

# Backlog – Verteilte Systeme

## Inhalte:

Anforderungsanalyse

Systemdesign

Virtuelles Model - Simulation

Tests

Projektplan

Anleitung -Anwendungen starten

Aufbau der einzelnen Anwendungen

## Anforderungsanalyse

### Aufgabe 1:

#### 1. Funktionale Anforderungen

- Sensoren simulieren fahrende Autos auf einem schachbrettartigen 5x5-Feld
  - startende Autos beginnen auf Feld 1 (rechts oben)
  - Position des Autos wird berechnet durch alte Position, Speed, Direction
    - Daten hierfür senden die jeweiligen Sensoren an den position-Sensor
- 4 Sensoren (Position, Speed, Direction, Roadconditions)
  - generieren Daten
  - erfassen jeweils eine Gegebenheit
  - jeder Sensor arbeitet als eigenständiger Prozess
  - senden per UDP-Daten an den Zentralserver
    - Datenformat für das UDP Datagramm liegt als String vor
  - Alle Sensoren werden über eine Anwendung gestartet (außer der Sensor Roadconditions, dieser ist separat über einen anderen Übergabeparameter ansteuerbar, bzw. zu starten)
- 1 Zentralserver für die Sensoren
  - empfängt UDP-Daten von den einzelnen Sensoren
    - bei Empfang:
      - Ausgabe der Daten, Port, IP und Typ des Sensors
      - speichert Daten in ein Array des jeweiligen Sensortyps
  - nutzt Threads
  - verfügt über einen HTTP-Server
    - verarbeitet HTTP GET Anfragen von unterschiedlichen Browsern
      - sendet je nach Anfrage Daten der Sensoren an den jeweiligen Browser als HTML-Format
    - Zugriffe auf einzelne Sensordaten über REST-API
      - mittels:
        - „/Speed“, „/Position“, usw. (Datenabfrage aller Daten des jeweiligen Sensors)
        - „/SpeedNow“, „/Position“, usw. (Datenabfrage der aktuellen Daten des jeweiligen Sensors)
        - „/All“ (Abfrage aller Daten von allen Sensoren)

- Ist eine GET-Anfrage unbekannt, wird dem jeweiligen Browser ein HTML-Format mit dem Inhalt 404 übermittelt
  - Starten / Ausführen der einzelnen Anwendungen
    - Werden mittels Skript (bat-Datei) ausgeführt
    - 4 Sensoren starten & 1 Zentralserver starten
    - Ports sind konfigurierbar per Parameterübergabe beim Programmstart
    - HTTP-Server kann gleichzeitig mit unterschiedlichen Browsern arbeiten
      - Dokumentiert, welcher Browser eine GET-Anfrage geschickt hat
- 2. Nicht-funktionale Anforderungen**
- Generierte Daten der Sensoren sind sinnvoll
  - Implementierter HTTP-Server benötigt keine fremden Bibliotheken oder weitere Hilfsmittel
  - Der Zentralserver bedient die 4 Sensoren und die Browser gleichzeitig

#### Aufgabe 2:

- 1. Funktionale Anforderungen**
- Die Zentralserver übermitteln ihre Daten mittels RPC (Thrift) an einen Server des Autoherstellers
  - Client und Server verfügen über eine Thriftimplementierung
  - Die Daten werden in einer Textdatei gespeichert
  - Es wird 1 Autoherstellerserver gestartet
  - Der Autoherstellerserver werden und Skript ausgeführt
- 2. Nicht-funktionale Anforderungen**
- Die Daten sind Persistent
  - Es gehen keine Daten beim Senden verloren.

#### Aufgabe 3:

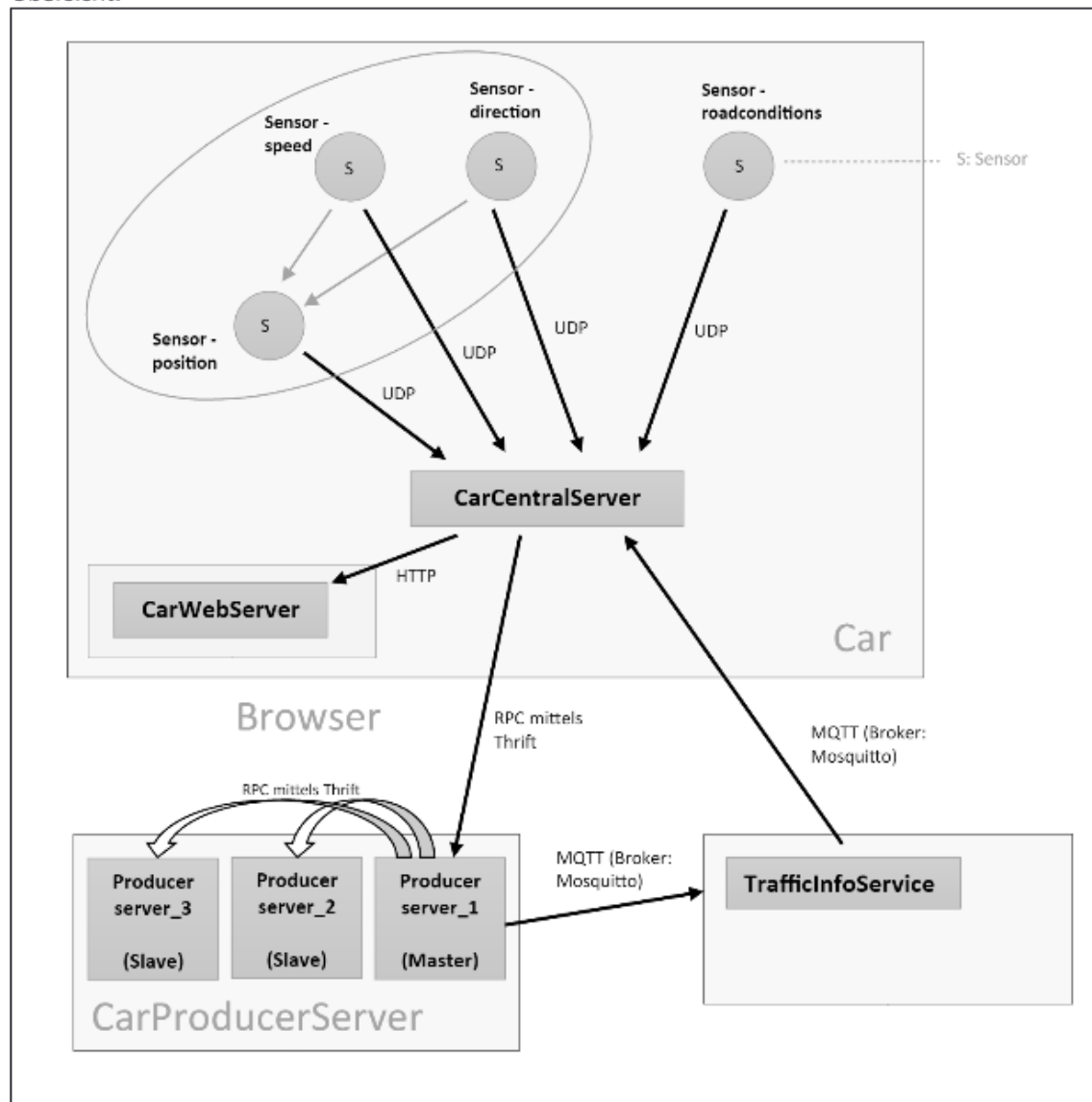
- 1. Funktionale Anforderungen**
- Die Daten des Herstellerservers werden auf weitere Server übertragen
  - Die Herstellerserver synchronisieren ihre Daten mittels eines RPCs (Thrift)
  - Ein Verkehrsinformationsserver erhält die Daten der Herstellerserver per Thrift
  - Der Verkehrsinformationsserver hat eine Thriftimplementierung
  - Der Verkehrsinformationsserver stellt die Daten textuell da
  - Es kommt zu simulierten Ausfällen der Herstellerserver
- 2. Nicht-funktionale Anforderungen**
- Alle Herstellerserver haben die selben Daten
  - Alle Daten sind in der selben Reihenfolge

#### Aufgabe 4:

- 1. Funktionale Anforderungen**
- Datenübertragung vom Herstellerserver zum Verkehrsinformationsserver erfolgt nun per MQTT (Mosquitto)
  - Der Verkehrsinformationsserver überträgt die Verkehrslage per MQTT (Mosquitto) an die Zentralserver der Autos
  - Die Verkehrslage wird anhand der erhaltenen Daten im Verkehrsinformationsserver berechnet
- 2. Nicht-funktionale Anforderungen**
- Kein Datenverlust
  - Nur die richtigen Daten an ein Auto senden

# Systemdesign

Übersicht:



## Aufgabe 1:

- Ein Auto besteht aus 4 Sensoren und 1 Zentralserver
- Sensoren senden mittels UDP einen Datenstring an den Zentralserver
- Ein Datenstring besteht aus: Sensortyp + Sensorname + Daten + Timestamp
- Die Sensoren bekommen mitgeteilt welchen Port sie ansprechen sollen
- Der Zentralserver muss den entsprechenden Port verwenden
- Zentralserver verfügen über HTTP-Server zum Anzeigen der Daten
- Die Daten sind nach Typen über festgelegte URLs aufrufbar.
- Die Zentralserver verfügen über mehrere Threads. Ein Thread für das UDP Handling und n Threads für die TCP Clients

## Aufgabe 2:

- n Autos
- 1 Herstellerserver

- Herstellerserver wird über den Port 9898 erreicht
- Daten werden mit Thrift von den Zentralservern der Autos an den Herstellerserver gesendet
- Die Daten werden mit der Id des Autos erweitert. Port = Id
- Die Daten werden in einer Liste über eine Thriftfunktion verschickt
- Die Liste enthält beim versenden bis zu 10 Datenstrings
- Speichern der Daten persistent in einer .txt Datei auf dem Herstellerserver
- Wenn Daten nicht an den Herstellerserver übermittelt werden konnten, werden sie erneut gesendet

#### **Aufgabe 3:**

- 3 Herstellerserver
- 1 Verkehrsinformationsserver
- Die Herstellerserver arbeiten im Masterslaveverfahren
- Server 1 mit dem Port 9898 ist der Master
- Der Master sendet seine Daten an die Slaves weiter
- Die Server mit den Ports 9899 und 9900 sind die 2 Slaves
- Der Datenaustausch erfolgt über Thrift
- Die Slaves verfügen jeweils über eine eigene .txt Datei
- Die Liste die über Thrift verschickt wird verfügt noch über einen weiteren Eintrag welcher den Filename der .txt Datei angibt
- Wenn ein Slave ausfällt merkt sich der Master welche Daten noch nicht an diesen Slave gesendet wurden und übermittelt sie erneut
- Wenn der Master ausfällt gibt es keine Datenweitergabe
- Wenn der Master wieder startet bekommt der die Daten von den Autos nachgeliefert und sendet wieder an die Slaves und den Verkehrsinformationsserver
- Der Verkehrsinformationsserver erhält vom Master die Positionsdaten der Autos
- Die relevanten Daten (ID, Position, Timestamp) werden gespeichert in ein Car Objekt
- Die Positionsdaten der Cars werden verglichen und der Verkehrszustand wird berechnet
- Zustände sind : Alles Ok, Auto in der Nähe, Unfall


#### **Aufgabe 4:**

- 3 Herstellerserver
- 1 Verkehrsinformationsserver
- n Autos
- Herstellerserver senden mittels MQTT an einen Verkehrsinformationsserver statt Thrift
- Herstellerserver ist der Publisher, der Verkehrsinformationsserver der Client
- Die Positionsdaten sind in einem String gespeichert, der ausgelesen wird
- Die relevanten Daten (ID, Position, Timestamp) werden gespeichert in ein Car Objekt
- Zentralserver der Autos bekommen die Daten mittels MQTT vom Verkehrsinformationsserver
- Verkehrsinformationsserver ist der Publisher, die Autos die Clients
- Jeder Client hat einen eigenen Container beim Broker
- Jedes Auto kann auf dem Webserver seinen Verkehrszustand anzeigen.

## **Simulation**

**Virtuelles Modell – Auto fährt auf 5x5-Feld**

**Start**

1 	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

### Beschreibung:

Jedes neu erzeugte Auto startet auf Feld 1 (position 1) und hat 4 Sensoren (für position, speed, direction, roadconditions), sowie einen Zentralrechner (CarCentralServer).

Die aktuelle Position wird anhand der Werte des speed- und des direction-Sensors und der alten Position berechnet. Fährt ein Auto in Richtung des Randes des 5x5-Feldes, kollidiert dieser nicht, sondern erscheint auf der gegenüberliegenden Seite des Feldes. So erscheint ein Auto auf Position 5, welcher zuvor von Position 1 nach Westen fahren wollte.

## Tests

(ausführliche Dokumentation der Tests im anderen PDF! (TestDoku.pdf))

### **Aufgabe 1:**

- Werden unzulässige Daten der Sensoren abfangen?
- Werden falsche URLs abgefangen?
- Werden die Browser der Clients alle erfasst?
- Gibt es Paketverlust? (Performancetest)

### **Aufgabe 2:**

- Herstellerserver empfängt die Daten per Thrift ?
- Werden die Daten gespeichert ?
- Können mehrere Autos an den Herstellerserver senden ?
- Fällt Thrift (ohne Senderegulierung) unter Belastung(schnelle/häufige Aufrufe) aus ? (Performancetest)

### **Aufgabe 3:**

- Empfangen die Slaves die Daten des Masters ?
- Werden die Daten werden gespeichert ?
- Gehen die Daten verloren wenn sie nicht empfangen werden können ?
- Der Master schickt immer größer werdende Datenpakete per Thrift an die Slaves (Performancetest)

### **Aufgabe 4:**

- Werden die Daten vom Master beim Verkehrsinformationsserver per MQTT empfangen ?
- Wird die Verkehrslage im Webbrowser des Autos angezeigt ?
- Erhält jedes Auto nur die für das Auto bestimmten Daten ?
- Der Master schickt immer größer werdende Datenpakete per MQTT an den Verkehrsinformationsserver (Performancetest)

# **Projektplan**

## **Aufgabe 1:**

1. Sensoren implementieren
  - Daten generieren
  - UDP Socket implementieren
  - Messwerte als String versenden
2. Zentralserver implementieren
  - UDP Socket implementieren
  - Daten empfangen und in Listen speichern
  - TCP Socket (Serversocket) implementieren
  - HTTP GET Verarbeitung implementieren
  - Multithreading implementieren für HTTP Server, 1 Thread pro Client
  - REST API implementieren
  - HTML Ausgabe implementieren
3. Startskript erstellen
4. Tests erstellen

## **Aufgabe 2:**

1. Thrift implementieren
  - Thrift compiler installieren
  - Code aus .Thrift Datei generieren
  - Generierte Klassen in Projekt einbinden
2. Zentralserver erweitern
  - Thriftclient implementieren
  - Datenpacketerstellung implementieren
  - Datensendung implementieren
3. Herstellerserver
  - Thriftserver implementieren
  - Datenempfang implementieren
  - Speicherung der Daten in Textfile implementieren
4. Startskript erstellen
5. Tests erstellen

## **Aufgabe 3:**

1. Verkehrsinformationsserver
  - Daten per Thrift empfangen implementieren
  - Datendarstellung implementieren
2. Herstellerserver nachbessern
  - Master Slave Verfahren implementieren
    - Kommunikation zwischen den Servern per Thrift implementieren
    - Neusenden bei Ausfall implementieren
  - Ausfallsimulation implementieren
  - Senden per Thrift an Verkehrsinformationsserver implementieren
3. Startskript erstellen
4. Tests erstellen

## **Aufgabe 4:**

1. Herstellerserver Master
  - MQTT implementieren
    - Senden an Verkehrsinformationsserver von Thrift auf MQTT umstellen
  -
2. Verkehrsinformationsserver



- Thrift durch MQTT ersetzen
    - Senden
    - Empfangen
  - Positionsdaten
    - Auslesen
    - Verkehrssituation berechnen
    - Wichtige Zustände an betroffene Autos senden
3. Zentralserver der Autos
- MQTT implementieren
    - Empfangen
  - Webanzeige der Verkehrssituation implementieren

#### 4. Tests erstellen

## Anleitung – Anwendungen starten

Anwendungen, wie z.B. für die Sensoren geschieht über die bat-Datei, welche sich im src befindet(startSensorsInPowerShell.bat).

Diese Datei führt Befehle aus, zur Kompilierung der java-Dateien, bis hin zur Erstellung einer ausführbaren Anwendung.

## Systemstart

### **1 Auto & seine Anwendungen:**

- **2 mal** Aufruf **MainOfSensors** mit Parameterübergabe(**1 oder 0, port**)
- **1 mal** Aufruf **CarCentralserver** mit Parameterübergabe(**port**(identisch zu MainOfSensors))

#### Info:

- Jedes Auto nutzt je einen anderen port
- Identifizierung eines Autos über die CarID (entspricht dem port)

### **4 Sensoren:**

Parameterübergabe bei Anwendungsstart: **0** oder **1** zur Unterscheidung zwischen Gruppe 1 und 2 (Gruppe 1: direction, speed, position; Gruppe 2: roadconditions) + **port**

Übertragung an CarCentralServer: ein Datenstring mittels UDP alle 2sec

#### Infos:

- Car-Identifizierung über port(identisch zu CarCentralserver)

### **CarCentralServer:**

Parameterübergabe bei Anwendungsstart: **port**

Datenempfang von Sensoren: History-Speicherung in Arrays je nach Sensortyp & Abspeicherung in Puffer für ProducerServer\_1

Übertragung an ProducerServer\_1: bis zu max. 10 Daten-Strings(Packetgröße) mittels Thrift

(Daten-String: index0=Identifizierung des Servers zum Abspeichern; index1=CarID; index > 1=Daten)

- Thriftkommunikation aller Anwendungen läuft über einen Handler, da Nutzung gleicher Methode, sprich write()
- Jedem Daten-String wird die CarID angehängt

#### **Webserver:**

- Ausgabe der Sensor-Daten je nach Angabe eine URL
  - Sensoren-Daten(alle, aktuellster)
  - Aktuelle Verkehrslage(Angabe bei Unfall oder Gefahr einer Kollision)

#### **ProducerServer\_1(Master):**

Datenempfang von CarCentralServer: Speicherung von bis zu max. 10 Daten-Strings(Packetgröße) in txt-Datei & Abspeicherung 2 Puffer für ProducerServer\_2 und ProducerServer\_3

- Geschieht im TransportHandler

Übertragung an TrafficInfoService: Übertragung von aktuellem Positionswert eines Autos mittels MQTT

Infos: Speicherung aller Car-Daten & Weiterleitung der Daten an die Slaves

#### **ProducerServer\_2 & ProducerServer\_3(Slaves):**

Datenempfang von ProducerServer\_1(Master): Speicherung von bis zu max. 10 Daten-Strings(Packetgröße) in txt-Datei

#### Infos:

- Bei Masterausfall werden keine Daten mehr empfangen
  - Ist dieser wieder online, werden die fehlenden Daten übertragen zur Gewährleistung der Redundanz
- Slaves-Server dienen der redundanten Speicherung der Daten

#### **TrafficInfoService:**

Datenempfang von ProducerServer\_1: Daten enthalten Position eines Autos (aktueller Wert)

Übertragung an CarCentralServer(Webserver): Übertragung der aktuellen Verkehrslage an alle aktiven Autos mittels MQTT

#### Infos:

- Datenverarbeitung zur Erstellung aktiver Cars zur Angabe dessen Position
  - Nur aktuelle Werte pro Auto werden abgespeichert