Fraunhofer
FOKUS

TECHNISCHE UNIVERSITÃT BERLIN
FakultÃt IV - Institut fÃ¼r Telekommunikationssysteme
Fachgebiet Architekturen der Vermittlungsknoten

Projektdokumentation
5G & IIoT Projekt

Visualisierung der Machine-2-Machine Kommunikation innerhalb einer Webapplikation

vorgelegt von: Teham Buiyan und Linh KÃstner
Betreuer: Daniel Nehls
eingereicht am: 7. August 2018

Eidesstattliche ErklÃrung

Wir, Teham Buiyan und Linh KÃstner, versichern hiermit an Eides statt, dass wir unsere Projektdokumentation - *5G & IIoT Projekt* mit dem Thema

> *Visualisierung der Machine-2-Machine Kommunikation innerhalb einer Webapplikation*

selbstÃndig und eigenhÃndig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.

Berlin, den 7. August 2018

Teham Buiyan und Linh KÃstner

Inhaltsverzeichnis

1 Einleitung

Durch den stetigen Fortschritt im Bereich der Informationstechnik, wurden in den vergangenen Jahren innerhalb der Industrie innovative MÃ¶glichkeiten in Gang gesetzt. Dabei spielt die Vision von mitdenkenden, sich gegenseitig kontrollierenden Maschinen, eine wichtige Rolle um den Menschen in diversen Aufgaben wie der Wartung oder der Kontrolle von Maschinen zu entlasten und somit den industriellen Gesamtprozess effektiver und sicherer zu gestalten beziehungsweise noch weiter zu automatisieren. Maschinen sollen nicht mehr nur strikten Anweisungen folgen sondern auch mitdenken und selbst agieren kÃ¶nnen. Dies kann nur auf Grundlage einer sicheren Kommunikation zwischen den Maschinen erfolgen. Diese sogenannte M2M Kommunikation (Maschine zu Maschine) spielt eine essentielle Rolle innerhalb des Industriellen Internets der Dinge und als Herausforderung ergeben sich damit diverse Aspekte, welche es zu beachten gilt. Zum einen produzieren unterschiedliche Maschinen auch unterschiedliche Datenformate oder arbeiten mit unterschiedlichen Protokollen und Sicherheitstechnologien. Eine Kommunikation kann somit nur mit einer vorherigen Standardisierung geschehen Die Kommunikation zwischen den Maschinen kann, besonders in einem Anwendungsfall mit mehreren hundert oder sogar tausend Maschinen sehr schnell unÃ¼bersichtlich werden und die gesamte Netzwerkstruktur verkomplizieren. Ferner sind die Maschinennachrichten so aufzubereiten, dass diese auch fÃ¼r Menschen lesbar ist. Dies ist vor allem fÃ¼r Problembehandlung von enormer Bedeutung. Es muss sich weiter um die Strukturierung und Aufbereitung der gesammelten Daten gekÃ¼mmert sowie Plattformen fÃ¼r Anwendungen geschaffen werden, welche es erlauben spezifische Applikationen laufen zu lassen, die jene Maschinendaten verwenden kÃ¶nnen. FÃ¼r all diese Probleme wurde die Forschung in den letzten Jahren intensiviert und es ergaben sich diverse AnsÃtze, welche Grundlage dieses Projektes sind und im weiteren Verlauf nÃher erlÃutert werden. Ziel des Projektes war es, eine industrienahe Kommunikation zwischen Maschinen zum Zwecke der Ãbersichtlichkeit und der Effizient bezÃ¼glich der Problembehandlung zu visualisieren. DafÃ¼r wurde eine Web-basierte Anwendung implementiert, welche es dem Nutzer erlaubt, eine Kommunikation live mitzuverfolgen sowie alle Netzwerkkomponenten zu sehen. Dies stellt einen wichtigen Schritt fÃ¼r die Ãbersicht und Durchsichtigkeit eines solchen Netzwerkes dar.

2 Theorie

Bevor das Projekt weiter erlÃutert wird, sollen auf die wichtigsten Theorieaspekte, welche als Grundlage fÃ¼r das Projekt dienen, eingegangen werden.

2.1 Die Verwaltungsschaale

Die bereits oben genannte Problematik der unterschiedlichen Datenformate und weiteren Aspekten wie der unterschiedlichen Sicherheitsprotokolle und Authentifizierungsstandards wurde durch das relativ aktuelle Konzept der Verwaltungsschale herangegangen. Der Ansatz sieht vor, alle Netzwerkkomponenten mit einer Verwaltungsschale (engl. **A**dministration **A**sset **S**hell (AAS)) auszustatten, welche die unterschiedlichen Datenformate, Funktionen und Protokolle in einem standardisierten Format zur VerfÃ¼gung stellt und fÃ¼r eine Kommunikation mit anderen Verwaltungsschalen aufbereitet. Somit kann ein Zugriff auf die Daten und Funktionen unterschiedlichster Maschinen erst ermÃ¶glicht werden. Abbildung 1 zeigt exemplarisch Inhalte, welche durch eine Verwaltungsschale bereitgestellt werden kÃ¶nnen.

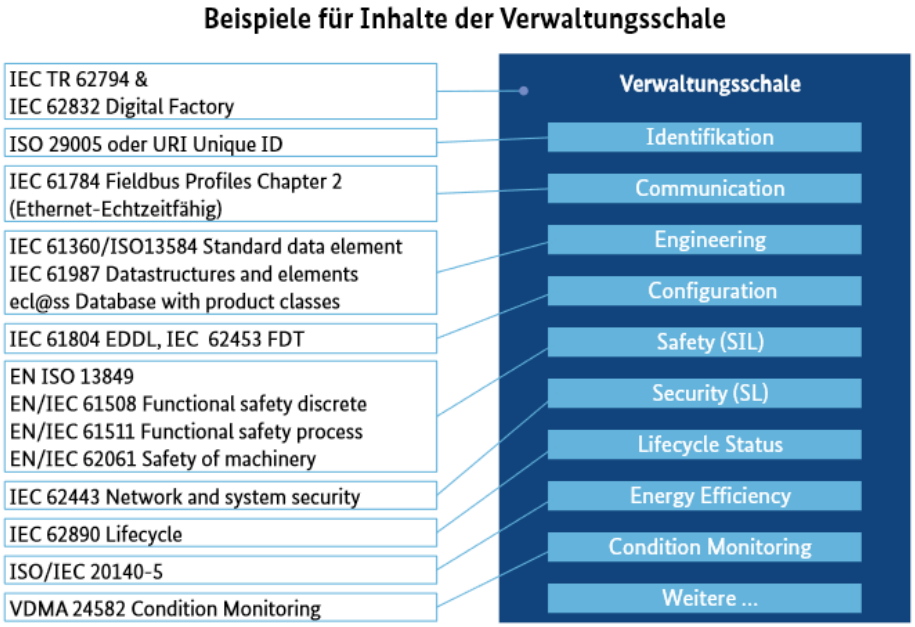

Abbildung 1: Beispielhafte Inhalte einer Verwaltungsschale (**Quelle: ZVEI SG Modelle und Standards**)

Hierin ist zu erkennen, dass die diversen Protokolle fÃ¼r verschiedenste Aspekte wie Sicherheit, Identifikation, Kommunikation oder Energieeffizienz, von der Verwaltungsschale standardisiert bereitgestellt werden muss, um eine konforme Kommunikation mit anderen Verwaltungsschalen zu ermÃ¶glichen.

In Abbildung 2 ist der detaillierte Aufbau einer Verwaltungsschale zu erkennen.

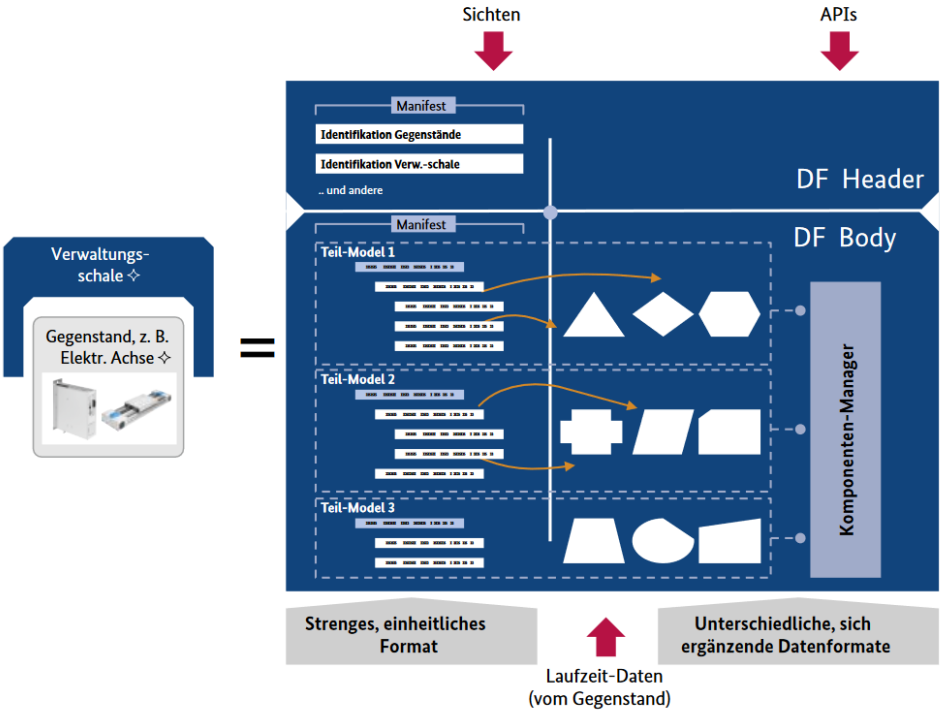

Abbildung 2: Detaillierte Struktur der Verwaltungsschale (**Quelle: ZVEI SG Modelle und Standards**)

Diese besteht aus einem Header und einem Body. Der Header beinhaltet dabei identifizierenden Merkmale der jeweiligen Komponente um damit klar auf IdentitÃt und FÃhigkeiten der jeweiligen Komponente zu verweisen. Der Body enthÃlt dann die spezifischen Teilmodelle zu den verschiedenen AnwendungsfÃllen/Sichten, welche exemplarisch bereits in Abbildung 1 aufgelistet wurden. Die verschiedenen Datenformate werden im Manifest auf ein strenges, einheitliches Format genormt um einen problemlosen Zugriff durch andere Schalen zu ermÃ¶glichen. Die Daten werden dabei mit dem oneM2M Standard genormt, sodass diese fÃ¼r die Nutzung innerhalb einer oneM2M Plattform wie beispielsweise OpenMTC genutzt werden kÃ¶nnen (Dazu spÃter mehr). Der Komponenten Manager ist fÃ¼r die Verwaltung der verschiedenen Teilmodelle zustÃndig. Ãber diesen erfolgt auch der Zugriff auf die spezifischen Daten und Funktionen der Teilmodelle.

Die wichtigsten Punkte der Verwaltungsschale sind im Folgenden zusammenfassend aufgelistet:

- Die Verwaltungsschale soll Daten und Funktionen zu verschiedensten Anwendungsszenarien in sogenannten Sichten bereitstellen. Sichten kÃ¶nnen beispielsweise Angaben zu Bereichen wie GeschÃftlich, Leistung, oder FunktionalitÃt sein.
- Die Verwaltungsschale besteht aus Body und Header.
- Der Header beinhaltet Identifikationsmerkmale, welche es erlauben auf den Verwendungszweck, mÃ¶gliche Funktionen der jeweiligen Komponente zu schlieÃen.
- Der Body beinhaltet die detaillierten Untermodelle und den Komponentenmanger, welcher Zugriff auf diese Untermodelle bietet.

2.2 OpenMTC

Eine auf dem oneM2M Standard basierte Plattform ist OpenMTC, welche am Fraunhofer Fokus entwickelt worden ist. Diese Plattform soll es dem Entwickler erleichtern, anwendungsspezifische Applikationen fÃ¼r die Verwendung innerhalb einer Industrie 4.0 Umgebung zu kreieren. Diese soll Daten von verschiedensten Komponenten Ã¼ber Gateways sammeln und dem OpenMTC Backend zur VerfÃ¼gung stellen, auf welches wiederum von den Applikationen zugegriffen werden kann. In Abblildung 3 ist das Prinzip verbildlicht.

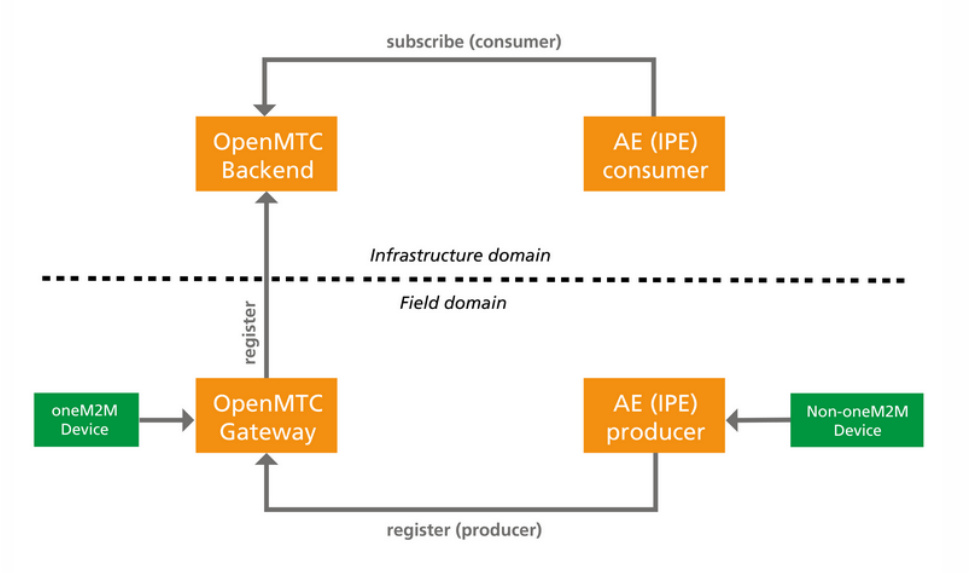

Abbildung 3: OpenMTC Struktur (**Quelle: openmtc.org)**

OneM2M-Komponenten werden automatisch erkannt und auf ein Gateway registriert. Durch die im vorigen Kapitel besprochenen Verwaltungsschalen kÃ¶nnen somit eben solche oneM2M Komponenten erzeugt werden, die automatisch von einem Gateway registriert werden.

FÃ¼r die Strukturierung der gesammelten Daten wird das Prinzip der "Containerisation"verwendet, welches es erlaubt, geschachtelte Containerstrukturen zu erstellen, in welchem sich Daten befinden. Diese kÃ¶nnen nach Belieben mit Labels oder Typen versehen werden, um eine Struktur und Trennung der Daten zu schaffen. OpenMTC stellt dem Entwickler dafÃ¼r eine Reihe nÃ¼tzlicher Funktionen zur VerfÃ¼gung, welche in Tabelle **??** erlÃutert sind.

Funktion	Beschreibung
create_application(application, path)	Erstellt eine Applikation innerhalb der OpenMTC Umgebung. Unter application muss der Name der App angegeben werden. Die Angabe des gewÃ¼nschten Pfades ist optional.
discover(path, filter_criteria)	Erkennt zu einem angegeben Elternpfad (path) die zugehÃ¶rigen Unterpfade und gibt diese innerhalb einer Liste von Adressen zurÃ¼ck. Unter filter_criteria kann zudem das Filterkriterium angegeben werden, um spezifische Containeradressen zu erhalten (bspw. alle Container mit Typ 14, etc.)
create_container(target, container, labels)	Erzeugt einen Container innerhalb der Ressourcen Struktur. Nach Wunsch kÃ¶nnen diese mit Label versehen werden und angegeben werden wieviele Container maximal angezeigt werden sollen.
push_content(container, content)	Sendet Daten (content) an einen Container (container). Dabei kÃ¶nnen als Daten Python Strings, Listen oder Dictionaries gesendet werden.
get_content(container)	Ruft Daten aus einem Container ab.
add_container_subscription(container, handler, data_handler, filter_criteria)	Subskription auf einen container, um automatisch die aktuellsten Daten dieses Containers zu erhalten. Mit filter_criteria kann auf spezifische Container innerhalb der Containerstruktur subskribiert werden.
emit(message, event)	Sendet Daten(message) via Websockets an das Frontend. Event ist dabei der "Kanal" auf welchem gesendet wird. Von Frontend Seite kann durch Angabe diesen Kanals, darauf gelauscht werden, um Daten automatisch zu bekommen

Tabelle 1: wichtige OpenMTC Funktionen

Obige Funktionen wurden im Projekt verwendet, um Daten in Container zu lagern

- **push_data()**

und aus Containern zu erhalten

- **get_content(), add_container_subscription()**

Weiterhin sind Funktionen von Relevanz, welche die Adressen spezifischer Container zurÃ¼ckgeben

- **discover()**.

2.3 RDF und JSON-LD

Das RDF-Modell (englisch fÃ¼r Ressource Description Framework) beschreibt, wie der Name bereits vermuten lÃsst, ein Konzept zur Formulierung und Beschreibung von Ressourcen innerhalb des Internets. Das Konzept beruht auf Graphen, welche eine Ansammlung von Aussagen Ã¼ber diese Ressourcen sind. Jene Aussagen werden immer als Triple formuliert, welche die Ressource nÃher beschreiben soll. Dabei ist die Ressource um die es geht das Subjekt, das durch ein PrÃdikat in Verbindung zum Objekt gebracht wird. Um das RDF Konzept zu serialisieren, also in einem speicherfÃhigem Format einzubetten, wird Ã¼blicherweise JSON-LD (JSON-Linked Data) genutzt. JSON-LD gibt als Erweiterung zum JSON-Format, den Informationen einen Kontext und eine Identifizierung, womit Daten von Maschinen leichter in einen gedeutet und verstanden werden kÃ¶nnen. AuÃerdem sind Informationen zu einem Subjekt durchweg mit anderen Informationen vernetzt, womit ein Subjekt durch Vernetzungen zu weiteren Informationen nÃher beschrieben werden kann. Abbildung 4 verbildlicht dies an einem Beispiel.

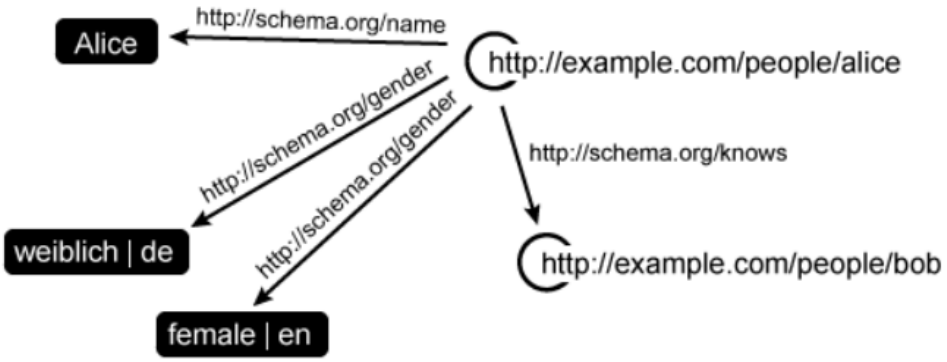

Abbildung 4: Beispiel eines Graphen nach dem RDF-Modell (**Quelle: w3.org/TR/json-ld**)

Subjekt und PrÃdikat stellen die Knoten im Graphen dar, wÃhrend die PrÃdikate die Kanten im Graphen sind. Subjekt und PrÃdikat sollten dabei eindeutige Adressen (IRIs) beinhalten wobei ein Objekt entweder eine weitere (IRI) beinhalten kann (die dann aufgelÃ¶st, weitere Informationen gibt) oder ein Text zur Beschreibung ist. In obigen Beispiel ist das Subjekt die Person Alice, welche durch die eindeutige Adresse zum Typ 'people' gehÃ¶rt. Dieses Subjekt ist, Ã¼ber die PrÃdikate 'gender', 'name' und 'knows' mit diversen Objekten vernetzt. In den Adressen der Objekte, befinden sich weitere Informationen, wie eine Spezifizierung, dass 'gender' eines von 2 Geschlechtern bei Menschen ist und dieses in unterschiedlichen Sprachen dargestellt werden kann. Ein Objekt kann dabei auch eine weitere IRI enthalten, wie beispielsweise die Person Bob, welche aufgelÃ¶st weitere Informationen enthÃlt.

Maschinennachrichten

Auch die im Projekt erzeugten Maschinennachrichten wurden mit dem JSON-LD Format serialisiert. Ebenfalls liegt das Turtle Format vor, welches dieselbe Information in einer leichter verstÃndlichen Syntax darstellt, vor, weswegen im Folgenden auf eine solche Nachricht zur Selektion einer blauen Schokolade an den Roboter, eingegangen wird.

```
1  @prefix ns1: <http://www.vendor.com/rdf/> .
2  @prefix ns2: <http://www.fokus.fraunhofer.de/ontologies/i40/> .
3
4  ns1:Message3CTF3N a ns2:Message ;
5      ns2:RID "/robot-cse/onem2m" ;
6      ns2:SID "/ip-cse-1/onem2m" ;
7      ns2:StepNumber "1" ;
8      ns2:TID ns2:T20000 ;
9      ns2:TRN "http://www.fokus.fraunhofer.de/ontologies/i40/T20000" ;
10     ns1:color "blue" .
```

Die Prefixe geben analog zum vorherigen Beispiel, den Kontext eines Subjekts oder Objekts an. In diesem Falle ist das Subjekt die Nachricht 'Message3CTF3N', die vom Typ Message ist. Dieser Typ wird weiter in den Code Zeilen 5 bis 10 weiter spezifiziert. Hierin ist zu erkennen, dass der Typ 'Message' Ã¼ber Informationen wie RID, SID, etc. verfÃ¼gt, auf die bei Bedarf zugegriffen werden kann. Der Graph einer solchen Nachricht kann analog zum Beispiel oben mit Abbildung 5 dargestellt werden.

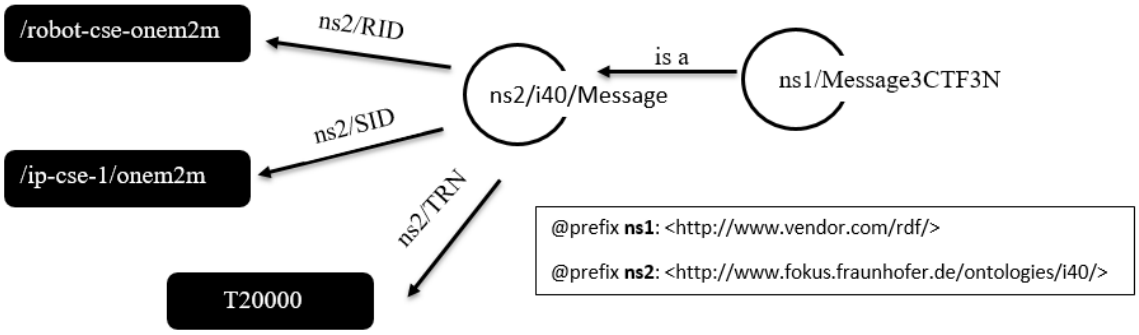

Abbildung 5: Graph einer Maschinennachricht

FÃ¼r das Projekt von Relevanz ist, die Informationen aus einem solchen Graphen zu extrahieren. Dazu stellt die 'RDF-Bibliothek' folgende Funktionen zur VerfÃ¼gung:

Funktion	Beschreibung
Graph()	Erstellt eine Graph-Variable, in die ein Graph eingebettet werden kann
parse(data, format)	Bettet einen Graphen (data) in eine Variable ein. Als zweites Argument wird das Format des Graphen angegeben (z.B.: JSON-LD).
value(predicate, object)	Entnimmt dem Graphen den entsprechenden Wert des angegebenen Objektes.

Tabelle 2: Funktionen der RDF Bibliothek

2.4 Websockets

Websockets sind eine effiziente MÃ¶glichkeit, Daten bidirektional in Echtzeit zwischen Backend (Webserver)- und Frontend (Webapp) auszutauschen. Dazu wird fÃ¼r eine Verbindung ein sogenanntes Socket (Sockel/ Verbindung) erstellt und es werden Daten auf einen bestimmten Kanal gesendet. Auf diesen Kanal kann von Frontend Seite aus 'gelauscht' werden, was einer Art Subskription gleichkommt. Die Verbindung ist dabei immer aktiv, sodass, im Gegensatz beispielsweise zu einem HTTP - Request, nicht jedes mal eine Anfrage gesendet werden muss. Somit muss kein Polling (kontinuierliches Abfragen) betrieben werden und aktuelle Daten werden in Echtzeit zum Frontend gesendet. Abbildung 6 verbildlicht oben genannten Vergleich.

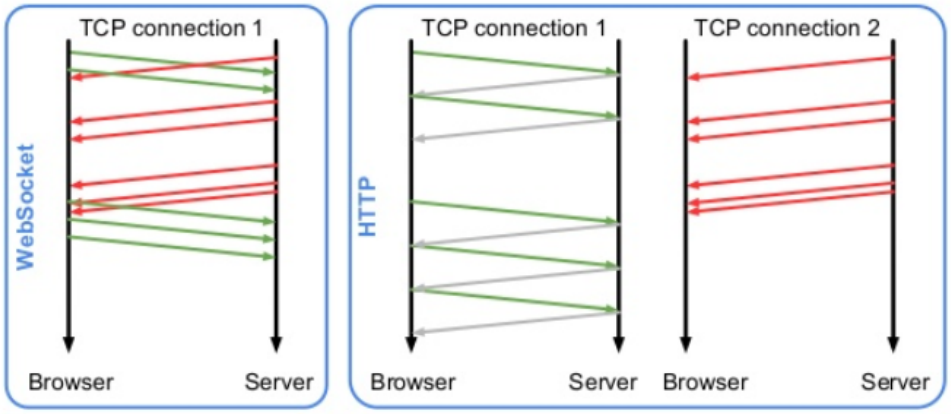

Abbildung 6: Websockets vs. HTTP-Anfragen (**Quelle: devcentral.f5.com**)

WÃhrend bei Websockets eine Bidirektionale Verbindung vorherrscht, benÃ¶tigt eine HTTP Anfrage mindestens 2 Verbindungen um Daten austauschen zu kÃ¶nnen. Durch diese Eigenschaft eigenen sich Websockets besonders gut fÃ¼r einen Chat-Raum, welcher auch Ziel des Projektes ist. Zur Verwendung von Websockets wurde im Projekt dabei die 'socket.io' Bibliothek benutzt. Im Folgenden soll kurz auf die Verwendung eingegangen werden.

Verwendung von Websockets mit socket.io

Mit

```
socket = io()
```

wird ein Socket erstellt. Wurden nun vorher auf Backend Seite mit

```
emit('message',data)
```

Daten auf den Kanal 'message' gesendet, so kann mit

```
socket.on('message', function(data){...}
```

eine Verbindung hergestellt und auf diesen Kanal gelauscht werden. Innerhalb dieser Funktion kann wird dann die Routine gestartet.

3 Setup und Motivation

Setup

Das Setup besteht aus drei Komponenten, welche jeweils mit einer AAS ausgestattet sind und somit vollstÃndige Komponenten im OpenMTC Netzwerk sind. Diese kÃ¶nnen miteinander interagieren und kommunizieren um eine gewÃ¼nschten Prozess automatisch zu bearbeiten. Konkret geht es um die Bestellung von verschiedenfarbiger Schokolade, welche durch eine GUI (ChoclateGUI) getÃtigt werden kann. Die Box (Intelligent Product) empfÃngt nach tÃtigen der Bestellung in der Chocolate-GUI die Informationen von der GUI und gibt diese an das Laufband weiter. Das Laufband erkennt anhand des Headers, die notwendigen Befehle und gibt diese an den Roboter weiter. Dieser beginnt dann mit der Abarbeitung der Bestellung. Ist diese beendet gibt der Roboter eine Notifikation an das Laufband, womit dieses automatisch weiter fÃhrt. Ein Demo Video ist auf Anfrage verfÃ¼gbar und veranschaulicht den Gesamtprozess. In Abbildung 7 ist das Setup verbildlicht.

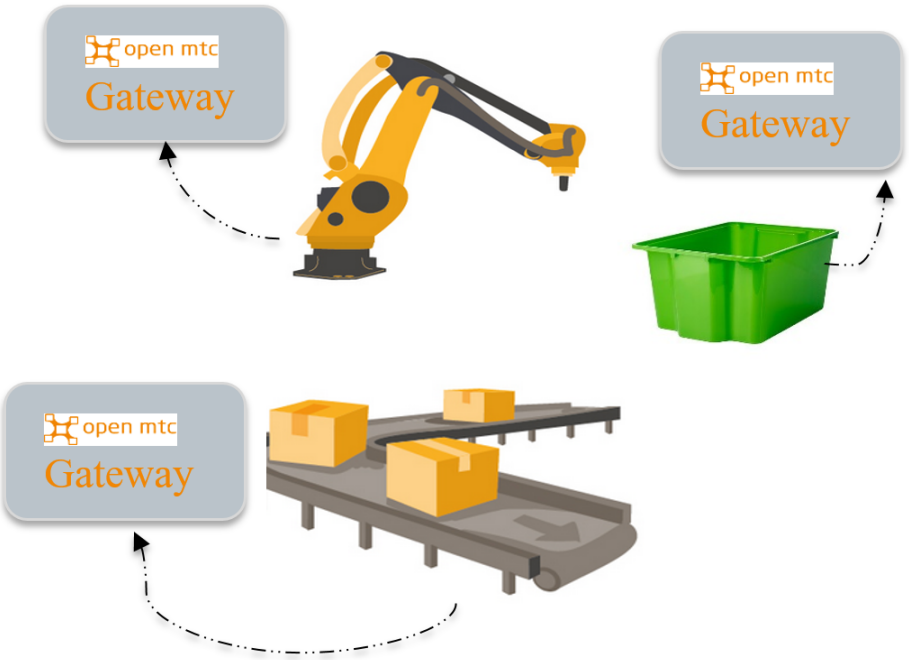

Abbildung 7: Setup **(Bildquellen: the-mtc.org, iot.do)**

Dort sind die drei erwÃhnten Komponenten zu sehen, welche die produzierten Daten/Nachrichten an das jeweilige Gateway (die AAS) senden. Die Daten/Nachrichten werden dabei im JSON-LD Format gesendet. Somit befindet sich, wie bereits schon in der Theorie prÃsentiert, in einer Maschinennachricht u.a. stets die Information Ã¼ber den Absender, den EmpfÃnger und die Nachricht, auf welche von jedem anderen OpenMTC Gateway zugegriffen werden kann. Dazu werden Funktionen der OpenMTC Plattform benutzt, wobei der detaillierte Ablauf und die Verwendung dieser Funktionen im Kapitel 'Implementierung - Backend' erlÃutert wird. Da die produzierten Daten an drei verschiedene Gateways gesendet werden, muss es Filterkriterien geben, welche eine Gruppierung der verschiedenen Gateways erlaubt. Dazu wird eine Containerstruktur angelegt, welche es ermÃ¶glicht

Daten strukturiert und mit Kriterien wie Label oder Typ zu versehen um dann durch die Vergabe von gleichen Labels oder Typen auf alle relevanten Gateways zuzugreifen.

Motivation

Die Visualisierung der Kommunikation zwischen den einzelnen Komponenten ist aufgrund diverser Aspekte, ein wichtiger Faktor, welcher noch nicht realisiert wurde. Dementsprechend war die Aufgabe eine solche Visualisierung zu implementieren. Dies hat durch folgende Punkte groÃe Relevanz:

- Die Sichtbarkeit der Kommunikation fÃ¶rdert die Durchsichtigkeit des Netzwerkes. Somit kann der Betrachter der Kommunikation zwischen den Maschinen folgen und nachvollziehen. Dies ist vor allem fÃ$\frac{1}{4}$r die Problembehandlung von hoher Bedeutung.
- Die Aufbereitung der schwer lesbaren Maschinennachrichten, erleichtert die Administration und trÃgt zu einer deutlich erhÃ¶hten VerstÃndlichkeit des Prozesses, auch fÃ$\frac{1}{4}$r AuÃenstehende, bei. AuÃerdem wird dadurch die Administration effektiver.
- Die Sichtbarkeit der Komponenten innerhalb des Netzwerkes fÃ¶rdert ebenfalls die Ãbersichtlichkeit des Netzwerkes. Dies geschieht vor allem im Hinblick auf ein deutlich komplexeres Netzwerk mit mehreren hundert Komponenten. Somit hat der Betrachter eine Ãbersicht Ã$\frac{1}{4}$ber die im Netz befindlichen GerÃte, was weiterhin ebenfalls in einer hÃ¶heren EffektivitÃt bei der Administration solcher Netzwerke resultiert.

4 Zielstellung

Die angestrebten Ziele kÃ¶nnen nach obiger Motivation wie folgt definiert werden:

- Erstellung einer Webbasierten Applikation zur Visualisierung der Maschinenkommunikation .
- Aufbereitung der Maschinennachricht zu einer Ã$\frac{1}{4}$bersichtlichen, auch fÃ$\frac{1}{4}$r AuÃenstehende, verstÃndlichen Form.
- Anzeige der Komponenten, welche sich innerhalb des Netzwerkes befinden.
- Erstellung eines ansprechenden Designs mit PersonalisierungsmÃ¶glichkeiten
- MÃ¶glichkeit weitere Funktionen in die Web Applikation zu integrieren

5 Konzeption

Backend

Wie bereits oben erwÃhnt, werden die Daten der drei Komponenten an ihre jeweiligen Gateways gesendet. Durch gleiche Labels kÃ¶nnen diese drei Gateways bildlich als ein Gateway dargestellt werden, wie dies in Abbildung 8 geschehen ist. FÃ¼r das Empfangen und die Aufbereitung der Daten aus dem Gateway ist ein Backend in Form einer Python App notwendig. Wichtig ist dabei, dass dies mittels des OpenMTC App Frameworks (Funktion create_application()) erstellt werden muss um eine richtige Ordnerstruktur zu schaffen und somit eine fehlerfreie FunktionalitÃt zu gewÃhrleisten. Die Auslagerung und Aufbereitung der Daten im JSON Format passiert ebenfalls primÃr mit den besprochenen Funktionen des OpenMTC Frameworks, das anschlieÃende Senden geschieht mittels Websockets, wobei hier auf Frontend Seite die Socket.io Bibliothek benutzt wurde.

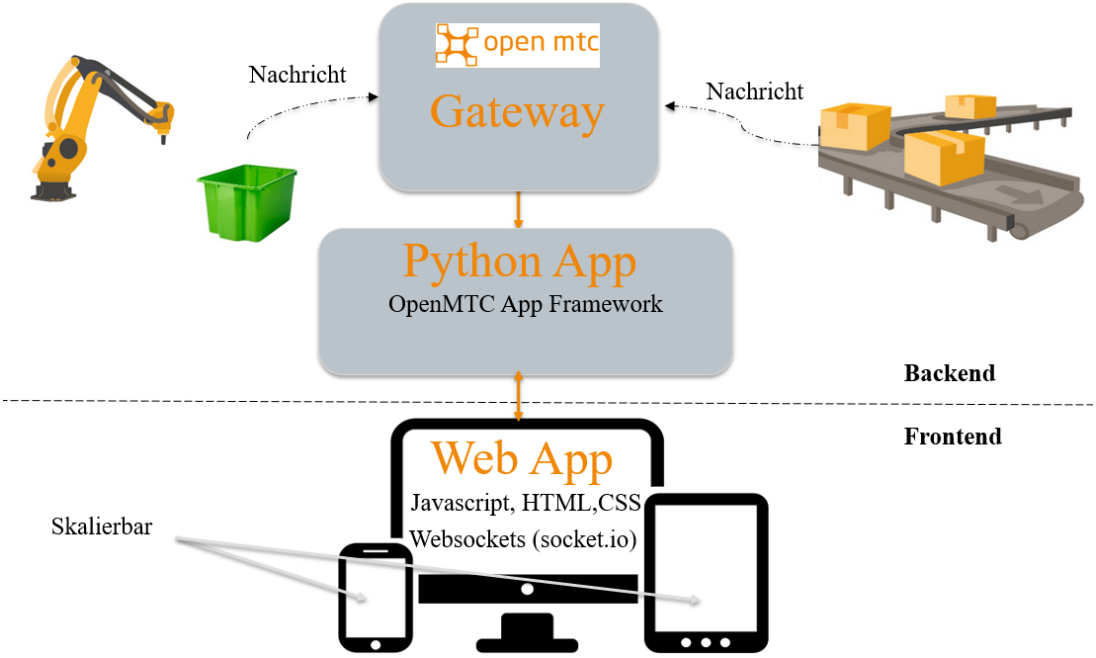

Abbildung 8: Konzeption **(Bildquellen: the-mtc.org, iot.do, fotolia.com)**

Frontend

Auf Frontend Seite wird eine Webapplikation implementiert, welche die Daten in angemessener Art und Weise darstellt. Dazu mÃ¼ssen die schon formatierten Maschinennachrichten mittels Javascript ausgelesen und auf in den jeweiligen Bereichen dargestellt werden. Dies wird im Detail im Kapitel 'Implentierung - Frontend' erlÃutert. Die Webapplikation wird dabei so implementiert, dass sie skalierbar ist und somit auch auf kleineren GerÃten wie Handys oder Tablets benutzt werden kann. Damit wird, wie Ã¼blich, HTML fÃ¼r das GrundgerÃ¼st, Javascript fÃ¼r die FunktionalitÃt und CSS zur ansprechenden Gestaltung der Website benutzt.

6 Implementierung

In diesem Kapitel wird die Implementierung des Konzeptes nÃher beschrieben. Dies erfolgt primÃr mit Ablaufdiagrammen sowie relevanten Codeauschnitten. Der Quellcode des gesamten Projektes kann auf Nachfrage bereitgestellt werden. Zu erwÃhnen ist, dass alle Ablaufdiagramme auf englisch sind, um eine intuitive Analogie zum Quellcode herzustellen, weil Funktionen im Quellcode grÃ¶Ãtenteils durch gleichnamige BlÃ¶cke im Diagramm dargestellt sind.

6.1 Backend

Um eine Testumgebung zu schaffen wurde mithilfe der auf openmtc.org verfÃ¼gbaren Sensor Demo Applikationen, eine Maschinenumgebung emuliert. Es wurden die Demo Applikationen so erweitert, dass diese den Maschinennachrichten entsprechend, SID, RID, eine zufÃllige Nachricht und alle im Netz befindlichen Komponenten bereits in einem JSON Format an das Frontend senden. Dies geschieht auÃerdem von verschiedenen Containeradressen. Somit konnte insbesondere der Frontend Code auch auÃerhalb des Labors getestet werden. Innerhalb des Labors mussten dann lediglich die Adressen der Datencontainer angepasst sowie die Routinen zur Extraktion der RDF codierten Daten implementiert werden. (Dazu im Kapitel 'reale Maschinenumgebung' mehr) Da der Backend Code der realen Maschinenumgebung auf dem Backend Code der emulierten Umgebung aufbaut wird erst auf letzteren eingegangen.

6.1.1 Emulierte Maschinenumgebung

Erzeugung emulierter Maschinennachrichten

Das Ablaufdiagramm zur Erzeugung der emulierten Maschinennachrichten ist in Abbildung 9 zu erkennen. Dieses besteht aus zwei Teilen: dem Erstellen der willkÃ¼rlich gewÃhlten Maschinen und den zufÃlligen Nachrichten sowie der Prozessierung dieser Daten. Letzterer Punkt umfasst die Erstellung einer Containerstruktur sowie dem Senden der Daten an das Gateway.

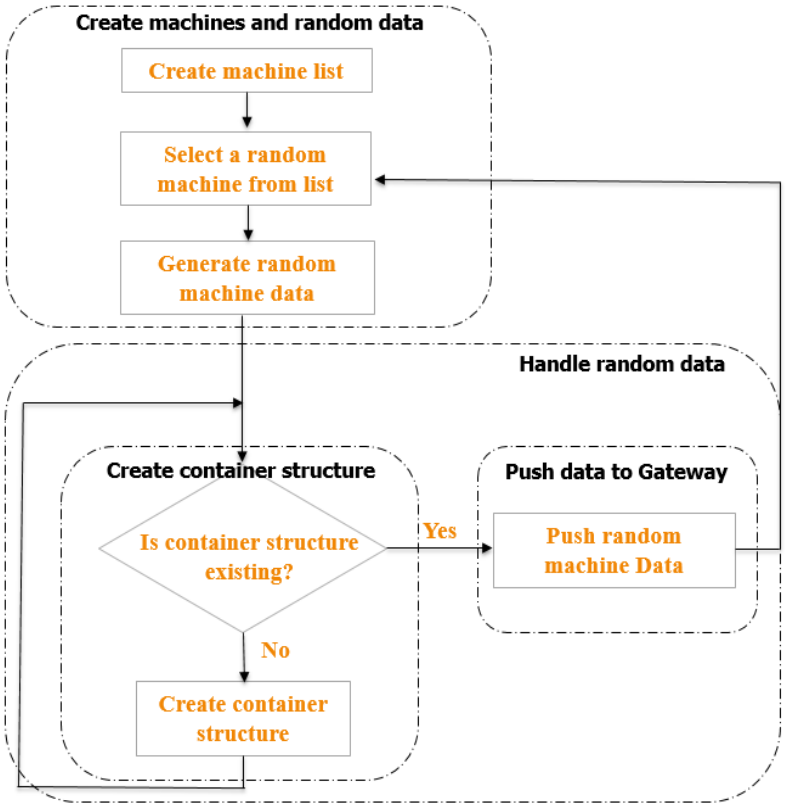

Abbildung 9: Programmablaufplan zur Erzeugung einer emulierten Maschinenumgebung

Die Erstellung der Maschinen geschieht im Block **Create machine list**, woran sich die zufÃllige Auswahl eines dieser Maschinen anschlieÃt. Darauffolgend werden die zufÃlligen Nachrichten in **Generate random machine data** erzeugt. Es wird zunÃchst Ã¼berprÃ¼ft, ob die Maschine bereits eine Containerstruktur fÃ¼r Daten dieser Maschine besitzt. Falls dies nicht der Fall ist, wird erst eine solche innerhalb des Blockes **Create container structure** erstellt um die Daten in diesen Container zu senden (Block **Push random machine data**. Dabei wird ein fÃ¼r die Container ein Label gewÃhlt. Dies ist fÃ¼r das spÃter behandelte Backend wichtig. Die Routine startet, dann bei der zufÃlligen Auswahl der nÃchsten Maschine (Block **Select random machine**), von neuem.

Empfangen der emulierten Maschinennachrichten und senden an das Frontend

Die emulierten Maschinennachrichten mÃ¼ssen, wie in der realen Umgebung spÃter auch, von einem Backend empfangen und an das Frontend weitergeleitet werden. Dazu wird das Ablaufdiagramm in Abbildung 10 betrachtet.

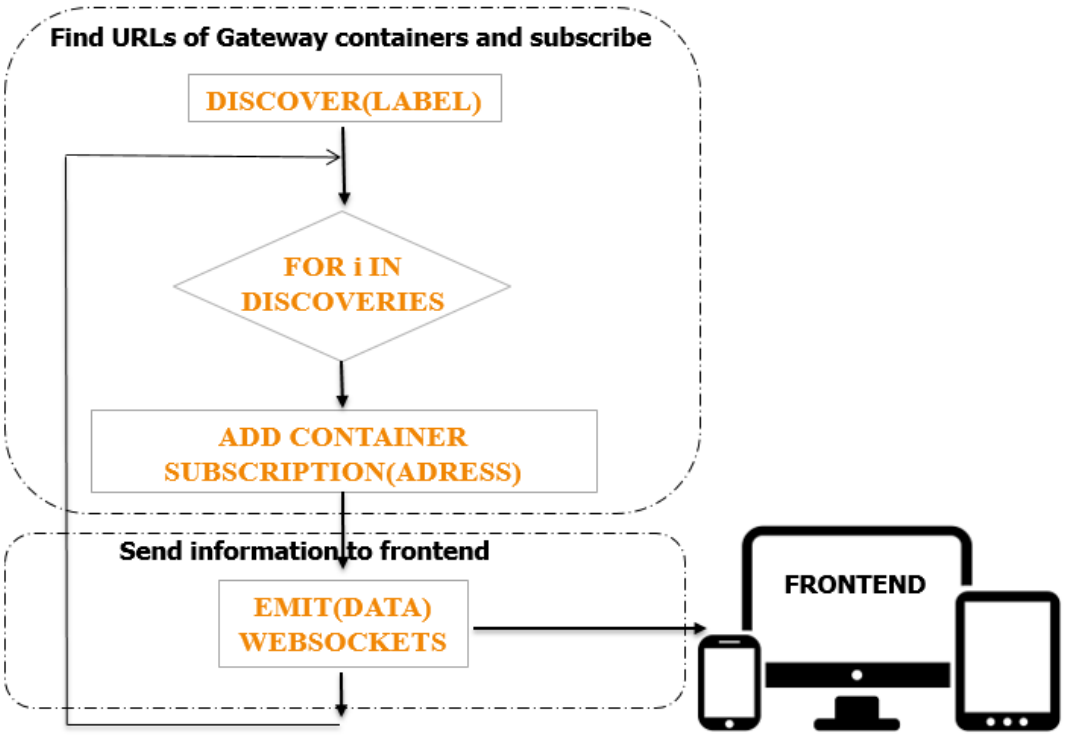

Abbildung 10: Programmablaufplan des Backendes zur Vermittlung der Maschinendaten an das Frontend

Zuerst werden im Block **DISCOVER(LABEL)** alle Adressen der Container mit dem angegeben Label gesucht und in einer Liste zurÃ¼ckgegeben. Die Label wurden dabei fÃ¼r alle Nachrichtencontainer innerhalb der emulierten Umgebung gleich gewÃhlt, sodass alle erkannt werden. AnschlieÃend wird mit einer Schleife Ã¼ber alle gefundenen Adressen iteriert und die $add_container_subscription()$ Funktion angewandt, womit auf jede dieser Container subskribiert wird. Innerhalb dieser Subskribierungsfunktion wird als Handler Funktion die emit() Funktion verwendet, welche die JSON Daten direkt an das Frontend sendet.

6.1.2 Reale Maschinenumgebung

Der Backend Code zur Verwendung innerhalb der realen Maschinenumgebung muss in Bezug zum oben behandelten Demo Code, noch um folgende Routinen erweitert werden:

- Routine zur Extraktion der RDF codierten Maschinennachrichten und Aufbereitung der Nachricht im JSON Format
- Routine zur Extraktion der RFD codierten Information Ã¼ber die tatsÃchlichen Namen der Komponenten.

Extraktion der RDF-codierten Maschinennachrichten

Die Extraktion der Maschinennachrichten wird im folgenden Codeauschnitt implementiert.

```
        #create graph variable
        msg_graph = Graph()
        #append data, which is in json-ld(in a graph) to our graph
        msg_graph.parse(data=data, format='json-ld')

        #get adress of message
        msg_uri = msg_graph.value(predicate=RDF.type,
        object=self.ns2.Message)

        # extract data from graph branches
        SID = msg_graph.value(msg_uri, self.ns2.SID)
        RID = msg_graph.value(msg_uri, self.ns2.RID)
        TRN = msg_graph.value(msg_uri, self.ns2.TRN)
```

Code Listing 1: Extraktion der RDF codierten Maschinennachrichten

Dabei werden die bereits in der Theorie erlÃuterten Funktionen der rdf-Bibliothek benutzt. Mit *Graph*() wird eine Graph-Variable erstellt, mit der ein Graph aufgenommen werden kann. Die Aufnahme des Graphen der Maschinendaten passiert mit der Funktion *parse*, wobei die Daten (data), analog wie bereits in der Demo Umgebung bereits erlÃutert, von der Subskiptionsfunktion geliefert werden und als format json-ld angegeben wird, da die Maschinendaten auch in diesem Format gesendet werden. AnschlieÃend wird dem Graphen der Maschinendaten, die gesendete Nachricht mit *value* entnommen. In diesem liegen wiederum die fÃ¼r das Frontend relevanten Information SID (Sender), RID (EmpfÃnger) und TRN (Nachricht), welche ebenfalls mit *value* entnommen werden.

Extraktion der tatsÃchlichen Komponentennamen

Weiterhin muss fÃ¼r die richtigen Namen der Komponenten auf ein Manifest zurÃ¼ckgegriffen werden. Dieses befindet sich ebenfalls innerhalb der Containerstruktur, sodass durch die entsprechende Adresse, die richtigen Namen der Komponenten mit einen *get_content*() Aufruf abgerufen werden kann. Auch diese Informationen sind RDF-codiert, sodass diese, wie schon bei den Maschinennachrichten, mit den Funktionen Graph(), parse() und value(), aus dem Graphen extrahiert werden kÃ¶nnen.

Programmablaufdiagramm

Der Teilbereich "Find URLs of Gateways and subscribeïm Ablaufdiagramm in Abbildung 11 wurde bereits fÃ¼r die Demo Umgebung behandelt und funktioniert analog. Die beiden oben behandelten Zusatzroutinen kÃ¶nnen 11 im dem **ENCODE(DATA)** Block zugeordnet werden. Der Block **JSONIFY(DATA)** beinhaltet die Aufbereitung der gewonnenen Informationen in einem JSON Konstrukt. Dies wird mit Folgendem Codeauschnitt implementiert.

```
        msg = {
              'sid': SID,
              'value': TRN,
              'rid': RID,
              'type': 'Message:',
              'devices': dev_list
         }
```

Code Listing 2: Aufbereitung der gewonnenen Informationen im JSON Format

Die Liste *dev_list* beinhaltet alle Komponenten im Netz. An diese wird dabei immer eine Komponente mit append() angefÃ¼gt, welches eine Nachricht sendet und noch nicht in der Liste vorhanden ist.

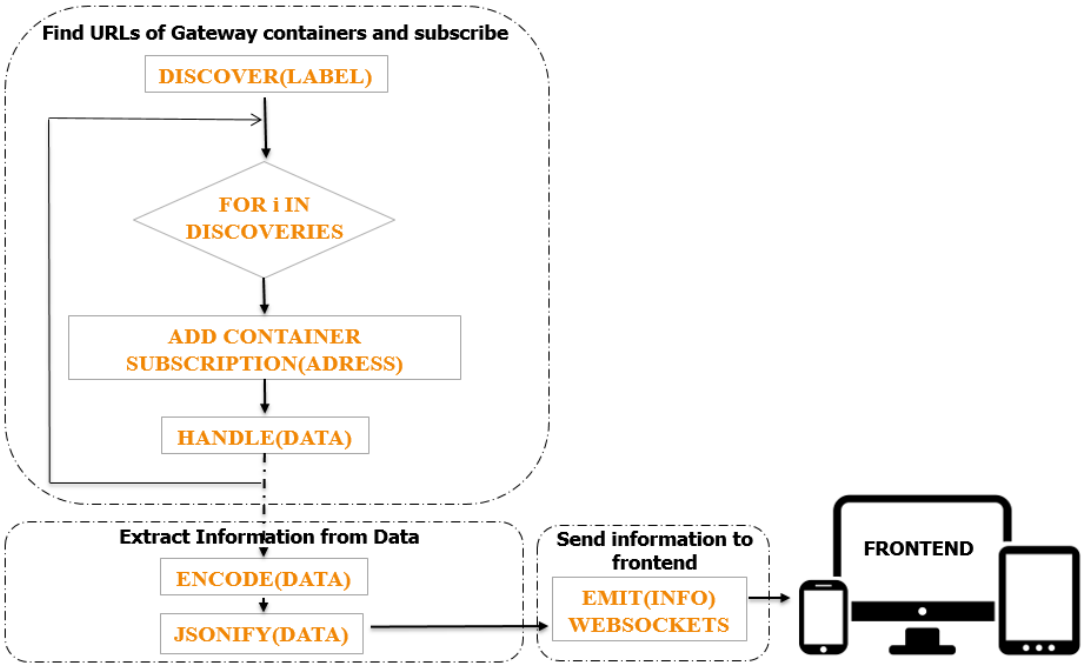

Abbildung 11: Programmablaufplan des Backendes zur Vermittlung der Maschinendaten an das Frontend

Der Block **EMIT(INFO)** sendet mit der emit() Funktion der socket.io Bibliothek, dann die aufbereiteten Daten an das Frontend.

6.2 Frontend

7 Zusammenfassung

Insgesamt wurde die Konzepte der Theorie verstanden und im Projekt erfolgreich angewandt. Damit konnten die anfangs definierten Ziele fÃ¼r das Projekt erreicht. Es konnte eine Web basierte Applikation zur Visualisierung der Kommunikation des betrachteten Prozesses implementiert werden, welche auch die MÃ¶glichkeit bietet, alle Netzwerkkomponenten zu sehen. ZusÃtzlich dazu wurde eine Filter Funktion implementiert um die Kommunikation aus Sicht einer Komponente anzuzeigen. Entsprechend wurde fÃ¼r diesen Fall auch das Design des Chat Raums angepasst. Die FunktionalitÃt der Applikation konnte im Labor erfolgreich getestet werden. Weiterhin wurde fÃ¼r Testzwecke eine emulierte Maschinenumgebung mithilfe der verfÃ¼gbaren Demo Applikationen geschaffen, welche es erlaubt auch auÃerhalb des Labors, die FunktionalitÃt zu Ã¼berprÃ¼fen sowie neue AnsÃtze auszutesten. Somit wurde eine Grundlage fÃ¼r weitere Projekte und Implementierungen geschaffen.

8 Ausblick

9 Anhang