DSE22Gruppe10-Designdokument

Diego Krupitza, Jan Müller, Kian Pouresmaeil

22. Mai 2022

Inhaltsverzeichnis

1 Software Architektur

1.1 Services & Komponenten

Insgesamt gibt es 4 Microservices und ein Gateway. In den folgenden Unterkapiteln werden die Microservices und das Gateway genauer beschrieben.

1.1.1 Gateway

Die Aufgabe des Gateways ist es Eingehende Anfrage außerhalb des Clusters basierend aufgrund von Routing regeln weiterzuleiten. Darüber hinaus findet sich am Gateway auch die Zentrale Swagger UI Dokumentationsplattform auf der die Endbenutzer die Exposed Rest Schnittstellen der Microservices ansehen können (bzw. die Dokumentation lesen können).

1.1.2 Entity-Service

Die Aufgabe des Entity-Services ist es die Metadatan und statischen Daten der Entitäten der Akteuren (Autos und Ampeln) zu verwalten. Die Akteure registrieren sich bei diesem Service um teil der Simulation zu werden. Über verschiedene Rest Schnittellen kann können diese Daten dann abgefragt werden.

1.1.3 Tracking-Service

Die Aufgabe des Tracking-Services ist es die dynamischen Daten der Entitäten der Akteuren (Autos und Ampeln) zu verwalten. Während der Simulation werden diese Daten an das Tracking-Service geliefert und dieses Persistiert diese. Über verschiedene Rest Schnittellen kann können diese Daten dann abgefragt werden.

1.1.4 Simulator-Service

Die Zentrale Aufgabe des Simulator-Services ist es die Simulation zu starten und zu beenden. Sobald die Simulation gestartet wird, simuliert dieses Service Ampeln und Fahrzeuge. Es registriert diese Akteure und spielt das Verhalten, welches im Algorithmus spezifiziert wurde, ab.

1.1.5 Flowcontrol-Service

Das Flowcontrol-Service hat zur Aufgabe anhand von dem aktuellen Stand eines Fahrzeuges die optimale Geschwindigkeit zu berechnen, sodass eine

grüne Phase erreicht wird.

1.2 Schnittstellen & Kommunikation der Services

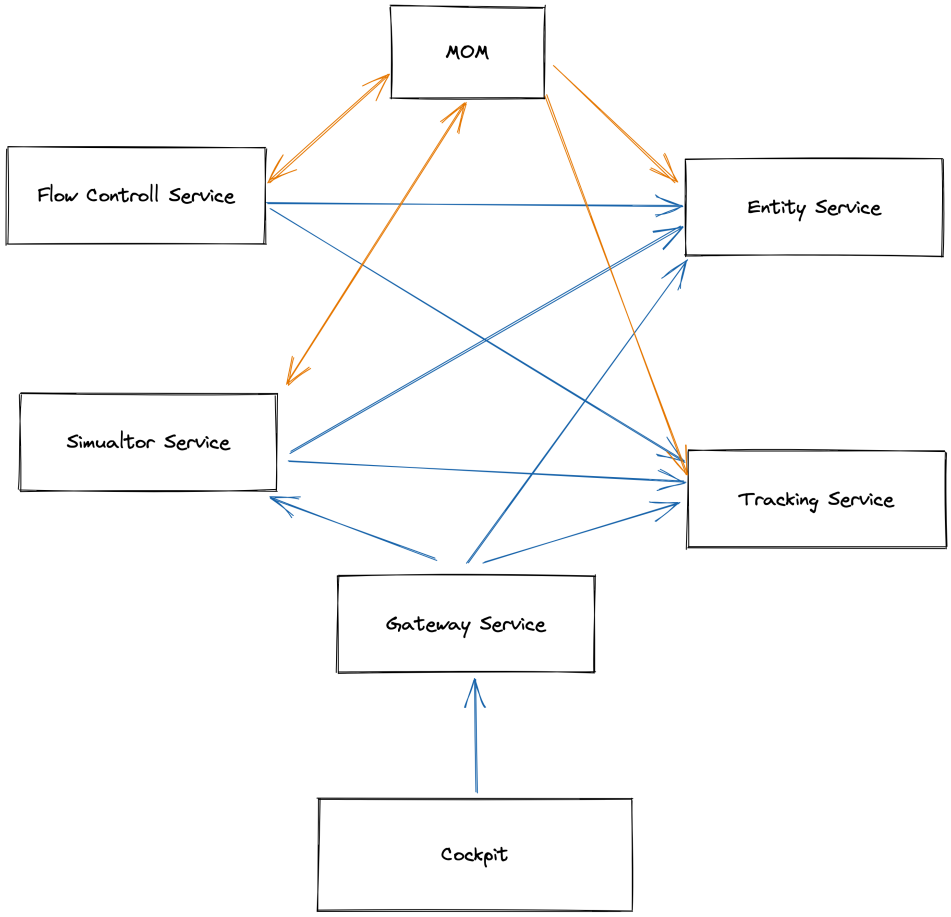

Abbildung 1: Übersicht aller Schnittstellen und Kommunikationen

Wie man der Abbildung 1 entnehmen kann, finden sich innerhalb unseres Systems 2 Arten von Schnittstellen. Zuerst gibt es die konventionelle Synchrone Schnittstelle nämlich REST (in Abbildung 1 mittels blauen Pfeilen dargestellt) und weiteres die Asynchrone Schnittstelle (in Abbildung 1 mittels orangen Pfeilen dargestellt), welche mittels einer Message Oriented Middleware (in diesem Falle RabbitMQ) realisiert wurde. Grundsätzlich verwenden wir asynchrone Kommunication damit Kommunikation, welche häu-

fig erfolgt, nicht die Services blockiert und synchrone für Operationen, die nicht Zeitkritisch sind bzw. asynchron nicht viel Sinn machen würde.

1.2.1 REST

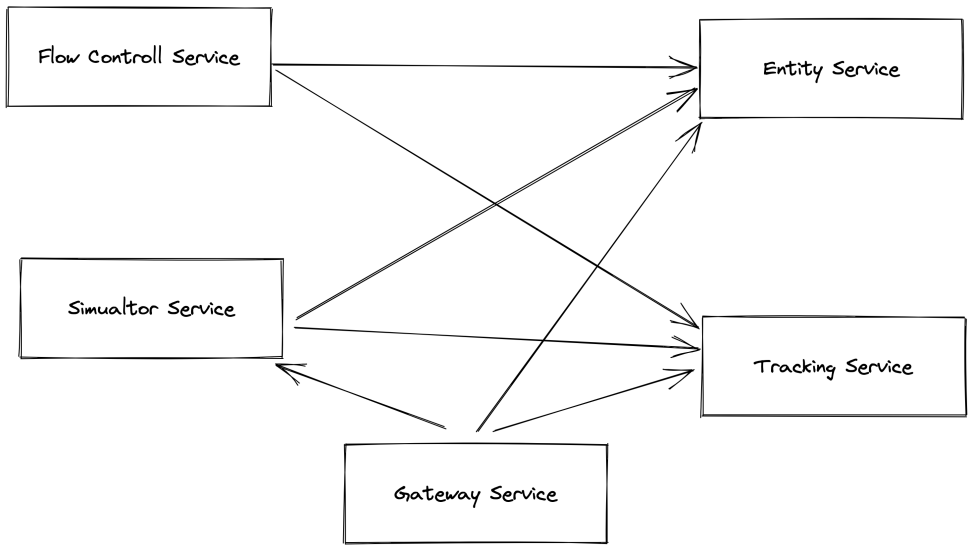

Abbildung 2: Überblick der Rest Kommunikation

In der Abbildung 2 kann man einen Überblick der REST Kommunikation der Services sehen.

Gateway Auf dem ersten Blick kann man sofort erkennen, dass das Gateway aufgrund seiner Zielfunktionalität mit allen Services kommuniziert, welche eine REST Schnittstelle besitzen und diese auch nach außen zugänglich machen wollen.

Flowcontrol-Service Wichtig ist hier hervorzuheben, dass das Flowcontrol-Service keine REST Schnittstelle bereitstellt für andere Services. Dieses Service kommuniziert mit dem *Entity-Service* und dem *Tracking-Service* damit alle Eingaben für den Algorithmus zur Berechnung der Optimal Geschwindigkeit angefragt werden können.

Simulator-Service Das Simulator-Service stellt insgesamt 3 REST Endpunkte zur verfügung, die für das starten/stoppen der Simulation gebraucht

werden. Das Simulator-Service kommuniziert mit dem *Entity-Service* und dem *Tracking-Service* um einen Stoppen bzw. Reset der Simulation zu erwirken.

Entity-Service Das Entity-Service stellt einige REST Endpunkte zur Verfügung um die Daten der Akteure in der Simulation abfragen zu können. Das Entity-Serivce selbst stellt niemals anfragen an andere Services im System.

Tracking-Service Das Entity-Service stellt einige REST Endpunkte zur Verfügung um den Stand der Akteure in der Simulation abfragen zu können. Das Tracking-Serivce selbst stellt niemals anfragen an andere Services im System.

1.2.2 Message Oriented Middleware

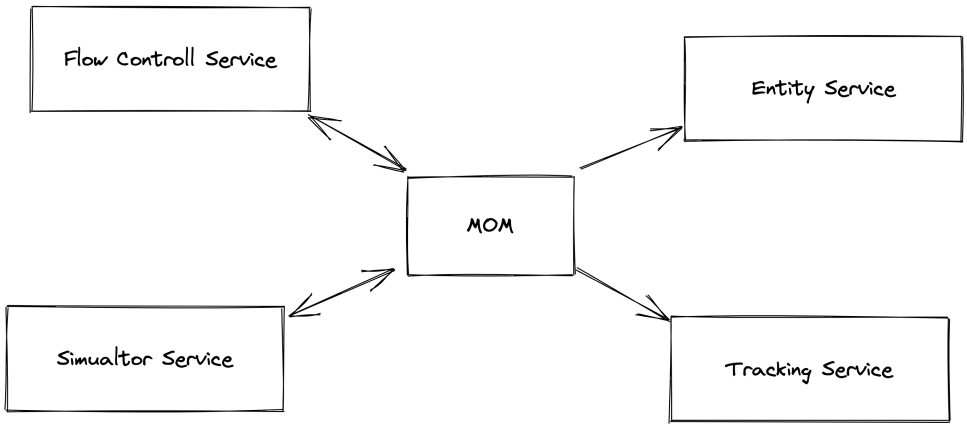

Abbildung 3: Überblick der MoM Kommunikation

In der Abbildung 3 kann man einen Überblick der MoM Kommunikation der Services sehen. Wie in der Abbildung zu sehen ist, empfangt das *Tracking-Service* und das *Entity-Service* nur Daten von der MoM und senden keine zu ihr. Auf der anderen Seite sieht man dass das *Flowcontrol-Service* und *Simulator-Service* Daten an die MoM senden und empfangen.

Insgesamt gibt es sechs Queues und eine Fanout-Exchange (mit dem Namen `fanout.carState`). Die Namen der Queues sind wie folgt:

1. `car-state-tracking`

2. `car-state-flow`
3. `car`
4. `traffic-light`
5. `traffic-light-state`
6. `car-speed`

In den folgenden Paragraphen werden wir uns ansehen wie die Queues und die Fanout-Exchange verwendet werden.

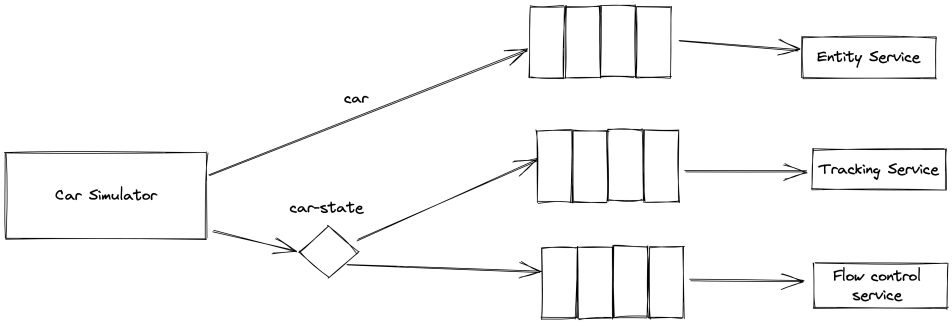

Abbildung 4: Verwendung der *Car*, *Car-State-** und Exchange

car, car-state-*, exchange In Abbildung 4 sieht man die Verwendung der Queues *Car*, *Car-State-** und der Exchange. Am Anfang der Simulation sendet der *Car-Simulator* an die Queue *car* die Informationen für die Registrierung des Fahrzeuges. Diese Queue wird vom Entity-Service konsumiert. Während der Simulation wird die aktuelle Position, Geschwindigkeit und weiteres dynamische Daten an die Fanhout-Exchange gesendet (welche in der Abbildung 4 als Raute gekennzeichnet ist). Die Fanout-Exchange leitet diese dann an die beiden Queues *car-state-flow* und *car-state-tracking* weiter. Diese Design Entscheidung wurde getroffen, da es sich hier um ein PubSub Szenario handelt und nicht einem Competing-Consumer.

car, car-state-*, exchange In Abbildung 5 sieht man die Verwendung der Queues *traffic-light*, *traffic-light-state*. Am Anfang der Simulation sendet der *TrafficLight-Simulator* an die Queue *traffic-light* die Informationen für die Registrierung der Ampel. Diese Queue wird vom Entity-Service konsumiert. Während der Simulation wird die verbleibende Zeit im State und

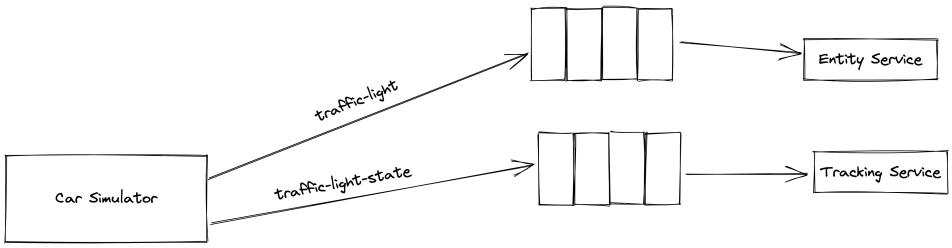

Abbildung 5: Verwendung der Queues *traffic-light*, *traffic-light-state*

weitere dynamische Daten an die Queue *traffic-light-state* gesendet, welceh vom *tracking-service* konsumiert wird.

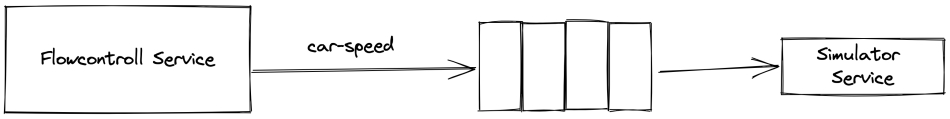

Abbildung 6: Verwendung der Queue *car-speed*

car-speed In Abbildung 5 sieht man die Verwendung der Queue *car-speed*. Das *Flowcontrol-Service* sendet nachdem die optimale Geschwindigkeit berechnet wurde, diese mittels der Queue *car-speed* and das *Simulator-Service*.

2 Logical View

TODO:

3 Physical/Deployment View

TODO: