Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Vadim Holstein

Entwurf und exemplarische Umsetzung eines Protokolls zur generischen Kommunikation mobiler Anwendungen mit RC-Modellen

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Vadim Holstein

Entwurf und exemplarische Umsetzung eines Protokolls zur generischen Kommunikation mobiler Anwendungen mit RC-Modellen

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Marc Hensel
Zweitgutachter : Prof. Dr.-Ing. Sebastian Rohjans

Abgegeben am 16. August 2018

Vadim Holstein

Thema der Bachelorthesis

Entwurf und exemplarische Umsetzung eines Protokolls zur generischen Kommunikation mobiler Anwendungen mit RC-Modellen

Stichworte

RC-Modell, Android, Activity, CoAP, Erneute Übertragung, Kommunikationstechnologie, Bluetooth Low Energy, WiFi

Kurzzusammenfassung

Gegenstand dieser Thesis ist die Entwicklung eines Kommunikationsprotokolls zwischen mobilen Anwendungen und RC-Modellen. In dieser Thesis wird sowohl das Kommunikationsprotokoll, als auch die zugehörige App entwickelt, die zur Steuerung der RC-Modelle dient. Dabei werden Anforderungen ermittelt, das Design des gesamten Systems erstellt und anschließend implementiert. Ein zentraler Punkt ist das Constraint Application Protocol, kurz CoAP, welches als Basis für das Design des Protokolls dient.

Vadim Holstein

Title of the paper

Design and exemplary implementation of a protocol for generic communication of mobile applications with RC models

Keywords

RC-Model, Android, Activity, CoAP, retransmission, communication technology, Bluetooth Low Energy, WiFi

Abstract

The subject of this thesis is the development of a communication protocol between mobile applications and RC models. In this thesis, both the communication protocol and the associated app are developed, which serves to control the RC models. Requirements are determined, the design of the entire system is created and then implemented. A central point is the Constraint Application Protocol, CoAP in short, which serves as the basis for the design of the protocol.

Danksagung

An dieser Stelle möchte ich meinen tiefsten Dank für meinen Prüfer Prof. Dr.-Ing. Marc Hensel aussprechen, der mir diese Bachelorarbeit ermöglicht und mir eine tolle Unterstützung und sehr gute Betreuung geboten hat.

Mein besonderer Dank gilt auch meinem Zweitprüfer Prof. Dr.-Ing. Sebastian Rohjans, der sich trotz bevorstehendem Auslandsaufenthalt kurzfristig bereit erklärt hat die Zweitprüfung zu übernehmen. Vielen Dank!

Abkürzungsverzeichnis

APK Android package

ART Android Runtime

ATT Attribute Protocol

BLE Bluetooth Low Energy

GATT Generic Attribute Profile

CoAP Constrained Application Protocol

DTLS Datagram Transport Layer Security

GATT Generic Attribute Profile

IETF Internet Engineering Task Force

IoT Internet of Things

PPMP Production Performance Management Protocol

MQTT Message Queue Telemetry Transport

REST Representational State Transfer

TLS Transport Layer Security

UUID Universally Unique Identifier

XML Extensible Markup Language

Inhaltsverzeichnis

1 Einführung

In diesem Kapitel werden der Aufbau der Thesis, Motivation, die Problemstellung und die Aufgabenstellung und Zielsetzung besprochen.

1.1 Aufbau der Thesis

In der **Einführung** werden die Motivation, Problem- und Aufgabenstellung und die Zielsetzung erläutert.

Das Kapitel **Theorie** behandelt die notwendigen Grundlagen, die zum Verstehen der Arbeit notwendig sind.

Im Kapitel **Anforderungen** werden die Anforderungen an das Kommunikationsprotokoll, das Gesamtsystem und die einzelnen Komponenten festgelegt.

Das Kapitel **Konzept und Design** beinhaltet den Entwurf des Kommunikationsprotokolls und der Android App. Der Entwurf wird mithilfe von MockUps und UML Diagrammen dargestellt.

In dem Kapitel **Implementierung** wird die Umsetzung des Designs in der Android App und dem Code der RC-Automodelle behandelt.

Im Kapitel **Test** wird dargestellt, wie das implementierte System auf eine erfolgreiche Umsetzung getestet wurde.

Das Kapitel **Zusammenfassung** gibt in Kurzform wieder, was getan und welche Ziele erreicht wurden. Außerdem bietet das Kapitel einen Ausblick an, was noch unmittelbar oder in Zukunft umgesetzt werden könnte.

Der komplette Anhang befindet sich auf einer CD und ist jeweils bei Prüfer Prof. Dr.Ing. Marc Hensel und bei Prüfer Prof. Dr.Ing. Sebastian Rohjans einzusehen.

1.2 Motivation

Ferngesteuerte Modelle, auch als RC-Modelle (engl. *Remote Controlled* oder *Radio Controlled*) bezeichnet, finden in der heutigen Zeit immer mehr Beliebtheit. Der Einsatzbereich von ferngesteuerten Modellen reicht vom Hobbybau, über Projekte an Hochschulen und Universitäten bis hin zum Einsatz in der Industrie und dem Militär. So können verschiedene Modelltypen, vom Auto bis hin zur Drohne, mittlerweile in vielen Geschäften bzw. in Online-Shops gekauft werden. Viele dieser Modelle bieten Programmierschnittstellen an, über welche das Verhalten des Modells und die Kommunikation mit dem zu steuernden Gerät selbst entwickelt werden kann. Dadurch bieten sich etliche Möglichkeiten zur Steuerung des Modells an. Eine beliebte Variante ist die Steuerung über ein Mobile Gerät, wie z.B. ein Smartphone oder Tablet.

1.3 Problemstellung

An der *Hochschule für Angewandte Wissenschaften Hamburg* werden regelmässig Bachelorprojekte durchgeführt, die sich mit der Steuerung von RC-Modellen über Mobile-Geräte beschäftigen. Dabei arbeiten diese Projekte relativ autonom. Die Kommunikation wird über verschiedene Technologien ausgeführt, wie z.B. Bluetooth, Bluetooth Low Energy oder WiFi, die sich von Projekt zu Projekt unterscheiden können. Im Rahmen des Projektes wird festgelegt, welche Daten zur Steuerung übertragen werden, sprich es wird ein Kommunikationsprotokoll festgelegt. Dazu wird eine passende App entwickelt über die das RC-Modell mit dem festgelegten Protokoll gesteuert wird.

Hieraus ergibt sich das Problem, dass für jedes Modell eine eigene Implementierung der Kommunikation entsteht und somit eine eigene App notwendig ist. Soll mehr als ein Modell um eine neue, gleiche Funktion erweitert werden, so muss wegen unterschiedlicher Kommunikationsprotokolle nicht nur die jeweilige App, sondern auch das Kommunikationsprotokoll erweitert oder gar angepasst werden. Dies führt zu einem Mehraufwand, der durch eine generische Kommunikation verhindert werden könnte.

1.4 Aufgabenstellung und Zielsetzung

Im Rahmen dieser Bachelorarbeit soll Folgendes entwickelt werden:

1. Ein leichtgewichtiges Kommunikationsprotokoll, welches zur generischen Kommunikation mit RC-Modellen verschiedenen Typs (zu Land, zu Wasser und in der Luft) geeignet ist.
2. Eine Android App, die über das entwickelte Protokoll mit RC-Modellen kommunizieren kann.
3. Den Code von ein bis zwei vorhandenen RC-Automodellen anpassen, so dass diese über das entwickelte Protokoll mit der steuernden Android App kommunizieren können.

Sowohl das Protokoll als auch die App sollen dabei leicht erweiterbar sein. Die RC-Automodelle sollen von der gleichen App gesteuert werden können.

2 Theorie

Dieses Kapitel behandelt die notwendigen Grundlagen, die für das Verständnis der Thesis essentiell sind. Alle Quellen und Bildquellen wurden zuletzt am 15.08.2018 aufgerufen.

2.1 Bluetooth Low Energy

Bluetooth Low Energy (BLE) ist eine Kommunikationstechnologie, die auf der Spezifikation von Generic Attribute Profile (GATT) basiert. GATT ist ein Profil zum Versenden von kleinen Datenpaketen über einen BLE-Link, die als *Attributes* bezeichnet werden. GATT baut auf das Attribute Protocol (ATT) auf. Daher wird diese Spezifikation oft auch als GATT/ATT bezeichnet (siehe Abb. 2.1). [11, Kap. 14]

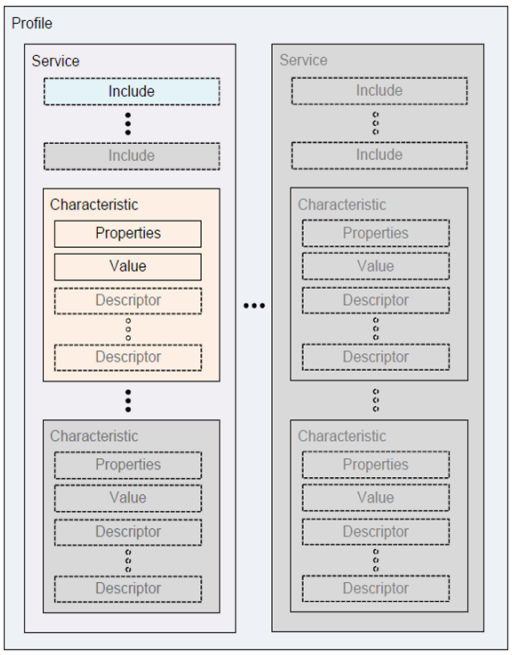

Abbildung 2.1: GATT/ATT
Bildquelle: https://www.bluetooth.com/specifications/gatt/generic-attributes-overview

Jedes Attribute Protocol (ATT) hat 1 bis n *Services*, welche durch ein Universally Unique Identifier (UUID), eine 128 Bit lange Nummer, eindeutig identifiziert werden. Jeder Service hat 1 bis n *Characteristics*, welche einen einzelnen Wert darstellen. Dieser Wert kann gelesen und/oder überschrieben werden, je nach dem welche Rechte bzw. *Properties* dieser Characteristic zugeschrieben wurden. Die Property *Notify* bietet die Möglichkeit, den *Subscriber* zu benachrichtigen, wenn sich der Wert der zugehörigen Characteristic geändert hat. Eine Characteristic hat 0 bis n *Descriptors*, welche eine Beschreibung der Characteristic darstellen. So kann es einen Service „Heart Rate Monitor“ mit einer Characteristic „Heart Rate Measurement“ geben. [3]

2.2 Android

Android steht sowohl für die Software-Plattform für mobile Geräte, wie z.B. Smartphones und Tablets, als auch für das zugehörige Betriebssystem [1]. Die Basis für Android ist der Linux-Kernel. Jede Android-Version wird über eine Versionsnummer und einen *Codename* definiert[1] [10, S. 11].

Alle weiteren Informationen über Android sind auf auf [2] zu finden.

Android Apps können mit Java, Kotlin oder C++ entwickelt werden. Beim Entwickeln einer Android-App muss eine minimale SDK Version angegeben werden. Dieses steht für die minimale Android-Version auf welcher die App noch ausführbar ist. Diese SDK Version wird mit dem *API-Level* angegeben. Jedes API-Level steht für eine bestimmte Android Version:

[1] Aktuelle Version: 8.1.0 (Oreo), Stand: Juli 2018

Platform Version	API Level
Android 8.1	27
Android 8.0	26
Android 7.1.1 Android 7.1	25
Android 7.0	24
Android 6.0	23
Android 5.1	22
Android 5.0	21
Android 4.4W	20
Android 4.4	19
Android 4.3	18

Tabelle 2.1: Android API Level
Quelle: https://developer.android.com/guide/topics/manifest/uses-sdk-element

Das Android SDK erstellt beim Kompilieren ein Android package (APK) aus dem Code, den Ressourcen und anderen Dateien. Die durch den Compiler erstellte Datei hat die Endung *.apk* und beinhaltet dem gesamten Inhalt der App, der für die Installation auf einem Android Geräte notwendig ist.

Jede Android App wird in einer eigenen *security sandbox*, sprich einer eigenen virtuellen Maschine, ausgeführt und läuft in einem eigenen Linux Prozess. Der Prozess wird gestartet wenn eine Komponente der App ausgeführt werden muss und wieder beendet, sobald keine der Komponenten benötigt werden. Ein Prozess kann außerdem durch das Android-Betriebssystem beendet werden, sollten Ressourcen für andere Prozesse benötigt werden, die eine höhere Priorität haben. Das Android Betriebssystem befolgt das *principle of least privilege*. Das bedeutet, dass jede App standardmäßig nur Zugriff auf die Komponenten hat, die für die Ausführung benötigt werden.

Eine Komponente (Component) stellt einen separaten Block einer App dar. Jede Komponente fungiert gleichzeitig als Einstiegspunkt, durch welche das Betriebssystem oder der Nutzer auf die App zugreifen kann. In Android existieren vier verschiedene App-Komponenten:

- Activities
- Services
- Broadcast receivers
- Content providers

Jede dieser Komponenten hat einen eigenen Lebenszyklus. Von diesen Komponenten sind für diese Thesis nur die *Activities* und *Services* relevant.

2.2.1 Android Architektur

Die Android Plattform besteht aus mehreren Komponenten (siehe Abb. 2.2). So sind Kern-Applikationen, wie z.B. Kontakte, Telefon, Browser enhalten. Das *Application Framework* bietet Schnittstellen an, die auch schon von den Kern-Applikationen genutzt werden. Durch diese Schnittstellen wird definiert wie sich die App verhält und wie sie aussieht. Hinter den APIs stehen die C und C++ Libraries, auf die über die APIs zugegriffen wird. Der Linux-Kernel stellt in Android die Treiber und Kerndienste wie Sicherheits- und Speicherverwaltung dar.[10, S. 3]

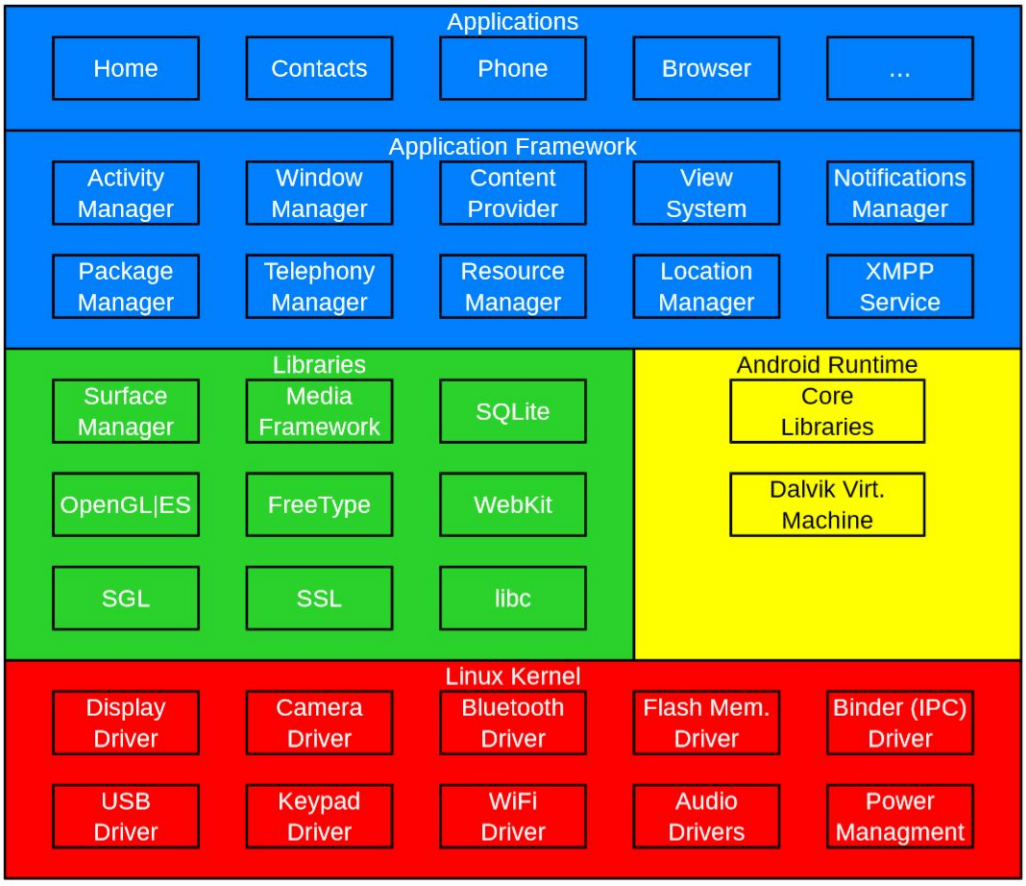

Abbildung 2.2: Android Architektur
Quelle: https://de.wikipedia.org/wiki/Android_(Betriebssystem)

2.2.2 Activities

Eine *Activity* stellt in Android eine Bildschirmsete dar, die dem Nutzer bestimmte Interaktion bietet und bestimmte Daten darstellt. In der Android Programmierung gibt es keinen festgelegten Startpunkt, durch welchen die App immer gestartet wird, wie z.B. durch eine **main()** Methode. Zwar wird bei der Entwicklung einer App angegeben, welche Activity beim Starten der App zuerst aufgerufen wird, prinzipiell kann aber jede Activity als Einstiegspunkt für die App dienen. So könnte z.B. eine App zum Lesen und Versenden von Emails zwei Activities haben. Die erste stellt ein Fenster dar, in dem alle Emails gelistet und gelesen werden können, während die zweite ein Fenster zum Verfassen und Versenden von Emails bietet. Zwar könnte die erstgenannte Activity beim Starten der App aufgerufen werden, gleichzeitig könnte aber eine Social Media App auch die Activity zum Versenden von Emails aufrufen. Im Code werden Activities durch das Erben der Klasse **Activity** angegeben.

Acitivites folgen einen definiertem Lebenszyklus, genannt *Activity Lifecycle* (siehe Abb. 2.3). Dabei nimmt eine Activity verschiedene *States* ein. So nimmt z.B. eine Activity bei ihrer Erstellung den Status *Created* an. Für jeden Status (State) kann ein Callback-Methode überschrieben werden, die beim Wechsel in diesen Status aufgerufen wird. Dabei muss **super()** innerhalb des überschriebenen Callbacks aufgerufen werden. Die Callback-Methode **onCreate()** ist der einzige Callback, den eine Activity zwingend überschreiben muss. Jedoch kann das Überschreiben anderer Callbacks wichtig für die Performance und die Robustheit der App sein, um zu verhindern dass...

- ...die App abstürzt wenn eine andere Activity in den Vordergrund rückt, z.B. durch einen eingehenden Telefonanruf
- ...nicht mehr benötigte Ressourcen nicht freigegeben werden, wenn die Activity gestoppt wird
- ...Abstürze oder Datenverluste zu verhindern wenn das Gerät vom Hochformat in den Querformat wechselt und andersrum

Der letzte Punkt ist dadurch bedingt, da eine Activity beim Wechsel des Formats zerstört und neu erstellt wird.

Eine Activity kann mehrere States durchlaufen und somit auch die entsprechenden Callbacks nacheinander aufrufen. So werden beim Erstellen der Activity die Callbacks **onCreate()**, **onStart()** und **onResume()** ausgeführt.

Jede Activity hat eine oder mehrere *Layout* Ressourcen Dateien, die angeben wie die Activity aussieht. Dabei wird vom Android System die für die Auflösung und das Format

passende Layout Ressource gewählt. Layout Ressourcen Dateien werden in Extensible Markup Language (XML) geschrieben.

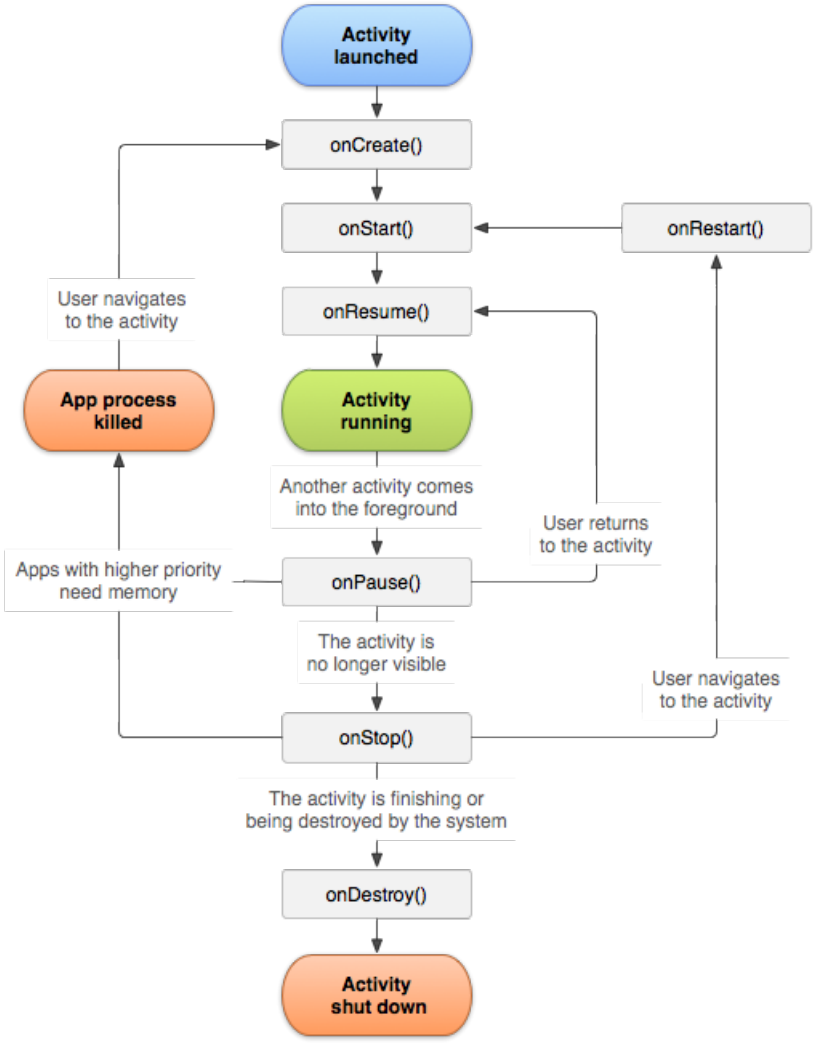

Abbildung 2.3: Activity Lifecycle
Bildquelle: https://developer.android.com/guide/components/activities/activity-lifecycle

2.2.3 Intents

Ein *Intent* ist ein Objekt, durch welches aus einer Activity andere Activities oder Services gestartet werden können. Dabei wird das erstellte Intent dem Android System übergeben (siehe Abb. 2.4). Das System erstellt die passende Activity und startet eventuell die zugehörige App, wenn die Activity nicht in derselben App existiert aus der sie gestartet wird. Es wird

hierbei zwischen *expliziten* und *impliziten* Intents unterschieden. Ein explizites Intent startet eine Activity oder einen Service, die in derselben App existieren. Dabei wird die Klasse der gewünschten Activity oder des gewünschtes Services als Parameter beim Erstellen des Intents übergeben. Bei Impliziten Intents wird keine spezifische Activity angegeben, sondern eine *Action*. Diese kann z.B. das Versenden einer Email sein. Das Android System zeigt dem Nutzer anschließend alle Apps an, die diese Action verarbeiten können. Aus diesen kann sich der Nutzer anschließend eine aussuchen.

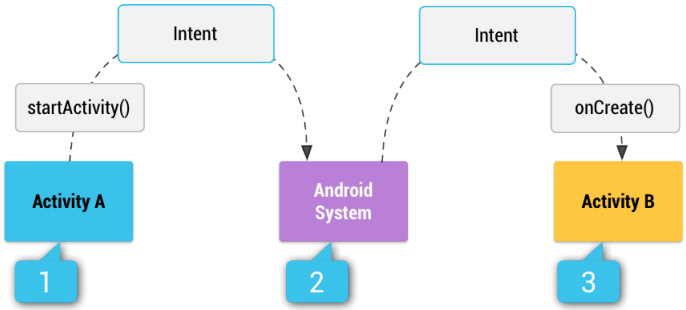

Abbildung 2.4: Intents
Bildquelle: https://developer.android.com/guide/components/intents-filters

Über Intents können Parameterdaten übergeben werden, die in der Zielkomponente ausgelesen werden können. Listing 2.1 zeigt ein explizites Intent, welches über die Methode **putExtra()** Datenparameter übergeben bekommt.

```
1 private void startScanActivity(ConnectionTechnology technology) {
2         Intent intent = new Intent(this, ScanningActivity.class);
3         intent.putExtra(ScanningService.SELECTED_TECHNOLOGY_EXTRA, technology);
4         startActivity(intent);
5     }
```

Listing 2.1: Explizites Intent

2.2.4 Fragments

Ein *Fragment* stellt einen Teilblock einer Activity dar und ist somit eine Art „Sub-Activity“. Wie Activities haben auch Fragments eine eigene Layout Ressource, die angibt wie das Fragment aussieht. Jede Activity kann keine, eines oder mehrere Fragments beinhalten. Fragments haben einen eigenen Lebenszyklus, der jedoch an den der Activity gebunden ist. Wird z.B. eine Activity pausiert, weil sie den Fokus verliert, so werden auch alle Fragments in dieser Activity pausiert. Fragments können während dem Lebenszyklus einer Activity ausgetauscht oder auch in mehreren Activities verwendet werden. Ein Fragment wird durch das Erben von der Klasse **Fragment** definiert.

2.2.5 Services

Services stellen Komponenten dar, die für längere Hintergrundoperationen gedacht sind. Services laufen weiter, selbst wenn der Nutzer zu einer anderen App wechselt. Anders als eine Activity hat ein Service kein User-Interface und somit auch keine zugehörige Layout Ressource. Services können entweder gestartet oder gebunden werden.

Started Services

Started Services werden während der Laufzeit einer App gestartet und laufen solange, bis sie manuell oder vom Android-System beendet werden. Das bedeutet, dass Services selbst nach beenden der App, die den Service gestartet hat, weiter existieren können.

Bound Services

Bound Services werden an Komponenten, d.h. andere Activities oder Services, gebunden. Der Service bleibt dann solange am Leben, bis alle Komponenten, die den Service gebunden haben, beendet wurden.

2.2.6 Android Back-Stack

Activites werden vom Android-System beim Aufrufen auf den sogenannten *Android Back-Stack* gelegt. Wird der Zurück-Button des Android Gerätes betätigt, so wechselt das System zu der Activity, die direkt auf dem Stack liegt. Auch Fragments können programmatisch auf den Back-Stack gelegt werden.

2.2.7 AndroidManifest.xml

Die Datei *AndroidManifest.xml* enthält essentielle Informationen über die Applikation. Diese werden vom Android-System ausgelesen und verarbeitet. Enthalten sind Daten über Activities und Services der Applikation, benötigte Rechte, etc.

2.3 Machine-to-Machine Protokolle

Für eine Machine-to-Machine Kommunikation existieren bereits etliche Kommunikationsprotokolle. Von diesen ist Constrained Application Protocol (CoAP) für das Thema dieser Thesis besonders relevant.

2.3.1 CoAP

Das Constrained Application Protocol (CoAP) ist ein auf die Grundzüge von Representational State Transfer (REST) basierendes Kommunikationsprotokoll, welches von der Internet Engineering Task Force (IETF) für das Internet of Things (IoT) entwickelt wurde [4]. Durch geringen Overhead und seiner Einfachheit ist CoAP für die Kommunikation zwischen Ressourcenbegrenzten Geräten (d.h. geringe Rechenleistung, Low-Power, etc.) ausgelegt. Die Kommunikation basiert auf dem Anfrage (Request)/Antwort (Response) Interaktionsmodell. Verwendet wird CoAP insbesondere bei Mikrocontrollern mit wenig RAM. In diesem Abschnitt werden nur die für das Protokolldesign wichtigen Aspekte von CoAP dargestellt.

Protokollformat

CoAP Nachrichten sind binär kodiert. Die Nachricht beginnt mit einem 4 Byte Header gefolgt von einen sogenannten *Token*, welches 0 bis 8 Byte lang sein darf. Die Optionen, die dem Token folgen sind optional und müssen nicht zwangsläufig angegeben werden. Ein Byte bestehend aus Einsen signalisiert den Beginn des Payloads, welches den Body der Nachricht darstellt. Die Felder im Header haben folgende Bedeutung [7, 3]:

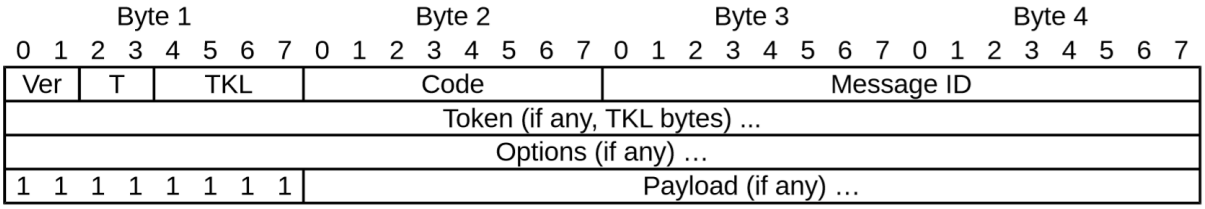

Abbildung 2.5: CoAP Format
Quelle: https://tools.ietf.org/html/rfc7252

- Version (VER): Steht für die Versionsnummer des Protokolls
- Type (T): Stellt dar, ob die Nachricht bestätigt werden muss (*confirmable*) oder nicht (*nonconfirmable*). Es kann auch für die Antwort bzw. Bestätigung einer Nachricht stehen (*acknowledge*) oder für den Fehlerfall (*reset*)

- Token Length (TKL): Gibt die Länge des Tokens in Byte an
- Code: Die ersten drei Bit geben Klasse an, d.h. ob es sich um ein Request, success Response, Client Error Response, etc. handelt, und die letzten fünf Bit stehen für Details zu der Nachricht. Die Details sind für jede Klasse definiert
- Message ID: Steht für die Identifizierungsnummer der Nachricht, zur Verhinderung von duplizierten Nachrichten und um eine Response einem bestimmten Request zuzuordnen
- Options: Verschiedene CoAP spezifische Optionen

Übertragung von Nachrichten

In CoAP können Nachrichten mit einer optionalen Bestätigungsaufforderung versandt werden. Dies wird im Header-Feld *Type* angegeben. Muss die Nachricht nicht bestätigt werden, so wird eine nicht zu bestätigende (nonconfirmable) Nachricht versandt (siehe Abb. 2.6) [7, 2.1]:

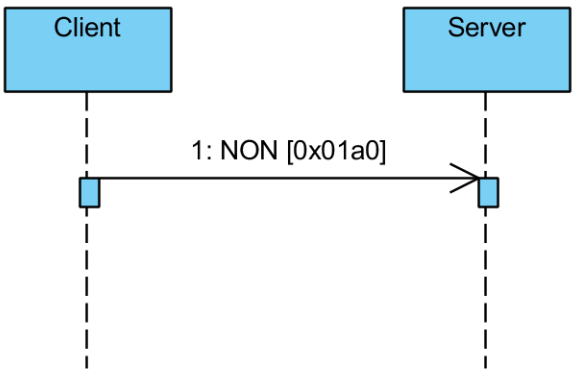

Abbildung 2.6: CoAP Nonconfirmable
Quelle: https://tools.ietf.org/html/rfc7252

Muss eine Nachricht bestätigt werden, so wird diese als zu bestätigende (confirmable) Nachricht versandt (siehe Abb. 2.7) [7, 2.1]:

Die Nachricht wird mit einer sogenannten *Piggybacked Response* beantwortet. Das bedeutet, dass die Bestätigung über den Erhalt der Nachricht (Acknowledge) zusammen mit den angefragten Daten in einer Nachricht versandt werden [7, 5.2.1]. Um die Übertragung einer zu bestätigenden Nachricht sicherzustellen wird die Nachricht erneut gesendet, wenn diese in einem definierten Zeitintervall nicht quittiert worden ist. Die Nachricht wird in immer kleiner werdenden Zeitintervallen verschickt, falls diese länger nicht quittiert wird. Nach einer

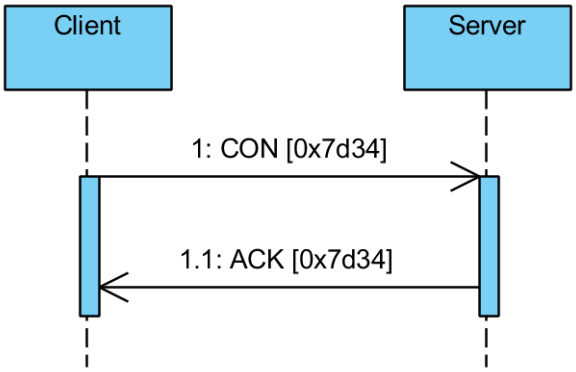

Abbildung 2.7: CoAP Confirmable
Quelle: https://tools.ietf.org/html/rfc7252

definierten Anzahl an erneuten, erfolglosen Übertragungsversuchen wird die ausführende Applikation darüber informiert, dass die Nachricht nicht versandt werden konnte. [7, 4.2]

2.3.2 Andere Protokolle

Neben CoAP existieren etliche weitere Kommunikationsprotokolle, die theoretisch für eine Machine-to-Machine Kommunikation in Frage kommen könnten, jedoch aus verschiedenen Gründen für den Protokollentwurf in dieser Thesis ungeeignet sind.

MQTT

Das Message Queue Telemetry Transport (MQTT) ist ein Nachrichtenprotokoll, dass für die Mashine-to-Machine Kommunikation entwickelt wurde. MQTT ist ein OASIS Standard Protokoll. Ein MQTT-Server, der als *Broker* bezeichnet wird, fungiert als Zustands-Datenbank, während einzelne, meist leistungsschwache Geräte, für den Broker Daten sammeln. Diese stellen MQTT-Clients dar. Ein leistungsstärkeres Gerät kann dann Befehle an den Broker senden, der diese zu den zugehörigen MQTT-Clients weiterleitet. [5]
Zwar könnte die App gleichzeitig als Broker und leistungsstarker Client fungieren, dies würde jedoch das Prinzip eines Brokers von MQTT aushebeln.

DTLS

Datagram Transport Layer Security (DTLS) ist ein auf Transport Layer Security (TLS) aufbauendes Protokoll zur verschlüsselten Übertragung von Daten. Es findet oft Verwendung

bei der Verschlüsselten Übertragung von CoAP-formatierten Nachrichten. Auch wenn verschlüsselte Übertragung ein wichtiger Aspekt ist, so ist dies nicht Bestandteil dieser Thesis und DTLS daher nicht geeignet. [6]

3 Anforderungen

In diesem Kapitel werden die Anforderungen beschrieben an das System, das Protokoll, die App und die Software des RC-Modells beschrieben.

3.1 System

Das System soll folgende Anwendungsfälle (Use Cases) implementieren (siehe Abb. 3.1):

- Der Benutzer soll nach dem Starten der App die Kommunikationstechnologie (BLE, WiFi, etc.) auswählen können
- Der Benutzer soll mit der ausgewählten Kommunikationstechnologie nach verfügbaren Geräten scannen können. Diese sollen auf dem Mobile-Gerät angezeigt werden
- Mit dem selektierten Gerät soll sich die App verbinden können
- Der Motor des RC-Modells soll an- und ausgeschaltet werden können
- Die Bewegungsrichtung des RC-Modells soll gesetzt werden können, d.h vorwärts und rückwärts
- Das RC-Modell soll beschleunigt werden können
- Die Auslenkung des RC-Modells soll verstellbar sein
- Der Benutzer soll das RC-Modell abbremsen können
- Der Benutzer soll in der Lage sein die Hupe des RC-Modells auszulösen
- Der Benutzer soll das Licht des RC-Modells an- und ausschalten können

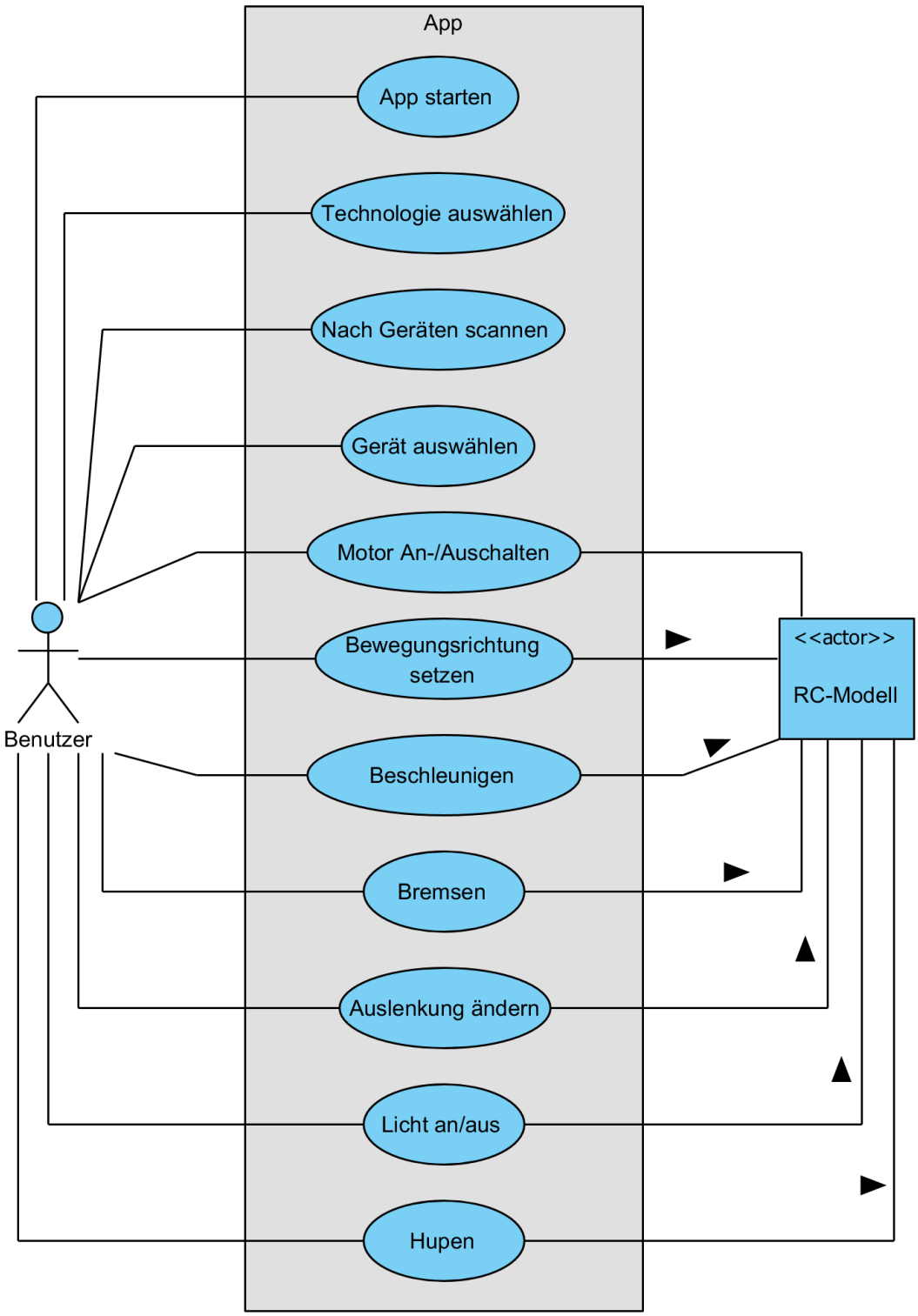

Abbildung 3.1: Anwendungsfälle des System

3.2 Protokoll

Das Protokoll soll folgende Anforderungen erfüllen:

- Das Protokoll soll leichtgewichtig sein, d.h. nur aus so vielen Bytes wie nötig bestehen
- Das Protokoll soll möglichst leicht erweiterbar sein
- Das Protokoll soll so generisch wie möglich sein, d.h. das die definierten Befehle sowohl für RC-Modelle zu Land als auch zu Wasser und in der Luft geeignet sein sollen

3.3 App

Die App für das steuernde Mobile-Gerät soll folgende Anforderungen erfüllen:

- Die App soll leicht um weitere Kommunikationstechnologien erweiterbar sein
- Die App soll feststellen können auf welcher Version des Protokolls das RC-Modell kommuniziert und anschließend Nachrichten auf dieser Version des Protokolls versenden
- Die App soll feststellen können welchen Typ (Fahrzeug, Schiff, Drohne, etc) das RC-Modell hat und die passende GUI zur Steuerung laden
- Die App soll erkennen, welche Funktionalitäten das zu steuernde RC-Modell hat. Ist eine bestimmte Funktionalität (z.B. Hupen) nicht vorhanden, so soll der Benutzer diese auch nicht betätigen können
- Die App soll einen Verbindungsverlust zum steuernden RC-Modell innerhalb eines festgelegten Zeitraums bemerken und den Benutzer über den Verbindungsverlust informieren
- Im Falle des Verbindungsverlustes soll die App dem Benutzer die Möglichkeit bieten eine erneute Verbindung mit dem Gerät aufzubauen
- Die App soll bemerken, ob der Benutzer noch mit der App interagieren kann. Falls nicht, z.B. weil eine andere App in den Vordergrund gerückt ist, soll diese das RC-Modell informieren

3.4 Software des RC-Modells

Die Software des RC-Modells muss folgende Anforderungen erfüllen:

- Die Software soll der App bei einer entsprechenden Anfrage mitteilen können welchen Typ (Fahrzeug, Schiff, Drohne, etc) das RC-Modell hat
- Die Software soll der App bei einer entsprechenden Anfrage mitteilen können auf welcher Version des Protokolls das RC-Modell kommuniziert
- Die Software soll einen Verbindungsverlust zum steuernden Mobile-Gerät innerhalb eines festgelegten Zeitraums bemerken können
- Die Software soll die App darüber benachrichtigen, wenn eine Anweisung nicht bekannt oder nicht verstanden wurde, und somit nicht ausgeführt werden kann

4 Konzept und Design

In diesem Kapitel wird das Konzept und das Design vorgestellt. Das Design wurde durch iteratives Vorgehen, bestehened aus wöchentlichen Absprachetermine, entwickelt. Die Entwicklung nach dem iterativen Vorgehensmodell wurde gewählt, da die Anforderungen iterativ erarbeitet wurden und das Design und die Architektur dementsprechend laufend angepasst werden mussten. Die Modellierung der Software erfolgt mit der Unified Modeling Language, kurz UML, mithilfe der Literatur [9].

4.1 UI-Ablaufplan der App

Für das Verhalten der App dient der UI-Ablaufplan aus Abb. 4.1. Aus dem UI-Ablaufplan sind drei Activities zu erkennen:

- **TechnologySelectionActivity:** Hier wählt der Benutzer die gewünschte Kommunikationstechnologie aus
- **ScanningActivity:** Hier kann der Benutzer mit der gewählten Kommunikationstechnologie nach Geräten scannen
- **ControlActivity:** In dieser Activity steuert der Benutzer das ausgewählte RC-Modell

Mithilfe des Android-Backstacks werden diese Activities verlassen und es wird zur letzten Activity gewechselt. Wird der Zurück-Button in der TechnologySelectionActivity betätigt, so wird die App verlassen.

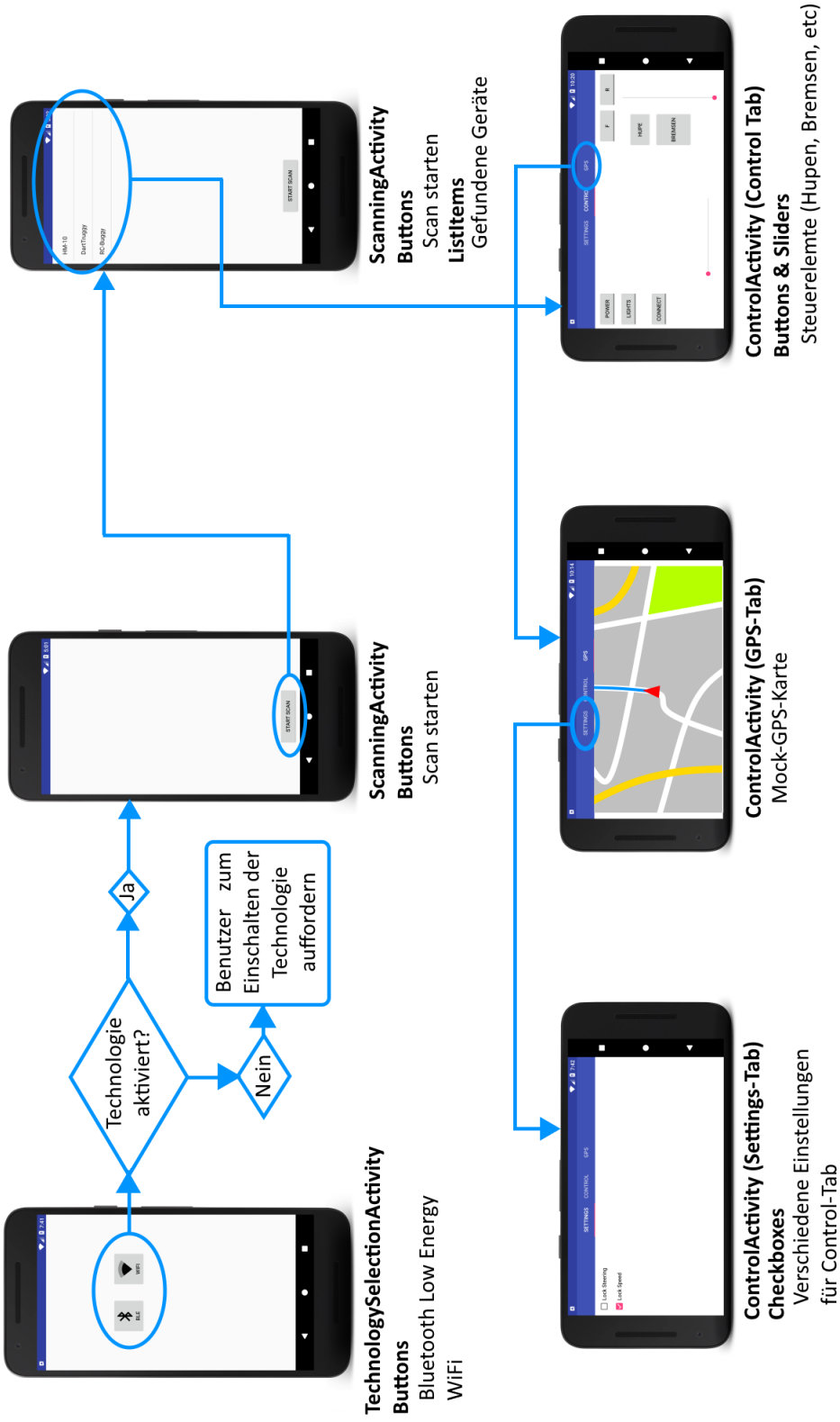

Abbildung 4.1: App UI-Ablaufplan

4.2 Verhalten des Systems

App starten

Beim Starten der App wird die Activity zum Auswählen der Kommunikationstechnologie gestartet und dem Benutzer alle implementierten Kommunikationstechnologien angezeigt.

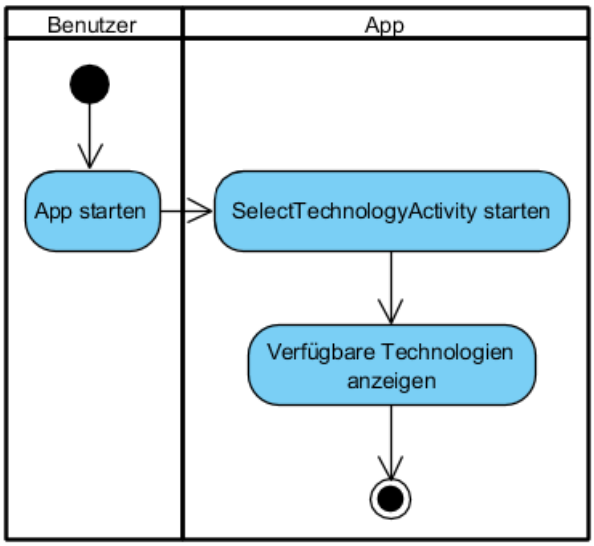

Abbildung 4.2: App starten

Technologie auswählen

Wählt der Benutzer über einen Klick eine Kommunikationstechnologie aus, so wird zuerst geprüft ob diese Technologie aktiviert ist. Ist dies nicht der Fall, wird der Benutzer durch einen modalen Dialog vom Android-System aufgefordert diese zu aktivieren. Aktiviert der Benutzer die Kommunikationstechnologie wird zur ScanActivity gewechselt. Bricht der Benutzer die Aktivierung der Kommunikationstechnologie ab, bleibt die App in der TechnologySelectionActivity.

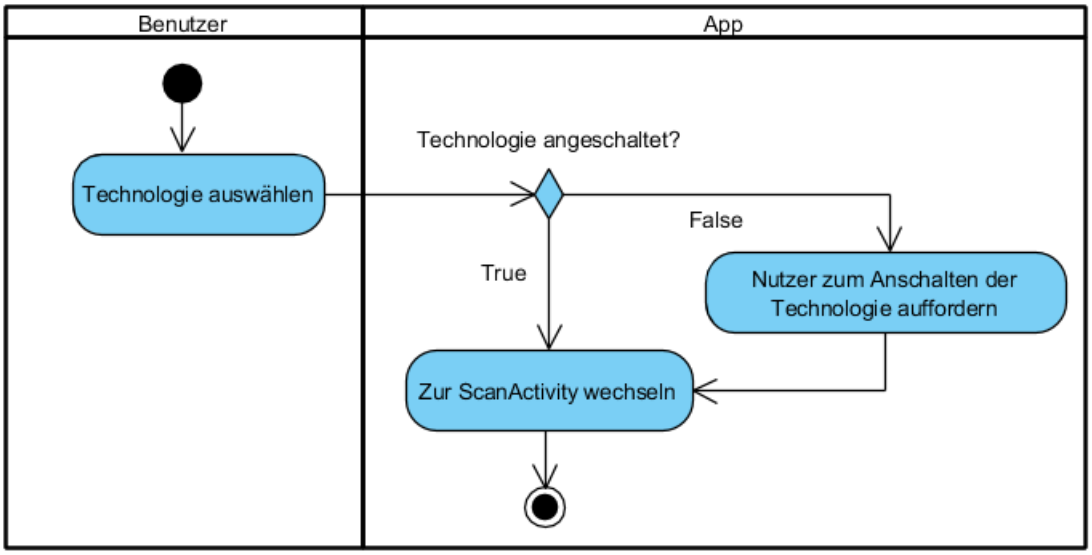

Abbildung 4.3: Kommunikationstechnologie auswählen

Nach Geräten scannen

Sobald der Benutzer das Scannen startet wird solange nach verfügbaren Geräten gescannt werden, bis entweder die festgelegte Scan-Zeit abgelaufen ist oder der Benutzer den Vorgang abbricht. In beiden Fällen werden dem Benutzer alle gefundenen Geräte angezeigt.

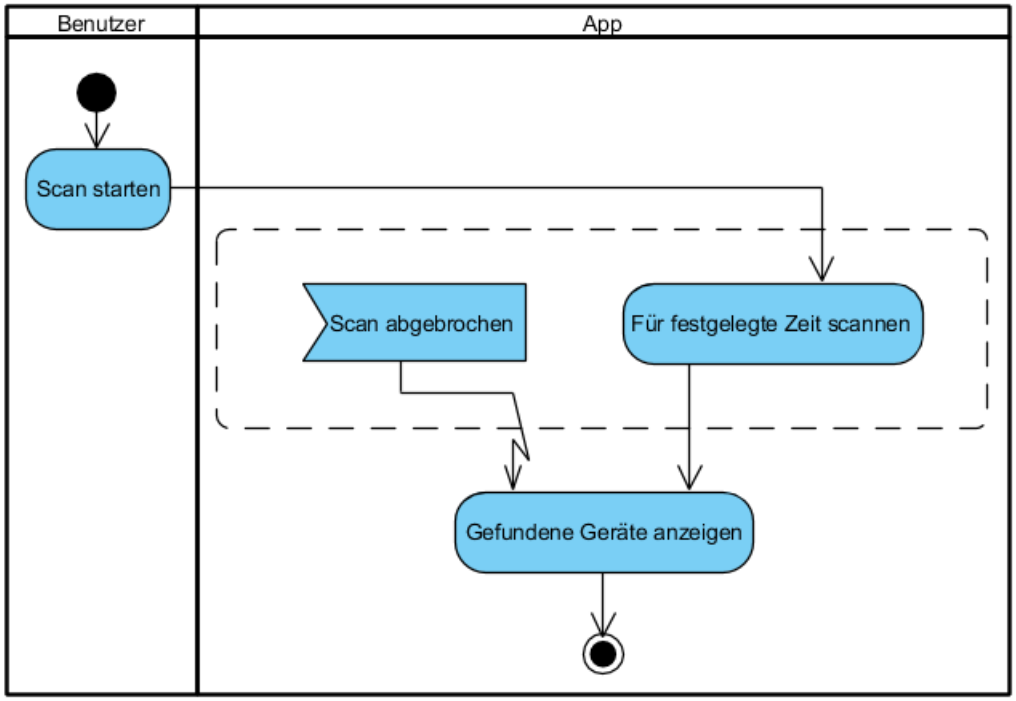

Abbildung 4.4: Nach Geräten scannen

Modell auswählen

Wählt der Benutzer ein angezeigtes RC-Modell aus, so wird die Activity zum Steuern des RC-Modells gestartet. Nach dem Starten werden Protokoll-Version und Modelltyp abgefragt und die passende GUI erstellt und angezeigt.

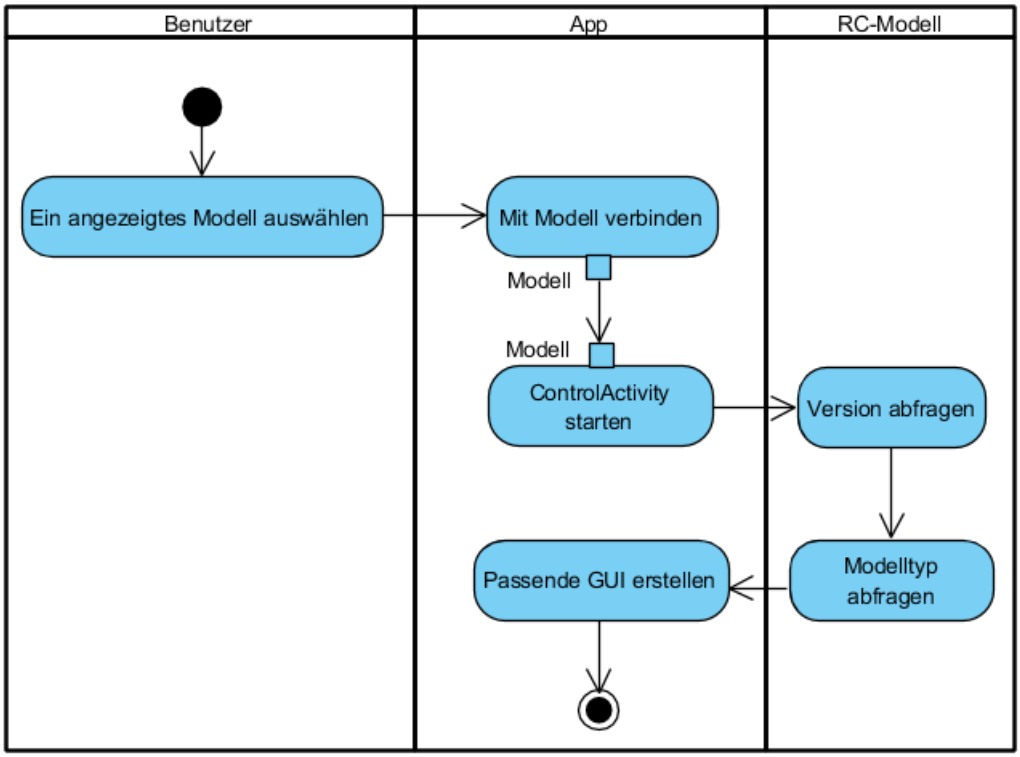

Abbildung 4.5: RC-Modell auswählen

Motor an-/auschalten

Soll der Motor an- oder ausgeschaltet werden, so wird eine passende Nachricht erstellt und an das RC-Modell übertragen. Das RC-Modell schaltet anschließend den Motor an bzw. aus.

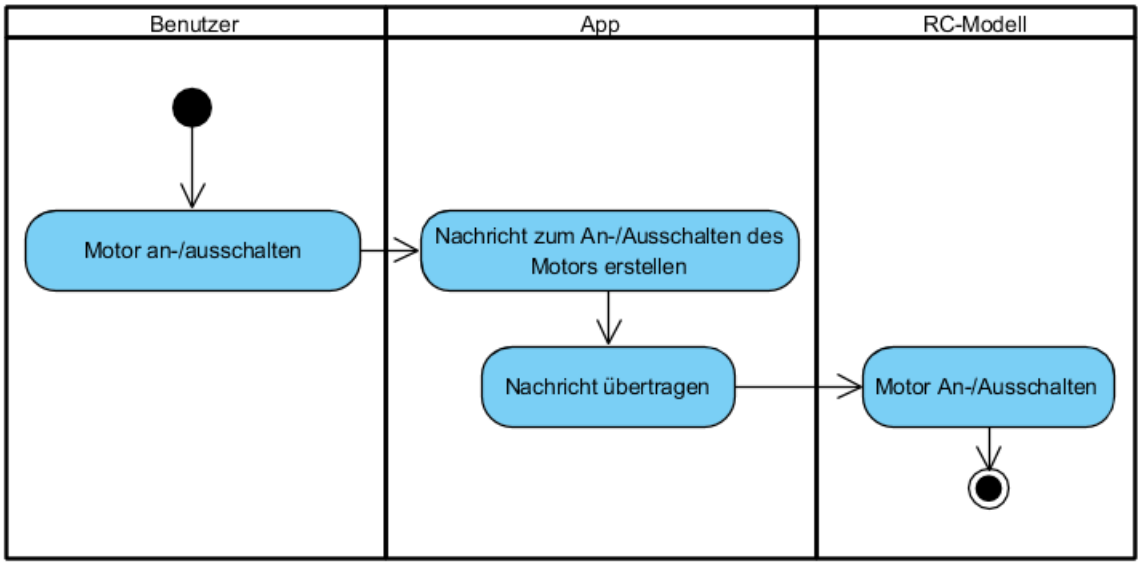

Abbildung 4.6: Motor an/ausschalten

Bewegungsrichtung setzen

Ändert der Benutzer die Bewegungsrichtung des RC-Modells, so wird eine entsprechende Nachricht erstellt und das RC-Modell übertragen.

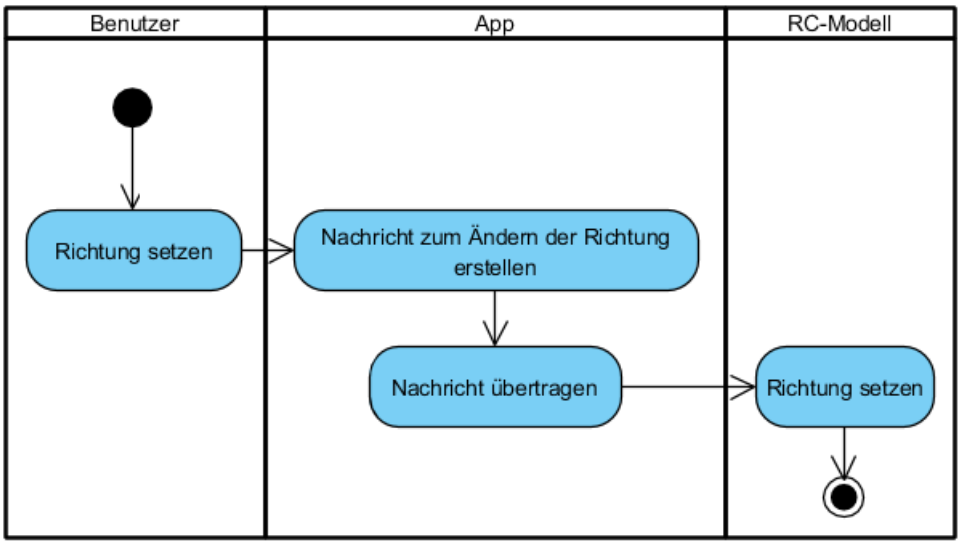

Abbildung 4.7: Bewegungsrichtung setzen

Geschwindigkeit setzen

Ändert der Benutzer die Geschwindigkeit wird eine entsprechende Nachricht erstellt und an das RC-Modell übertragen. Es wird maximal alle 50 ms eine neue Nachricht zum Ändern der Geschwindigkeit gesendet, um das RC-Modell nicht mit Nachrichten zu überladen. Da das PWM-Signal eines analogen Servos nur ungefähr alle 20 ms erneuert wird, würde das Übertragen von Nachrichten zum ändern der Geschwindigkeit, in einer höheren Frequenz wenig Sinn machen. [8]

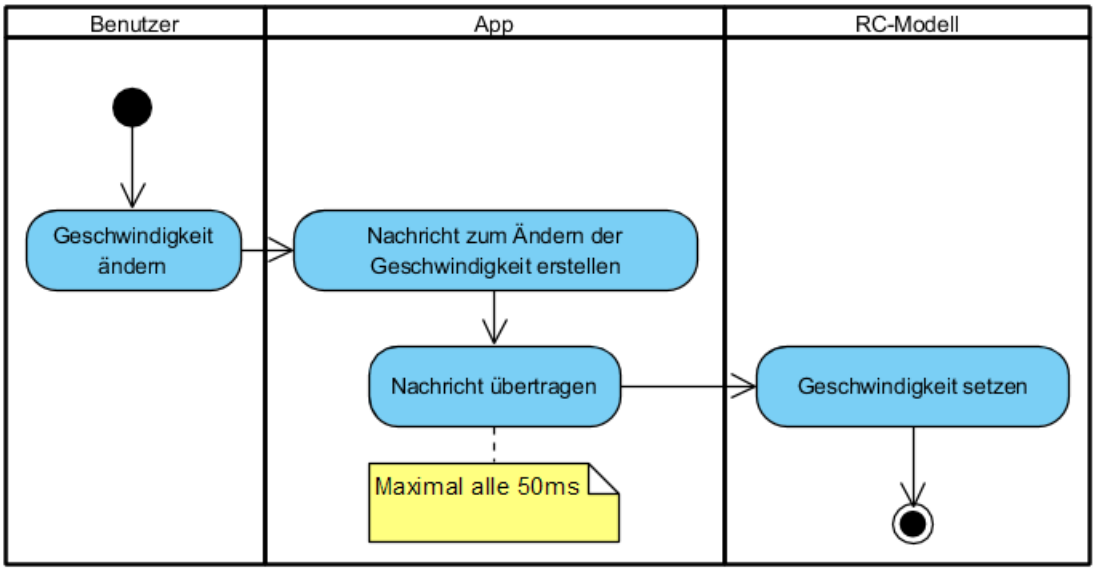

Abbildung 4.8: Geschwindigkeit setzen

Bremsen

Betätigt der Benutzer die Bremse wird eine Nachricht zum Bremsen erstellt und an das RC-Modell übertragen. Das RC-Modell löst daraufhin seine Bremse aus. Sobald der Benutzer die Bremse wieder loslässt wird eine Nachricht zum loslassen der Bremse an das RC-Modell übertragen.

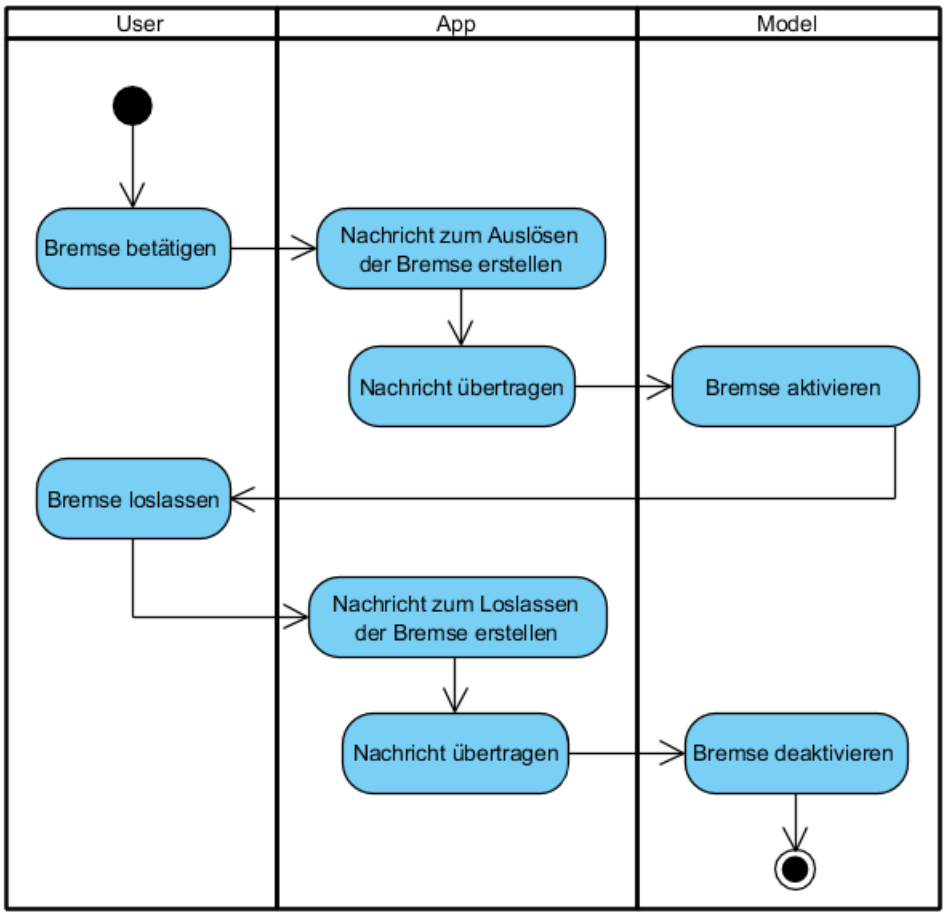

Abbildung 4.9: Bremsen

Auslenkung ändern

Beim Ändern der Auslenkung durch den Benutzer verhält sich das System analog zum Ändern der Geschwindigkeit. Auch in diesem Fall wird eine passende Nachricht erstellt und maximal alle 50ms an das RC-Modell geschickt.

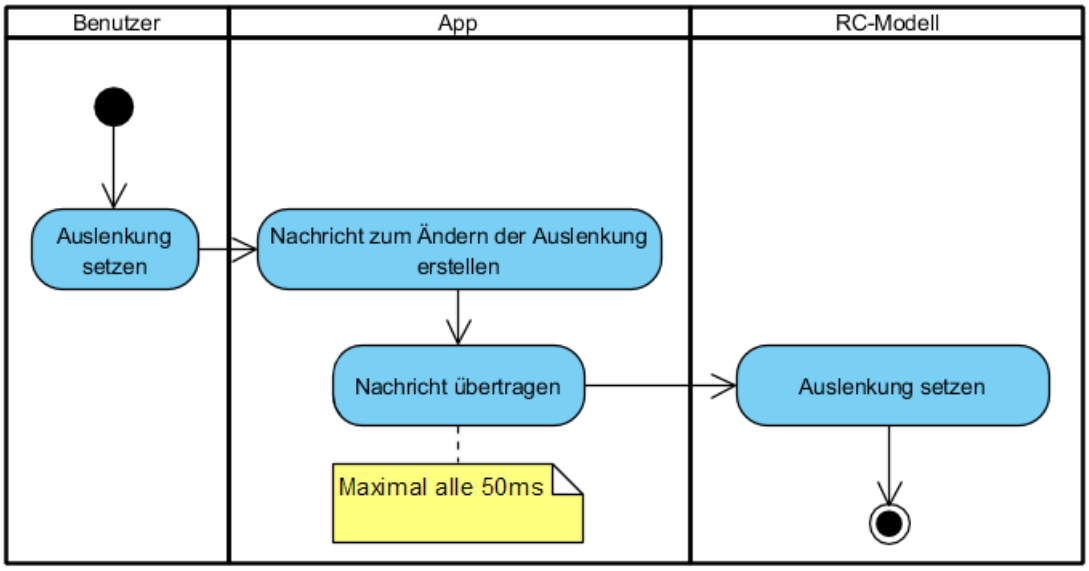

Abbildung 4.10: Auslenkung setzen

Licht ein/aus

Zum An- bzw. Ausschalten des Lichtes wird eine passende Nachricht erstellt und an das RC-Modell übertragen.

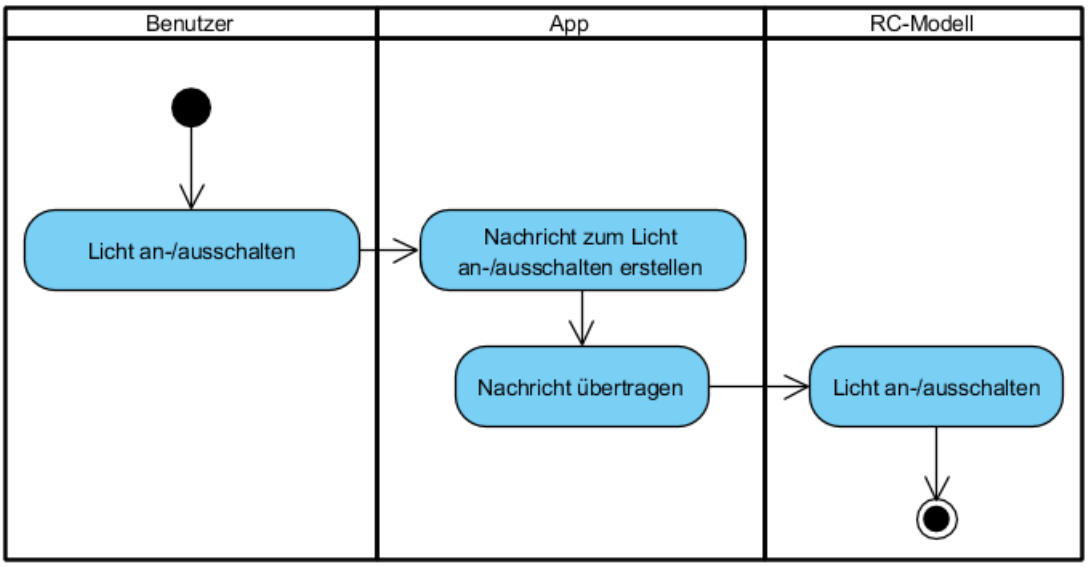

Abbildung 4.11: Licht an-/ausschalten

Hupen

Betätigt der Benutzer die Hupe, so wird eine dementsprechende Nachricht erstellt und an das RC-Modell übertragen. Nachrichten im selben Format werden in einem festgelegten Intervall an das RC-Modell geschickt. Das RC-Modell löst für einen festen Zeitraum seine Hupe aus, sobald es eine Nachricht zum Auslösen der Hupe empfängt. Lässt der Benutzer die Hupe los werden keine weiteren Nachrichten zum Hupen übertragen.

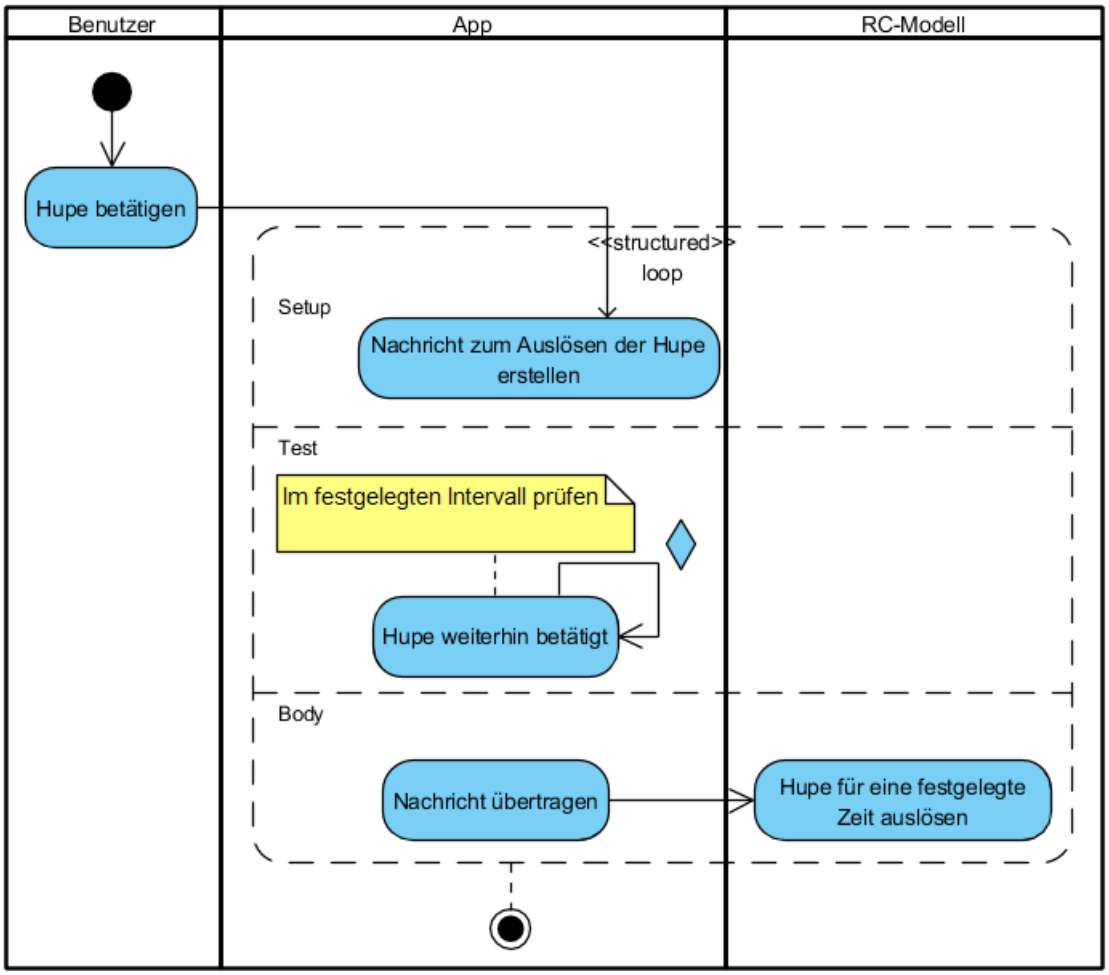

Abbildung 4.12: Hupe betätigen

4.3 Entwurf des Kommunikationsprotokolls

4.3.1 Format

Das Format des Kommunikationsprotokolls orientiert sich stark an dem Format von CoAP (siehe 2.3.1):

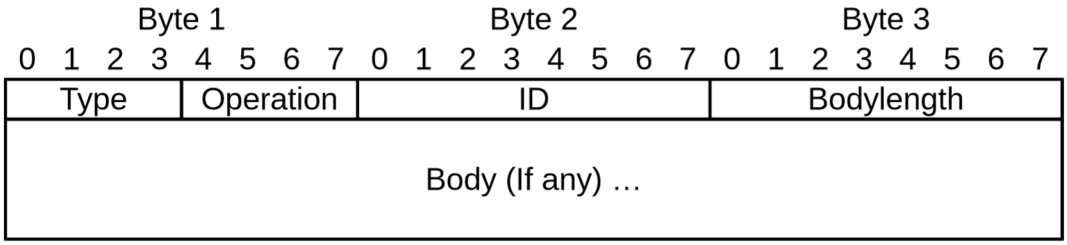

Abbildung 4.13: Kommunikationsprotokoll Format

Der Header besteht immer aus drei Bytes, gefolgt von einem optionalen Body mit variabler Anzahl an Bytes. Anders als beim CoAP hat das Protokoll keine Felder für ein Token, Optionen oder eine Message-ID. Nur die für die erste Version nötigen Felder sind definiert.

Die Felder des Protokolls haben folgende Bedeutung:

- **Type:** Dies steht für den Typ der Nachricht. Dabei wurden die Bezeichnungen *CONFIRMABLE* für zu bestätigende Nachrichten, *NONCONFIRMABLE* für Nachrichten, die nicht bestätigt werden müssen und *ACKNOWLEDGE* für das Bestätigen bzw. Quittieren einer Nachricht aus dem CoAP übernommen. Anstatt *RESET* ist *ERROR* als Typ für nicht erfolgreich verarbeitete Nachrichten definiert. Der Grund dafür ist, dass in dieser ersten Version des Protokolls bei einer nicht verstandenen Nachricht immer ein Fehler zurückgeschickt werden soll.
- **Operation:** Dieses Feld definiert die Methode, d.h. welche Aktion durchgeführt werden soll. Für die erste Version sind fünf verschiedene Operationen definiert:
  - **GET** Wird für das Abfragen eines bestimmten Wertes verwendet
  - **SET** Wird für das Setzen einen bestimmtes Wertes verwendet
  - **STATUS** Wird zum Abfragen von Zuständen verwendet, z.B. den Status der Verbindung

  - **SUBSCRIBE** Wird verwendet, um über Änderungen eines Wertes informiert zu werden, d.h diesen Wert zu abonnieren. Der Empfänger soll den Sender über Änderungen an diesem Wert informieren
  - **NOTIFIY** Benachrichtigt den Abonnenten über die Änderung eines Wertes, der vorher abonniert wurde
- **ID:** Dieses Feld steht für den Wert, auf welchen eine Operation durchgeführt werden soll. Die in der ersten Version definierten IDs sind:
  - **CONNECTION** Zum Prüfen der Verbindung zum RC-Model
  - **POWER** Für das An- bzw. Ausschalten des Motors des RC-Modells
  - **SPEED** Die Geschwindigkeit des RC-Modells in Prozent. Von 0% (Stillstand) bis 100% (Maximale Geschwindigkeit)
  - **DIRECTION** Zum Setzen der Bewegungsrichtung, d.h. vorwärts oder rückwärts
  - **STEERING** Die Auslenkung des RC-Modells in Prozent. Von -100% (Vollausschlag links) bis 100% (Vollausschlag rechts)
  - **BRAKE** Zum Abbremsen (Vollbremsung) des RC-Modells
  - **HONK** Zum Betätigen der Hupe
  - **LIGHTS** Zum Ein- bzw. Ausschalten des Lichts
  - **VERSION** Zum Abfragen der Version des Kommunikationsprotokolls
  - **MODELTYPE** Zur Abfrage um welchen Typ es sich beim RC-Model handelt, d.h. ob es sich um ein Auto, Schiff, etc. handelt
  - **CONTROL** Zum Benachrichtigen des RC-Modells über den Kontrollstatus
- **Bodylength:** Dieses Feld gibt die Anzahl an Bytes an, aus denen der Body besteht
- **Body:** Dieses Feld steht für den Payload der Nachricht. Hier wird der Wert gesetzt, der mit der Nachricht übertragen werden soll

Das Protokoll ist damit leicht um weitere Operationen und IDs erweiterbar. Durch einen drei Byte Header ist es außerdem sehr leichtgewichtig. Die definierten IDs können prinzipiell sowohl für RC-Automodelle, RC-Schiffsmodelle, etc. verwendet werden.

4.3.2 Definierte Nachrichten

Für das Protokoll sind bestimmte Nachrichten, d.h. Kombinationen von Types, Operations und IDs, definiert. Die Nachrichten werden im Format *TYPE : OPERATION : ID* dargestellt:

- **Verbindungscheck:** Zum Überprüfen der Verbindung muss eine Nachricht mit leeren Body und dem Header: *CONFIRMABLE : STATUS : CONNECTION* verschickt werden. Diese muss mit einem *ACKNOWLEDGE : STATUS : CONNECTION* quittiert werden.
- **Version und Modeltyp:** Sowohl die Version des Protokolls und als auch der Modeltyp dürfen nur mit der Nachricht *CONFIRMABLE : GET : VERSION* bzw *CONFIRMABLE : GET : MODELTYPE* abgefragt werden. Diese müssen mit einem *ACKNOWLEDGE* und dem Wert der Version bzw. des Modeltyps im Body beantwortet werden.
- **Kontrollstatus:** Diese Nachricht soll das RC-Modell darüber informieren, ob der Nutzer das RC-Modell steuern kann. Wenn z.B. die App auf dem steuernden Mobile Gerät in den Hintergrund rückt weil ein Anruf kommt, so soll das RC-Modell den Kontrollverlust mitbekommen. Dies wird über das Versenden einer Nachricht im Format *CONFIRMABLE : STATUS : CONTROL* mit einem boolschen Wert im Body mitgeteilt.

4.4 Design der App

4.4.1 Darstellung über ein MockUp

In Abschnitt 4.1 werden drei Activities genannt, die die App beinhaltet:

- **TechnologySelectionActivity** In dieser Activity wählt der Benutzer die Kommunikationstechnologie aus
- **ScanningActivity** In dieser Acitvity kann der Benutzer über die ausgewählte Technologie nach Geräten scannen
- **ControlActivity** Diese Activity ermöglicht dem Benutzer das ausgewählte RC-Modell zu steuern

Das äußerliche Design der einzelnen Activities erfolgt über ein MockUp. Android Studio bietet hierfür einen Emulator an, der das Verhalten jedes echten Mobile-Modells simulieren kann. Die Darstellung erfolgt über ein emuliertes Nexus 5X.

TechnologySelectionActivity

Die TechnologySelectionActivity zeigt dem Benutzer alle implementierten Kommunikationstechnologien an (siehe Abb. 4.14) . Wählt der Benutzer eine Kommunikationstechnologie aus, so wird die ScanningActivity gestartet. Falls diese Kommunikationstechnologie nicht aktiviert ist, wird der Benutzer dazu aufgefordert, diese zu aktivieren.

Abbildung 4.14: TechnologySelectionActivity

ScanningActivity

In der ScanningActivity kann der Benutzer nach Geräten mit der ausgewählten Kommunikationstechnologie scannen (siehe Abb. 4.15).

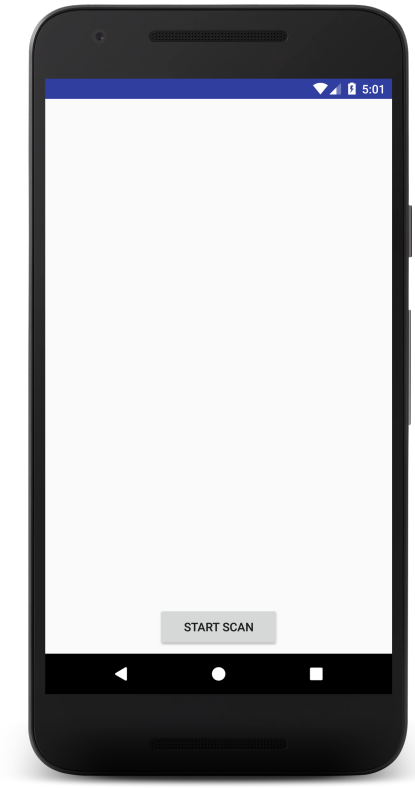

Abbildung 4.15: ScanningActivity

Wird das Scannen durch den Benutzer oder nach einer festgelegten Zeit beendet, werden alle gefunden Geräte aufgelistet mit welchen das Mobile-Gerät eine Verbindung herstellen kann (siehe Abb. 4.16). Dabei wird nicht unterschieden, ob das gefundene Gerät ein steuerbares RC-Modell ist.

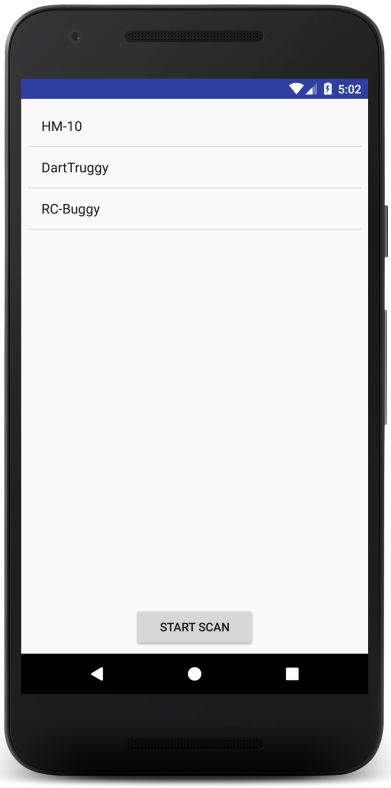

Abbildung 4.16: ScanningActivity mit gefundenen Geräten

ControlActivity

Die ControlActivity erlaubt es dem Benutzer das ausgewählte RC-Modell zu steuern. In dieser Activity gibt es drei Tabs zur Auswahl. Abbildung 4.17 zeigt den Control-Tab. Hier kann der Benutzer Interaktionen wie das Licht ein- und ausschalten, Ändern der Auslenkung und Geschwindigkeit, etc. durchführen.

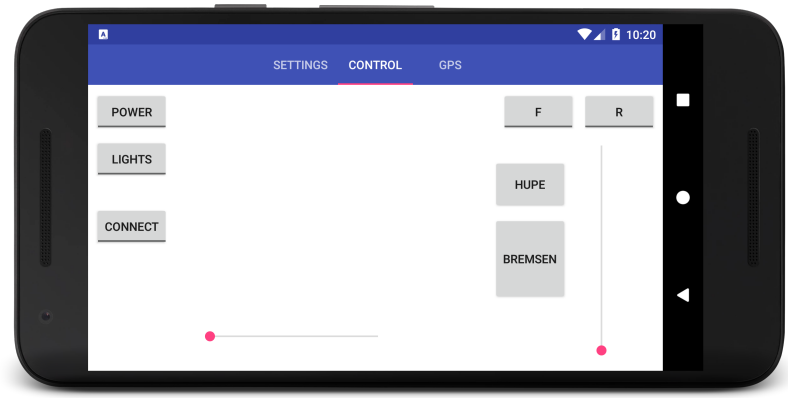

Abbildung 4.17: ControlActivity im Control-Tab

Abbildung 4.18 stellt den GPS-Tab dar. Diese beinhaltet nur ein Bild einer Karte, wie sie in ähnlicher Form bei Google-Maps zu finden wäre. Sie stellt einen Platzhalter für zukünftige Funktionen dar, in welcher der Benutzer z.B. GPS-Koordinaten für das RC-Modell setzen kann.

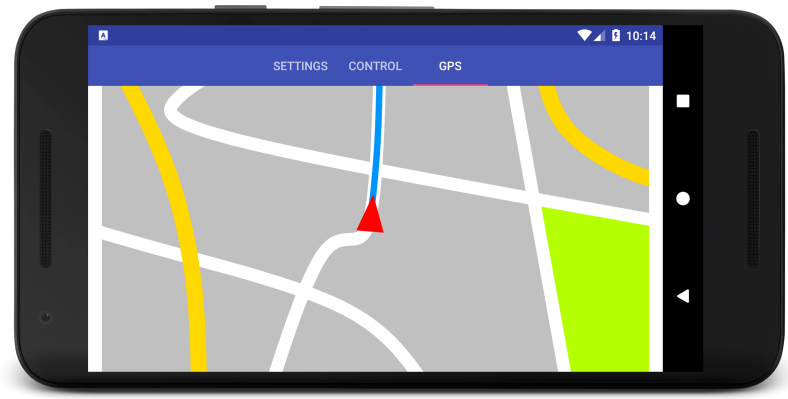

Abbildung 4.18: ControlActivity im GPS-Tab

Abbildung 4.19 stellt den Settings-Tag dar. Hier werden verschiedene Einstellungen festgelegt, wie die Anzahl der Versuche eine nicht bestätigte Confirmable-Nachricht erneut zu übertragen.

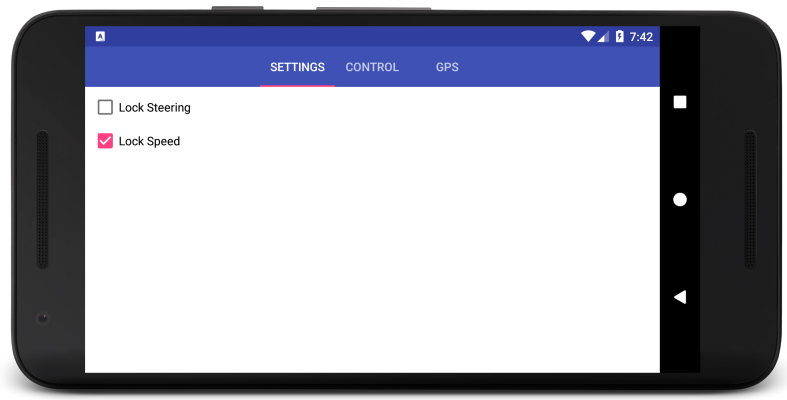

Abbildung 4.19: ControlActivity im Settings-Tab

4.4.2 Klassenarchitektur

Dieser Abschnitt behandelt die statische Architektur der App. Das Klassendiagramm in Abb. 4.22 ist nicht direkt mit den Klassendiagrammen in Abb. 4.23 und Abb. 4.24 verbunden. Dies wurde so gewählt, weil wie in Kapitel 2 erwähnt, Activities relativ unabhängig voneinander existieren. Für Android-Apps ist es daher sinnvoll, Activities in einzelne Klassendiagramme zu modellieren. Alle Services sind als bounded Services zu verstehen.

Aus Kapitel 2 ist bekannt, dass Activities relativ unabhängig voneinander existieren. Dies ist auch der Fall für die Activities TechnologySelectionActivity, ScanningActivity und ControlActivity. Es existieren jedoch Klassen, die als Parameter für die Activities fungieren. Abb. 4.20 stellt die Klassen dar, die mehr als einer Activity bekannt sind. Das Enum **ConnectionTechnology** wird als Parameter in die Intents zum Starten der ScanningActivity oder der ControlActivity übergeben um den Activities mitzuteilen, welche Kommunikationstechnologie ausgewählt wurde. Die Klasse **Device** beinhaltet den Namen und die Adresse des in der ScanningActivity ausgewählten Gerätes.

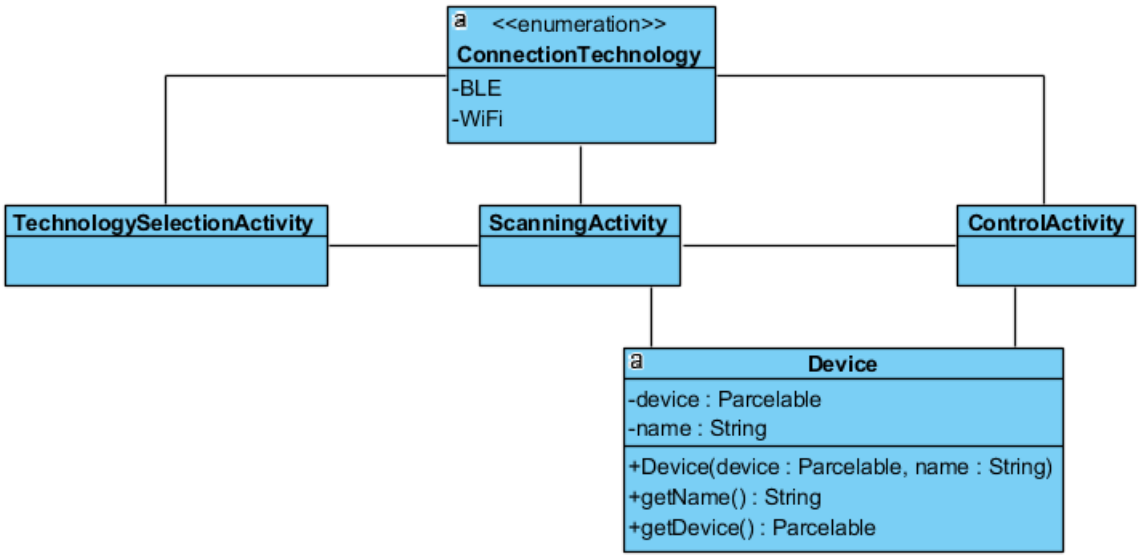

Abbildung 4.20: Klassendiagramm der App Activities

Die TechnologySelectionActivity verwendet das bereits erwähnte Enum **ConnectionTechnology** um der ScanningActivity mitzuteilen, welche Kommunikationstechnologie durch den Benutzer ausgewählt wurde (siehe Abb. 4.21).

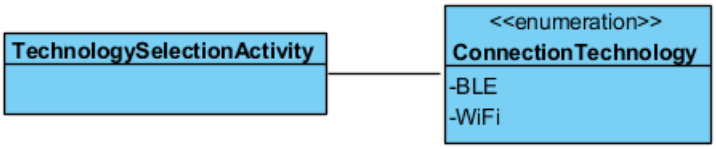

Abbildung 4.21: Klassendiagramm TechnologySelectionActivity

Das Klassendiagramm in Abb. 4.22 stellt die statische Architektur der ScanningActivity dar. Die einzelnen Klassen und Interfaces haben hierbei folgende Funktion:

- Das Interface **ScanningProxy** legt Methoden fest, die von den realisierten Scanning-Proxys eingebunden werden müssen. Diese müssen Methoden zum Scannen und zurückgeben aller gefundenen Geräte implementieren
- Die Klassen **BleScanningProxy** und **WiFiScanningProxy** stellen die Realisierung des Interfaces ScanningProxy dar
- Der **ScanningService** wird beim Starten der Activity erstellt und an diese gebunden. Beim binden des Services wird der Proxy für die ausgewählte Kommunikationstech-

nologie gewählt. Der Service bietet der Activity Methoden zum Scannen und erfassen aller gefundenen Geräte an

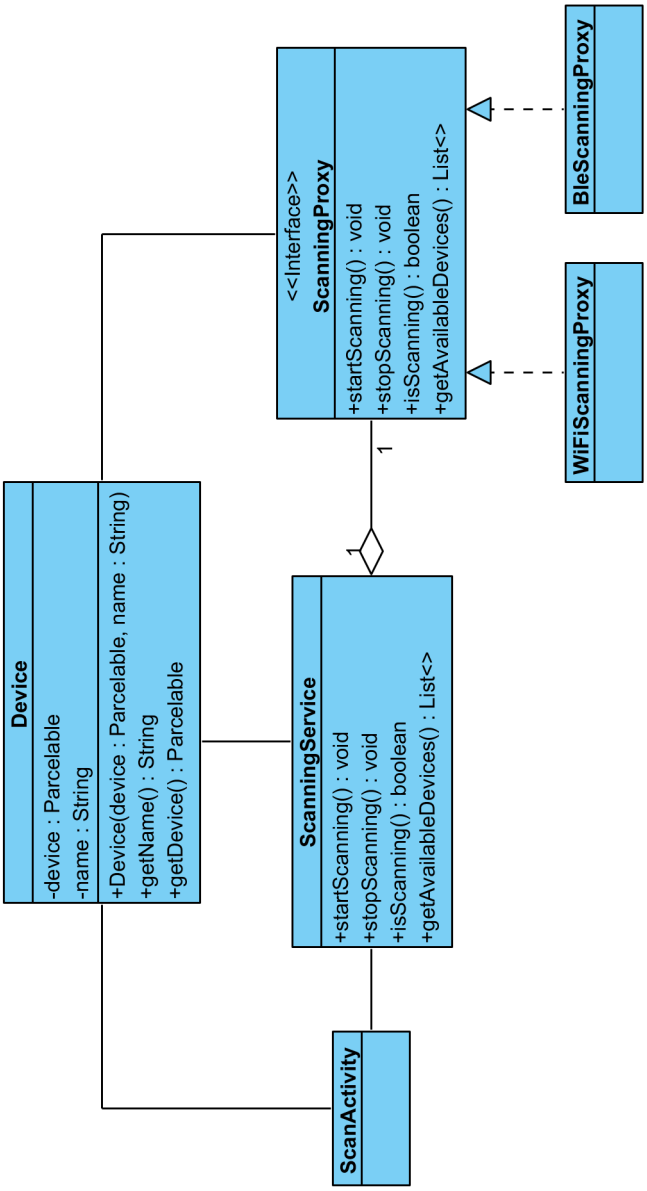

Abbildung 4.22: Klassendiagramm ScanningActivity

Die Klassendiagramme in Abb. 4.23 und Abb. 4.24 stellen die statische Architektur der ControlActivity dar. Die einzelnen Klassen und Interfaces haben hierbei folgende Funktion:

- Das **BaseControlFragment** ist das Fragment, dass im Tab „Control“ angezeigt wird. Es stellt die Basisklasse für ableitende Fragments dar. Abhängig vom Modelltyp wird

es durch ein für diesen Modelltyp geeignetes Fragment (z.B. CarControlFragment für Fahrzeuge) ersetzt

- Das **SettingsFragment** wird beim Wechsel in den Tab „Settings“ angezeigt. Es bietet dem Benutzer die Möglichkeit, verschiedene Einstellungen anzupassen
- Das **NavigationFragment** wird beim Wechsel in den Tab „GPS“ angezeigt. Es bietet dem Benutzer an verschiedene GPS-Interaktionen auszuführen, wie das setzten von Navigationskoordinaten.
- Das Interface **MessageListener**, welches von den ControlFragments und der ControlActivity implementiert wird, legt eine Methode fest, die beim Empfangen von Nachrichten des RC-Modells aufgerufen wird
- Das Interface **ConnectionListener**, welches von den ControlFragments und der ControlActivity implementiert wird, legt Methoden fest, die bei einer Änderung des Verbindungsstatus zum RC-Modell aufgerufen werden
- Der Service **MessagingService** wird beim Erstellen der ControlActivity gebunden und bietet der ControlActivity an, Nachrichten zu verschicken und zu Empfangen. Beim Empfangen einer Nachricht leitet der Service diese an alle seine Listener weiter. Zum Verschicken von Nachrichten sind Methoden, wie z.B. setValue() definiert
- Der Service **ConnectionService** wird beim Erstellen der ControlActivity gebunden und prüft beim Verbinden zu einem RC-Modell regelmässig, ob noch eine Verbindung vorhanden ist. Der Service informiert außerdem alle seine Listener über Änderungen am Verbindungsstatus zum RC-Modell
- Der abstrakte Service **TransmissionService** ist die Basisklasse für die Services der App. Der Service beinhaltet eine Nachrichten-Queue und bietet Methoden behandeln der Queue und zum Versenden von Nachrichten über eine Proxy-Klasse an
- Die Klasse **Message** stellt eine Nachricht nach der ersten Version des designeten Protokolls dar. Die Klasse beinhaltet statische Enums für die Protokollfelder Type, Operation und ID und hat für jedes Protokollfeld ein Attribut, einschließlich eines Attributes für den Protokollbody. Die Klasse bietet außerdem eine Methode, die die Nachricht als Byte-Array zurückgibt
- Der **MessageParser** erstellt Message Objekte aus übergebenen Parameter. Der Parser kann ein Message Objekt sowohl aus den einzelnen Protokollfeldern, die als Parameter übergeben werden, als auch aus einer Nachricht in Byte-Array Form erstellen. Der Parser erstellt Nachrichten in Form der Protokollversion, die im Parser bei der Erstellung festgelegt wird

- Jede Service implementiert das Interface **ProxyListener**. Dieser Listener legt Methoden fest, die fast analog zum MessageListener und ConnectionListener sind. Der Unterschied ist, dass empfangene Nachrichten nicht als Message-Objekt sondern als Byte-Array an diesen Listener übergeben werden
- Das Interface **Proxy** legt Methoden fest, die für die Kommunikation mit einem RC-Modell von der realisierenden Klasse, wie **BleProxy** oder **WiFiProxy**, implementiert werden müssen. Mit dem Proxy kann über die Services der Verbindungsstatus geändert und Nachrichten als Byte-Array an das RC-Modell verschickt und von diesem Empfangen werden. Damit der Proxy empfangene Nachrichten an die Services weiterleitet und sie über den Verbindungsstatus informiert, binden sich diese als Proxy-Listener an den Proxy

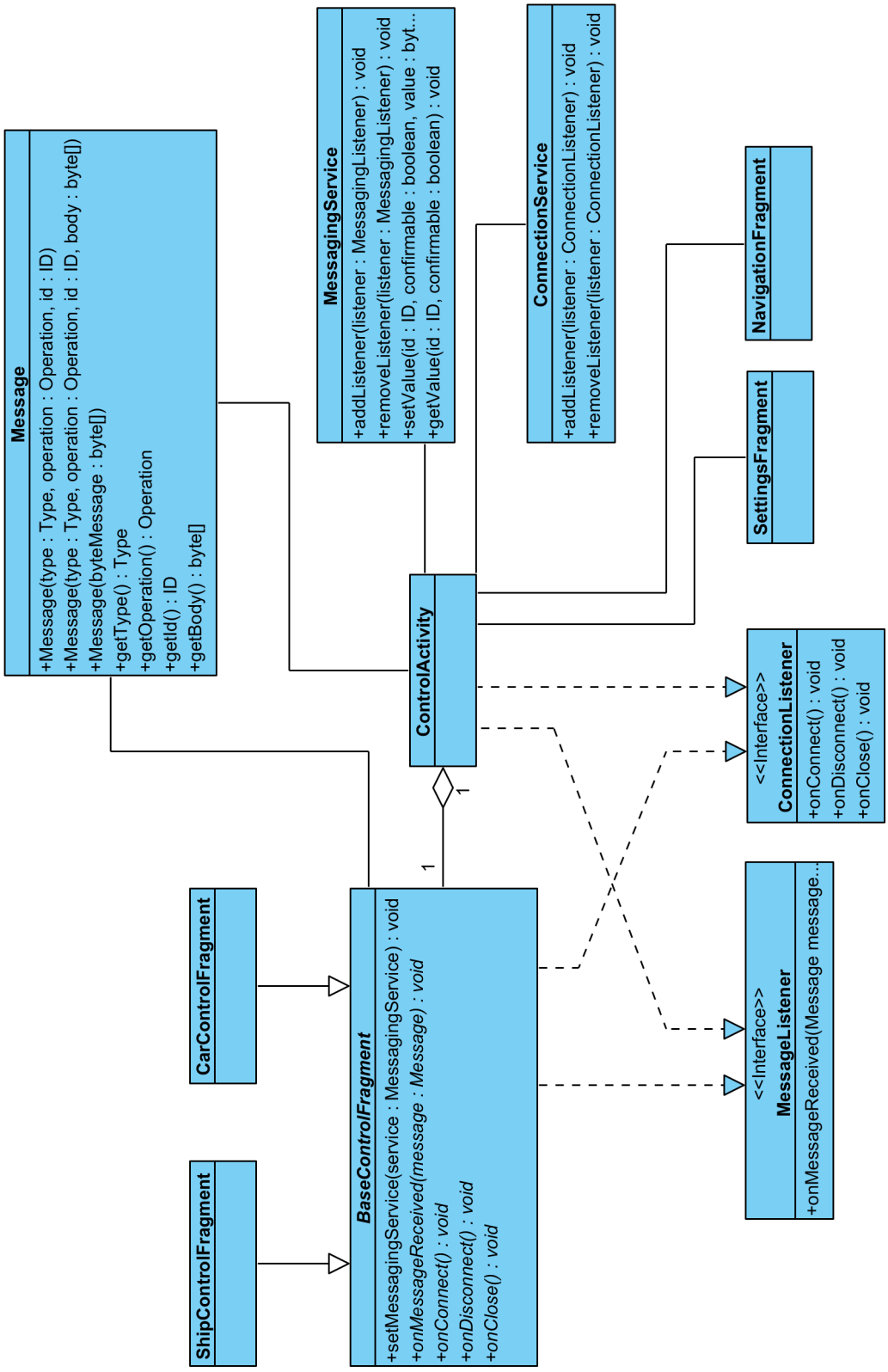

Abbildung 4.23: Klassendiagramm ControlActivity - UI-Seite

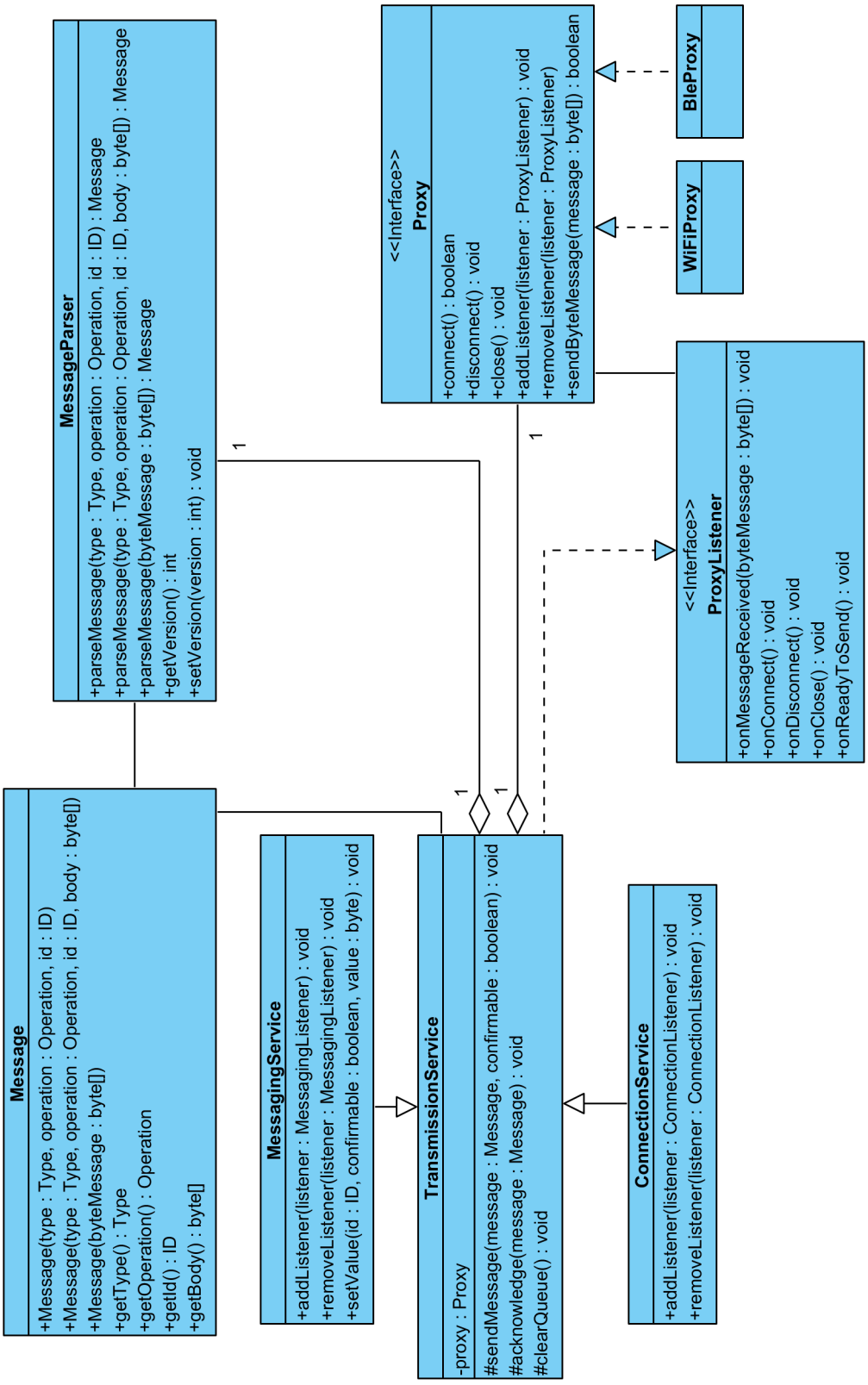

Abbildung 4.24: Klassendiagramm ControlActivity - Proxy-Seite

4.4.3 Klassenverhalten

In Abb. 4.25 wird der Vorgang beim Scannen nach Gräten dargestellt. Der Scanvorgang wird durch den Benutzer durch den zugehörigen Button gestartet. Über den ScanningService wird dies an den Proxy weitergeleitet. Sobald der Scan durchgelaufen oder vom Benutzer abgebrochen wird, teilt der ScanningService dies dem Proxy mit. Anschließend werden alle gefundenen Geräte vom Proxy abgefragt und in der Activity dargestellt.

In Abb. 4.26 wird beispielhaft die Kommunikation zwischen der App und dem RC-Modell beim Übertragen eine Confirmable-Nachricht dargestellt. In diesem Beispiel fragt die App einen Wert des RC-Modells ab. Die Activity ruft die entsprechende Methode des MessagingServices auf. Dieser erstellt ein Message-Objekt, welches einer Nachricht nach dem designeten Protokoll enstpricht, und leitet dieser an den Proxy weiter. Der Proxy überträgt die Nachricht anschließend an das RC-Modell. Das RC-Modell schickt nach Bearbeitung der Nachricht eine Antwort mit dem gewünschten Wert.

Abb. 4.27 stellt das Verhalten bei verloren gegangenen Nachrichten dar. Der Ablauf zum Übertragen einer Nachricht verläuft analog zur Abb. 4.26 mit dem Unterschied, dass diese Nachricht erneut gesendet wird, da sie vom RC-Modell nicht quittiert wurde. Hierbei wird die Nachricht mehrfach in immer kürzeren Zeitabständen erneut versendet bis die Nachricht quittiert wird oder die maximale Anzahl an erneuten Übertragungsversuchen erreicht wird.

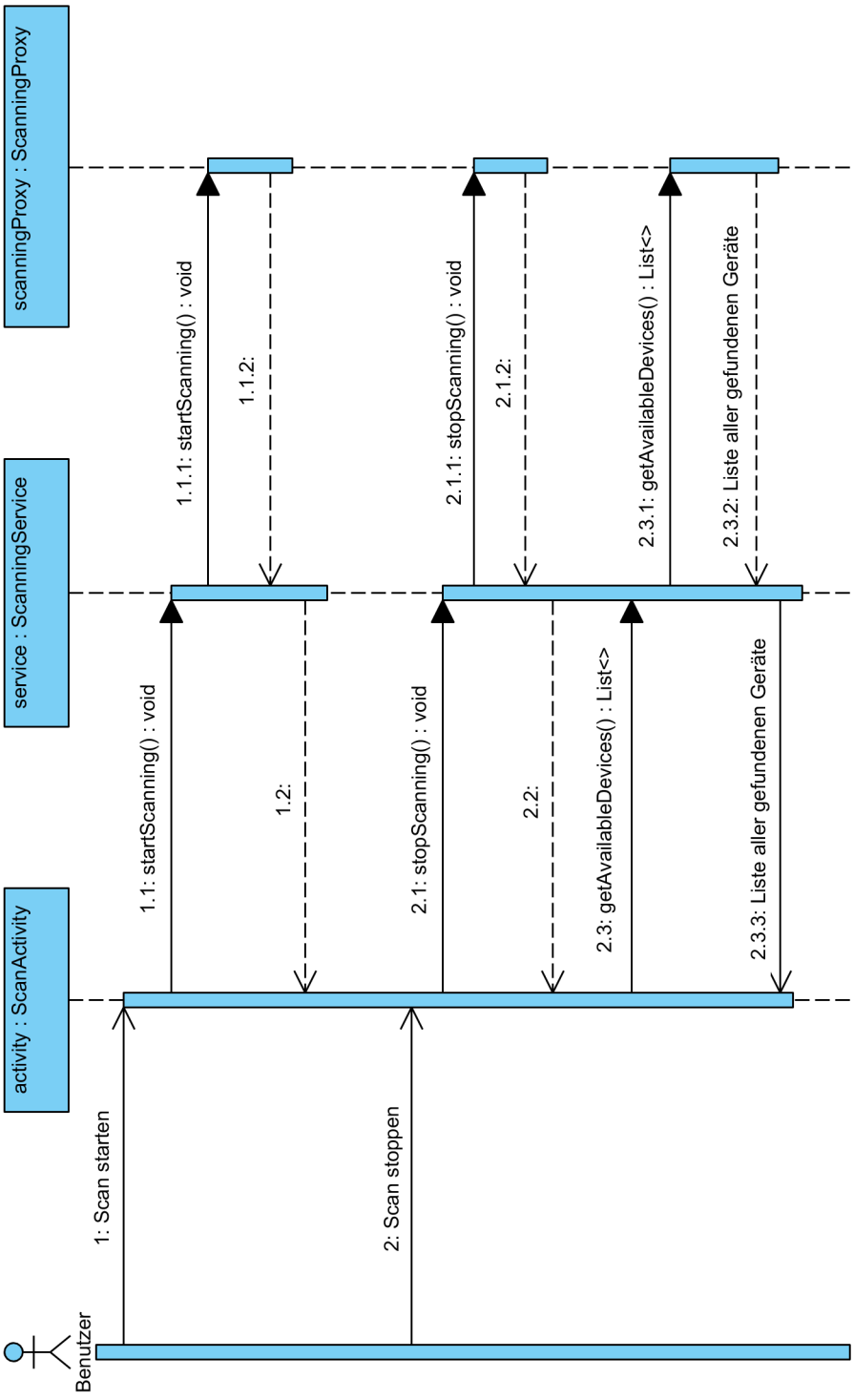

Abbildung 4.25: Sequenzdiagramm Nach Geräten Scannen

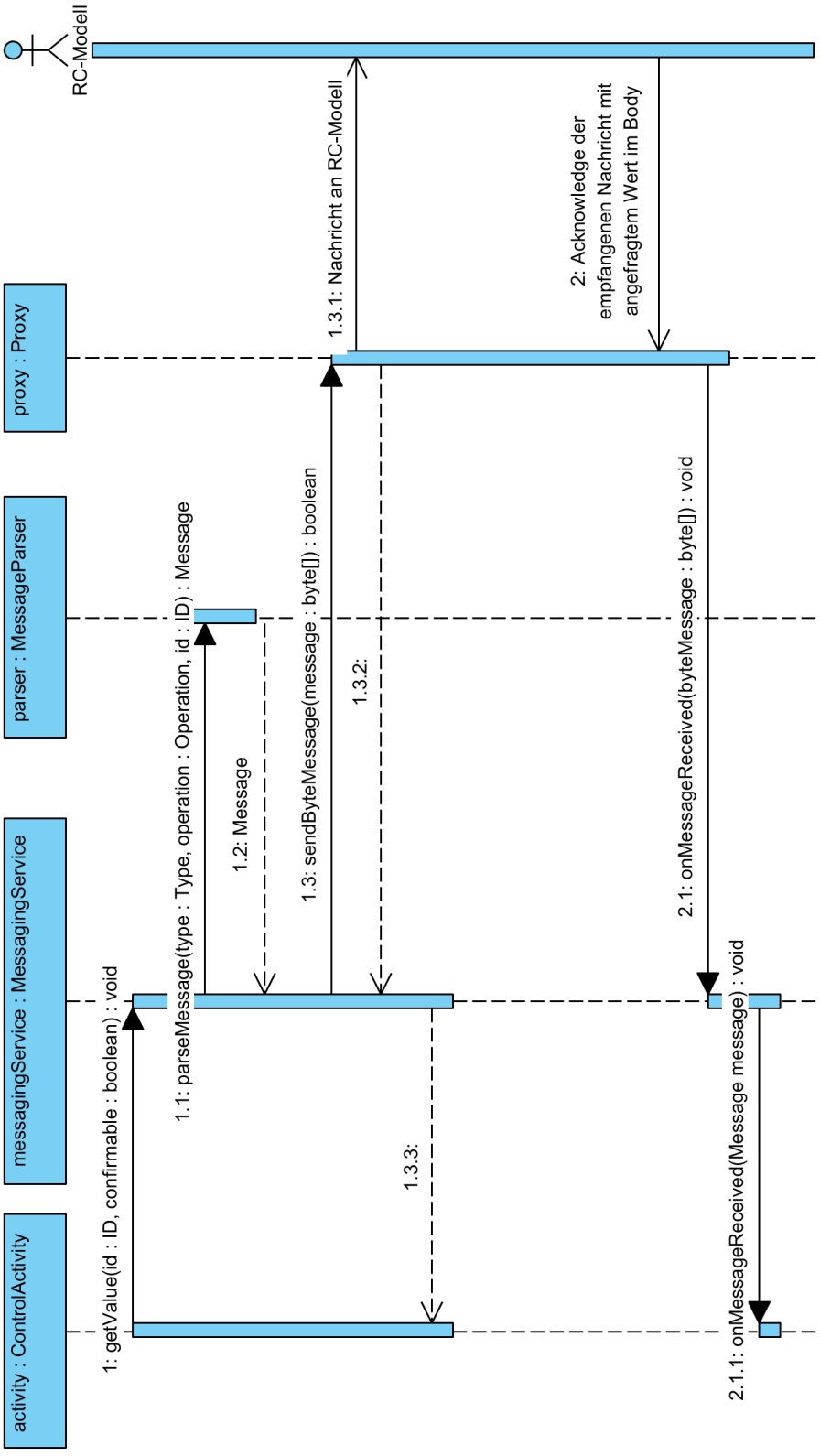

Abbildung 4.26: Sequenzdiagramm Nachricht übertragen

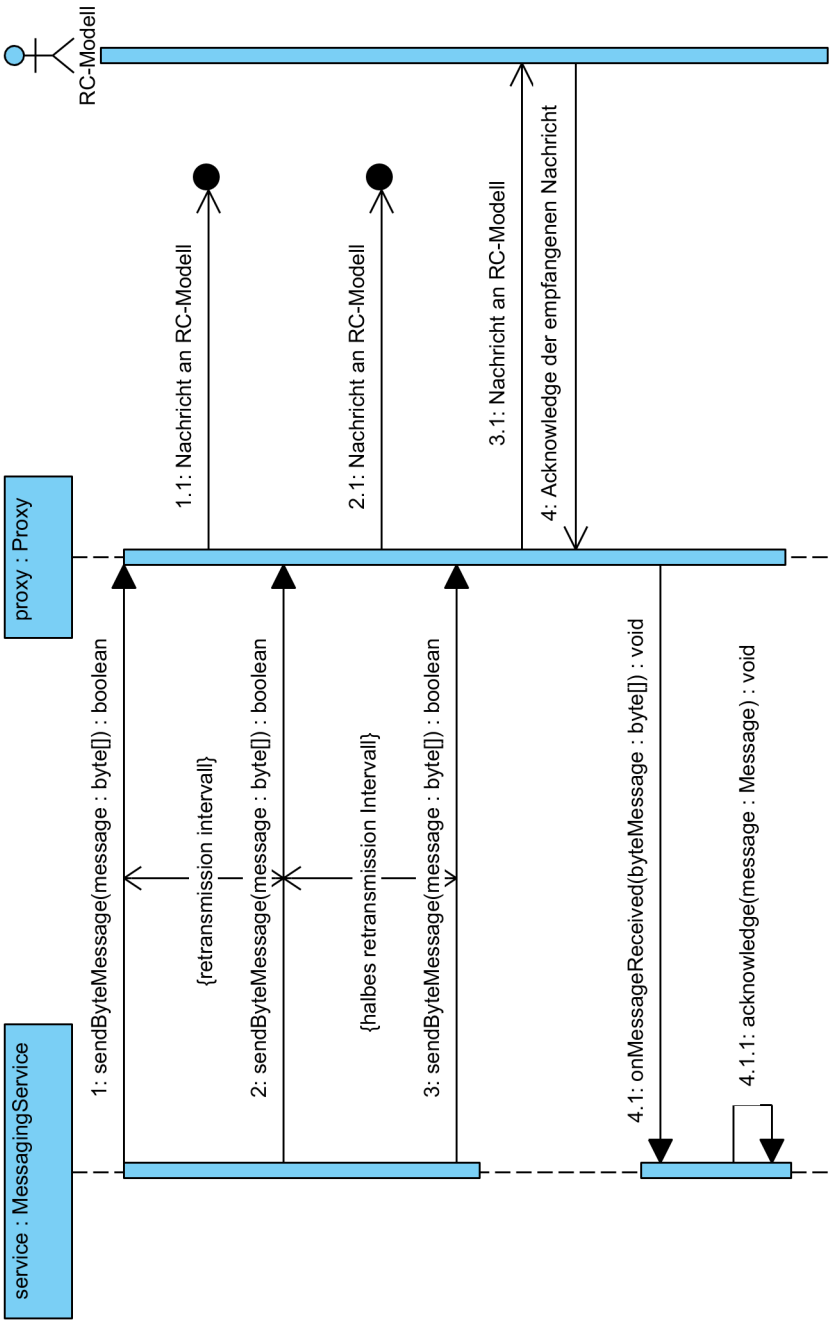

Abbildung 4.27: Sequenzdiagramm erneutes Übertragen von Nachrichten

4.5 Design der Software des RC-Modells

Es ist kein festes Design der Software des RC-Modells notwendig, da die RC-Modelle bereits mit aufgespielter Software, die von vorherigen Projekten entwickelt wurden, erhalten wurden. Diese Software muss an die Kommunikation mit dem entwickelten Protokoll angepasst werden. Dies wird im Kapitel 5 behandelt.

4.6 Festellung des Verbindungs- oder Kontrollverlustes

Zur Feststellung des Verbindungsverlustes wird von der App in einem festgelegten Zeitintervall eine Nachricht zum Überprüfen der Verbindung an das RC-Modell übertragen. Dieses wird vom RC-Modell nach Empfang quittiert. Die App stellt den Verbindungsverlust dadurch fest, dass nach einer festgelegten Zeitspanne die übertragene Nachricht nicht quittiert wurde. Das RC-Modell stellt den Verbindungsverlust fest, wenn für eine festgelegte Zeitspanne keine Nachricht zum Prüfen der Verbindung empfangen wurde.

5 Implementierung

In diesem Kapitel wird die Implementierung des Konzeptes und des Designs vorgestellt. Nach einer Auflistung der verwendeten Hardware und Software, einschließlich der verwendeten Version, wird die Implementierung durch exemplarische Code-Beispiele dargestellt.

5.1 App

Die Implementierung der App erfolgt über Android Studio mit Java als Programmiersprache. Java wurde gewählt, da diese Programmiersprache weit verbreitet und vielen Studenten während des Studiums als erste objektorientierte Sprache beigebracht wird. Dies erleichtert auf dieser Thesis aufbauenden Projekten die Einarbeitung in den implementierten Code. Java ist außerdem noch immer der State-of-the-Art für die Entwicklung von Android-Apps. Als mobile Gerät wird ein Samsung Galaxy S4 mini verwendet (siehe Abb. 5.1).

Abbildung 5.1: Samsung Galaxy S4 mini
Bildquelle: https://www.samsung.com/us/mobile/phones/galaxy-s/samsung-galaxy-s4-mini-verizon-sch-i435zkavzw/

Dieses Modell hat das API-Level 19, was der Android-Version 4.4.4 entspricht. Da Bibliotheken (Libraries) und Funktionen aus höheren APIs nicht verwendet werden können, sind einige Teile des entwickelten Codes auf App Seite veraltet bzw. deprecated. Diese Codestellen sind mit entsprechenden ToDo-Tags markiert.

Für das Zeitintervall zum Prüfen der Verbindung wurden 500 ms gewählt. Daraus ergeben sich 250 ms für die Zeit, nach welcher eine nicht quittierte Nachricht zum ersten Mal erneut versandt wird.

Als erste Kommunikationstechnologie wurde ein Proxy für Bluetooth Low Energy implementiert. Bei der Auswahl der Technologie in der TechnologySelectionApp stehen sowohl BLE als auch WiFi zur Auswahl. WiFi wurde in dieser Thesis jedoch nicht umgesetzt.

5.2 Exemplarische Code-Beispiele der App

5.2.1 Selektieren von BLE als Kommunikationstechnologie

Wird in der TechnologySelectionActivity BLE als Kommunikationstechnologie ausgewählt, so wird zuerst geprüft ob Bluetooth auf dem Gerät aktiviert ist. Ist dies nicht der Fall so wird der

Nutzer aufgefordert Bluetooth zu aktivieren. Ist Bluetooth bereits aktiviert wechselt die App zur ScanningActivity.

```
1  public void onSelectBle(View view) {
2          if (getPackageManager().hasSystemFeature(PackageManager.
               FEATURE_BLUETOOTH_LE)) {
3              BluetoothManager btManager = (BluetoothManager) getSystemService(
                   Context.BLUETOOTH_SERVICE);
4              BluetoothAdapter btAdapter = btManager.getAdapter();
5
6              if (btAdapter.isEnabled()) {
7                  startScanActivity(ConnectionTechnology.BLE);
8              } else {
9                  Intent enableBtIntent = new Intent(BluetoothAdapter.
                       ACTION_REQUEST_ENABLE);
10                 startActivityForResult(enableBtIntent, ENABLE_BT_RESULT_CODE);
11             }
12         } else {
13             Toast.makeText(this, R.string.ble_not_supported, Toast.LENGTH_SHORT).
                   show();
14         }
15     }
16
17     public void onSelectWifi(View view) {
18         Toast.makeText(this, "WiFi not implemented", Toast.LENGTH_SHORT).show();
19     }
```

Listing 5.1: Selektieren der Kommunikationstechnologie

5.2.2 Nachrichtenübertragung im MessagingService

Für das Setzen und Abfragen von Werten bietet der MessagingService entsprechende Methoden an. Mithilfe des MessageParsers wird ein Message Objekt erstellt und dieses über den Proxy an das RC-Modell übertragen. Für das Setzen von Werten wird ein Byte als Parameter an die Methode übergeben, welcher den Body der Nachricht darstellt.

```
1  public void getValue(Message.ID id, boolean confirmable) {
2          Message.Type type = (confirmable) ? Message.Type.CONFIRMABLE : Message.
               Type.NONCONFIRMABLE;
3          Message message = parser.parseMessage(type, Message.Operation.GET, id);
4          sendMessage(message, confirmable);
5      }
6
7      public void setValue(Message.ID id, boolean confirmable, byte input) {
8          Message.Type type = (confirmable) ? Message.Type.CONFIRMABLE : Message.
               Type.NONCONFIRMABLE;
```

```
9           byte[] value = ByteBuffer.allocate(1).put(input).array();
10          Message message = parser.parseMessage(type, Message.Operation.SET, id,
                value);
11          sendMessage(message, confirmable);
12      }
```

Listing 5.2: Nachrichtenübertragung im MessagingService

5.2.3 Prüfen der Verbindung im ConnectionService

Für das Überprüfen der Verbindung (um ein Verbindungsverlust schnell zu bemerken) sendet der ConnectionService beim Verbinden mit dem RC-Modell eine Nachricht zum Überpüfen der Verbindung. Das RC-Modell quittiert durch eine entsprechende Nachricht (siehe Kapitel 4).

```
1   private void startConnectionChecking() {
2           final Message message = parser.parseMessage(Message.Type.CONFIRMABLE,
3                   Message.Operation.STATUS, Message.ID.CONNECTION);
4
5           handler.post(new Runnable() {
6               @Override
7               public void run() {
8                   sendMessage(message, true);
9                   handler.postDelayed(this, CONNECTION_CHECK_INTERVALL);
10              }
11          });
12      }
13
14      private void stopConnectionChecking() {
15          handler.removeCallbacksAndMessages(null);
16      }
```

Listing 5.3: Prüfen der Verbindung im ConnectionService

5.2.4 Erneutes Übertragen von Nachrichten

Die Implementierung für das erneute Versenden von Nachrichten erfolgt in der abstrakten Klasse TransmissionService:

```
1   public abstract class TransmissionService extends Service {
2       protected Proxy proxy = null;
3       protected Handler handler = new Handler();
4       private List<TransmissionHandler> messageQueue = new ArrayList<>();
```

Listing 5.4: Klassendefinition TransmissionService

Der TransmissionService besitzt mehrere Attribute:

Das Attribut **proxy** wird nach dem binden des Services durch die ControlActivity gesetzt. Hinter dem Interface steht das Objekt der Proxy-Klasse, die der ausgewählten Kommunikationstechnologie entspricht. Ein Handler ist eine von Android bereitgestellte Klasse, mit der Code asynchron ausgeführt werden kann. Die Liste **messageQueue** hält Objekte vom Typ TransmissionHandler. Diese Klasse führt die erneute Übertragung einer zu bestätigenden Nachricht durch, falls diese nicht in festgelegter Zeit quittiert wird (siehe Listing 5.5).

Für die Zeit, nach welcher eine Nachricht erneut versendet wird, wurden 250 ms gewählt. Die Anzahl der maximalen Übertragungsversuche beträgt fünf.

```
1  private class TransmissionHandler {
2          private final static int MAX_TRANSMISSIONS = 5;
3          private final static int TRANSMISSION_TIME = 250;
4          private final Message message;
5          private int transmissionCount = 0;
6          private final Runnable runnable = new Runnable() {
7              @Override
8              public void run() {
9                  if (transmissionCount <= MAX_TRANSMISSIONS) {
10                     proxy.sendByteMessage(message.getByteMessage());
11                     transmissionCount++;
12                     handler.postDelayed(this, TRANSMISSION_TIME /
                           transmissionCount);
13                 } else {
14                     removeFromQueue(TransmissionHandler.this);
15                 }
16             }
17         };
18
19         private TransmissionHandler(Message message) {
20             this.message = message;
21         }
22
23         private void startTransmission() {
24             handler.post(runnable);
25         }
26
27         private boolean acknowledge(Message message) {
28             boolean isAcknowledge = message.getType().equals(Message.Type.
                   ACKNOWLEDGE);
29             boolean sameOperation = this.message.getOperation().equals(message.
                   getOperation());
```

```
30              boolean sameId = this.message.getId().equals(message.getId());
31
32              if (isAcknowledge && sameOperation && sameId) {
33                  handler.removeCallbacks(runnable);
34                  return true;
35              }
36              return false;
37          }
38
39          private void cancelTransmission() {
40              handler.removeCallbacks(runnable);
41          }
42      }
```

Listing 5.5: Klassendefinition TransmissionHandler

Listing 5.6 stellt die wichtigsten Methoden des TransmissionService dar. Durch diese Methoden können Nachrichten versandt, als auch quittiert und somit aus der Queue genommen werden.

```
1   protected void clearQueue() {
2           for (TransmissionHandler transmissionHandler : messageQueue) {
3               transmissionHandler.cancelTransmission();
4           }
5           messageQueue.clear();
6       }
7
8       protected void sendMessage(Message message, boolean confirmable) {
9           if (confirmable) {
10              TransmissionHandler transmissionHandler = new TransmissionHandler(
                    message);
11              messageQueue.add(transmissionHandler);
12              transmissionHandler.startTransmission();
13          } else {
14              proxy.sendByteMessage(message.getByteMessage());
15          }
16      }
17
18      protected void acknowledge(Message message) {
19          for (Iterator<TransmissionHandler> iterator = messageQueue.iterator();
                iterator.hasNext();) {
20              TransmissionHandler transmissionHandler = iterator.next();
21              if (transmissionHandler.acknowledge(message)) {
22                  iterator.remove();
23              }
24          }
25      }
```

Listing 5.6: Methoden der Klasse TransmissionHandler

5.3 RC-Modelle und zugehörige Software

Als RC-Modelle werden zwei Automodelle verwendet. Der Code zur Steuerung der RC-Modelle wurde bereits von vorherigen Projekten entwickelt, wurde jedoch an die Kommunikation über das entwickelte Protokoll angepasst. Die verwendeten RC-Automodelle sind in Abb. 5.2 und Abb. 5.3 zu sehen. Der Code der RC-Modelle wird mit der Entwicklungsumgebung Arduino Studio 1.8.5 entwickelt. Die Programmiersprache ist C und der Code wurde mithilfe der Literatur [12] entwickelt. Jedes RC-Modell hat einen Mikrocontroller vom Typ Arduino Nano. Dieser hat zwei Funktionen vorgegeben: **setup()** und **loop()** [13]. Dabei wird die Funktion setup() beim Starten des Mikrocontrollers durchlaufen. Die Funktion loop() wird anschließend als unendliche Schleife durchlaufen. [15] [14]

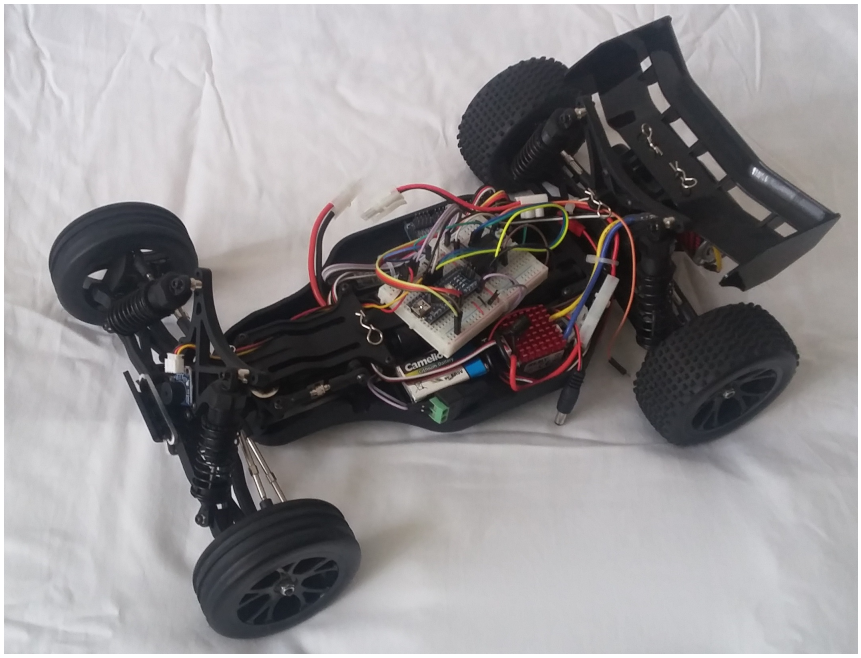

Abbildung 5.2: RC-Modell RC-Buggy

Abbildung 5.3: RC-Modell DartTruggy

5.3.1 Nachrichtenverarbeitung

Für das Format des entwickelten Protokolls dient das Struct **Message**. In dem Struct werden alle Felder der Nachricht gespeichert. Zum Dekodieren des Headers dient die Funktion **decodeHeader()**, während die Body-Bytes nach dem Dekodieren des Header ausgelesen werden (siehe Listing 5.7).

Die Nachricht wird anschließend über mehrere Switch-Anweisungen ausgewertet (siehe Listing 5.8).

```
1  struct message {
2    byte type;
3    byte operation;
4    byte id;
5    byte bodylength;
6    byte *value;
7  };
8  typedef struct message Message;
9
10 void decodeHeader(Message *m, byte *byteHeader) {
11   m->type = (byteHeader[0] & 0xF0) >> 4;
12   m->operation = byteHeader[0] & 0x0F;
13   m->id = byteHeader[1];
```

```
14   m->bodylength = byteHeader[2];
15 }
```

Listing 5.7: Message Struct

```
1  bool processMessage(Message *m) {
2    switch(m->operation) {
3      case STATUS:
4        return processStatus(m);
5      case GET:
6        return processGet(m);
7      case SET:
8        return processSet(m);
9    }
10   return false;
11 }
```

Listing 5.8: Switch-Anweisung zum Auswerten einer Nachricht

5.3.2 Quittieren von Nachrichten

Das Quittieren von Nachrichten erfolgt über die Funktion **acknowledgeMessage**. Hierbei wird die ID und Operation der Nachricht nicht verändert.

```
1  void acknowledgeMessage(Message *m) {
2    m->type = ACKNOWLEDGE;
3    sendMessage(m);
4  }
5
6  void acknowledgeMessage(Message *m, byte b) {
7    m->type = ACKNOWLEDGE;
8    m->bodylength = 1;
9    m->value = &b;
10   sendMessage(m);
11 }
```

Listing 5.9: Quittieren von Nachrichten

5.3.3 Versenden von Nachrichten

Das Versenden von Nachrichten erfolgt über die Funktion **sendMessage()**. In der Funktion werden die Daten des Structs Message in Byteform kodiert und über das angeschlossene Bluetooth-Modul versendet.

```
1  void sendMessage(Message *m) {
2    int mLen = HEADER_SIZE + m->bodylength;
3    byte *em = (byte*) calloc(mLen, sizeof(byte));
4    em[0] = (m->type << 4);
5    em[0] |= m->operation;
6    em[1] = m->id;
7    em[2] = m->bodylength;
8    for(int i = 0; i < m->bodylength; i++) {
9      em[HEADER_SIZE + i] = m->value[i];
10   }
11   softwareSerial.write(em, mLen);
12   free(em);
13 }
```

Listing 5.10: Versenden von Nachrichten

5.3.4 Feststellung des Verbindungsverlustes

Zum Feststellen eines Verbindungsverlustes zum steuernden Mobile-Gerät wird der Zeitabstand zwischen eingehenden Nachrichten zum Prüfen der Verbindung gemessen. Dabei wird die aktuelle Zeit beim Eintreffen einer solchen Nachricht gespeichert. Beim kontinuierlichen Durchlaufen der Funktion **loop()** wird die in dem Moment gemessene Zeit mit der Zeit verglichen, wann die letzte Nachricht zum Prüfen der Verbindung erhalten wurde. Überschreitet der Zeitabstand einen Schwellenwert, so wird im RC-Modell von einem Verbindungsverlust ausgegangen und das Automodell kommt zum Stillstand.

```
1 bool checkConnection() {
2   unsigned long endTime = millis();
3   if (endTime - startTime >= CONNECTION_CHECK_TIMEOUT) {
4     return false;
5   }
6   return true;
7 }
```

Listing 5.11: Feststellung des Verbindungsverlustes

6 Test

In diesem Kapitel wird beschrieben, wie die Implementation getestet wurde und es werden die Ergebnisse dargestellt und diskutiert. Anschließend werden aufgetretene Probleme der Implementierung beschrieben

6.1 Test mit Arduino Uno

Bei diesem Test wurde geprüft, ob Nachrichten über die App korrekt übertragen und im korrekten Format empfangen werden. Es wird außerdem geprüft, ob jeweils ein Verbindungsverlust und Kontrollverlust von der Software des RC-Modells erkannt werden.

Für das Testen wird das Verhalten eines RC-Modells mit einem Mikrocontroller vom Typ *Arduino Uno* getestet. Dafür wurde auf den Arudino Uno der Code der RC-Modelle aufgespielt. Die Zeilen des Codes, die Pins an den RC-Modellen ansprechen wurden dabei auskommentiert. Der Mikrocontroller wurde an den Entwicklungsrechner angeschlossen. Die eingehenden und ausgehenden Nachrichten des Mikrocontrollers wurden auf der Konsole der Entwicklungsumgebung Arduino Studio auf dem Entwicklungsrechner ausgegeben. Es ergaben sich folgende Ergebnisse:

- Auf dem Mikrocontroller wurde erfolgreich eine Abfrage der Version des Protokolls, auf welchem der Mikrocontroller kommunizieren kann, beim Wechseln in die ControlActivity der App empfangen. Der Mikrocontroller antwortete korrekt mit einer Nachricht mit der Version im Body
- Es wurde erfolgreich eine Abfrage des Modelltypes empfangen, die beim Wechsel in die ControlActivity gesendet wird. Der Mikrocontroller antwortete korrekt mit einer Nachricht mit dem Modeltyp (hier Automodell) im Body
- Im Intervall von einer halben Sekunde wurde vom Mikrocontroller eine Nachricht zum Überprüfen der Verbindung empfangen. Der Mikrocontroller quittierte die Nachricht korrekt
- Auf Nachrichten, die keine valide Anweisung für den Mikrocontroller sind, antwortete der Mikrocontroller korrekt mit einer Fehlernachricht

- Wurde die App auf dem Mobile-Gerät in den Hintergrund gerückt, so versendete diese erfolgreich eine Nachricht an den Mikrocontroller über den Kontrollverlust
- Ist die App auf dem Mobile-Gerät wieder in den Vordergrund gerückt, wurde erfolgreich eine Nachricht an den Mikrocontroller gesendet, die diesen über die erneute Kontrolle informierte
- Der Verbindungsverlust wurde vom Mikrocontroller bemerkt, als im Mobile-Gerät die Bluetooth Funktion deaktiviert wurde

6.2 Test des Systems mit RC-Automodellen

Zum Testen des Systems auf dem RC-Buggy und dem DartTruggy wurden der Kontrollverlust, Verbindungsverlust und alle erarbeiteten Anwendungsfälle getestet:

Anwendungsfall	RC-Buggy	DartTruggy
App starten	Erfolgreich	
Technologie auswählen	Erfolgreich	
Nach Geräten scannen	Erfolgreich	
Gerät auswählen	Erfolgreich	Erfolgreich
Motor an/-ausschalten	Erfolgreich	Nicht implementiert
Bewegungsrichtung setzen	Erfolgreich	Nicht implementiert
Beschleunigen	Erfolgreich	Nicht implementiert
Bremsen	Erfolgreich	Nicht implementiert
Auslenkung ändern	Erfolgreich	Nicht implementiert
Licht an/aus	Erfolgreich	Nicht implementiert
Hupen	Teilweise Erfolgreich	Nicht implementiert
Kontrollverlust bemerken	Erfolgreich	Nicht implementiert
Verbindungsverlust bemerken	Erfolgreich	Nicht implementiert

Tabelle 6.1: Testergebnisse an RC-Automodellen

Die System wurde erfolgreich mit dem RC-Buggy getestet. Da es einen Hardwarefehler bei der Hupe des RC-Buggys gab, wurde diese beim Testen bewusst nicht ausgelöst. Auf dem seriellen Monitor war jedoch zu sehen, dass der Befehl zum Hupen erfolgreich empfangen wurde. Zum Zeitpunkt der Abgabe konnte die Software für das DartTruggy nicht mehr angepasst werden. Dementsprechend konnten keine Tests mit dem DartTruggy durchgeführt werden.

7 Zusammenfassung

In diesem Kapitel wird der Inhalt der Thesis kurz zusammengefasst. Es wird erläutert, welche Anforderungen umgesetzt werden konnten und welche nicht. Anschließend wird eine Aussicht darauf gegeben, was noch umgesetzt werden könnte und es werden Vorschläge gemacht, wie diese umgesetzt werden kann.

7.1 Auswertung

Zu Anfang dieser Arbeit wurde dargestellt, welche Probleme eine nicht generische Kommunikation bei RC-Modellen darstellt. Es wurde recherchiert, welche Kommunikationsprotokolle aktueller Standard bei der Machine-to-Machine Kommunikation ist und diese miteinander verglichen. Dabei stach das Constraint Application Protocol, kurz CoAP, als besonders geeignet für die ermittelten Anforderungen heraus. Mit Orientierung an CoAP wurde ein leichtgewichtiges Kommunikationsprotokoll entworfen. Es wurde eine Android-App entwickelt, welche über das entwickelte Kommunikationsprotokoll RC-Modelle, die über dasselbe Protokoll kommunizieren können, steuern kann. Aus zwei vorangegangenen Bachelorprojekten wurde die Software der in diesen Projekten jeweils verwendeten RC-Automodelle, für die Kommunikation über das entwickelte Protokoll angepasst.

Leider konnten im Rahmen dieser Arbeit nicht alle Anforderungen erfüllt werden. Es erfolgt eine Aufzählung von Anforderungen die nicht oder nur teilweise erfüllt wurde:

- Die App sendet beim verbinden mit dem RC-Modell eine Anfrage nach dem Typ des Modells. Es wird zurzeit aber nur die GUI zur Steuerung eines RC-Automodells geladen, unabhängig vom eigentlichen Typ des Modells
- Die App erkennt nicht, welche Funktionalitäten (z.B. Bremse, Hupe) ein RC-Modell bietet. Dadurch kann der Benutzer mit GUI-Elementen interagieren, die auf dem RC-Modell keine Funktion auslösen
- De App bemerkt einen Verbindungsverlust zum RC-Modell, bietet dem Benutzer jedoch nicht die Möglichkeit eine erneute Verbindung aufzubauen

- Wird eine übertragene Nachricht vom RC-Modell nicht verstanden, so wird die App darüber informiert. Der Benutzer wird jedoch nicht informiert

7.2 Aussicht

Fehlercode im Body einer Fehlernachricht

Im Falle einer fehlerhaften oder nicht verstandenen Nachricht quittiert das RC-Modell diese mit ERROR im Feld Type des Headers. Unterschiedliche Gründe für diesen Fehler könnten mit einem Fehlercode im Body unterschieden werden.

Erkennen von duplizierten Nachrichten

CoAP erkennt duplizierte Nachrichten über die MessageID. Durch diese Nummer ist jede Nachricht eindeutig identifizierbar. Dies ist nicht nur hilfreich beim Erkennen von duplizierten Nachrichten. Es würde außerdem die Quittierung von Nachrichten gleicher Art verhindern. Sollen z.B. zwei Nachrichten mit dem gleichen Feldern im Header, d.h. Type, Operation und ID, versandt werden, so könnte es passieren, dass die zweite Nachricht mit quittiert wird, obwohl diese noch nicht versandt wurde.

Erkennen von validen Geräten

Die App verbindet sich mit dem ausgewählten Gerät, selbst wenn es kein valides RC-Modell ist, dass über das entwickelte Kommunikationsprotokoll kommuniziert. Im Falle eines Gerätes, welches nicht über das entwickelte Protokoll kommuniziert, sollte der Benutzer informiert werden und die App wieder auf die ScanningActivity wechseln.

Werte über physikalische Größen setzen

Werte werden in dieser Version des Protokolls nur über einen boolschen Wert oder einer Prozentangabe gesetzt. Es sollte auch möglich sein, Werte in physikalischen Größen, wie z.B. Geschwindigkeit in Kilometern pro Stunde oder Auslenkung in Grad, zu setzen.

Sicherstellung der Nachrichtenübertragung auf Modell-Seite

Die Sicherstellung der Nachrichtenübertragung durch das erneute Übertragen, falls eine Nachricht nicht quittiert wird, sollte auch auf der Modell-Seite implementiert werden.

Statusabfrage beim Verbinden

Beim Verbinden sollte die App das RC-Modell nach dem Status ihrer Werte fragen, d.h. ob der Motor bereits an ist, und die erzeugte GUI dementsprechend anpassen.

Intervall zur Verbindungsprüfung einstellbar

Im Tab „Settings“in der ControlActivity sollte der Benutzer das Zeitintervall, in welchem die Verbindung zum RC-Modell geprüft wird, einstellen können. Dabei wäre es sinnvoll, nach Änderung des Intervalls, das RC-Modell über diese Änderung zu benachrichtigen.

RegressionsTests

Es sollten Regressionstests entwickelt werden, die den Code der App auf Korrektheit prüft. Hierbei wären Regressionstests insbesondere für die Klassen **MessageParser** und **Message** sinnvoll.

Funktionalitäten des RC-Modells abfragen

Die App sollte beim Verbinden die Funktionalitäten des RC-Modells abfragen. Sollte das Modell eine Funktionalität, die in der entsprechenden GUI selektierbar ist (z.B. Hupe), nicht besitzen, so sollte die App die zugehörigen GUI-Elemente deaktivieren.

Realistischere Bremsstärke

Das RC-Modell sollte nicht immer eine Vollbremsung beim betätigen der Bremse ausführen. Wie bei einem echten Fahrzeug ist eine Implementierung mit variabler Bremsstärke sinnvoll.

Broadcasts statt Listener

Android bietet eine Broadcast-Funktionalität an. Durch diese können Komponenten wie Services oder Activities andere Komponenten benachrichtigen. Die Broadcast-Funktionalität sollte daher die Listener der App ersetzen.

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

Literaturverzeichnis

[1] : *Android (Betriebssystem)*. – URL https://de.wikipedia.org/wiki/Android_(Betriebssystem)

[2] : *AndroidDeveloper*. – URL https://developer.android.com

[3] : *AndroidDeveloper Bluetooth Low Energy*. – URL https://developer.android.com/guide/topics/connectivity/bluetooth-le

[4] : *Constrained Application Protocol*. – URL https://de.wikipedia.org/wiki/Constrained_Application_Protocol

[5] : *MQTT Version 3.1.1 - OASIS Standard*. – URL https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

[6] : *RFC 4347 - Datagram Transport Layer Security*. – URL https://tools.ietf.org/html/rfc4347

[7] : *RFC 7252 - The Constraint Application Protocol (CoAP)*. – URL https://tools.ietf.org/html/rfc7252

[8] BÜCHI, Roland: *2,4-GHz-Fernsteuerungen: Grundlagen und Praxis*. VTH Neue Medien GmbH, 2013. – ISBN 978-3-8818-0449-3

[9] CHRISTOPH KECHER, Alexander S.: *UML 2.5 - Das Umfassende Handbuch*. Rheinwerk Computing, 2015. – ISBN 978-3-8362-2977-7

[10] DAWN GRIFFITHS, David G.: *Head First Android Development*. O'Reilly Media, 2017. – ISBN 987-1-491-97405-6

[11] DEMBOWSKI, Klaus: *Computerschnittstellen und Bussysteme*. VDE Verlag, 2013. – ISBN 978-3-8007-3448-1

[12] ERLENKÖTTER, Helmut: *C: Programmieren von Anfang an*. Rowohlt Taschenbuch Verlag, 1999. – ISBN 978-3-4996-0074-6

[13] KAPPEL, Benjamin: *Arduino: Elektronik, Programmierung, Basteln*. Rheinwerk Computing, 2016. – ISBN 978-3-8362-3648-5

[14] SOMMER, Ulli: *Arduino Mikrocontroller-Programmierung mit Arduino/Freeduino*. Franzis Verlag GmbH, 2013. – ISBN 978-3-654-65147-9

[15] TIMMIS, Harold: *Arduino in der Praxis*. Franzis Verlag GmbH, 2012. – ISBN 978-3-654-65132-5

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 15. August 2018	
Ort, Datum	Unterschrift