Technische Universität Braunschweig

ISF

SocialPal

How soon is "now"?

Software-Entwicklungspraktikum (SEP)
Sommersemester 2016

Technischer Entwurf

Auftraggeber:
Technische Universität Braunschweig
Institut für Informationssysteme
Prof. Dr. Wolf-Tilo Balke
Mühlenpfordtstr. 23
38106 Braunschweig

Betreuer: Kinda El Maarry, José María González Pinto

Auftragnehmer:

Name	E-Mail-Adresse
Alexey Ballard	a.ballard@tu-braunschweig.de
Anna-Liisa Ahola	a.ahola@tu-braunschweig.de
Ceyda Gökay	c.goekay@tu-braunschweig.de
David Haase	david.haase@tu-braunschweig.de
Hendrik Garz	h.garz@tu-braunschweig.de
Jasmin Kaya	j.kaya@tu-braunschweig.de
Jonas Lutz	j.lutz@tu-braunschweig.de
Malte-Christian Roesch	m.roesch@tu-braunschweig.de
Marvin Bienek	m.bienek@tu-braunschweig.de
Pawel Sas	p.sas@tu-braunschweig.de

Braunschweig, 29. Juni 2016

Bearbeiterübersicht

Kapitel	Autoren	Kommentare
1	Ceyda Gökay	...
1.1	Ceyda Gökay	...
2	Ceyda Gökay, Jasmin Kaya, Hendrik Garz, David Haase, Alexey Ballard, Marvin Bienek, Anna-Liisa Ahola, Jonas Lutz	...
2.1	Jasmin Kaya	...
2.2	Hendrik Garz	...
2.3	David Haase	...
2.4	David Haase, Anna-Liisa Ahola	...
2.5	Alexey Ballard	...
2.6	Alexey Ballard, Anna-Liisa Ahola	
2.7	Marvin Bienek	...
2.8	Marvin Bienek	...
2.9	Anna-Liisa Ahola	...
2.10	Jasmin Kaya	...
2.11	Hendrik Garz	...
3	Anna-Liisa Ahola, Marvin Bienek, Hendrik Garz, Malte-Christion Roesch, Ceyda Gökay	...
3.1	Anna-Liisa Ahola, Marvin Bienek, Hendrik Garz, Malte-Christion, Roesch, Ceyda Gökay	...
3.2	Anna-Liisa Ahola, Marvin Bienek	...
3.3	Anna-Liisa Ahola, Jasmin Kaya	...

3.4	Marvin Bienek, Hendrik Garz, Malte-Christion Roesch	...
4	Jasmin Kaya, Jonas Lutz	...
5	Pawel Sas, David Haase, Hendrik Garz, Anna-Liisa Ahola, CeydaGokay	...
5.1	Hendrik Garz, Anna-Liisa Ahola, Jasmin Kaya	...
5.2	Hendrik Garz, Anna-Liisa Ahola, Ceyda Gökay	...
5.3	Hendrik Garz, Anna-Liisa Ahola	...
5.4	Hendrik Garz, Anna-Liisa Ahola, Ceyda Gökay, Malte-Christion Rösch	...
5.5	Hendrik Garz, Anna-Liisa Ahola	...
5.6	Hendrik Garz, Anna-Liisa Ahola	...
5.7	Anna-Liisa Ahola	...
5.8	Hendrik Garz, Anna-Liisa Ahola	...
5.9	Pawel Sas, David Haase	...
6	Anna-Liisa Ahola, David Haase	...
6.1	Anna-Liisa Ahola, David Haase	...
6.2	Anna-Liisa Ahola, David Haase	...
7	Jonas Lutz, Malte-Christian Roesch	...

8	Anna-Liisa Ahola, Jasmin Kaya, Pawel Sas, Jonas Lutz	...
9	Alexey Ballard, Jasmin Kaya, Anna-Liisa, Ceyda Gökay	...
9.1	Alexey Ballard, Jasmin Kaya, Anna-Liisa, Ceyda Gökay	...
9.2	Jasmin Kaya, Alexey Ballard	...
9.3	Jasmin Kaya, Alexey Ballard	...
10	Alexey Ballard, Anna-Liisa Ahola, Ceyda Gökay, David Haase, Hendrik Garz, Jasmin Kaya, Jonas Lutz, Malte-Christian Roesch, Marvin Bienek, Pawel Sas	...

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

In diesem Dokument werden unsere Entwurfsentscheidungen dokumentiert. Da es inhaltlich um das Abbilden der grundlegenden Implementierungen geht, muss anhand dieses Dokumentes jeder Softwareentwickler in der Lage sein, das Projekt „SocialPal“ zu entwickeln. Mit Hilfe von verschiedenen Diagrammen soll ein Überblick über das Verhalten und die anfallenden sowie die verwendeten Daten des Projekts „SocialPal“ verschafft werden. „SocialPal“ ist eine Android-Applikation, die durch die Minimierung der Ablenkung durch das Smartphone das sichere Fahren im Verkehr gewährleisten soll. Hierfür wird außerdem eine Engin-Control-Unit (ECU) entwickelt, welche das Navigationssystem im Fahrzeug darstellt. Diese wird für Testzwecke auf einem Computer im Webbrowser simuliert.

Das Dokument ist in zehn Kapiteln inklusive Glossar unterteilt. Das zweite Kapitel befasst sich mit den Produktfunktionen, diese werden detailliert analysiert und jeweils mit einem Sequenzdiagramm abgebildet. Im dritten Kapitel wird mit Hilfe von Komponentendiagrammen ein Überblick über die zu entwickelnden Komponenten und Subsysteme geliefert. Es wird sowohl auf die Komponenten- als auch auf die Schnittstellenspezifikation eingegangen. Im darauf folgenden Abschnitt wird die verteilte Anwendung des Produktes anhand eines Verteilungsdiagrammes visualisiert. Im fünften Abschnitt dieses Dokumentes wird der Implementierungsentwurf mit Hilfe von Klassendiagrammen und Erläuterungen bezüglich diesen anschaulich gemacht. Fortlaufend wird auf die Konfiguration und die Änderungen gegenüber des Fachentwurfs eingegangen. Im letzten Abschnitt wird erläutert, welche Muss-, Soll-, und Kannkriterien die App beziehungsweise der Server erfüllt. Im Speziellen werden die Komponenten angegeben, die die entsprechenden Funktionen implementieren.

Im folgenden Aktivitätsdiagramm (siehe Abbildung 1.1) wird die Kommunikation zwischen dem Benutzer, der App und der ECU abgebildet.

Zu Beginn startet die ECU. Sobald sich ein Mobiltelefon mit installierter „SocialPal“-App im Auto befindet startet die App automatisch und befindet sich anschließend im „Driving“-Modus. Sofern sich der Benutzer im Fahrerbereich befindet verbindet sich die App mit dem Webserver über welchen diese mit der ECU kommuniziert. Ist dies nicht der Fall verbindet sich die App entsprechend nicht.
Eine Auswahl von zwei verschiedenen Benutzerprofilen dient zu Testzwecken, dabei muss sich der Benutzer für ein Profil entscheiden. Fortlaufend kommunizieren die ECU und die App über den Webserver miteinander. Dementsprechend wechselt die App in den entsprechenden Modus, welcher von der durch die ECU übermittelte Geschwindigkeit abhängt. Solange der Fahrer weniger als 30 km/h fährt, verbleibt die App im „Driving“-Modus. Aus Sicherheitsgründen wird ab 30 km/h der „Silent“-Modus aktiviert, welcher das Mobiltelefon in einen stummen Modus versetzt. In diesem werden keine Benachrichtigungen über eingehende Nachrichten und Anrufe angezeigt. Hierzu muss sich das Mobiltelefon des Benutzers im Fahrerbereich befinden, welcher durch die Signalstärke des Beacons ermittelt wird. Anrufe von Favoriten, welche der Benutzer in der App hinzufügen kann, werden immer als dringlich deklariert, dies gilt nicht für Nachrichten. Die Dringlichkeit einer Nachricht wird durch den Naive Bayes Klassifizierungsalgorithmus, welcher eigenständig durch die eingehenden Textnachrichten lernt, ermittelt. Der Nutzer kann im Vorfeld individuell eine Standartantwort für den „Silent“-Modus definieren und auf Wunsch jederzeit deaktivieren. Diese wird bei Eingang von Anrufen oder Nachrichten automatisch an den Anrufer oder Absender gesendet. Sobald ein dringlicher Anruf oder eine dringliche Nachricht empfangen wird, informiert die ECU den Benutzer und ermittelt die nächstgelegene Parkmöglichkeit. Anschließend wird dem Benutzer die Möglichkeit geboten die Navigation zu dieser Parkmöglichkeit anzunehmen oder abzulehnen. Im Falle einer Ablehnung wird die bisherige Route fortgeführt.

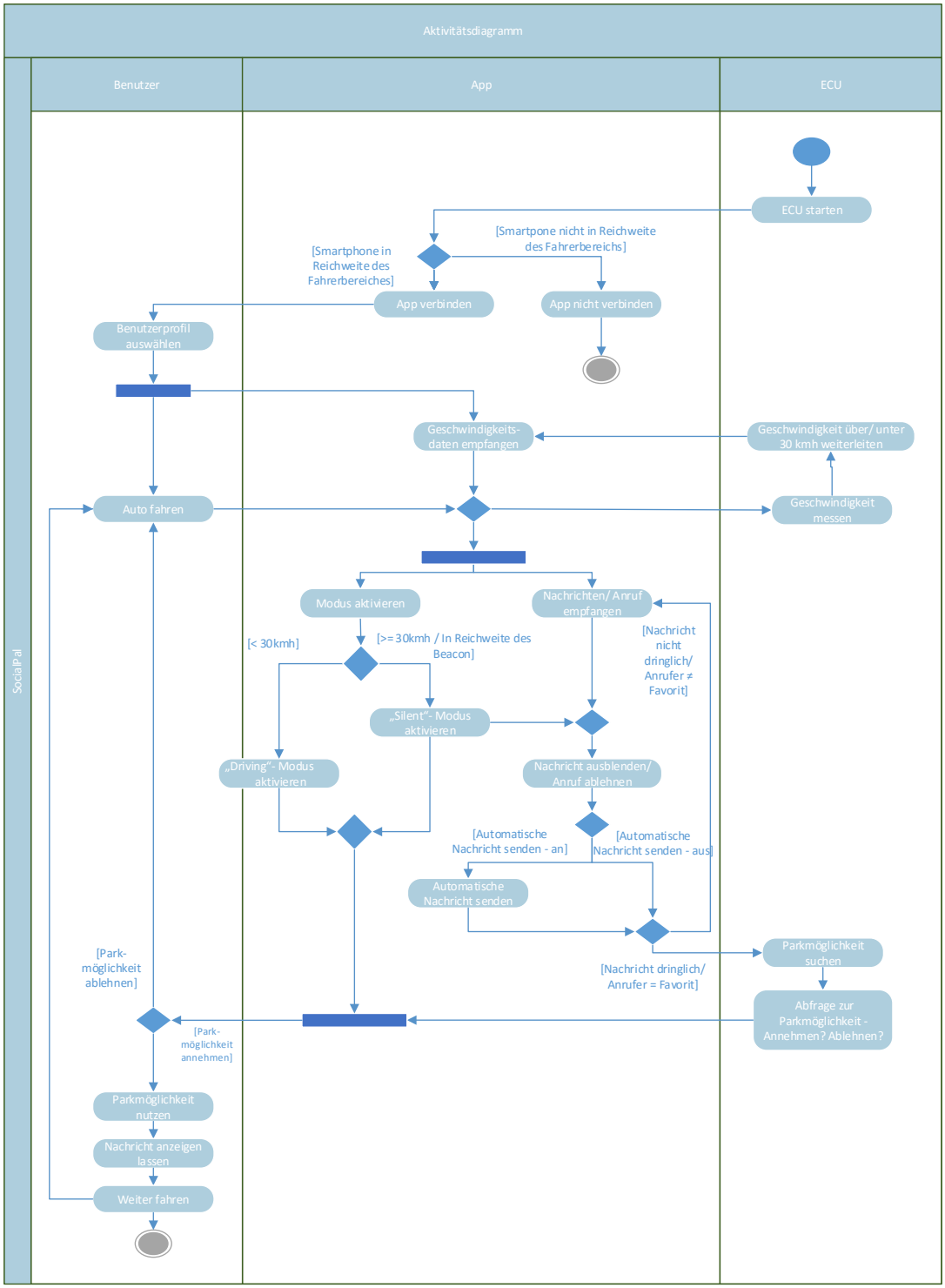

Abbildung 1.1: Aktivitätsdiagramm *Kommunikation zwischen Benutzer, App und ECU*

1.1 Projektdetails

Um die in der Einleitung gezeigte Struktur von „SocialPal“ noch detaillierter aufzuzeigen, wird in diesem Abschnitt noch genauer auf den Prozess der wechselnden Modi eingegangen und anhand zweier Zustandsdiagramme visualisiert.

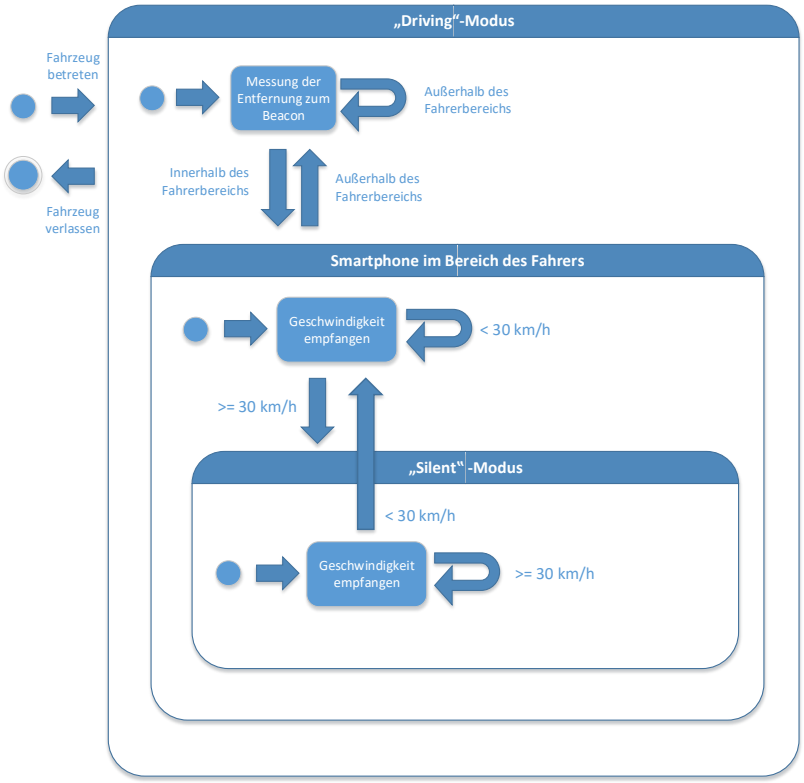

Abbildung 1.2: Zustandsdiagramm der Modi der App

Die Abbildung 1.2 visualisiert die Zustände der App. Sobald sich das Smartphone im Fahrzeug befindet, wechselt die App in den „Driving“-Modus. In diesem Zustand wird dauerhaft die Entfernung zum Beacon gemessen. Sofern sich das Smartphone innerhalb des Fahrerbereichs befindet, wechselt die App in den Zustand „Smartphone im Bereich des Fahrers“ und empfängt zusätzlich dauerhaft die aktuelle Fahrgeschwindigkeit. Bei einer Geschwindigkeit ab 30 km/h geht die App in den Zustand „Silent“-Modus über. Sollte die Fahrgeschwindigkeit wieder unter 30 km/h fallen, verlässt die App wieder den Zustand „Silent“-Modus. Beim Verlassen des Fahrzeugs wird der Zustand „Driving“-Modus verlassen.

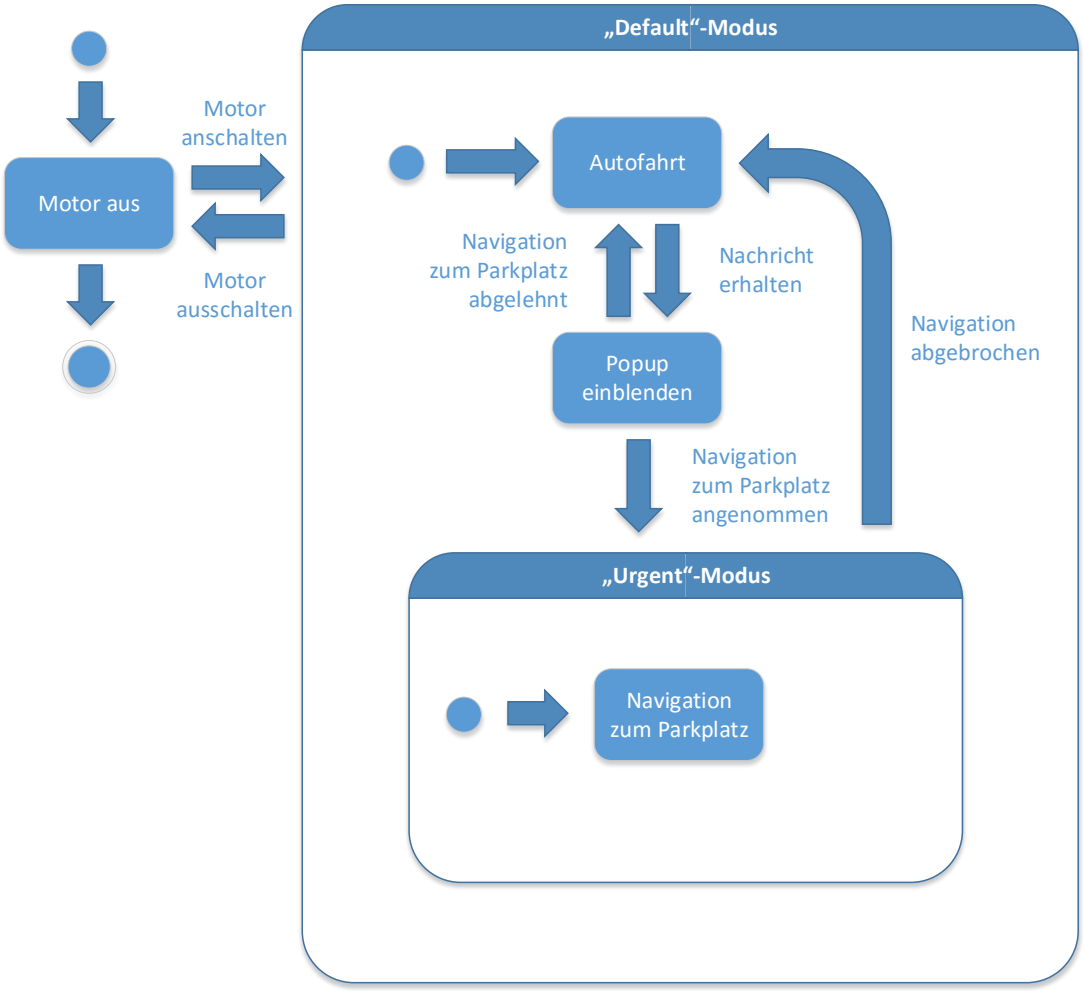

Abbildung 1.3: Zustandsdiagramm der Modi der ECU

Die Abbildung 1.3 visualisiert die Zustände der ECU. Sobald der Motor des Fahrzeuges gestartet wird, fährt die ECU hoch und befindet sich im „Default“-Modus. Sobald die Information über eine Dringlichkeit von dem Webserver erhalten wurde, erscheint ein Popup auf der Benutzeroberfläche und bietet dem Fahrer die Möglichkeit die Navigation zum nächstgelegenen Parkplatz anzunehmen oder abzulehnen. Bei einer Annahme, wechselt die ECU in den „Urgent“-Modus und der Fahrer wird zum nächstgelegenen Parkplatz navigiert.

2 Analyse der Produktfunktionen

In diesem Kapitel werden die einzelnen Produktfunktionen des Projekts „SocialPal“ näher analysiert und erläutert. Grundlegende Voraussetzung der Funktionalitäten ist die Berechtigung des Systems zum Zugriff auf die Kontaktdaten, Nachrichten, Anrufdaten und Bluetoothfunktionalitäten. Beim erstmaligen Start der App wird der Benutzer um seine Erlaubnis für diese gebeten. Auf weitere Voraussetzungen wird explizit bei der Analyse der einzelnen Funktionalitäten eingegangen. Das Illustrieren erfolgt mit Hilfe von Sequenzdiagrammen. Jede Produktfunktion besitzt ein eigenes Sequenzdiagramm, dies dient dazu eine geeignete Architektur zu finden und sich auf diese festlegen zu können. Das Pflichtenheft dient hierfür als Basis. Ebenfalls können die einzelnen Funktionen dem Pflichtenheft entnommen werden.
In den Sequenzdiagrammen kommunizieren die Entitäten untereinander oder mit dem Akteur. Hierbei wird der Benutzer als Akteur wiedergegeben, da dieser nicht selbst Bestandteil des Systems ist, sondern mit diesem interagiert und einen oder mehrere Anwendungsfälle auslöst.

2.1 Analyse von Funktionalität F10: Favoriten auswählen

Die Funktion *Favoriten auswählen* (siehe Abbildung 2.1) beschreibt das Auswählen von favorisierten Kontakten in der App. Die Voraussetzung für die Funktionalität ist, dass die App gestartet ist. Dann kann der Benutzer unter der Menüleiste „Favoriten“ auswählen, woraufhin er auf die Favoriten-Seite weitergeleitet wird. Der Benutzer kann nach vorherig erteilter Freigabe der persönlichen Kontakte für die App die für ihn als wichtig eingestufte Kontakte auswählen, welche dann gesondert in den „Favoriten“ mit dem jeweiligen Namen angezeigt werden. Diese Kontakte werden dann in der App als Favoriten gespeichert. Sobald ein Anruf von einem dieser Kontakte eingeht, informiert die App mittels des Webservers die ECU über die Dringlichkeit (siehe Abbildung 2.10).

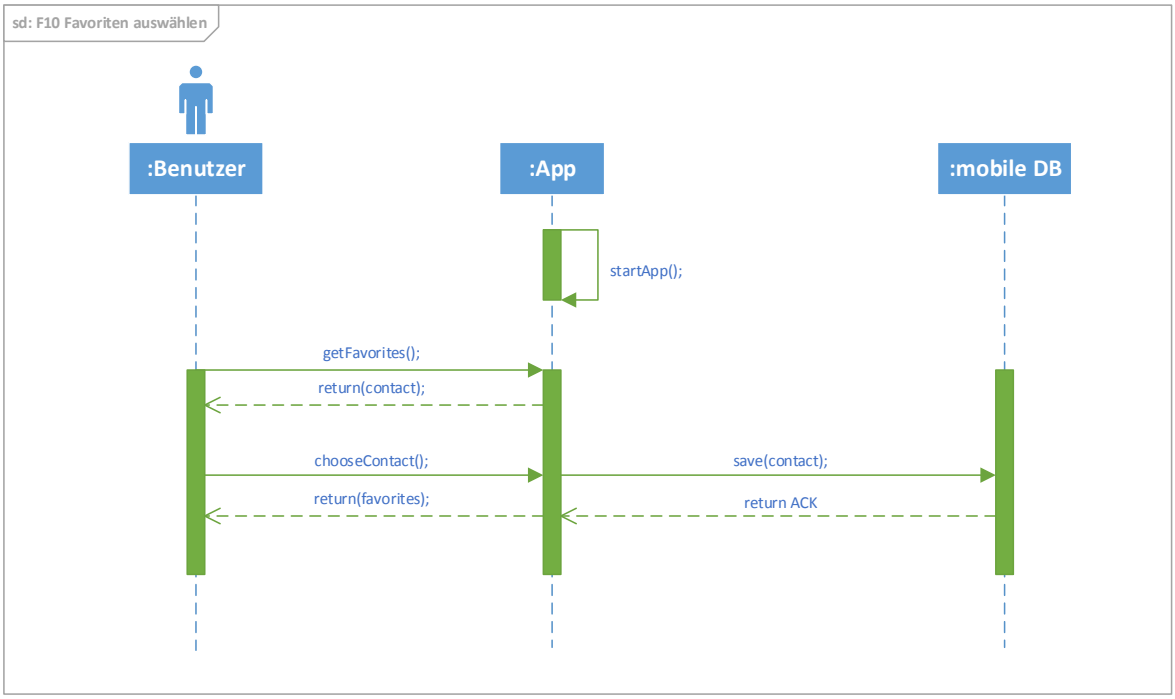

Abbildung 2.1: Sequenzdiagramm Favoriten auswählen

2.2 Analyse von Funktionalität F20: ECU-App-Kommunikation

Die Funktion *ECU-App-Kommunikation* (siehe Abbildung 2.2) beschreibt den Datenaustausch zwischen ECU und App. Dieser findet über einen Webserver statt. Voraussetzung hierfür ist, dass ECU und App über eine aktive Internetverbindung verfügen.Die ECU loggt sich beim Start als Client auf dem Webserver ein. Sofern sich das Smartphone im Bereich des Fahrers befindet, verbindet sich die App mit dem Webserver, dies wird mittels des Beacons ermittelt. Sobald das Auto eine Geschwindigkeit von mindestens 30 km/h erreicht, sendet die ECU eine Nachricht an den Webserver. Die App empfängt diese Benachrichtigung und wechselt in den „SilentModus. Wenn nun eine dringliche Nachricht eingeht sendet die App eine Benachrichtigung an den Webserver, welche die ECU empfängt und den Fahrer über die Nachricht informiert. Der anschließende Wechsel in den „UrgentModus ist in Funktion F40 beschrieben (siehe Abbildung 2.4).

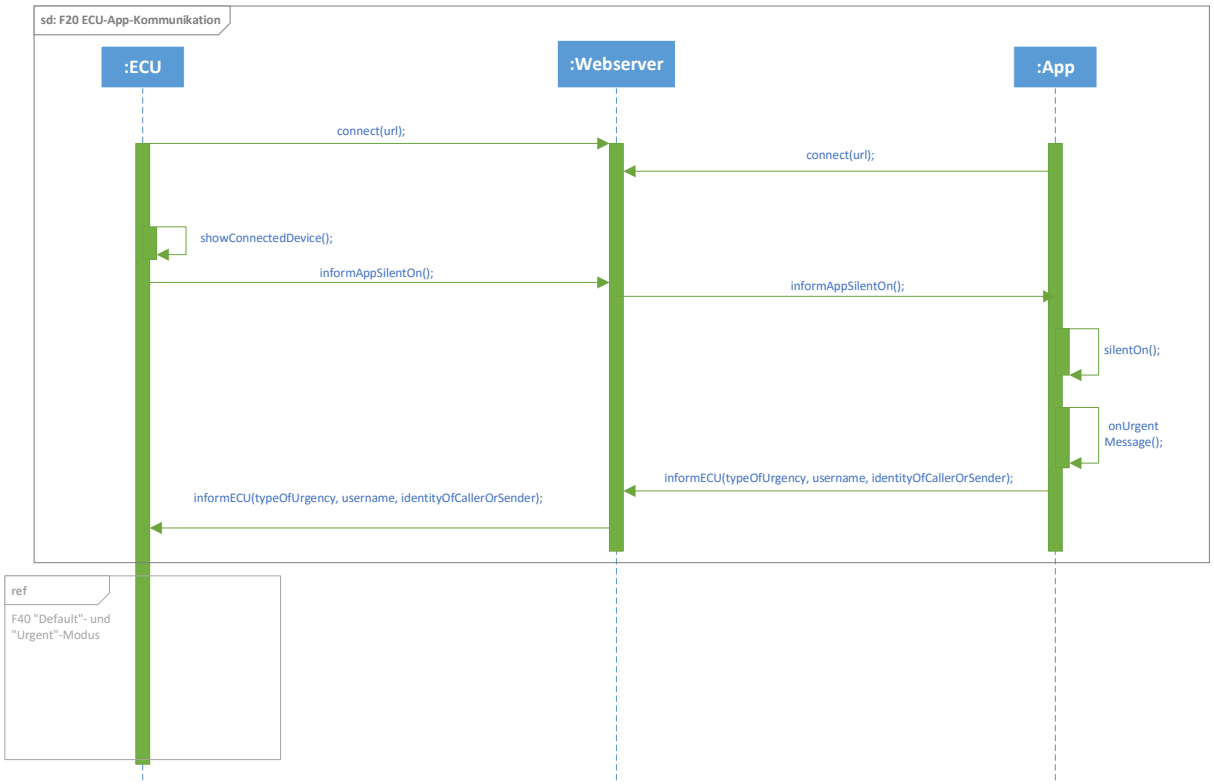

Abbildung 2.2: Sequenzdiagramm ECU-App-Kommunikation

2.3 Analyse von Funktionalität F30: Karte darstellen

Die Funktion *Karte darstellen* (siehe Abbildung 2.3) beschreibt die Darstellung der Karte. Wenn der Benutzer die ECU startet, zeigt die ECU eine Karte aus Google Maps mit dem aktuellem Standort an. Voraussetzung hierfür ist, dass eine Internetverbindung vorhanden ist und der aktuelle Standort über GPS abrufbar ist. Anschließend kann der Benutzer einen Zielort eingeben, zu dem er navigiert werden möchte. Die ECU stellt nun eine Karte mit der Route zum Zielort dar.

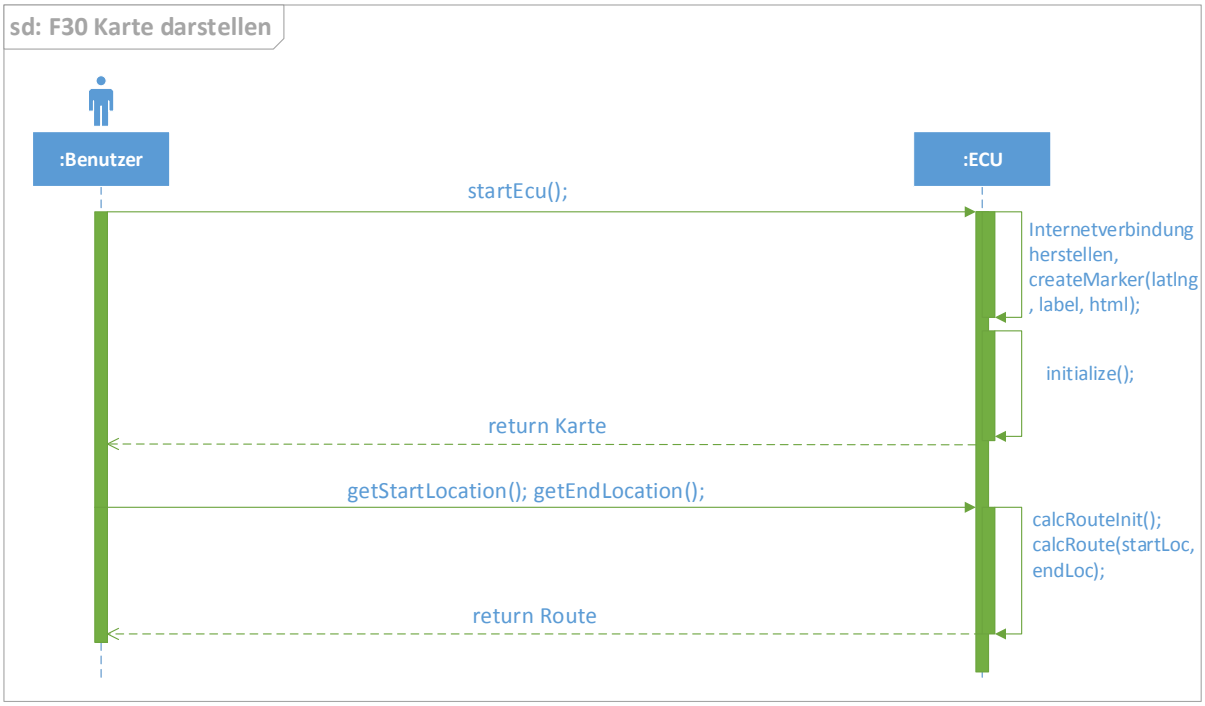

Abbildung 2.3: Sequenzdiagramm Karte darstellen

2.4 Analyse von Funktionalität F40: „Default“- und „Urgent“-Modus

Die Funktion *„Default“- und „Urgent“-Modus* (siehe Abbildung 2.4) beschreibt die Funktionsweise des „Default“- und den „Urgent“-Modus der ECU. Die ECU ist zunächst immer im „Default“-Modus. Erst bei Eingang einer dringlichen Nachricht oder einem Anruf von einem Favoriten übermittelt die App die Informationen über dieses Ereignis an den Webserver. Diese Nachricht beinhaltet den Typ der Dringlichkeit, den Benutzernamen der App, sowie den Kontaktnamen oder die Telefonnummer des Anrufers beziehungsweise des Absenders. Eine Dringlichkeit kann von drei unterschiedlichen Typen sein: eine dringliche SMS, eine dringliche Nachricht des Facebook Messengers oder ein Anruf eines Favoriten. Der Benutzername der Applikation wird für die Sprachausgabe der ECU benötigt, um für den Fall, dass sich mehrere Mobiltelefone im Bereich des Fahrers befinden, ausmachen zu können, an welchen Benutzer die dringliche Mitteilung adressiert ist. Sollte der Autor der dringlichen Nachricht bzw. der Anrufer nicht im Kontaktbuch des Mobiltelefons eingespeichert sein, wird statt des Kontaktnamen die Telefonnummer übermittelt. Folgend leitet der Webserver die Informationen weiter an die ECU, welche den Fahrer mittels einer Sprachausgabe über das Ereignis informiert und ihm anbietet ihn zur nächstgelegenen Parkmöglichkeit zu navigieren. Der Fahrer kann dies annehmen, womit die ECU in den „Urgent“-Modus wechselt und den Fahrer zum nächsten Parkplatz navigiert. Oder er lehnt die Navigation ab und die Fahrt wird wie geplant fortgesetzt. Voraussetzung ist, dass eine Verbindung zwischen dem Webserver mit der ECU und der App besteht und diese auch Signale empfangen können.

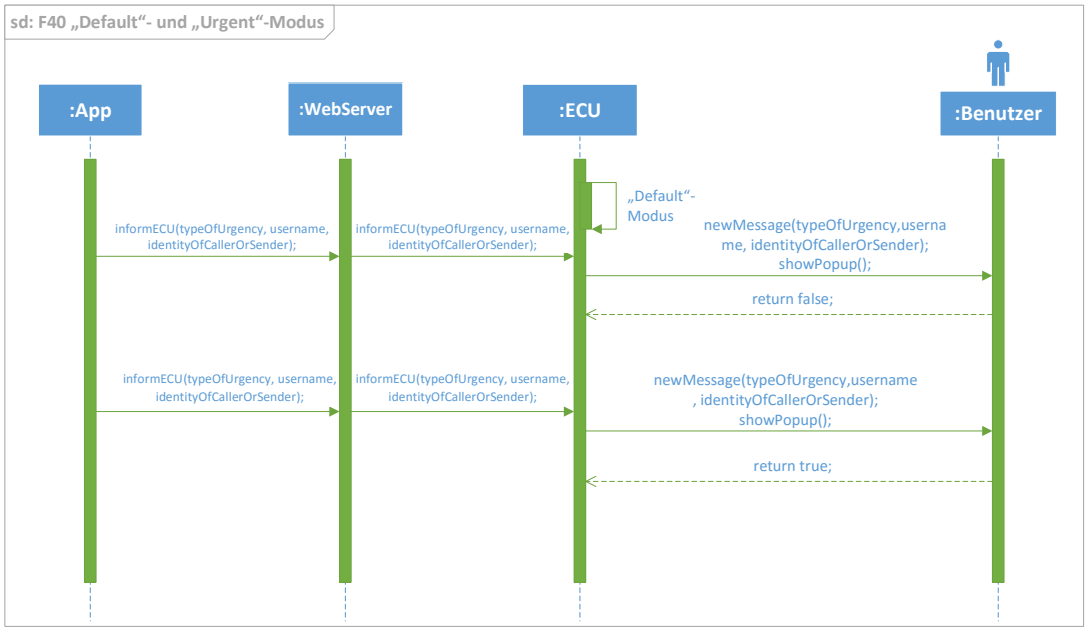

Abbildung 2.4: Sequenzdiagramm „Default“- und „Urgent“-Modus

2.5 Analyse von Funktionalität F50: „Driving“-Modus

Die Funktion *„Driving“-Modus* (siehe Abbildung 2.5) teilt dem Benutzer mit, dass die App für die Fahrt bereit ist. Voraussetzung für die Aktivierung des „Driving“-Modus ist dass sich das Smartphone im Auto befindet. Bei einer erfolgreichen Prüfung auf Anwesenheit des Beacons wechselt die App in den „Driving“-Modus.

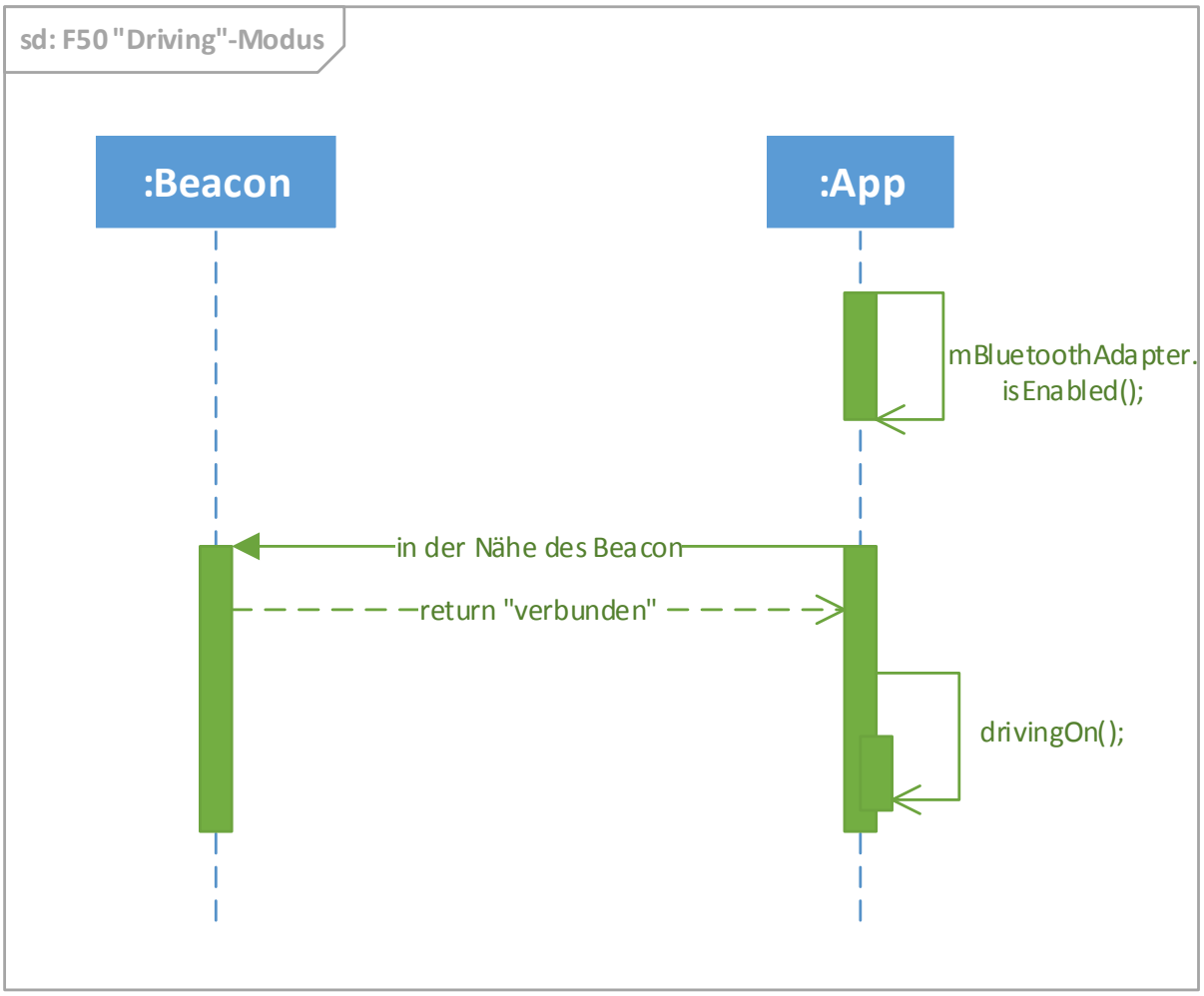

Abbildung 2.5: Sequenzdiagramm „Driving“-Modus

2.6 Analyse von Funktionalität F60: „Silent“-Modus

Die Funktion *„Silent“-Modus* (siehe Abbildung 2.6) schaltet das Mobile Gerät stumm und lässt keine akustischen oder optischen Mitteilungen über eingehende Nachrichten oder Anrufe zu. Für den Modus muss sich die App im „Driving“-Modus befinden und den Schwellenwert der Signalstärke des Beacons zur Ermittlung der Fahrerposition unterschreiten. Wenn diese Voraussetzungen erfüllt sind empfängt die App Information über die Fahrgeschwindigkeit mittels des Webservers von der ECU. Wird eine Fahrgeschwindigkeit von 30 km/h erreicht, übermittelt die ECU über den Weberver die Benachrichtigung zum Aktivieren des „Silent“-Modus. Ist die Geschwindigkeit mindestens 30 Sekunden lang unter 30 km/h sendet die ECU die Benachrichtigung zum Beenden des „Silent“-Modus.

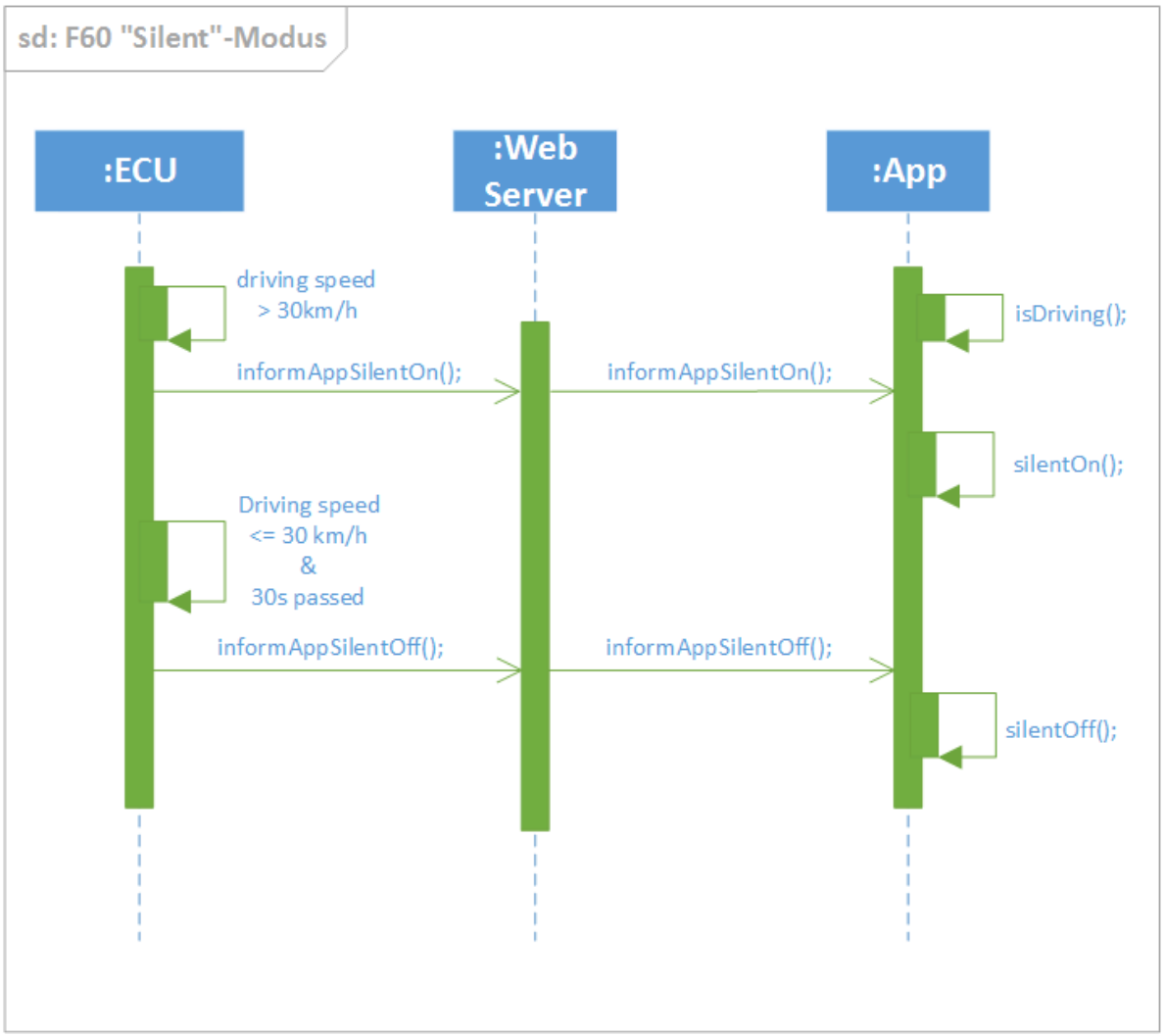

Abbildung 2.6: Sequenzdiagramm „Silent“-Modus

2.7 Analyse von Funktionalität F70: Parkmöglichkeit anzeigen

Die Funktion *Parkmöglichkeit anzeigen* (siehe Abbildung 2.7) erlaubt es dem Benutzer die nächste Parkmöglichkeit anzuzeigen. Somit kann dieser sich dorthin navigieren lassen, um seine empfangene Nachricht, welche für ihn als dringlich eingestuft worden ist, zu lesen und gegebenenfalls darauf zu antworten. Als Voraussetzung für diese Funktion gilt, dass die ECU und die App gestartet und mit dem Webserver verbunden sind. Zudem muss die App sich im „Silent"-Modus befinden, da andernfalls alle Nachrichten ohne Mitteilung an die ECU auf dem Smartphone angezeigt werden. Des Weiteren muss eine dringliche Nachricht eingegangen sein, worüber die ECU von der App informiert worden ist. In dem Sequenzdiagramm, ist zu sehen, dass die ECU einen Dialog öffnet, der den Benutzer fragt, ob er zur nächsten Parkmöglichkeit geleitet werden möchte. Der Benutzer hat daraufhin die Möglichkeit diese Option anzunehmen oder sie abzulehnen. In Folge einer Ablehnung wird die ursprüngliche Route fortgesetzt. Wenn der Benutzer die Option allerdings annimmt, so sucht die ECU die nächste Parkmöglichkeit und wechselt daraufhin in den „Urgent"-Modus. In diesem Modus wird der Benutzer zur nächstgelegenen Parkmöglichkeit navigiert.

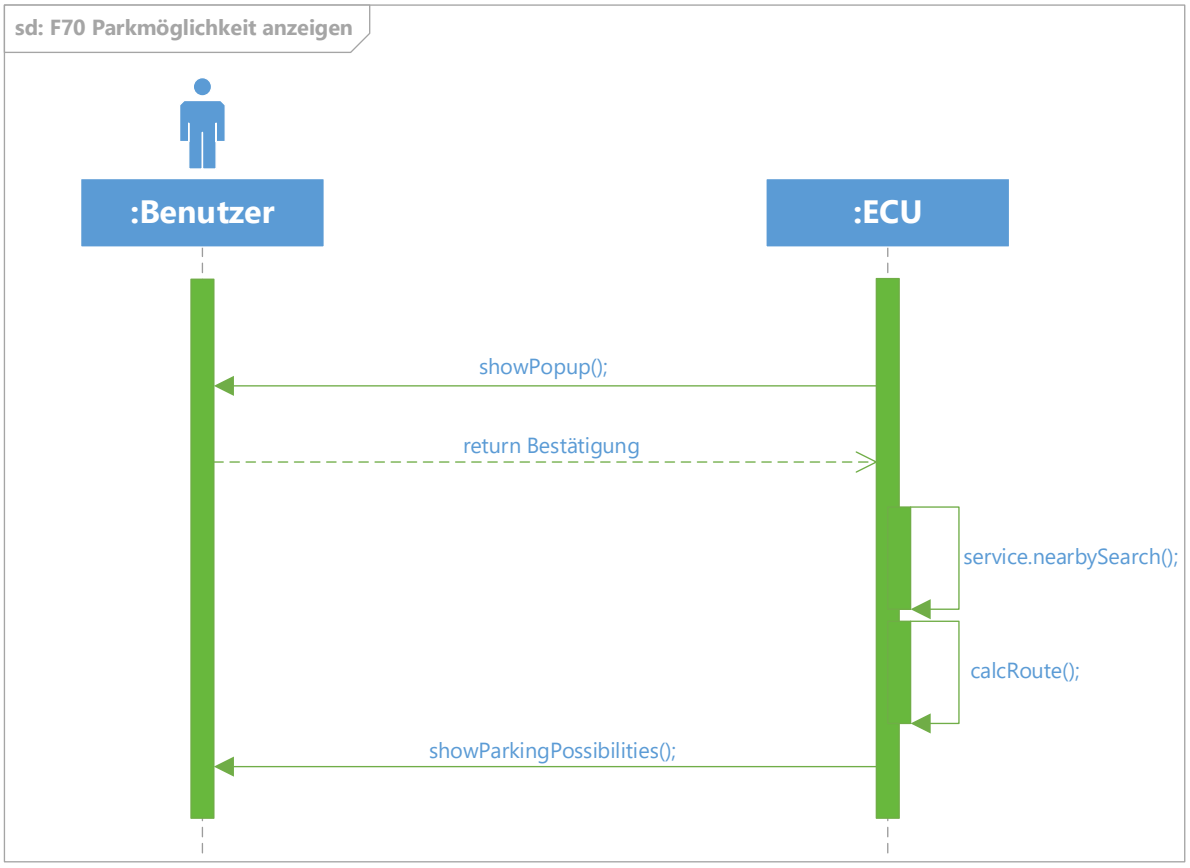

Abbildung 2.7: Sequenzdiagramm Parkmöglichkeit anzeigen

2.8 Analyse von Funktionalität F80: Sprachausgabe bei Empfang dringenden Nachrichten

Die Funktion *Sprachausgabe bei Empfang dringenden Nachrichten* (siehe Abbildung 2.8) ermöglicht es dem Benutzer über eine dringliche Nachricht zu informieren ohne ihn vom Straßenverkehr abzulenken, da dieser den Text nicht von einem Display ablesen muss. Als Voraussetzung für diese Funktion gilt, dass die App, die ECU und der Webserver gestartet sind und die App und ECU mit dem Webserver verbunden sind. Zudem muss die App sich im „Silent"-Modus befinden, da andernfalls alle Nachrichten ohne Benachrichtigung der ECU auf dem Mobiltelefon angezeigt werden. Des Weiteren muss eine dringende Nachricht eingegangen sein, worüber die ECU von der App informiert worden ist. In dem Sequenzdiagramm ist zu sehen, dass die App Informationen an den Webserver schickt, welche vom Webserver an die ECU weitergeleitet werden. Diese Informationen verarbeitet die ECU weiter und kann den Benutzer daraufhin mit einer Sprachausgabe über eine neue dringliche Nachricht informieren.

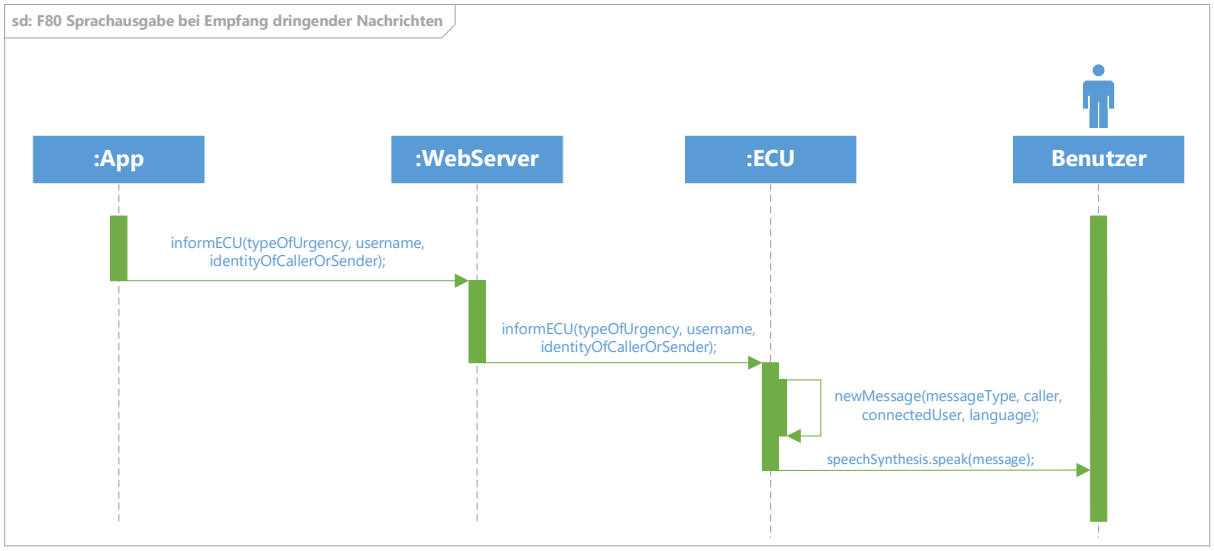

Abbildung 2.8: Sequenzdiagramm Sprachausgabe bei Empfang dringenden Nachrichten

2.9 Analyse von Funktionalität F90: Empfang von Nachrichten und Anrufen

Die Funktion F90 *Empfang von Nachrichten und Anrufen* beinhaltet zwei Funktionen, welche im folgenden getrennt betrachtet und anhand von Sequenzdiagrammen visualisiert werden. Voraussetzungen für die Funktionen sind, dass die App und die ECU mit dem Webserver verbunden sind und sich die App im „Silent“-Modus befindet.

2.9.1 F90.1 Empfang von Anrufen

Die Funktion *Empfang von Anrufen* (siehe Abbildung 2.9) ermöglicht das Abfangen von Anrufen im „Silent“-Modus. Detailliert heißt dies, dass Anrufe, sofern sich das Mobiltelefon im „Silent“-Modus befindet, automatisch abgelehnt werden und dem Anrufer im Falle der Aktivierung des Features „Senden einer Standardantwort“ automatisch eine Nachricht über den lokalen Nachrichtendienst zugesandt wird. Falls es sich bei dem Anrufer um einen zuvor als Favorit deklarierten Kontakt handelt, wird die ECU über den Webserver benachrichtigt und der Fahrer folgend mittels einer Sprachausgabe der ECU über den Anruf informiert, da Anrufe von Favoriten immer als dringlich deklariert werden. Daraufhin informiert die ECU den Fahrer mittels einer Sprachausgabe über den Anruf und bietet ihm die Möglichkeit zur nächsten Parkmöglichkeit navigiert zu werden. Dies wird unter 2.7 Analyse von Funktionalität F70 näher erläutert (siehe Abbildung 2.7).

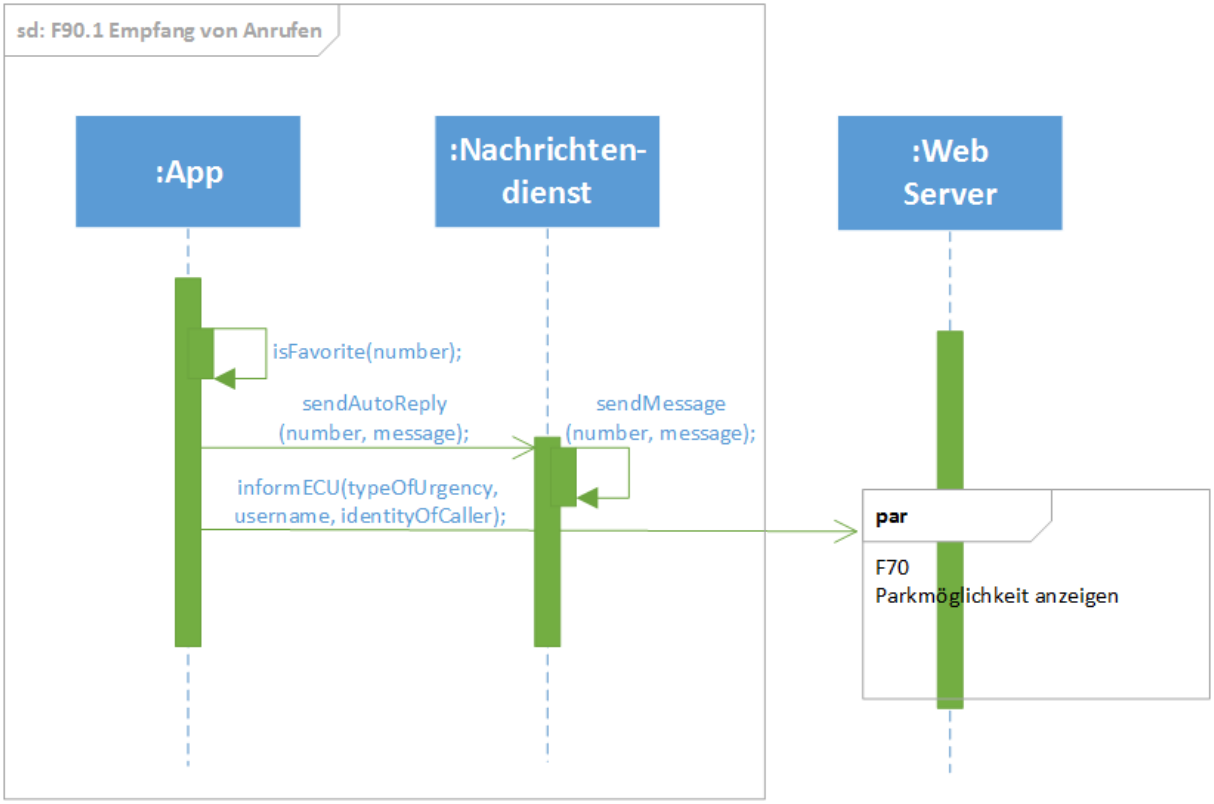

Abbildung 2.9: Sequenzdiagramm Empfang dringlicher Anruf

2.9.2 F90.2 Empfang von Nachrichten

Die Funktion *Empfang von Nachrichten* (siehe Abbildung 2.10) ermöglicht das Abfangen und Klassifizieren von SMS und Nachrichten über den Facebook Messenger im „Silent“-Modus. Zu erwähnen ist, dass Nachrichten von Favoriten nicht automatisch als dringlich deklariert werden. Sobald eine Nachricht eintrifft wird im Falle der Aktivierung des Features „Senden einer Standardantwort“ senden dem Absender eine Nachricht über den lokalen Nachrichtendienst zugestellt und der Inhalt der empfangenen Nachricht mittels eines Klassifizierungsalgorithmus auf seine Dringlichkeit überprüft. Dabei handelt es sich um einen Naives Bayes Klassifizierungsalgorithmus, welcher anhand der Präferenzen des Benutzers trainiert wurde um selbstständig zu erkennen, wann eine dringliche Nachricht vorliegt. Falls es sich um eine dringliche Nachricht handelt, wird mittels einer Abfrage auf die Kontaktdaten des Mobiltelefons, wenn vorhanden dessen Kontaktname ermittelt. Wenn kein Kontakt mit der Telefonnummer eingespeichert ist, wird lediglich die Telefonnummer als Information weitergegeben. Dies wird anschließend über den Webserver an die ECU übermittelt. Daraufhin informiert die ECU mittels einer Sprachausgabe den Fahrer über den Anruf und bietet ihm die Möglichkeit zur nächsten Parkmöglichkeit navigiert zu werden. Dies wird unter 2.7 Analyse von Funktionalität F70 näher erläutert (siehe Abbildung 2.7).

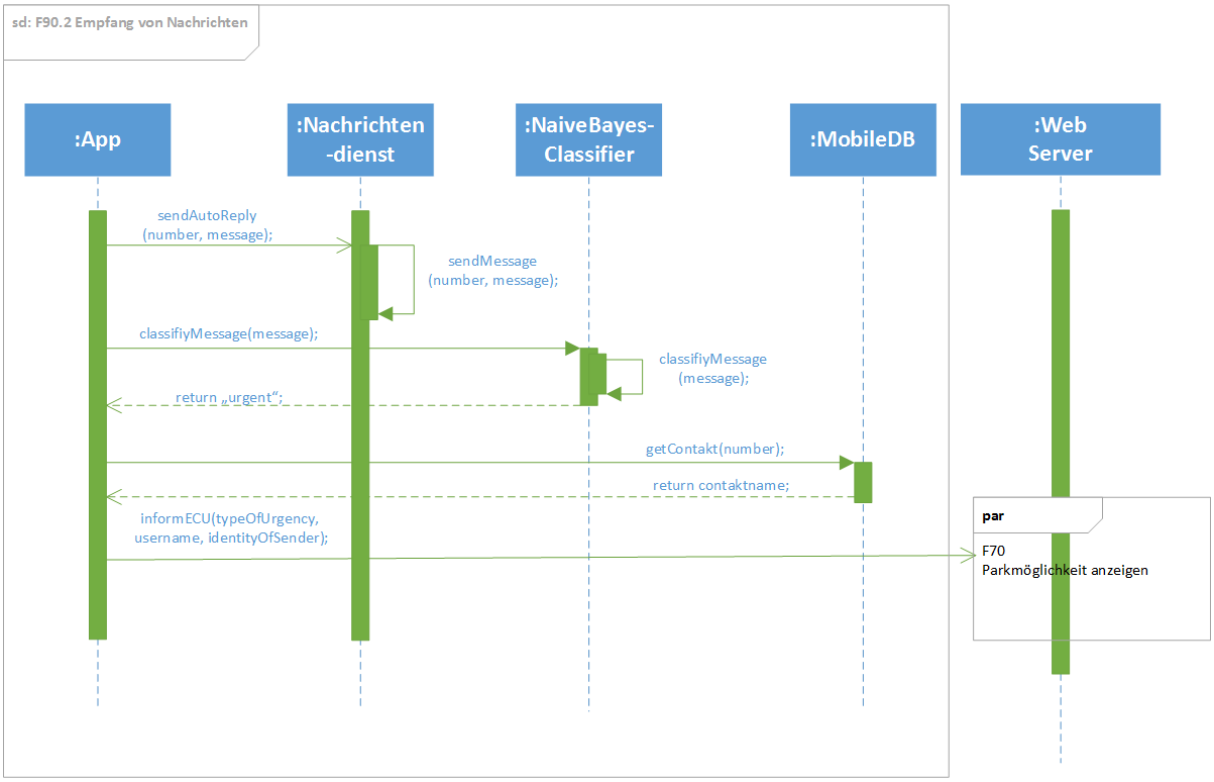

Abbildung 2.10: Sequenzdiagramm Empfang dringliche Nachricht

2.10 Analyse von Funktionalität F100: Senden einer Standardantwort

Die Funktion *Senden einer Standardantwort* (siehe Abbildung 2.11) beschreibt das Senden einer Standardantwort auf eingehende Nachrichten und Anrufe. Diese automatische Standardantwort kann vom Benutzer im Vorfeld individuell in den Einstellungen definiert werden und auf Wunsch auch jederzeit deaktiviert werden. Als Voraussetzung für diese Funktion gilt, dass die ECU sowie auch die App gestartet sind und beide in Verbindung mit Webserver stehen. Des Weiteren muss sich die App im „Silent"-Modus befinden, da im „Driving"-Modus das Smartphone wie gewohnt genutzt werden kann und somit eine Standardantwort nicht notwendig ist.

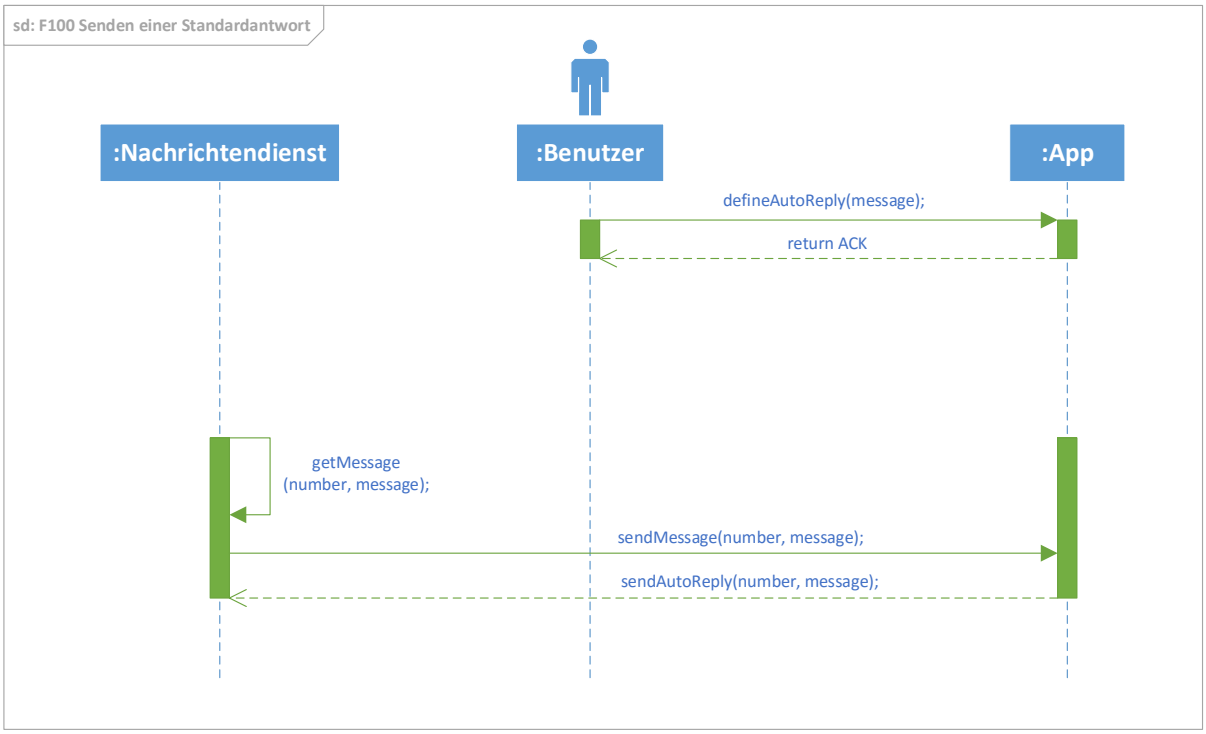

Abbildung 2.11: Sequenzdiagramm Senden einer Standardantwort

2.11 Analyse von Funktionalität F110: Automatischer App Start

Die Funktion *Automatischer App Start* (siehe Abbildung 2.12) beschreibt den automatischen Start der App, wenn eine Verbindung zum Beacon hergestellt wird. Voraussetzung hierfür ist, dass Bluetooth am Smartphone aktiviert ist. Wenn das Smartphone die Anwesenheit des Beacons registriert hat, sich dieses somit im Auto befindet, startet die App automatisch.

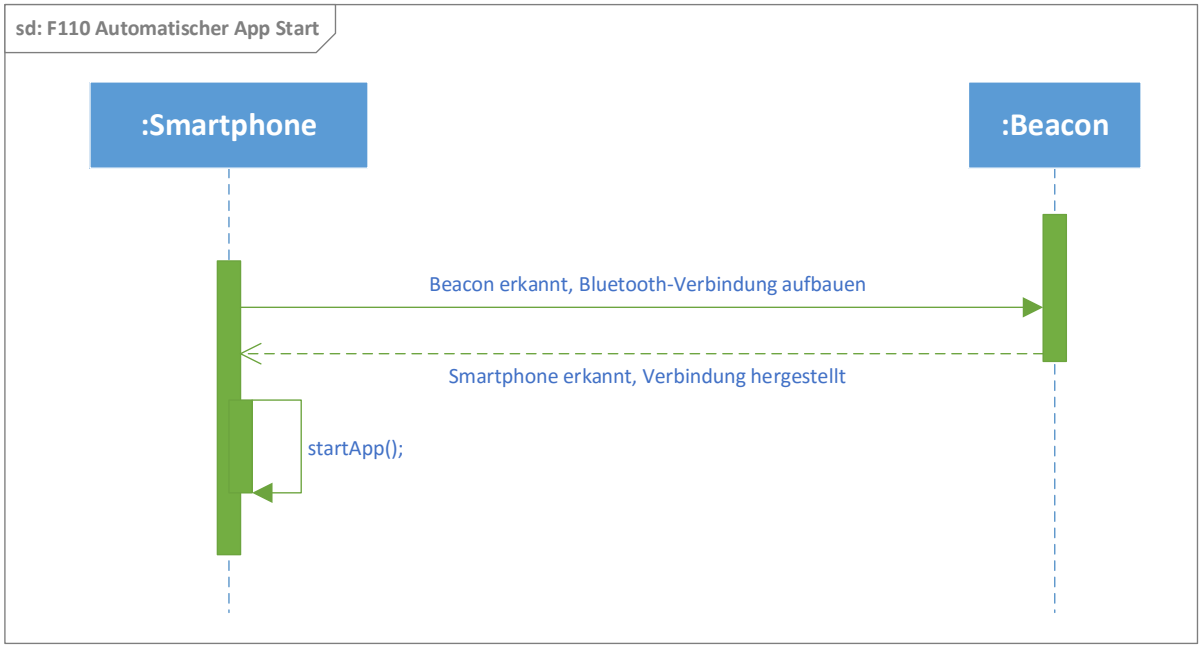

Abbildung 2.12: Sequenzdiagramm Automatischer App Start

3 Resultierende Softwarearchitektur

Dieses Kapitel beschäftigt sich mit dem Aufbau des Service „SocialPal“ und befasst sich mit den einzelnen Komponenten des Systems. Der Service „SocialPal“ besteht aus den übergeordneten Subsystemen App, Webserver und ECU (Engine-Control-Unit, welche den Bordcomputer des Autos simuliert), welche im weiteren Verlauf dieses Kapitels mitsamt ihrer jeweiligen Unterkomponenten dargestellt werden. Die beiden Clients App und ECU übernehmen dabei die eigentliche Arbeit und nutzen den Webserver lediglich zur Kommunikation miteinander. Sowohl die App als auch die ECU haben dabei eine grafische Oberfläche und bieten dem Nutzer über diese Interaktionsmöglichkeiten. Auf Seiten der App kann der Nutzer dabei aus den Kontakten seine persönlichen Favoriten auswählen, die Sprache der App wechseln, den Inhalt seiner automatischen Antwort individualisieren, sowie die Funktion der App manuell deaktivieren. Auf Seiten der ECU kann der Nutzer die Route auswählen und bei Meldung einer dringlichen Nachricht durch das Smartphone die Navigation zur nächsten Parkmöglichkeit akzeptieren oder verwerfen.

In dem folgenden Abschnitt werden die eingangs erwähnten Komponenten des Service „SocialPal“ detaillierter beschrieben. Das in Abbildung 3.1 dargestellte Komponentendiagramm dient dabei zur Veranschaulichung der Komponenten App, Beacon, ECU, Smartphone und Webserver.

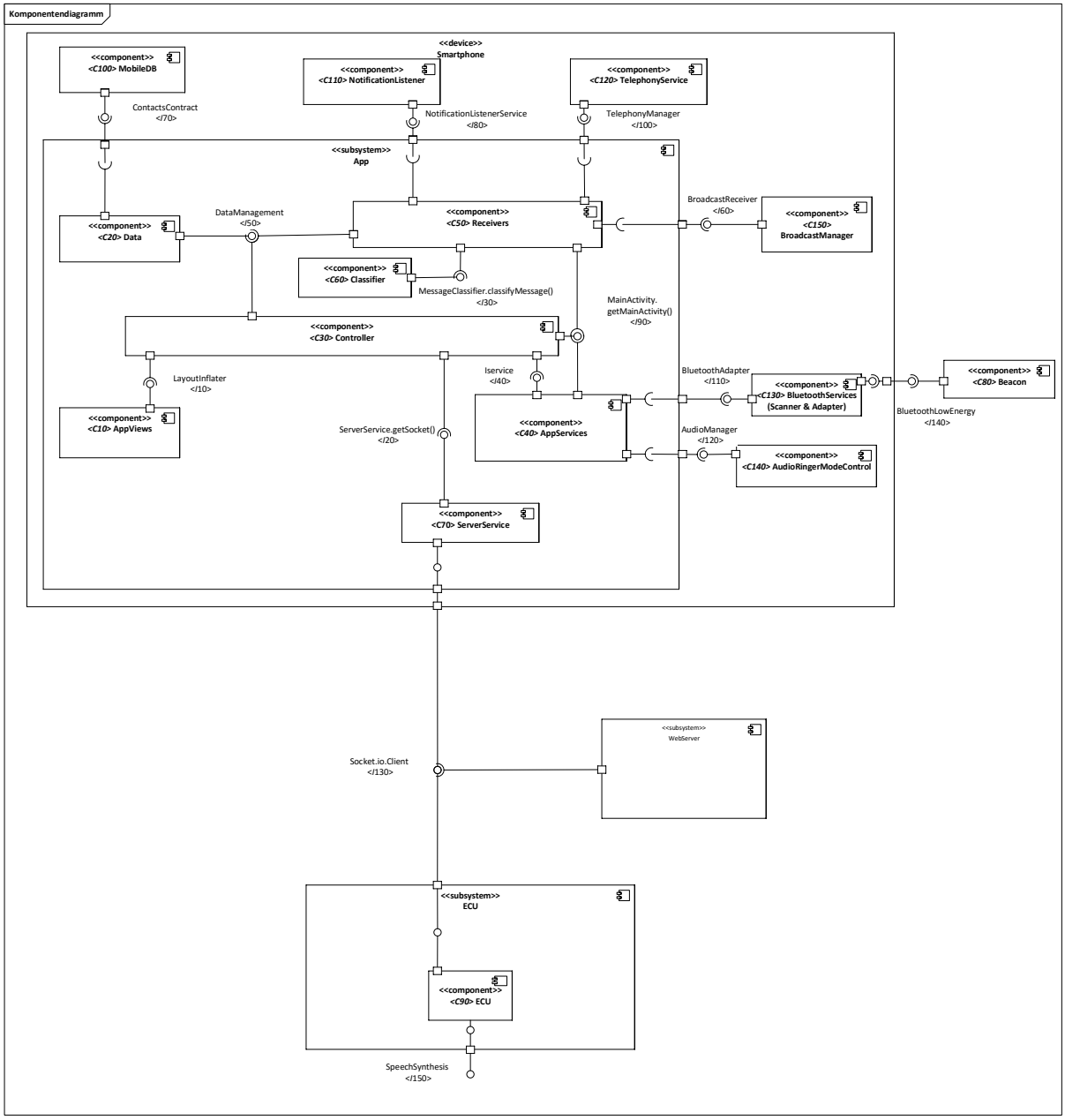

Abbildung 3.1: Komponentendiagramm „SocialPal“

3.1 App

3.1.1 Komponentenspezifikation

Komponente $\langle C10 \rangle$**: AppViews**
Die Komponente AppViews stellt die Oberfläche der Android-App dar. Die Benutzeroberflächenelemente sind in XML Dateien definiert und die Steuerung dieser verschieden Layouts erfolgt über die **C30** *Controller* Komponente.

Komponente $\langle C20 \rangle$**: Data**
Die Komponente Data verwaltet die vom Nutzer gewählten, favorisierten Kontakte sowie die Einstellungen. Außerdem besitzt sie eine Schnittstelle zu der lokalen Kontaktdatenbank des Smartphones (**C100** MobileDB), welche benötigt wird um Kontakte aus dem Adressbuch den Favoriten hinzuzufügen.

Komponente $\langle C30 \rangle$**: Controller**
Die Komponente Controller besitzt die Aufgabe die **C10** *Views* mit Daten zu füllen und diese zu managen (Model View Controller Pattern). Außerdem besitzt sie Schnittstellen zu den **C50** *Receiver*, **C40** *AppServices* und **C70** *ServerServices* Komponenten. Sie kümmert sich um die Koordination der unterschiedlichen Komponenten und sorgt dafür, dass bei eintretenden Ereignissen die richtigen Aktionen ausgelöst werden. Ebenfalls leitet sie abhängig von den ihr gelieferten Informationen Signale an den Server.

Komponente $\langle C40 \rangle$**: AppServices**
Die Komponente AppServices implementiert die Funktionalitäten bezüglich der Verbindung mit der Komponente **C80** *Beacon* mittels Bluetooth Low Bluetooth und des Stumm schalten des Smartphones.

Komponente $\langle C50 \rangle$**: Receivers**
Die Komponente Receiver besitzt die Funktionalitäten eingehende SMS, Anrufe und Nachrichten über den Facebook Messenger abzufangen. Anrufe werden abgefangen und die Inhalte der Nachrichten an die Komponente **C60** *Classifier* zur Klassifizierung der Dringlichkeit weitergeleitet. Außerdem setzt die Komponente die optionale Funktionalität des Sendens einer Standardantwort an den Absender einer Nachricht bzw. des Anrufers um.

Komponente $\langle C60 \rangle$: Classifier
Der Classifier untersucht die Inhalte der ihm übermittelten Nachrichten und prüft diese mit den ihm vorliegenden Präferenzen auf Dringlichkeit. Bei erkannter Dringlichkeit teilt er dies dem auf ihn zugreifenden **C50** *Receiver* mit.

Komponente $\langle C70 \rangle$: ServerService
Der ServerService wickelt die Kommunikation zwischen App und dem Webserver ab, welcher die Funktionalität besitzt Informationen zwischen den Subsystemen App und ECU zu übermitteln.

Komponente $\langle C80 \rangle$: Beacon
Der Beacon sendet dauerhaft ein Bluetooth Low Energy Signal zur Information über seine Anwesenheit.

3.1.2 Schnittstellenspezifikation

Schnittstelle $\langle I10 \rangle$: LayoutInflator

Android stellt systemeigene Klasse LayoutInflator zur Verfügung, welche den Zugriff auf die mittels XML definierten Benutzeroberflächen bietet.

Operation	**Beschreibung**
LayoutInflator.inflate(int resource, ViewGroup root);	Liefert ein View Objekt der übergebenen XML Ressource und ermöglicht über die Methode View.findViewById(int resource) einen direkten Zugriff auf die Elemente der ViewGroup.

Schnittstelle $\langle I20 \rangle$: ServerService.getSocket()

Der ServerService stellt der Komponente **C50** *Controller* eine Schnittstelle über die Methode getSocket() welche eine Socket Instanz zurückgibt zur Verfügung. Über diese Instanz kann auf die von der Komponente **C70** *ServerService* implementierten Serverfunktionalitäten der Socket.io Client API zugegriffen werden.

Operation	**Beschreibung**
ServerService.getSocket().connect()	Baut die Verbindung zum Webserver auf.
ServerService.getSocket().disconnect()	Trennt die Verbindung vom Webserver.
ServerService.getSocket().emit(String event, Object data)	Sendet über den Webserver Informationen an das Subsystem ECU.
ServerService.getSocket().on(String event, Object data)	Empfängt ein Event und Daten des Webservers.

Schnittstelle $\langle I30 \rangle$: MessageClassifier

Die Schnittstelle MessageClassifier wird von der Komponente **C60** *Classifier* für den Zugriff auf die Klassifizierungsfunktion des WEKA-Algorithmus von Texten bereitgestellt. Die Komponente **C50** *Receivers* nutzt diese Schnittstelle um die empfangenen Texte an die Komponente **C60** *Classifier* zur Klassifizierung zu übergeben.

Operation	**Beschreibung**
MessageClassifier.classifyMessage(String message)	Die öffentliche Methode classifyMessage() ruft den Klassizierungsalgorithmus auf und übergibt ihm die zu klassifizierende Nachricht. Als Rückgabe gibt sie die Dringlichkeit der Nachricht im Stringformat zurück.

Schnittstelle $\langle I40 \rangle$**: IService**

Die Schnittstelle IService wird von der Komponente **C40** *AppServices* zur Verfügung gestellt. Die Komponente **C30** *Controller* nutzt diese Schnittstelle um das Audioprofil des Smartphones zu verwalten. Ebenfalls wird über diese Schnittstelle die Entfernung zwischen Smartphone und Beacon übermittelt, aus welcher die Position im Fahrzeug resultiert.

Operation	**Beschreibung**
setRingerMode()	Stellt das Smartphone auf stumm beziehungsweise laut.
BeaconHelper.init()	Initiiert ein Object der BeaconHelper Klasse, welche sich um die Verbindung mit dem Beacon kümmert.

Schnittstelle $\langle I50 \rangle$: DataManagement

Die Schnittstelle DataMangament wird von der Komponente **C60** *Data* zur Verfügung gestellt. Die Komponente **C50** *Receivers* nutzt diese, um den Inhalt der zu versendenden Standardantwort zu erhalten und den Kontaktnamen des aktuellen Anrufers zu ermitteln, um zu bestimmen, ob es sich um einen Favoriten handelt. Die Komponente **C30** *Controller* greift auf die Schnittstelle zu, um vorgenommene Einstellungen und die Favoritenliste zu speichern, sowie bei App-Start die Einstellungen aus dem Speicher des Android Smartphones zu lesen.

Operation	**Beschreibung**
saveToSharedPreferences()	Speichert die aktuell geänderten Einstellungen in den SharedPreferences.
readFromSharedPreferences()	Gibt die aktuellen Einstellungen zurück.
String getContactName(String phoneNumber, Context context)	Falls die übergebene Telefonnummer nicht in der lokalen Kontaktdatenbank des Smartphones eingespeichert ist, gibt die Methode den String notInContacts zurück. Andernfalls gibt sie den Kontaktnamen zurück.
getContactFromMobileDB()	Gibt den Kontakt Namen, die Kontakt-ID und die Rufnummer zurück.
Boolean isNumberFavorite(String phoneNumber)	Überprüft ob die übergebene Telefonnummer als Favorit eingespeichert ist.

Schnittstelle $\langle I60 \rangle$: BroadcastReceiver

Android bietet die systemeigene Klasse BroadcastReceiver, welche von den Receiverklassen der Komponente **C50** *Receivers* erweitert werden und für die abzufangende Events registriert werden können. Mit Receiverklassen sind die Klassen gemeint, welche die Funktionalität des Abfangens der SMS, Anrufe und Facebook Messenger Nachrichten implementieren (FacebookMessengerReceiver, CallReceiver und SmsReceiver).

Operation	**Beschreibung**
Context.registerReceiver(BroadcastReceiver receiver, String event)	Methode, welche den Receiver für die Anwendung in der Komponente **C30** *Controller* registriert. Context ist ein von Android gegebenes Interface, welches globale Informationen zur Applikationsumgebung enthält.
onReceive(Context context, Intent intent)	Die Receiverklassen überschreiben die Methode des BroadcastReceivers. Diese wird aufgerufen wenn das für den Receiver definierte Event ausgelöst wird.

Schnittstelle $\langle I70 \rangle$: ContactsContract

Die Komponente **C100** *MobileDB* bietet die von Android bereitgestellte Schnittstelle ContactsContract, welche der Komponente **C20** *Data* die Funktionalität für das Einsehen und Bearbeiten der Android internen Kontaktliste ermöglicht.

Operation	**Beschreibung**
getColumnIndex()	Methode bietet die Möglichkeit auf bestimmte Attribute der lokalen Kontaktdatenbank des Smartphones zu zugreifen.

Schnittstelle ⟨*I*80⟩: NotificationListenerService

Die Komponente **C110** *NotificationListener* bietet die von Android bereitgestellte Schnittstelle NotificationListenerService. Die Komponente **C50** *Receivers* greift auf diese Schnittstelle zu, um den Inhalt von eingegangenen Facebook Messenger Nachrichten abzufangen.

Die Komponente **C50** *Receiver* erweitert die Klasse NotificationListener und überschreibt folgende Methoden.

Operation	**Beschreibung**
onNotificationPosted (StatusBarNotification sbn)	Die Methode wird aufgerufen, wenn eine Notifikation eintrifft. Folgend werden Facebook Messenger Nachrichten raus gefiltert und der Inhalt der Komponente Classifier zur Klassifizierung übermittelt.

Schnittstelle ⟨*I*90⟩: MainActivity.getMainActivity()

Die Komponente **C30** *Controller* bietet über die öffentliche Methode getMainActivity() der Klasse MainActivity eine Schnittstelle um auf öffentliche Methoden der Klasse zuzugreifen.

Operation	**Beschreibung**
Boolean getSilentMode()	Gibt Informationen darüber, ob sich die App im „Silent“-Modus befindet.
setSilentMode(Boolean mode)	Setzt den „Silent“-Modus der App.
informECU(int type, String identity)	Gibt der Komponente Controller die Information über eine eingetroffene Dringlichkeit, deren Typ (SMS, Anruf, Facebook Messenger Nachricht) und die Identität des Anrufers bzw. Versenders der Nachricht.

Schnittstelle $\langle I100 \rangle$**: TelephonyManager**

Die Komponente **C120** *TelephonyService* bietet die von Android bereitgestellte Schnittstelle TelephonyManager. Die Komponente **C50** *Receivers* greift auf diese Schnittstelle zu, um Informationen über eingehende Anrufe zu erhalten.

Operation	**Beschreibung**
Context.getSystemService (Context.TELEPHONY_SERVICE)	Gibt eine Instanz des TelephonyManagers zurück.

Schnittstelle $\langle I110 \rangle$**: BluetoothAdapter**

Die Komponente **C130** *BluetoothServices* bietet die von Android bereitgestellte Schnittstelle BluetoothAdapter, welche den AppServices die Funktionalität für das Suchen und Verbinden mit Bluetooth Geräten ermöglicht.

Operation	**Beschreibung**
Context.getSystemActivity(). getAdapter()	Initiiert den lokalen BluetoothAdapter des Smartphones.
BluetoothAdapter.isEnabled()	Überprüft ob Bluetooth am Gerät eingschaltet ist.
BluetoothAdapter. isMultipleAdvertisementSupported()	Überprüft ob Bluetooth Low Energy Advertising unterstütz wird.
BluetoothAdapter. getBluetoothLeScanner()	Initiiert ein Objekt für die Suche nach Bluetooth Low Energy Geräten.
BluetoothAdapter.startScan(new SampleScanCallback)	Initiiert die Suche nach anderen Bluetooth Low Energy Geräten in der Nähe.
BluetoothAdapter.getDevice(). getAddress()	Gibt die Hardware-Adresse des verbundenen Bluetooth Low Energy Gerätes zurück.
BluetoothAdapter.getDevice(). getName()	Gibt den Namen des verbundenen Bluetooth Gerätes zurück.

Schnittstelle ⟨*I*120⟩**: AudioManager**

Die Komponente **C140** *AudioRingerModeControl* bietet die von Android bereitgestellte Schnittstelle AudioManager, welche der Komponente **C40** *AppServices* die Funktionalität für die Änderung der Audio Profile des Smartphones ermöglicht.

Operation	**Beschreibung**
Context.getSystemService (Context.AUDIO Service)	Gibt eine Instanz des AudioManager's zurück.
setRingerMode(RingerMode: int)	Setzt den Ringer Mode des Smartphones. Als Modi besitzt dieser Lautlos, Normal und Vibration.

Schnittstelle ⟨*I*130⟩**: Socket.io.Client**

Socket.io ist ein Framework für Echtzeit Webanwendungen und bietet eine Client API für Javascript und Android Anwendungen über welche eine Verbindung mit dem Webserver aufgebaut werden kann und Informationen übermittelt werden können. Diese API wird sowohl von den Subsystemen ECU als auch App genutzt und über die Bibliothek eingebunden.

Operation	**Beschreibung**
Socket.connect(String webserveraddress)	Verbindet die App oder ECU als Client mit dem Webserver
Socket.on(String event, Object object)	Warten auf ein Event des Webservers und empfängt ein Objekt beim Eintreffen.
Socket.emit(String event, Object object)	Löst ein Event des Webservers aus und übermittelt gleichzeitig ein Objekt.

Schnittstelle $\langle I140 \rangle$: BluetoothLowEnergy

Die Komponente **C130** *BluetoothServices* stellt die von Android bereitgestellte Schnittstelle zur Bluetooth Kommunikation dar. Diese empfängt in regelmäßigen Abständen die vom Beacon ausgesendeten Nachrichtenbündel und informiert das System so über dessen Anwesenheit.

Operation	**Beschreibung**
getSystemService(Context.BLUETOOTH_SERVICE)	Bietet Zugriff auf das verbaute Bluetooth Modul.

3.2 ECU

3.2.1 Komponentenspezifikation

Im folgenden Kapitel wird die Komponete der ECU und die dazu gehörigen Schnittstellen beschrieben.

Komponente $\langle C90 \rangle$**: ECU**
Die Komponete *ECU* kümmert sich um die gesamte Anzeige und Steuerung der ECU. Dazu zählen das Anzeigen des Wetters, der Route, der Uhrzeit und der Geschwindigkeit. Des Weiteren können Informationen über den Geschwindigkeitsstatus an den Webserver gesendet werden und Informationen zu einer neuen dringlichen Nachricht empfangen werden. Durch letzteres wird eine Sprachausgabe ausgelöst und des Weiteren kann die Suche nach der nächsten Parkmöglichkeit ausgelöst werden, dabei werden beide Aktionen von der ECU durchgeführt.

3.2.2 Schnittstellenspezifikation

Schnittstelle $\langle I150 \rangle$**: SpeechSynthesis**

Die Komponente **C90** *ECU* bietet die vom Google Chrome Browser bereitgestellte Schnitstelle SpeechSynthesis, welche es ermöglicht Texte in eine Sprachausgabe umzuwandeln und dann über ein Ausgabegerät auszugeben.

Operation	**Beschreibung**
SpeechSynthesis.speak(String message)	Die Methode speak() wandelt einen String in ein Audiosignal um und gibt dieses über ein Ausgabegerät aus.

Schnittstelle $\langle I130 \rangle$**: Socket.io.Client**

Dieses Interface wurde bereits beschrieben, siehe **I130**.

3.3 Smartphone

3.3.1 Komponentenspezifikation

Hier wird kurz auf die Komponenten vom Android Smartphone, auf welche die App zugreift, eingegangen.

Komponente $\langle C100 \rangle$**: MobileDB**
Die Komponente MobileDB ist die lokale Kontaktdatenbank des Smartphones, wo die Kontakte des Nutzers gespeichert sind.

Komponente $\langle C110 \rangle$**: NotificationListener**
Die Komponente NotificationListener ist ein von Android zur Verfügung gestellter Dienst der alle Push-Benachrichtigungen des Smartphones empfängt, wie den Eingang von Nachrichten und Anrufen. Der NotificationListener bietet eine Schnittstelle zu **I80** *NotificationListenerService*, auf die die App zugreifen kann.

Komponente $\langle C120 \rangle$**: TelephonyService**
Die Komponente TelephonyService ist eine in Android integrierte Komponente, welche Informationen über Dienste des Smartphones, wie etwa Anrufe und SMS Nachrichten zur Verfügung stellt. Außerdem bietet sie die Schnittstelle **I100** *TelephonyManager* für die App.

Komponente $\langle C130 \rangle$**: BluetoothServices**
Die Komponente BluetoothServices kümmert sich im Allgemeinen um alle Bluetooth Aktivitäten des Smartpohnes. Außerdem bietet sie die Schnittstelle **I110** *BluetoothAdapter* für die App.

Komponente $\langle C140 \rangle$**: AudioRingerModeControl**
Der AudioRingerModeControl ist für die Ringer Modi (Lautlos,Normal,Vibration) und alle Audio Systeme des Smartphones zuständig. Er bietet die Schnittstelle **I120** *AudioManager* für die App.

Komponente $\langle C150 \rangle$**: BroadcastManager**
Der BroadcastManager bietet über die Schnittstelle **I60** *BroadcastReceiver* die Möglichkeit Events auf dem Smartphone abzufangen.

3.4 Protokolle für die Benutzung der Komponenten

3.4.1 App

In diesem Abschnitt wird die Verwendung der einzelnen Komponenten des Service „SocialPal“ mit Hilfe von State-Charts verdeutlicht.

Das Subsystem App kann als solche auch in anderen Navigationssystemen verwendet werden, dies erfordert allerdings eine gewisse Adaption auf der Seite des Bordcomputers des potentiellen Fahrzeugs. So muss darauf geachtet werden, dass die Geschwindigkeit in einem für die App verständlichen Format übergeben wird und die von der App übergebenen Parameter, welche Informationen über die eingetroffenen Nachrichten und Anrufe erhalten, entsprechend verarbeitet werden können. Des Weiteren muss die App auf die Kommunikationsschnittstelle des Fahrzeugs, beispielsweise Bluetooth, angepasst werden.

AppViews

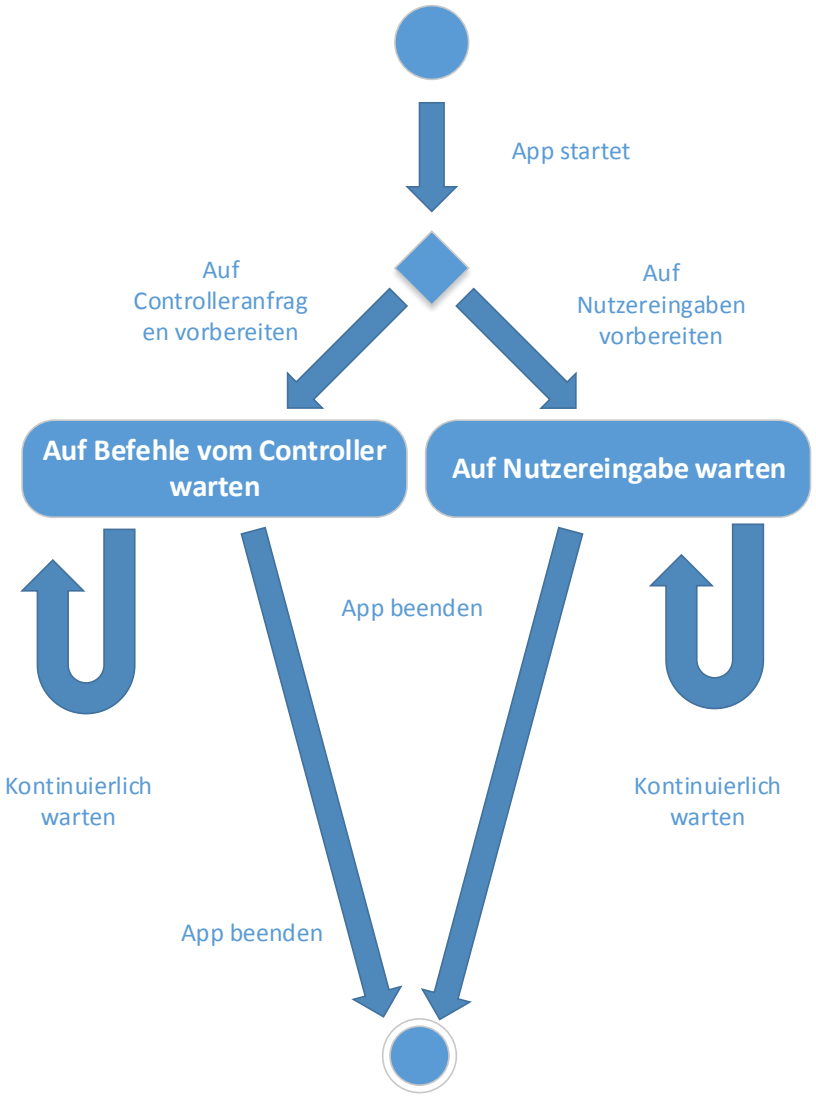

Abbildung 3.2: State-Chart zu Komponente **C10** AppViews

Die Abbildung 3.2 zeigt die möglichen Zustände der Komponente **C10** AppViews, welche die Verwaltung der grafischen Oberfläche übernimmt. Sie wartet sowohl auf Bildschirmeingaben des Nutzers als auch auf Befehle der Komponente **C30** Controller. Sobald entweder der Nutzer eine Eingabe vornimmt oder der Controller eine Änderung der Oberfläche veranlasst wird in der Komponente AppViews die entsprechende XML-Datei geladen und die grafische Oberfläche entsprechend aktualisiert.

Classifier

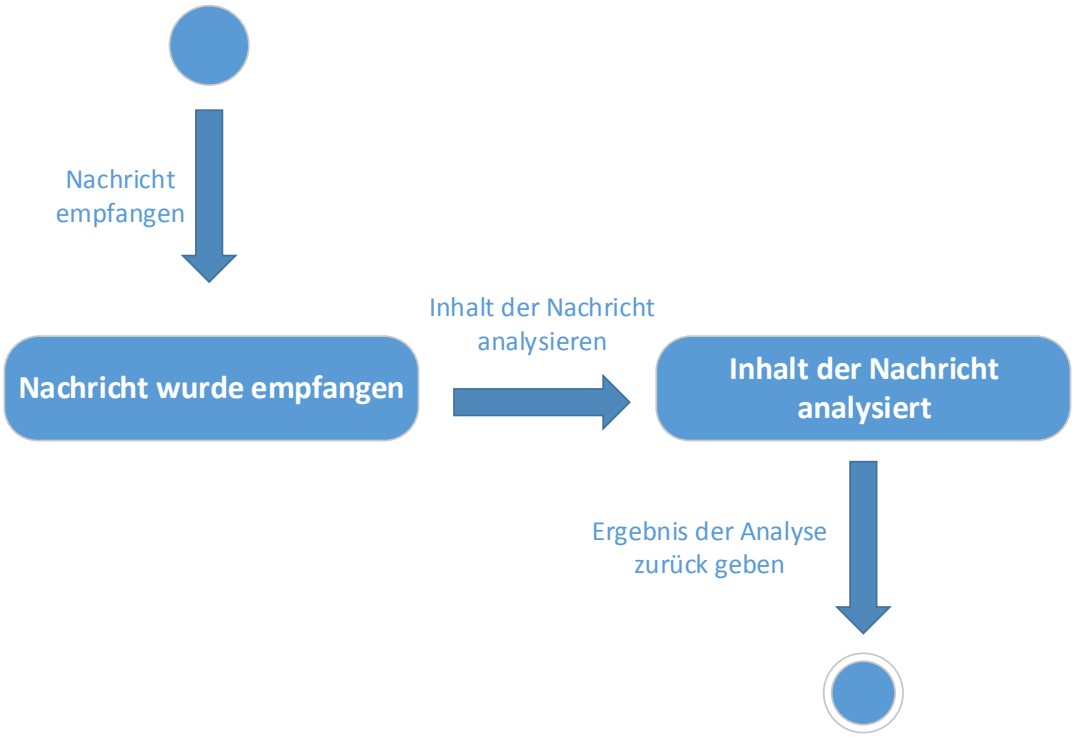

Abbildung 3.3: State-Chart zu Komponente **C60** Classifier

Die Abbildung 3.3 zeigt alle möglichen Zustände der Komponente **C60** *Classifier*, welche sich um die inhaltliche Analyse der empfangenen Nachrichten kümmert. Sobald der Classifier eine SMS- oder Facebook Messenger-Nachricht von der Komponente **C50** *Receivers* übergeben bekommt, wird der Inhalt der Nachricht analysiert. Das Ergebnis der Analyse wird an die aufrufende Komponente zurückgegeben.

Receivers

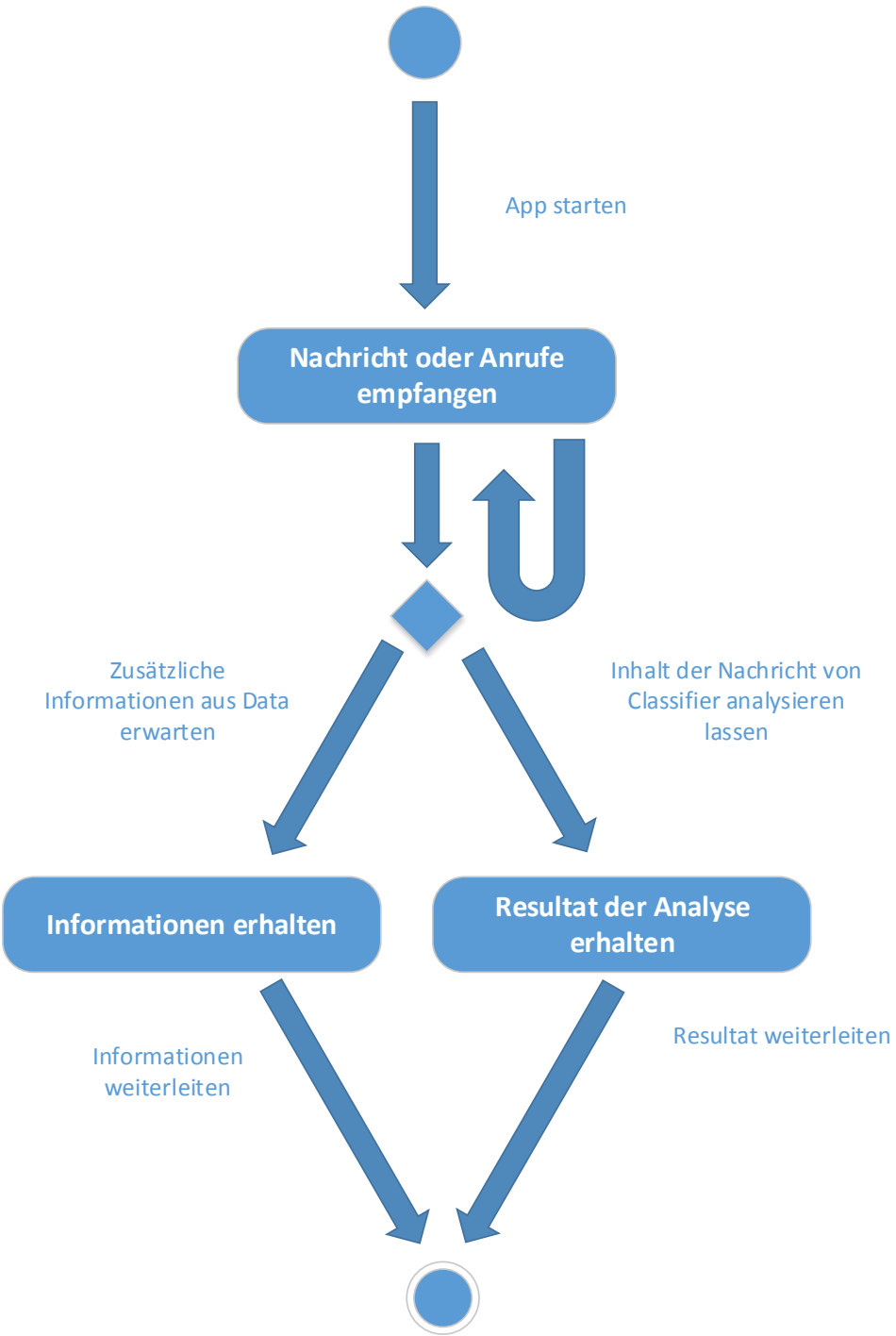

Abbildung 3.4: State-Chart zu Komponente **C50** Receivers

Die Abbildung 3.4 zeigt alle möglichen Zustände der Komponente **C50** *Receivers*, welche sich um die Weiterleitung von eingehenden Nachrichten und die automatische Standardantwort auf Anrufe und Nachrichten kümmert. Die Komponente überprüft dauerhaft ob neue Anrufe, SMS- oder Facebook Messenger-Nachrichten eingehen. Sobald eine Nachricht oder ein Anruf eintrifft wird automatisch der Versand einer automatischen Standardantwort an die eingehende Rufnummer eingeleitet. Beim Eingang einer Nachricht wird der Inhalt selbiger an die Komponente **C60** *Classifier* übergeben, um ihn inhaltlich auf Dringlichkeit überprüfen zu lassen. Gleichzeitig wird aus der Komponente **C20** *Data* der Name des Anrufers oder Verfassers der Nachricht abgefragt und überprüft, ob es sich bei diesem um einen Favoriten handelt. Diese Informationen werden gemeinsam mit dem Resultat der Klassifizierung an die Komponente **C30** *Controller* weitergeleitet.

3.4.2 ECU

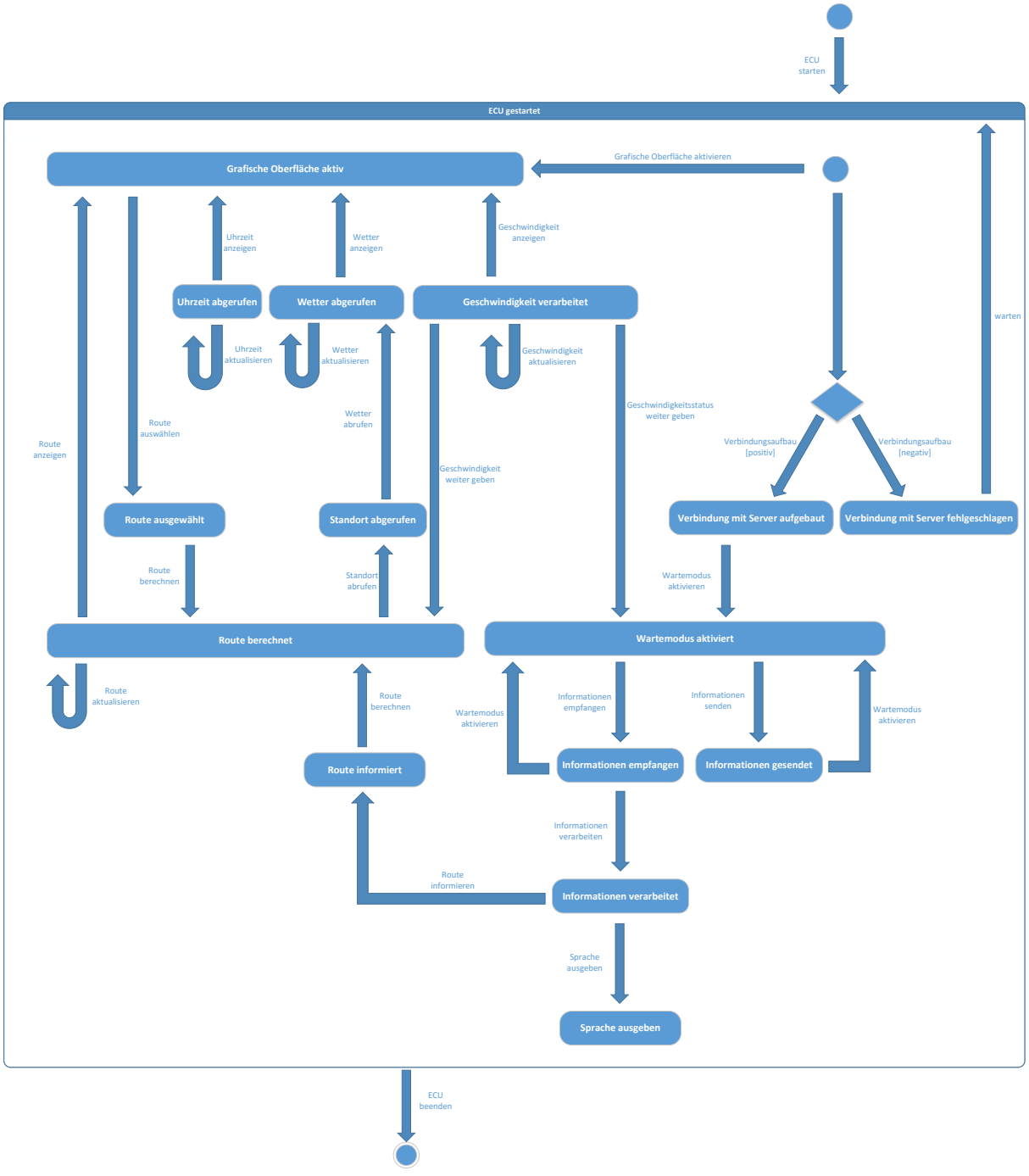

Abbildung 3.5: State-Chart zu Komponente **C90** ECU

Die Abbildung 3.5 zeigt die möglichen Zustände der Komponente **C90** *ECU*, welche sich um den gesamten Ablauf der ECU kümmert. In dem Diagramm ist zu sehen, dass die ECU zunächst einmal gestartet werden muss. Ist dies der Fall folgt die Aktivierung der grafischen Oberfläche,

sowie zeitgleich der Versuch eines Verbindungsaufbaus mit dem Webserver. Ist dieser negativ wird dieser Schritt wiederholt. Bei einem positiven Verbindungsaufbau hingeben wird eine Wartemodus aktiviert, welcher darauf wartet Informationen von dem Webserver zu erhalten oder welche zusenden. Auf der grafischen Oberfläche wird derweil das Wetter, die Uhrzeit und die Geschwindigkeit angezeigt, welche sich auch automatisch aktualisieren. Zudem wird es über die grafische Oberfläche ermöglicht eine Route auszuwählen. Wenn dies getan wird, wird eine Route berechnet und wieder auf der grafischen Oberfläche fortlaufend aktualisiert. Dabei erhält die Route seine Bewegungsinformationen von der Geschwindigkeit. Des Weiteren kann über die Route der Standort abgerufen werden, welcher dafür benötigt wird, um das Wetter richtig abzurufen. Der Geschwindigkeitsstatus wird bei bestimmten Events an den Webserver weiter geleitet. Dieser Status ist für den Wechsel zwischen dem „Silent“- und dem „Driving“-Modus wichtig. Jedoch kann die ECU auch Informationen von dem Webserver empfangen, dies geschieht wenn die App eine neue dringlichen Nachricht erkannt hat. In der ECU wird dann diese Information bearbeitet und als Sprache ausgegeben, zudem wird die Route informiert und ggf. angepasst. Sowie beim Senden, als auch beim Empfangen von Informationen wird der Wartemodus unterbrochen, aber nach dem Senden bzw. Empfangen wieder aktiviert.

4 Verteilungsentwurf

In der folgenden Abbildung 4.1 wird die Verteilung der einzelnen Komponenten in einem Verteilungsdiagramm beschreiben.

Die App läuft auf einem Smartphone, welches mit dem Android Betriebssystem der Version 4.3 oder höher ausgestattet sein muss. Die Komponenten der App werden dann von der darauf laufenden Java 7 Virtual Machine ausgeführt. Für die Klassifizierung von Nachrichten als dringlich oder nicht dringlich wird von der App ein importierter Algoritmus, der Naive Bayes Classifier von der Universität Waikatos in Neuseeland, namens WEKA (Waikato Environment for Knowledge Analysis), genutzt (siehe Komponente **C60** *Classifier*).

Die Kommunikation zwischen der App und der ECU läuft über einen Webserver mittels TCP/IP Protokoll. Der Webserver wird mit Node.js auf Heroku erstellt, dies wird näher erläutert in Kapitel 7. App und ECU nutzen dabei den Socket.io Client, um Events über Websockets zu Senden und zu Empfangen.

Die ECU läuft auf jedem beliebigen Betriebssystem, welches den Webbrowser Google Chrome besitzt. Sie greift auf die API von Google Maps für die Implementierung der Navigation zu. Außerdem wird auf die Funktion aus der Javascript-Bibliothek jQuery simpleWeather zur Darstellung der Wetterlage zugegriffen.

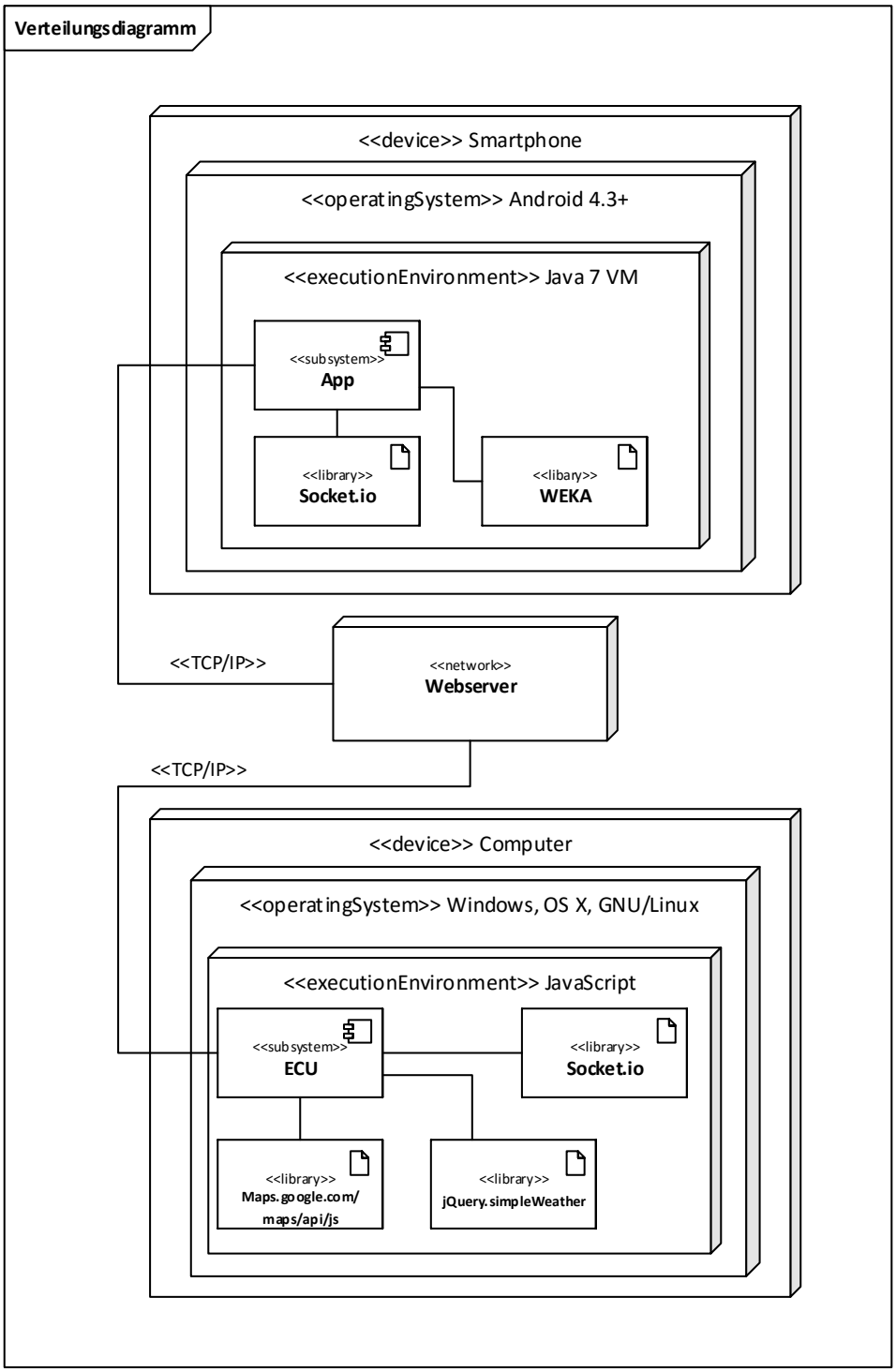

Abbildung 4.1: Verteilungsdiagramm „SocialPal“

5 Implementierungsentwurf

Die folgende Abbildung 5.1 gibt einen Überblick über alle Klassen des Subsystems App.

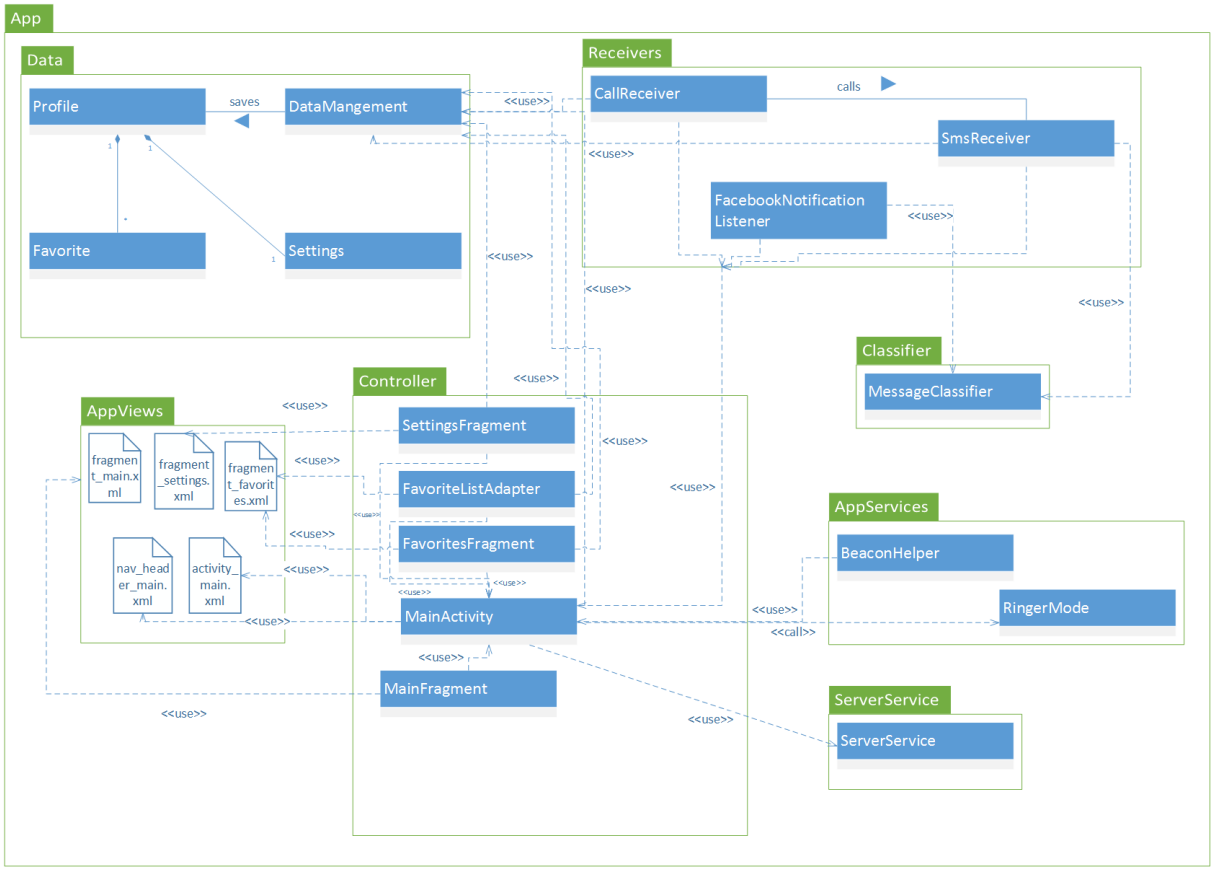

Abbildung 5.1: Klassendiagramm des Subsystems App

5.1 Implementierung von Komponente C10: AppViews

Die Komponente AppViews kümmert sich um die Darstellung der Oberfläche. In Form von XML Dateien sind die verschiedenen Layouts(Seiten) der Applikation definiert. Diese XML Dateien werden von der Komponente **C30** *Controller* abhängig von den Nutzereingaben aufgerufen und mit Funktionen gefüllt.

5.1.1 Paket-/Klassendiagramm

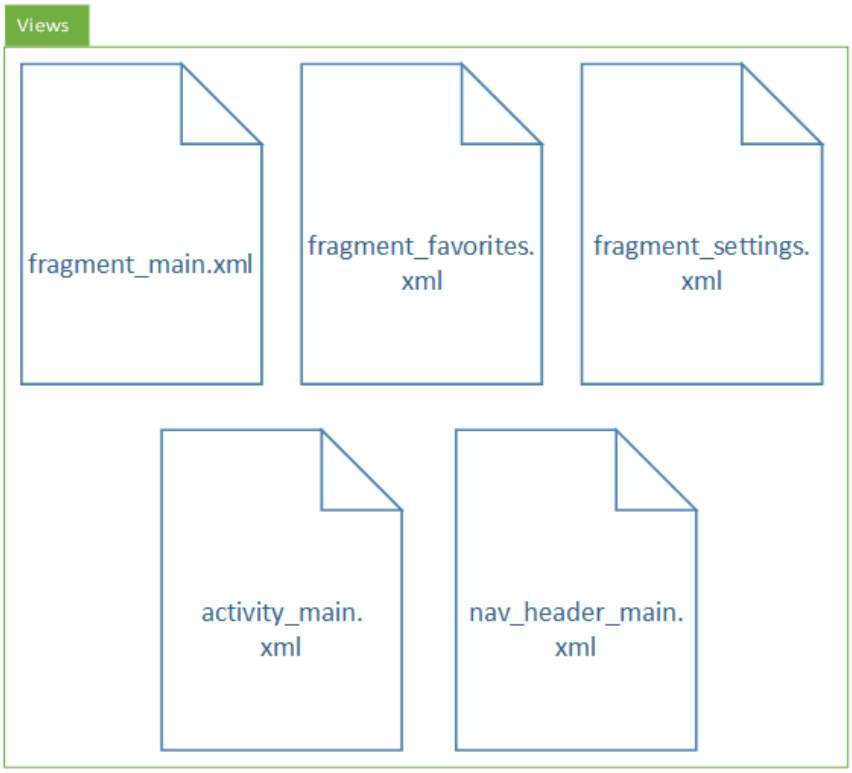

Abbildung 5.2: Klassendiagramm der Komponente AppViews **C10**

5.1.2 Erläuterung

Abbildung 5.3: Screenshot der Layout-Datei fragment_main.xml

fragment_main.xml⟨*CL*10⟩

Aufgabe

Darstellung des App Hintergrundes abhängig von dem aktuellen Modus, „Silent"-Modus oder „Driving"-Modus, in dem sich das Smartphone aktuell befindet.

Kommunikationspartner

C30 *Controller*

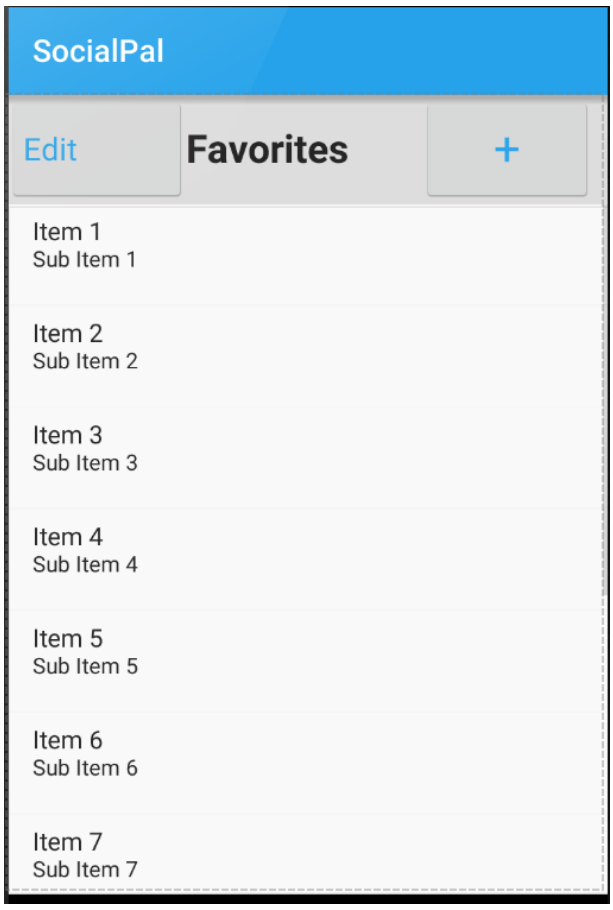

Abbildung 5.4: Screenshot der Layout-Datei fragment_favorites.xml

fragment_favorites.xml⟨*CL*20⟩

Aufgabe

Darstellung des Hintergrundes für das Aus- und Abwählen von Favoriten aus der Kontaktliste des Smartphones.

Kommunikationspartner

C30 *Controller*

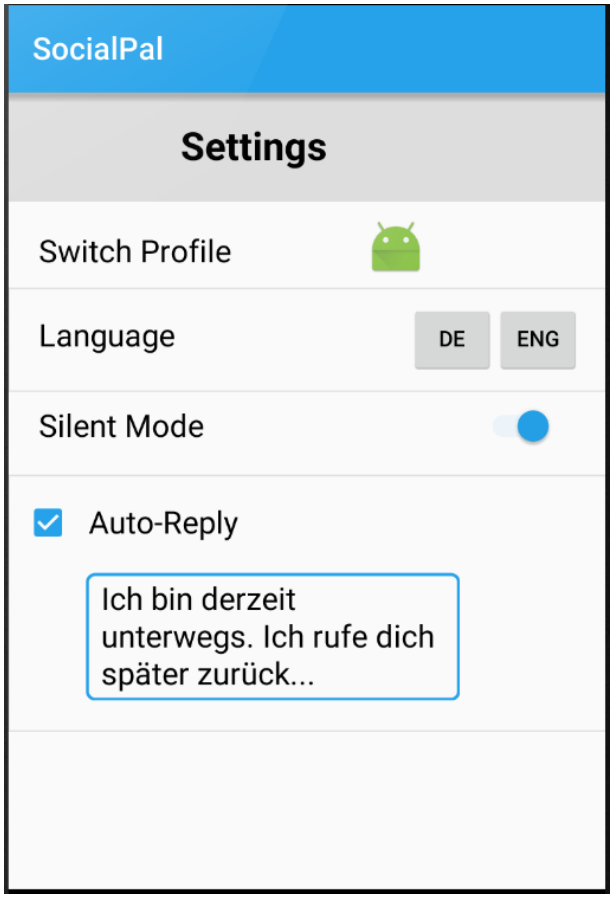

Abbildung 5.5: Screenshot der Layout-Datei fragment_settings.xml

fragment_settings.xml⟨*CL*30⟩

Aufgabe

Darstellung der App Einstellungen zum Wechseln der Sprache, manuellen Ein- bzw. Ausstellen des „Silent"-Modus, De- und Aktivieren der automatischen Standardantwort und Verändern dieser.

Kommunikationspartner

C30 *Controller*

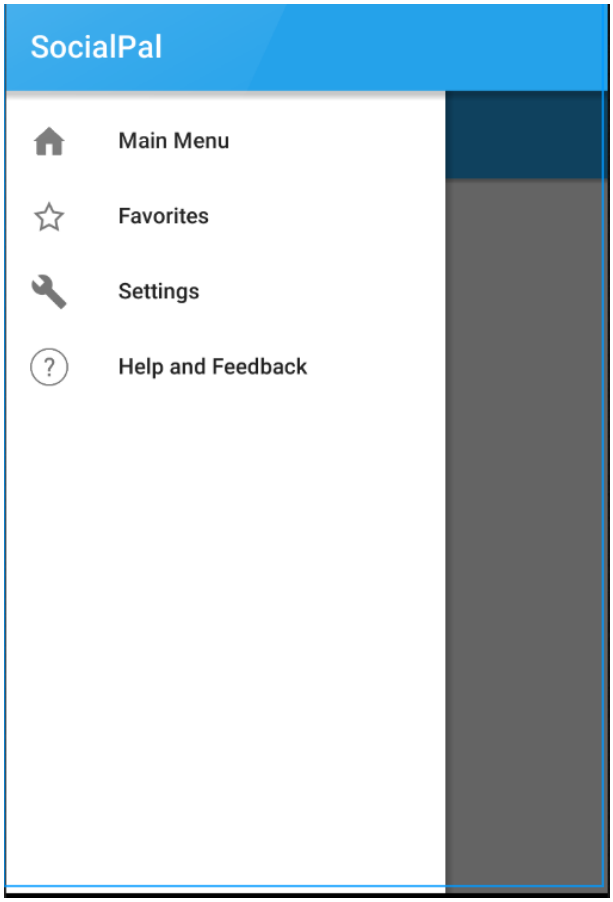

Abbildung 5.6: Screenshot der Layout-Datei activity_main.xml

activity_main.xml⟨*CL*40⟩

Aufgabe

Darstellung der App Oberfläche zum Wechseln zwischen dem Hauptbildschirm, den Favoriten und den Einstellungen.

Kommunikationspartner

C30 *Controller*

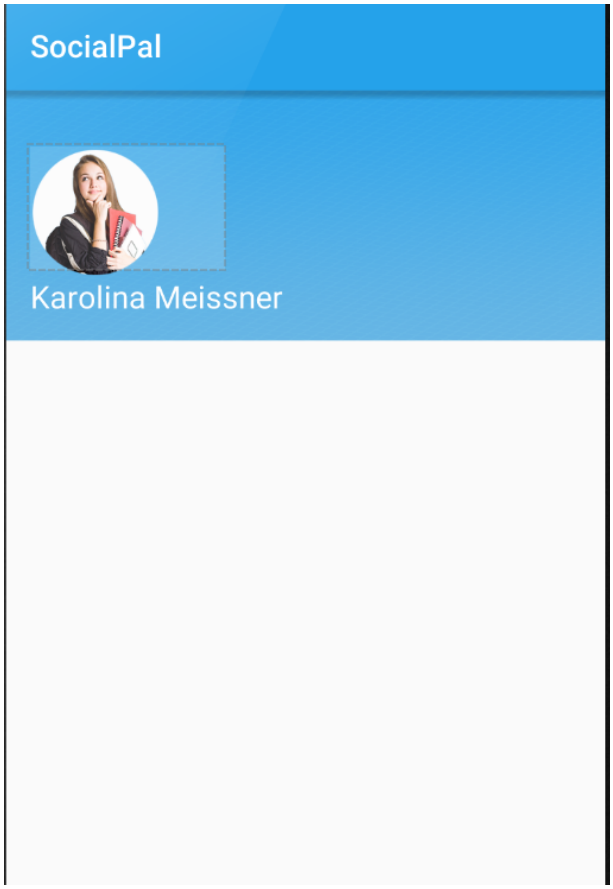

Abbildung 5.7: Screenshot der Layout-Datei nav_header_main.xml

nav_header_main.xml⟨*CL*50⟩

Aufgabe

Darstellung eines Fotos des aktuellen Nutzers der App inklusive seines Namen über dem Navigationsmenü.

Kommunikationspartner

C30 *Controller*

5.2 Implementierung von Komponente C20: Data

Die Komponente Data verwaltet die von der App gespeicherten Daten und bietet eine Schnittstelle zu der lokalen Datenbank des Smartphones.

5.2.1 Paket-/Klassendiagramm

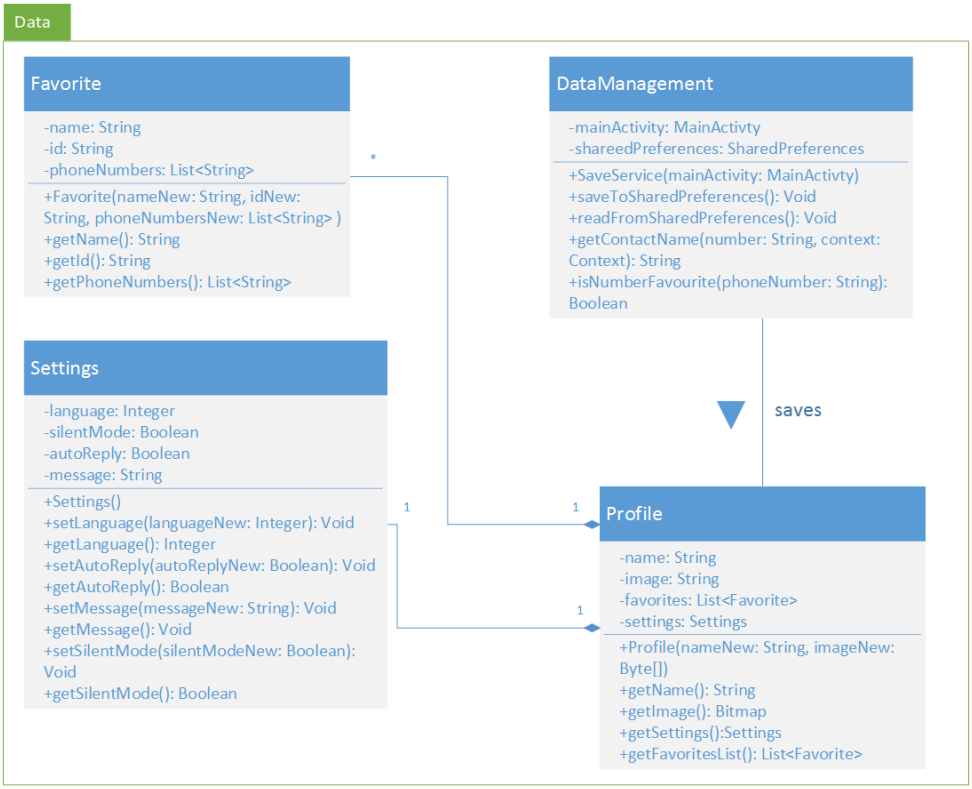

Abbildung 5.8: Klassendiagramm der Komponente **C20** Data

5.2.2 Erläuterung

Favorite⟨$CL10$⟩

Aufgabe

Dient der Speicherung der individuell auswählbaren Favoriten.

Attribute

String name Name
String id Id
List<String> phoneNumber Telefonnummern

Operationen

Favorite(String nameNew, String idNew, List<String> phoneNumbersNew) : Der favorisierte Kontakt wird mit seiner Id, seinem Namen und den eingespeicherten Telefonnummern gespeichert.

Profile⟨$CL20$⟩

Aufgabe

Dient der Speicherung des Benutzerprofils.

Attribute

String name Name
String image Bild
List<Favorite> favorites Liste von Favoriten
Settings settings Objekt der Klasse Settings zur Speicherung der Einstellungen

Operationen

Profile(String nameNew, Byte[] imageNew) : Dem Konstruktor wird der Name und das Bild als Byte[] Array, welche umgehend in eine neue Bitmap umgewandelt wird übergeben. Außerdem wird ebenfalls die Liste der favorisierten Kontakte und das Objekt Einstellungsklasse erstellt.

DataManagement⟨*CL*30⟩

Aufgabe

Dient dem Zugriff auf die gespeicherten Daten und der Komponente **C100** *MobileDB*.

Attribute

MainActivity mainActivity Eine Instanz der Klasse MainActivity um auf die SharedPreferences der App zuzugreifen.

SharedPreferences sharedPreferences Eine Instanz der Klasse SharedPreferences. Ermöglicht das Speichern von primitiven Datentypen.

Operationen

saveToSharedPreferences() : Speichert die aktuell geänderten Einstellungen in den SharedPreferences.

readFromSharedPreferences() : Gibt die aktuellen Einstellungen zurück.

Boolean isNumberFavourite(String phoneNumber) : Überprüft, ob die übergebene Telefonnummer als Favorit eingespeichert ist.

String getContactName(String phoneNumber, Context context) : Falls die übergebene Telefonnummer nicht in der lokalen Kontaktdatenbank des Smartphones eingespeichert ist, gibt die Methode den String „notInContacts“ zurück. Andernfalls gibt sie den Kontaktnamen zurück.

Kommunikationspartner

C30 *Controller*

Settings⟨*CL*40⟩

Aufgabe

Dient der Speichert der individuellen Einstellungen des Benutzers.

Attribute

Integer language Sprache

Boolean silentMode Boolischer Wert, welcher speichert, ob das optionale Feature des manuellen Ausschaltens des „Silent“-Modus aktiviert ist.

Boolean autoReply Boolischer Wert, welcher speichert, ob das optionale Feature „Standardantwort senden“ des „Silent“-Modus aktiviert ist

String message Speichert die individuell konfigurierbare Nachricht des Features „Standardantwort senden“.

5.3 Implementierung von Komponente C30: Controller

Die Komponente Controller dient der Steuerung der App. Sie ist die Verbindung zwischen **C10** *App Views*, **C20** *Data*, **C40** *AppServices*, **C50** *Receivers* und **C70** *ServerService*.

5.3.1 Paket-/Klassendiagramm

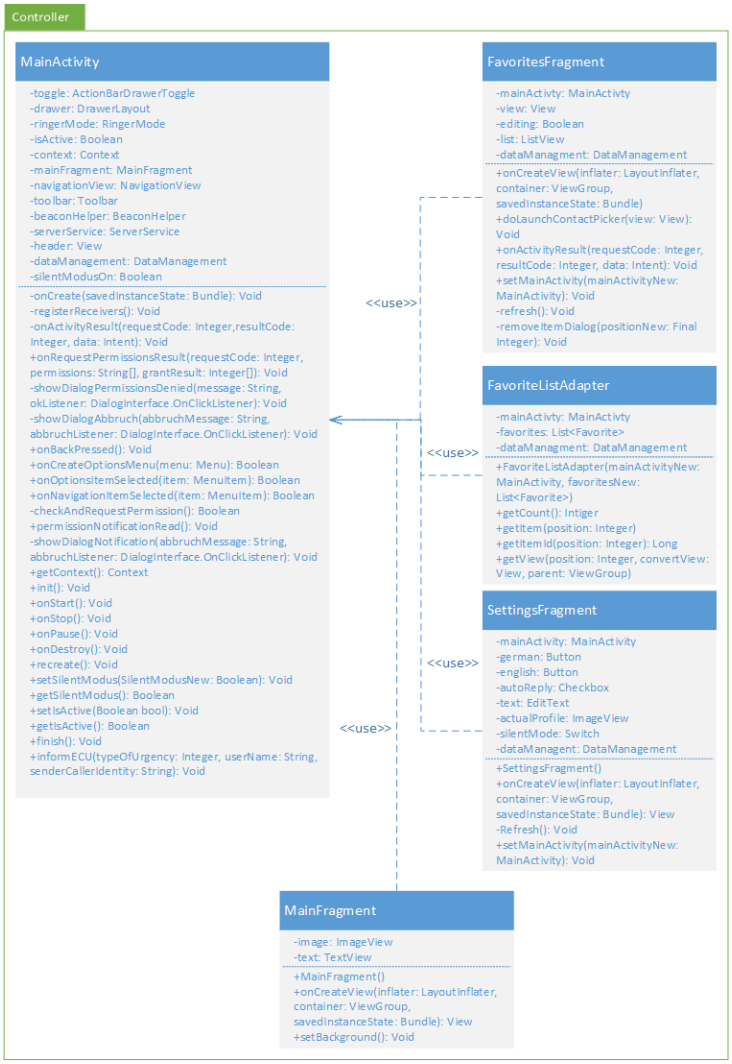

Abbildung 5.9: Klassendiagramm der Komponente **C30** Controller

5.3.2 Erläuterung

MainActivity⟨$CL10$⟩

Aufgabe

Die MainActivity regelt die Nachfrage nach den Berechtigungen und die Kommunikation zu anderen Klassen.

Attribute

ActionBarDrawerToggle toggle Eine Instanz der Klasse ActionBarDrawerToggle zur Darstellung des Öffnen und Schließen der Navigationsansicht.
DrawerLayout drawer Eine Instanz der Klasse DrawerLayout des Containers der Hauptansicht der App.
RingerMode ringerMode Eine Instanz der Klasse RingerMode.
Boolean isActive Boolischer Wert, welcher zeigt, ob die App aktiv ist.
Context context Systemeigenes Attribut, welches vom Android-Betriebssystem zugewiesen wird und Aufschluss über den Zusammenhang der Instanz gibt.
MainFragment mainFragment Ein Instanz der Klasse MainFragment zur Darstellung des Hauptmenüs.
NavigationView navigationView Eine Instanz der Klasse NavigationView zur Darstellung der Navigationsansicht.
Toolbar toolbar Eine Instanz der Klasse Toolbar zur Darstellung der Funktionsleiste.
BeaconHelper beaconHelper Eine Instanz der Klasse BeaconHelper.
ServerService serverService Eine Instanz der Klasse ServerService.
View header Eine Instanz der Klasse View zur Darstellung der Kopfzeile der Oberfläche.
DataManagement dataManagement Eine Instanz der Klasse DataManagement.
Emitter.Listener newMessageServer Eine Instanz der Klasse Emitter.Listener, welche auf ein Event des serverService hört.
Boolean silentModus Boolischer Wert, welcher zeigt, ob der „Silent“-Modus aktiv ist.

Operationen

onCreate(Bundle savedInstanceState) : Die onCreate Methode wird beim Start der Application ausgeführt und ist dafür zuständig, andere Methoden wie z.B. die Initiierung der Verbindung zum Server oder die Nachfrage für die benötigten Berechtigungen nach dem Start der App auszuführen.
registerReceivers() : Initiiert den Sms- und CallReceiver in der MainActivity.
onActivityResult(Integer requestCode, Integer resultCode, Intent data) :
onRequestPermissionsResult(Integer requestCode, String[] permissions, Integer[] grantResult) : Überprüft welche Eingaben der Nutzer bei der Nachfrage nach Berechtigungen für die App gemacht hat.
Beim Zulassen der Berechtigungen wird die init() Methode des BeaconHelper ausgeführt.

Nach Zulassen aller Berechtigungen außer der Standort-Berechtigung wird auf eingeschränkte Funktionalität hingewiesen und die init() Methode des BeaconHelper aufgerufen.
Beim erstmaligem Ablehnen der Berechtigungen wird eine Fehlermeldung ausgegeben und erneut nach den Berechtigungen gefragt, bzw. die Möglichkeit zum Schließen der App gegeben.
Falls ausgewählt wird, dass nicht wieder nach den Berechtigungen gefragt werden soll, und die Berechtigung abgelehnt wird, wird die App mit einer Fehlermeldung automatisch geschlossen.
showDialogOK(String message, DialogInterface.OnClickListener okListener) : Gibt eine Dialog Einblendung mit der Möglichkeit 'OK' oder 'Close app' auszuwählen.
showDialogAbbruch(String abbruchMessage, DialogInterface.OnClickListener abbruchListener) : Gibt eine Dialog Einblendung mit der Möglichkeit 'OK' auszuwählen.
onBackPressed() : Schließt die App bei Klicken der 'zurück' Taste des Smartphone, sofern diese aktiv war.
onCreateOptionsMenu(Menu menu) : Füllt die Aktionsleiste der App mit Objekten.
onOptionsItemSelected(MenuItem item) : Kümmert sich um die Klicks auf die Aktionsliste.
onNavigationItemSelected(MenuItem item) : Füllt die Navigationsliste der App und zeigt bei Klick das ausgewählte Fragment der App an.
checkAndRequestPermission() : Fragt nach den Berechtigungen für SMS, Telefon, Kontakte und Standort, falls diese nicht bereits gegeben sind.
permissionNotificationRead() : Fragt nach der Berechtigung alle Benachrichtigungen lesen zu dürfen und leitet den Nutzer in die Einstellungen weiter, sofern diese nicht bereits gegeben ist.
showDialogNotification(String abbruchMessage, DialogInterface.OnClickListener abbruchListener) : Gibt eine Dialog Einblendung mit der Möglichkeit 'OK' auszuwählen.
init() : Initiiert ein BeaconHelper Objekt, wenn nicht bereits vorhanden und startet die Init() Methode des BeaconHelper, sofern alle Berechtigungen geben sind.
onStart() : Wird aufgerufen wenn die App geöffnet wird und verbindet sich daraufhin über den ServerService mit dem Webserver, wenn nicht bereit verbunden.
onStop() : Wird aufgerufen wenn die App beendet wird. Bei diesem Löschen der App aus dem Arbeitsspeicher, werden die App Daten gesichert und die Verbindung zum Webserver wird mittels des ServerService beendet.
onPause() : Wird aufgerufen wenn die App geschlossen wird. Bei diesem Pausieren der App, werden die Daten der App gesichert.
onDestroy() : Wird aufgerufen wenn die App gelöscht wird. Die Verbindung zum Webserver wird mittels des ServerService beendet.
recreate() : Startet die App neu beim Aufrufen der Methode.
finish() : Beim Aufruf wird die Variable setIsActive auf false gesetzt.

informECU(Integer typeOfUrgency, String userName, String senderCallerIdentity) : Erstellt ein 'JSONObject' für den Webserver und übergibt dieses an die Komponente **C70** *ServerService*

Kommunikationspartner

C10 *AppViews*
C20 *AppServices*
C70 *ServerService*
C50 *Receiver*
C20 *Data*

FavoritesFragment⟨*CL*20⟩

Aufgabe

Regelt das Hinzufügen, Löschen und Speichern der favorisierten Kontakte.

Attribute

MainActivity mainActivity Eine Instanz der Klasse MainActivity.

DataManagement dataManagement Eine Instanz der Klasse DataManagement.

View view Eine Instanz der Klasse View zur Darstellung der Favoriteneinstellungen.

Boolean editing Boolischer Wert, welcher zeigt, ob die Kontaktliste des Smartphone bearbeitet werden kann.

ListView list Eine Instanz der Klasse ListView zur Darstellung der Favoritenliste.

Operationen

onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) : Füllt die Oberfläche des Favoriten Fragments mit Funktionalitäten.

doLaunchContactPicker(View view) : Wird aufgerufen, wenn der Benutzer einen Kontakt zu seinen Favoriten hinzufügen möchte und öffnet einen Dialog zur Auswahl der Kontakte aus dem lokalen Kontaktbuch des Smartphones.

onActivityResult(Integer requestCode, Integer resultCode, Intent data) : Bei Auswahl eines Kontaktes in der Kontaktliste wird die Kontaktnummer zu den favorisierten Kontakten hinzugefügt.

removeItemDialog(Final Integer positionNew) : Fragt den Benutzer ob er einen ausgewählten Kontakt aus den Favoriten löschen möchte und löscht diesen beim Bestätigung durch den Benutzer.

Kommunikationspartner

C20 *Data*

C10 *AppViews*

FavoriteListAdapter⟨*CL*30⟩

Aufgabe

Anzeigen der Liste der favorisierten Kontakte.

Attribute

MainActivity mainActivity Eine Instanz der Klasse MainActivity.

List<Favorite> favorites Eine Liste, welche die momentan ausgewählten Favoriten beinhaltet.

DataManagement dataManagment Eine Instanz der Klasse DataManagement.

Operationen

FavoriteListAdapter(MainActivity mainActivityNew, List<Favorite> favoritesNew) : Initiiert ein FavoritListAdapter Objekt.

Kommunikationspartner

C20 *Data*

C10 *AppViews*

MainFragment⟨*CL*40⟩

Aufgabe

Das Hintergrundbild der App in Abhängigkeit zum aktuellen Modus ändern.

Attribute

ImageView image Ein ImageView, welcher ein Bild für den „Driving“-Modus enthält.

TextView text Ein TextView, welcher den Text für den „Driving“-Modus enthält.

Operationen

MainFragment() : Initiiert den Konstruktor MainFragment.

onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) : Füllt die Oberfläche für das MainFragment.

setBackground() : Ändert den Hintergrund der App in Abhängigkeit von der Entfernung zum Beacon und der Geschwindigkeit des Fahrzeuges.

Kommunikationspartner

C10 *AppViews*

SettingsFragment⟨$CL50$⟩

Aufgabe

Darstellung und Implementierung der Funktionalitäten der Einstellungsansicht für den Benutzer.

Attribute

MainActivity mainActivity Eine Instanz der Klasse MainActivity.
Button german Eine Instanz der Klasse Button, welcher die Funktionalität implementiert auf Wunsch des Benutzers die Sprache der App auf Deutsch umzustellen.
Button english Eine Instanz der Klasse Button, welcher die Funktionalität implementiert auf Wunsch des Benutzers die Sprache der App auf Deutsch umzustellen.
Checkbox autoReply Eine Checkbox für das De- und Aktivieren des Sendens einer Standardantwort.
EditText text Ein Textfeld, zum Anzeigen und Bearbeiten der Standardantwort.
ImageView actualProfile Ein ImageView, welcher ein Bild des aktuell ausgewählten Profils enthält.
Switch silentMode Ein Regler, welcher regelt, ob der „Silent"-Modus aktiviert ist.
DataManagement dataManagement Eine Instanz der Klasse DataManagement.

Operationen

SettingsFragment() : Initiiert den Konstruktor SettingsFragment.
onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) : Füllt die Oberfläche des Settings Fragment, ermöglicht das Ändern der Sprache, das Ändern und Speichern der Standardantwort, sowie das manuelle Aktivieren/Deaktivieren des „Silent"-Modus.
refresh() : Aktualisiert die Oberfläche des Settings Fragments.

Kommunikationspartner

C10 *AppViews*
C20 *Data*

5.4 Implementierung von Komponente C40: AppServices

Die Komponente AppServices implementiert die Funktionalität die Audioprofile des Smartphones zu ändern und nach Beacons zu suchen, sowie auf diese zu reagieren.

5.4.1 Paket-/Klassendiagramm

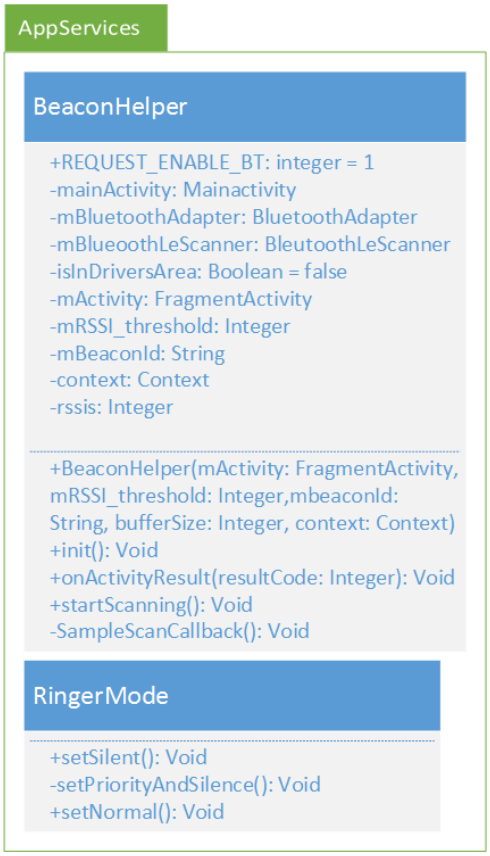

Abbildung 5.10: Klassendiagramm der Komponente **C40** AppServices

5.4.2 Erläuterung

BeaconHelper⟨$CL10$⟩

Aufgabe

Sucht nach Beacons in der Nähe über Bluetooth Low Energy und reagiert auf diese.

Attribute

Integer REQUEST_ENABLE_BT: Wert, welcher dem Android System mitteilt, dass eine Anfrage zum Aktivieren von Bluetooth ausgegeben werden soll.

MainActivity mainActivity Eine Instanz der Klasse MainActivity.
BluetoothAdapter mBluetoothAdapter Instanz, welche den Zugriff auf das im Smartphone verbaute Bluetooth Modul gibt.
BleutoothLeScanner mBlueoothLeScanner Eine Instanz des BluetoothLeScanners, welcher nach Bluetooth Low Energy Signalen scannt.
Boolean isInDriversArea Wert, welcher Auskunft darüber gibt, ob die Signalstärke des Beacons den Schwellenwert des Fahrerbereichs überschreitet oder nicht.
FragmentActivity mActivity Eine Instanz der Klasse FragmentActity, um auf die Systemservices des Smartphones zuzugreifen.
Integer mRSSI_threshold Übergebener Wert, welcher den Schwellenwert des Fahrerbereichs definiert.
String mBeaconId Die ID des Beacons welcher verwendet wird.
Context context Systemeigenes Attribut, welches vom Android Betriebssystem zugewiesen wird und Aufschluss über den Zusammenhang der Instanz gibt.
Integer rssis Berechneter Durchschnitt der empfangenen Signalstärke des Beacons. Dies erlaubt eine konstante Abgrenzung der Zonen im Fahrzeug.

Operationen

BeaconHelper(FragmentActivity mActivity, Integer mRSSI threshold, String mbeaconId, Integer bufferSize, Context context) : Dies ist der Konstruktor der Klasse BeaconHelper. Er wird bei der Erzeugung einer neuen Instanz der Klasse BeaconHelper automatisch aufgerufen und übergibt die ihm übermittelten Parameter an die Klasseninstanz.
init() : Initialisierungsmethode, welche das im Smartphone verbaute BluetoothModul initialisiert und überprüft, ob Bluetooth eingeschaltet ist. Sollte Bluetooth ausgeschaltet sein, so wird eine Anfrage mit der Bitte nach Aktivierung an das Android Betriebssystem gesendet. Wenn Bluetooth aktiviert ist wird überprüft, ob das verbaute BluetoothModul den Bluetooth Low Energy Standard unterstützt. Ist dies der Fall, so wird automatisch die Methode startScanning() aufgerufen.
onActivityResult(Integer resultCode) :
startScanning() : Diese Methode löst den eigentlichen Scan-Vorgang aus. Sie empfängt das vom Beacon ausgestrahlte Signal und reagiert auf die empfangenen Werte. So startet sie beim Betreten des Sendebereichs die App, informiert die Controller-Komponente der App über die Entfernung (in Form von Booleans) und schließt beim Verlassen des Sendebereichs die App.

Kommunikationspartner

C30 *Controller*

RingerMode⟨*CL*20⟩

Aufgabe

Wechseln des Audioprofiles des Smartphones in Abhängigkeit des aktuellen Modus.

Attribute

keine

Operationen

setSilent() : Stellt das Smartphone stumm.

setNormal() : Setzt das Smartphone in den normalen lauten Modus.

Kommunikationspartner

C30 *Controller*

5.5 Implementierung von Komponente C50: Receivers

Die Komponente Receivers implementiert die Funktionalitäten des Abfangens von SMS, Anrufen und Mitteilungen über den Facebook Messenger. Außerdem behandelt sie das optionale Senden einer Standardantwort an den Anrufer bzw. Absender einer Nachricht.

5.5.1 Paket-/Klassendiagramm

Hier werden die Klassen der Komponente anhand eines Klassendiagramms und einer näheren Erläuterung der Klassen dargestellt.

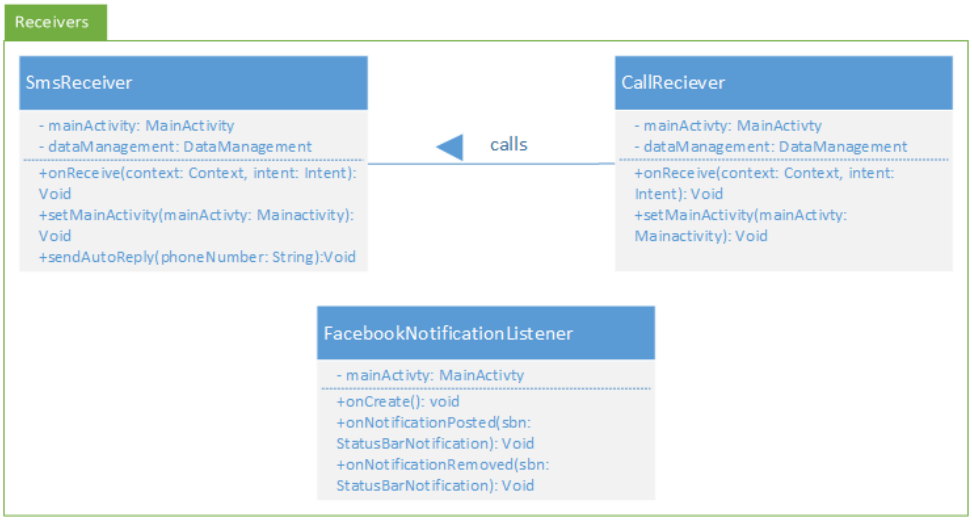

Abbildung 5.11: Klassendiagramm der Komponente **C50** Receivers

5.5.2 Erläuterung

CallReceiver⟨$CL10$⟩

Aufgabe

Abfangen und Ablehnen von Anrufen, sofern sich die App im „Silent“-Modus befindet.

Attribute

MainActivity mainActivity Eine Instanz der Klasse MainActivity.
DataManagement dataManagement Eine Instanz der Klasse DataManagement.

Operationen

onReceive(Context context, Intent intent) : Wird aufgerufen, sobald ein Anruf eingeht und dieser wird sofern sich die App im „Silent“-Modus befindet abgelehnt. Daraufhin wird über das DataManagement überprüft, ob der Benutzer das optionale Feature Standardantwort aktiviert hat, und sendet in diesem Fall die Nachricht an den Anrufer. Folgend wird überprüft, ob der Anrufer unter den Favoriten gespeichert und im Falle der Bestätigung wird dies an die Controller Komponente zum Informieren der ECU weitergeleitet.
setMainActivity(MainActivity mainActivity) Setzt die Instanz der MainActivity.

Kommunikationspartner

C30 *Controller* Kommuniziert mit der Controller Komponente, um den „Silent“-Modus abzufragen und über einen dringlichen Anruf zu informieren.
C20 *Data* Wird genutzt, um den Status und die Nachricht des Features Standardantwort, die Liste der Favoriten und den Kontaktnamen des Anrufers abzufragen.
C50 *SmsReceiver* SmsReceiver.sendAutoReply(String phoneNumber, String message) wird aufgerufen, um eine Nachricht zu versenden.

SmsReceiver⟨*CL*20⟩

Aufgabe

Abfangen von Mitteilungen(SMS), das Weiterleiten des Inhalts an die Classifier Komponente und das Senden einer Standardantwort, sofern sich die App im „Silent“-Modus befindet.

Attribute

MainActivity mainActivity Eine Instanz der Klasse MainActivity.
DataManagement dataManagement Eine Instanz der Klasse DataManagement.

Operationen

onReceive(Context context, Intent intent) : Wird ausgelöst, wenn eine SMS eingeht und falls der „Silent“-Modus der App aktiv ist, wird der Inhalt der Nachricht an die Komponente Classifier zur Überprüfung des Inhaltes auf Dringlichkeit weitergeleitet. Wenn die Nachricht als dringlich eingestuft wurde, wird der Kontaktname über den DataManagement bezogen und die Informationen an die Controller Komponente weitergeleitet. Im Falle der Aktivierung des Features Standardantwort wird die Methode sendAutoReply() aufgerufen.
sendAutoReply(String phoneNumber, String message) : Sendet die übergebene Nachricht an die übergebene Telefonnummer.
setMainActivity(MainActivity mainActivity) : Setzt die Instanz der MainActivity.

Kommunikationspartner

C30 *Controller* Kommuniziert mit der Controller Komponente um den „Silent“-Modus abzufragen und über eine dringliche Nachricht zu informieren.
C20 *Data* Wird genutzt um den Status und die Nachricht des Features Standardnachricht, die Liste der Favoriten und den Kontaktnamen des Anrufers abzufragen.
C60 *Classifier* Wird die zu klassifizierende Nachricht übergeben.

FacebookNotificationListener⟨$CL30$⟩

Aufgabe

Das Abfangen von Notifikationen des Facebook Messengers und das Weiterleiten des Inhaltes an die Komponente **C60** *Classifier*.

Attribute

MainActivity mainActivity Eine Instanz der Klasse MainActivity.

Operationen

onNotificationPosted(StatusBarNotification sbn) : Wird aufgerufen, wenn eine Notifikation eintrifft und filtert die Nachrichten des Facebook Messengers aus und übergibt den Inhalt an die Komponente Classifier zur Überprüfung der Dringlichkeit. Falls die Nachricht als dringlich eingestuft wird, wird folgend die Komponente **C30** *Controller* informiert.

Kommunikationspartner

C30 *Controller* Kommuniziert mit der Controller Komponente, um den „Silent“-Modus abzufragen und über eine dringliche Nachricht zu informieren.
C60 *Classifier* Wird die zu klassifizierende Nachricht übergeben.

5.6 Implementierung von Komponente C60: Classifier

Der Classifier untersucht die Inhalte der ihm übermittelten Nachrichten und prüft diese auf Dringlichkeit.

5.6.1 Paket-/Klassendiagramm

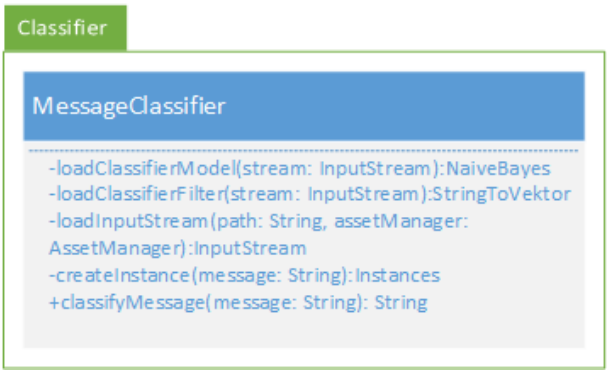

Abbildung 5.12: Klassendiagramm der Komponente **C60** Classifier

5.6.2 Erläuterung

MessageClassifier⟨$CL10$⟩

Aufgabe

Bestimmen der Dringlichkeit einer Nachricht.

Attribute

keine

Operationen

NaiveBayes loadClassifierModel(InputStream inputStream) : Lädt die zuvor trainierte Instanz des NaiveBayes Klassifizierungsalgorithmus.

StringToVektor loadClassifierFilter(InputStream inputStream) : Lädt einen StringToVektor-Filter, welcher auf die mit der zu klassifizierenden Nachricht erstellten Instanz angewandt wird. Dieser wandelt String Attribute in eine Gruppe von Attributen um, welche die Auftretenswahrscheinlichkeit repräsentieren.

Instances createInstance(String message) : Erstellt eine Instanz mit der zu klassifizierenden Nachricht und fügt diese in eine Liste von Instanzen(Instances) hinzu. Dies muss geschehen um den StringToVektor-Filter anzuwenden zu können.

String classifyMessage(String message) : Klassifiziert die ihm übergebene Nachricht und

gibt als Rückgabewert die Dringlichkeit in Form eines String Objektes zurück.

Kommunikationspartner

C50 *Receiver* Bekommt von der Komponente Receiver eine Nachricht zur Klassifizierung übermittelt.

5.7 Implementierung von Komponente C70: ServerService

Die Komponente ServerService stellt die Verbindung zum Webserver auf und empfängt und versendet Informationen an diesen.

5.7.1 Paket-/Klassendiagramm

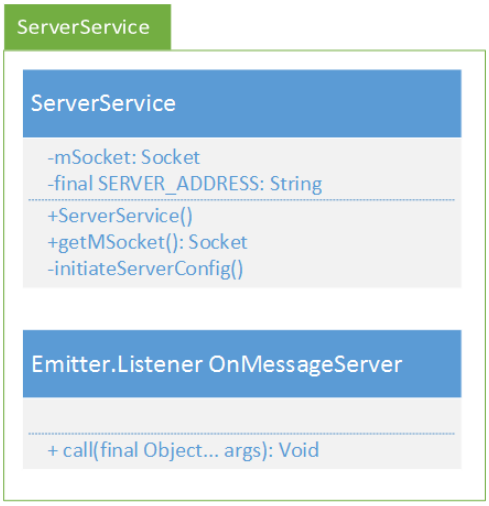

Abbildung 5.13: Klassendiagramm der Komponente **C70** ServerService

5.7.2 Erläuterung

Die Klasse ServerService bietet den Zugriff das Socket Objekt, über welches die Verbindung zum Server auf- und abgebaut werden kann. Außerdem können über den Socket Nachrichten an den Webserver übermittelt und von diesem empfangen werden.

ServerService$\langle CL10 \rangle$

Aufgabe

Aufbauen und Trennen der Verbindung zum Webserver. Und das Empfangen und Senden von Informationen an diesen.

Attribute

mSocket Socket Instanz der Klasse Socket der Socket.io.Client API.

final SERVERADDRESS String Ein festgelegter String, welcher die Adresse zu dem Webserver enthält.

Operationen

ServerService() : Initiiert das Socket Objekt bei der Initiierung des ServerServices.
Socket getSocket() : Die öffentliche Methode bietet die Möglichkeit auf die Socket Instanz zuzugreifen und somit die Schnittstelle dieser Komponente.

Kommunikationspartner

C30 *Controller*
Webserver

5.8 Implementierung von Komponente C80: Beacon

Die Komponente Beacon sendet ein ständiges Bluetooth Low Energy Signal aus, welches von anderen Bluetooth Geräten empfangen werden kann. Anhand dieser Signale kann ein Smartphone die ungefähre Entfernung zum Beacon ermitteln.

Beacon$\langle CL10 \rangle$

Aufgabe

Senden eines Bluetooth Low Energy Signales zur Ortung durch andere Bluetooth Geräte.

Kommunikationspartner

C130 *BluetoothServices*

5.9 Implementierung von Komponente C90: ECU

Die Komponente ECU ist der Boardcomputer eines Fahrzeugs. Sie dient zur Ausgabe und Anzeige von Informationen für den Benutzer.

Da die Programmierung der ECU in HTML, CSS und Javascript vorgenommen wurde und diese Programmiersprachen über keine Klassen im eigentlichen Sinne verfügen, dienen die Diagramme zur Visualisierung der Strukturen der ECU. Da die einzelnen Funktionalitäten in separaten Dateien aufgeteilt wurden, wurde auch die Aufteilung in den Diagrammen nach den Dateien vorgenommen. Sinngemäß bildet eine Datei also eine Klasse ab.

5.9.1 Paket-/Klassendiagramm

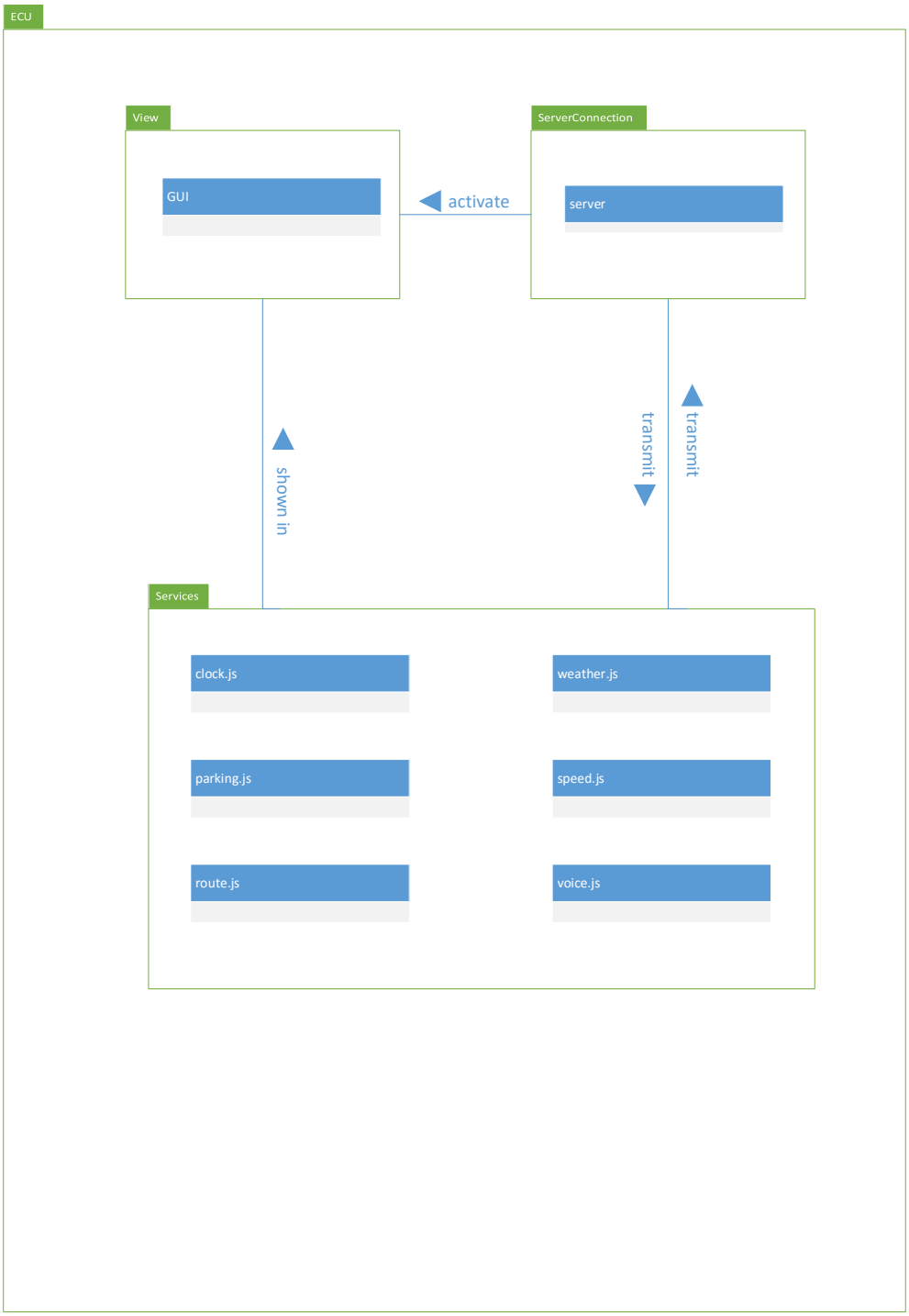

Abbildung 5.14: Klassendiagramm für Komponente **C90**

5.9.2 Erläuterung

View

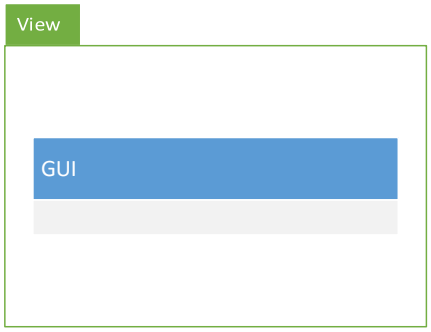

Abbildung 5.15: Klassendiagramm für View

GUI⟨$CL10$⟩

Aufgabe

Die zur View gehörende GUI bietet die Benutzeroberfläche und stellt die Informationen der Services grafisch dar. Da es sich bei der View um eine HTML-Datei handelt und die Definition eines Attributs hier nicht definiert beziehungsweise unklar ist, verwenden wir optisch klar abgrenzbare Elemente der GUI als Attribute. Hierbei kommt es zu Überschneidungen mit den Namen der Kommunikationspartner. Der Grund hierfür ist, dass die GUI die entsprechenden Rückgabewerte der Services anzeigt. Die GUI verfügt auch über eine Verbindung zum Server, da der Server die GUI beim initialen Start aufruft.

Attribute

map
clock
weather
speed
status

Operationen

Die GUI verfügt über keine Operationen, da dies die Services übernehmen.

Kommunikationspartner

clock
weather
speed
route
parking
server

Services

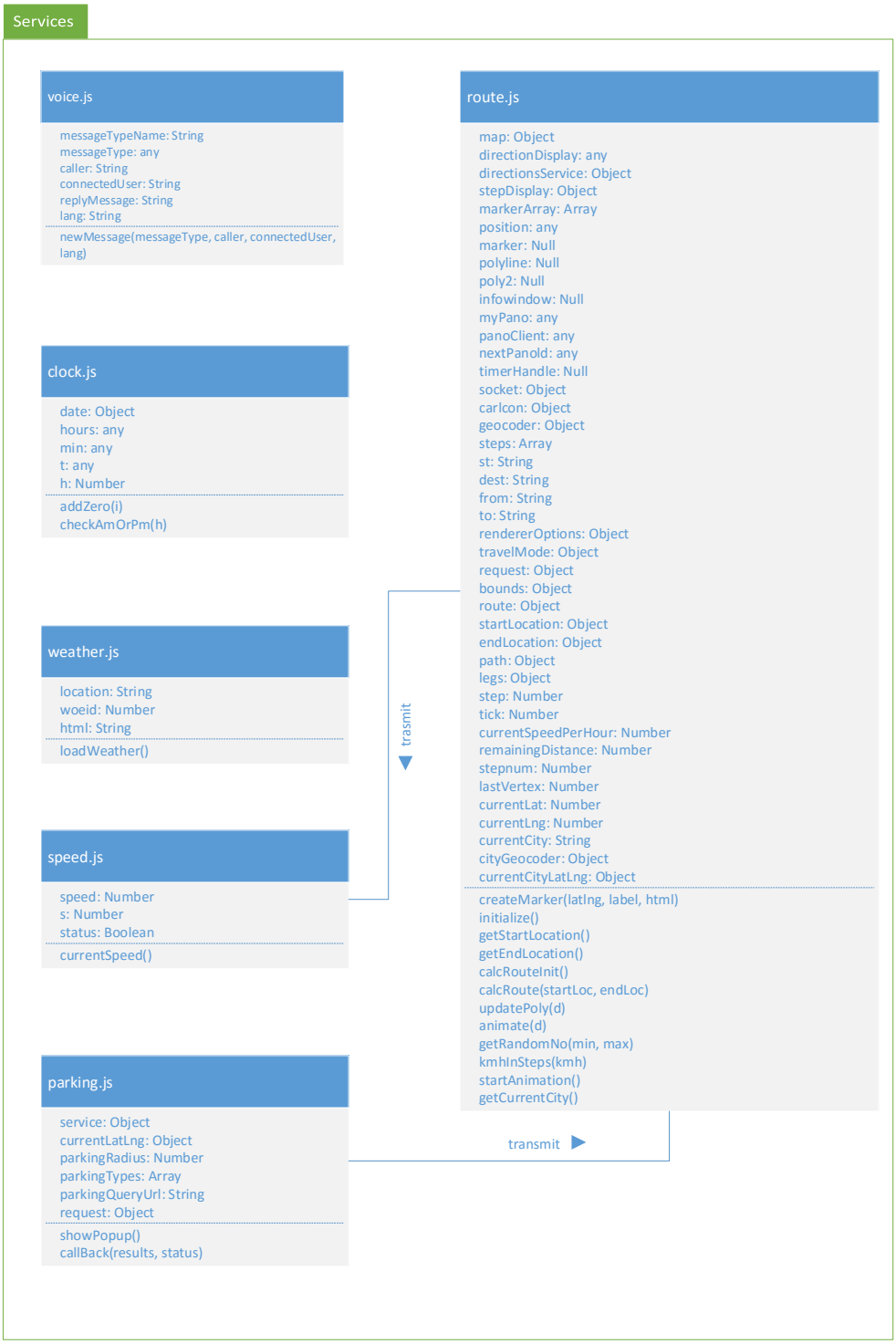

Abbildung 5.16: Klassendiagramm für Services

clock⟨$CL20$⟩

Aufgabe

Anzeigen der aktuellen Uhrzeit. Clock gehört zu den Services.

Attribute

Object date Objekt für das aktuelle Datum

Number hours Die Stunden

Number min Die Minuten

Number t Temporäre Variable

Number i Iterator

Number h Temporäre Variable zur Prüfung ob AM oder PM

Operationen

addZero(i) : Fügt bei der Minutenanzeige eine Null hinzu, falls die Minuten unter zehn sind.

checkAmOrPm(h) : Gibt die Uhrzeit als AM oder PM zurück, je nachdem wie spät es ist.

Kommunikationspartner

GUI

weather⟨$CL30$⟩

Aufgabe

Anzeigen der aktuellen Temperatur, Wetterbeschreibung, sowie des aktuellen Orts. Weather gehört zu den Services.

Attribute

String location Beschreibt den aktuellen Ort

Number woeid Where on Earth are you ID: Einzigartige ID für jede Position auf der Welt

String html Setzt sich zusammen aus den Wetterinformationen die man abrufen möchte

Operationen

loadWeather() : Ruft die geforderten Wetterdaten ab.

Kommunikationspartner

GUI

voice⟨$CL40$⟩

Aufgabe

Akustische Vermittlung von Informationen an den Benutzer, gehört zu den Services.

Attribute

String messageTypeName Variable zur Unterscheidung der Typen der Nachricht (z.B. Nachricht oder Anruf)

any messageType Der eigentliche Nachrichtentyp

String caller Anrufer

String connectedUser Name des Benutzers

String replyMessage Nachricht bei Annahme oder Ablehnung der Navigation zum nächsten Parkplatz

Operationen

newMessage(int messageType, String caller, String connectedUser, String lang) : Benachrichtigt den Benutzer bei einer eingehenden Nachricht oder eingehendem Anruf.

voiceIn() : Spracheingabe

Kommunikationspartner

Server Connection

speed⟨*CL*50⟩

Aufgabe

Erhält die Geschwindigkeit von der route.js, verarbeitet sie und gibt sie an den View und den Weberver weiter.

Attribute

Number speed Zeigt aktuelle Geschwindigkeit

Number s Zählvariable

Boolean status

Operationen

currentSpeed() : Gibt die aktuelle Geschwindigkeit aus.

Kommunikationspartner

route

GUI

Server Connection

route⟨*CL*60⟩

Aufgabe

Auswählen und Anzeigen der aktuellen Route auf Google Maps, sowie Navigation zum Zielort. Route gehört zu den Services und gibt die aktuelle Geschwindigkeit an die speed.js weiter.

Attribute

Object map Google Maps Karte

Object directionDisplay Objekt zum Anzeigen der Route

Object directionsService Objekt zum Anzeigen der Route

Object stepDisplay Objekt zum Anzeigen der Route

Array markerArray Array mit Marken zum Anzeigen von Positionen auf der Karte

Object position Aktuelle Position als LatLng-Objekt

Object marker Marke zum Anzeigen von Positionen auf der Karte

Object polyline Linie zur Darstellung der Route

Object poly2 Linie zur Darstellung der Route

Object infowindow Fenster mit Informationen auf der Karte

Object timerHandle Timer für Ladezeiten

Object socket Objekt für Serververbindung

Object carlcon Auto Icon statt des herkömmlichen Markers

Object geocoder Ermöglicht das Umwandeln von Adressen in Breiten- und Längengrade

Array steps Array für Schritte auf der Route

String st Temporäre Variable für die Startposition
String dest Temporäre Variable für das Ziel
String from Temporäre Variable für die Startposition
String to Temporäre Variable für das Ziel
Object rendererOptions Objekt für die Optionen des Renderers
Object travelMode Objekt für die Einstellung des Reisemodus
Object request Anfrage für die Route
Object bounds Bereich auf der Karte
Object route Die eigentliche Route
Object startLocation Ausgangsort
Object endLocation Zielort
Object path Pfad auf der Route
Object legs Schritte auf der Route
Number step Dauer eines Schritts auf der Karte
Number tick Wiederholung eines Schritts auf der Karte
Number currentSpeedPerHour Geschwindigkeit in km/h
Number remainingDistance Verbleibende Entfernung bis zum Ziel in Metern
Number stepnum Anzahl der Schritte
Number lastVertex Letzter Knoten auf der Route
Number currentLat Aktueller Breitengrad
Number currentLng Aktueller Längengrad
String currentCity Aktueller Ort als String
Object cityGeocoder Geocoder für den aktuellen Ort
Object currentCityLatLng Aktueller Ort als LatLng-Objekt

Operationen

createMarker(Object latlng, String label, String html) : Erstellt den Marker zum Anzeigen von Positionen.
initialize() : Initialisierung der Karte ohne Route.
getStartLocation() : Erhält den Ausgangsort aus einer Auswahlliste.
getEndLocation() : Erhält den Zielort aus einer Auswahlliste
calcRouteInit() : Initiale Funktion zur Berechnung der Route.
calcRoute(String startLoc, String endLoc) : Berechnet und zeigt die Route auf der Karte an.
updatePoly(Number d) : Aktualisiert die Linie der Route.
getCurrentCity(Object latLngObject, function cb): Gibt den aktuellen Ort als String zurück.
animate(Number d) : Animiert den Marker entlang der Route
getRandomNo(Number min, Number max) : Kreiert eine Zufallszahl zwischen den Para-

metern min und max.
kmhInSteps(Number kmh) : Umwandlung der km/h in Schritte.
startAnimation() : Startet die Animation.

Kommunikationspartner

parking
speed
GUI
Google Maps

parking⟨*CL*70⟩

Aufgabe

Option zur Ermittlung des nächstgelegenen Parkplatzes. Leitet den gefundenen Parkplatz an die route.js zur Berechnung der Route weiter. Parking gehört zu den Services.

Attribute

Object service Objekt für den Places Service von Google
Object currentLatLng Aktueller Standort als LatLng-Objekt
Number parkingRadius Radius für die Parkplatzsuche in Metern
Array parkingTypes Typen der Parkmöglichkeiten
Object request Anfrage für den nächstgelegenen Parkplatz

Operationen

showPopup(): Gibt die Option „Yes or No“ zur Parkplatzsuche.
callBack(Object results, String status) : Suche des nächstgelegenen Parkplatzes.

Kommunikationspartner

route
GUI
ServerConnection

Server Connection

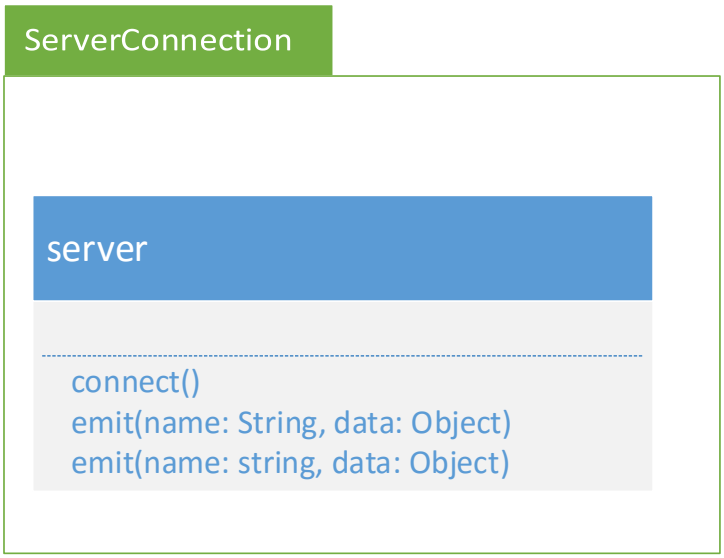

Abbildung 5.17: Klassendiagramm für Server Connection

server⟨*CL*80⟩

Aufgabe

Dient zur Verbindung zum Webserver, dieser gehört zu der ServerConnection.

Attribute

keine

Operationen

connect(): Stellt die Verbindung zum Webserver her.

emit(String name, Object data): Sendet Informationen an den Webserver.

on(String name, Object data): Empfängt eine Nachricht des Webservers.

Kommunikationspartner

speed

voice

parking

GUI

6 Datenmodell

Die ECU speichert keine Daten dauerhaft ab. Abfahrts- und Ankunftsort der Route werden für die Demonstration festgelegt. Entsprechend der Vorgabe wird die Geschwindigkeitsgrenze zum Übergang in den „Silent“-Modus auf 30 km/h festgesetzt.

Seitens der Applikation werden der Name des Benutzers, ein Benutzerbild, die Standardnachricht, eine Liste der favorisierten Kontakte und andere Einstellungen als JSON(„JavaScript Object Notation“) Objekt in einer lokalen „Shared Preferences“-Datei abgespeichert. JSON definiert ein leichtgewichtiges und einfach verständliches Datenformat. Android bietet eine „Shared Preferences API“, welche das Speichern und Abrufen von primitiven Datentypen ermöglicht.

Außerdem greift die App auf den internen Speicher des Mobiltelefons zu, um Kontakte zu den Favoriten hinzuzufügen und im Falle einer eingehenden dringlichen Nachricht den Namen des Absenders, falls dieser im Kontaktbuch eingespeichert ist, herauszufinden. SMS oder Nachrichten über den Facebook Messanger werden nicht dauerhaft gespeichert. Es wird lediglich auf diese zugegriffen beim Eingang der Nachricht um deren Dringlichkeit mit Hilfe des Klassifizierungsalgorithmus zu bestimmen.

6.1 Diagramm

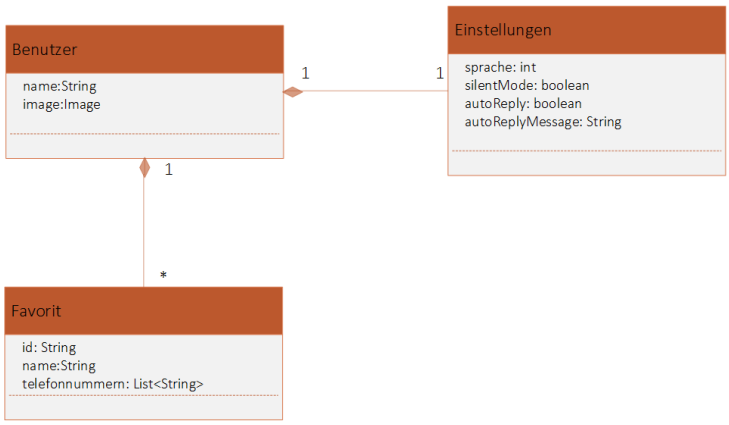

Abbildung 6.1: Klassendiagramm der App

6.2 Erläuterung

Von den Klassen Benutzer und Einstellungen existiert nur jeweils eine Instanz („Singleton“), da auf dem jeweiligen Mobiltelefon nur ein Benutzer mit einer Konfiguration von Einstellungen existiert.

Benutzer $\langle E10 \rangle$

Beziehung	**Kardinalität**	**Erwartete Datenmenge**	**Beschreibung**
Einstellungen	1	Min:1, Max:1	Jeder Benutzer besitzt genau ein Objekt der Klasse Einstellungen. In den Einstellungen werden die persönlich konfigurierbare Standardantwort und die aktuell ausgewählte Sprache gespeichert. Außerdem kann der Benutzer in der App die Deaktivierung des „Silent“-Modus auswählen, die Information über die Aktivierung dieses Zustandes wird ebenfalls in den Einstellungen abgespeichert.
Favorit	0..*	gering	Ein Benutzer besitzt die Möglichkeit in der App Kontakte als Favoriten auszuwählen, die Anrufe dieser werden immer als dringlich deklariert. Ein Benutzer kann null bis unendlich Kontakte zu seinen Favoriten hinzufügen.

Einstellung $\langle E20 \rangle$

Beziehung	**Kardinalität**	**Erwartete Datenmenge**	**Beschreibung**
Benutzer	1	Min:1, Max:1	Die Klasse Einstellungen ist existenzabhängig von der Klasse Benutzer.

Favoriten $\langle E30 \rangle$

Beziehung	**Kardinalität**	**Erwartete Datenmenge**	**Beschreibung**
Benutzer	1	Min:1, Max:1	Die Klasse Favorit ist existenzabhängig von der Klasse Benutzer, da allein der Benutzer für das Erzeugen und Löschen von Favoriten verantwortlich ist. Ein Favorit gehört außerdem immer zu genau einem Nutzer, da nur ein Benutzerprofil auf dem einzelnen Smartphone existiert und die Daten lokal auf dem Gerät gespeichert sind.

7 Konfiguration

Dieses Kapitel behandelt die Anforderungen an die für die Ausführung des Service „SocialPal“ benötigte Hard- sowie Software. Dabei wird genau auf die einzelnen Komponenten sowie auf eventuell benötigte Konfigurationsdateien eingegangen.

Der Service „SocialPal“ benötigt zum Betrieb ein Smartphone mit dem Betriebssystem Android in der Version 4.3 oder höher, einer Internetverbindung und einem Bluetooth Modul. Für den Empfang und den Versand von SMS-Nachrichten ist eine Mobilfunkverbindung auf dem Smartphone vonnöten.

Für den Demo-Betrieb zur Präsentation am Tag der jungen Software-Entwickler 2016, einer Veranstaltung im Rahmen des Software-Entwicklungspraktikums der Technischen Universität zu Braunschweig, sind vordefinierte Klassifizierer erforderlich, welche in Abhängigkeit vom aktuell gewählten Profil über die Dringlichkeit des Inhalts von eingehenden Nachrichten entscheiden. Diese Klassifizierer sind in der gelieferten Android-App enthalten (Siehe Kapitel 3.3).

Die ECU benötigt für die Simulation des fahrenden Fahrzeugs in Google Maps eine Netzwerkverbindung. Für die korrekte Darstellung aller Komponenten und Funktionen wird zudem der Google Chrome Webbrowser mit aktiviertem JavaScript benötigt.

Die Kommunikation zwischen App und ECU geschieht über einen auf der Cloud-Plattform Heroku mit Node.js erstellten Webserver mit WebSockets über Socket.io. Auf dem Server müssen eine Node.js-Version ab 4.0 und der Node Package Manager zur Installation weiterer Programme installiert sein. Zum Demo-Betrieb und während der Entwicklung wird Heroku in der Basis-Version genutzt. Die weiteren benötigten Programme Embedded JavaScript Templates, Express.js und Socket.io werden in der Datei package.json festgelegt und von Heroku mit dem Node Package Manager beim Upload der Dateien automatisch installiert. Eine spezielle Server-Konfiguration für den Kunden ist nicht erforderlich. Die ECU verbindet sich beim Start automatisch mit dem Server und hält diese Verbindung bis sie beendet wird. Die App verbindet sich nur mit dem Server, wenn das Smartphone sich im Fahrerbereich befindet. Verlässt das Smartphone diesen Bereich, wird die Verbindung wieder getrennt.
Um eine reibungslose Kommunikation mit der App und dem Benutzer zu gewährleisten, werden ein Mikrofon für die Spracheingabe und Lautsprecher für die Sprachausgabe.

Die ECU verwendet keine Datenbank, dementsprechend wird keine config-Datei benötigt.

8 Änderungen gegenüber Fachentwurf

8.1 Analyse der Produktfunktionen

In dem folgenden Abschnitt werden die Änderungen gegenüber des Fachentenwurfs des 2. Kapitels beschrieben.

8.1.1 Allgemeine Änderungen

In den folgenden Unterkapitel werden die Änderungen im Hinblick auf die Produktfunktionen und deren Analyse bezüglich des Fachentwurfs erläutert.

Die Sequenzdiagramme wurden syntaktisch so verändert, dass die abgebildeten Funktionen Java-Methoden entsprechen.

Des Weiteren erfolgt die Verbindung zwischen der ECU und der App aufgrund technischer Restriktionen nicht mehr mittels Bluetooth, sondern über einen Webserver. Die ECU verbindet sich beim Starten mit dem Server und streamt Änderungen an der Fahrtgeschwindigkeit, wenn die Grenze von 30 km/h erreicht wird. Die App verbindet sich erst mit dem Server, wenn es den Fahrerbereich erreicht und die Anwesenheit des Beacons registriert hat. Geht dann eine dringende Nachricht ein, wird eine Mitteilung über den Server an die ECU gesendet. Sobald das Smartphone den Bereich wieder verlässt, trennt die App die Verbindung zum Server. Dadurch, dass nur das Smartphone im Fahrerbereich mit dem Server verbunden ist, wird verhindert, dass die ECU benachrichtigt wird, wenn ein anderes Smartphone eine Nachricht erhält.

Die Auswirkungen auf die Umsetzung einiger Funktionalitäten, werden im Folgenden weiter beschrieben.

8.1.2 Analyse von Funktionalität F20: Bluetooth Kommunikation

Da die Implementierung der Bluetooth Kommunikation nicht möglich war, wurde die Kommunikation zwischen App und ECU, wie beschrieben, auf einen Webserver umgestellt. Die Funktion wurde daher umbenannt und im Sequenzdiagramm wurde der Server zwischen den beiden Komponenten eingefügt.

8.1.3 Analyse von Funktionalität F40: „Default“- und „Urgent“-Modus

Aufgrund der Änderungen an der Kommunikation wurde der Webserver zwischen die Komponenten geschaltet. Trotz der Änderungen am technischen Kommunikationsweg hat sich die inhaltliche Funktion der Benachrichtigungen nicht geändert.

8.1.4 Analyse von Funktionalität F60: „Silent“-Modus

Wie beschrieben erfolgt die Kommunikation zwischen ECU und App über einen Webserver. Daher wurde auch in der Abbildung 2.6 eine gleichnamige Komponente hinzugefügt.

8.1.5 Analyse von Funktionalität F110: Automatischer App Start

Die App startet nun automatisch, sobald eine Verbindung zum Beacon hergestellt wurde. Ein vorheriger Start der ECU ist nicht erforderlich.

8.2 Erfüllung der Kriterien

Das Kriterium **RS5** wurde entfernt, da es inhaltlich das Selbe beschreibt wie das Kriterium **RM17**.

Aufgrund der Verdeutlichung des Kriteriums **RM12** (Die App muss Nachrichten und Telefonanrufe abfangen können) wurden drei Weitere erstellt. Hierbei handelt es sich um **RM18**, **RM10** und **RM20**. Auf diese wurde in Kapitel 9 näher eingegangen.

8.3 Konfiguration

Bei der Weiterentwicklung wurde festgestellt, dass aufgrund technischer Restriktionen, eine reibungslose Kommunikation zwischen der App und ECU über Bluetooth nicht gewährleistet werden kann. Aus diesem Grund kommunizieren die beiden Komponenten nun über einen Webserver und benötigen dafür eine Netzwerkverbindung. Ein Bluetooth Adapter ist auf Seiten der ECU nicht mehr erforderlich.

9 Erfüllung der Kriterien

In diesem Kapitel werden die Muss-, Soll- und Kannkriterien aus dem Pflichtenheft aufgelistet und hinsichtlich ihrer Erfüllung durch das Produkt „SocialPal“ beschrieben. Hierbei werden auf die im dritten Kapitel beschriebenen Komponenten eingegangen.

9.1 Musskriterien

Die folgenden Kriterien sind unabdingbar und müssen durch das Softwareprodukt in jedem Fall erfüllt werden, damit es genutzt werden kann:

RM1 *Die App muss dem Benutzer das Auswählen von für ihn favorisierten Kontakten anbieten.*
Dieses Kriterium wird erfüllt. Der Nutzer kann für ihn wichtige Kontakte als Favoriten auswählen dies wird über die Komponente **C20** verwaltet. Dabei wird auf die lokale Kontaktdatenbank des Smartphhones zugegriffen, der MobileDB, siehe Komponente **C100**.

RM2 *Die ECU und die App müssen via Bluetooth Low Energy kommunizieren können.*
Dieses Kriterium wird nicht erfüllt. Stattdessen findet die Kommunikation zwischen ECU und App über einen Webserver mittels TCP/IP Protokoll statt.

RM3 *Die ECU muss den Start und Zielort anzeigen, sowie die dazugehörige Strecke.*
Dieses Kriterium wird erfüllt. Die Klasse Route ist für das Auswählen und Anzeigen der aktuellen Route zuständig, diese wird in der Methode calcRoute(startLoc, endLoc) berechnet. Außerdem findet vom Startort eine Navigation zum Zielort statt. Dieser Service gehört zur Komponente **C90**.

RM4 *Die ECU muss eine Bewegung des Fahrzeugs auf der Karte darstellen.*
Dieses Kriterium wird erfüllt. Die Methode animate(d) in der Klasse Route animiert die Bewegung des Fahrzeugs in der Form eines Autos auf der Karte, siehe Komponente **C90**.

RM5 *Die ECU muss einen „Default“- und „Urgent“-Modus besitzen.*
Dieses Kriterium wird erfüllt. Die ECU befindet sich von Beginn an im „Default“-Modus. Geht eine dringende Nachricht ein, zeigt die Methode showPopup() eine Anfrage, ob in den „Urgent“-Modus gewechselt werden soll. Bestätigt der User, wird der „Urgent“-Modus aktivert. Siehe Komponente **C90**.

RM6 *Die ECU muss die Geschwindigkeit anzeigen können und ab einer Geschwindigkeit von 30km/h diese Information der App zur Initialisierung des „Silent“-Modus mitteilen.*
Dieses Kriterium wird erfüllt. Die Methode currentSpeed() zeigt die aktuelle Geschwindigkeit, bei einer Geschwindigkeit von 30 km/h wird über den Websocket, mit der Methode socket.emit('silentOn', 'silent on'), die App benachrichtigt. Siehe Komponente **C90**.

RM7 *Die App muss die Anwesenheit des Beacon registrieren und mit der Aktivierung des „Driving“-Modus darauf reagieren.*
Dieses Kriterium wird erfüllt. Der Beacon (Komponente **C80**) sendet ein ständiges Bluetooth Low Energy Signal in dem Bereich des Fahrersitz aus. Dieses Signal wird sobald ein Smartphone sich in diesem Bereich befindet aufgenommen und die App startet automatisch und geht zeitgleich bei bestehender Internetverbindung in den „Driving“-Modus.

RM8 *Die App muss sich automatisch beim simulierten Einschalten des Motors starten.*
Dieses Kriterium wird nicht erfüllt. Die App startet sobald sie sich in der Nähe des Beacon befindet.

RM9 *Die ECU muss im Falle einer für den Fahrer dringenden Nachricht die nächstgelegene Parkmöglichkeit finden und auf der Karte darstellen.*
Dieses Kriterium wird erfüllt. Durch die Implementierung der Komponente **C90**, kann der Fahrer sich zu einer Parkmöglichkeit navigieren lassen. Mithilfe der Methode calcRoute() wird nach einer Parkmöglichkeit in der Umgebung gesucht und auf der Karte dargestellt.

RM10 *Die ECU muss den Benutzer über Mitteilungen per Sprachausgabe informieren können.*
Dieses Kriterium wird erfüllt. In der Komponente **C90** ermöglicht die Schnittstelle **I150** das Umwandeln von einem Text in eine Sprachausgabe, sowie die Wiedergabe über ein Ausgabegerät. Dafür wird die Methode speak() aufgerufen.

RM11 *Die App muss ab Android 4.3 Jelly Bean (API Level 18) kompatibel sein.*
Dieses Kriterium wurde implementiert. Die App ist somit von jedem Android Smartphone ab Android 4.3 Jelly Bean (API Level 18) nutzbar.

RM12 *Die App muss Nachrichten und Telefonanrufe abfangen können.*
Dieses Kriterium ist erfüllt. Die App Empfängt eingehende SMS-Nachrichten, Anrufe und Facebook-Messenger-Nachrichten. Dieser Service wird in der Komponente **C50** implementiert.

RM13 *Die App muss die Inhalte eintreffender Nachrichten sowie die Namen von Anrufern analysieren und abhängig von ihrer Dringlichkeit für den Fahrer darauf reagieren.*
Dieses Kriterium ist erfüllt. Die eingetroffenen Nachrichten werden an die Komponente **C60** übergeben. Der Classifier prüft durch Aufruf der Methode classifyMessage() die Inhalte auf Dringlichkeit und gibt eine Rückmeldung an die Komponente **C50** zurück. Durch die Implementierung der Komponente **C20** können Kontakte den Favoriten hinzugefügt werden. Sobald ein Anruf von einer Person kommt, der in den Favoriten vorhanden ist, wird der Fahrer darüber informiert.

RM14 *Die App muss in der Lage sein mit einer zuvor definierten Standardantwort auf eingehende Nachrichten reagieren zu können.*
Dieses Kriterium ist erfüllt. Sobald eine Nachricht eingetroffen ist wird eine Autoreply-Message gesendet. Dieser Service wird in der Komponente **C50** implementiert.

RM15 *Die ECU muss sich automatisch beim simulierten Einschalten des Motors starten.*
Dieses Kriterium ist erfüllt. Es wird ein simulierter „startEngine“ Button für den Motor angezeigt. Nach Betätigen dieses Buttons wird die ECU automatisch gestartet.

RM16 *Die ECU muss vollständig im Google Chrome Browser und unter einem Desktop-Betriebssystem laufen (Linux/OS X).*
Dieses Kriterium ist erfüllt. Es funktioniert sogar auf weiteren Betriebssystemen wie Windows.

RM17 *Der Benutzer muss über Buttons in der ECU auswählen können, ob er zur nächstgelegenen Parkmöglichkeit geleitet werden möchte oder nicht.*
Dieses Kriterium ist erfüllt. Nachdem eine dringliche Nachricht eingetroffen ist, wird der Benutzer über die Komponente **C90** durch den Aufruf der Methode showPopup() gefragt ob man zu einer Parkmöglichkeit navigiert werden will. Der Benutzer kann dieses ablehnen oder akzeptieren.

RM18 *Die Dringlichkeit von Nachrichten muss ermittelt werden.*
Das Kriterium ist erfüllt. Die Receivers Komponente **C50** fängt über die Klasse SmsReceiver SMS und über die Klasse FacebookMessangerListener die Nachrichten im Silent-Modus ab und übergibt diese zur Ermittlung der Dringlichkeit an die Classifier Komponente **C60**.

RM19 *Anrufe müssen im Silent-Modus abgelehnt werden.*
Das Kriterium ist erfüllt. Die Klasse CallReceiver der Komponente **C50** lehnt Anrufe, sofern sich die Applikation im Silent-Modus befindet.

RM20 *Im Falle der Aktivierung des Features Standardnachricht muss im Silent-Modus dem Anrufer oder Absender einer SMS eine Nachricht gesendet werden.*
Das Kriterium ist erfüllt. Die Klasse SmsReceiver der Komponente **C50** besitzt eine Methode sendAutoReply(String phoneNumber, String message), welche eine Nachricht an die übergebene Telefonnummer sendet.

9.2 Sollkriterien

Die Erfüllung folgender Kriterien sind für die Lauffähigkeit des Produkts nicht zwingend erforderlich, für die Erreichung der Projektziele aber erfüllt werden sollten:

RS1 *Die ECU soll das aktuelle Wetter anzeigen können.*
Dieses Kriterium wird erfüllt. Mittels der Methode loadWeather() wird über den Standort, welche von der Klasse route übermittelt wird, die Temperatur und die Wetterlage abgerufen. Dieser Service ist in der Komponente **C90** implementiert.

RS2 *Die ECU soll die aktuelle Zeit anzeigen können.*
Dieses Kriterium wird erfüllt. Der in der Komponente **C90** implementierte Service Clock ist für das anzeigen der aktuelle Uhrzeit zuständig.

RS3 *Die ECU und die App sollen intuitiv bedienbar sein.*
Dieses Kriterium wird erfüllt. Das Design der Oberfläche der App (siehe **C10**) und der ECU (siehe **C90**) wurden leicht verständlich und übersichtlich gehalten. Damit der Nutzer ohne vorheriger Einweisung das Produkt bedienen kann.

RS4 *Die App soll im Hintergrund laufen und Prozesse wie die Musikwiedergabe nicht behindern.*
Dieses Kriterium wird erfüllt. Die App greift auf keine Funktionen von Apps die im Hintergrund laufen ein, lediglich visuelle und akustische Benachrichtigungen werden ausgeblendet.

RS5 *Die ECU soll nicht zur nächsten Parkmöglichkeit navigieren, wenn das ursprüngliche Ziel näher ist.*
Dieses Kriterium wird erfüllt. Über die Methode callBack(results, status) in der Klasse Parking wird der nächstgelegene Parkplatz gesucht. Sollte der Fall eintreffen, dass diese Route weiter weg ist als das ursprüngliche Ziel, so wird der Fahrer weiterhin zum Ziel navigiert.

9.3 Kannkriterien

Die Erfüllung folgender Kriterien für das abzugebende Softwareprodukt wird angestrebt, falls noch genügend Kapazität vorhanden sind:

RC1 *Die App unterstützt mehrere Sprachen.*
Dieses Kriterium wird erfüllt. Die App unterstützt die englische und die deutsche Sprache. In den Einstellungen kann der Benutzer eine Auswahl zwischen den beiden Sprachen treffen, welche gespeichert werden, siehe **C30**.

RC2 *Die ECU unterstützt mehrere Sprachen.*
Dieses Kriterium wird erfüllt. Die ECU unterstützt die englische und die deutsche Sprache. Die Sprachausgabe der ECU funktioniert in beiden Sprachen, siehe **C90**.

RC3 *Die App wandelt Sprachnachrichten auf dem Anrufbeantworter in Text um und analysiert deren Inhalt auf Dringlichkeit für den Fahrer.*
Dieses Kriterium wird voraussichtlich nicht erfüllt werden.

RC4 *Die ECU kann auf Spracheingaben reagieren.*
Dieses Kriterium wird von uns voraussichtlich nicht erfüllt werden, da der Sprecher aufgrund seiner Aussprache und den Geräuschen in der Umgebung nicht gut verstanden wird.

10 Glossar

Android: Betriebssystem von Google, welches meistens auf Smartphones und Tablets eingesetzt wird.

Beacon: Sender oder Empfänger, die Bluetooth Low Energy (BLE) benutzen.

Bluetooth Low Energie (kurz BLE): Funktechnik, mit der sich Geräte in einer Umgebung von etwa 10 Meter vernetzen lassen. Im Vergleich zum „klassischen“ Bluetooth hat BLE einen deutlich geringeren Stromverbrauch.

„Default“-Modus: Ist ein Modus der ECU, der den Start- und Endpunkt der aktuellen Route, die graphische Darstellung der aktuellen Position und die aktuelle Geschwindigkeit beinhaltet.

Dringlichkeit: Ein Maß dafür, ob auf eine Mitteilung eine umgehende Nutzerinteraktion erfolgen muss oder ob dies auch zu einem späteren Zeitpunkt geschehen kann. Diese ist individuell für jeden einzelnen Nutzer definiert.

„Driving“-Modus: Ist ein Modus der App, der unterhalb einer bestimmten Geschwindigkeitsgrenze oder außerhalb des Radius zum Fahrer aktiviert wird und alle Benachrichtigungen anzeigt.

ECU: Steuergeräte (engl. electronic control unit) sind elektronische Module, die überwiegend an Orten eingebaut werden, an denen etwas gesteuert oder geregelt werden muss.

Express.js: Serverseitiges Webframework für Node.js, erweitert Node.js um einige vereinfachende Entwicklungs-Tools.

Heroku: Cloud-Plattform zur Entwicklung und Ausführung von Applikationen. Unterstützt unter anderem Node.js.

JSON: JSON („JavaScript Object Notation“) definiert ein leichtgewichtiges und einfach verständliches Datenformat.

Klassifizierung: Beschreibt das Zusammenfassen von bestimmten Objekten zu Klassen.

MVC-Pattern: MVC (Model View Controller) beschreibt Softwarearchitekturmuster zur Erstellung von Benutzeroberflächen.

Node.js: Serverseitige JavaScript-Plattform zur Entwicklung von Netzwerkanwendungen. Verwendet ereignisgesteuerte, asynchrone und nicht blockierende Architektur und ist daher ideal für datenintensive Echtzeitanwendungen.

Node Package Manager: Manager zur Installation von Packages zur Erweiterung von Node.js.

„Silent"-Modus: Ist ein Modus der App, der ab einer bestimmten Geschwindigkeitsgrenze und innerhalb des Radius zum Fahrer aktiviert wird und der Benutzer nur noch über für ihn dringliche Benachrichtigungen mit Hilfe des „Urgent"-Modus informiert wird.

Socket.io: Bibliothek für Node.js, die Echtzeit-Kommunikation zwischen Node.js und Webbrowsern über Websockets ermöglicht.

„Urgent"-Modus: Ist ein Modus der ECU, der mittels einer Sprachausgabe über eine für den Benutzer dringende Benachrichtigung informiert, um ihn anschließend zur nächsten Parkmöglichkeit zu navigieren.

Websocket: TCP-basiertes Netzwerkprotokoll, das bidirektionale Verbindung zwischen einer Webanwendung und einem Webserver herstellt.