framework zur Entwicklung serverseitiger Webanwendungen in Javascript. Es wurde 2009 von Ryan Dahl veröffentlicht und hat seitdem viel Aufmerksamkeit erregt, weil Anwendungen in node.js

- hoch performant
- skalierbar
- und echtzeitfähig sind.

Diese Eigenschaften sind größtenteils dem Konzept des asynchronen, nicht blockierenden I/O von Javascript im Allgemeinen und node.js im Besonderen geschuldet. Javascript ist von Anfang an asynchron konzipiert für die Verwendung im Webbrowser, wo synchrone Verarbeitung wegen der Verzögerung der Seitendarstellung nicht in Frage kommt. Den gleichen Ansatz übernimmt node.js für die Serverseite.

Node.js arbeitet single-threaded und eventbasiert. Die zentrale Kontrollstruktur, die den Programmablauf steuert, ist der Event-Loop. Er empfängt Events, die von Programm- oder Nutzeraktionen ausgelöst werden und setzt sie in Callback-Funktionen um. Kommt es im Programmablauf zur Interaktion mit einer externen Ressource, wird diese Interaktion in einen neuen Prozess ausgelagert und mit einer Callback-Methode versehen. Anschließend kann der Event Loop weitere aufgelaufene Events verarbeiten. Ist die Interaktion abgeschlossen bekommt der Event Loop ein Signal und setzt beizeiten die Verarbeitung mit der entsprechenden Callback-Methode fort [Tei12].

Node.js bringt als Laufzeitumgebung die Google open source V8-Javascript-Engine mit, die die Ausführung von Javascript-Code durch Just-In-Time-Kompilierung optimiert. Außerdem bietet node.js eine direkte Unterstützung für das HTML5-Websocket-Protokoll. Mit der Unterstützung des JSON-Datenformats sind alle notwendigen Bausteine zusammen für skalierbare, echtzeitfähige Serveranwendungen. Außerdem lassen sich mit dem node.js Package Manager npm jederzeit weitere Pakete aus dem wachsenden Angebot nachinstallieren und verwalten. Als konkrete Pakete für Websockets standen innerhalb von node.js zum Zeitpunkt der Implementierung (November 2012) die Bibliotheken websocket¹⁰ und socket.io¹¹ zur Verfügung. Die Bibliothek websocket genügt der HTML5-Websocket-API-Spezifikation (s.o). Socket.io erweitert die Funktionalität des websockets um heartbeats, timeouts and disconnection support. Außerdem kapselt socket.io die Details des Nachrichtenaustauschs: Bei Browsern, die Websockets nicht unterstützen, handelt socket.io die bestmögliche Verbindungsalternative aus in der Reihenfolge: -> WebSocket -> Adobe® Flash® Socket -> AJAX long polling -> AJAX multipart streaming -> Forever Iframe -> JSONP Polling¹². Um socket.io zu nutzen können, muss im Browser-Client die Datei socket.io.js geladen werden.

⁹ <http://www.nodejs.org/>

¹⁰ <https://github.com/Worlize/WebSocket-Node>

¹¹ <http://socket.io>

¹² <http://socket.io/#browser-support>

3.6 Google Dart

Motivation für Dart

Dart ist eine von der Firma Google als open source Projekt seit ca. 2 Jahren explizit für Webanwendungen entwickelte Programmiersprache. Das Ziel ist es, eine Sprache zu entwickeln, die komplexe Webanwendungen besser unterstützt als Javascript mit seinen historisch bedingten Mängeln und Schwächen. Das Entwicklerteam definiert die Design-Ziele folgendermaßen¹³: Dart soll

- eine sowohl strukturierte als auch flexible Web-Programmiersprache sein
- sich für Programmierer vertraut anfühlen und intuitiv erlernbar sein
- mit seinen Sprachkonstrukten performant sein und schnell zur Ausführung kommen
- auf allen Webdevices wie Mobiles, Tablets, Laptops und Servern gleichermaßen lauffähig sein und alle gängigen Browser unterstützen.

Spracheigenschaften von Dart

- Dart arbeitet ereignisbasiert und asynchron und in einem einzigen Thread ganz nach dem Vorbild von node.js.
- In Browsern mit der Dart-Virtual-machine (z.Zt. nur Google Dartium) kann nativer Dart-Code ausgeführt werden. In allen anderen Browsern wird der Dart-Code zu Javascript kompiliert. Dazu muss im Browser-Client die Datei dart.js aus dem Dart-Package 'browser' geladen werden.
- Dart verwendet Klassen als wohlbekanntes Sprachkonzept zur Kapselung und Wiederverwendung von Methoden und Daten. Jede Klassen definiert implizit ein Interface.
- Die Typisierung in Dart ist optional, das heißt sie führt nicht zu Laufzeitfehlern. Sie ist als Werkzeug für den Entwickler gedacht, zur besseren Verständlichkeit des Codes, besonders bei umfangreichen Anwendungen und als Hilfe beim Debuggen.
- Die Gültigkeitsbereiche von Variablen in Dart gehorchen einfachen, intuitiv nachvollziehbaren Regeln: Variablen sind gültig in dem Block (...), in dem sie definiert sind.
- Zur Parallelverarbeitung nutzt Dart das Konzept von Isolates (übernommen von ER-LANG), eine Art Lightweight Processes. Isolates greifen nicht auf einen gemeinsamen Speicherbereich zu und teilen nicht denselben Prozessor-Thread. Isolates kommunizieren miteinander ausschließlich über Nachrichten (über SendPort und ReceivePort). Sie werden gesteuert von einem übergeordneten Event Loop.
- Google stellt Entwicklern den DartEditor zur Verfügung, eine Entwicklungsumgebung für die Entwicklung von Dart Web- und Serverapplikationen. Sie beinhaltet das Dart SDK und den Dartium Browser mit der Dart VM.

¹³ <http://www.dartlang.org/docs/technical-overview/#goals>

- Der dart2js Compiler ist ebenfalls im DartEditor enthalten und kompiliert Dart-Code zu Javascript-Code, der für die Chrome V8-Javascript-Engine optimiert ist.
- Mit 'Pub' verfügt Dart über einen Package Manager vergleichbar dem node.js Package Manager npm.

[Mü12][Lad12]

Einbindung von Javascript-Bibliotheken in Dart mit js-interop

Für die Verwendung von Javascript-Bibliotheken in Dart-Code existiert die Dart-Bibliothek 'js-interop'¹⁴. Damit können Dart-Anwendungen Javascript-Bibliotheken verwenden und zwar sowohl in nativem Dart-Code, der in der Dart-Virtual-Machine ausgeführt wird als auch in zu Javascript kompiliertem Dart-Code.

Wenn das Dart-Package js in eine Dart-Anwendung eingebunden ist, kann ein sogenannter Proxy zum Javascript-Kontext der Seite erstellt werden. Referenzen an diesen Proxy werden automatisch in den Javascript-Kontext umgeleitet. Auf oberster Ebene lassen sich damit Javascript-Arrays und -Maps generieren, die mit den entsprechenden Objekten in Dart korrespondieren. Über diesen Proxy können aber auch Proxies zu beliebigen Javascript-Objekten erstellt werden, deren Eigenschaften und Methoden im Javascript-Kontext zur Verfügung stehen.

Um Dart-Funktionen aus dem Javascript-Kontext heraus aufzurufen, wird die entsprechende Funktion in ein Callback-Objekt geladen, das entweder ein einziges Mal (js.Callback.once(dart function)) oder mehrmals (js.Callback.many(dart function)) aufrufbar ist. Um die Lebensdauer dieser Proxies und Callback-Objekte zu verwalten, benutzt Dart das Scope-Konzept: Per default haben alle Proxies nur lokale Gültigkeit. Sollen sie den Ausführungszeitraum des Scopes überdauern, können sie ausdrücklich aufbewahrt werden (js.retain(js.Proxy-Object)), müssen dann aber zu Vermeidung von memory leaks auch explizit wieder freigegeben werden (js.release(js.Proxy-Object)). Dasselbe gilt für Callback-Objekte, die mehrmals aufrufbar sind [Men12].

Dart-Websockets

Für serverseitiges Dart, das auf der serverseitigen Dart-VM läuft, existiert das Paket Dart:io. Es ermöglicht Zugriff auf das Dateisystem und auf Prozesse. In Dart:io existiert auch eine Websocket-Implementierung¹⁵, mit der bereits einfache Websocket-Server geschrieben werden können¹⁶.

¹⁴ <https://github.com/dart-lang/js-interop>

¹⁵ http://api.dartlang.org/docs/releases/latest/dart_io/WebSocket.html

¹⁶ <http://www.dartlang.org/docs/dart-up-and-running/contents/ch05.html>

4 Implementierungen

4.1 Strategie bei der Vorgehensweise

Zunächst wird eine Implementierung gewählt, die die besten Chancen hat, alle funktionalen und nicht funktionalen Anforderungen zu erfüllen. Dies ist eine Lösung in Javascript mit dem node.js-Framework und dem socket.io-Websocket (Abschnitt 4.4.1 und 4.4.2). Node.js-Serveranwendungen werden schon länger mit guten Ergebnissen in Netzwerken eingesetzt, besonders für Real-Time-Anwendungen mit vielen gleichzeitig verbundenen Clients. Das socket.io-Paket wird genutzt, weil durch die Kapselung der verschiedenen Transportmechanismen die Bedienung einer maximalen Anzahl verschiedener Browser, auch in älteren Versionen möglich ist, ohne den Implementierungsaufwand unverhältnismäßig zu erhöhen. Die Implementierung dieser Lösung steht im Zeitplan vorn, um der Anforderung von Unternehmensseite nach einer zeitnahen Umsetzung und Auslieferung zu entsprechen. In einem zweiten Schritt wird eine vergleichbare Implementierung in Google Dart ausgeführt. Ein Jahr nach der offiziellen Vorstellung (Oktober 2011) befindet sich Dart inzwischen im Zustand „Technology Preview“, so dass ein Einsatz clientseitig im Testbetrieb möglich sein sollte. Der zweite Beta-Release fand im Dezember 2012 statt. Ein dritter Beta-Release ist angekündigt. Obwohl ein ausschließlicher Einsatz von Dart im Produktivsystem noch nicht möglich ist, soll Dart als mögliche Alternative zu Javascript für komplexe Webanwendungen begutachtet werden. Die Lösung in Dart testet die Fähigkeiten und Möglichkeiten, die Dart im Vergleich zu Javascript hat und bietet, um eine datenintensive Real-Time-Anwendung umzusetzen.

4.2 Notwendige Strategie-Korrekturen

Der ursprüngliche Plan, sowohl einen Server als auch einen Client in Dart zu schreiben, kann so nicht realisiert werden, weil mit dem Dart-Websocket-Server einige der grundlegenden Anforderungen noch nicht umzusetzen sind. Zum einen unterstützt Dart keine JSON-over-TCP-Kommunikation, wie sie für die Abfrage des JSON-Datenstroms vom Rohdatenserver erforderlich sind. Und zum anderen gibt es zum Zeitpunkt der Implementierung noch keinen Redis-Client für Dart. Der publish/subscribe Mechanismus der Redis-Datenbank wird für die Verteilung der Positions-Updates benötigt.

Deshalb wird nur die Client-Anwendung in Dart implementiert (in Abschnitt ??), während als Server der in Javascript Implementierte genutzt werden soll. Das ist allerdings nicht ohne Anpassung möglich, weil der socket.io-Websocket-Server auf dem Client die socket.io.js-Datei erfordert, die in Dart nicht zur Verfügung steht.

Also wird neben dem socket.io-Server ein zweiter Server in Javascript implementiert, der eine WebSocket-Verbindung nach der HTML5-WebSocket-API-Spezifikation aufbaut (in Abschnitt 4.4.3), die in Dart clientseitig mit dem Paket dart:html unterstützt wird. Die Änderung der Server-Anwendung von einem socket.io-Socket-Server zu einem HTML5-WebSocket-Server ist in node.js mit dem websocket-Modul ¹ recht einfach möglich.

4.3 Das Problem der Vergleichbarkeit

An dieser Stelle stellt sich die Frage, ob beide Serverlösungen vergleichbare Ergebnisse liefern. Denn Unterschiede zwischen den node.js-Servern (mit socket.io bzw. HTML5-WebSocket) würden in das Ergebnis des Vergleichs zwischen den Clients in Dart und Javascript mit einfließen. Deshalb wird für den HTML5-kompatiblen WebSocket-Server noch ein Javascript-Client geschrieben. Dieser ist dann direkt vergleichbar mit dem Dart-Client. Zunächst ist zu beurteilen, inwieweit die Javascript-Implementierung durch den Verzicht auf das socket.io-Framework, das in der ersten Lösung verwendet wird, ausgebremst wird. Das geschieht über einen Performance-Vergleich in Abschnitt 5.2.1. Es werden also zwei Vergleiche durchgeführt (siehe auch Abbildung 4.1)

- Innerhalb von Javascript wird der socket.io-WebSocket gegen den HTML5-WebSocket getestet.
- Unter Verwendung des HTML5-Websockets wird der Javascript-Client gegen den Dart-Client getestet

		HTTP-Client	
		Javascript	Google Dart
HTTP-Server	socket.io-WebSocket-Server (4.4.1)	socket.io-Client (4.4.2)	✗
	HTML5-WebSocket-Server (4.4.3)	HTML5-Client (4.4.4)	HTML5-Client (4.4.5)

Abbildung 4.1: Übersicht über ausgeführte Server-und Client-Implementierungen

¹ <https://github.com/Worlize/WebSocket-Node/wiki/Documentation>

4.4 Beschreibung der ausgeführten Implementierungen

In der ersten Implementierung werden Lösungen entwickelt für die in den Anforderungen beschriebenen Aufgaben. In allen weiteren Implementierungen werden diese Lösungen möglichst übernommen und andernfalls eine Alternative entwickelt.

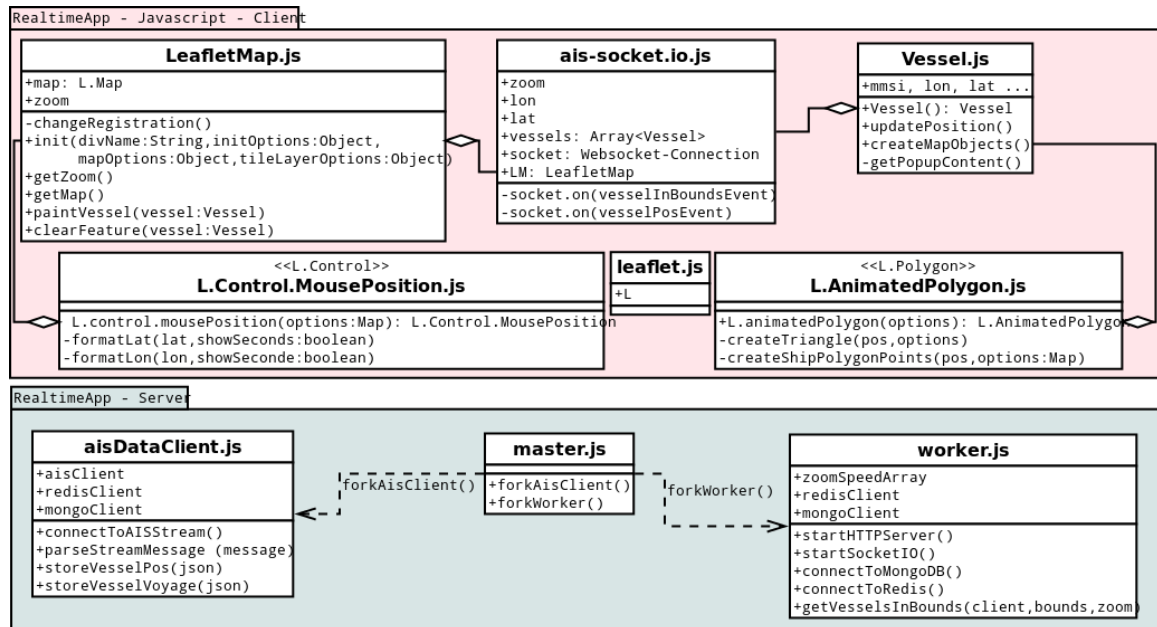


Abbildung 4.2: Schematische Übersicht über die implementierten Javascript-Dateien und -Objekte in der node.js-socket.io-Anwendung. In der Darstellung wird die Notation des UML-Klassendiagramms verwendet, was natürlich (Klassen existieren nicht in Javascript) nicht als solches gelesen werden kann. Weil aber die Anwendung objektorientiert konzipiert ist, war dieser Ansatz sowohl während der Entwicklung als auch bei der Beschreibung der Anwendung sehr nützlich. Die Attribute und Methoden sind nicht vollständig dargestellt, sondern so ausgewählt, dass Zusammenhänge und Abhängigkeiten erkennbar werden. Dasselbe gilt für Abbildung 4.3, die den Dart-Client schematisch darstellt.

4.4.1 socket.io-Server

Die zu entwickelnde Serveranwendung hat grundsätzlich zwei Aufgaben:

1. Daten über eine JSON-over-TCP-Verbindung vom Rohdatenserver abzurufen und
2. einen Websocket zu betreiben, der die Daten an Websocket-Clients weitergibt

Weil node.js singlethreaded ist (vgl. 3.5), würde der Server beide Aufgaben in einem einzigen Prozess bearbeiten. Um das Potential an Parallelverarbeitung eines Dualcore oder Multicore-Servers zu nutzen, ist es daher sinnvoll, für jede Aufgabe einen eigenen Prozess zu generieren. Dazu wurde das node.js-Modul `child_process` genutzt. Die Start-Datei `master.js` generiert damit zuerst einen Prozess, der den AIS-Daten-Client (`aisData-client.js`) abzweigt, um Daten vom Rohdaten-Server abzufragen und anschließend einen worker-Prozess (`worker.js`), um einen Websocket -Server für Client-Verbindungen zur Verfügung zu stellen (siehe Listing 4.1).

```

/* AIS-Client - Process*/
child.fork(path.join(__dirname, 'aisData-client.js'));

/*worker- Process*/
child.fork(path.join(__dirname, 'worker.js'));

```

Listing 4.1: Generierung von Kindprozessen in master.js

Bei der Weitergabe der Daten durch den worker-Prozess sind zwei Fälle zu unterscheiden:

- ein Client verbindet sich neu oder ändert den Kartenausschnitt. In diesem Fall (Vessels-in-Bounds-Request) sind die Schiffs- und Positionsdaten aller sich aktuell in diesem Bereich (Bounds) befindlichen Schiffe an den Client zu senden.
- ein Schiff sendet ein Positions-Update, das an alle Clients verteilt werden muss, deren Kartenausschnitt die betreffende Schiffsposition enthält. Dieses Ereignis wird im Folgenden Vessel-Position-Update genannt.

Vessels-in-Bounds-Request

Der Vessels-in-Bounds-Request macht eine Zwischenspeicherung der Daten unumgänglich. Denn es sollen für jeden Bereich weltweit sofort alle Schiffe zurückgegeben werden können, die in den vergangenen 10 Minuten ihre Position aus diesem Bereich gesendet haben. Wegen der großen Anzahl gleichzeitig empfangener Schiffe (weltweit ca. 60.000) und der Notwendigkeit, einen geographischen Index zu verwenden, wird einer persistenten gegenüber einer transienten Speicherung der Vorzug gegeben.

Für die Persistierung wird MongoDB² verwendet, weil MongoDB als NoSQL-Datenbank mit geringem Overhead schnelle Antwortzeiten und außerdem einen geographischen Index bietet³. Der Serverprozess in `aisData-client.js` schreibt die Daten (siehe Listing 4.2) in eine MongoDB-Collection namens `'vesselsCollection'`. Die MMSI eines Schiffes ist eindeutig und wird als Schlüssel verwendet (siehe Abschnitt 3.1). Über die Option `upsert:true` wird der MongoDB mitgeteilt, dass entweder ein `insert`-Befehl oder, falls die MMSI bereits in der MongoDB-Collection vorhanden ist, ein `update`-Befehl auf den entsprechenden Datensatz auszuführen ist.

```

vesselsCollection.update(
  { mmsi: obj.mmsi },
  { $set: obj },
  { safe: false, upsert: true }
);

```

Listing 4.2: Schreiben in die mongoDB in `aisData-client.js`

In Listing 4.3 ist zu sehen, wie über das Schlüsselwort `"2d"` der MongoDB-Geo-Index auf das Feld `'pos'` mit den Koordinaten des Schiffes gesetzt wird. Damit ist garantiert, dass nicht jede Anfrage des Servers an die Datenbank sämtliche Datensätze durchlaufen muss, um die

² <http://www.mongodb.org/>

³ <http://docs.mongodb.org/manual/core/indexes/#geospatial-indexes>

Schiffe in einem bestimmten geographischen Ausschnitt zu finden. Aufbau und Unterhalt des Geo-Indexes findet im aisData-client-Prozess statt, der schreibend auf die Datenbank zugreift.

```
vesselsCollection.ensureIndex({ pos: "2d", sog: 1, time_received: 1 },  
    function(err, result) {... });
```

Listing 4.3: Aufbau des Geo-Indexes in aisData-client.js

Dabei handelt es sich um einen zusammengesetzten Index, weil neben der Geo-Position auch der Zeitpunkt des Empfangs der Nachricht und die Geschwindigkeit des Schiffes Filterkriterien sind, wenn der zweite Prozess (worker.js) lesend auf die Datenbank zugreift. Zum einen sollen nur Positionen zurückgegeben werden, die nicht älter als 10 Minuten sind und zum anderen werden abhängig vom Zoom-Level nur Schiffe mit einer Mindest-Geschwindigkeit an den Client zurückgeliefert. Darüber wird in geringen Zoom-Levels die Menge angezeigter Schiffe verringert. In Listing 4.4 ist zu sehen, wie der Prozess worker.js mit den vom Client in einem Vessels-in-Bounds-Request übermittelten Geo-Daten (bounds) und Zoom-Level (zoom) die MongoDB anfragt.

```
var vesselCursor = vesselsCollection.find({  
    pos: { $within: { $box: [ [bounds._southWest.lng,bounds._southWest.lat],  
        [bounds._northEast.lng,bounds._northEast.lat] ] } },  
    time_received: { $gt: (new Date() - 10 * 60 * 1000) },  
    sog: { $exists:true },  
    sog: { $gt: zoomSpeedArray[zoom]},  
    sog: {$ne: 102.3}  
});  
vesselCursor.toArray(function(err, vesselData) {  
    client.sendUTF(JSON.stringify({ type: 'vesselsInBoundsEvent', vessels:  
        vesselData}));  
});
```

Listing 4.4: Vessel-in-Bounds-query in worker.js

Vessel-Position-Update

Für die Kommunikation eines Vessel-Position-Updates (AIS-Nachrichtentyp 1-3) zwischen dem aisData-client.js-Prozess und dem worker.js-Prozess wird der publish/subscribe-Mechanismus einer Redis-Datenbank genutzt⁴. Der aisData-client.js-Prozess publiziert jedes Positions-Update auf dem Kanal 'vessel-Pos' der Redis-Datenbank. Der worker.js-Prozess meldet sich als subscriber am Kanal 'vessel-Pos' der Redis-Datenbank an und wird so bei jedem Positions-Update benachrichtigt. Um diese Nachricht an die betroffenen Websocket-Clients weiterzuleiten, ist eine serverseitige Verwaltung der Clients notwendig. Die Serveranwendung muss bei jeder Positionsmeldung wissen, welche Clients benachrichtigt werden müssen. Die Client-Verwaltung ist ein Feature des socket.io-Paketes. Für jeden Client wird bei der Registrierung zusätzlich das Zoomlevel und der Ausschnitt (Bounds) der Karte gespeichert.

⁴ <http://redis.io/topics/pubsub>

```

io.sockets.on('connection', function(client) { ...
  log(' Connection from client accepted. ');
  client.on('register', function(bounds, zoom) {
    client.set('zoom', zoom);
    client.set('bounds', bounds, function() {
      getVesselsInBounds(client, bounds, zoom);
    });
  });
});
client.on('unregister', function() {
  client.del('bounds');
  client.del('zoom');
});
});

```

Listing 4.5: Speichern der übermittelten Client-Daten in worker.js

Bei jedem Vessel-Position-Update, das der worker.js-Prozess empfängt, geht er die Liste der Clients durch und benachrichtigt diejenigen, in deren Bereich das Positions-Update fällt.

```

redisClient.on('message', function(channel, message) {
  if (channel == 'vesselPos') {
    ...
    var json = JSON.parse(message);
    ...
    var clients = io.sockets.clients();
    var lon = json.pos[0];
    var lat = json.pos[1];
    var sog = json.sog/10;
    var cog = json.cog/10;
    clients.forEach(function(client) {
      client.get('bounds', function(err, bounds) {
        if (bounds != null && lon != null && lat != null)
        {
          /* check, if Client-Connection is affected by Vessel-Position-
             Update */
          if (positionInBounds(lon, lat, bounds))
          {
            client.get('zoom', function(err, zoom)
            {
              if(sog !=null && sog > (zoomSpeedArray[zoom]) && sog !=
                102.3)
              {
                client.emit('vesselPosEvent', message);
              }
            });
          }
        }
        ...
      });
    });
  }
});

```

Listing 4.6: Weiterleitung von Positions-Updates an Websocket-Clients in worker.js

4.4.2 socket.io-Client

Der socket.io-Client hat folgende Aufgaben.

1. Es ist eine html-Seite zu erstellen, die die benötigten Source-Dateien lädt und eine HTML-Struktur aufbaut mit den DOM-Elementen für die Karte.
2. URL-Parameter sollen optional übergeben werden können.
3. Optionen sollen zentral an einer Stelle der Anwendung geändert werden können.
4. Eine Karte auf Basis des auf dem unternehmenseigenen Server gehosteten Kartenmaterials mit Navigations- und Infobereichen ist in den Kartenbereich zu rendern.
5. Zum Socket.io-Websocket-Server soll eine Verbindung aufgebaut werden, die
 - 5.1 Vessel-In-Bounds- und Vessel-Position-Events empfängt und
 - 5.2 bei Positionsänderungen auf der Karte eine register-Nachricht sendet
6. Aus den empfangenen JSON-Daten sind geeignete Objekte zu erstellen und zu speichern.
7. Die Objekte sind als Features auf die Karte zu rendern
8. Objekte, die den Status 'Moving' haben, sind entsprechend ihrer Geschwindigkeit zu animieren.

Punkt 1 geschieht in der Datei `ais-socket.io.html`, die zum Start der Anwendung vom Browser-Client aufgerufen wird. Dort werden die benötigten Javascript- und css-Dateien eingebunden, inklusive der JQuery- und der Leaflet-Bibliothek. Nach dem Laden führt der Browser die Anweisungen innerhalb der Funktion `$(document).ready(function() ...` in der Datei `ais-socket.io.js` aus. Falls Url-Parameter übergeben worden sind für initialen Zoomlevel und Kartenausschnitt (Punkt 2) werden sie nun mit der Funktion `getParam(name)` aufgegriffen, sonst wird ein Defaultwert für zoom und bounds benutzt. Wie unter Punkt 3 gefordert, kann dieser Defaultwert und alle weiteren Einstellungen (z.B. Server-IP, Server-Port, Map-Server-Url (Punkt 4)) in dieser Datei zentral angepasst werden.

Als Javascript-Bibliothek zur Darstellung der Schiffe auf der Karte wurde die Leaflet-Bibliothek (siehe Abschnitt 3.3) ausgewählt. Die Cockpit-Anwendung nutzt zwar die OpenLayers-Bibliotheken, diese sind jedoch im Vergleich sehr viel sperriger in der Nutzung und werden inzwischen weniger aktiv von der Community weiterentwickelt. Mithilfe der Leaflet-Bibliothek wird die Karte als Javascript-Objekt in der Datei `LeafletMap.js` realisiert. Dazu wird das 'Revealing Module Pattern' genutzt, mit dem sich die API der Karte von ihrer internen Implementierung trennen lässt. Nur die in der return-Klausel zurückgegebenen Funktionen bilden die öffentliche Schnittstelle des Karten-Objektes (Punkt 4).

```
var LMap = function() {
  var map, featureLayer, tileLayer, zoom, socket, boundsTimeout,
      boundsTimeoutTimer;
  function init(elementid, initOptions, mapOptions, tileLayerOptions) { ... }
  function changeRegistration() { ... }
  function getMap(){ ... }
  function getZoom(){ ... }
```

```
function addToMap(feature, animation, popupContent){ ... }
function removeFeatures(vessel){ ... }
return {
  init: init,
  getMap: getMap,
  getZoom: getZoom,
  addToMap: addToMap,
  removeFeatures: removeFeatures }
}();
```

Listing 4.7: ‘Revealing Module Pattern’ in LeafletMap.js

Nach dem Initialisieren der Karte wird die Websocket-Verbindung (Punkt 5) hergestellt (siehe Listing 4.8). Für den Empfang der Nachrichten des Websocket-Servers (Punkt 5a) genügen dazu zwei Listener:

```
var socket = io.connect('http://' + WEBSOCKET_SERVER_LOCATION + ':' +
  WEBSOCKET_SERVER_PORT);
socket.on('vesselsInBoundsEvent', function (data) {...})
socket.on('vesselPosEvent', function (data) {...})
```

Listing 4.8: Client-Seite der socket.io-Websocket-Verbindung in ais-socket.io.js

Um eine Liste aller im Kartenbereich befindlichen Schiffe vom Server zu bekommen, muss der Client eine ‘register’-Nachricht mit den aktuellen Bounds an den Server senden (Punkt 5b). Dies geschieht einmal nach dem Initialisieren der Karte und soll anschließend durch den von der Leaflet-Map nach jeder Änderung des Kartenausschnitts getriggerten moveend-Event ausgelöst werden, bzw. spätestens nach der über BOUNDS_TIMEOUT konfigurierten Zeitspanne. Weil dieser Event innerhalb des LeafletMap-Objektes auftritt, wird dem LeafletMap-Objekt bei der Initialisierung eine Referenz auf die Websocket-Connection übergeben. Diese Lösung ist unproblematisch, weil beim Verlust der socket-Verbindung ohnehin ein Reload der Seite stattfindet.

Um geeignete Objekte aus den Ais-Messages zu erstellen (Punkt 6), wird in der Datei Vessel.js eine Konstruktor-Funktion zur Verfügung gestellt, mit der Instanzen von Vessel-Objekten generiert werden können. Diese Instanzen werden in einem assoziativen Array namens ‘vessels’ unter ihrem Attribut MMSI als Schlüsselwert gespeichert. Beim Empfang eines Vessel-Position-Events kann mit vessels[mmsi] nach dem passenden vessel-Objekt zum Update gesucht werden.

Schließlich sind die Vessel-Objekte auf die Karte zu rendern (Punkt 7). Dazu wird in Vessel.js eine asynchrone Funktion genutzt (siehe Listing 4.9), mit der zuerst je nach Status (moving / not moving) und Zoomlevel unterschiedliche Features (Polygon, Triangle, Speedvector, Circle) für ein Schiff erstellt und auf die Karte gerendert werden. Anschließend wird das Vessel-Objekt mit seinen Features in ‘vessels’ gespeichert. Das Speichern ist notwendig, um die Features bei Vessel-Position-Updates von der Karte zu entfernen, bevor sie an eine neue Position gerendert werden.

```
vessel.paintToMap(LMap.getZoom(), function(){
  vessels[vessel.mmsi] = vessel;
});
```

Listing 4.9: Aufruf der public function paintToMap des Vessel-Objekts in ais-socket.io.js

Für die letzte Aufgabe, die Animation (Punkt 8), wird das Polygon-Objekt des Leaflet-Frameworks erweitert zur Klasse `L.AnimatedPolygon`. Ein Polygon in Leaflet ist eine Polyline, die mehrere Punkte auf der Karte verbindet. Die Animation eines Punktes entlang einer Linie mit einer bestimmten Geschwindigkeit wurde aus dem Leaflet-Plugin `L.AnimatedMarker`⁵ übernommen. Die dort präferierte `css3-Transition` zur Animation konnte aber nicht verwendet werden, weil dazu ein Objekt im DOM-Baum identifiziert werden muss. Das von Leaflet für ein Polygon erstellte DOM-Objekt (`svn-Graphik`) ist aber durch die Leaflet-Bibliothek gekapselt. Der Versuch, die jeweilige `svn-Grafik` im DOM-Baum für die Animation zu selektieren, ist gescheitert.

Deshalb muss auf eine Lösung zurückgegriffen werden, die die `MapFeatures` des Schiffes (Polygon und Richtungsdreieck) in jedem Animationsschritt neu berechnet und rendert.

Ein Schiffspolygon berechnet sich aus der Schiffsposition (Positionsangabe in der AIS-Nachricht) und aus der relativen Position des AIS-Transceivers an Bord (Abstand zum Bug, zum Heck, nach Backbord und nach Steuerbord). Um zu wissen, in welche Richtung der Bug eines Schiffes zeigt, wird die AIS-Angabe zur Fahrtrichtung bei fahrenden Schiffen verwendet (`cog` = `Course over Ground`). Aus dieser Richtungsangabe und der übermittelten Geschwindigkeit (`sog` = `Speed over Ground`) wird ein `Speedvector` berechnet, der als Linie auf die Karte gezeichnet wird. Er ist genauso lang wie die Strecke, die das Schiff bei kontinuierlicher Weiterfahrt in den nächsten 30 Sekunden zurücklegen wird. Bei der Erstellung des `AnimatedPolygon`-Objektes wird dieser `Speedvector` übergeben, damit das Polygon entlang dieser Linie verschoben werden kann. Über die Optionen `'distance'` und `'interval'` wird bestimmt, in wieviele Teilschritte der Vektor unterteilt wird und wieviel Zeit zwischen zwei Animationsschritten liegt. Nach jedem Animationsschritt wird das Polygon neu berechnet, von der Karte gelöscht und neu gezeichnet.

⁵ <https://github.com/openplans/Leaflet.AnimatedMarker>

4.4.3 HTML5-Server

Diese Server-Implementierung soll genau dieselbe Funktionalität besitzen wie die socket.io-Server-Implementierung (4.4.1). Lediglich der socket.io-Websocket wird durch einen node.js-Websocket nach der HTML5-Websocket-Spezifikation ausgetauscht⁶. Dazu wird in der Datei worker.js das entsprechende Paket ('websocket') eingebunden. Einige Features des socket.io-Paketes müssen jetzt selbst organisiert werden:

- die Clientverwaltung erfolgt in einem Array 'clients', in dem zu jeder Zeit alle verbundenen Websocket-Clients mitsamt ihren Attributen stehen.
- in der Websocket-API können keine eigenen Events definiert werden (siehe API-Dokumentation⁷). Deshalb wird der message-Event genutzt und der Name der aufzurufenden Funktion wird innerhalb der message als "type" übermittelt. Unten ist eine identische Nachricht in den beiden unterschiedlichen Formaten zu sehen.

```
[{"_id":"50d9fdb4bcc2e678a9278c18","aisclient_id":57,"callsign":"OVYC2 ","cog":285,"dest":"HAMBURG ","dim_bow":70,"dim_port":10,"dim_starboard":4,"dim_stern":30,"draught":54,"imo":"9363170","mmsi":220515000,"msgid":1,"name":"RIKKE THERESA ","nav_status":0,"pos":[9.8375,53.54885],"rot":0,"ship_type":80,"sog":10.3,"time_captured":1366734056000,"time_received":1366733855248,"true_heading":286}]
```

Listing 4.10: vom socket.io-Server gesendete Nachricht

```
message origin=ws://127.0.0.1:8090, data={"type":"vesselsInBoundsEvent","vessels":[{"_id":"50d9fdb4bcc2e678a9278c18","aisclient_id":57,"callsign":"OVYC2 ","cog":285,"dest":"HAMBURG ","dim_bow":70,"dim_port":10,"dim_starboard":4,"dim_stern":30,"draught":54,"imo":"9363170","mmsi":220515000,"msgid":1,"name":"RIKKE THERESA ","nav_status":0,"pos":[9.8375,53.54885],"rot":0,"ship_type":80,"sog":10.3,"time_captured":1366733896000,"time_received":1366733855248,"true_heading":286}]}
```

Listing 4.11: vom HTML5-Websocket-Server gesendete Nachricht

4.4.4 HTML5-Client in Javascript

Die HTML5-Client-Implementierung bietet die gleiche Funktionalität wie die socket.io-Client-Implementierung und unterscheidet sich nur marginal.

- wie in Listing 4.11 zu sehen, muss der HTML5-Client zuerst den message-type abfragen, um die Daten korrekt zuzuordnen.
- weil dem Leaflet-Map-Objekt die Websocket-Connection als Parameter übergeben wird, muss der Aufbau der Websocket-Connection abgewartet werden. Deswegen wird die Map-Initialisierung erst durch den onopen-Event des Websockets angestoßen.

⁶ <https://github.com/Worlize/WebSocket-Node>

⁷ <https://github.com/Worlize/WebSocket-Node/wiki/Documentation>

4.4.5 HTML5-Client in Google Dart

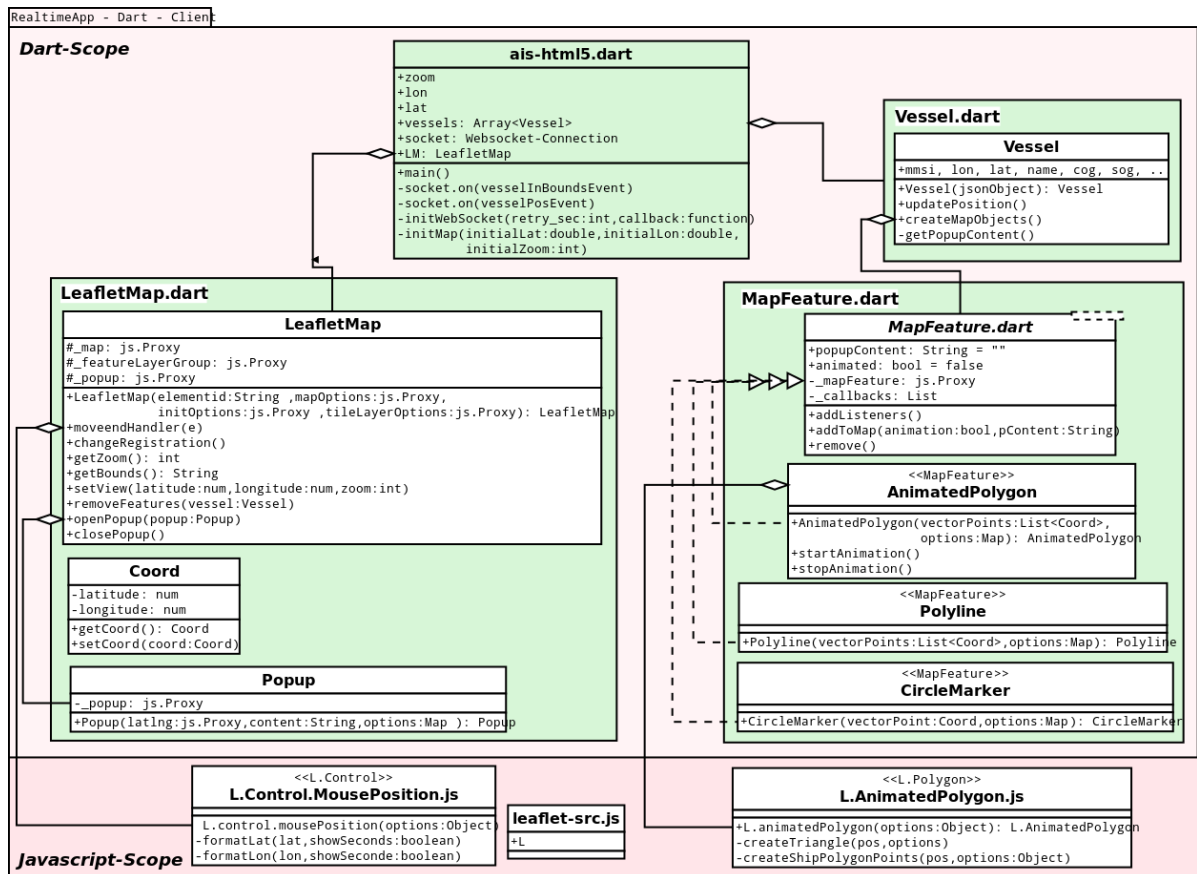


Abbildung 4.3: Schematische Übersicht über implementierte Dart-Dateien und -Klassen, sowie die verwendeten Javascript-Dateien, siehe auch Anmerkungen zu Abbildung 4.2

Die Implementierung des HTML5-Websocket-Clients in Dart orientiert sich an der Implementierung des HTML5-Websocket-Clients in Javascript. Oberstes Ziel dabei ist es, eine mindestens gleichwertige Funktionalität zu erreichen unter Ausnutzung der sprachspezifischen Vorteile von Google Dart.

Die Modularisierung, also die Verteilung der Objekte und Funktionalitäten auf mehrere Dateien, ist in der Dart-Client-Implementierung ähnlich gestaltet wie in Javascript (siehe Abbildungen 4.2 und 4.3). Die verwendeten Javascript-Dateien 'leaflet-src.js', 'L.AnimatedPolygon.js' und 'L.Control.Mouseposition.js' sind identisch mit denen des Javascript-Clients (Abschnitt 4.4.4). Um sie aus Dart heraus nutzen zu können, wird das Dart-Paket js-interop (siehe Abschnitt 3.6) eingebunden.

Eine Dart-Anwendung wird über eine `main`-Funktion (in `ais-html5.dart`) gestartet. Die `main`-Funktion initialisiert den HTML5-Websocket-Client. Im Falle eines erfolgreichen Verbindungsaufbaus mit dem HTML5-Websocket-Server wird eine Callback-Funktion ausgeführt, die das `LeafletMap`-Objekt erstellt. Beide Objekte sind Singletons und bleiben über die Laufzeit der Anwendung erhalten.

Das LeafletMap-Objekt kapselt für die Anwendung den Zugriff auf das Javascript L.Map-Objekt der leaflet.js-Bibliothek (siehe Listing 4.12). Dafür wechselt es in den Javascript-Kontext der Anwendung und initialisiert in diesem ein L.Map-Objekt und ein L.LayerGroup-Objekt. Beide Objekte werden mit der Anweisung `js.retain(...)` im Javascript-Kontext als globale Variable eingeführt, so dass sie für jede folgende Funktion im Javascript-Kontext zur Verfügung stehen.

In der Gegenrichtung muss eine Möglichkeit existieren, vom Javascript-Kontext der Anwendung auf den Dart-Code zuzugreifen. Ein Beispiel dafür ist der `'moveend'`-Listener, der Teil der leaflet.js-Bibliothek ist. Für den Javascript-Listener wird im Dart-Kontext ein `Callback.many`-Objekt erstellt (`moveendHandler(e)`), das die `changeRegistration`-Funktion ausführt. Das `Callback.many`-Objekt kann im Unterschied zum `Callback.once`-Objekt mehrmals aufgerufen werden. Das heißt, jedes Mal, wenn im Javascript-Kontext der `'moveend'`-Listener einen `'moveend'`-Event registriert, ruft er über den Handler die Dart-Funktion `changeRegistration()` auf. Diese Funktion sendet an den HTML5-Websocket eine Nachricht vom Typ `'register'` mit den aktuellen Bounds der Karte.

```
LeafletMap(String elementid, js.Proxy mapOptions, js.Proxy initOptions, js.
Proxy tileLayerOptions){
  boundsTimeout = initOptions['boundsTimeout']*1000;
  js.scoped(() {
    map = new js.Proxy(js.context.L.Map, elementid, mapOptions);
    featureLayerGroup = new js.Proxy(js.context.L.LayerGroup);
    map.addLayer(featureLayerGroup);
    js.retain(featureLayerGroup);
    var tileLayer = new js.Proxy(js.context.L.TileLayer, tileLayerOptions[
      'tileURL'], tileLayerOptions);
    map.addLayer(tileLayer);
    var mouseOptions = initOptions['mousePosition'];
    if( mouseOptions != false)
    {
      var mousePosition = new js.Proxy(js.context.L.Control.MousePosition
        , mouseOptions);
      mousePosition.addTo(map);
    }
    map.setView(new js.Proxy(js.context.L.LatLng, initOptions['lat'],
      initOptions['lon']), initOptions['zoom']);
    js.retain(map);
    map.on('moveend', new js.Callback.many(moveendHandler));
  });
  changeRegistration();
}
```

Listing 4.12: Konstruktor des LeafletMap-Objektes mit Zugriff auf den Javascript-Kontext

Der Websocket-Server antwortet auf diese Nachricht mit einer Nachricht vom Typ `"Vessel-InBoundsEvent"` (siehe Listing 4.11).

Diese Nachricht und die “VesselPositionEvent”-Nachricht empfängt und verarbeitet der Dart-Client genauso wie der Javascript-Client in Listing 4.9. Bei der Ausführung der `paintToMap`-Methode des `Vessel`-Objektes allerdings besteht ein wichtiger Unterschied darin, dass in Javascript direkt `Map-Feature-Objekte` der `Leaflet.js` Bibliothek (`L.Polyline`, `L.AnimatedPolygon` und `L.CircleMarker`) erzeugt werden, wohingegen in Dart der Konstruktor einer Unterklasse von `MapFeature` (`Polyline`, `AnimatedPolygon` oder `CircleMarker`) aufgerufen wird. Die Konstrukturen aller Unterklassen von `MapFeature` wechseln in den Javascript-Kontext, erstellen dort über einen Proxy ein entsprechendes Objekt aus der `Leaflet-Bibliothek` und stellen dieses als Attribut des `MapFeature`-Objektes im Dart-Kontext zur Verfügung (siehe Beispiel Klasse `Polyline` in Listing 4.13).

```
class Polyline extends MapFeature{
  Polyline(List<Coord> vectorPoints, Map options) {
    js.scoped(() {
      var latlng = js.context.L.LatLng;
      var points = js.array([]);
      for(var x = 0; x < vectorPoints.length; x++)
      {
        var lat = vectorPoints[x].latitude;
        var lng = vectorPoints[x].longitude;
        points.push(new js.Proxy(latlng, lat, lng));
      }
      var lineOptions = js.map(options);
      _mapFeature= new js.Proxy(js.context.L.Polyline, points, lineOptions);
      js.retain(_mapFeature);
    });
  }
}
```

Listing 4.13: Konstruktor des Dart-Objekts `Polyline`

Auf diese Weise erhält jedes `Vessel`-Objekt im Array ‘`Vessels`’, der alle auf der Karte dargestellten Schiffe enthält, als Attribut `.feature` und/oder `.polygon` eine Assoziation zu einer Dart-Klasse, die über einen Proxy die Verwaltung des dazugehörigen Javascript-Objektes im DOM-Baum für die Anwendung kapselt.

Nach demselben Prinzip funktionieren Popups in der Anwendung. Um Popups auf den `MapFeature`-Objekten über `MouseEvents` öffnen und schließen zu können, werden auf den `mouseover` und den `mouseout`-EventListener des Javascript-Proxies eines jeden `MapFeature`-Objektes `EventHandler` als Dart-Callback.many-Funktionen registriert (siehe Listing 4.14). Diese `Callback.many`-Funktionen führen im Dart-Kontext alle notwendigen Anweisungen aus. Zum Beispiel erstellt der `MouseoverHandler` ein Dart-Popup-Objekt, das im Konstruktor einen Proxy zu einem `L.Popup`-Objekt erstellt und als Instanzvariable speichert. Operationen auf dem `L.Map`-Objekt müssen über das Dart `LeafletMap`-Objekt ausgeführt werden, das den Javascript-Proxy zum `L.Map`-Objekt verwaltet. Im Falle des `onMouseoverHandlers`, wird das Dart-Popup-Objekt der Funktion `LMap.openPopup(popup)` übergeben, die Funktion wechselt in den Javascript-Kontext und öffnet auf dem `L.Map`-Objekt den Javascript-Proxy

des Popup-Objektes (siehe Listing 4.15)

```
void addListeners(){
  onMouseoutHandler(e){
    LMap.closePopup();
  }
  onMouseoverHandler(e){
    var ll = e.latlng;
    ll = new js.Proxy(js.context.L.LatLng ,ll.lat, ll.lng);
    var popupOptions = {'closeButton': false,
                        'autoPan': false,
                        'maxWidth': 150,
                        'offset' : [50,-50]};

    var popup = new Popup(ll, popupContent, popupOptions);
    LMap.closePopup();
    LMap.openPopup(popup);
  }

  _callbacks.add(new js.Callback.many(onMouseoverHandler));
  _callbacks.add(new js.Callback.many(onMouseoutHandler));

  _mapFeature.on('mouseover', _callbacks[0]);
  _mapFeature.on('mouseout', _callbacks[1]);
}
```

Listing 4.14: EventHandling mithilfe von Callback-Funktionen

```
class LeafletMap {
  js.Proxy map;
  ...
  closePopup(){
    js.scoped(){
      map.closePopup();
    };
  }

  openPopup(Popup popup){
    js.scoped(){
      map.openPopup(popup._popup);
    };
  }
  ...
}
```

Listing 4.15: Zugriff auf das L.Map-Objekt (im Javascript-Kontext) über das LeafletMap-Objekt (im Dart-Kontext) in LeafletMap.dart

5 Ergebnisse

5.1 Evaluation der Anwendung

5.1.1 Funktionale Anforderungen

Alle funktionalen Anforderungen aus Abschnitt 2.2 sind im Prototyp der Anwendung in Javascript mit dem node.js-Framework und dem socket.io-Websocket-Server aus Abschnitt 4.4.1 und -Client aus Abschnitt 4.4.2) vollständig umgesetzt. Im Screenshot des Hamburger Hafens (Abb. 5.1) kann man erkennen, dass liegende Schiffe als Kreise dargestellt werden und fahrende Schiffe als Richtungsdreiecke. Wurden Reisedaten zu einem Schiff übermittelt (AIS-Nachricht vom Typ 4), sind maßstabsgetreue Polygone eingezeichnet, deren Farbe den Schiffstyp kennzeichnet. Ein Popup mit Detailinformationen zum dem Schiff links daneben ist geöffnet. In dieser Zoomstufe ist die Animation bereits aktiv, das heißt für den Betrachter, dass alle angezeigten Richtungsdreiecke sich ihrer tatsächlich Geschwindigkeit angemessen bewegen.

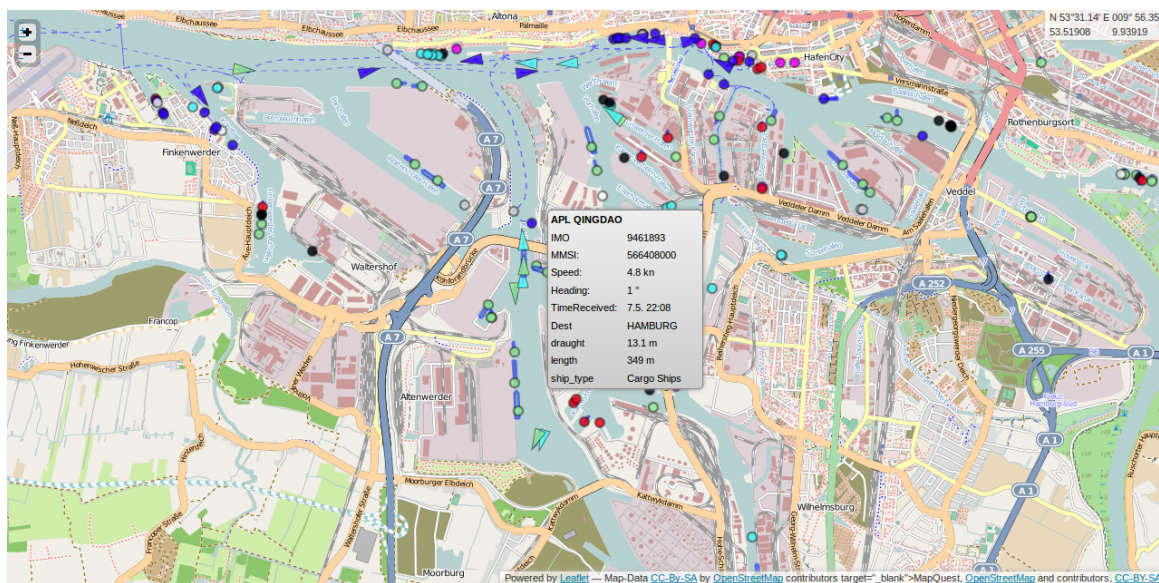


Abbildung 5.1: Anzeige aller Schiffe im Hamburger Hafen

Abbildung 5.2 zeigt die Situation bei niedriger Zoomstufe. Links oben in der Karte ist der Hinweis eingeblendet, dass nur Schiffe mit einer Geschwindigkeit über 12 Knoten angezeigt werden. Die Verteilung des Schiffsverkehrs lässt sich gut erkennen, während die große Zahl hafenliegender Schiffe aus Performacegründen ausgespart bleibt. Aus demselben Grund ist die Animation ausgeschaltet. Durch die hohe Frequenz empfangener Positionsmeldungen ist die Darstellung trotzdem in Bewegung.

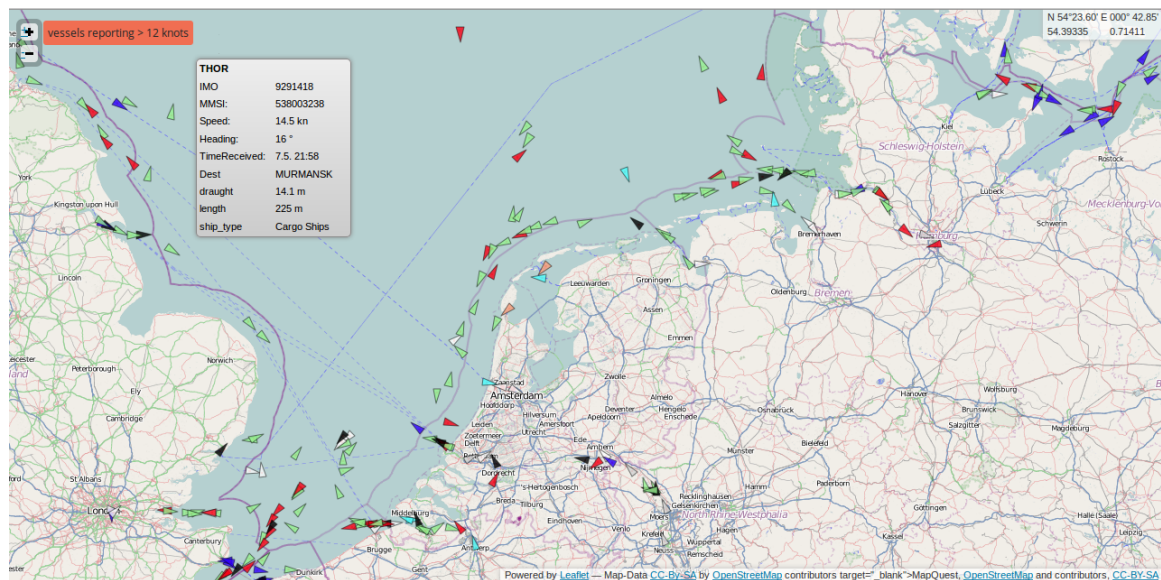


Abbildung 5.2: Auf schnell fahrende Schiffe reduzierte Anzeige am Beispiel der Nordsee

Schließlich ist in hohen Zoomstufen eine Beobachtung von aktuellen Schiffsmanövern möglich, wie in Abbildung 5.1.1 das Schleppen eines Frachtschiffes durch zwei Schleppschiffe. Die hohe Frequenz der Positions-Meldungen bei fahrenden Schiffe und die eingebaute Animation lassen den Betrachter die Szene wie in einem Animationsfilm erleben.



Abbildung 5.3: Schleppmanöver

5.1.2 Nicht funktionale Anforderungen

Die vorgestellte Implementierung erfüllt auch die wichtige nicht funktionale Anforderung nach einer zeitnahen Umsetzung. Sie wurde Ende Februar 2013 an das Unternehmen Vesseltracker übergeben und wird auf github als privates Repository gehostet. Die gesamte Anwendung ist mit open source Produkten entwickelt worden und verwendet das von der Vesseltracker gehostete OpenstreetMap-Kartenmaterial.

Die Anwendung wurde auf den unten angegebenen Browserclients in den angegebenen Versionen positiv auf Funktionalität getestet und unterstützt damit die gängigen Browser in den einschlägigen Versionen ¹:

- Firefox Version 15.0.1 und 20.0
- Google Chrome 26.0
- Internet Explorer Version 9.0 und 10.0
- Safari 6.0.4

¹ <http://www.browser-statistik.de/statistiken/versionen/>

Die Latenzzeit zwischen dem Empfang der AIS-Positions-Meldung durch den Rohdatenserver und der Propagierung derselben Position auf der Karte sollte unterhalb 500 msec liegen. Hier sei verwiesen auf die Ergebnisse in Abschnitt 5.2.1 und Abbildung 5.5. Die gemessenen Latenzzeiten liegen vollständig innerhalb der geforderten Geschwindigkeit.

Die Anzahl der Verbindungen, die der Server gleichzeitig bedienen kann, ist mit einem node.js-Script getestet worden, das alle 500 ms eine neue Clientverbindung erstellt bis 750 Clients verbunden sind. Der Aufbau der Verbindung geschah auch mit steigender Verbindungsanzahl zuverlässig, jedoch ist in Abbildung 5.4 erkennbar, dass die Anzahl der Fälle zunimmt, in denen ein Client lange auf Antwort warten muss.

Eine Skalierung der Serveranwendung ist seitens des socket.io-Servers kein Problem. Statt einen einzigen Worker-Prozess zu generieren, können auch mehrere Worker-Prozesse parallel gestartet werden, die alle dieselbe mongo-Datenbank-Collection abfragen und sich bei derselben redis-Datenbank im Channel 'positionUpdate' registrieren können. Allerdings muss sichergestellt sein, z.B. über unterschiedliche Ports oder unterschiedliche (virtuelle) Server, dass ein verbundener Client mit jedem neuen Request auf demselben Worker-Prozess landet.

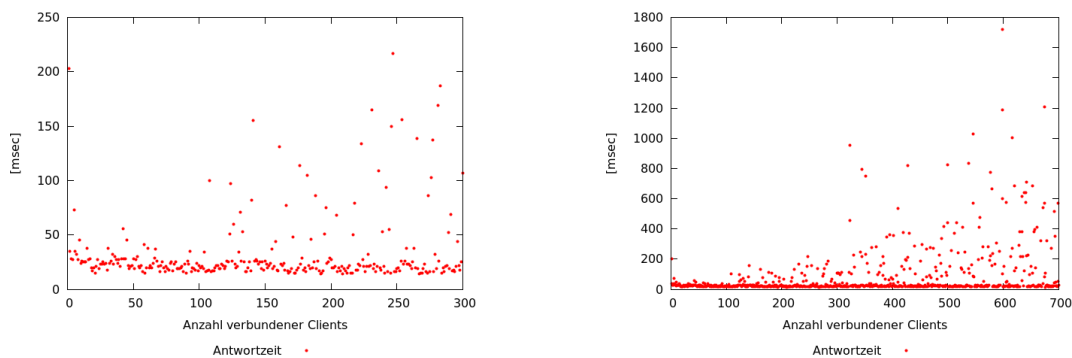


Abbildung 5.4: Antwortzeiten des socket.io-Websocket-Servers in Abhängigkeit von der Anzahl verbundener Clients

5.2 Vergleichende Evaluation der Javascript- und der Dart-Anwendung

Für alle folgenden Tests wird auf dem Rohdatenserver ein Port eingerichtet, der nur die Daten von drei AIS-Antennen (Hamburg, Wedel, Geesthacht) als Datenstrom ausgibt. Dies ist notwendig, weil der als Server (und Client) verwendete Arbeitsplatzrechner (HP 530 Notebook PC mit 1,8 GHz Intel Celeron Processor und 1,5 GB Arbeitsspeicher) die Datenmenge aller AIS-Meldungen weltweit nicht in der Geschwindigkeit verarbeiten kann, in der sie eingeht. Dadurch wächst die Messagequeue des Node.js-Servers und erzeugt eine zusätzliche Latenzzeit, die die Ergebnisse verzerrt.

Die in den Abschnitten 4.2 und 4.3 begründete zusätzliche Implementierung eines HTML5-kompatiblen node.js-Websocket-Servers muss nun in einem ersten Schritt mit dem node.js-socket.io-Server verglichen werden. Anschließend wird die in Google Dart geschriebene Client-Anwendung mit dem in Javascript programmierten Client verglichen.

5.2.1 Vergleich des socket.io-Servers mit dem HTML5-Server

Implementierungsaufwand

Im Implementierungsaufwand unterscheiden sich beide Server- und Client-Anwendungen kaum. Die Unterschiede in Zeilen Code betragen weniger als 10 Zeilen.

Einige Features der socket.io-Bibliotheken (z.B. die Clientverwaltung, Parameter für 'Production' und 'Development'-Umgebung bezüglich Log-Leveln, Client-Minifikation oder Client-Zip) sind praktisch und müssten in der Alternativimplementierung für den Einsatz in einer Produktiv-Umgebung anderweitig gelöst werden. Durch die in socket.io eingeführten Events vermindert sich der Kommunikationsaufwand zwischen Server und Client geringfügig, was in diesem Fall einer datenintensiven Anwendung mit großen Datenmengen pro Nachricht kaum ins Gewicht fällt.

Latenzzeit

Die Leistungsfähigkeit beider Implementierungen wird verglichen, indem die Zeit gemessen wird, die eine Positionsmeldung braucht für den Weg vom Rohdatenserver bis zur Präsentation auf der Karte. Auf dem Rohdatenserver wird jeder AIS-Message beim Empfang ein Zeitstempel (time_received) hinzugefügt. Dazu läuft auf dem Rohdatenserver ein ntp-Daemon zur Zeitsynchronisation. Genau so ein Daemon läuft auf dem Client, auf dem der Browser gestartet wird. Ein Zeitstempel wird genommen, nachdem das Schiff mit der neuen Position auf die Karte gerendert wurde. Die Latenzzeit berechnet sich dann aus der Differenz zwischen der time_received und diesem Zeitstempel.

Um eine ähnliche Situation in beiden Szenarien abzubilden, wird jeweils eine bestimmte Position im Hamburger Hafen mit maximaler Zoomstufe angesteuert und ein Timer in die Client-Anwendungen integriert, der jeweils nach einer Minute um eine Stufe herauszoomt. Da ab Zoomlevel 11 und kleiner nur noch Schiffe mit einer jeweils definierten Mindestgeschwindigkeit angefordert werden, nimmt die Anzahl empfangener Schiffe von dieser Zoomstufe an wieder ab. Zur Auswertung wird auf dem Client ein LogFile geschrieben, das für jede Positionsmeldung deren 'time_received' und einen aktuellen Zeitstempel festhält. Die Differenz wird als Latenzzeit interpretiert. Anschließend wird über das Logfile berechnet, wieviele Positionsmeldungen in einer Minute an den Client gesendet wurden. In den Abbildungen 5.5 und 5.6 sind jeweils beide Ergebnisse über die Zeit der Beobachtung (ca. 15 min) aufgetragen.

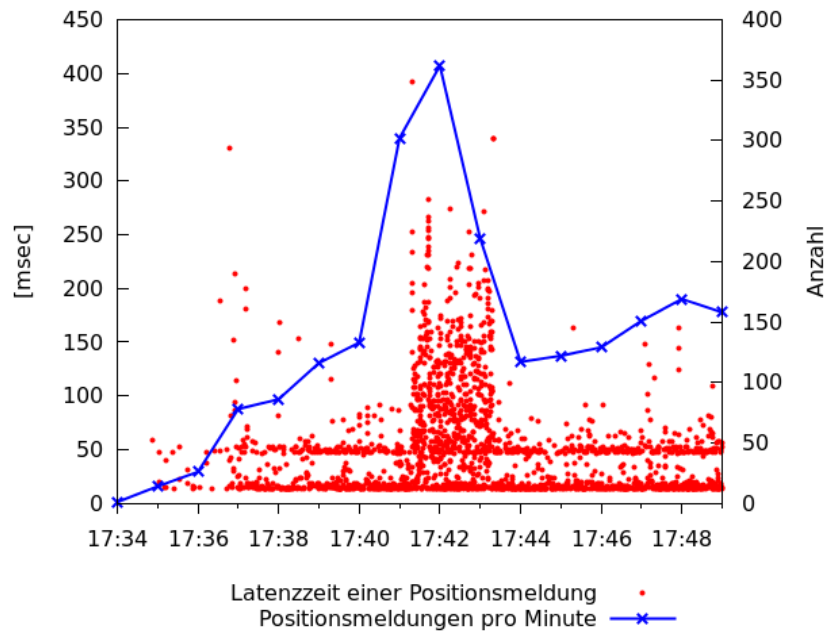


Abbildung 5.5: socket.io-Websocket-Server: Latenzzeit der Positionsmeldungen und Anzahl empfangener Schiffe

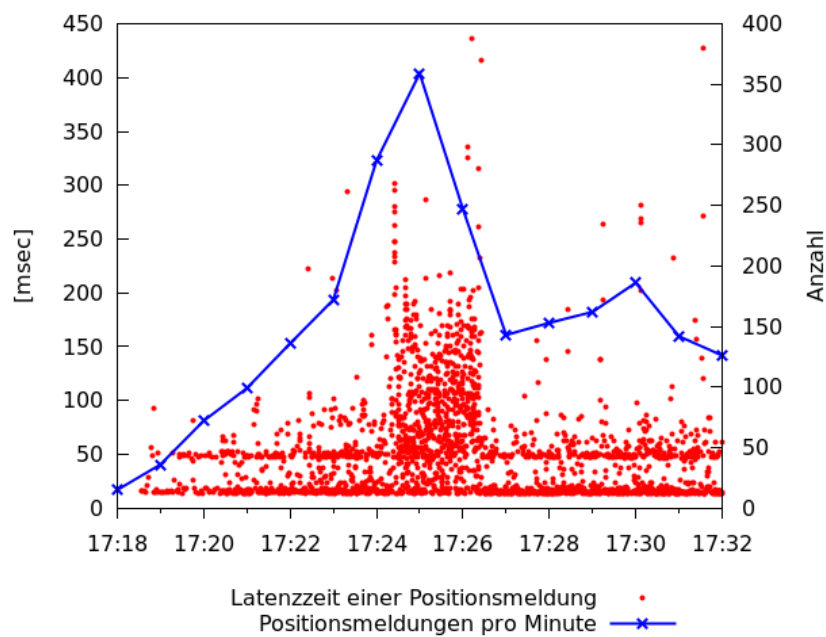


Abbildung 5.6: HTML5-Websocket-Server: Latenzzeit der Positionsmeldungen und Anzahl empfangener Schiffe

Es ist offensichtlich, dass die Geschwindigkeit in der Darstellung von der Anzahl empfangener Nachrichten linear abhängt. Darüber hinaus ist zu erkennen, dass beide Implementierungen ihre Aufgabe in vergleichbarer Geschwindigkeit erledigen. Das bedeutet, dass die HTML5-Node.js-Anwendung in ihrer Leistungsfähigkeit der socket.io-Node.js-Anwendung gleichwertig ist und im Folgenden als repräsentative Javascript-Lösung verwendet werden kann.

Browserunterstützung

Für die node.js-HTML5-Websocket-Anwendung wird wie für die socket.io-Websocket-Anwendung in Abschnitt 5.1 die Unterstützung durch die gängigen Browser getestet (Tabelle 5.1).

Browser	Version	socket.io	HTML5
Firefox	20.0	websocket	websocket
Google Chrome	26.0	websocket	websocket
Internet Explorer	9	Flashsocket	nicht unterstützt
Internet Explorer	10	websocket	websocket
Safari	6.0.4	websocket	websocket

Tabelle 5.1: Unterstützung von Browsern durch die socket.io- und HTML5-Websocket-Clients

Erwartungsgemäß unterstützen alle gängigen aktuellen Browser HTML5-kompatible Websockets². Für ältere Browser ohne Websocket-Unterstützung wie den IE 9 bietet socket.io einen Fallback auf Adobe Flashsocket. Listing 5.1 zeigt den Client-Request des socket.io-Clients im IE 9, der vom Server den Flashsocket (WebSocketMain.swf) anfordert. In der HTML5-Anwendung dagegen beendet der IE 9 mit einem Scriptfehler die Anwendung.

```
Anforderung      GET /socket.io/static/flashsocket/WebSocketMain.swf HTTP/1.1
Accept           */*
Accept-Language  de-DE
Referer          http://192.168.1.214:8090/ais-socket.io_js.html
x-flash-version  10,0,32,18
Accept-Encoding  gzip, deflate
User-Agent       Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident
                /5.0)
Host             192.168.1.214:8090
```

Listing 5.1: socket.io Client-Request in Internet Explorer 9

² <http://caniuse.com/#search=websocket>

5.2.2 Vergleich des Javascript-Clients mit dem Dart-Client

Implementierungsaufwand

Die Möglichkeit des objektorientierten Programmierens in Dart bringt beim Implementieren des Dart-Clients die größte Erleichterung gegenüber der Implementierung in Javascript. Die geschaffene Struktur ist in Abbildung 4.3 zu erkennen. Javascript bietet zwar zahlreiche Lösungen, um Objektorientierung nachzubilden, z.B. über Funktionen als Klassen-Objekte und Konstrukte wie das Revealing Module Pattern. Dadurch wird das Programmieren in Javascript aber mit zunehmender Komplexität der Anwendung unübersichtlicher und fehleranfälliger, weil die Features objektorientierter Sprachen sozusagen per Hand gepflegt werden müssen. Die entstehenden Strukturen sind weniger transparent und selbsterklärend als in Dart (4.2).

Die Dart-Implementierung der Anwendung wird an dem Punkt etwas aufwändiger, wo über das Paket js-interop die Javascript-Dateien integriert werden und Proxies und Callback-Funktionen geschrieben werden müssen zur Kommunikation zwischen dem Javascript- und dem Dart-Kontext. Der quantitative Mehraufwand spiegelt sich auch in einem signifikanten Unterschied in der Anzahl Zeilen Code (Tabelle 5.2).

Client	Zeilen Code
Javascript	611
Dart	867

Tabelle 5.2: Anzahl Zeilen Code im Vergleich der Clients

Mit dem dart2js-Compiler ließ sich der Dart-Client-Code zu Javascript kompilieren und war dann auch auf Browsern ohne Dart VM ausführbar. Dabei traten gelegentlich Fehler auf, die entweder auf Bugs zurückzuführen waren und mit dem nächsten Update behoben waren oder Fehler, die durch Änderungen im Dart-Code behoben werden mussten, obwohl der originäre Dart-Code von der Dart VM korrekt interpretiert wurde. Dazu ein Beispiel: Wird innerhalb des Javascript-Kontexts eine Methode auf einen Javascript-Proxy (hier `_map`) aufgerufen und ein Javascript-Proxy wird zurückgegeben, dann ist es möglich, auf diesen Proxy, der in diesem Fall vom Typ `LatLngBounds` ist, eine Methode der Klasse `LatLngBounds` aufzurufen (siehe Listing 5.2). Im zu Javascript kompilierten Dart-Code lief die Anwendung damit jedoch in einen Fehler:

“=> TypeError: t1.get\$_map(...).getBounds\$0(...).getSouthWest\$0 is not a function”

```
List getBounds(){
  var south, west, north, east;
  js.scoped((){
    south= _map.getBounds().getSouthWest().lng;
    west = _map.getBounds().getSouthWest().lat;
    north = _map.getBounds().getNorthEast().lng;
    east = _map.getBounds().getNorthEast().lat;
  });
  return [west, south, east, north];
}
```

```

changeRegistration(){
  int zoom = getZoom();
  var boundsList = getBounds();
  Map _southWest = {"lng":boundsList[0],"lat":boundsList[1]};
  Map _northEast= {"lng":boundsList[2],"lat":boundsList[3]};
  Map bounds = {"_southWest": _southWest,"_northEast":_northEast};

  Map message = new Map();
  message['function'] = 'register';
  message['zoom'] = zoom;
  message['bounds'] = bounds;
  socket.send(stringify(message));

  boundsTimeoutTimer = new Timer(new Duration(milliseconds:boundsTimeout),
    changeRegistration);
}

```

Listing 5.2: LeafletMap.dart mit ursprünglicher Abfrage des Bounds-Objektes

Weil kein Bug zu diesem Problem über die Dart-Community³ zu finden ist, wird das Problem mit einem Workaround behoben. Und zwar wird über die Methode 'getBBoxString' des Bounds-Objekt ein String mit den Bounds geholt. Aus diesem String sind anschließend die Double-Werte für die Bounds zu parsen (Listing 5.3).

```

String getBounds(){
  String bBox;
  js.scoped((){
    bBox = \_map.getBounds().toBBoxString();
  });
  return bBox;
}

changeRegistration(){
  int zoom = getZoom();
  var boundsArray = getBounds().split(",");
  Map _southWest = {"lng":double.parse(boundsArray[0]),"lat":double.parse(
    boundsArray[1])};
  Map _northEast= {"lng":double.parse(boundsArray[2]),"lat":double.parse(
    boundsArray[3])};
  Map bounds = {"_southWest": _southWest,"_northEast":_northEast};
  Map message = new Map();
  message['function'] = 'register';
  message['zoom'] = zoom;
  message['bounds'] = bounds;
  socket.send(stringify(message));
  boundsTimeoutTimer = new Timer(new Duration(milliseconds:boundsTimeout)
    ,changeRegistration);
}

```

Listing 5.3: LeafletMap.dart mit workaround über BBoxString

³ <https://www.dartlang.org/community/>

Performance

Um die Performance des Javascript-Clients mit der des eigentlichen Dart-Clients und des zu Javascript kompilierten Dart-Clients zu vergleichen, wurde der VesselsInBounds-Event genutzt. Gemessen wurde die Zeit, die benötigt wird, um nach Empfang eines VesselInBounds-Events alle in der Message enthaltenen Schiffe auf die Karte zu rendern. Verglichen wurden die Clients in den Browsern Dartium, Chrome und Firefox.

- Google Dartium interpretiert den originären Dart-Client mit der Dart VM. Beim Aufruf des Javascript-Clients in Dartium wird der Javascript-Code mit der V8-Javascript-Engine interpretiert.
- Google Chrome mit der V8-Javascript-Engine führt beim Aufruf des Dart-Clients den zu javascript kompilierten Dart-Code aus und beim Aufruf des Javascript-Clients den originären Javascript-Code.
- ebenso führt Firefox mit der SpiderMonkey Javascript Engine den zu Javascript kompilierten Dart-Client-Code, bzw. den originären Javascript-Client-Code aus.

In einer ersten Testserie wurden die Browserclients auf dem HP 530 Notebook PC mit 1,8 GHz Intel Celeron Processor und 1,5 GB Arbeitsspeicher ausgeführt, auf dem auch die Server-Anwendung läuft.

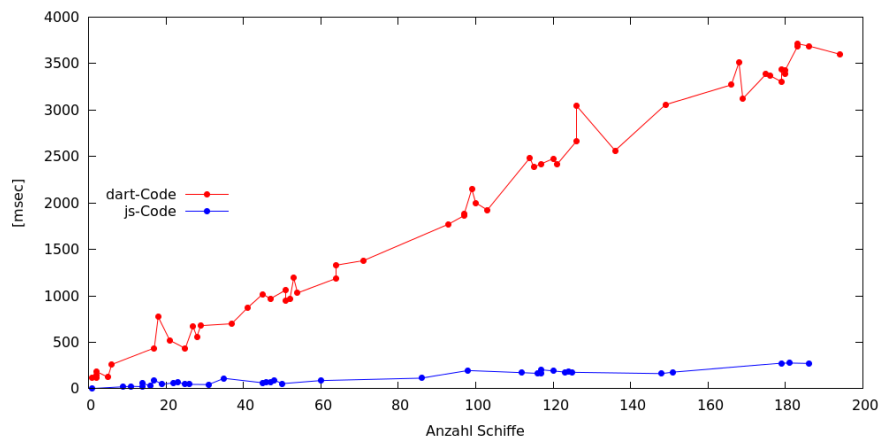


Abbildung 5.7: Verarbeitungsdauer in Dartium (HP 530 Notebook PC)

5 Ergebnisse

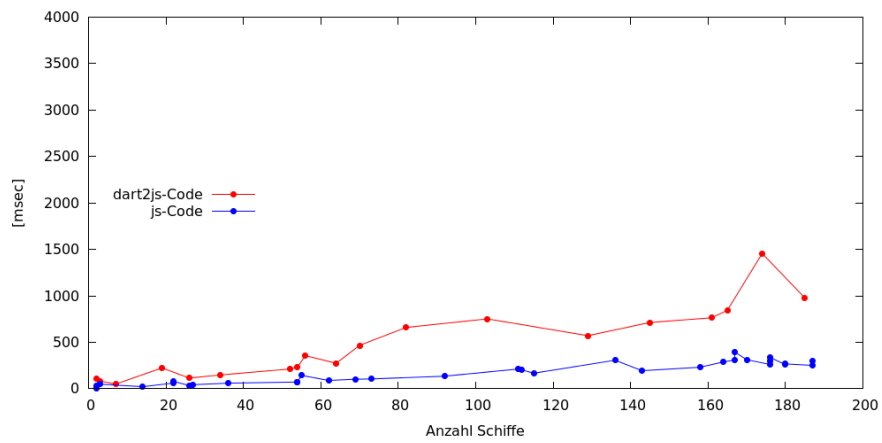


Abbildung 5.8: Verarbeitungsdauer in Chrome (HP 530 Notebook PC)

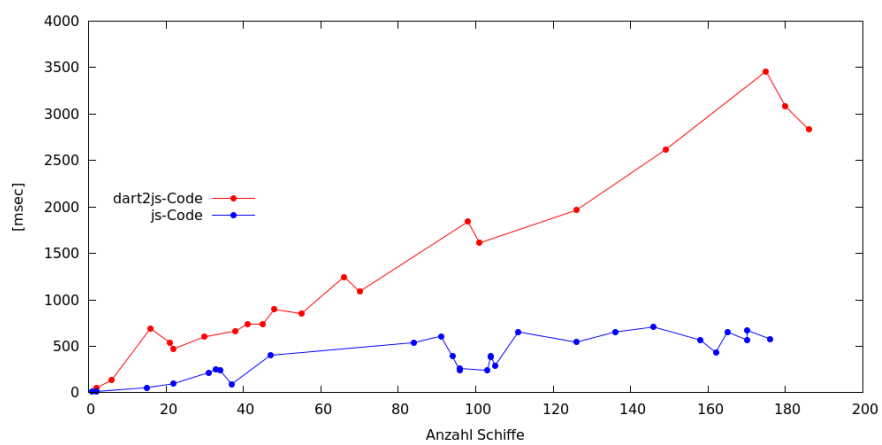


Abbildung 5.9: Verarbeitungsdauer in Firefox (HP 530 Notebook PC)

In einer zweiten Testserie wurden die Browser-Clients auf einem Apple MacBook Pro 10 mit 2,3 Ghz Intel Core Processor und 8 GB Arbeitsspeicher ausgeführt.

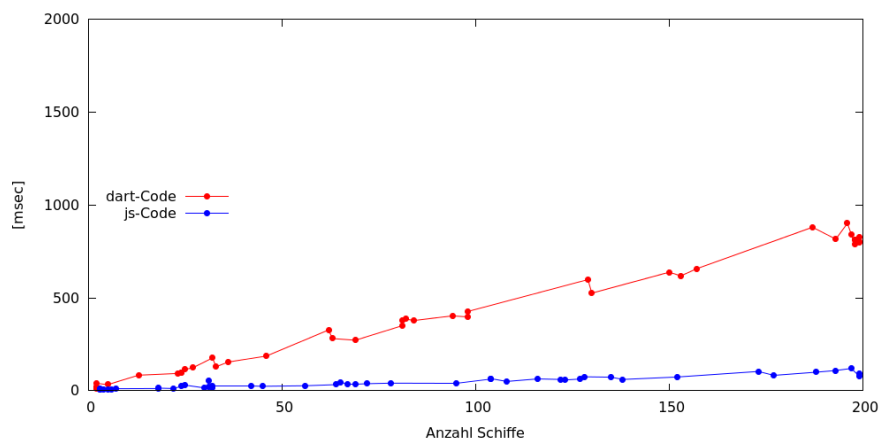


Abbildung 5.10: Verarbeitungsdauer in Dartium (Apple MacBook Pro)

5 Ergebnisse

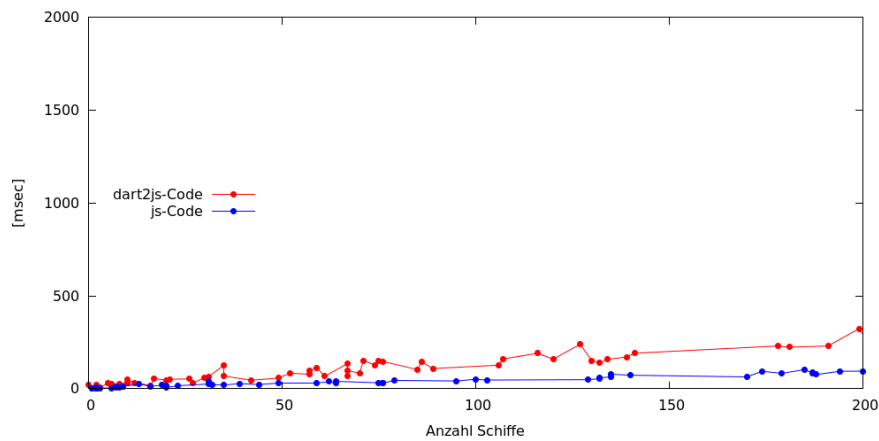


Abbildung 5.11: Verarbeitungsdauer in Chrome (Apple MacBook Pro)

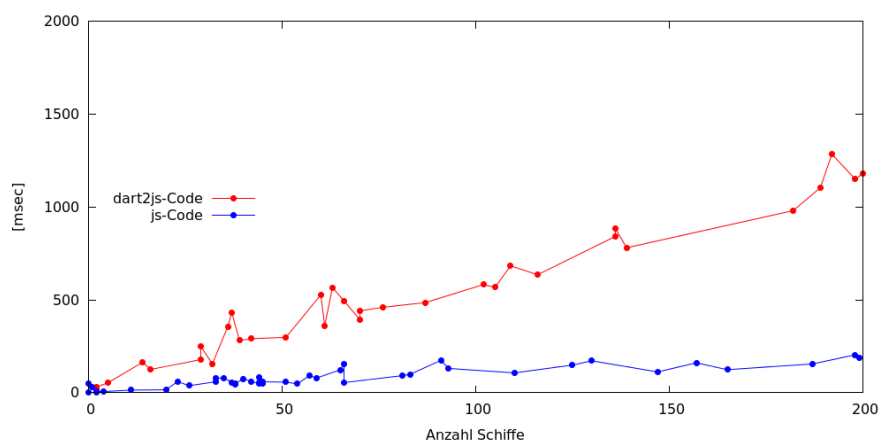


Abbildung 5.12: Verarbeitungsdauer in Firefox (Apple MacBook Pro)

Insgesamt ist erkennbar, dass der Javascript-Code schneller ausgeführt wird als der Dart-Code. Am schnellsten führt die V8-Javascript-Engine in Dartium und Chrome den originären Javascript-Code aus. Firefox mit der Spidermonkey-Javascript-Engine benötigt deutlich länger.

Auffällig ist, dass der Dart-Code von der Dart VM in Dartium langsamer ausgeführt wird als der zu Javascript kompilierte Code. In der zweiten Testserie bestätigen sich diese Ergebnisse. Es ist offensichtlich, dass sich die Geschwindigkeit der Darstellung durch einen leistungstärkeren Client-Rechner verbessern lässt. Dabei ist deutlich erkennbar, dass der Dart-Client die größte relative Performance-Steigerung erfährt, also am meisten von der zusätzlichen Rechenleistung profitiert.

Browserunterstützung

Zusätzlich zu den in den Performance-Tests verwendeten Browsern wurde die Anwendung im InternetExplorer 10 und Safari 6 positiv auf Funktionalität getestet. Das heißt, alle gängigen aktuellen Browser können den zu Javascript kompilierten Dart-Code interpretieren.

6 Fazit und Ausblick

6.1 Die Real-Time-Anwendung

Der für die Firma Vesseltracker programmierte Prototyp einer Real-Time-Anwendung in Javascript unter Verwendung von node.js mit socket.io ist im Zuge dieser Arbeit Ende Februar 2013 abgeschlossen worden. Die Anwendung soll als zusätzliches Angebot in die Webanwendung der Firma integriert werden. Für einzelne Kunden (z.B. das Maritime Museum Hamburg) ist die Anwendung bereits zugeschnitten und ausgeliefert worden, weitere Kundenprojekte sind geplant.

Für die bessere Nutzbarkeit der Anwendung sollte noch eine Möglichkeit geschaffen werden, favorisierte Häfen oder Positionen zu speichern, die per Klick angesteuert werden können, z.B. in einer aufklappbaren Liste.

Zur weiteren Optimierung der Anwendung, sollte die Möglichkeit, die Animation der Richtungsdreiecke und Schiffspolygone über css3-Transition-Funktionen zu realisieren, unbedingt weiterverfolgt werden. Damit könnte die vom Browserclient zu erbringende Rechenleistung erheblich reduziert werden, weil die Objekte nicht neu berechnet und erstellt werden müssten, sondern nur über ihre css-Eigenschaften bewegt werden könnten.

6.2 Vergleich Javascript und Google Dart

Das Erlernen von Dart ist für Umsteiger von Javascript relativ unkompliziert. Von den vielen Features und Möglichkeiten, die Dart bietet, kommt in der Client-Anwendung nur ein kleiner Teil zum Tragen. Der Fokus lag hier nicht auf den Möglichkeiten von Google Dart, sondern auf der Bewältigung einer konkreten Anforderung.

Wie in den Abbildungen zur Struktur der Anwendung erkennbar (Abbildung 4.3), ist mit Dart sehr einfach eine Klassen-Struktur herzustellen. Das macht die Anwendung verständlicher und entsprechend einfacher zu entwerfen, zu schreiben und zu warten.

Hilfreich bei der Entwicklung war der DartEditor als Entwicklungsumgebung. Der Compiler gibt bei Syntax-Fehlern aussagekräftige Fehlermeldungen.

Google Dart befindet sich noch in der Entwicklungsphase, so dass gelegentlich nach den ca. wöchentlichen Versions-Updates von Dart die Anwendung unter der neuen Version nicht mehr lauffähig ist und angepasst werden muss. Belohnt wird dieser Aufwand sozusagen mit einer kontinuierlichen und spürbaren Verbesserung der Performance (Beobachtung im Zeitraum November 2012 bis April 2013). Das Einbinden existierender Javascript-Bibliotheken mit js-interop erweitert die Nutzungsmöglichkeiten von Dart ganz entscheidend.

Das Arbeiten in zwei unterschiedlichen Kontexten (Javascript und Dart) ist am Anfang sehr fehleranfällig. Fehler sind schwierig zu debuggen, weil auch die Debugger (hier Firebug und der Debugger im DartEditor) nur jeweils bis an die Grenzen dieses Kontextes reichen.

Dass in Testserie 2 die Ausführung des Dart-Codes besonders von der besseren Rechenleistung profitiert, lässt vermuten, dass die Kommunikation zwischen den Kontexten innerhalb der Anwendung über Proxies und Callback-Funktionen auch in der Ausführung des Programms einen deutlich höheren Rechenaufwand erzeugt. Die in dieser Arbeit geschriebene Realtime-Anwendung ist für den ausführenden Client ohnehin sehr rechenintensiv.

1. werden mit hoher Frequenz Objekte erstellt und gelöscht (besonders durch die Animation der Polygone und Richtungsdreiecke),
2. werden häufig Events im Dart-Kontext erzeugt (Vessel-Position-Events auf dem Websocket), die Aktionen im Javascript-Kontext notwendig machen und
3. werden bei Benutzeraktionen Events im Javascript-Kontext erzeugt (mouseover-, mouseout- oder moveend-Event), die Aktionen im Dart-Kontext notwendig machen.

Dart bleibt also in der Performance deutlich hinter Javascript zurück. Bei der bisherigen Geschwindigkeit der Entwicklung, insbesondere im Hinblick auf die erlebte Performance-Steigerung, hat Google Dart aber durchaus das Potential, Javascript Konkurrenz zu machen. Die Einbindung von Javascript-Bibliotheken in Dart ist eine gute Möglichkeit, an die bestehende Web-Welt anzuknüpfen, obwohl der Mehraufwand, der durch die Kommunikation zwischen den beiden Kontexten entsteht sowohl in der Implementierung als auch in der Rechenlast im Browser-Client erheblich ist.

Vermutlich wird die Zukunft von Dart sich an zwei Fragen entscheiden:

- Wie wird die kritische OpenSource Entwicklergemeinde die Sprache aufnehmen, das heißt werden in Zukunft Dart-Bibliotheken in ähnlicher Vielzahl und Diversität entwickelt werden wie in Javascript-Bibliotheken?

Dr. Axel Rauschmayer beantwortet die Frage, wie quelloffen Google Dart im Vergleich zu Javascript wirklich ist: " In contrast, Dart is almost proprietary: It is not developed in the open and only shown to the public once it is finished (Brendan Eich calls this 'delayed-open'). Plus, all of the developers work for Google. It's great that it is free and eventually open source, both of which will help its popularity. But it probably will never be nearly as open as JavaScript – Google does not have a good track record when it comes to writing specifications for things it open-sources. A recently leaked Android email reminds us that Google is not always as innocent about openness as they sound. Quoting it verbatim:

- Do not develop in the open. Instead, make source code available after innovation is complete
- Lead device concept: Give early access to the software to partners who build and distribute devices to our specification (ie, Motorola and Verizon). They get a non-contractual time to market advantage and in return they align to our standard.

“[Rau11]

- Wie werden sich die anderen großen Mitspieler auf dem Browsermarkt wie Mozilla, Microsoft und Apple zu Dart positionieren¹, das heißt, wird es in Zukunft auch in anderen Browsern Dart Virtual Machines geben? Dr. Axel Rauschmayer äußert sich dazu: “If you are a competing browser vendor, your prospects for adopting Dart are as follows:
 - It is difficult to integrate into your existing infrastructure (as you couldn’t give early feedback).
 - Its development is mainly controlled by Google.
 - You give Google a head start of over two years.

“[Rau11].

¹ <http://www.netzwelt.de/news/89916-dart-google-streitet-microsoft-apple-mozilla.html>

Literaturverzeichnis

- [Dor04] Dornhöfer, Dipl.-Psych. Sascha M.: *Veränderungsblindheit - Drei explorative Untersuchungen in statischer und dynamischer verkehrsbezogener Umgebung*, Fakultät Mathematik und Naturwissenschaften der Technischen Universität Dresden, Dissertationsschrift, August 2004
- [Lad12] Ladd, Seth: *Dart: Build HTML5 Apps Fast*. <http://www.drdoobbs.com/open-source/dart-build-html5-apps-fast/240005631>. Version: September 25, 2012
- [Men12] Menon, Vijay: *Using JavaScript from Dart: The js Library*. <http://www.dartlang.org/articles/js-dart-interop/>. Version: September 2012
- [MU12] Malte Ubl, Eiji K.: *Introducing WebSockets: Bringing Sockets to the Web*. <http://www.html5rocks.com/en/tutorials/websockets/basics/>. Version: March 13, 2012
- [Mü12] Müller, Frank: *Ein Jahr JavaScript-Konkurrent Dart*. <http://www.heise.de/developer/artikel/Ein-Jahr-JavaScript-Konkurrent-Dart-1727116.html>. Version: October 12, 2012
- [Rau11] Rauschmayer, Dr. A.: *Google Dart to 'ultimately ... replace JavaScript'*. <http://www.2ality.com/2011/09/google-dart.html>. Version: September 2011
- [Tei12] Teixeira, Pedro: *Professional Node.js*. Wrox, 2012. – 408 S.
- [Var11] Varaksin, Oleg: *Kommunikationstechnologien für webbasierte Client-Server-Anwendungen*, Fernuniversität Hagen, Diplomarbeit, 2011
- [Wik13] Wikipedia: *Automatic Identification System*. http://de.wikipedia.org/wiki/Automatic_Identification_System. Version: 2013. – [Online; Stand 13. Mai 2013]

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

Ort, Datum

Unterschrift