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	Prof. Mustera

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation für diese Arbeit

Beschreibung der Aufgabenstellung Die Vesseltracker.com GmbH ist ein Schiffsmonitoring und -reporting-Dienstleister. Die Daten resultieren aus den AIS (Automatic Identification System) – Meldungen, die von allen Schiffen über Funk regelmäßig zu senden sind. Das Unternehmen unterhält ein Netzwerk von ca. 800 terrestrischen AIS-Antennen, mit denen diese 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.

Die Kernanwendung des Unternehmens, eine Webanwendung, speichert die AIS-Daten in einer Geodatenbank, kombiniert sie mit weiteren Datenquellen (Schiffs-Stammdaten, Satelliten-AIS-Daten) und stellt sie angemeldeten Nutzern über das Unternehmensportal (www.vesseltracker.com) zur Verfügung. Damit können Kunden sich über alle weltweit fahrenden Schiffe detailliert informieren. Zusätzlich können sie sich über eine geographische Visualisierung der Schiffspositionen auf Openstreetmap-Karten einen Überblick über Schiffspositionen verschaffen. Diese Karten zeigen alle Schiffe an, die sich in dem frei wählbaren Kartenausschnitt zu der Zeit befinden. Zu einzelnen Schiffen kann sich der Kunde die gefahrene Route der letzten 24 h anzeigen lassen. Die Positionsinformationen werden jeweils initial bei Änderung des betrachteten Bereichs abgerufen und anschließend minütlich aktualisiert.

In letzter Zeit gewinnen real-time-Anwendungen zunehmend an Bedeutung und ihre Verbreitung wird durch den Fortschritt der verfügbaren Webtechnologien auf breiter Basis unterstützt. Da die empfangenen AIS-Daten aufgrund ihrer hohen Aktualisierungsrate für eine real-time-Darstellung prädestiniert sind, entstand die Idee, eine real-time Darstellung der Schiffsbewegungen im Browser zu ermöglichen. Damit soll es dem Kunden möglich

sein, Schiffsmanöver wie Anlegen, Festmachen, Ablegen, Schleppen, Lotsen, Betanken oder Schleusendurchfahrten sozusagen 'live' mitzuverfolgen.

1.2 Aufbau der Arbeit

2 Beschreibung der Anwendung: Schiffsverfolgung per AIS-Daten-Strom

2.1 AIS-Nachrichtentypen und ihre Verarbeitung

2.2 Visualisierung von geographischen Bewegungsdaten im Webbrowser

2.3 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 vesseltracker repository gehalten
- als Kartenmaterial sind die von vesseltracker gehosteten OpenstreetMap-Karten zu verwenden
- verwendete Software-Module sollten frei zugänglich sein (open source)

2.3.1 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

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 Kartenausschnittes an den Server senden und der Server jederzeit den Client über relevante, aus dem JSON-Datenstrom ausgelesene, Schiffsbewegungen im betrachteten Kartenausschnitt informieren kann.

Bidirektionale Client-Server-Verbindungen über HTTP existieren schon seit längerem (HTTP Long Polling, HTTP Streaming, Ajax on demand). Diese Verbindungen teilen jedoch alle den Nachteil, dass sie das HTTP-Protokoll nutzen und so bei jeder Nachricht einen Overhead an Informationen mitsenden, was zu hohen Latenzzeiten führt. Dieses Problem ist mit der Einführung des Websocket-Protokolls 2011 überwunden worden, weil Websockets nach einem initialen Handshake in HTTP einen Upgrade der Verbindung auf das Websocket-Protokoll aushandeln. Damit sind Websockets eine vielversprechende Technologie für datenintensive Realtime-Anwendungen mit vielen simultanen Client-Verbindungen, weshalb sie in der geplanten Anwendung zum Einsatz kommen sollen.

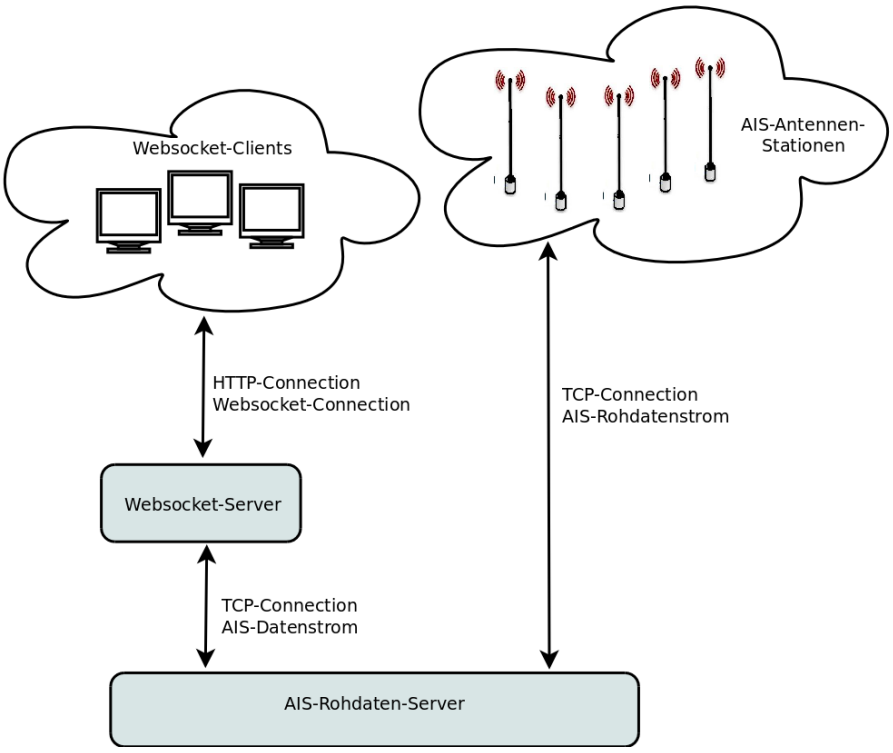

Abbildung 2.1: Entwurf der Realtime-Anwendung

3 Grundlagen

3.1 Bidirektionale Kommunikation über Websockets

3.1.1 HTML5-Websocket API-Spezifikation

Um über die Einschränkungen des request/response Musters hinwegzukommen, bei dem Anfragen des Client vom Server beantwortet werden, sind kontinuierlich Fortschritte in Richtung einer bidirektionialen Verbindung zwischen Client und Server gemacht worden. Mit HTTP Long Polling, HTTP Streaming und Ajax on demand ist es für den Server möglich, beim 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. Das Problem dieser Technologien liegt aber darin, dass sie, weil sie http-Nachrichten austauschen, einen großen Überhang an Header-Informationen mitsenden müssen, der sich in Summe negativ auf die Verzögerung auswirkt. Damit sind sie für zeitkritische (realtime) Anwendungen nicht geeignet.

Das 2011 eingeführte Websocket-Protokoll dagegen beschreibt eine API, die eine echte 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.

3.1.2 Vorstellung verschiedener Implementierungen von Websockets

Nodejs-Websockets

Dart-Websockets mit Dart:IO

Die Gleichung

$$a^2 + b^2 = c^2 \tag{3.1}$$

ist allseits bekannt und bedarf wohl keiner weiteren Erläuterung.

3.2 Google Dart als moderne Programmiersprache für Webanwendungen

Auch nicht schlecht ist **??**. Aber überhaupt keinen Sinn macht **??**. Hieran sieht man den Vorteil des autoref-Befehls und das so Links erstellt werden.

3.2.1 Motivation für Dart

Formen	Städte
Quadrat	Bunkenstedt
Dreieck	Laggenbeck
Kreis	Peine
Raute	Wakaluba

Tabelle 3.1: eine sinnlose Tabelle

3.2.2 Eigenschaften und Besonderheiten der Programmiersprache Dart

3.2.3 Dart Tools (DartEditor, dart2js-compiler)

3.2.4 Einbindung von Javascript-Bibliotheken in Dart mit js-interop

Komplexe Tabellen sind nicht sehr einfach:

		dies			
		von dort	und dort	über hier	zu Los
das	hier	bla	bla	bla	bla
	dort	bla	bla	bla	bla
	da	bla	bla	bla	bla

Tabelle 3.2: eine kompliziertere Tabelle mit viel Beschreibungstext, der aber nicht im Tabellenverzeichnis auftauschen soll

4 Vergleichende Evaluation

4.1 Implementierungsaufwand

4.1.1 Socket.io-Server und -Client

4.1.2 HTML5-Server und Javascript-Client

4.1.3 HTML5-Server und Dart-Client

4.2 Kommunikationsaufwand

4.2.1 Socket.io-Server und -Client

4.2.2 HTML5-Server und Javascript-Client

4.2.3 HTML5-Server und Dart-Client

4.3 Performanz und Robustheit im Praxistest

4.3.1 Socket.io-Server und -Client

4.3.2 HTML5-Server und Javascript-Client

4.3.3 HTML5-Server und Dart-Client

4.4 Browserunterstützung

4.4.1 Socket.io-Server und -Client

4.4.2 HTML5-Server und Javascript-Client

5 Fazit

5.1 Ergebnisse

Allgemeine Symbole

Symbol	Bedeutung
a	der Skalar a
$\vec{x}$	der Vektor $\vec{x}$
$\mathbf{A}$	die Matrix $\mathbf{A}$

5.2 Ausblick

A Anhang

A.1 Quelltexte

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