Architektur und Design Dokument

DSE-2021S Gruppe 08 - Eder, Eidelpes und Zeisler

1. Beschreibung der Software Architektur

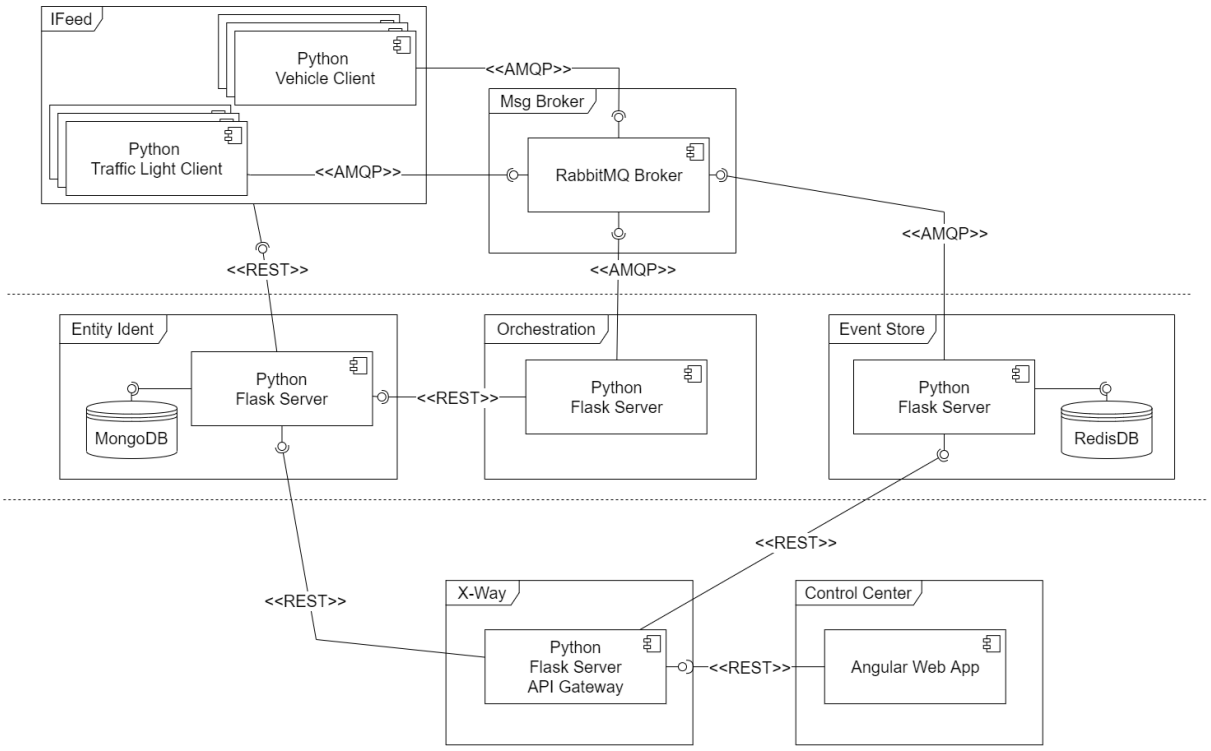

Abbildung 1 - Systemarchitektur Final

Die Architektur des Projekts ist in Abbildung 1 zu sehen. Es gibt insgesamt 7 Micro Services:

IFeed

Das IFeed ist ein Micro Service, das die im autonomen Fahrgeschehen enthaltenen Fahrzeuge simuliert. Es spawned durch Environment Variablen konfigurierbare Autos (Python Vehicle Clients) und Ampeln (Python Traffic Light Clients). Diese Clients senden periodisch Daten via asynchronem AMQP Protokoll und dem zugehörigen RabbitMQ Message Broker Statusupdates an den Orchestration Service.

Fahrzeuge senden in einem definierbaren Intervall ein Datenaufzeichnung für Autonomes Fahren (DAF) Objekt, welches die Informationen

- Vehicle Identification Number (VIN),
- Near Crash Event (NCE) ausgelöst,
- derzeitige GPS Location,
- Zeitstempel der Nachricht und
- derzeitige Geschwindigkeit,

sendet.

Als Antwort berechnet das Orchestration Service und sendet
- VIN,
- Zeitstempel der Nachricht und
- die neue Richtgeschwindigkeit,

mit der potentiell eine grüne Welle erreicht werden könnte. Diese Antwort trifft asynchron über AMQP ein, worauf das Fahrzeug seine Geschwindigkeit dementsprechend anpasst. An einem vordefinierten Kilometerpunkt löst ein Fahrzeug ein NCE Event aus, wodurch die Geschwindigkeit auf 0 km/h reduziert wird. Das Fahrzeug wartet eine vordefinierte Zeitspanne um sich zu “recovern” und fährt dann mit der zuletzt bekannten Geschwindigkeit wieder los. Während des Recovery Prozesses ignoriert das Fahrzeug die Nachrichten des Orchestrators.

Ampeln senden nach jedem Farbwechsel ein Statusupdate via AMQP über den Message Broker an den Orchestration Service, welches die
- Traffic Light ID,
- neue Farbe und
- den Zeitstempel des Switches,

enthält.

Message Broker Service

Das Message Broker Service ist eine RabbitMQ Instanz, die von
- IFeed Geräten (Fahrzeuge und Ampeln),
- Orchestration Service und
- Event Store Service

als Broker verwendet wird, um asynchron über das Advanced Message Queue Protocol (AMQP) Nachrichten auszutauschen.

Entity Ident Service

Das Entity Ident ist ein Python Flask Server mit einer anbindung an eine MongoDB Datenbank. Das Entity Ident persistiert und liefert die statischen Daten zu allen verfügbaren Fahrzeugen und Ampeln.

Dabei speichert es bei den Fahrzeugen
- OEM,
- Model Type und
- VIN.

Bei den Ampel liefert das Service Infos über deren
- ID,
- Fixed GPS Location,
- Viewing Range,
- Switching Time und
- Starting Color.

Das Entity Ident verfügt nur über REST Schnittstellen, da es hauptsächlich informationen zum Initialisieren von anderen Services liefert. Außerdem verfügt das Entity Ident über eine Schnittstelle die für die derzeitige Location des Fahrzeuges bekannt gibt, ob es in der Reichweite einer Ampel ist. Auch wenn das ein stärkeres zeitliches Coupling zwischen Entity Ident und Orchestration bewirkt, sollte dies keine negativen Auswirkungen haben, da insgesamt der Call zwischen Fahrzeug und Orchestrator asynchron über AMQP und den Message Broker verläuft.

Das Entity Ident verfügt des weiteren über eine REST Schnittstelle, auf die der X-Way Service Zugreifen kann, um das Frontend mit den möglichen Fahrzeugen und Ampeln zu initialisieren.

Orchestration Service

Das Orchestration Service ist ein Python Microservice welches periodisch Status Daten von IFeed Geräten erhält (DAF Objekt). Dadurch ist dem Orchestration Service jederzeit bekannt, wo sich ein Fahrzeug befindet, wie schnell es fährt und ob ein NCE stattgefunden hat. Es weiß außerdem auch bescheid, wann und wie oft eine Ampel schaltet und kann über den Entity Ident Service in Erfahrung bringen, ob sich ein Auto in Reichweite einer Ampel befindet.

Basierend auf diesen Daten berechnet das Orchestration Service eine neue Richtgeschwindigkeit für das Status sendende Fahrzeug und antwortet mit einem Target Velocity Objekt, welches

- VIN,
- neue Richtgeschwindigkeit und
- Zeitstempel der Antwort,

enthält. Dadurch soll eine Grüne Welle und ein potentielles durchfahren ohne Stopps erreicht werden.

Die Kommunikation mit dem Orchestration Service erfolgt in erster Linie asynchron über den Message Broker via AMQP. Des weiteren called der Orchestration Service infolge der Richtgeschwindigkeitsberechnung via REST das Entity Ident, das potentiell die Ampel(n) zurückliefert, welche das Auto in Sichtweite haben.

Das Orchestration Service bietet keine eigene Schnittstelle für das X-Way an, da dies nicht benötigt wird.

Event Store Service

Das Event Store Service agiert als Mithörer des asynchronen AMQP Datenfluss von IFeed Geräten und dem Orchestrator. Dabei werden alle Nachrichten die vom IFeed an den Orchestrator und Vice Versa gesendet werden, gleichzeitig auch an diesen Service weitergeleitet. Das Event Store Service logged diese Daten, in dem es sie kategorisiert und dann in der angebundenen Redis DB persistiert.

Dabei gibt es folgende Kategorien:

- DAF:<VIN> (DAF Objekte nach VIN geteilt)
- TL:<TLID> (Traffic Light Status Wechsel nach Traffic Light ID geteilt)
- TV:<VIN> (Target Velocity Antworten nach VIN geteilt)
- UNKNOWN (Unbekannte Objekte, die der Event Store nicht zuteilen könnte falls vorhanden)

Zusätzlich zu dieser Funktion bietet das Event Store Service einen REST Endpoint für den X-Way Service an, worüber die für das Frontend relevanten Verkehrsdaten regelmäßig aus den logs abgerufen und dort angezeigt werden.

X-Way Service

Das X-Way Service bildet als Python Flask Server eine Fassade für die Micro Services. Es enthält die nach außen zugängliche API für das Frontend (Control Center Service). Es aggregiert die REST Schnittstellen von

- Entity Ident Service und
- Event Store Service

Über das Entity Ident Service macht dieser Server die Daten über Verfügbare IFeed Geräte (Fahrzeuge und Ampeln) verfügbar.
Über das Event Store Service können

- Datenaufzeichnung für autonomes Fahren (DAF) Events,
- Target Velocity (TV) Events und
- Ampel Switch (TL) Events

abgerufen werden.

Control Center Service

Das Control Center Service ist das von außen Erreichbares Frontend der Simulation, welches in Angular entwickelt wird. Es greift via REST auf die vom X-Way Service bereitgestellte Facade zu. Dort werden die verfügbaren IFeed Daten abgerufen (verfügbare Fahrzeuge und Ampeln). Danach werden einmal pro Sekunde die verfügbaren DAF und Traffic Light Events gepollt und auf einer Google Map visuell dargestellt.

Die Fahrzeuge bewegen sich dadurch von Süden nach Norden und die Ampeln switchen je nach Event zwischen rot und grün. Die IFeed Elemente können angeklickt werden, wodurch Detailinformationen aufscheinen.

Für ein Fahrzeug ist dadurch

- VIN
- OEM
- Model Type
- Derzeitige Geschwindigkeit
- Zeitstempel des letzten Events
- und NCE eingetreten

einzusehen.

Bei den Ampeln ist

- die ID,
- Zeitdauer zwischen den Switches,
- sichtbarerer Bereich,
- derzeitige Farbe,
- und der Zeitstempel des letzten Switches

ersichtlich.

Das Frontend selbst bietete keine Möglichkeiten, etwaige Skalierungen zu konfigurieren. Das Geschieht über Environment Variablen. Diese können bei Bedarf geändert werden. Die Änderungen treten nach einem Neustart des betreffenden Services in Kraft.

2. Logical View

Die logical view Sektion enthält ein vereinfachtes Klassendiagramm des ganzen Systems (Abbildung 2), sowie State Diagramme der wichtigsten Komponenten:

- IFeed Ampel (Abbildung 3)
- IFeed Fahrzeug (Abbildung 4)
- Orchestrator (Abbildung 5)
- Event Logger (Abbildung 6)

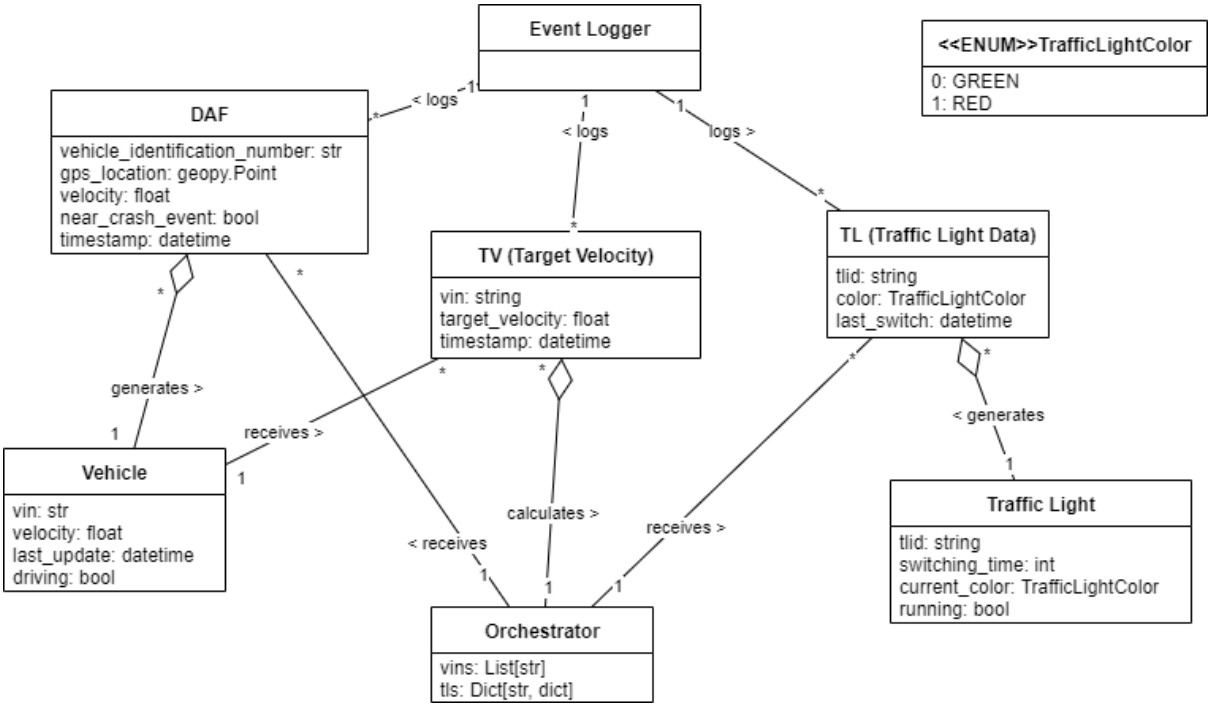

Abbildung 2 - Klassendiagramm

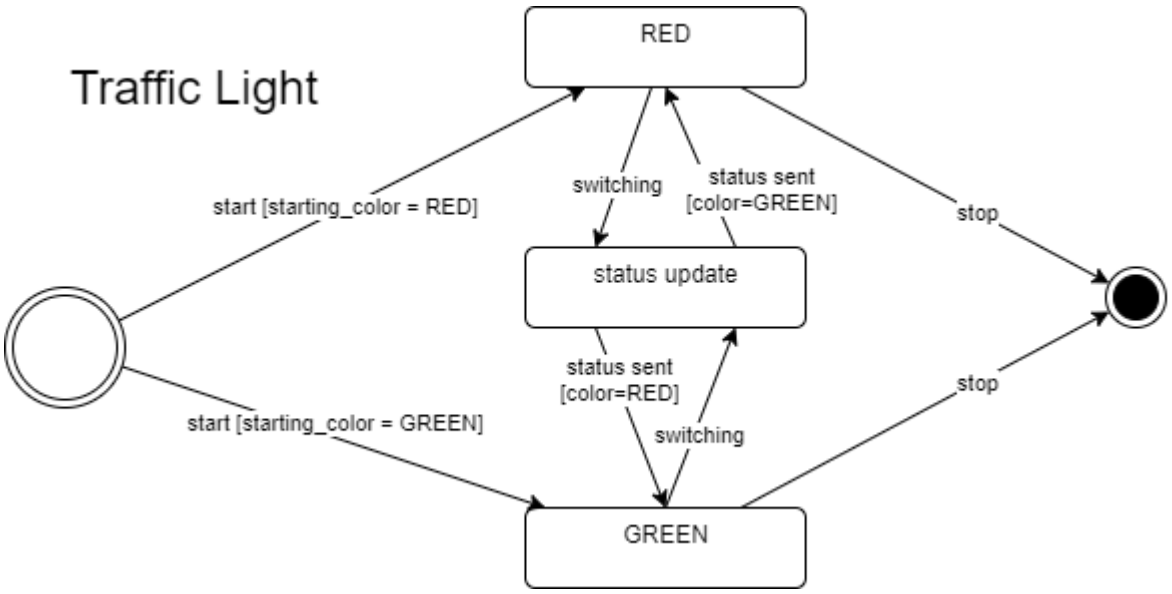

Abbildung 3 - IFeed Ampel State Diagramm

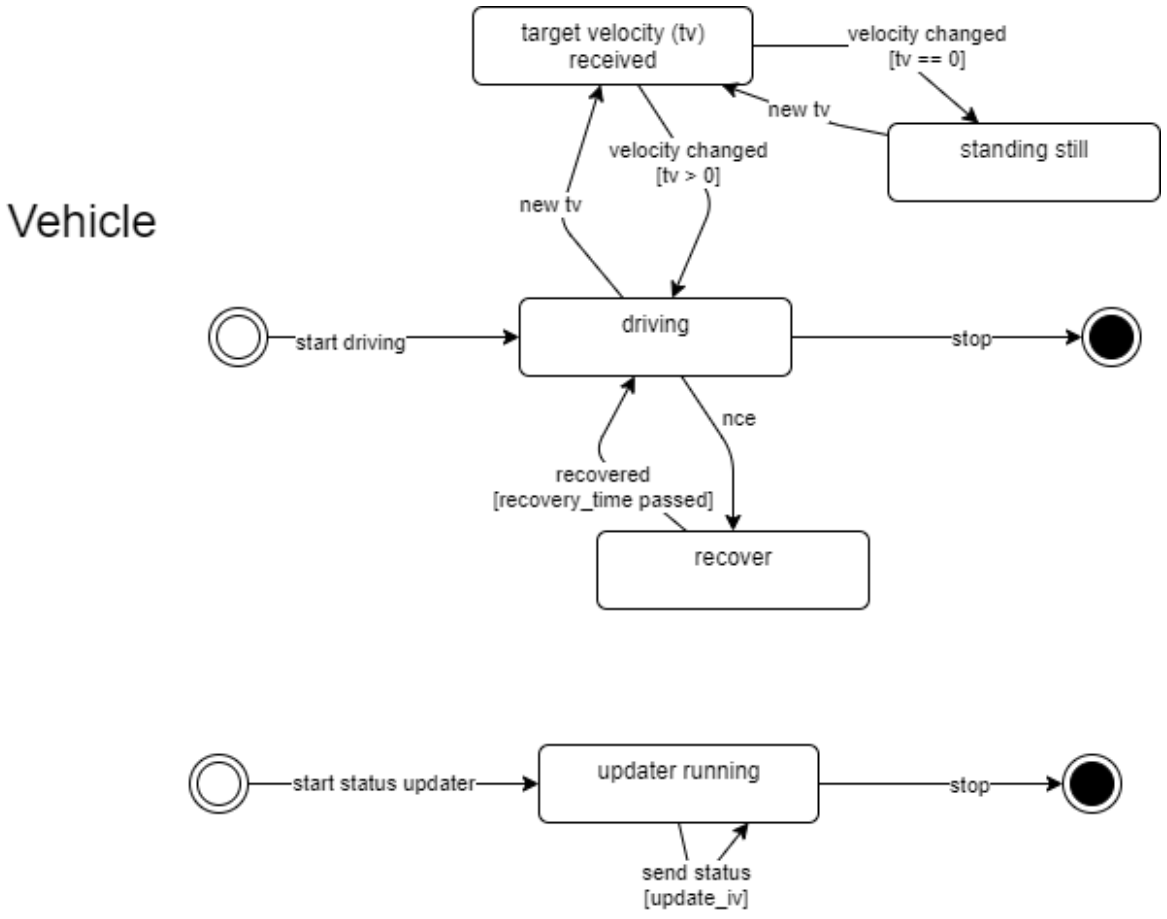

Abbildung 4 - IFeed Fahrzeug State Diagramme

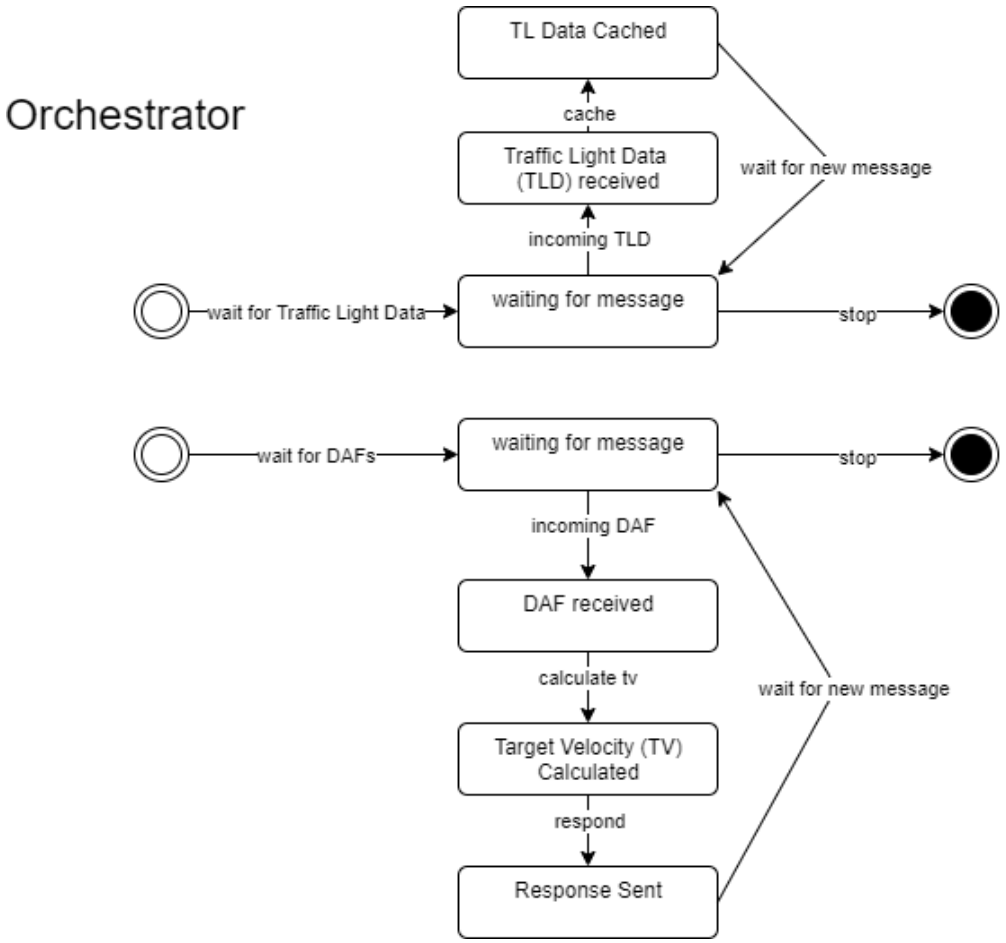

Abbildung 5 - Orchestrator State Diagramme

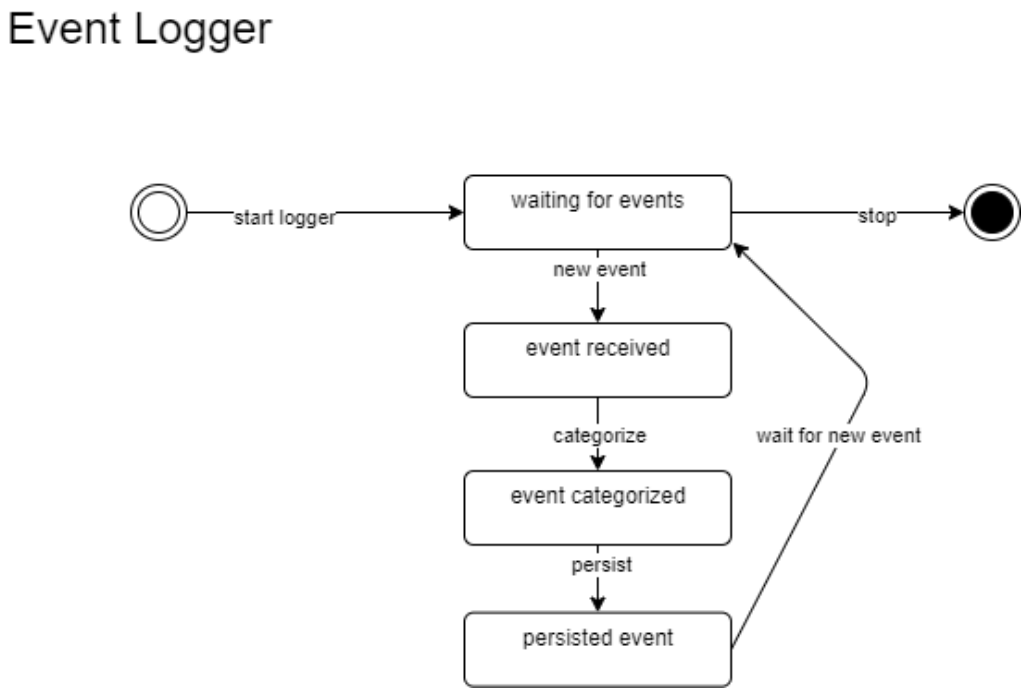

Abbildung 5 - Event Logger State Diagramm

3. Physical View

Abbildung 7 zeigt das Deployment Diagramm des Systems. Die dependencies sind vertikal aufzulösen. Jedes <<device>> entspricht einem eigenen Computing Node. Das <<execution environment>> sind verschiedene Docker Container.

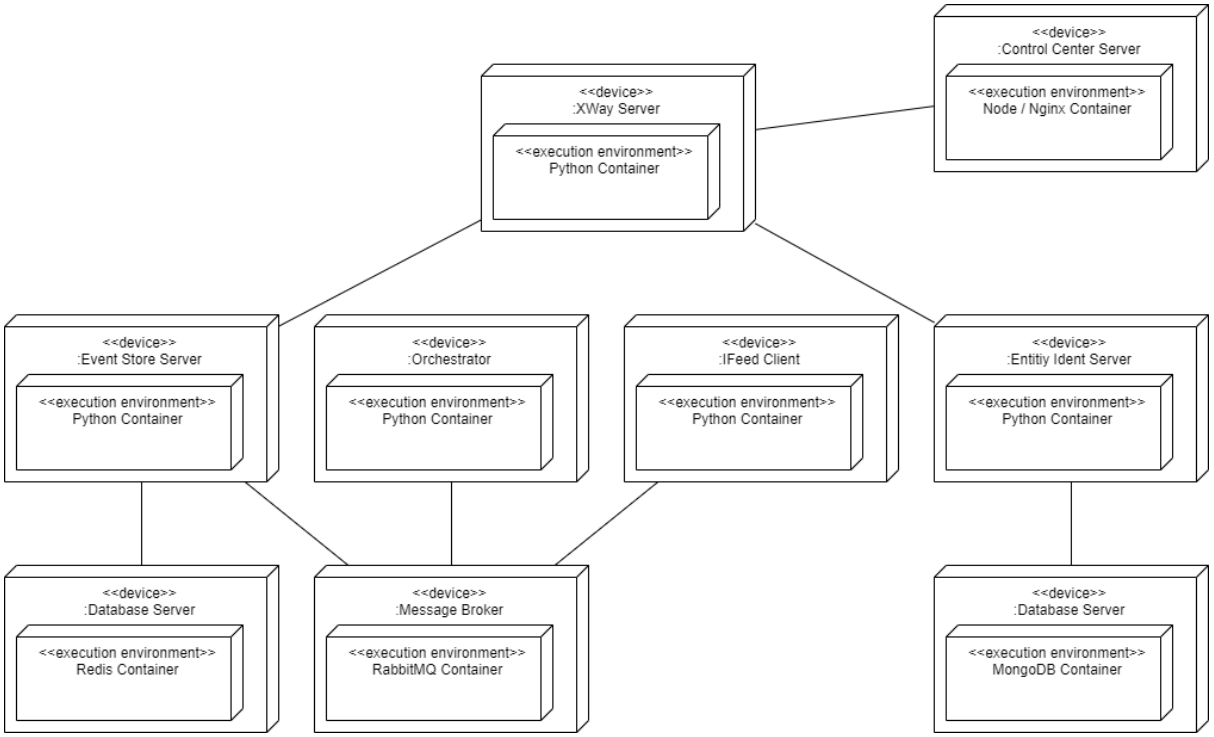

Abbildung 7 - Deployment Diagramm