Fraunhofer
FOKUS

TECHNISCHE UNIVERSITÄT BERLIN
Fakultät IV - Fachgebiet DCAITI

Projektdokumentation
5G & IIoT Projekt

Konzeption und Implementierung einer Augmented Reality basierten Applikation für historische Gebäude und Baustellen im Kontext einer unterstützenden Wahrnehmung im Raum

vorgelegt von: Herbert Potechius und Linh Kästner
Betreuer: Daniel Nehls
eingereicht am: 7. August 2018

Eidesstattliche Erklärung

Wir, Herbert Potechius und Linh Kästner, versichern hiermit an Eides statt, dass wir unsere Projektdokumentation - *5G & IIoT Projekt* mit dem Thema

> *Konzeption und Implementierung einer Augmented Reality basierten Applikation für historische Gebäude und Baustellen im Kontext einer unterstützenden Wahrnehmung im Raum*

selbständig und eigenhändig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.

Berlin, den 7. August 2018

Herbert Potechius und Linh Kästner

Inhaltsverzeichnis

1 Einleitung

Der stetige Fortschritt im Bereich der Informationstechnik setze in den vergangenen Jahren in nahezu allen Bereichen der Gesellschaft innovative Möglichkeiten in Gang. Das Konzept einer erweiterten Realität (Augmented Reality - AR) in der die reale Umgebung mittels virtuellen Instanzen erweitert wird, erlangt dabei, aufgrund seiner vielfältigen Einsatzmöglichkeiten, zunehmende Wichtigkeit. AR kann die Nutzerinteraktion erheblich verbessern und Aufgaben wie Wartung, Steuerung oder Kontrolle effizienter und einfacher gestalten. Durch die Visualisierung innerhalb der realen Umgebung trägt AR ferner zur Unterstützung des räumlichen Verständnisses bei und hat damit einhergehend positive Auswirkungen auf das Nutzererlebniss. Wesentliche Probleme beziehungsweise Herausforderung beim Einsatz von AR ist der Koordinatenabgleich zwischen Weltkoordinaten und dem verwendeten Darstellungsgerät - meist das Smartphone oder sogenannte Head Mounted Devices wie die Microsoft Hololens um die Illusion zu schaffen, dass Daten und Hologramme sich innerhalb der echten Welt befinden. Dazu wurden in den vergangenen Jahren vielfältige Forschung und Methoden entwickelt auf welche im späteren Verlauf noch eingegangen wird. Weitere Herausforderungen siedeln sich im Bereich der Computervision zur intelligenten Erkennung und Auswertung der Pixeldaten Computergrafik und der Bildverarbeitung, beispielsweise das Rendern von Objekten in die Szene oder die korrekte Darstellung der Objekte an.

2 Zielstellung

Ziel dieses Projektes war es eine AR basierte Applikation zu implementieren, welche es dem Nutzer erlaubt innerhalb der realen Umgebung, virtuelle Instanzen zu platzieren. Konkret soll es sich dabei um Gebäude und Gegenstände handeln, um somit eine Erweiterung von Baustellen oder Ruinen durch jene virtuellen Instanzen zu erreichen und dabei unterstützend bei der Konstruktion, Planung und Design von neuen Gebäuden zu dienen. Außerdem kann, beispielsweise durch die Erweiterung von Ruinen und der Platzierung von historischen Objekten ein interessantes Nutzererlebniss ermöglicht werden.

3 Theorie

Um ein optimales AR Erlebnis bieten zu können, müssen, wie bereits eingangs erwähnt, Aspekte aus verschiedenen Teildisziplinen miteinander verknüpft werden. Wichtige Disziplinen sind die Computervision sowie die Computergrafik. Außerdem ist der Abgleich er Koordinatensysteme von essentieller Bedeutung. Im folgenden wird auf die Theorie und Ansatzpunkte dieser Problemstellungen eingegangen.

3.1 Augmented Reality

kurze Erklärung was das ist hauptprobleme und aspekte bildverarbeitung rendering usw auch ein paar bilder zu anwendungsgebieten usw reinmachen

übergang koordinatenabgleich wichtigste

3.2 Der markerbasierte Koordinatenabgleich

Wie bereits vorher erwähnt ist ein Kernaspekt um ein angemessenes AR Erlebnis zu ermöglichen, der sogenannte Koordinatenabgleich der Weltkoordinaten mit denen der Kamera. Grundsätzlich geht es darum die Position des Nutzers und dessen Sichtfeld und Blickrichtung innerhalb der Weltkoordinaten zu lokalisieren. Praktisch entspricht dies der Orientierung und Position der Kamera des AR Geräts (Smartphone, HMD, etc.). Wichtig hierbei ist, dass die Kamera individuell kalbibriert werden muss. Forschungen, welche z.T noch hochaktuell sind haben zahlreiche Algorithmen und Lösungsansätze hervorgebracht, wobei die wichtigsten Paradigmen im folgenden aufgelistet sind:

- **Visuelle Erkennungsmethoden**. Hier wird die Position der Kamera anhand von visuellen Informationen geschätzt. Das System tätigt Rückschlüsse auf Position der Kamera anhand von Informationen, welche es visuell mitbekommt. Nachteile dieser Methoden ergeben sich bei unbekannten Umgebungen, womit vorher erst genügend Daten gesammelt werden müssen um eine korrekte Poseschätzung zu erreichen. Der Markerbasierte Ansatz gehört zu dieser Art von Erkennung, welcher es durch vordefinierte Muster, jenes Problem löst. Dies wird im weiteren Verlauf detailliert erläutert. Weitere Methoden der visuellen Erkennung sind beispielsweise die Modellbasierten Erkennungsmethoden, welche einen Abgleich zwischen visuellen Daten mit vordefinierten Modellen (bspw. CAD Modellen) bewerkstelligen.

- **Sensorbasierte Erkennungsmethoden** Hier werden verschiedene, dem System verfügbare Sensordaten wie beispielsweise, Tiefendaten oder Lokalisierungsdaten bei der Positionsbestimmung verwendet. Dafür werden spezielle Geräte verwendet, welche diese zusätzlichen Informationen bereitstellen können, beispielsweise Gyroskope, GPS oder Tiefenkameras. Sensordaten können entweder die Position oder Orientierung des Nutzers wiedergeben. Die Kombination aus Position und Orientierung wird auch **Pose** genannt.
- **hybride Ansätze** Hybride Ansätze vereinen oben genannte Methoden um eine optimierte/genauere Schätzung zu erreichen Ein Ansatz beispielsweise die Kombination zwischen einem GPS Signal, welches die Position der Kamera vermitteln kann und visuellen Informationen, welche dann die Orientierung der Kamera festlegen.

In der Praxis hat sich unter den oben vorgestellen Methoden der Markerbasierte Ansätz als Kompromiss zwischen Genauigkeit und Einfachheit in der Implementierung als weit verbreitet herausgestellt. Durch die vordefinierten Markerinformationen wird erheblich Zeit bei der Erkennung eingespart. Viele der führenden AR Framework nutzen die Markerbasierte Erkennung (ARToolkit oder ARTag). Die vordefinierten Informationen beispielsweise eine MarkerID zu welchem bestimmte Objekte eingeblendet werden sollen sind weitere Vorteile des Ansatzes Nachteile sind, dass Marker für einige Anwendungsfälle nicht geeignet sind, beispielsweise bei Anwendungen in gefährlichen Gebieten in welchem dann auf die Sensorbasierten bzw der Kombination aus visuellen Informationen mit zusätzlichen Senrdaten gesetzt wird.

Im Rahmen des Projektes wurde aufgrund obiger Erkenntnisse und der Empfehlung durch den Betreuer, der Markerbasierte Abgleich genutzt. Auf diesen wird im folgenden detailliert eingegangen.

3.2.1 Prozedur der Markererkennung

Trotz der erwähnten Vorteile der Marker gibt es einige Aspekte zu beachten. Beispielsweise können ungünstige Lichtverhältnisse oder schlechter Kontrast dazu führen dass der Marker nicht erkannt wird. Aufgrund von Farbveränderungen, welche durch ungünstiges Licht auftreten können werden grundsätzlich nur schwarz-weiss Marker benutzt. Optimalerweise reichen 4 bekannte Punkte aus um die Pose des Markers zu erkennen. Daher eignen sich Quadratische Formen als Marker am besten. Es gibt zahlreiche bekannte Bibliotheken, welche diese Art von Markern verwendet. Eine davon ist die Aruco Bibliothek, welche teil des OpenCV Frameworks ist, welches im späteren Verlauf noch erläutert wird. Aruco benutzt Schachbrettmuster um individuelle Marker zu erzeugen. Diese Marker sind durch Ihre individuelen Muster mit einer ID versehen. Dies ist nützlich um beispielsweise bestimmte Objekte mit einem bestimmten Marker zu versehen. Der erst Schritt einer Erkennung ist jedoch die Kanten des Markers zu bestimmen um zu erkennen ob es sich tatsächlich um einen Marker handelt. Folgende Schritte sind bei der Erkennung und Bestimmung des Markers notwendig:

- Umwandlung des Bildes in ein Greyscale Bild, welches die Intensitätswerte beinhaltet.
- Das erkannte Greyscale Bild wird verarbeitet indem Ungenaugikeiten gefiltert werden. Die Kanten und Ecken im Bild werden erkannt.

- Dektektierung von potentiellen Markern anhand der vorherigen Kanten und Ecken. Auch werden Kanten und Ecken, welche offensichtlich zu keinem Marker gehören herausgefiltert.
- Dekodierung der Marker
- Berechnung der Position und Orientierung des Markers (Posebestimmung)

3.2.2 Preprocessing

Im folgenden werden alle bereits oben aufgelisteten Schritte zur Markerdetektion erläutert.

Wie bereits aufgelistet wird zuerst ein Greyscale Bild benötigt. Aus diesem müssen dann die Kanten und Ecken eines potentiellen Markers extrahiert werden. Dafür kann entweder ein Treshhold Bild, also .... , genutzt werden um in diesem dann binären Bild nach Markern zu suchen oder es werden direkt die Kanten aus dem Greyscale Bild detektiert.

Der Treshold Ansatz erweisst sich als stabilder gegen mögliche Helligkeitsschwankungen. Nachem 'getresholdet' wurde liegt ein binäres Bild bestehend aus Hintergrund und Objekten vor aus denen dann die Marker erkannt werden können. Dafür werden die Kanten aller potentiellen Marker im Bild markiert und geprüft ob genau 4 gerade Linien erkannt wurden.

Sind diese Vorrausetzungen erfüllt werden die Ecken des potentiellen Markers analysiert. Wird erkannt, dass der potentielle Marker genau 4 Ecken hat wird dieser als tatsächlicher Marker gespeichtert und erkannt. Wichtig ist, dass davor die Verzerrung durch die inverse Verzerrungsfunktion, welche beim Kalibrieren der Kamera gewonnen wird, vorgenommen wird. IN Abbildung **??** ist das Greyscale Bild mit Intensitätsinformationen zu sehen, von welchem aus alle nachfolgenden Schritte zur Erkennung ausgehen. Die in diesem Projekt verwendete Aruco Bibliothek nutzt auch erläutertes Preprocessing Verfahren zur Detektion von Markern.

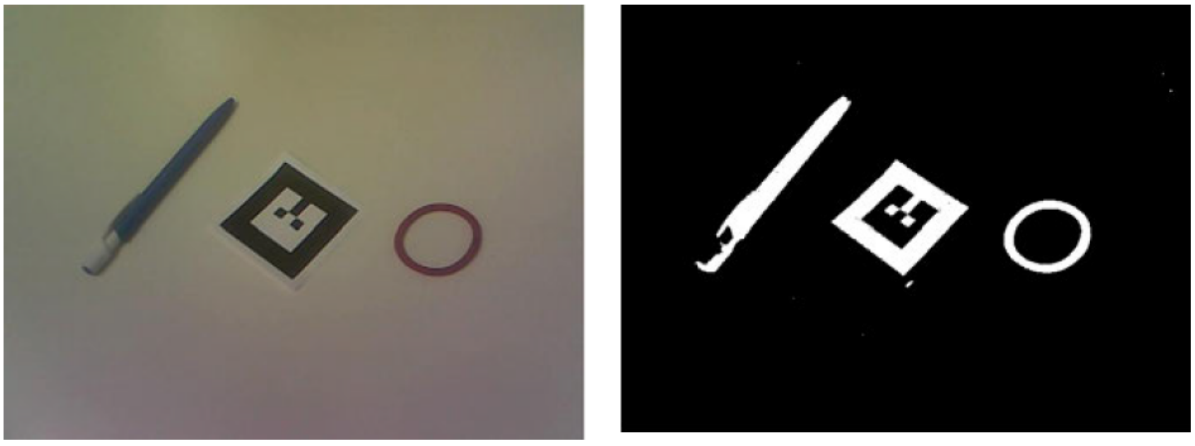

Abbildung 1: links: Originalbild, rechts: Greyscale Bild [**?**] (aus grundquelle)

Obige Preprocessing Schritte hinreichend genau auf die Markererkennung und Genauigkeit dieser zu implementieren ist eine Herausforderung, an welcher z.T. noch geforscht

wird um optimierte Lösungen zu finden. wenn beispielsweise ein falscher Treshold Wert, Quantisierungsfehler oder Rauschen und andere Störquellen, sind typische Fehlerquellen, die eine robuste Markererkennung behindern.

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 gewünschten Pfades ist optional.
discover(path, filter_criteria)	Erkennt zu einem angegeben Elternpfad (path) die zugehörigen Unterpfade und gibt diese innerhalb einer Liste von Adressen zurück. 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 können 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 können 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 zurückgeben

- **discover()**.

3.3 RDF und JSON-LD

Das RDF-Modell (englisch für Ressource Description Framework) beschreibt, wie der Name bereits vermuten lässt, ein Konzept zur Formulierung und Beschreibung von Ressourcen innerhalb des Internets. Das Konzept beruht auf Graphen, welche eine Ansammlung von Aussagen über diese Ressourcen sind. Jene Aussagen werden immer als Triple formuliert, welche die Ressource näher beschreiben soll. Dabei ist die Ressource um die es geht das Subjekt, das durch ein Prädikat in Verbindung zum Objekt gebracht wird. Um das RDF Konzept zu serialisieren, also in einem speicherfähigem Format einzubetten, wird üblicherweise 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 können. Außerdem sind Informationen zu einem Subjekt durchweg mit anderen Informationen vernetzt, womit ein Subjekt durch Vernetzungen zu weiteren Informationen näher beschrieben werden kann. Abbildung **??** verbildlicht dies an einem Beispiel.

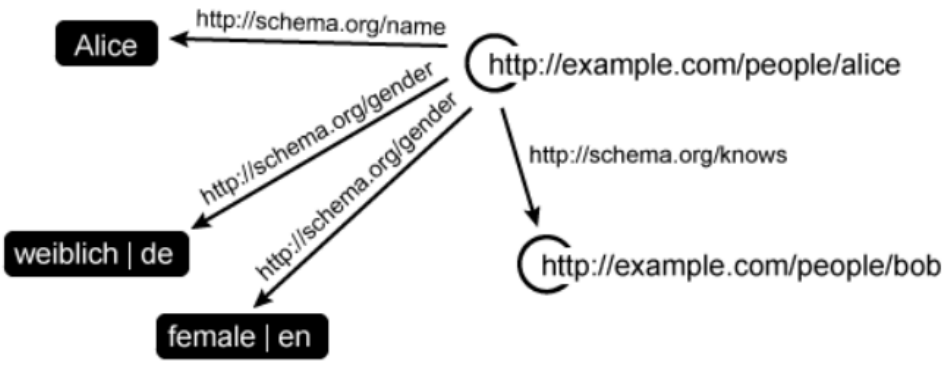

Abbildung 2: Beispiel eines Graphen nach dem RDF-Modell (**Quelle: w3.org/TR/json-ld**)

Subjekt und Prädikat stellen die Knoten im Graphen dar, während die Prädikate die Kanten im Graphen sind. Subjekt und Prädikat sollten dabei eindeutige Adressen (IRIs) beinhalten wobei ein Objekt entweder eine weitere (IRI) beinhalten kann (die dann aufgelöst, 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' gehört. Dieses Subjekt ist, über die Prädikate '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 aufgelöst weitere Informationen enthält.

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 verständlichen 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' über Informationen wie RID, SID, etc. verfügt, auf die bei Bedarf zugegriffen werden kann. Der Graph einer solchen Nachricht kann analog zum Beispiel oben mit Abbildung **??** dargestellt werden.

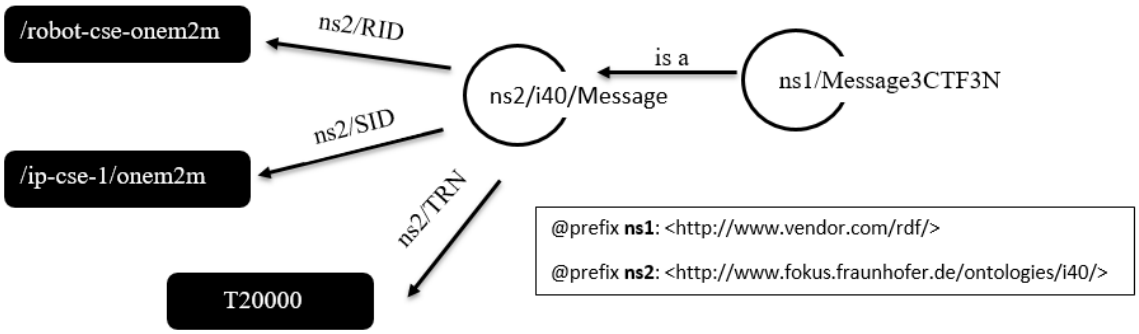

Abbildung 3: Graph einer Maschinennachricht

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

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

3.4 Websockets

Websockets sind eine effiziente Möglichkeit, Daten bidirektional in Echtzeit zwischen Backend (Webserver)- und Frontend (Webapp) auszutauschen. Dazu wird für 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 **??** verbildlicht oben genannten Vergleich.

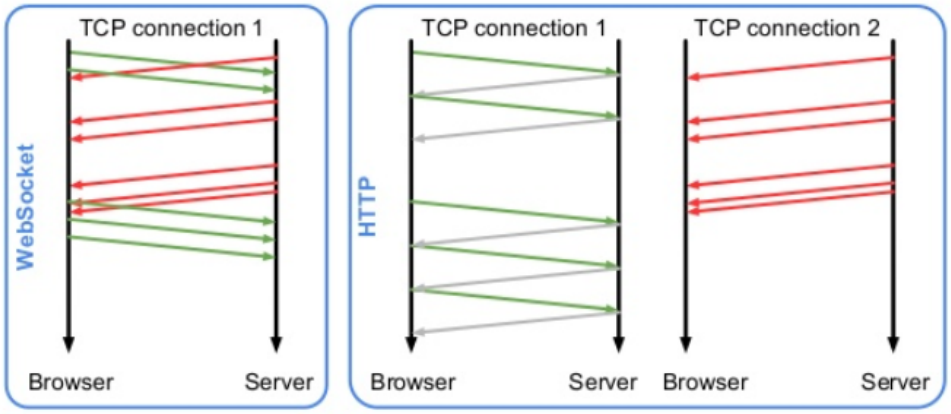

Abbildung 4: Websockets vs. HTTP-Anfragen (**Quelle: devcentral.f5.com**)

Während bei Websockets eine Bidirektionale Verbindung vorherrscht, benötigt eine HTTP Anfrage mindestens 2 Verbindungen um Daten austauschen zu können. Durch diese Eigenschaft eigenen sich Websockets besonders gut für 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.

4 Setup und Motivation

Setup

Das Setup besteht aus drei Komponenten, welche jeweils mit einer AAS ausgestattet sind und somit vollständige Komponenten im OpenMTC Netzwerk sind. Diese können miteinander interagieren und kommunizieren um eine gewünschten Prozess automatisch zu bearbeiten. Konkret geht es um die Bestellung von verschiedenfarbiger Schokolade, welche durch eine GUI (ChoclateGUI) getätigt werden kann. Die Box (Intelligent Product) empfängt nach tätigen 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 fährt. Ein Demo Video ist auf Anfrage verfügbar und veranschaulicht den Gesamtprozess. In Abbildung **??** ist das Setup verbildlicht.

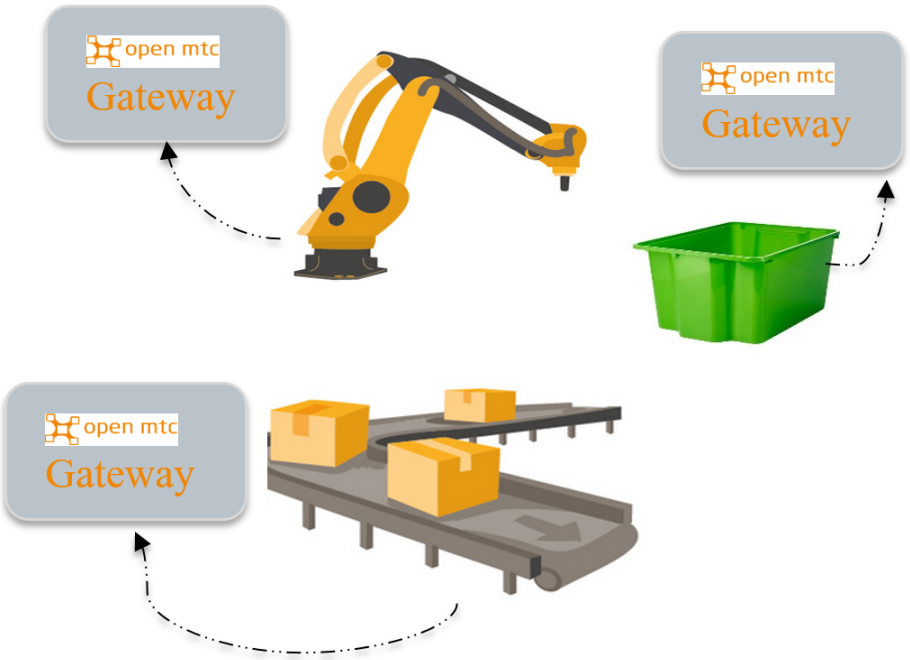

Abbildung 5: Setup **(Bildquellen: the-mtc.org, iot.do)**

Dort sind die drei erwähnten 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 präsentiert, in einer Maschinennachricht u.a. stets die Information über den Absender, den Empfänger 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' erläutert 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 ermöglicht

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 große Relevanz:

- Die Sichtbarkeit der Kommunikation fördert die Durchsichtigkeit des Netzwerkes. Somit kann der Betrachter der Kommunikation zwischen den Maschinen folgen und nachvollziehen. Dies ist vor allem für die Problembehandlung von hoher Bedeutung.
- Die Aufbereitung der schwer lesbaren Maschinennachrichten, erleichtert die Administration und trägt zu einer deutlich erhöhten Verständlichkeit des Prozesses, auch für Außenstehende, bei. Außerdem wird dadurch die Administration effektiver.
- Die Sichtbarkeit der Komponenten innerhalb des Netzwerkes fördert ebenfalls die Übersichtlichkeit des Netzwerkes. Dies geschieht vor allem im Hinblick auf ein deutlich komplexeres Netzwerk mit mehreren hundert Komponenten. Somit hat der Betrachter eine Übersicht über die im Netz befindlichen Geräte, was weiterhin ebenfalls in einer höheren Effektivität bei der Administration solcher Netzwerke resultiert.

5 Zielstellung

Die angestrebten Ziele können nach obiger Motivation wie folgt definiert werden:

- Erstellung einer Webbasierten Applikation zur Visualisierung der Maschinenkommunikation .
- Aufbereitung der Maschinennachricht zu einer übersichtlichen, auch für Außenstehende, verständlichen Form.
- Anzeige der Komponenten, welche sich innerhalb des Netzwerkes befinden.
- Erstellung eines ansprechenden Designs mit Personalisierungsmöglichkeiten
- Möglichkeit weitere Funktionen in die Web Applikation zu integrieren

6 Konzeption

Backend

Wie bereits oben erwähnt, werden die Daten der drei Komponenten an ihre jeweiligen Gateways gesendet. Durch gleiche Labels können diese drei Gateways bildlich als ein Gateway dargestellt werden, wie dies in Abbildung **??** geschehen ist. Für 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 Funktionalität zu gewährleisten. Die Auslagerung und Aufbereitung der Daten im JSON Format passiert ebenfalls primär mit den besprochenen Funktionen des OpenMTC Frameworks, das anschließende Senden geschieht mittels Websockets, wobei hier auf Frontend Seite die Socket.io Bibliothek benutzt wurde.

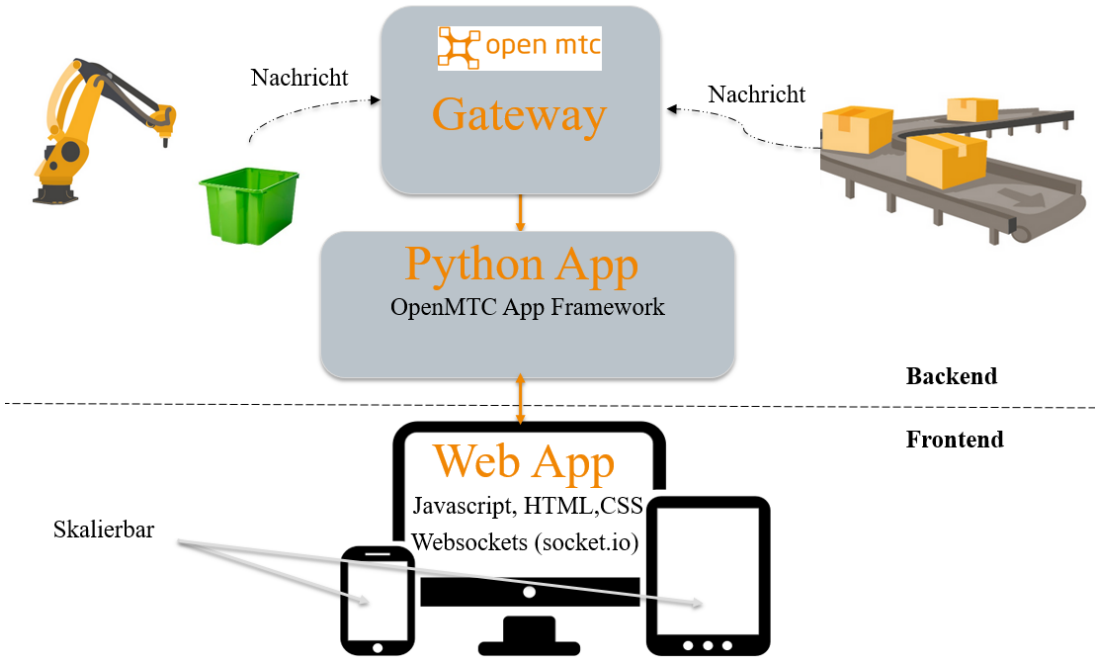

Abbildung 6: Konzeption **(Bildquellen: the-mtc.org, iot.do, fotolia.com)**

Frontend

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

7 Implementierung

In diesem Kapitel wird die Implementierung des Konzeptes näher beschrieben. Dies erfolgt primär mit Ablaufdiagrammen sowie relevanten Codeauschnitten. Der Quellcode des gesamten Projektes kann auf Nachfrage bereitgestellt werden. Zu erwähnen ist, dass alle Ablaufdiagramme auf englisch sind, um eine intuitive Analogie zum Quellcode herzustellen, weil Funktionen im Quellcode größtenteils durch gleichnamige Blöcke im Diagramm dargestellt sind.

7.1 Backend

Um eine Testumgebung zu schaffen wurde mithilfe der auf openmtc.org verfügbaren Sensor Demo Applikationen, eine Maschinenumgebung emuliert. Es wurden die Demo Applikationen so erweitert, dass diese den Maschinennachrichten entsprechend, SID, RID, eine zufällige Nachricht und alle im Netz befindlichen Komponenten bereits in einem JSON Format an das Frontend senden. Dies geschieht außerdem von verschiedenen Containeradressen. Somit konnte insbesondere der Frontend Code auch außerhalb 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.

7.1.1 Emulierte Maschinenumgebung

Erzeugung emulierter Maschinennachrichten

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

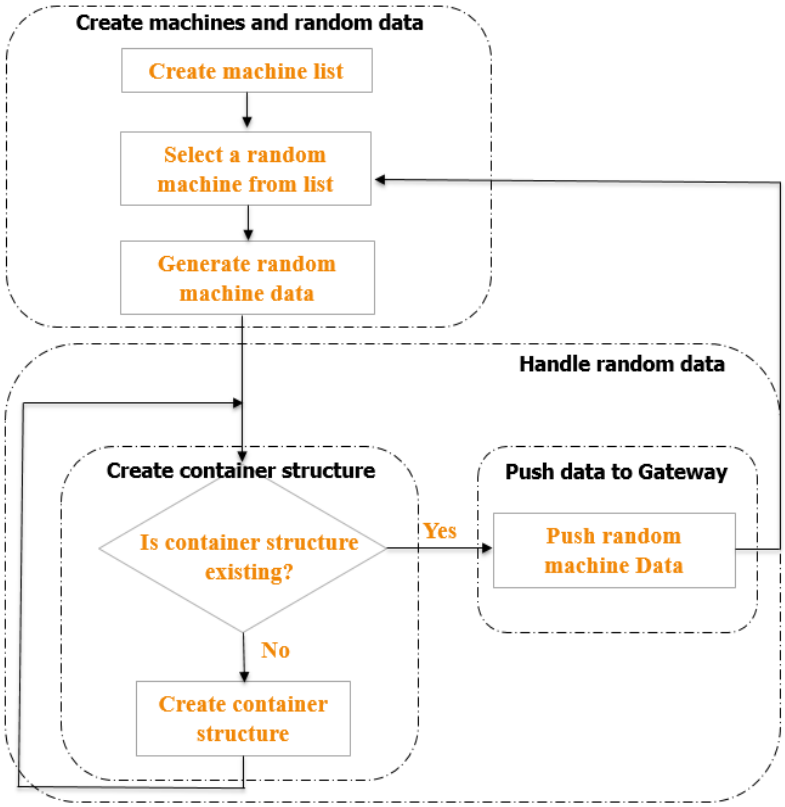

Abbildung 7: Programmablaufplan zur Erzeugung einer emulierten Maschinenumgebung

Die Erstellung der Maschinen geschieht im Block **Create machine list**, woran sich die zufällige Auswahl eines dieser Maschinen anschließt. Darauffolgend werden die zufälligen Nachrichten in **Generate random machine data** erzeugt. Es wird zunächst überprüft, ob die Maschine bereits eine Containerstruktur für 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 für die Container ein Label gewählt. Dies ist für das später behandelte Backend wichtig. Die Routine startet, dann bei der zufälligen Auswahl der nächsten Maschine (Block **Select random machine**), von neuem.

Empfangen der emulierten Maschinennachrichten und senden an das Frontend

Die emulierten Maschinennachrichten müssen, wie in der realen Umgebung später auch, von einem Backend empfangen und an das Frontend weitergeleitet werden. Dazu wird das Ablaufdiagramm in Abbildung **??** betrachtet.

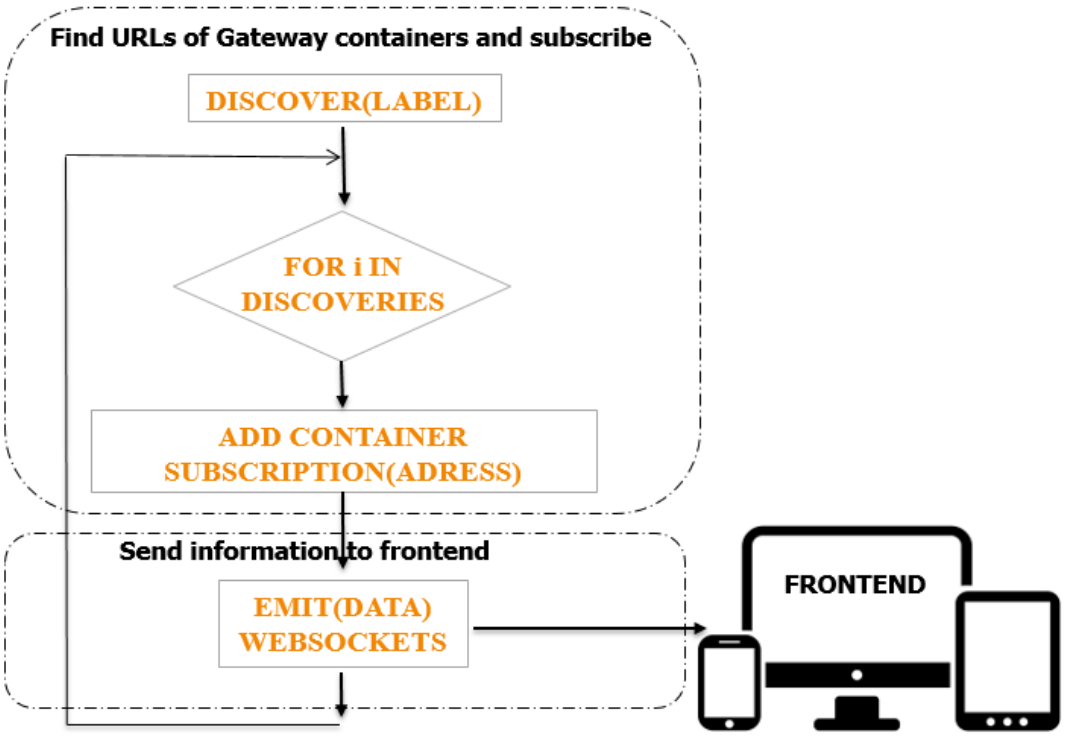

Abbildung 8: Programmablaufplan des Backendes zur Vermittlung der Maschinendaten an das Frontend

Zuerst werden im Block **DISCOVER(LABEL)** alle Adressen der Container mit dem angegeben Label gesucht und in einer Liste zurückgegeben. Die Label wurden dabei für alle Nachrichtencontainer innerhalb der emulierten Umgebung gleich gewählt, sodass alle erkannt werden. Anschließend wird mit einer Schleife über 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.

7.1.2 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 über die tatsächlichen Namen der Komponenten.

Extraktion der RDF-codierten Maschinennachrichten

Die Extraktion der Maschinennachrichten wird im folgenden Codeauschnitt implementiert.

```
1         #create graph variable
2         msg_graph = Graph()
3         #append data, which is in json-ld(in a graph) to our graph
4         msg_graph.parse(data=data, format='json-ld')
5
6         #get adress of message
7         msg_uri = msg_graph.value(predicate=RDF.type,
8         object=self.ns2.Message)
9
10        # extract data from graph branches
11        SID = msg_graph.value(msg_uri, self.ns2.SID)
12        RID = msg_graph.value(msg_uri, self.ns2.RID)
13        TRN = msg_graph.value(msg_uri, self.ns2.TRN)
```

Code Listing 1: Extraktion der RDF codierten Maschinennachrichten

Dabei werden die bereits in der Theorie erläuterten 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 erläutert, von der Subskiptionsfunktion geliefert werden und als format json-ld angegeben wird, da die Maschinendaten auch in diesem Format gesendet werden. Anschließend wird dem Graphen der Maschinendaten, die gesendete Nachricht mit *value* entnommen. In diesem liegen wiederum die für das Frontend relevanten Information SID (Sender), RID (Empfänger) und TRN (Nachricht), welche ebenfalls mit *value* entnommen werden.

Extraktion der tatsächlichen Komponentennamen

Weiterhin muss für die richtigen Namen der Komponenten auf ein Manifest zurückgegriffen 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 können.

Programmablaufdiagramm

Der Teilbereich "Find URLs of Gateways and subscribeïm Ablaufdiagramm in Abbildung **??** wurde bereits für die Demo Umgebung behandelt und funktioniert analog. Die beiden oben behandelten Zusatzroutinen können **??** 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.

```
1        msg = {
2              'sid': SID,
3              'value': TRN,
4              'rid': RID,
5              'type': 'Message:',
6              'devices': dev_list
7         }
```

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() angefügt, welches eine Nachricht sendet und noch nicht in der Liste vorhanden ist.

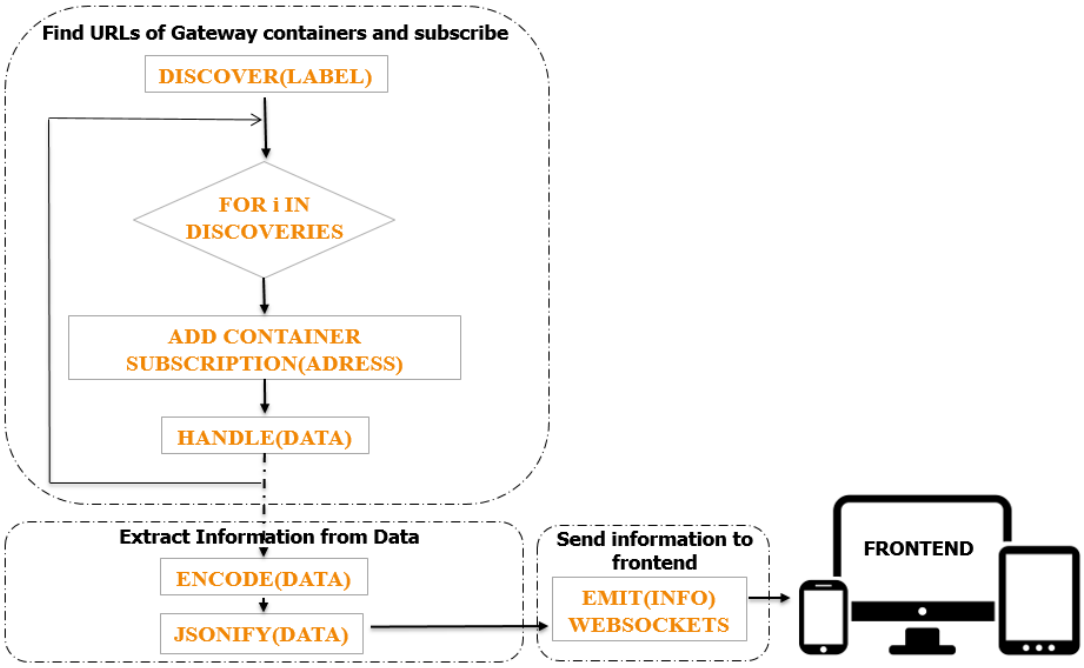

Abbildung 9: Programmablaufplan des Backendes zur Vermittlung der Maschinendaten an das Frontend

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

7.2 Frontend

8 Zusammenfassung

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

9 Ausblick

10 Anhang