Entwurfsdokument - VINJAB: VINJAB Is Not Just A Boardcomputer

Vladimir Bykovski Jonas Haas David Grajzel Nicolas Schreiber
Valentin Springsklee Yimeng Zhu

16. Dezember 2015

Inhaltsverzeichnis

1 Vorgänge

1.1 Datenrepräsentanten

Ein Hauptteil der Kommunikation auf dem Server wird über einen zentralen Bus mit Broker gehandelt. Auf ihm gibt es verschiedene Veröffentlicher und Abonnenten (sogenannte Publisher und Subscriber) die alle eine abstrakte Klasse 'BusAccess' erweitern.

Alle Nachrichten sind Objekte von Klassen welche die abstrakte Klasse 'Message' erweitern und implementieren. Diese ist Serializable, also auch die Unterklassen. Hier wurden abstrakte Klassen gewählt da mehrere Funktionen zentral implementiert werden andere jedoch, wie z.B. Getter und Setter, von dem Nachrichtentyp abhängen.

Die Rohnachrichten kommen unter anderem von dem Blutooth-Modul, dass als Publisher die erhaltenen Daten auf den Bus legt.

Auf dem Server gibt es für jedes Endgerät ein Objekt, welches das physische Gerät repräsentiert. Diese Objekte sind Instanzen einer Klasse 'Proxy' und abonnieren die benötigten Signale auf dem Nachrichtenbus.
Außerdem besitzt jeder Proxy ein eigenes Objekt der Klasse 'PeerConnection' das für die Verbindung zwischen Server und Client zuständig ist.
Die subscribten Daten werden vom Proxy über die PeerConnection an das Endgerät verschickt und wenn Daten von der PeerConnection, also vom Endgerät, ankommen werden diese auf den Bus gelegt.

Aggregierte Funktionen und virtuelle Sensoren sind Instanzen einer Klasse 'VirtualSensor'. Sie abonnieren die benötigten Signale, berechnen neue Werte sobald die Signale ankommen, und legen diese dann als neues Signal wieder auf den Bus.

Zentral gespeichert werden alle Daten in einer LevelDB-Datenbank. Ihr vorgeschaltet ist eine Klasse die eine Schnittstelle zwischen Datenbank und Bus darstellt. In dieser werden alle Signale und Konfigurationsnachrichten abonniert und dann in der Datenbank gespeichert. Außerdem empfängt diese Klasse auch Befehle über den Bus und kann dann z.B. gewünschte Daten aus der Datenbank auf den Bus legen.

Auf dem Endgerät wird der gleiche Datenbus genutzt wie auf dem Server. Auch hier gibt es einen Proxy mit einer PeerConnection.

Dashes sind dann Subscriber des von ihnen angezeigten Signals. Sie erweitern die Klasse Widget

1.2 Chain of Responsibility

Die Werte der Sensoren und die Werte der aggregierten Funktionen können sich zu beliebigen Zeitpunkten ändern. Diese sind als subscribable Value gespeichert und so wie es

im Publish-Subscribe-Bus Entwurfsmuster vorgesehen ist, werden alle Objekte, die als Beobachter vorhanden sind bei Änderungen der subscribten Werte benachrichtigt. Die Beobachter sind die Endgerät-Proxys. Wenn diese eine Nachricht empfangen (es sind neue Werte vorhanden), schicken Sie diese Information nach dem Chain of Responsibility Prinzip weiter an die PeerConnection. Da die Nachrichten alle Serializable sind kann sehr einfach eine Zeichenkette aus dem Objekt erzeugt werden. Die PeerConnection erhält dann diese Zeichenketten und leiten sie über die bestehende Netzwerkverbindung (sei es entweder LAN oder Internet) weiter. Dafür wird der RTC Data Channel genutzt. Die ankommende Zeichenketten auf dem Endgerät werden durch das Serializable wiederhergestellt und das Endgerät kann die Nachrichtenobjekte behandeln. Die werden genauso dargestellt, wie die losgeschickt worden (vom Endgerät-Proxy). Der Bus leitet dann alle Informationen an die passenden Dashes oder die Einstellungen weiter.

2 Klassendiagramme

2.1 Bus

In dem folgenden Diagramm sieht man den Aufbau des Busses, mit dem Interface für die Publisher und Subscriber sowie den verschiedenen Typen von Messages.

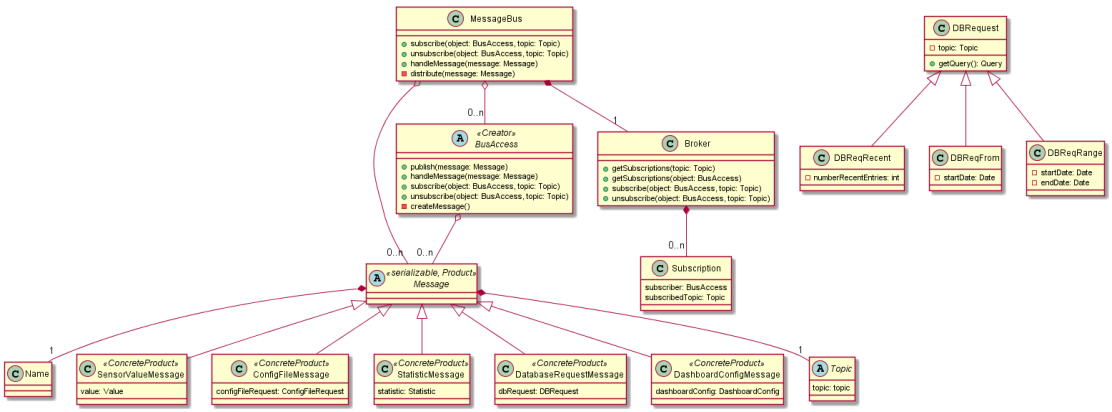


Abbildung 2.1: Bus Klassendiagramm

2.2 Virtual Sensors

Hier sieht man den Aufbau der Virtuellen Sensoren.

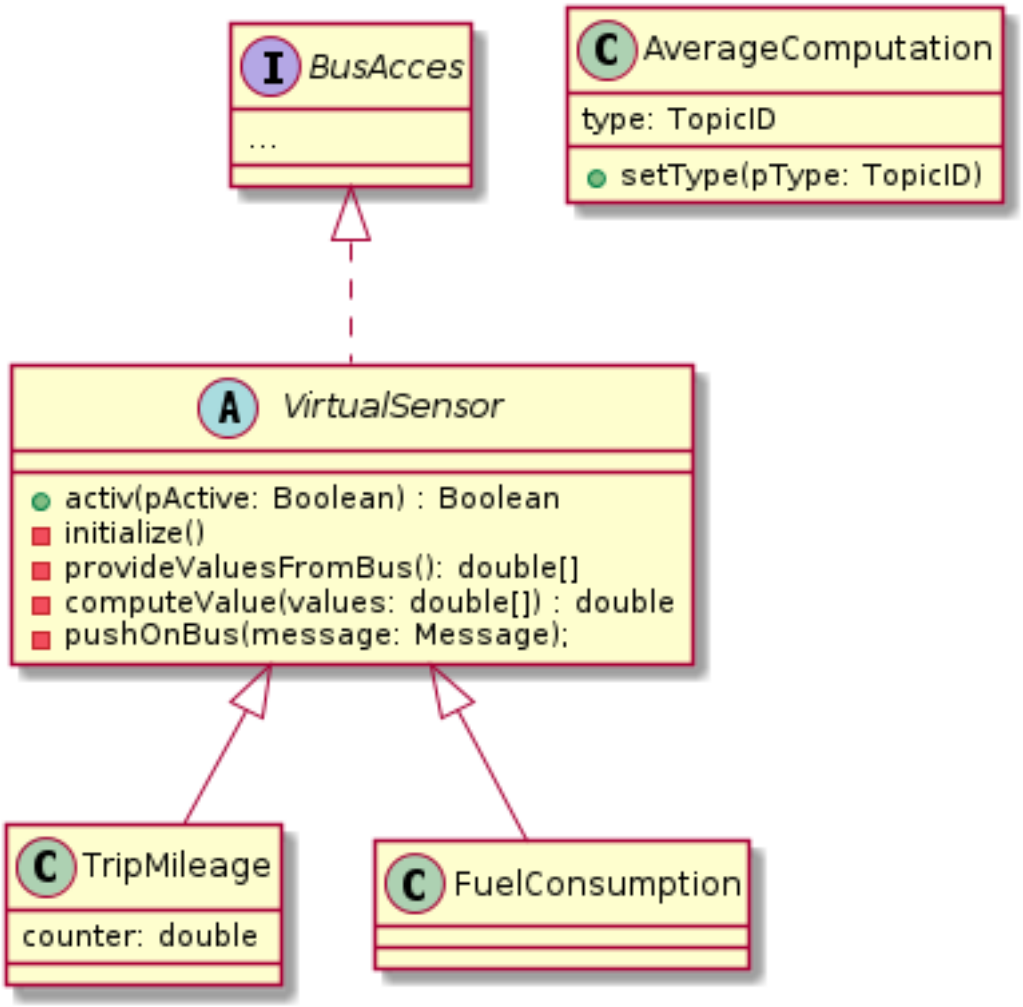
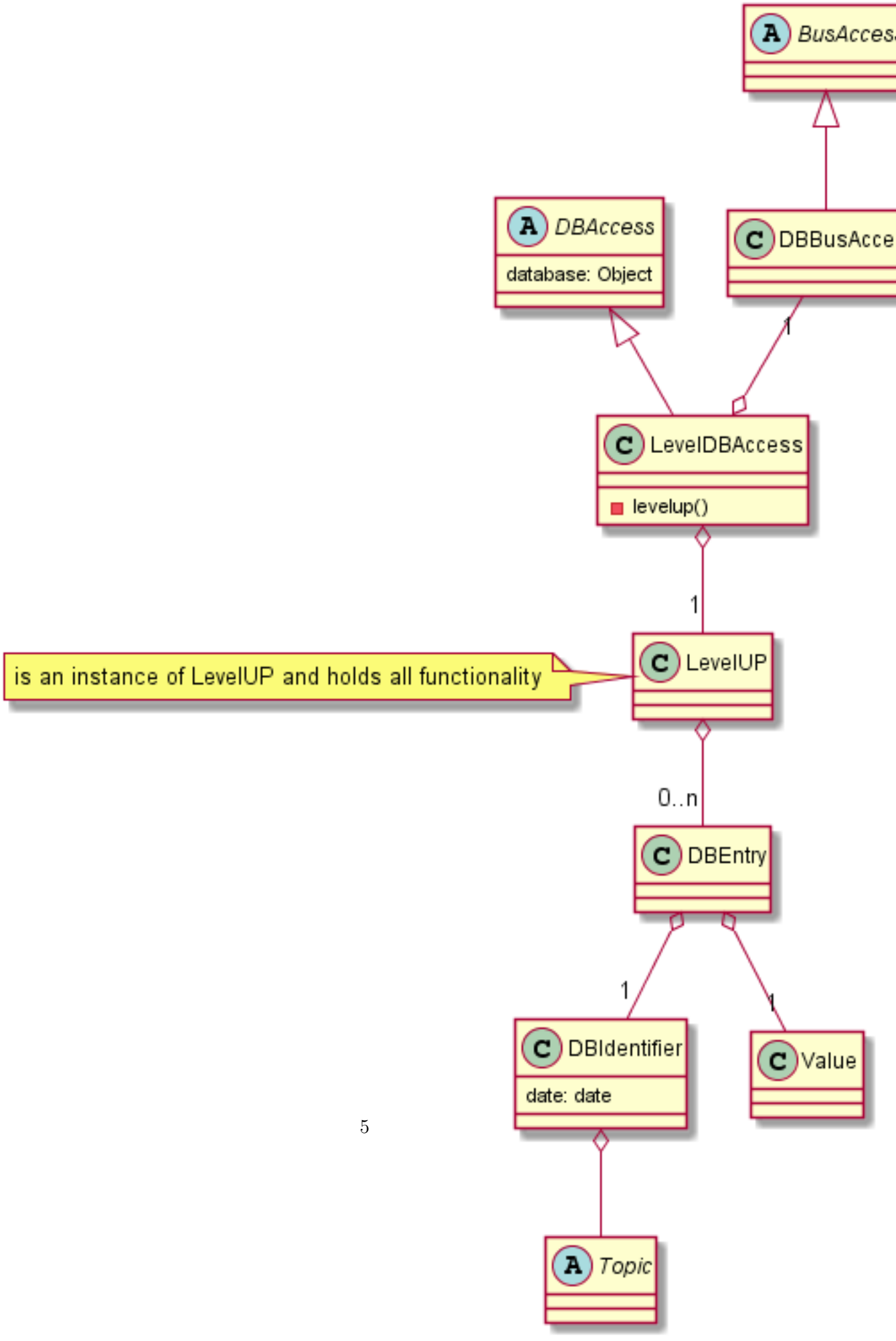


Abbildung 2.2: Virtuelle Sensoren Klassendiagramm

2.3 Database Access

Im folgenden Diagramm ist der Aufbau der Datenbank und des Datenbankzugriffs zu sehen.


BusAcces
DBAccess
database: Object
DBBusAcce
1
LevelDBAccess
levelup()
1
is an instance of LevelUP and holds all functionality
LevelUP
0..n
DBEntry
1
1
DBIdentifier
date: date
Value
Topic

2.4 User Interface

In dem folgenden Diagramm sieht man den Aufbau der UI auf der Seite des Terminals

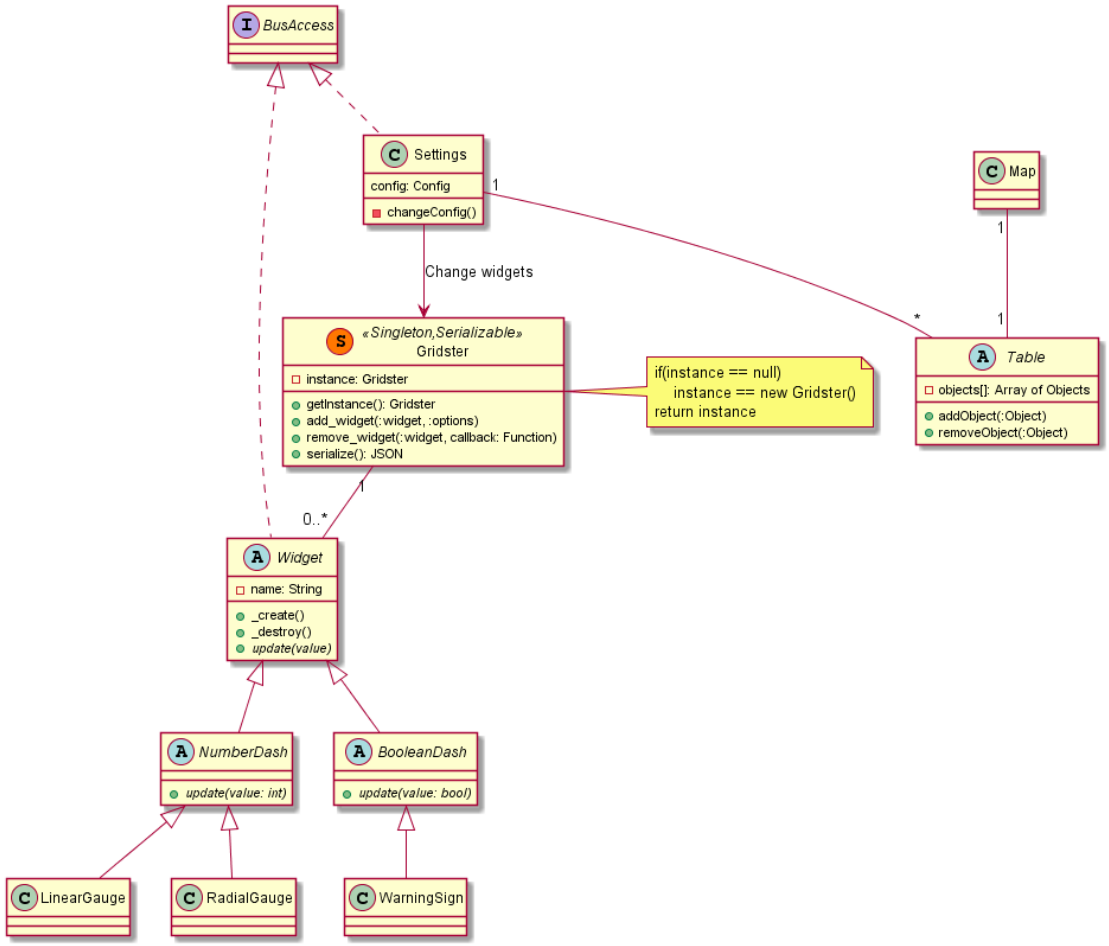


Abbildung 2.4: UI Klassendiagramm

2.5 Parking Sensor

Hier ist zu sehen wie das Rückfahrassistenzsystem aufgebaut ist. Siehe auch

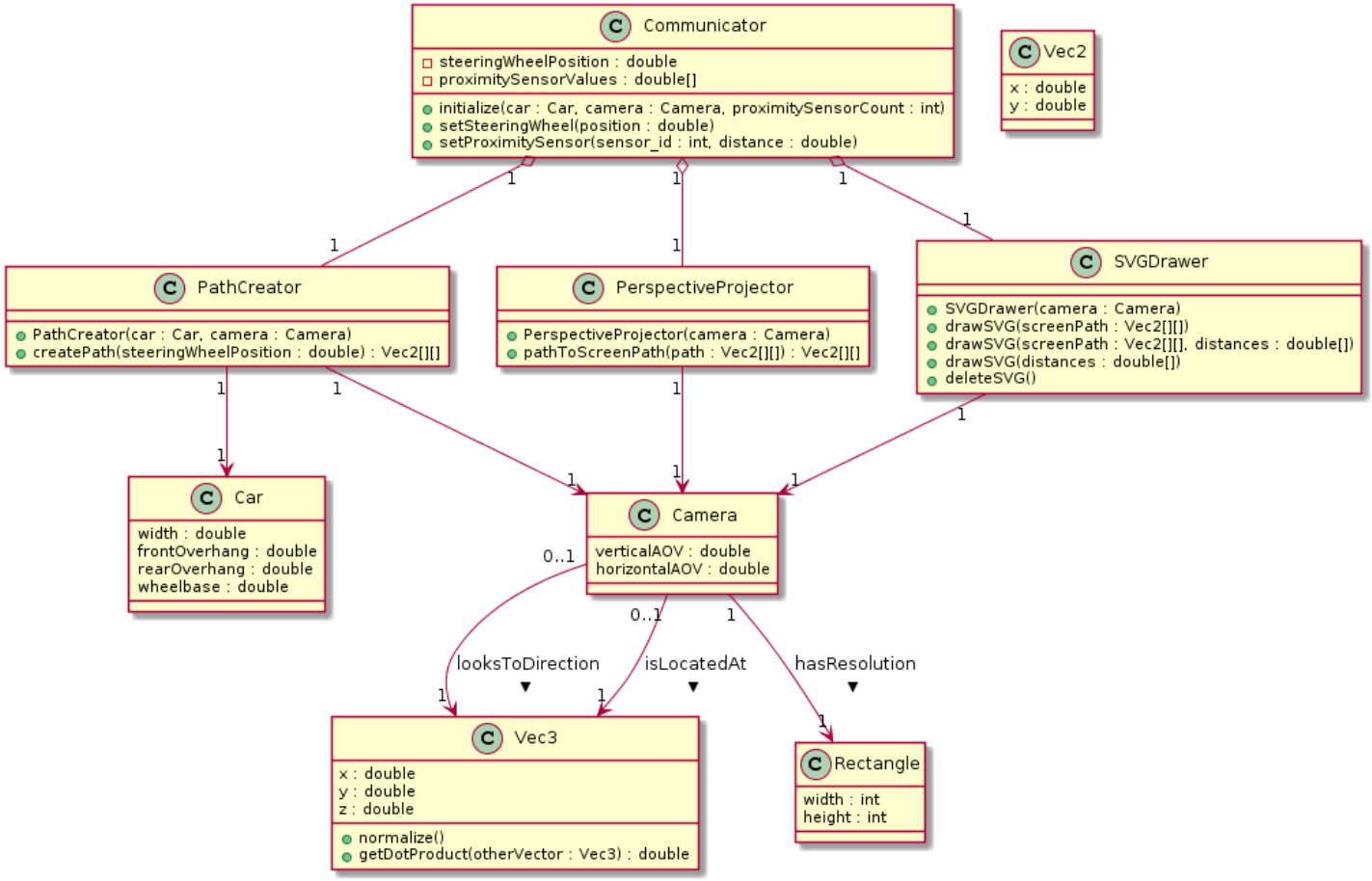


Abbildung 2.5: Rückfahrsystem Klassendiagramm

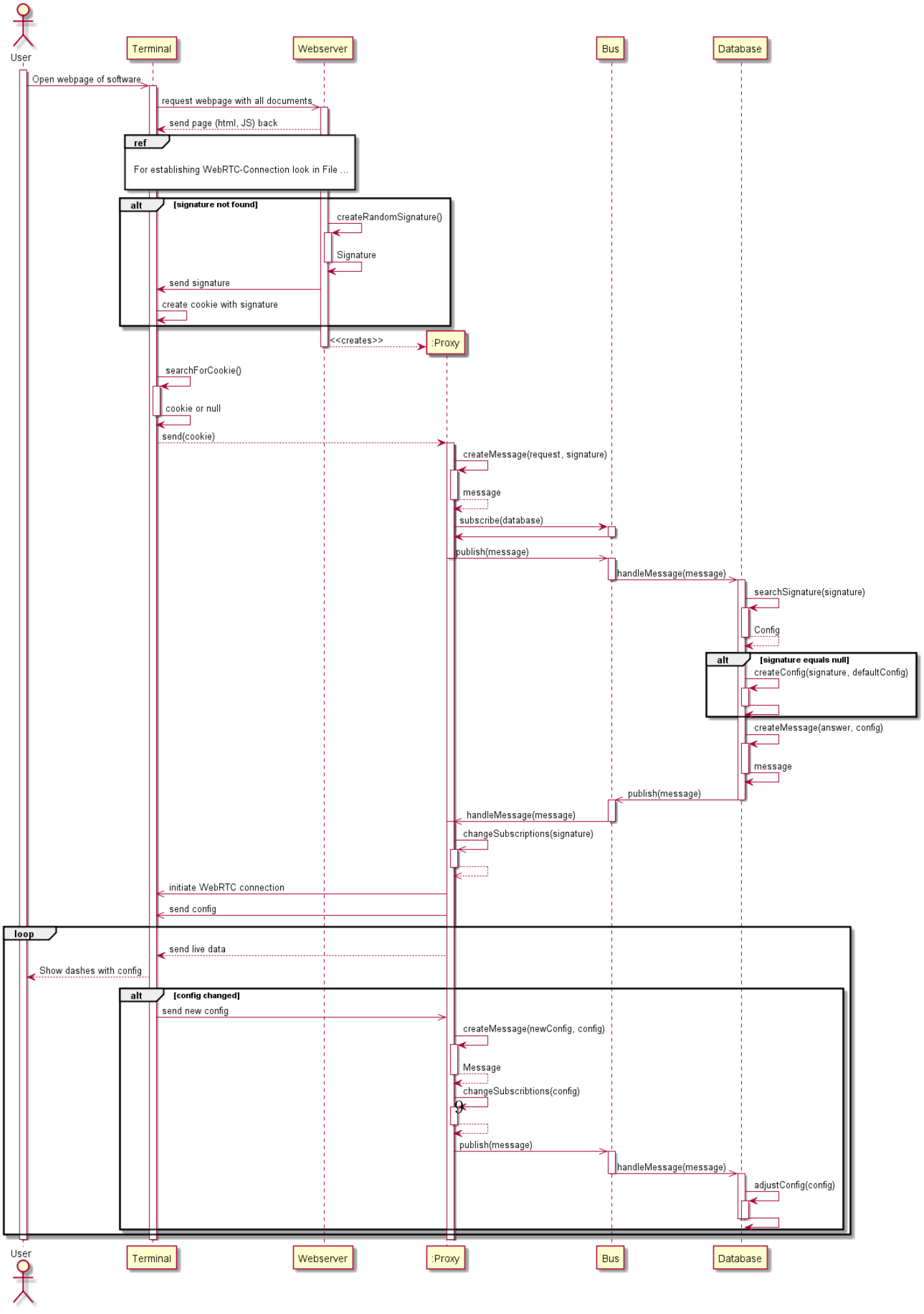
3 Sequenzdiagramme

3.1 Terminal connect

In dem folgenden Diagramm sieht man den Ablauf vom Verbinden eines Endgeräts mit dem Server bis zur Live-Übertragung der Signale und die Änderung der Anzeigekonfiguration auf dem Endgerät.

Hierbei handelt es sich beim Terminal um das Endgerät auf dem über einen Browser die Vinjab-Seite aufgerufen wird. Der Befehl openWebpage() vom Terminal zum Webserver nutzt das HTT-Protokoll.

Das Hypertext Transfer Protocol (HTTP, englisch für Hypertext-Übertragungsprotokoll) ist ein Protokoll zur Übertragung von Daten auf der Anwendungsschicht über ein Rechnernetz und gehört dementsprechend zur Internetprotokollfamilie. Es wird hauptsächlich eingesetzt, um Webseiten (Hypertext-Dokumente) aus dem World Wide Web (WWW) in einen Webbrowser zu laden. Es ist jedoch nicht prinzipiell darauf beschränkt und auch als allgemeines Dateiübertragungsprotokoll sehr verbreitet.


User
Terminal
Webserver
Bus
Database
Open webpage of software
request webpage with all documents
send page (html, JS) back
ref
For establishing WebRTC-Connection look in File ...
alt
[signature not found]
createRandomSignature()
Signature
send signature
create cookie with signature
<<creates>>
:Proxy
searchForCookie()
cookie or null
send(cookie)
createMessage(request, signature)
message
subscribe(database)
publish(message)
handleMessage(message)
searchSignature(signature)
Config
alt
[signature equals null]
createConfig(signature, defaultConfig)
createMessage(answer, config)
message
publish(message)
handleMessage(message)
changeSubscriptions(signature)
initiate WebRTC connection
send config
loop
send live data
Show dashes with config
alt
[config changed]
send new config
createMessage(newConfig, config)
Message
changeSubscribtions(config)
publish(message)
handleMessage(message)
adjustConfig(config)
User
Terminal
Webserver
:Proxy
Bus
Database

3.2 WebRTC connection

In dem folgenden Diagramm wird dargestellt wie eine WebRTC-Verbindung zwischen dem Endgerät und dem Webserver hergestellt wird.

WebRTC ist eine Sammlung von Kommunikationsprotokollen und Programmierschnittstellen (API) für die Implementierung in Webbrowsern, die diesen Echtzeitkommunikation über Rechner-Rechner-Verbindungen ermöglichen. Damit können Browser nicht mehr nur Datenressourcen von Backend-Servern abrufen, sondern auch (Echtzeitinformationen) von Browsern anderer Benutzer.

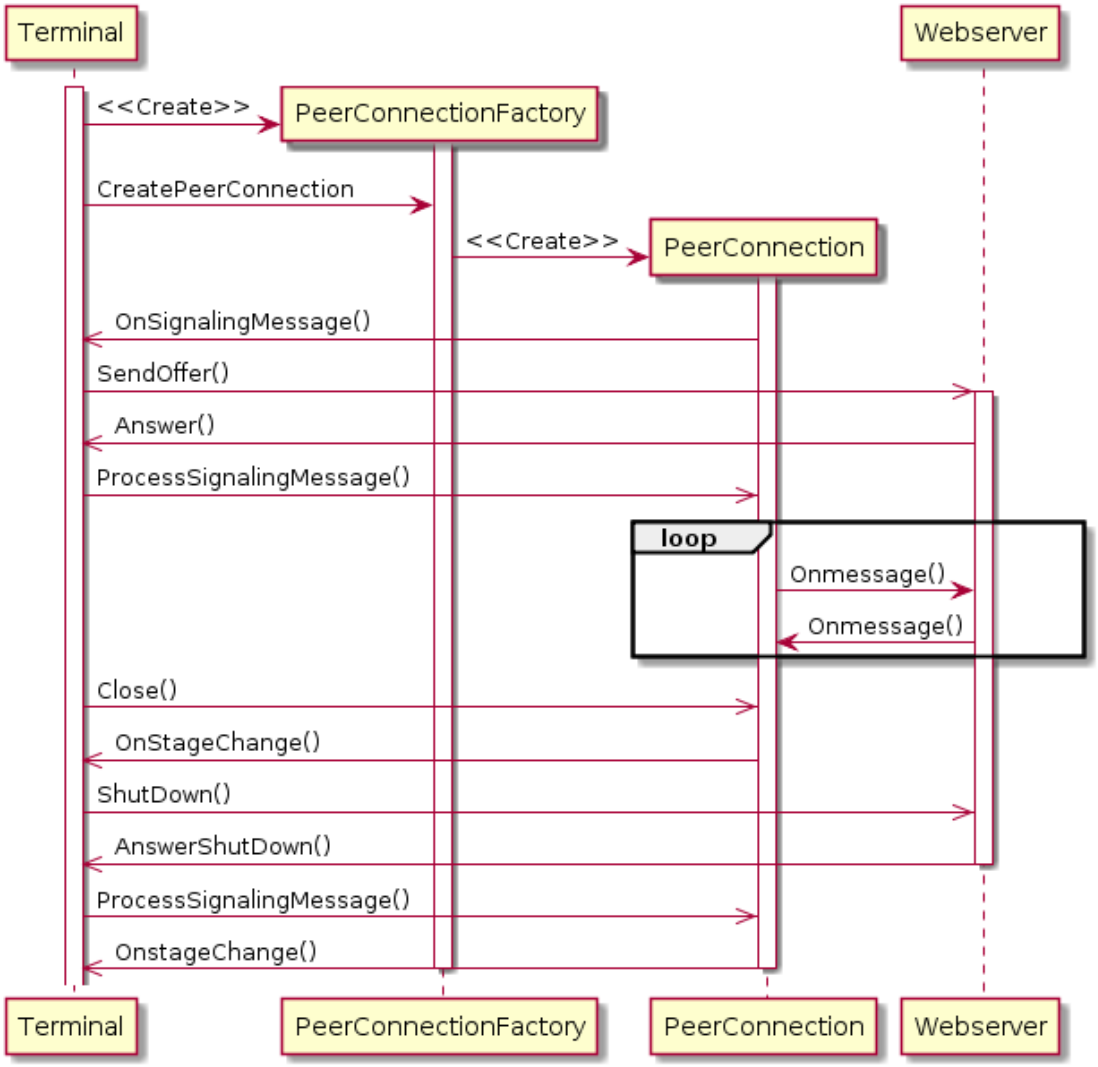


Abbildung 3.2: Verschiedene Dash-Anzeigeelemente.

3.3 Changed config

Nach hergestellter Verbindung befindet sich das System in einer Schleife in der eigentlich nur noch Live-Daten vom Server zum Client geschickt wird. Nur wenn die Anzeigeeinstellungen der Dashes geändert wurden

Abbildung 3.3: Verschiedene Dash-Anzeigeelemente.

3.4 Parkingsensor

Der Ablauf beim Start des Parksensors

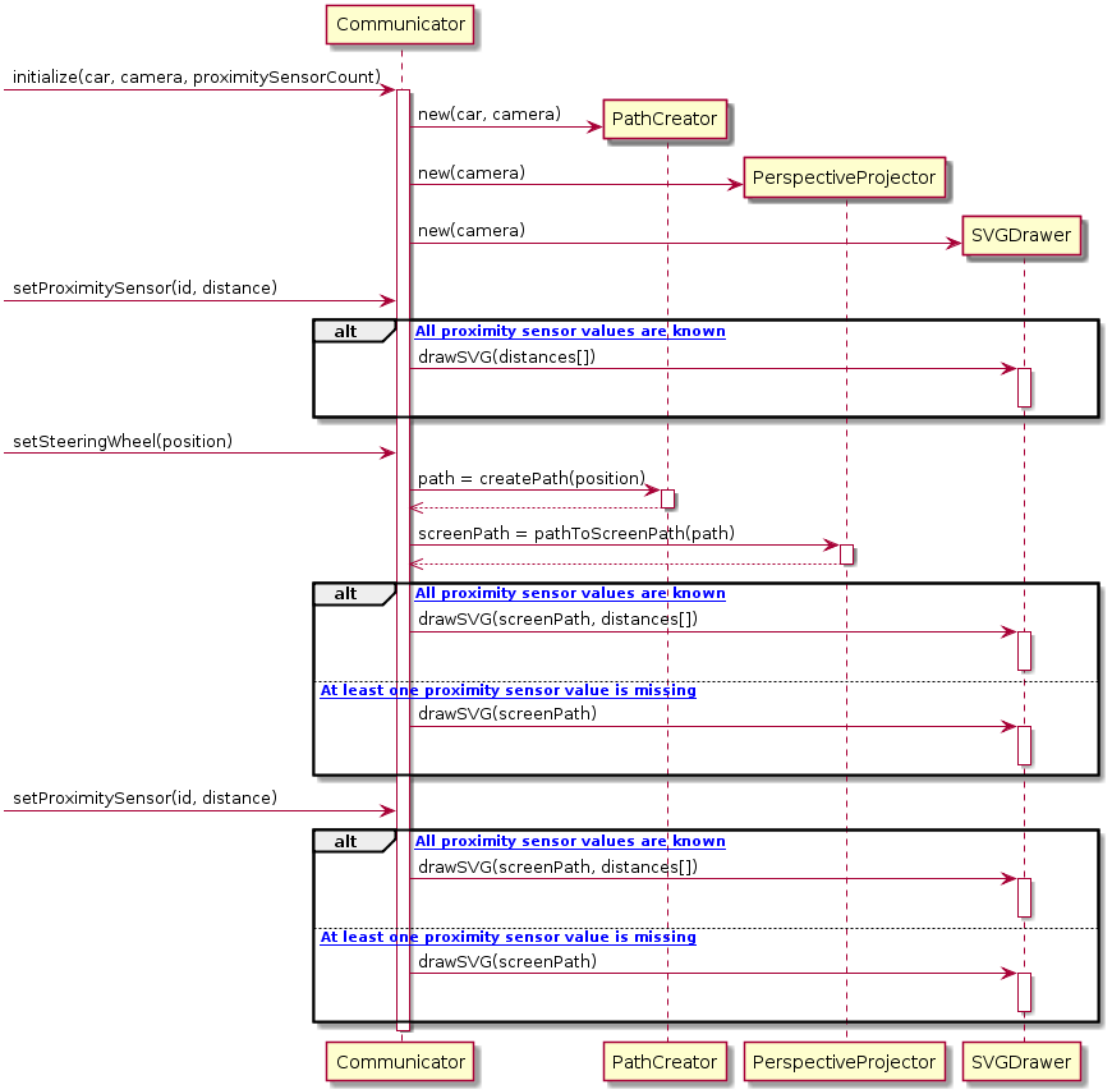


Abbildung 3.4: Verschiedene Dash-Anzeigeelemente.