Conference Paper Title*

*Note: Sub-titles are not captured in Xplore and should not be used

1st Justin Frommberger
Interaktionstechnik und Design
Hochschule Hamm-Lippstadt
City, Country
email address or ORCID

2nd Jonas Gerken
Interaktionstechnik und Design
Hochschule Hamm-Lippstadt
City, Country
email address or ORCID

3rd Benedikt Lipinski
Interaktionstechnik und Design
Hochschule Hamm-Lippstadt)
Soest, Deutschland
benedikt.lipinski@stud.hshl.de

4th Phillip Wagner
Interaktionstechnik und Desgin
Hochschule Hamm-Lippstadt
City, Country
email address or ORCID

***Abstract*—This document is a model and instructions for LaTeX. This and the IEEEtran.cls file define the components of your paper [title, text, heads, etc.]. *CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.**

***Index Terms*—component, formatting, style, styling, insert**

Identify applicable funding agency here. If none, delete this.

I. Parts of Integration

II. Server

In diesem Kontext spielt der Server eine sehr wichtige Rolle in der Kommunikation zwischen den ausführenden Parts des Projektes. Durch den Server und seine Strukturen wird letztendlich erst eine Plattform geschaffen, die allen Fahrzeugen und den Kunden (Usern) eine Möglichkeit bietet, eine Verbindung untereinander zu schaffen und weitere Aufgaben zu erledigen. Konkret waren die Aufgaben des Servers:

Anmeldung von Clients
Das nächstgelegene Fahrzeug finden
Jedem Client eine eindeutige ID zuordnen
Übermittelt Fahrzeugdaten an den Kunden
Interne Verarbeitung einer Fahrzeugbestellung

A. Anmeldung von Clients

Aufgabe des Servers ist es, eine Anmeldung von Clients zu ermöglichen, um einerseits nur aus dem Pool der aktuell aktiven, freien Fahrzeuge auszuwählen und andererseits einer unbekannten Menge an Fahrzeugen und Kunden die Möglichkeit zu bieten, am Angebot teilzuhaben. Zu den Clients gehören sowohl die Kunden (User), wie auch alle Fahrzeugtypen. Damit sind alle Servicefahrzeuge mit den Unterkategorien: Police, Firefighter, Ambulance gemeint und zuletzt auch Fahrzeuge der Kategorie Taxi.
Um auf eine unbekannte Anzahl an sich zu registrierenden Clients reagieren zu können, muss der initiative Schritt durch den Client erfolgen. Zur Registrierung sendet der Client eine Nachricht mittels MQTT-Protokoll über die Adressen des Users oder der Fahrzeugtypen mit dem ersten Teil ”adresse = hshl/mqtt_exercise/” und der Endung des jeweiligen Fahrzeugtypen, also: ”user,taxi,police,firefighter,ambulance” an den Server. Dieser verarbeitet die Nachricht bei Erhalt in der Funktion

```
def receive()
    ....
```

```
        messageprocessing(temp)
        ....
```

wobei die Aufgabe der Funktion *receive()* eher die generelle Verarbeitung der MQTT-Nachricht darstellt und nicht die Zuordnung der Nachricht zu einen bestimmten Anwendungszweck. Dieser wird anschließend durch den Aufruf der Funktion *messageprocessing* und eine Teilung des übergebenen Arrays in die Informationen zu Inhalt und Adresse der Nachricht erreicht. Um ein Auslesen der Nachricht überhaupt zu ermöglichen ist es notwending, die empfangene Nachricht erst in das Json-Format zu codieren, um anschließend die Vorteile einer Verarbeitung mit Json-Datasets zu nutzen. Dies wird mit der Textzeitle

```
def messageprocessing(msg)
        json.loads(str(msg[1]))
```

erreicht.
Zur besseren Identifizierung und Klassifizierung als Registrierungs-Nachricht, wird in dieser die ID des Clients durch das Wort ”register” ersetzt. Dies dient- wie schon erwähnt- einerseits der besseren Einordnung und andererseits half das Lesbar-Halten von Nachrichten für Menschen bei der Entwicklung ungemein. Einen negativen Einfluss auf den erfolgreichen Ablauf der Registrierung des Clients hat dies nicht, da eine ID erst mit Antwort des Servers an den Client vergeben wird. Die Nachricht, die ein Client zur Registrierung/Anmeldung senden muss, sieht wie folgt aus:

```
    data={
    "id": "register",
    "name": name,
    "coordinates":coor
    }
```

Zudem wird in der internen Verarbeitung ein neuer Kanal für die weitere Kommunikation mit dem Client geschaffen, sodass eine direkte Kommunikation mit diesem möglich ist, ohne dass andere Clients hierdurch beeinträchtigt werden. Die Adresse des neuen Kanals wird unter Zuhilfenahme der durch den Server in den Funktionen *regestrationUser(data)* und *registrationCar(data, type)* vergebenen ID geöffnet und mittels einer Nachricht auf dem Kanal ”hshl/mqtt_exercise/user/back” an diesen zurück gesendet.
Anhand des Quellcodes für das Registrieren des Users wird gezeigt, wie die Vergabe einer neuen ID und das Einspeichern des Clients in den Server funktioniert. Dies ist ganz ähnlich für das Vorgeben bei der Registrierung von Fahrzeugen mit dem einzigen Unterschied, dass in diesem Fall mittels einer Separierung durch den Übergabewert ”type” die einzelnen Fahrzeuge unterschieden werden können. Des weiteren wird bei der Registrierung der Fahrzeuge noch der Status ”free” vergeben.
Zuerst wird durch den Aufruf der Funktion *findid(user)* die kleinste noch freie ID aus der Liste der angemeldeten Fahrzeuge gesucht, indem die höchste vergebene ID gesucht wird und um einen Zähler höher zurückgegeben wird. Anschließend wird in der Methode *registrationUser(data)* durch ein weiteres Durchlaufen der Liste geprüft, ob bereits ein User mit demselben Namen vorhanden ist und bei negativem Ergebnis in die Liste aller User eingetragen.
Der User bekommt abschließend auf dem Rückkanal eine Nachricht mit seiner eindeutigen ID.

B. Bestellen eines Fahrzeugs

Nach erfolgreicher Registrierung ist es für den Kunden möglich, Fahrzeuge zu bestellen und für Fahrzeuge ist es möglich, durch einen Kunden bestellt zu werden. In diesem Fall spielt nun der Server eine verbindende Rolle, indem er eine Anfrage des Kunden entgegennehmen kann und diese an ein von ihm ausgewähltes Fahrzeug weiterleitet. Die Wahl des passenden Fahrzeugs trifft hierbei der Server, da nur er die Positionen aller Teilnehmer kennt und somit das nächstgelegene Fahrzeug auswählen kann.
Eine Anfrage durch einen Kunden wird unter dem im Vorfeld bei der Regestrierung neu geöffneten Kanal in Verbindung mit der ID des Kunden und einer Nachricht mit einem Inhalt, der Informationen über den Kunden, den Koordinaten des Kunden und dem gewünschten Fahrzeugtyp, gestartet. Beispielsweise kann durch den Kunden $ID : 0, Name : Peter, Koordinaten : 2, 4$ unter der $Adresse =$ hshl/mqtt_exercise/user/[ID] mit folgender Nachricht ein Taxi bestellt werden

```
data = {
        "type": "taxi",
    "id": id,
    "coordinates": coordinates
    }
```

Intern verarbeitet der Server die Anfrage des Kunden zuerst, indem er aus dem Type die richtige Liste an die Funktion *findnextcar(gpsUser, car)* übergibt, die das nächstgelegene Fahrzeug des Typs Taxi durch die Koordinaten, die durch den Kunden übermittelt wurden findet.
Nach erfolgreicher ermittlung des nächstgelegenen fahrzeugs wird dem kunden der datensatz des fahrzeugs auf dem kanal hshl/mqtt_exercise/user/”id”/order/back , wobei ”id” die kundenspezifische id des antragstellers darstellt und somit nur der Kunde antwort erhält, der in diesem moment auch ein Fahrzeug bestellt hat. Als weiteren wichtigen schritt ist sorge dafür zu tragen, das die einhaltung der anforderung **F-S02** und durch die anwendung von **F-S08** nicht verletzt wird und nur fahrzeuge vergeben werden, die den status free besitzen, das wird erreicht indem der status eines freien fahrzeugs durch den status busy ersetzt wird, wenn er an einen kunden verschickt wird.

1) Finde das nächstgelegene Fahrzeug: Um das Fahrzeug zu finden, das dem kunden am nächsten gelegen ist, muss zu erst die vorhandene karte abgebildet werden können, dies wird erreicht in dem planquadrate eingezeichnet werden. Durch die festgelegten planquadrate lässt sich anschließen durch iterierern der planquadrate und abgleichen der liste der fahrzeuge, das gesuchte fahrzeug finden. So in der Vereinfachten theroie, allerdings entstehen bei der Umschlüsselung auf den konkreten fall diverse probleme, die das finden des nächst gelegenen

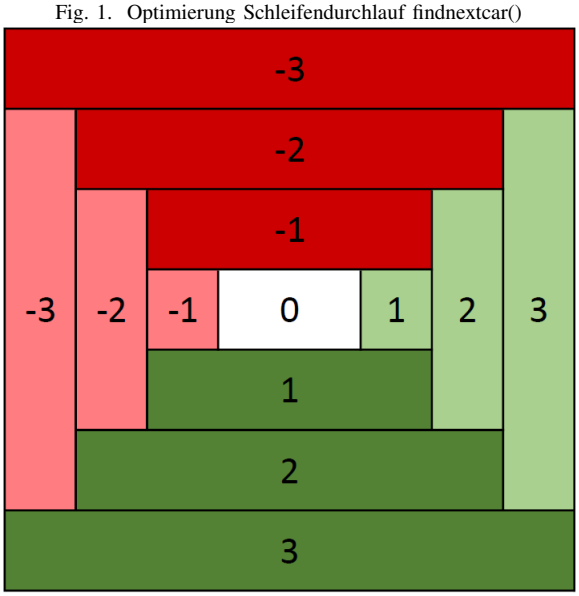
fahrzeug und nicht irgend eines fahrzeugs komplizierter gestalten als zu anfang angenommen.
durch den Umstand, dass der mittelpunkt der suche, bedingt durch den variablen standort des Kunden nicht zwingend der Mittelpunkt der Karte darstellen muss, kam als optimalste lösung ein ringartiges erweitern der Suche um den standort des Kunden in frage dieses wurde mittels schleifen iteration geschaffen.
Die erste Schleife mit der laufvariable ”k” gib an um welchen ring,um den Aktuellen standot des Kunden es sich handelt. Die 2. Schleife steht für die y-Achse der Karte und gibt somit die reihen der Quadrate an, sie iteriert von einem starpunkt an, der sich auszeichnet durch die gegenzahl von der ringstufe k minus einem zähler $i_{starpunkt} = (k-1)*(-1)$ bis zu dem endpunkt der durch k plus einem zähler beschrieben werden kann $i_{endpunkt} = k+1$. Die X-Achse wird durch einen weiteren schleifen durchlauf abgebildet und spiegelt die spalten der Karte wieder. Die x-Achsen Schleife läuft vom startpunkt $j_{starpunkt} = i$ also der linken oberen ecke des aktuellen rings bis zum endpunkt $j_{endpunkt} = k+1$ also der rechten unteren ecke des aktuellen rings. Eine letze schleife vergleicht letztendlich die angegebene position mit der liste der angemeldeten fahrzeuge ab, um zu prüfen ob sich an der aktuellen koordinate ein fahrzeug befindet.

```
def findnextCar(gpsUser,car):
    cancle = 0
    for k in range(int(gpsUser.split(";")[0]),5):
        for i in range((k-1)*(-1), k+1):
            print("i ist:"+str(i))
            for j in range(i, k+1):

                print("j ist:"+str(j))
                for c in range(0,len(car)):
                    print("search for car: "+ str(car[c]))
...
```

Ein nachteil den der beschriebene code inne hat ist allerdings, das bei jedem durchlauf nur um einen ring erweitert wird.In wirklichkeit wird hingegen nicht ein ring geprüft sondern ein quadrat jedesmal um einen ring erweitert, was zum nachteil hat das bei jedem durchlauf auch jedes mal bereits geprüfte felder ein weiteres mal geprüft werden, was ein deutlich hemmdenden einfluss auf die laufzeit des programmes hat und letztendlich auch ein unnötiges unterfangen ist. demnach ist bei einer möglichen Optimierung darauf zu achten, bereits geprüfte felder nicht ein weiteres mal zu überprüfen. Eine mögliche umsetzung wäre die vorhandene schleifen für die X- Achse und die Y- Achse jeweils 2 Schleifen aufzuspalten, wobei in diesem fall die 1. schleife den negativen bereich auf der linken und oberen seite des Standortes prüfen würde und die 2. schleifen den Rechten und unteren positiven bereich der Karte. Anschließend wird als letzter schritt innerhalb der letzten schleife noch abgeglichen ob sich ein freies Fahrzeug mit den passenden koordinaten auf der entsprechenden aktuell geprüften position befinden beispielhaft dargestellt für den bereich $-y,+x = \{-,+\}$. Aus der Tabelle II-B1 kann in verbindung mit den aktuellen Schleifen variablen die aktuell zu prüfende position bestimmt werden, wie in Beispiel positionsabgleich zusehen. Zudem wird noch sicherheitshalber ein counter erstellt, der einen exit nach 3 durchläufen ohne ergebnis ermöglicht.

Fig. 1. Optimierung Schleifendurchlauf findnextcar()




```
#Bespiel Positions-Abgleich#
...
elif int(car[c][2].split(";")[0]) == -i
and int(car[c][2].split(";")[1]) == j:
        if car[c][3]== "free":#Requirements: F-S11
        return car[c]
    else:
        c=c-1
        cancle = cancle+1
```

TABLE I
ABGLEICHTABELLE FÜR DIE POSITIONSBESTIMMUNG DER FAHRZEUGE

–,–	–,-	–,=	–,+	–,++
-,–	-,-	-,=	-,+	-,++
=,–	=,-	=,=	=,+	=,++
+,–	+,-	+,=	+,+	+,++
++,–	++,-	++,=	++,+	++,++

Als Rückgabewert wird der datensatz des Fahrzeugs, dass dem Kunden am nächsten liegt an den Aufruf zurück gegeben, dieser kann anschließend an den Kunden übermittelt werden.

REFERENCES

[1] QUELLEN