Institut für Mathematik und Informatik
Fernuniversiät Hagen

Vergleichende Implementierung und Evaluierung einer ereignisgesteuerten, nicht blockierenden I/O Lösung für eine datenintensive Real-Time Webanwendung in Javascript und Dart

Bachelorarbeit

Barbara Drüke
Matrikel-Nummer 7397860

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

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

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-Postionsdaten. 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 Geodatenbank speichert und sie mit den Schiffs-Stammdaten und Satelliten-AIS-Daten in Beziehung setzt.

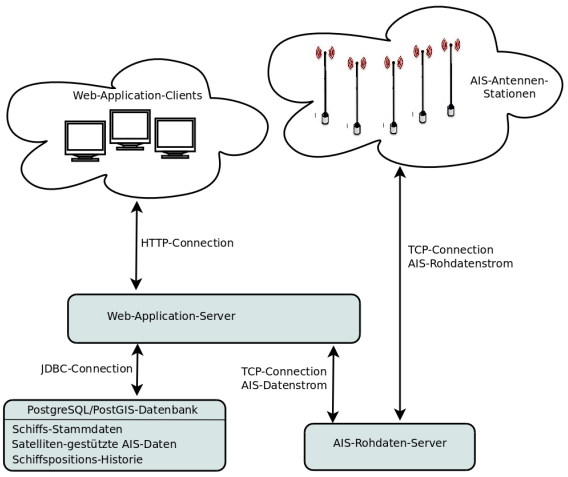

Abbildung 1.1: Vesseltracker_Webapplikation

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 h anzeigen lassen.

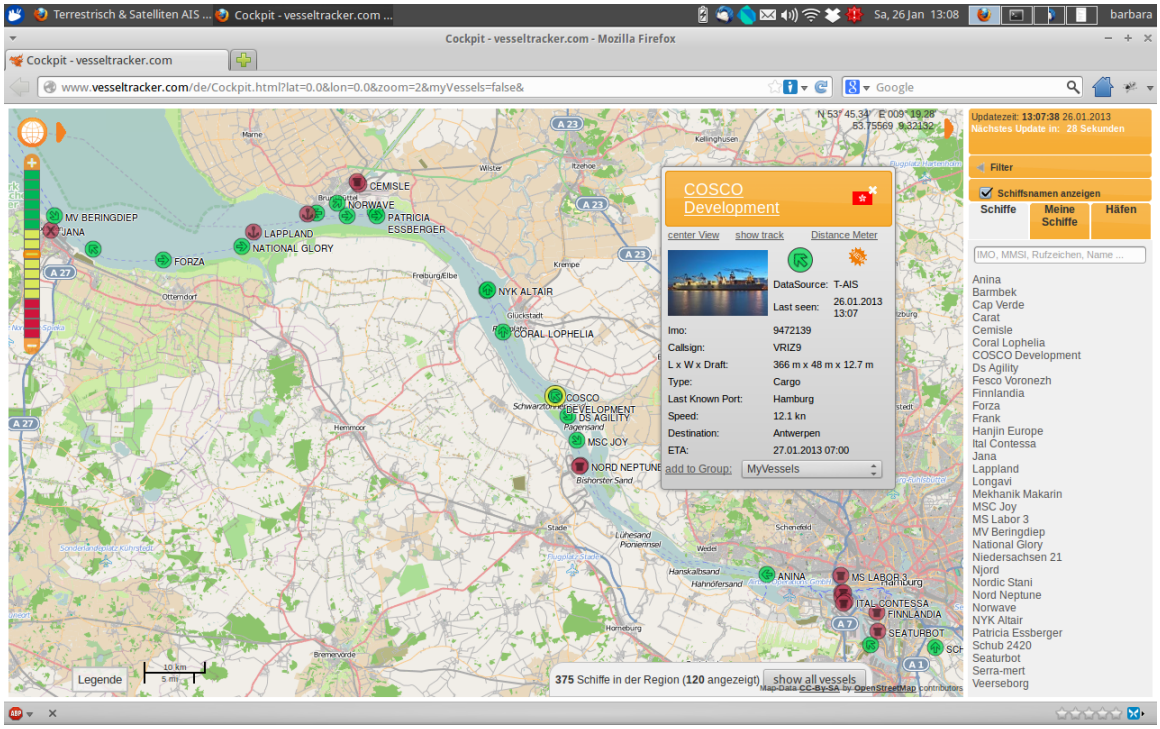

Abbildung 1.2: Cockpit_Elbe

1.1 Motivation für diese Arbeit

Aus mehreren Gründen erscheint es angebracht, die geographischen Schiffspositionen nicht nur über die Cockpit-Anwendung anzubieten, sondern alternativ als real-time-Darstellung.

- Aufgrund der herausragenden Qualität des vesseltracker.com Antennen-Netzwerks sind die verfügbaren AIS-Daten aktuell, aktualisieren sich kontinuierlich und erreichen eine hohe weltweite Abdeckung. Damit ist es möglich, die Schiffsverkehrslage beliebiger Häfen, Wasserstraßen, Küstengebiete weltweit und sekundengenau zu präsentieren.
- Ein Phänomen in der menschlichen Wahrnehmung lässt die geplante Anwendung sehr viel zweckmäßiger erscheinen als die bisherige Cockpit-Anwendung. 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. Im Cockpit werden nach dem Laden neuer Positionsdaten alle Schiffsicons neu gerendert und unter Umständen Namens-Fähnchen gelöscht oder hinzugefügt, was zu einem kurzen "Flackern" führt. Dadurch ist es dem Betrachter nahezu unmöglich, die Positionsänderung eines Schiffes auf der Karte nachzuvollziehen.

- Real-time-Anwendungen gewinnen zunehmend 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 innezuhaben, ist eine Realtime zwingend erforderlich, in einem nächsten Schritt sicher auch als Anwendung für Mobile Devices.

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. Kapitel 3.1 gibt eine kurze Einführung in die Websocket-Technologie, Kapitel 3.2 stellt die Programmiersprache Google Dart vor. In Kapitel werden zunächst die Gründe für die getroffene Auswahl an Implementierungen dargelegt und anschließend die Vorgehensweise bei der Implementierung erläutert. Die Implementierungen werden auszugsweise vorgestellt. Die ausgearbeiteten Implementierungen werden dann in Kapitel 5 getestet und nach verschiedenen Aspekten verglichen. Kapitel 6 fasst die Ergebnisse zusammen.

2 Realtime-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 welches Schiff schleppen, wo Lotsen an oder von Bord gehen, von welchen Tankern Schiffe betankt werden. Sie kontrollieren die eigenen Aufträge oder auch die der Mitbewerber. Die Anwendung läuft hierbei eher statisch, das heißt Zoomstufe und Kartenausschnitt ändern sich nur selten. Es ist also notwendig, dass die Anwendung unabhängig von Aktionen des Nutzers sich laufend oder regelmäßig aktualisiert.

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

Weitere Anwendungsfälle sind Nutzer aus der Passagierschifffahrt, Schiffsfotografen oder Sicherheitsorgane (z.B. die Wasserschutzpolizei), bei denen die Beobachtung / Überwachung bestimmter Wasserverkehrswege oder Häfen von besonderem Interesse ist.

Die vesseltracker.com GmbH nutzt die Realime-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 das in der Einleitung beschriebene “Flackern” dargestellt werden. Damit kann vesseltracker.com die höhere Genauigkeit

und Aktualität der eigenen Daten gegenüber denen anderer Anbieter herausstellen.

Die Anwendungsfälle verdeutlichen noch einmal, dass der zusätzliche Nutzen der Realtimeanwendung gegenüber der Cockpitanwendung nicht ausschließlich im Informationsgehalt liegt. Die Daten im Cockpit sind ja ebenfalls im Minutenbereich aktuell. Der Vorteil liegt vielmehr in der Lebendigkeit der Darstellung. Bewegte Darstellungen binden stärker und für einen längeren Zeitraum die Aufmerksamkeit des Betrachters.

2.2 Beschreibung der Anforderungen

Die funktionalen Anforderungen sind:

- als Datenquelle sollen ausschließlich die vom Rohdatenserver als JSON-Datenstrom zur Verfügung gestellten AIS-Informationen dienen
- Schiffe sollen an ihrer aktuellen (realtime) 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 Icons dargestellt werden, die den Navigationsstatus und gegebenenfalls den Kurs wiederspiegeln
- bei hoher Auflösung und ausreichend statischen AIS-Informationen soll ein Schiff als Polygon in die Karte eingezeichnet werden.
- bei geringer Auflösung ist ein Überblick über die Verteilung der empfangenen Schiffe zu vermitteln
- Detail-Informationen zu jedem Schiff sollen als Popups über das Icon abrufbar sein

Nicht funktionale Anforderungen sind:

- die von den Antennen empfangenen AIS-Daten sind mit minimaler Verzögerung (< 500 msec) auf der Karte darzustellen

- die Anwendung sollte ca. 300 Verbindungen gleichzeitig erlauben und skalierbar sein
- als Clients der Anwendung sollten die gängigsten Browser unterstützt werden (IE, Chrome, Firefox, Safari, Opera)
- die Implementierungen werden auf Github als privates repository gehalten
- als Kartenmaterial sind die von vesseltracker gehosteten OpenstreetMap-Karten zu verwenden
- verwendete Software-Module sollten frei zugänglich sein (open source)
- ein Prototyp der Anwendung soll schnell zur Verfügung stehen. Dieser Prototyp soll Mitarbeitern und Partnern ermöglichen, ihre Anforderungen genauer zu spezifizieren oder sogar neue 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 zweifaches zu leisten hat, nämlich

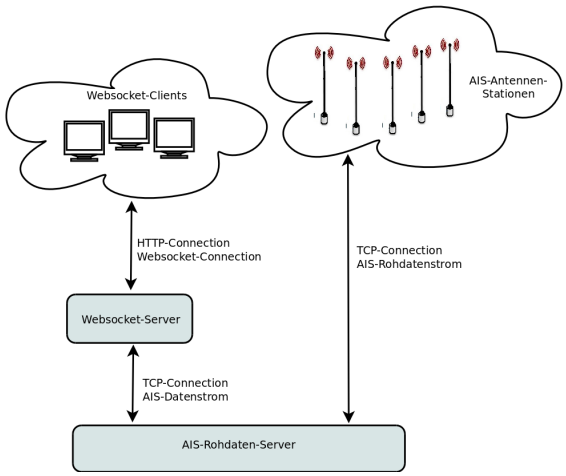

Abbildung 2.1: Architektur-Entwurf der Realtime Webapplikati

1. eine tcp-socket-Verbindung zum Rohdatenserver zu unterhalten und
2. eine bidirektionale Verbindung zum HTTP-Client zu halten, in der der Client jederzeit Änderungen des

betrachteten Karten-
ausschnittes an den
Server senden und der
Server jederzeit den
Client über relevan-
te, aus dem JSON-Datenstrom
ausgelesene, Schiffs-
bewegungen im be-
trachteten Kartenaus-
schnitt informieren kann.

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 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 umgebenden Schiffe übertragen werden, sondern auch Schiffsdaten (Schiffsname, MMSI-Nummer, Funkrufzeichen, etc.). AIS ist mit UKW-Signalen unabhängig von optischer Sicht und Radarwellenausbreitung.
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 Berufsschifffahrt 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 (LAT, LON, COG, SOG, UTC) erhält der AIS-Transceiver vom integrierten GPS-Empfänger, bei Klasse A auch von der Navigationsanlage des Schiffes. Die Kursrichtung (Heading = HDG) 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-Nummer
- Schiffsname
- Rufzeichen
- MMSI-Nummer

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

Dynamische Schiffsdaten

- Navigationsstatus (unter Maschine, unter Segeln, vor Anker, festgemacht, manövrierunfähig u. a.)
- Schiffsposition (LAT, LON, in WGS 84)
- Zeit der Schiffsposition (nur Sekunden)
- Kurs über Grund (COG)
- Geschwindigkeit über Grund (SOG)
- Vorausrichtung (HDG)
- Kursänderungsrate (ROT)

Reisedaten

- aktueller maximaler statischer Tiefgang in dm
- Gefahrgutklasse der Ladung (IMO)
- Reiseziel (UN/LOCODE)[5]
- geschätzte Ankunftszeit (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 Geschwindigkeit.

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 ein Schiff seine Daten aussendet

Für AIS-Daten sind 22 standardisierte Nachrichtentypen bzw. Telegramme festgelegt: In dieser Arbeit werden nur Class A Positionsmeldungen betrachtet (Typ 1-3) un.

ID	Nachrichtentyp
1	reguläre Positionsmeldung eines Klasse-A-Transceivers
4	Meldung einer Basisstation
5	reguläre Meldung von Schiffs- und Reisedaten eines Klasse-A-Transceivers
9	Positionsmeldung eines SAR-Luftfahrzeuges
12	sicherheitsbezogene Nachricht - adressiert
14	sicherheitsbezogene Nachricht - an alle
18	reguläre Positionsmeldung eines Klasse-B-Transceivers
21	Positions- und Statusmeldung eines AtoN-Transceivers

Tabelle 3.2: Die wichtigsten AIS-Telegrammtypen

Zur landseitigen AIS-Infrastruktur gehören sogenannte AIS-Basisstationen und AIS-Empfänger. Basisstationen dienen einerseits zur Erfassung des Verkehrs in dem von ihnen abgedecken Seegebiet, andererseits können diese Geräte die Übertragung von AIS-Transceivern an Bord gezielt steuern (z.B. Hochsetzen der Melderate). AIS-Empfänger sind reine AIS-Empfangsgeräte, die keine Daten senden.

3.2 Bidirektionale Kommunikation über HTML5 Websockets

In der Entwicklung der Kommunikationstechnologien im Internet galt lange Zeit das request/response Paradigma, nach dem Anfragen vom Client vom 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 initieren. Dabei handelt es sich eigentlich nur um einen aufgeschobenen response auf einen zuvor gestellten client-Request.
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. Damit sind diese Technologien für zeitkritische (realtime) Anwendungen nicht unbedingt

geeignet.
Das 2011 eingeführte Websocket-Protokoll dagegen spezifiziert eine API (HTML5-Websocket API-Spezifikation), 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 intialen HTTP-handshake aufgebaut, indem Server und Client einen Upgrade der Verbindung auf das Websocket-Protokoll aushandeln.

3.3 Node.js

Node.js ist ein Framework zur Entwicklung serverseitiger Webanwendungen in Javascript. Es wurde 2009 von Ryald Dahl veröffentlich 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 asynchron konzipiert für die Verwendung im Webbrowser, wo synchrone Verarbeitung wegen der Verzögerung des 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 Verabeitung mit der entsprechenden Callback-Methode fort.
Node.js bringt als Laufzeitumgebung die 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 HTTP-Protokoll Websockets. Mit der Unterstützung des JSON-Datenformats sind alle notwendigen Bausteine zusammen für skalierbare, echtzeitfähige Serveranwendungen. Außerdem lassen sich mit

dem Node 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 websocket (https://github.com/Worlize/WebSocket-Node) und socket.io (http://socket.io) 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 Polling .

3.4 Google Dart

Motivation für Dart

Dart ist eine von der Firma Google als OpenSource 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 Ungereimtheiten und Schwächen. Das Entwicklerteam definiert die Design-Ziele folgendermaßen: 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
- alle gängigen Browser unterstützen.

Spracheigenschaften von Dart

- Dart arbeitet **ereignisbasiert** und **asynchron** und in einem einzigen Thread ganz nach dem Vorbild von node.js.

- Dart läuft nativ in der **Dart-Virtual-machine**, kann aber auch nach Javascript kompiliert werden.
- **Klassen** sind ein wohlbekanntes Sprachkonzept zur Kapselung und Wiederverwendung von Methoden und Daten. Jede Klassen definiert implizit ein Interface.
- **Optionale Typisierung:** 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 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 ERLANG), eine Art Leightweigth Processes. Isolates greifen nicht auf einen gemeinsamen Speicherbereich zu teilen nicht denselben Prozessor-Thread. Isolates kommunizieren miteinander ausschließlich über Nachrichten (über SendPort und ReceivePort). Sie werden gesteuert von einem übergeordneten Event Loop.
- Der **DartEditor** ist eine Entwicklungsumgebung für die Entwicklung von Dart Web- und Serverapplikationen. Sie beinhaltet das **Dart SDK** und den **Dartium Browser** mit der **Dart VM**.
- 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 Package Manager npm.

Einbindung von Javascript-Bibliotheken in Dart mit js-interop

Für die Verwendung von Javascript-Bibliotheken in Dart-Code existiert die Dart-Bibliothek **js-interop**. 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 mit dart2js zu Javascript kompilierten Dart-Code.

Nachdem die Bibliothek in eine Dart-Anwendung eingebunden worden ist, kann ein sogenannter **Proxy** zum javascript-Kontext der Seite erstellt werden. Referenzen an diesen Proxy werden automatisch zu

Javascript 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-Scope zur Verfügung stehen.

Um Dart-Funktionen aus dem javascript-Scope heraus aufzurufen, wird die entsprechende Funktion in ein **Callback-Objekt** umgewandelt, das entweder ein einziges Mal oder mehrmals 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, müssen dann aber zu Vermeidung von memory leaks auch explizit wieder freigegeben werden. Dasselbe gilt für Callback-Objekte, die mehrmals aufrufbar sind.

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-Implementierung, mit der bereits einfache Websocket-Server geschrieben werden können.

3.5 NoSQL-Datenbanken

3.5.1 MongoDB

3.5.2 Redis

4 Implementierungen

4.1 Implementierungsplan

4.1.1 Lösung in Javascript

Zunächst wird eine Implementierung gewählt, die die besten Chancen hat, alle Anforderungen zu erfüllen. Diese steht im Zeitplan ganz vorne, um der Anforderung von Unternehmensseite nach einer zeitnahen Umsetzung und Auslieferung zu entsprechen. Dies ist eine Lösung in Javascript mit dem node.js-Framework und dem socket.io-Websocket. Node.js-Serveranwendungen werden schon länger mit guten Ergebnissen in Netzwerken eingesetzt, besonders für Realtime-Anwendungen und vielen gleichzeitig verbunden Clients. Das socket.io-Paket wird genutzt, weil durch die Kapselung der verschiedenen Transportmechanismen die Bedienung einer maximalen Anzahl an Browser-Clients möglich ist, ohne den Implementierungsaufwand unverhältmismäßig zu erhöhen.

4.1.2 Lösung in Google Dart

In einem zweiten Schritt wird eine vergleichbare Implementierung in Google Dart ausgeführt. Die Entwicklung von Dart befindet sich noch in der Beta-Phase. Der zweite Beta-Release fand im Dezember 2012 statt. Ein dritter Beta-Release ist angekündigt. Ein zeitnaher ausschließlicher Einsatz von Dart im Produktivsystem ist somit ausgeschlossen und diese Lösung ist als Investition in die Zukunft zu sehen. Der Vergleich beider Implementierungen (Javascript vs. Dart) ist deshalb nicht weniger interessant.

Schwerwiegende Probleme bei der Implementierung in Dart

Der ursprüngliche Plan, sowohl Server als auch Client in Dart zu schreiben, musste korrigiert werden, weil mit dem Dart-Websocket-Server einige der grundlegenden Anforderungen nicht umzusetzen waren. Zum einen unterstützt Dart keine JSON-over-TCP -Kommunikation,

wie sie für die Abfrage der Daten vom Rohdatenserver erforderlich ist. Und zum anderen gab es noch keinen Redis-Client für Dart. Der publish/subscribe Mechanismus der Redis-Datenbank wird für die Verteilung der Positionsupdates benötigt. Deshalb wird nur der Client in Dart implementiert. Damit taucht auch schon das nächste Problem auf: der socket.io-Websocket-Server entspricht nicht der HTML5-Websocket-API-Spezifikation und benötigt deshalb auf Clientseite zusätzliche Bibliotheken. Diese Bibliotheken stehen in Dart nicht zur Verfügung. Dart unterstützt Websocketverbindungen clientseitig mit dem Paket dart:html unterstützt. Darin wird ein HTML5-Websocket erwartet. Folglich muss neben dem socket.io-Server ein zweiter Server (in Javascript) implementiert werden, der eine Websocket-Verbindung nach der HTML5-Websocket-API-Spezifikation aufbaut. Dies ist relativ einfach möglich: in node.js kann hierfür das Modul websocket eingebunden werden.

Vergleichbarkeit

An dieser Stelle stellt sich die Frage, ob beide Lösungen direkt vergleichbar sind. Mögliche Unterschiede zwischen den node.js-Servern (socket.io vs. HTML5) würden in das Ergebnis des Vergleichs zwischen den Clients (Dart vs Javascript) einfliessen. Deshalb wird der Javascript-Client noch einmal mit dem HTML5-Websocket implementiert, der dann auf denselben HTML5-Websocket-Server zugreift wie der Dart-Client (siehe Tabelle). Es werden also zwei Vergleiche durchgeführt:

- In 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

Zur besseren Übersicht habe ich die Implementierungen in einer Tabelle zusammengefasst. Auf diese Weise wird die präferierte Lösung in node.js mit socket.io (S1) nicht unmittelbar sondern mittelbar über die Javascript-Lösung mit HTML5-Websocket (H1) gegen die mögliche Implementierung mit einem Google Dart Client (H2) getestet.

		HTTP-Client	
		Javascript	Google Dart
HTTP-Server	socket.io websocket-Server	socket.io (S1)	✗
	HTML-5 websocket-Server	HTML5 (H1)	HTML5(H2)

Tabelle 4.1: Übersicht über Server-und Clientimplementierungen

4.2 Implementierung des Prototypen in Javascript

In dieser ersten Implementierung werden Lösungen entwickelt für die in den Anforderungen beschriebenen Aufgaben. In der Vergleichsimplementierung werden diese Lösungen übernommen und wo das nicht möglich ist, wird eine Alternative entwickelt und dies an der jeweiligen Stelle angemerkt.

4.2.1 socket.io-Server

Die zu entwickelnde Serveranwendung hat die Aufgabe, eine JSON-over-TCP-Verbindung zum Rohdatenserver aufzubauen und die Daten über Websocket-Verbindungen an die Clients weiterzugeben. Weil Node.js singlethreaded ist (**??**) würden beide Aufgaben in einem einzigen Prozess bearbeitet. Um das Potential an Parallelverabeitung eines Dualcore oder Multicore-Servers zu nutzen, ist des daher sinnvoll, mindestens zwei Prozesse zu generieren. Dazu wurde das node.js-Modul child_process genutzt. Die ausführbare Datei master.js generiert damit zuerst einen Prozess, der den AIS-Client (ais_client.js) startet, um Daten vom Rohdaten-Server abzufragen und anschließend einen Prozess (worker.js), um einen Websocket -Server für Client-Verbindungen zur Verfügung zu stellen (4.1).

```
16 /* AIS-Client - Process*/
17 function forkAISClient() {
18   var errors;
19   try {
20     child.fork(path.join(__dirname, 'ais_client.js'));
21   }
22   catch (err) {
23     errors = true;
24     log('Error forking AIS client process: ' + err);
25     log('Exiting ...');
26     process.exit(1);
27   }
```

```
28    if (errors == null) log('Forked AIS client process');
29
30    forkWorker();
31  }
32
33  /*worker- Process*/
34  function forkWorker(){
35    var errors;
36    try
37    {
38      child.fork(path.join(__dirname, 'worker.js'));
39    }
40    catch (err) {
41      errors = true;
42      log('Error forking worker process: ' + err);
43      log('Exiting ...');
44      process.exit(1);
45    }
46    if (errors == null) log('Forked worker process');
47  }
```

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 sind alle im Bereich befindlichen Positionsdaten zu senden (vessel-in-Bounds Request)
- ein Schiff, das sich im beobachteten Kartenausschnitt eines oder mehrerer Clients befindet sendet ein Positions-Update (vessel-Position-Event)
  in diesem Fall sind alle Clients, deren Kartenausschnitt die betreffende Schiffsposition enthält, zu informieren.

Der erste Fall macht eine Zwischenspeicherung der Daten unumgänglich. 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 hier MongoDB verwendet, weil MongoDB als NoSQL-.Datenbank einen geringen Overhead, schnelle Antwortzeiten und einen geographischen Index bietet. Der Serverprozess in ais_client.js schreibt die Daten. Dabei ist die MMSI eines Schiffes eindeutig und wird als unique key verwendet (s.o.). Über die Option upsert:true wird der Mongo Datenbank mitgeteilt, dass entweder ein insert oder, falls mmsi bereits vorhanden das set upgedated werden soll ().

```
321  vesselsCollection.update(
322    { mmsi: obj.mmsi },
323    { $set: obj },
324    { safe: false, upsert: true }
```

```
325   );
```

Listing 4.2: Schreiben in die Datenbank in ais_client.js

Derselbe Prozess unterhält den Geo-Index ().

```
218 function ensureIndexes() {
219   log('(MongoDB) Ensuring indexes ... ')
220   vesselsCollection.ensureIndex({ pos: "2d", sog: 1, time_received: 1 },
          function(err, result) {
221     if (err) {
222       log(err);
223     }
224     else {
225       log('(MongoDB) Ensuring index ' + result);
226     }
227   });
228   ...}
```

Listing 4.3: Aufbau des Geo-Indexes in ais_client.js

Der Serverprozess worker.js liest die Daten für die vom Client übermittelten Geo-Daten aus().

```
220 function getVesselsInBounds(client, bounds, zoom) {
221   var timeFlex = new Date().getTime();
222   var vesselCursor = vesselsCollection.find({
223    pos: { $within: { $box: [ [bounds._southWest.lng,bounds._southWest.
          lat], [bounds._northEast.lng,bounds._northEast.lat] ] } },
224    time_received: { $gt: (new Date() - 10 * 60 * 1000) },
225    $or:[{sog: { $exists:true },sog: { $gt: zoomSpeedArray[zoom]},sog: {
          $ne: 102.3}},/*{msgid:4},*/{ $gt:{msgid: 5}}]
226   });
227   vesselCursor.toArray(function(err, vesselData)
228   {
229     var boundsString = '['+bounds._southWest.lng+','+bounds._southWest.
          lat+']['+bounds._northEast.lng+','+bounds._northEast.lat+']';
230     if (!err)
231     {
232      console.log('(Debug) Found ' + vesselData.length + ' vessels in
           bounds ' + boundsString +" with sog > "+zoomSpeedArray[zoom]);
233           client.sendUTF(JSON.stringify({ type: 'vesselsInBoundsEvent',
                vessels: vesselData}));
234     }
235    ...
236   });
237 }
```

Listing 4.4: query in worker.js

Also speichert der Server alle als JSON empfangenen Datensätze in einer MongoDatenbank und liest sie aus, sobald ein vessel-in-Bounds Request vom Client gestellt wird.

Der zweite Fall macht eine Speicherung der Clients auf Serverseite unumgänglich. Die Serveranwendung muss bei jeder Positionsmeldung wissen, welche Clients benachrichtigt werden müssen.

Der Datenaustausch zwischen beiden Prozessen funktioniert über zwei nosql-Datenbanken. In einer mongo-Datenbank werden alle vom ais_client-Prozess empfangen Positions- und Reisedaten unter der mmsi eines

Schiffes gespeichert. Dabei wird die upsert-Option von Mongo genutzt, so dass nicht vorhandene Schiffe eingefügt und vorhande aktualisiert werden.

Der worker-Prozess greift auf die Mongo-Datenbank zu, um für einen vom Client angefragten Kartenausschnitt die entsprechenden Schiffe abzufragen. Dabei wird der in Mongo zur Verfügung stehende Geo-Index auf der Schiffsposition verwendet.

Zur Verteilung der Positionsmeldungen (msgid 1,2 und3) wird außerdem eine Redis-Datenbank verwendet, die über einen publish/subscribe-Mechanismus verfügt. Über einen Kanal "vesselpos"publiziert der AIS-Client-Prozess die Positionsmeldungen und der worker-Prozess meldet sich am selben Kanal an und wird über jede Positionsmeldung benachrichtigt, die er an seine verbunden Websocket-Clients weiterreichen kann.

4.2.2 socket.io-Client

Der zugehörige Client (socket.io-Client) wird ebenfalls in Javascript implementiert und bindet die socket.io-Bibliotheken ein. Das socket.io Paket bietet Features wie die interne Clientverwaltung durch den Websocket.

4.3 Vergleichsimplementierung in Google Dart

Nachdem die Anwendung als Prototyp in Javascript fertiggestellt ist, soll eine vergleichbare Implementierung in Google Dart realisiert werden. Das paket dart:io bietet die entsprechende Unterstützung für HTML5-Websocket-Server und dart:html für HTML5-Websocket-Clients. Allerdings ergeben sich auf der Serverseite einige schwerwiegende Probleme: - zum Zeitpunkt der Umsetzung (Dezember 2012) fehlt noch ein Redis-Client in Dart, so dass für die publish/subscribe-Lösung mit Redis [siehe] eine Alternative entwickelt werden müsste, die wiederum die Vergleichbarkeit beider Implementierungen herabsetzt. - der socket.io-Server nutzt 'JSON over TCP', um die Daten vom Rohdatenserver abzufragen. 'JSON over TCP' ist in Dart (noch) nicht implementiert. Ohne die Schnittstelle zum Rohdatenserver zu verändern ist also keine vergleichbare Lösung in Dart umsetzbar.

Der Vergleich zwischen der Javascript- und der Google Dart-Anwendung ist also zu diesem Zeitpunkt lediglich auf der Clientseite möglich beziehungsweise sinnvoll.

4.3.1 HTML5-Server

Für den HTML5-Server ist es nun möglich, zwei vergleichbare HTML5-Websocket-Clientanwendungen jeweils in Javascript (js-client) und Dart (dart-client) zu bauen, die beide eine Websocketverbindung nach der HTML5-Spezifikation zum HTML5-Server aufbauen.

4.3.2 js-Client

Die Funktionalität entspricht exakt der des socket.io-Clients.

4.3.3 dart-Client

nicht unterstützt Zum Schluß wird der Client für den HTML5-Server in Dart geschrieben.

5 Vergleichende Evaluation

Die realisierten Implementierungen lassen zwei Vergleiche zu:

- Node.js-server mit socket.io-Websocket-Server vs. node.js-Server mit HTML5-Websocket-Server, wobei die Javascript-Clients sich nur marginal unterscheiden.
- Javascript-Client vs. Dart-Client, wobei beide auf denselben node.js-Server mit HTML5-Websocket-Server zugreifen

5.1 Socket.io-Websocket vs. HTML5-Websocket

5.1.1 Implementierungsaufwand

Anzahl zeilen code

5.1.2 Latenzzeit

querytime

time received

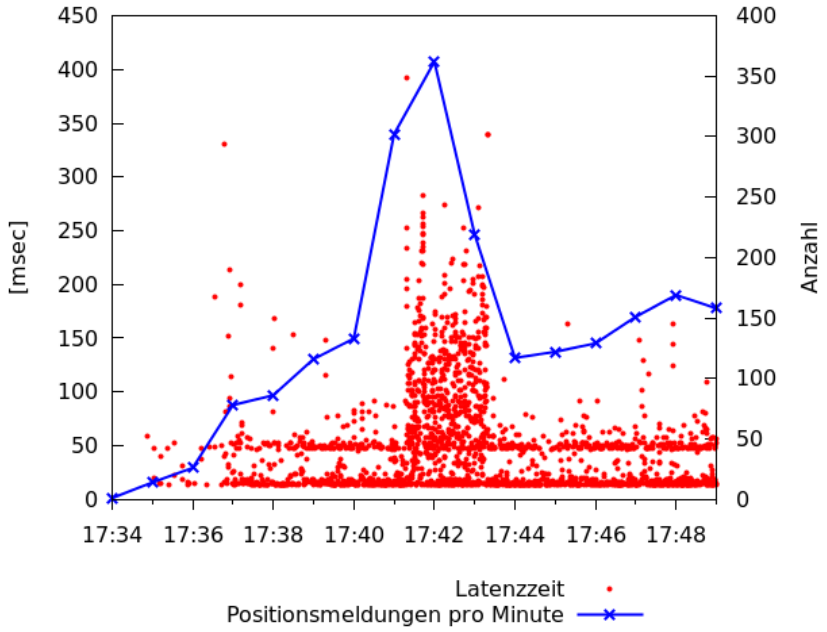

Abbildung 5.1: socket.io-Websocket-Server: Latenzzeit der Positionsmeldungen und Anzahl empfangener Schiffe

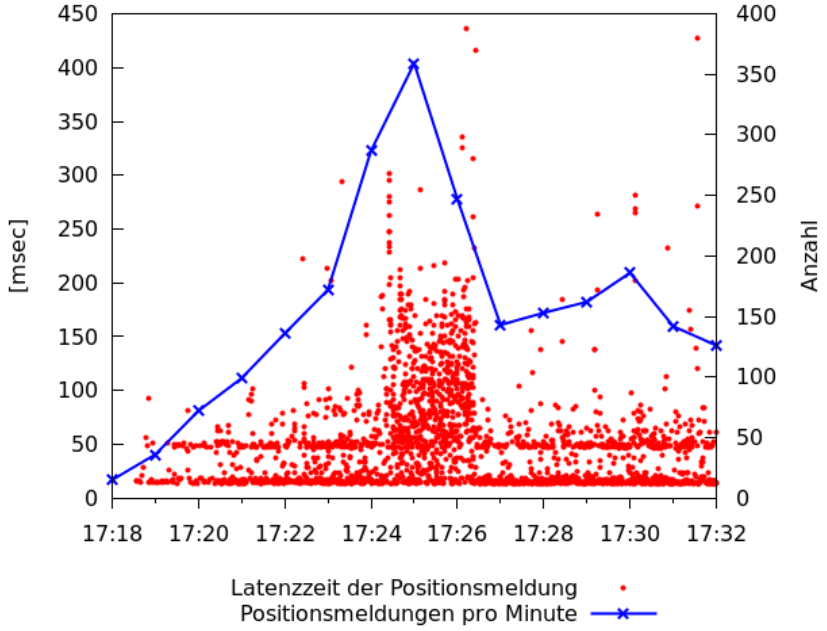

Abbildung 5.2: HTML5-Websocket-Server: Latenzzeit der Positionsmeldungen und Anzahl empfangener Schiffe

5.1.3 Performance

paintToMap

5.1.4 Browserunterstützung

Firefox, Chrome, IE, Safari

5.2 Javascript-Client vs. Dart-Client

5.2.1 Implementierungsaufwand

js-Client

Zeilen Code

5.2.2 Latenzzeit

queryTime

5.2.3 Performance

paintToMap

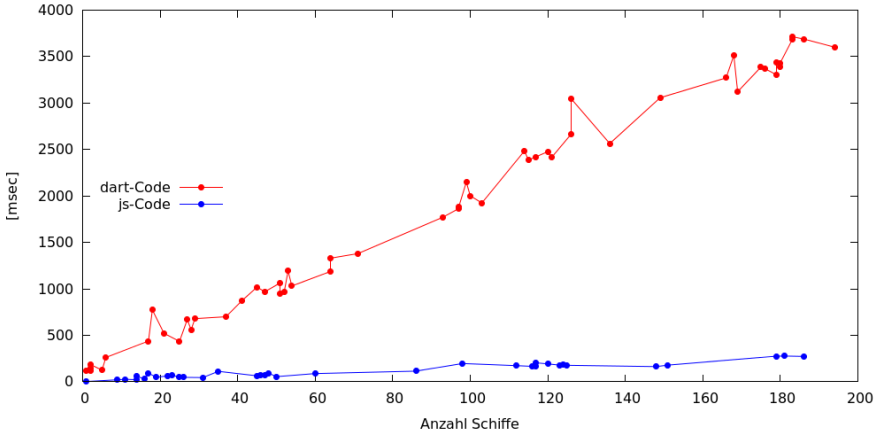

Abbildung 5.3: Dauer des Renders in Dartium

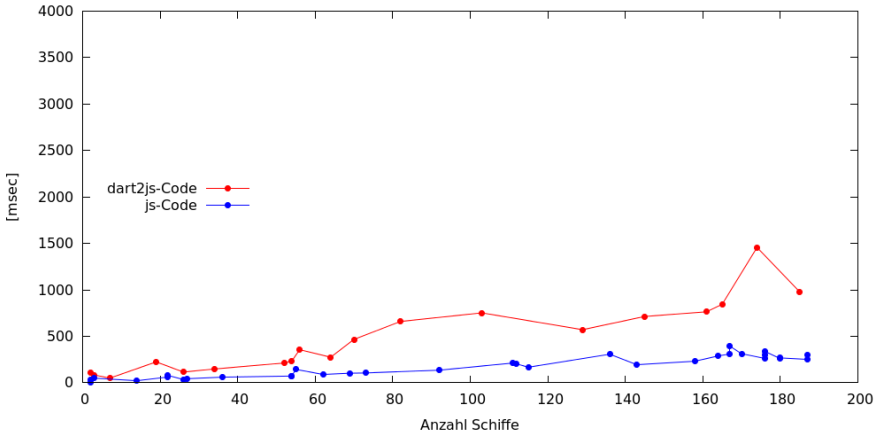

Abbildung 5.4: Dauer des Renders in Chrome

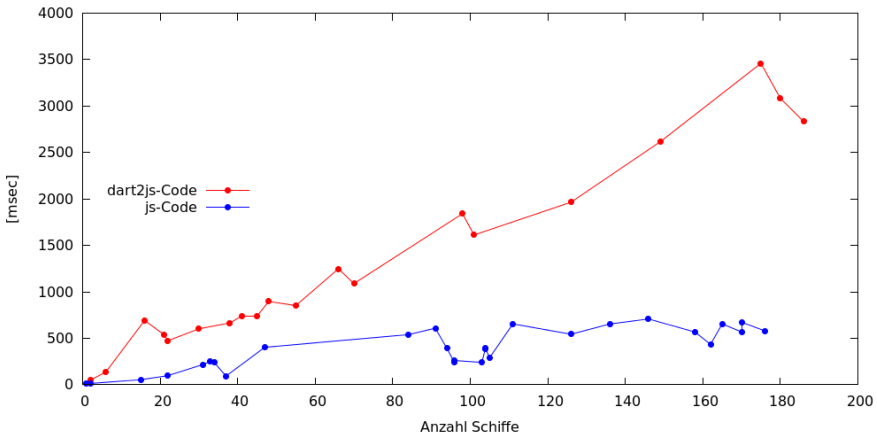

Abbildung 5.5: Dauer des Renders in Firefox

5.2.4 Browserunterstützung

Dartium

Firefox, Chrome, IE, Safari

Der dart-Client kompiliert den in Dart geschriebenen Code zu Javascript.

Dabei traten Fehler auf, die unter Dartium (also im originalen Dart-Code) nicht auftraten. 1. Wird innerhalb des Javascript-Scopes eine Methode auf einen javascript-Proxy (hier _map) aufgerufen und ein proxy wird zurückgegeben, dann ist es nicht möglich auf diesen Proxy, der in diesem Fall vom Typ LatLngBounds sein müsste, eine Methode der Klasse LatLngBounds aufzurufen. => TypeError: t1.get$_map(...).getBounds$0(...).getSou is not a function

dart-client: web/leaflet_maps.dart

List getBounds() var south, west, north, east; js.scoped(() south= _map.getBounds().getSo west = _map.getBounds().getSouthWest().lat; north = _map.getBounds().getNorthEast().l east = _map.getBounds().getNorthEast().lat;); return [west, south, east, north];

In diesem Fall wird einfach als work-Around eine andere Methode verwendet (getBBoxString), die einen String mit den Bounds zurückgibt. Aus den Teilen dieses Strings werden mit der Methode parse(string) der Klasse double die Werte der Eckpunkte der Bounds generiert.

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

Weil dadurch der message-Parameter 'bounds' kein number-Array, sondern ein String ist, muss im html5-Server der String einmal zum Float geparst werden.

2. Ein Feld (ÏMO") wird auf null und auf > 0 geprüft.

6 Fazit

6.1 Ergebnisse

6.2 Ausblick

-Satellitendaten in die Anwendung einbinden

Literaturverzeichnis

[1] Seth Ladd, SSorry, at the time of this writing, I'm not aware of a socket.io port for Dart. socket.io is nice because it has a bunch of implementation options for browsers that don't support Web sockets. Sounds like a good idea for a hackathon project!",2012 Oct 15, http://stackoverflow.com/questions/12882112/is-there-a-socket-io-port-to-dart.

[2]

[3] http://nbn-resolving.de/urn:nbn:de:swb:14-1114955960020-08344

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