HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

MASTERARBEIT

Herr
Tariq Mahfoud

App zur Ansteuerung von ATLAS-Fahrzeugen über Bluetooth

2020

Fakultät: Ingenieurwissenschaften

MASTERARBEIT

App zur Ansteuerung von ATLAS-Fahrzeugen über Bluetooth

Autor:
Herr Tariq Mahfoud

Studiengang:
Elektrotechnik

Seminargruppe:
ET15w1-M

Erstprüfer:
Prof. Dr.-Ing.Christian Schulz

Zweitprüfer:
Prof. Dr.-Ing. Jan Thomanek

Einreichung:
Mittweida, 08.10.2020

Faculty of Engineering Sciences

MASTER THESIS

App for controlling ATLAS-vehicles via Bluetooth

author:
Mr. Tariq Mahfoud

course of studies:
Electrical Engineering

seminar group:
ET15w1-M

first examiner:
Prof. Dr.-Ing.Christian Schulz

second examiner:
Prof. Dr.-Ing. Jan Thomanek

submission:
Mittweida, 08.10.2020

Bibliografische Angaben

Mahfoud, Tariq:

App zur Ansteuerung von ATLAS-Fahrzeugen über Bluetooth

App for controlling ATLAS-vehicles via Bluetooth

89 Seiten, Hochschule Mittweida, University of Applied Sciences, Fakultät Ingenieurwissenschaft, Masterarbeit, 2020

Abstract

Die vorliegende Arbeit befasst sich mit der Entwicklung einer Android App. Mit Hilfe dieser App soll das Fahrzeug im Rahmen des ATLAS-Projektes über Bluetooth Low Energy (BLE) angesteuert werden. Darüber hinaus ist das Ziel der Masterarbeit, die Konzeption, Entwurf und Entwicklung einer BLE-App. Dabei werden die verschiedenen Ebenen der Kommunikation Technik und die verwendeten Protokolle erläutert.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ADB	Android Debug Bridge
API	Application Programming Interface
Apk	Android Package
ASCII	American Standard Code für Information Interchange
ATLAS	Asynchroner Transport, Logistik und Automatisierungsmodus auf der Schiene
ATT	Attribute Protocol
BLE	Bluetooth Low Energy
CCCD	Client Characteristic Configuration Descriptor
GAP	Generic Access Profile
GATT	Generic Attribute Profile
GFSK	Gaussian Frequency Key Shiftings
GUI	Grafische Benutzeroberfläche
IPC	Interprozesskommunikation
IR	Infrarot
MVC	Model View Controller
PHY	Physikalischen Schicht
PWM	Pulsweitenmodulation
RSSI	Received Signal Strength Indication
RxD	Receive Data
SDK	Software Development Kit
SIG	Special Interest Group
Soc	Systems-of-Chip
TxD	Transmit Data
UART	Universal asynchronous Receiver Transmitter
UUID	Universal Unique Identifier
XML	Extensible Markup Language

Abbildungsverzeichnis

Tabellenverzeichnis

Vorwort

An dieser Stelle möchte ich denjenigen danken, die mich bei dieser Arbeit unterstützt haben.

Mein besonderer Dank gilt Prof. Dr.-Ing. Christian Schulz und Prof. Dr.-Ing. Jan Thomanek für die Beurteilung und Betreuung meiner Arbeit.

Ich möchte ebenso meinen Kollegen Herr Liu bei ‚ATLAS-Projekt' für die gute Zusammenarbeit danken.

Abschließend möchte ich meinen Eltern danken, die mich während meines Studiums immer motiviert und finanziell unterstützt haben.

Einleitung

Die vorliegende Arbeit ist innerhalb der Forschungsgruppe des ATLAS-Projekts an der Hochschule Mittweida entstanden. Diese Forschungsgruppe ist Teil der Professur Automatisierungstechnik. Es soll eine Datenübertragung per nanoNET ersetzt werden und ein paar Erweiterungen bzw. neue Funktionen hinzufügt werden. Smartphones und Tablets stellen eine sehr Leistungsfähige Hardware bereit und fast alle aktuellen Geräte verfügen über Bluetooth, daher kommt die Idee, warum sollte man also nicht ein Smartphone zum Steuern einsetzen?

Abbildung 1: alte ATLAS-nanoNet Fernbedienung

1.1 Motivation

In den letzten Jahren hat die drahtlose Kommunikation unseren Lebensstil verändert. Die Integration einer Kommunikation Technik wie Bluetooth in Mobiltelefone stieg enorm an. Sie ersetzte die Infrarot Kommunikation, die für die Datenübertragung eine Sichtverbindung erforderte. Dabei wurde Bluetooth Classic erfolgreich durch SIG in BLE umgewandelt und schnell übernommen. Da BLE eine kostengünstige und energieeffiziente Technologie ist, lässt sie sich leicht in Single Modul oder Smart-Geräte integrieren.

Android Anwendungs-Framework (Applikation-Framework) bietet einen Zugriff auf die Bluetooth-Funktionalität über die Android-Bluetooth-APIs. was ermöglicht Android Apps zu entwickeln. die Daten mit diesem Modul über BLE austauschen und sammeln zu können. Dadurch bietet sich die Möglichkeit eins fern gesteuerten Modells über App zu entwickeln. Solche fern gesteuerten Modelle finden in der heutigen Zeit immer mehr Beliebtheit aufgrund ihrer Mobilität und Energie Verbrauchs. Außerdem ist ihre Einsatz Bereich vielfältig.

In diesem Zusammenhang fokussieren wir uns auf die Entwicklung einer Android App um eine Daten Kommunikation über BLE zu erzielen. Diese App hat die Aufgabe das Fahrzeug im Rahmen des Atlas Projekts zu steuern und Daten aus den Positionssensoren zu sammeln.

1.2 Definition der Aufgabenstellung und Zielsetzung

Das Hauptziel unseres Projekts ist die Steuerung des ATLAS-Projekt Fahrzeug mittels Android-Applikation. Das Fahrzeug ist in der Regel eine elektromechanische Maschine. Sie besteht aus einem DC Motor die durch eine App und elektronische Programmierung (Controller) gesteuert wird. Die verwendete MCU M16C/62 hat keine integrierte BLE-Funktion. Aus diesem Grund soll das BLE-Modul über den UART mit dem MCU verbunden, so dass eine Kommunikation mit einem Smartphone oder Tablet möglich ist. Aufgrund der Tatsache, dass unser nRF52832 Modul nur BLE Version 4.2 unterstützt, müssen wir eine BLE Daten-Kommunikation herstellen. Hierbei gilt diese Kommunikation mit Hilfe von UART als Bidirektionale, was ermöglicht die Entwicklung einer ferngesteuerten Benutzerschnittstelle (GUI) zur Steuerung dieses Fahrzeugs und zum Empfang von Daten aus den Positionssensoren. Mit anderen Worten soll diese Android App in der Lage sein Motorsteuerungsbefehle und Kommandos an das Fahrzeugs senden sowie Fahrstraßen steuern. Außerdem sie soll die aktuellen Status von Balisen auslesen und anzeigen lassen.

In dieser Arbeit soll auch einen grundlegenden Einblick in die Bluetooth Low Energy und Android Entwicklung bieten. Dieses Projekt ist in verschiedene Kerngebiete aufgeteilt wie Elektronik, Embedded System Programmierung und Android Entwicklung usw., Zu meinen Aufgaben gehörten die Entwicklung und der Entwurf dieser Android App als Lösung für dieses ferngesteuertes Modell. (siehe Abbildung 2)

1.3 Funktionsprinzip

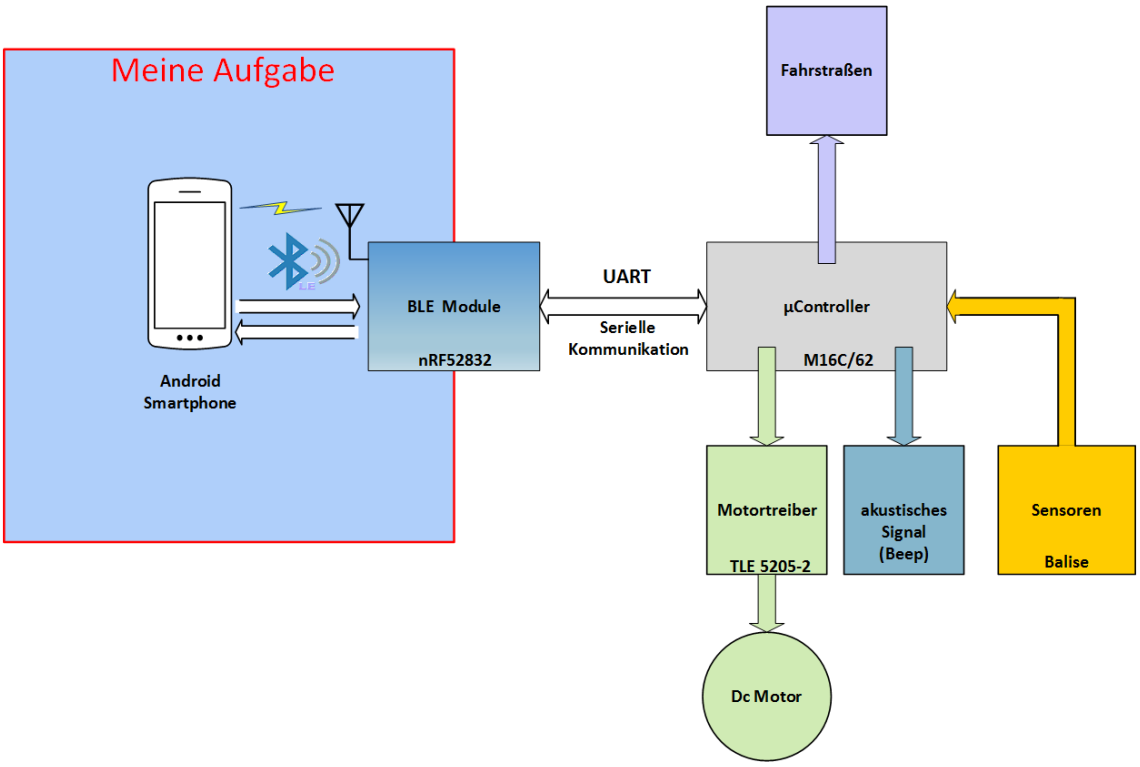


Abbildung 2: Blockschaltbild des ATLAS Projektes

Bevor ein solches System jedoch zum Einsatz kommen kann, sind viele Entwicklungsschritte notwendig, um für den Anwendungsfall geeignete Hard- und Software zu finden. Zudem müssen die Hard- und Softwarekomponenten miteinander interagieren.
Wir werden wie bereits erwähnt eine BLE Kommunikation als Schnittstelle zwischen Controller (M16c) und Android entwickeln. Der Controller ist über UART an das BLE - Modul (nRF) angeschlossen werden damit unterstützt er, dass die Daten in beide Richtungen weitergegeben werden. Der DC Motor wird über Treiber angeschlossen.
Das technische Ziel dieser Arbeit ist, eine Anwendung für Android mit der Programmiersprache Java zu entwickeln um das Blockschaltbild (Abb. 2) zu ermöglichen:
Wir bieten einen Überblick über das Fahrzeug, zuerst soll es sich durch unsere App vorwärts, und rückwärts bewegen, seine Geschwindigkeit kontrolliert werden und angehalten werden. Außerdem soll Mithilfe der App die Fahrstraßen gesteuert werden. Diese Kommandos die das BLE-Modul vom Android-Smartphone empfängt, werden als Eingabe nach einer seriellen Kommunikation in den Controller eingespeist. MCU verarbeitet die Daten und die Steuerung wirkt entsprechend je nachdem auf den DC-Motor des Fahrzeugs oder auf die Fahrstraßen. Des Weiteren soll mit Hilfe eines Positionssensors (Balise) die absolute Position bestimmt werden. Diese Informationen werden von dem Mikrocontroller verarbeitet und durch UART an das BLE Modul weitergegeben und anschließend von der App empfangen und dargestellt.
Für die Umsetzung braucht man PWM mit den digitalen Ausgängen vom MCU um die

Geschwindigkeitsteuerung zu erreichen. Mit unserem Motortreiber haben wir drei Möglichkeiten, den DC-Motor anzutreiben: vorwärts, rückwärts und bremsen.

1.4 Struktur der Arbeit

Diese Masterarbeit ist wie folgt strukturiert:

Kapitel 2-Grundlagen: In diesem Kapitel werden die wichtigsten fachlichen Grundlagen erläutert. Dazu gehört BLE, die im Projekt teilgenommene Hardware und die eingesetzten Verbindungsprotokolle sowie das Betriebssystem Android und die Grundlagen einer Android Anwendung.

Kapitel 3-Anforderungen: In diesem Kapitel werden die Anforderungen definiert, Anwendungsfälle vorgestellt. Anschließend einen schnellen Überblick über das Verfahren, das für die App eingesetzten werden.

Kapitel 4-Konzept und Design: Die Konzeption der App ist in diesem Kapitel beschrieben, die Kommandos werden definiert, Entwicklungsumgebung wird vorgestellt und die App strukturiert.

Kapitel 5-Umsetzung: zeigt die Implementierung der App.

Kapitel 6-Test: In diesem Kapitel wird die App sowie die Ergebnisse präsentiert und Erweiterungsmöglichkeiten vorgeschlagen.

Kapitel 7-Zusammenfassung und Ausblick: Hier wird eine Zusammenfassung gegeben Anforderungen mit der realisierten App abgeglichen und einen Ausblick für mögliche Weiterentwicklungen gegeben.

1.5 Zusammenfassung

Heutzutage kommen immer mehr Apps auf den Markt, um bestimmte Alltagssituationen zu erleichtern. Je nach Wünsche und Bedürfnisse sind sie immer beliebt. Diese Masterarbeit beschreibt den Entwurf und die Implementierung einer Android Benutzeroberfläche, die Bluetooth Low Energy (BLE) als Kommunikationstechnologie um eine drahtlose Datenübertragung unter Verwendung des Nordic nRF52 BLE SoC einzusetzen, welche die Fernsteuerung und Nutzung eines Fahrzeugs im Rahmen des ATLAS Projekt ermöglichen soll. In dieser Arbeit wird gezeigt, wie eine solche BLE App zu realisieren ist. Nachdem wir die Anforderungen festgelegt haben Anschließend werden die einzelnen Komponenten getestet. Erweiterungen und weitere Funktionalitäten werden auch evaluiert.

2 Grundlagen

In diesem Kapitel werden relevante Grundlagen, die für diese Arbeit wichtig sind, dargestellt. Darunter zählen Bluetooth Low Energy, das Betriebssystem Android sowie eine technische und theoretische Beschreibung über die Hardware Komponenten des ATLAS-Projektes.

2.1 Bluetooth Low Energy

2.1.1 Überblick

Bluetooth Low Energy (auch Bluetooth LE oder BLE genannt und wird als Bluetooth Smart vermarktet) ist ein globaler Standard für drahtlose Technologien mit kurzer Reichweite. Die Unterschiede zwischen "klassischem" Bluetooth und BLE sind bereits auf der physikalischen Schicht erkennbar. Anstatt 1-MHz-Kanäle zu verwenden, teilt die BLE das 2,4-GHz-ISM-Band in 40 Kanäle mit je 2 MHz Bandbreite. Die Modulation aller Kanäle wird mithilfe “Gaussian Frequency Key Shiftings” (GFSK) erzeugt.
Es gibt zwei Arten von Kanälen: Kanäle: 37, 38 und 39 werden als Advertisings Kanäle zum Discovery, Aufbau von Verbindungen (Initiation) und zum Senden von Broadcast-Daten oder Advertisingspaket verwendet. Die letzten restlichen 37 Kanäle werden als Data Kanäle für eine bidirektionale Datenübertragung zwischen zwei verbundenen Geräten (connected) verwendet. [14]
Die Bluetooth-Kernsystemarchitektur wird von der Bluetooth Special Interest Group (Bluetooth SIG) in der so genannten Kernspezifikation definiert. (Core_v4.2) Das Dokument beschreibt, wie die BLE-Kommunikation auf allen Ebenen des Bluetooth-Stacks implementiert werden kann. [23]

BLE-Module sind von verschiedenen Herstellern erhältlich: wie Texas Instrument, Nordic Semiconductor, Cypress Semiconductor…Wir werden also das BLE Protokoll Stack von Nordic nRF52832 Modul behandeln. Das BLE Modul wurde im Rahmen einer Bachelorarbeit ausgewählt [26]. Nordic Semiconductor ist ein Halbleiterunternehmen, das sich auf drahtlose Systems-of-Chip (SoC) mit geringer Leistung und allgemeiner Verbindungstechnik über das 2,4 GHz ISM-Band spezialisiert hat. Sein Schwerpunkt ist die Bereitstellung von Schaltungen mit geringem Stromverbrauch für die Elektronik Verbraucher.Nordic Semiconductor bietet seinen Protokoll Stack unter dem Namen SoftDevices an. SoftDevice dient als Schnittstelle zwischen der Anwendungsschicht und der physikalischen Schicht von Nordic. Sie enthält die Implementierung von GATT, GAP, L2Cap, ATT, Link Layer und Sicherheitsmechanismen. Nordic Semiconductor stellt sie als qualifizierte, vorkompilierte Binärdateien zur Verfügung. SoftDevice-Versionen S132 werden für nRF52832 unterstützt. Dieser Protokoll Stack kann von der Website von Nordic

heruntergeladen werden und kann leicht auf nRF52-System-on-Chips implementiert werden. Der S132 Soft device: Bluetooth 4.2-konformer Single Modus BLE. Das BLE-Stack implementiert eine Schichtenarchitektur wie in der Abb. 3 dargestellt wird:

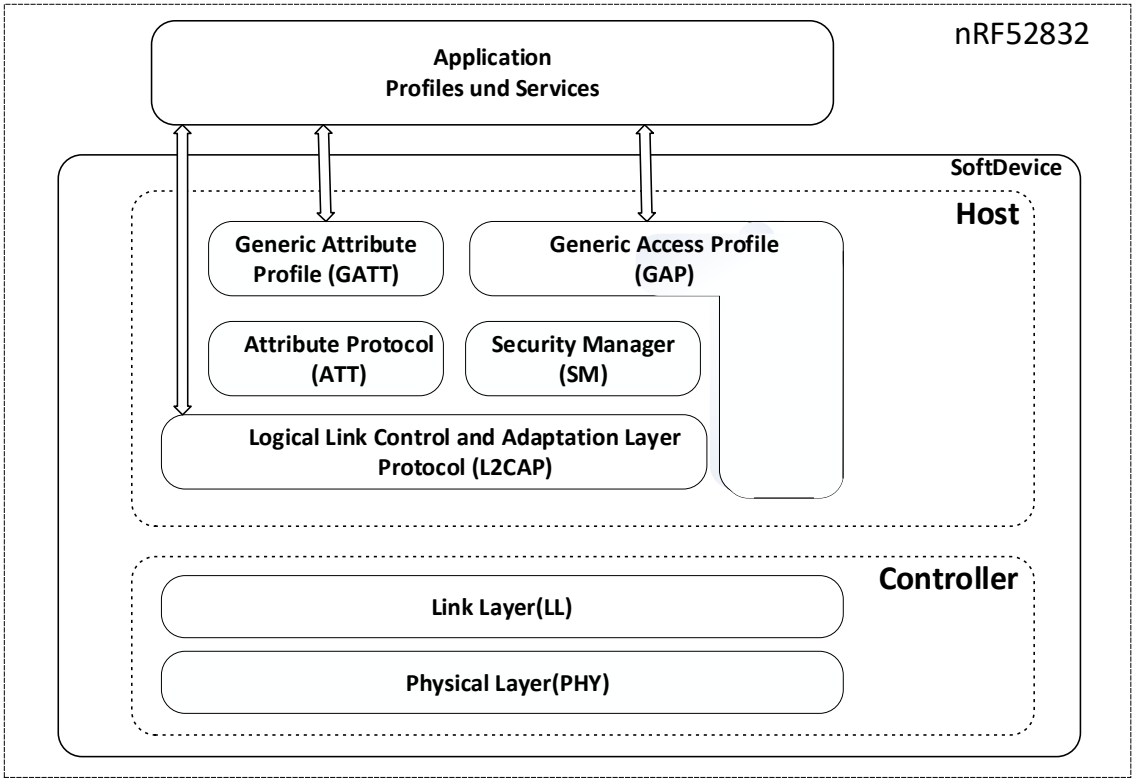


Abbildung 3: BLE-Stack des nRF52832 Modul [18]

Link Layer ist die Schicht, die direkt mit der Physikalischen Schicht (PHY) oder Bitübertragungsschicht verbunden ist. Sie setzt mehrere Einschränkungen, um die Link Layer Zustandsmaschine sehr einfach zu halten. Daraus wurden fünf Zustände definiert: Advertising, Scanning, Initiating, Standby, Connection.

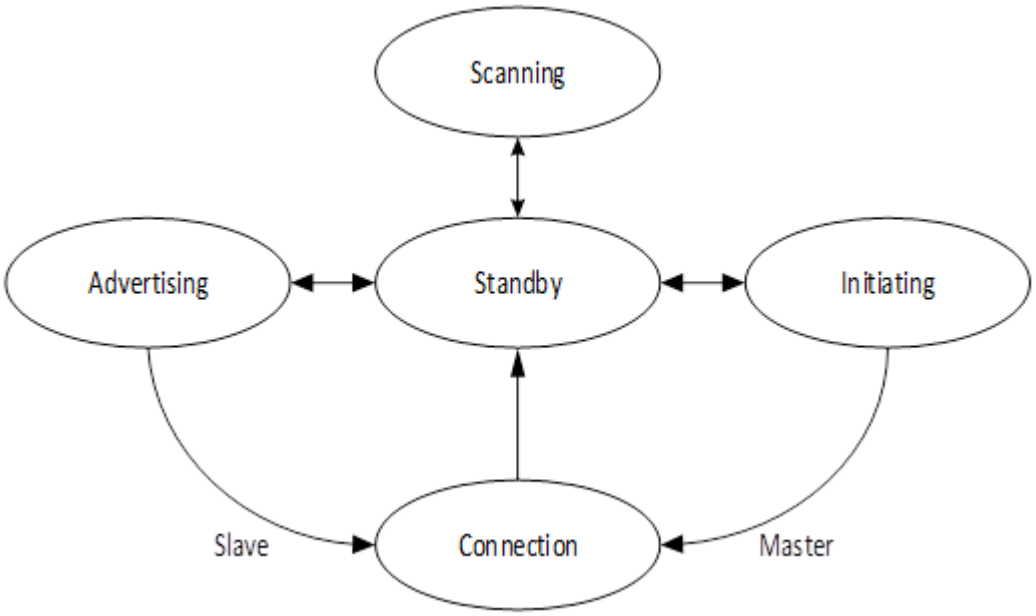


Abbildung 4: Link Layer Zustandsmaschine [15]

2.2 Einführung

2.2.1 Generic Access Profile (GAP)

Es gibt ein grundlegendes Profil, das alle BLE-Geräte unterstützen müssen, und zwar das Generic Access Profile (GAP). -GAP: macht das Gerät nach außen sichtbar, bestimmt bzw. definiert die Rolle des Gerätes im Netzwerk und beschreibt seine Bindung und Privatsphäre. Die verschiedenen Rollen eines BLE-Gerätes sind: Broadcaster, Observer, Central, Peripheral.

2.2.2 Generic Attribute Profile (GATT)

Das Generic Attribute Profile, das als GATT-Profil bekannt ist, ist eine BLE-Spezifikation, die das Attribute Protocol verwendet, um eine Art der Kommunikation zwischen Bluetooth LE-Geräten zu definieren. GATT ist ein Framework von Services und Characteristics, dass diese Attribute als Bausteine verwendet. GATT ist sehr hilfreich bei der Entwicklung, wenn in einem Gerät mehrere Services unterstützt werden müssen. Er gruppiert diese Attribute und definiert, wie alle diese Daten erkannt und dann zwischen den Geräten übertragen werden. Mit Abbildung 5 ist eine Form als Hierarchie, wie Daten gespeichert werden. [17]

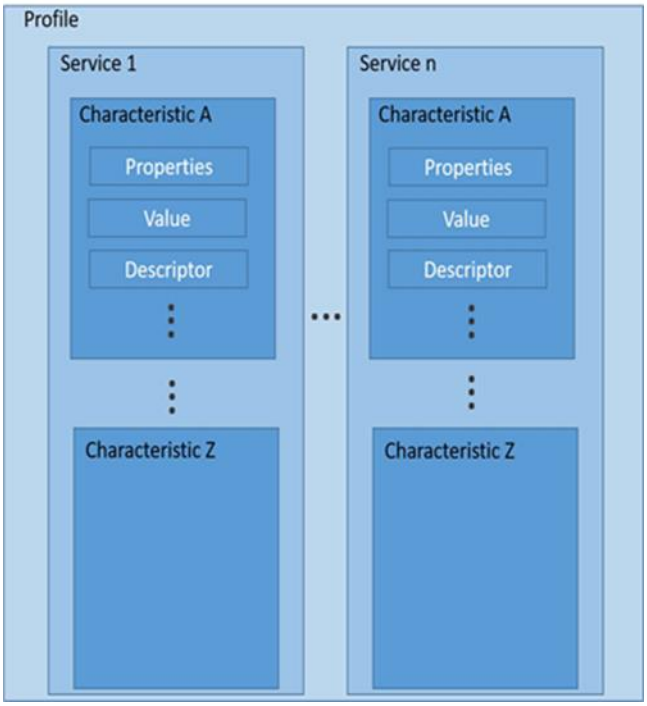


Abbildung 4: GATT Hierarchie [23]

Alle Standard-BLE-Profile (Gerät) müssen GATT-konform sein, d.h. die Anwendungs- und Anwenderdaten werden auf Basis der GATT-Definitionen strukturiert, verpackt und gesendet. Dadurch wird die Kompatibilität zwischen den BLE-Geräten verschiedener Hersteller gewährleistet. Nach ATT werden GATT Rollen auch definiert: GATT Client Rolle und GATT Server Rolle.

2.2.3 Attribute Protocol (ATT) und Attribute Methoden

Die Daten werden im Attributprotokoll gespeichert, das ATT-Format definiert, wie das Attribut verwaltet wird, der Typ des Attributs und wie man auf diesen Attribut zugreifen kann. Das Attributprotokoll ist atomar und sequenziell im Betrieb. Es gibt in BLE sechs Arten von Operationen auf das Charakteristic: **Command, Request, Response, Notification, Indication, Confirmation.**Das Attribut Protocol ist sequentiell im Betrieb: Request -> Response **;** Confirmation ← Indication.

- Command: die vom Client an den Server gesendet werden um eine Aktion auszuführen und keine Response erfordert. Ein Beispiel für einen Befehl ist das Write Command.
- Request: die vom Client an den Server gesendet werden und eine Response erfordern. Einige Beispiel für einen Request sind: Read Request, find Request, Write Request.
- Response: die vom Server als Antwort auf eine Anfrage gesendet werden. Ein Beispiel für eine Response ist das Read -Response, Write-response. Der Client kann immer nur ein ATT Request auf einmal schicken. Eine einzelne Komplete Anfrage wird nach Erhalt der Response abgeschlossen. Diese reduziert die Komplexität und

Speicherbedarf auf dem Server. Außerdem ermöglicht die Implementierung eines Attributservers sehr wenig Code. Für jede Anfrage kann es nur zwei mögliche Responses geben: Eine Response, die direkt mit der Anfrage verknüpft ist, oder eine Error Response, die die Auskunft gibt, warum die Response fehlgeschlagen wurde.

- Notification: vom Server an den Client gesendet, um den Client darüber zu informieren, dass sich ein bestimmte Characteristic Value (Attribute) geändert wurde. Damit diese vom Server ausgelöst und gesendet werden kann, muss der Client Notification vom interessierten Characteristic aktiviert werden. Man muss beachten, dass eine Notification keine ACK des Clients erfordert um den Empfang zu bestätigen. Die Kommunikation via Notification ist schneller.
- Indication: ähneln sehr stark mit Notification, erfordern aber, dass eine ACK vom Client zurückgeschickt wird, um den Server darüber zu informieren, dass die Indication erfolgreich empfangen wurde. Außerdem kann der Server immer nur eine Indication auf einmal senden, d.h. erst nach dem Empfang einer Bestätigung ATT Confirmation für eine vorherige Indication wird dann die nächste Indication gesendet werden. Die Kommunikation via Indication ist langsamer. [14]

2.2.4 Characteristic

Ein Characteristic besteht aus drei Grundelementen:

1.Characteristic Deklaration: Sie ist ein wichtiger Bestandteil eines Characteristics und beschreibt die Eigenschaften von Characteristic (Attribute Value) als:-Characteristic UUID-Characteristic handle (einziger handle eines Gerätes um das Attribut zu identifizieren bzw. Adressieren)
2.Characteristic properties :(read, write, indicate, notify, broadcast, etc.)
3.Characteristic Value: Eine Charakteristic Value wird als Attribute genannt. Zustandsdaten, die ein Gerät ausstellt, sind in einem Attribut Value verfügbar. Jedes Attribut hat einen Wert von einer Länge von null bis maximal 512 Byte lang sein, manche Attribute haben eine festgelegte Länge. Attribute Values können geschützt sein. ATT erlaubt nicht, dass mehrere Attribut Value in einer einzigen PDU (Protocol Data Unit) übertragen werden. Eine PDU enthält nur eine Attribute Value. Wenn dieser Wert in einer einzelnen PDU zu lang ist, kann ATT sie auf mehrere PDUs aufteilen. [15]

2.2.5 Characteristic Descriptor

Die Characteristics Deskriptoren liefern zusätzliche Informationen über das Characteristic beliebige Nummer, Characteristic Benutzer Beschreibung und Kundenkonfiguration und können auch herstellerspezifische Informationen sein. **Client Characteristic Configuration Descriptor (CCCD**): Dieser ist einer der wichtigsten und am häufigsten verwendeten Descriptor. Er wird verwendet, wenn man Notifikation oder Indikation für eine Characteristic aktivieren, deaktivieren oder konfigurieren muss.

2.2.6 Universell Unique Identifier (UUID)

Es gibt verschiedene möglichen Arten von Datentypen. Deshalb wird eine 128-Bit-Nummer verwendet, um den Typ des Attributs zu identifizieren. Diese eindeutige Identifikation wird als Universell Unique Identifier (UUID) bezeichnet. Im Allgemeinen wird eine kürzere Form von 16 Bit UUIDs verwendet. Die Bluetooth SIG hat eine einzige 128-Bit-UUID definiert, die als Bluetooth Base UUID bezeichnet wird und sie wird mit der kürzerem 16-Bit kombiniert. Das ermöglicht einen effizienten Transfer von Datentypen zwischen Geräten. Eine 16 Bit UUID kann so aussehen: 0x1815. Jede Characteristic und jeder Service wird durch seine eigene UUID identifiziert. Manche UUIDs sind für bestimmte Zwecke reserviert. Die Bluetooth Base UUID ist wie folgt definiert:00000000-0000-1000-8000-00805F9B34FB
Die 128-Bit-Wert kann aus der 16-Bit-Wert abgeleitet werden, indem man das xxxx der Bluetooth_Base_UUID durch den hexadezimalen Wert der 16-Bit-UUID ersetzt: 0000xxxx-0000-1000-8000-00805F9B34FB. [15] , [17]

2.2.7 Applikation

Die Applikation Schicht befindet sich über dem BLE Controller und dem Host. Sie ist eine direkte Oberfläche mit dem Benutzer und eine BLE-Schnittstelle, in der die Programmierung erfolgt. GATT definiert die Attributen als Gruppe von Characteristic und Services, während die Applikation die Spezifikationen dieser Gruppen-Attribute definiert. Services stellen Characteristics aus. Sie definieren das Verhalten, das durch Characteristics exponiert wird. Characteristics definieren die kompatiblen Datenformate. [15]

2.3 M16C/62

Als Controller wurde der M16C/62 ausgewählt, er ist ein leistungsfähiger Mikrocontroller der M16C-Familie (16-Bit-Mikrocontroller) von Mitsubishi Electric entwickelt und wird inzwischen von Renesas Electronics, dem ausgegliederten Halbleiterbereich von Mitsubishi, weiterentwickelt und hergestellt. Mit 1M Bytes Adressraum er ist in der Lage, Befehle mit hoher Geschwindigkeit auszuführen. Außerdem verfügt er über einen eingebauten Multiplikator und DMAC, wodurch er sich ideal für die Steuerung von Büro- und Kommunikationssystemen und Automotive Anwendungen eignet. [24]

2.3.1 Betrieb von Funktionsblöcken

M16C/62 enthält bestimmte Einheiten in einem einzigen Chip. Diese Einheiten umfassen ROM (Kapazität 256k Byte) und RAM (20k Byte), um Anweisungen, Daten zu speichern und die Zentraleinheit (CPU) arithmetische/logische Operationen auszuführen. Er verfügt über eine große Anzahl On Chip Peripherie: peripherie-Einheiten wie Timers, Serial I/O, D-A Umwandler, CRC calculation Unit, A-D Umwandler, und I/O Ports. Zwei DMAC-Kanäle (Direct Memory Access Controller), die das Senden von Daten an den Speicher ohne Verwendung der CPU ermöglichen.
Der Renesas M16c/62 verfügt über drei unabhängige UARTs, von denen jeder eine

serielle Vollduplex-Kommunikation mit externen Geräten, einschließlich anderer MCUs, ermöglicht. Der M16C/62 UART wurde so entworfen, dass er flexibel und einfach zu bedienen ist. [24]

2.3.2 Universal asynchronous Receiver Transmitter (UART)

MCU muss häufig Daten mit anderen Mikrokontroller oder Peripheriegeräten austauschen. Der Datenaustausch kann mit Hilfe von parallelen oder seriellen Techniken erfolgen. In den meisten Mikrocontrollern ist ein UART eingebaut, der für die Realisierung einer seriellen Schnittstelle gedacht ist. Das Prinzip der üblichen seriellen Datenübertragung beruht auf der asynchronen Betriebsart, d.h. die Daten werden ohne Taktsignal übertragen. und die Synchronisierung wird im Datenstrom transportiert. Zuerst wird ein Startbit (ST: aktiv Low) und dann die Dateninformation als ASCII, wobei das LSB zuerst übertragen. Das Ende des Zeichens wird meist durch ein oder zwei Stopp-Bits (SP), welche aktiv High (1) sind, gekennzeichnet. Zur Fehlererkennung kann ein Paritätsbit (P) mit übertragen werden. Die asynchrone Datenübertragung setzt ein exakt definiertes Übertragungsformat voraus, welches beim Sender und Empfänger identisch eingestellt sein muss. Die wichtigsten Parameter zur Definition dieses Formats sind:
• Baudrate (Bit/s): Anzahl der Datenbits: Die Datenübertragungsraten werden normalerweise als Baud oder Bits pro Sekunde angegeben. 9600 Baud bedeutet beispielsweise, dass die Daten mit 9600 Bits pro Sekunde übertragen werden.
• Paritätsbit: kein Paritätsbit, gerade oder ungerade
• Anzahl der Stoppbits: 1, 1.5, 2.
Jedes über den UART gesendete Zeichen wird in einen Daten-Frame verpackt:

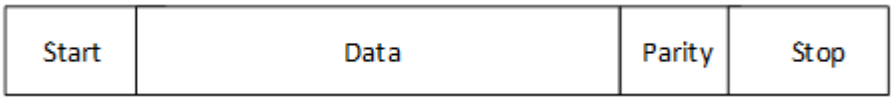


Abbildung 5: UART Daten Frame [10]

Im einfachsten Fall besteht eine serielle Schnittstelle lediglich aus drei Leitungen: einer Empfangsleitung, einer Sendeleitung, sowie einer Masseleitung (GND).
Für eine gleichzeitige Datenübertragung und Empfang, also Vollduplex, ist eine separate Hardware für Übertragung und Empfang erforderlich. Der UART verwendet die TXD- und RXD-Register direkt zum Senden und Empfangen von Daten.

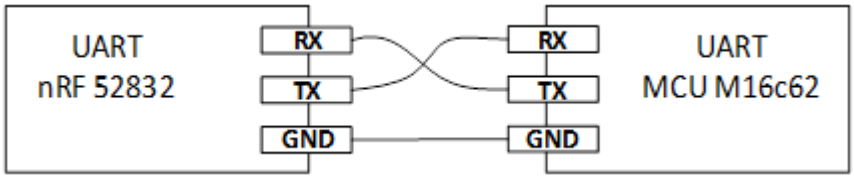


Abbildung 6: nRF52832 - MCU M16c62 UART Verbindung [10]

Der Pin TXD des nRF ist der serielle Ausgang und muss mit dem RXD-Pin des MCUs verbunden werden. Parallel ist der Pin RXD des nRF´s der serielle Eingang und muss mit dem TXD-Pin des MCU verbunden sein.

2.3.3 ASCII

American Standard Code für Information Interchange kurz ASCII ist eine standardisierte Sieben-Bit-Methode zur Zeichenkodierung. Es ist seit vielen Jahren im Einsatz. Allerdings ist ASCII immer noch die verbreitetste Methode zur Kodierung alphanumerischer Daten. Zum Beispiel wird der Großbuchstabe C in ASCII als 0x kodiert und das Symbol 0x43 gibt die hexadezimale Zahlendarstellung an.

2.3.4 PWM

Ein Elektromotor ist eine elektrische Maschine, die elektrische Energie in Form von rotarischen Bewegungen oder (Drehmomenten) in mechanische Energie umgewandelt. In diesem Abschnitt diskutieren wir eine bekannte Methode zur Steuerung der Drehzahl eines DC Motors mit Hilfe eines PWM-Signals. Das zugrundeliegende Konzept ist wie folgt: Wenn wir einen Gleichstrommotor einschalten und die erforderliche Spannung bereitstellen, läuft der Motor mit seiner maximalen Geschwindigkeit. Durch Änderung des Betriebszyklus können wir die Geschwindigkeit eines Gleichstrommotors nach Wunsch steuern. Wir wollen also ein solches Geschwindigkeitsprofil generieren (Abb. 7):

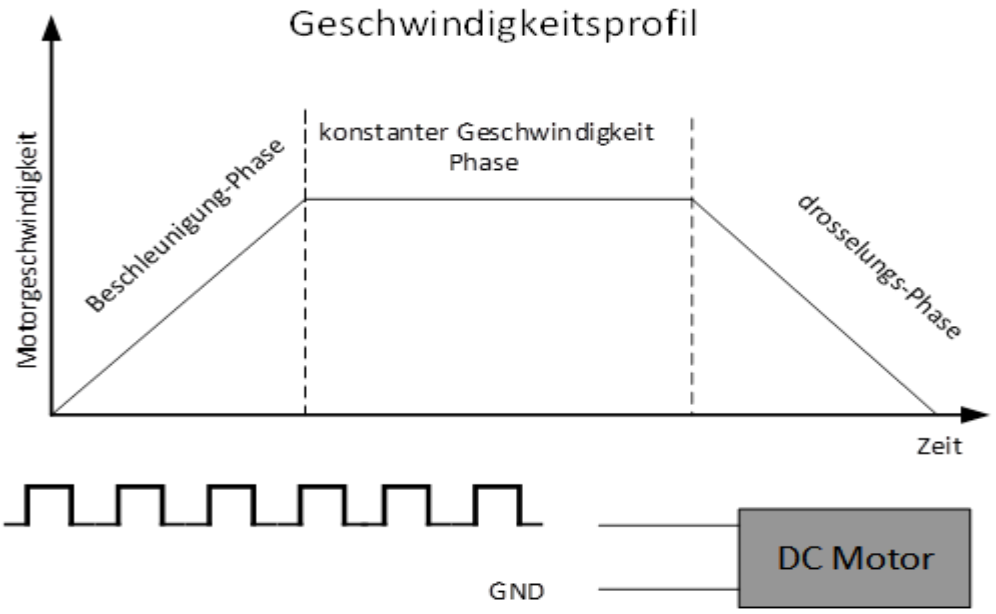


Abbildung 7: Geschwindigkeitsprofil eines DC Motors mit PWM [21]

Die Abbildung zeigt das Geschwindigkeitsprofil eines Gleichstrommotors im Laufe der Zeit, wenn ein PWM-moduliertes Signal an den Motor angelegt wird. Ein PWM-Signal

hat eine konstante Frequenz, die Trägerfrequenz genannt wird. Obwohl die Frequenz eines pwm-Signals konstant ist, kann man die Breite der Impulse (Tastgrad-engl.duty cycle) variieren, um die gewünschte, an die Last anzulegende Spannung zu erhalten. Der Tastgrad variiert (von 0% bis 100%) je nach Amplitude des Originalsignals. PWM-Ausgangssignale können einfach bei niedrigen Frequenzen mit reinen Softwaretechniken erzeugt werden: Unser MCU verfügt über elf Timer, die aus fünf Timer A und sechs Timer B bestehen.
Die Timer A können alle als PWMs verwendet werden. Sie geben Impulse mit einer bestimmten Breite aus. Der PWM-Modus hat zwei " Unter" Modi: 16-Bit und 8-Bit:
-Im 16-Bit-Modus bestimmt der Wert des 16-Bit-Ai-Timer-Registers die Puls-Weite und die Frequenz die, auf fXin / 65535 fest ist. Daher beträgt die maximale Frequenz der 16-Bit-PWM (unter der Annahme eines 20MHz fXin) ca. 305 Hz Die Pulsweite (wenn das Signal in einem hohen Zustand ist) der 16-Bit-PWM ist
Pulse width (high) = n / fin, or % duty = n / 65535 * 100, wobei n der in das Ai-Zähler-Register geladene Wert ist.
- Im 8-Bit-Modus werden die 8-Bits höherer Ordnung zur Bestimmung der Pulsbreite und die untere 8-Bit zur Bestimmung der Frequenz, wobei die Frequenz fin / 255 fest ist, oder eine maximale Frequenz von ca. 78.431 Hz ist. Die Pulsbreite (wenn das Signal in einem hohen Zustand ist) der 8-Bit-PWM ist:
Pulse width (high) = n * (m+1) / fin, or n / (255 * fPWM), % duty = n / 255 * 100 wobei n der in das Ai-Zähler-Register höhere Ordnung geladene Wert ist. und m in die Adresse des Ai-Zählerregisters niedriger Ordnung geladen wird. Die Pulsweite kann jederzeit durch Schreiben in das Ai-Zählerregister geändert werden.[22]

2.4 Motortreiber TLE 5205-2

Viele Motortreiber gibt's als fertig aufgebaute Platinen. Der TLE 5205-2 ist eine integrierte Leistungs-H-Brücke mit DMOS-Ausgangsstufen zur Ansteuerung von DC-Motoren und zur Optimierung für DC-Motor-Management-Anwendungen: Bei einer H-Brücke (Abb. 8) handelt es sich um eine elektronische Schaltung zum Ansteuern von Gleichstromelektromotoren, die sich drehen, sobald Strom durch sie fließt. Die Drehrichtung hängt von der Stromrichtung ab. Die Bauteile sind in Form eines großen H angeordnet. Sie hat ihren Namen von dem Aussehen des Schaltbildes.

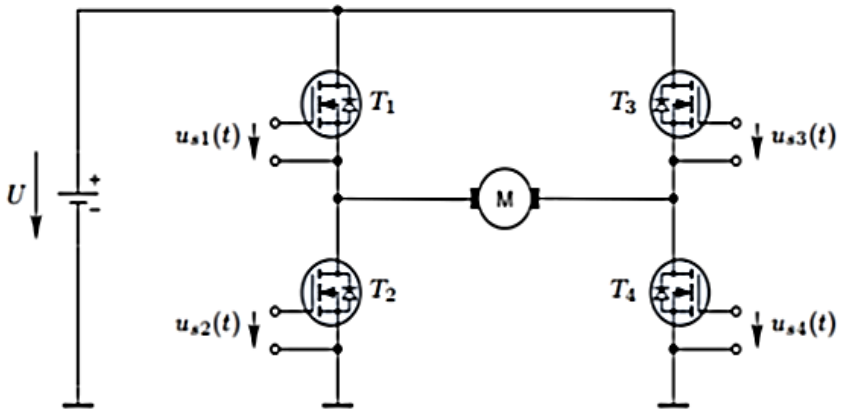


Abbildung 8: H-Brücke mit MOSFET Transistoren [11]

Nun geht es darum, den Motor in Bewegung zu versetzen. Wenn T2 und T3 leiten, während T1 und T4 sperren, dreht sich der Motor in die eine Richtung. Wir gehen davon aus, dass sich der Motor dann nach links dreht. Leiten hingegen T1 und T4, während T2 und T3 sperren, dreht er sich in die andere Richtung. Hinweis: Es muss natürlich darauf geachtet werden, dass niemals die Transistoren T1 und T2 bzw. Die Transistoren T3 und T4 gleichzeitig eingeschaltet sind (Kurzschluss).
Die Betriebsarten Vorwärts (cw), Rückwärts (ccw), Bremse und hohe Impedanz werden von nur zwei Steuerpins mit TTL/CMOS-kompatiblen Pegeln aufgerufen. Außerdem sind diese Transistoren mit zusätzlichen Dioden antiparallel geschaltet. Diese sogenannten Freilaufdioden ermöglichen es, dass der Strom durch die Induktivität des Motors auch während und nach dem Umschaltvorgang weiterfließen kann. Dadurch wird verhindert, dass hohe Induktionsspannungen entstehen, welche die Transistoren zerstören könnten.

2.5 Nordic nRF52832 (ATLAS Modul)

Bluetooth Low Energy wird unter Nordic nRF52832 System on Chip (SoC) basiert auf dem ARM Cortex TM M4F-Prozessor, mit einen 512KB Flash, 64KB RAM, eingebettetem 2,4GHz Multiprotokoll Transceiver und verwendet einer integrierten Chip-Antenne Das Modul enthält 32 konfigurierbare ein/Ausgabe -Ports und unterstützt verschiedene Peripherie-Protokolle wie SPI, I2C, UART, PWM, ADCs. Es ist ein energiesparender, günstiger Baustein und kann von der App direkt angesprochen werden.
Das SoftDevice Application Programming Interface (API) steht Anwendungen als eine C-Programmiersprachen-Schnittstelle zur Verfügung, die auf SuperVisor-Aufrufen (SVC) basiert und in einer Reihe von Header-Dateien definiert ist. SVCs sind von der Software ausgelöste Unterbrechungen, die einem Prozeduraufruf-Standard für die

Parameterübergabe und Rückgabewerte entsprechen. Jeder SoftDevice-API-Aufruf löst ein SVC-Interrupt aus. Application Code ist hier unserer Nordic UART Service.[19]

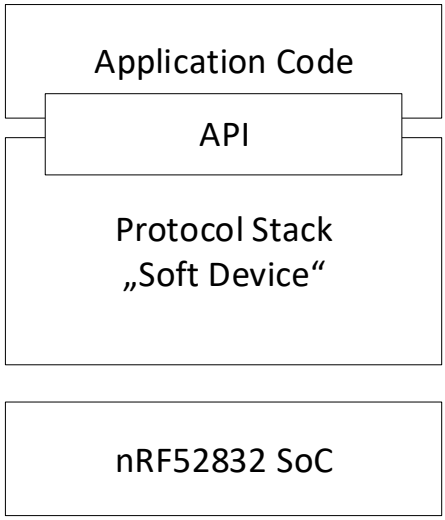


Abbildung 9: Nordic SoC mit Soft Device

2.6 Fahrstraßensteuerung

Die zu den Fahrstraßen gehörenden Weichen werden [25] von Fahrzeug über eine Infrarot-Schnittstelle angesteuert Diese werden bei dem ATLAs-Projekt eingesetzt. Diese fertigen Module verfügen über IR Empfänger. Somit wird Mikrocontroller IR Steuersignals nach Fahrstraßensteuerung übertragen. Die App übernimmt weiter die Steuerung über BLE UART-Kommandos und lässt sich einfach die Fahrstraßensteuerung bedienen. Ziel der Fahrstraßensteuerung ist es, Kollisionen zu vermeiden das heißt: die Fahrzeuge können ohne Halt bzw. ohne Fahrunterbrechung Gleis wechseln. Die erweiterte Idee ist es, mittels App ATLAS-Fahrzeug über eine Fernsteuerung zu führen und die Strecken oder Gleise freigeben lassen. Dieses Model (Abb.10) zeig den Übergang von Gleis 1 auf Gleis 2 Mit Hilfe von einer Fahrstraßensteuerung.

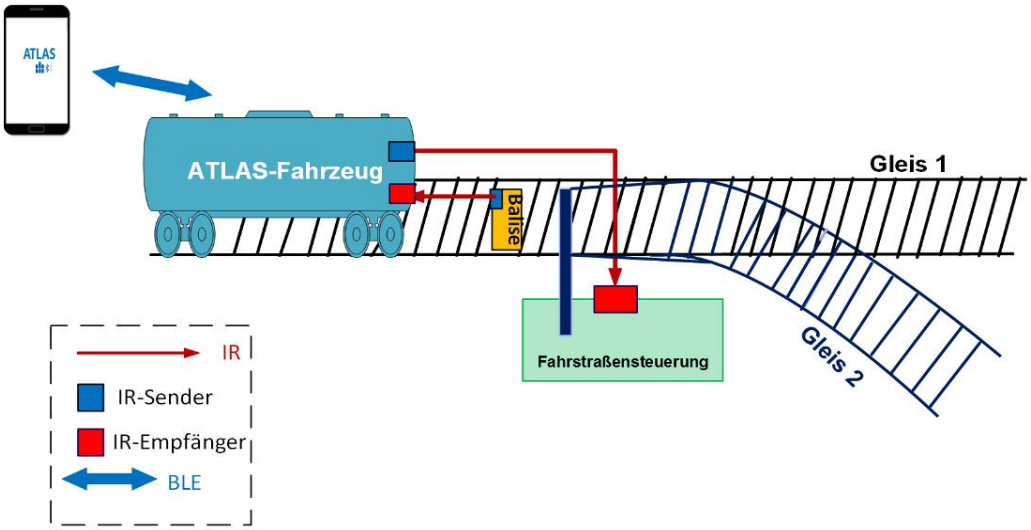


Abbildung 10: Gesamtüberblick über das ATLAS-Projekt

2.7 Central-Peripheral und Client-Server Architektur

Funktionsprinzip anhand BLE Rollen:

Es ist wichtig, dass man BLE verstehen muss, bevor man mit der Entwicklung beginnt, da BLE sehr asymmetrisch ist. Die BLE Geräte operieren nach einem bestimmten Netzwerkkonzept und müssen bestimmte Rollen anhand ihrer Kapazitäten, Ressourcen und Fähigkeiten innerhalb der Verbindung übernehmen. Diese Rollen werden durch die Profile GAP und GATT vorgegeben, die wir schon einmal im Abschnitt 2.2 vorgestellt haben. Diese Rollen sind essentiell zum Verständnis des Konzepts von Bluetooth Low Energy. BLE Rollen sind in Vor und Nachverbindungen (Connection) unterteilt. Diese Terminologien werden auch durch Bluetooth Low Energy Core Specifications definiert und häufig in der Literatur verwendet. [23]

2.7.1 Vor Verbindung:

Beim Start haben wir zwei Geräte ein Central und ein Peripheral.
-Peripheral fängt mit Advertising an und wartet darauf, dass ein Central sich mit ihm verbindet. (Paket-Format befindet sich im Anhang A1)
Der Central scannt nach dem Periphal und kann nach einem SCAN_REQ (Anhang A2) zusätzliche Daten vom Advertiser wie (Name, RSSI, Service UUID) als Scan Response erhalten. (Anhang A3) Anhand von Advertisings Daten wählt der Scanner den Peripheral und initiiert mit ihm eine Verbindung. (Anhang A4) Der Initiator wird Master genannt während der Advertiser Slave Rolle übernehmen wird, nachdem er diese Connection Anfrage angenommen hat.

2.7.2 Nach Verbindung:

Nachdem eine Verbindung erfolgreich hergestellt wurde, können die beiden Geräte entweder Client oder Server sein. Abhängig vom GATT Implementierung (Client-Server Architecture) werden deswegen beide Rollen als GATT Client und GATT Server bezeichnet. In unserem Fall ist der Master ein GATT Client, und ruft die Remote Ressourcen ab. Der Slave ist ein GATT Server und hat eine lokale Datenbank von Ressourcen (GATT Profile: Services und Characteristic) und stellt diese Ressourcen für den Remote Client zur Verfügung. Der Client implementiert Uses Cases abhängig vom GATT-Profil und verwendet ATT Methoden nach Attribute Protokoll zum Daten Austausch.[12] Der GATT Client muss gleichzeitig einen discovery Prozess durchführen, um die Attribute (Characteristics) des Servers zu kennen. Sobald die Identifizierung der Services abgeschlossen ist, kann er diese Daten-Attribute je nach Properies: lesen, schreiben.oder vom Server benachrichtigt (Notification) werden.
Für unser Projekt sieht der GATT Server des ATLAS Bluetootth Moduls so aus.
Eine GATT Server Tabelle: sie wurde durch die App nRF Connect geladen.

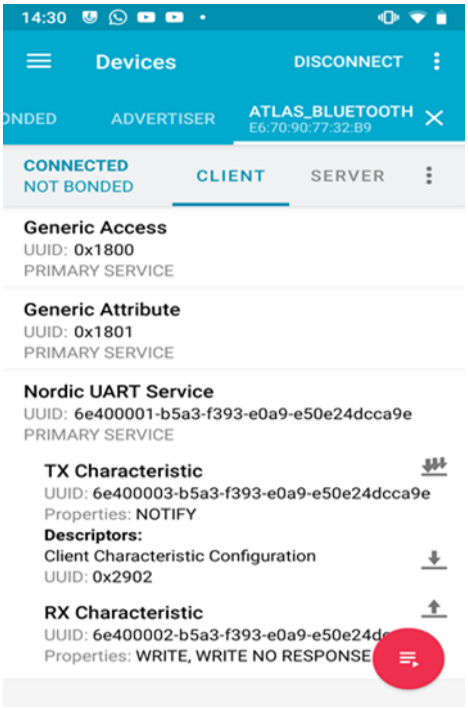


Abbildung 11: Aufbau des GATT Server Profils vom ATLAS Bluetooth

Die Anwendung umfasst zwei Services: GAP Service und den Nordic UART Service.
-GAP Service (UUID: 0x1800) und seine dazugehörige Characteristic (NAME und Appearance) müssen bei jedem Gerät implementiert werden.
-Die UUID des nordischen UART-Service lautet:
6E400001-B5A3-F393-E0A9-E50E24DCCA9E (Herstellerspezifische: 128-Bit-Service) Dieser Service weist zwei Characteristics auf eines zum Senden und eines zu

Empfangen von Daten. RX Characteristic (UUID: 6E400002-B5A3-F393-E0A9-E50E24DCCA9E): Das Smartphone kann Daten an das Gerät senden, indem er in die RX-Charakteristik des Services schreibt. ATT Write Request kann verwendet werden Die empfangenen Daten werden über die UART-Schnittstelle gesendet.
TX Characteristic (UUID: 6E400003-B5A3-F393-E0A9-E50E24DCCA9E): Wenn die App Notification für das TX-Characteristic aktiviert hat, kann das NRF Modul daten als Notification an die App senden. Das BLE Modul überträgt alle über UART empfangenen Daten als Notification weiter.
Zusammenfassend:
-Wenn der Client Daten an den Server senden muss, verwenden wir Write.
-Wenn der Server Daten an den Client senden muss, ohne dass der Client sie vorher anfordert, verwenden wir notify.[18]

2.8 Android

Android ist ein Open Source- und Linux basiertes Betriebssystem für mobile Geräte wie Smartphones und Tablet-PCs. Android ist im Besitz der Open Handset Alliance. Das Betriebssystem ist sehr verbreitet und stetig schnell wachsend. Android gilt als Marktführer unter den Plattformen für mobile Endgeräte.

2.8.1 Android-Architektur

Android-Architektur oder Android-Software-Stack:

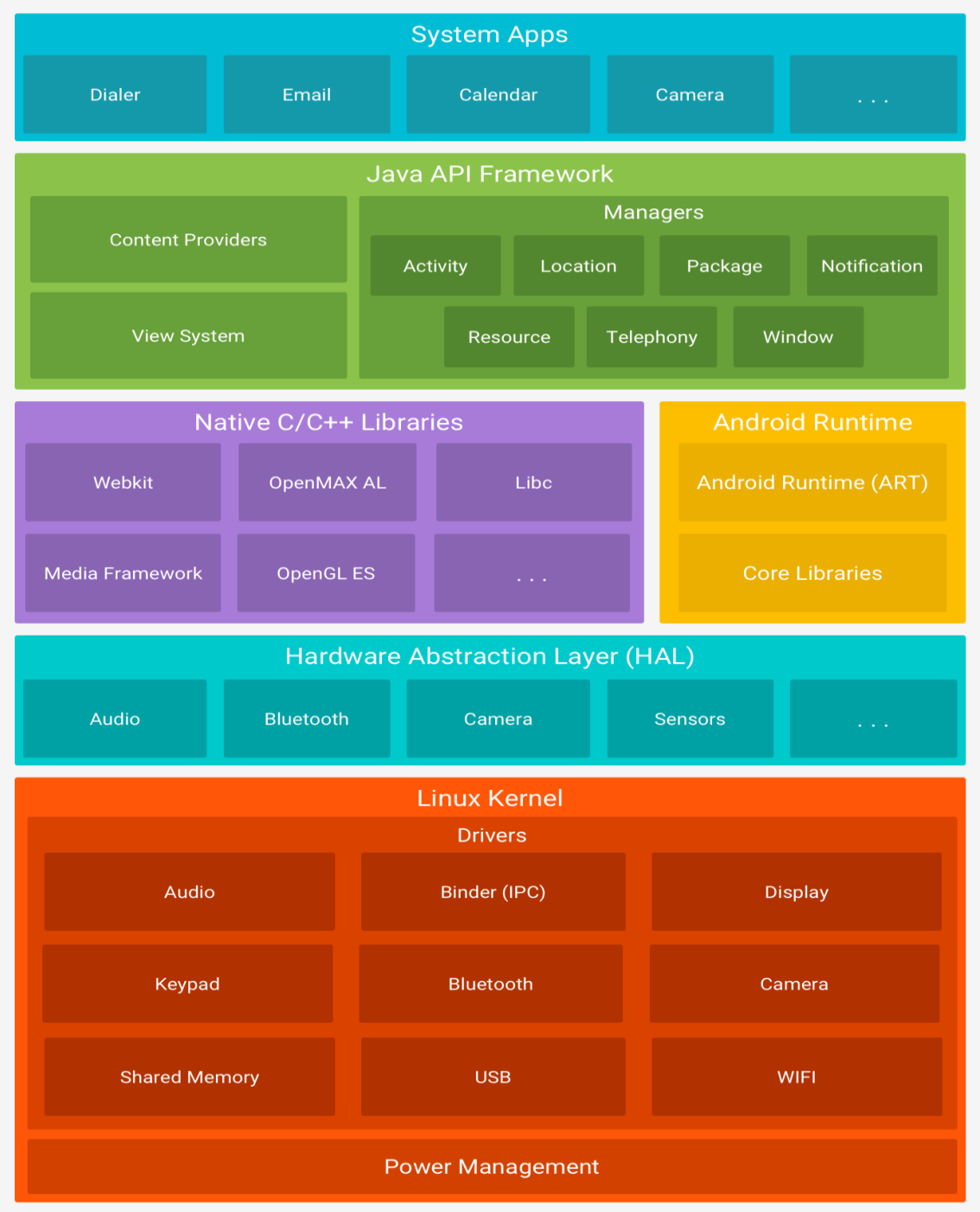


Abbildung 12: Android-Architektur [6]

Das Android-Software-Stack besteht aus:

1.System Apps

Die Anwendungen befinden sich auf der höchsten Ebene und sind die Tools, mit denen jeder, der Android benutzt, am besten vertraut ist. Android wird mit verschiedenen robusten Anwendungen geliefert, die die alltäglichen Funktionen unterstützen, wie Messaging, E-Mail, Internet-Browsing und verschiedene Anwendungen von Drittanbietern.

2.Java API framework

Android bietet Entwicklern Möglichkeiten und Werkzeuge umfangreiche, interaktive, reichhaltige grafische Anwendungen zu erstellen, mit dem Ziel diese Anwendungen im

Google Play Store zu implementieren. Android bietet eine Reihe von APIs (Application Programming Interfaces) für die Entwicklung einer Anwendung. Die Entwickler haben Zugang zu denselben APIs, die auch in den Kernanwendungen verwendet werden, sowie zu fast allen bestehenden Java-Bibliotheken. Diese APIs sind im Android SDK als Package gelistet. Sie können je nach Hostgerät (API Level) variieren. Diese soll es erlauben, Komponente wiederzuverwenden und die Struktur für Entwickler zu vereinfachen.

3.Native Libraries

Die nativen Bibliotheken sind kompiliert und C/C++-Binärdateien vorinstalliert, auf die das Android-System basiert.

4.Android Runtime

Innerhalb der Android-Runtime befinden sich zwei primäre Komponenten:

- Die Java-Kernbibliotheken, die Android zur Verfügung stellt: Android enthält ebenfalls eine Reihe von Kernlaufzeitbibliotheken, die den Großteil der Funktionalität der Programmiersprache Java, einschließlich einiger Java-8-Sprachfunktionen, die das Java-API-Framework verwendet, bereitstellen.
- Android Runtime (ART): Bei Geräten mit Android Version 5.0 (API-Level 21) oder höher wird jede Anwendung in einem eigenen Prozess und mit einer eigenen Instanz der ART ausgeführt. Die Hardware-Abstraktionsschicht (HAL) bietet Standardschnittstellen, die die Fähigkeiten der Gerätehardware dem übergeordneten Java-API-Framework zur Verfügung stellen.

5. Linux kernel

Android wurde auf dem Open-Source-Kernel von Linux erstellt. Der Linux-Kernel ermöglicht den Zugriff so nah wie möglich an der Hardware. Als Ergebnis werden Treiber in den Kernelbereich geschrieben, um möglichst schnell und effizient zu funktionieren. Der Linux-Kernel ist verantwortlich für Gerätetreiber, Energieverwaltung, Speicherverwaltung, Geräteverwaltung und Ressourcenzugriff.[6]

2.8.2 BLE im Android

Die Android-Plattform umfasst Unterstützung für den Bluetooth-Stack, der es einem Gerät ermöglicht, drahtlos Daten mit anderen Bluetooth-Geräten auszutauschen. Das Anwendungsframework ermöglicht den Zugriff auf die Bluetooth-Funktionalität über die Android-Bluetooth-APIs. Die Bluetooth LE API wurde komplett in Android 5.0 (API 21) eingeführt.[6] Android bietet einen Standard-Bluetooth-Stack, der sowohl Classic Bluetooth als auch Bluetooth Low Energy unterstützt. Android-Geräte mit einem qualifizierten Chipsatz können entweder Classic Bluetooth oder sowohl Classic Bluetooth als auch BLE implementieren. BLE ist nicht abwärtskompatibel mit älteren Bluetooth-Chipsätzen. Die Bluetooth-Anwendung (App) kommuniziert mit dem Bluetooth-Prozess über Binder. Der Bluetooth-Prozess (Bluetooth Service + Bluetooth Profile) verwendet JNI (Java Native Interface) zur Kommunikation mit dem Bluetooth-Stack und bietet Entwicklern Zugang zu verschiedenen Bluetooth-Profilen. Innerhalb des Anwendungs-Frameworks ist

der Anwendungscode, der die Android. Bluetooth-APIs zur Interaktion mit der Bluetooth-Hardware nutzt. Intern ruft dieser Code den Bluetooth-Prozess über den Binder-IPC-Mechanismus auf. [5]

2.8.3 Grundlagen der Android Applikationen

Zum besseren Verständnis der Funktionsweise einer App werden im Folgenden die grundlegenden Komponenten erläutert. Die wichtigen Komponenten einer App im Überblick: App-Komponenten sind die wesentlichen Bausteine einer Android-App.
Die Intents erlauben die Kommunikation zwischen Komponenten.

a. Activity

Eine Activity ist der grundlegende Baustein einer Android-App, einer der wichtigsten Komponente in der Programmierung mit Android. Sie ist wie ein einzelner Bildschirm mit Benutzeroberfläche. Eine Aktivität ist der Hauptzugang zu einer Android-App. Eine Aktivität interagiert mit dem Benutzer, so dass ein Fenster zur Platzierung von UI-Elementen entsteht. Eine Android-Anwendung kann mehrere Aktivitäten enthalten, d.h. viele verschiedene Bildschirme, die miteinander interagieren können. In welcher Reihenfolge diese aufgerufen werden, kann von Aktionen des Benutzers abhängen (beispielsweise, indem er einen Befehl in der Action Bar antippt) oder durch die Programmlogik vorgegeben sein. Der vollständige Lebenszyklus einer Activity ist in Abbildung 13 zu sehen. Er beginnt mit dem ersten Aufruf von **`onCreate()`** und endet mit einem einmaligen Aufruf von **`onDestroy()`**. [7]

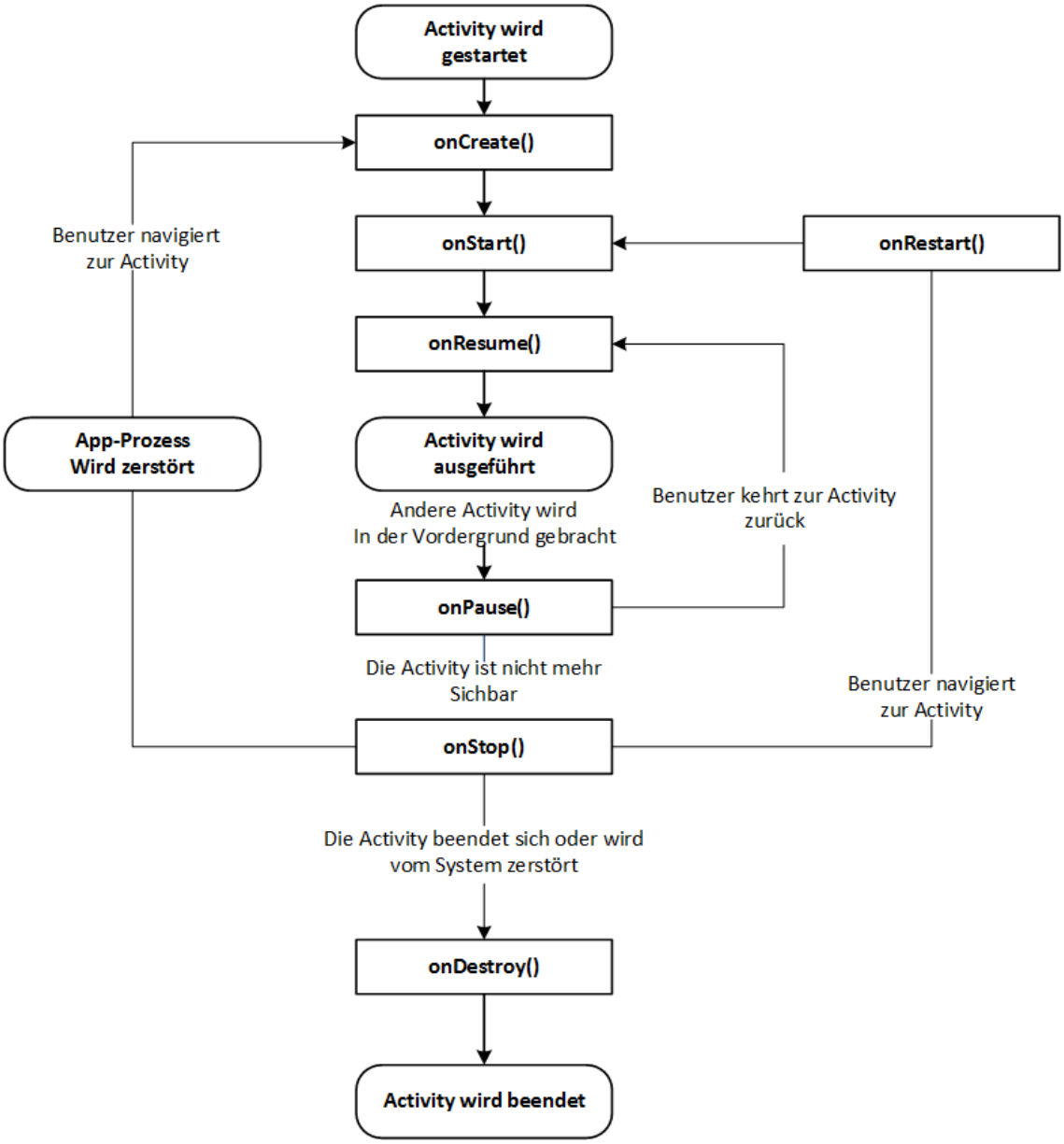


Abbildung 13: Activity - Lebenszyklus [7]

b. Der Activity-Stapel

Eine App kann aus Sammlung von Aktivitäten bestehen. Diese Aktivitäten werden in einem Stapel angeordnet. Der Zustand jeder Aktivität wird durch ihre Position auf dem Aktivitätsstapel bestimmt. Neue Aktivitäten werden über die bereits bestehenden gestartet, alte Aktivity bleibt im Hintergrund (onStop) Mit der Rücktaste wird die aktuelle Aktivity beendet und zurück zur darunterliegenden gegangen. Die nächste Aktivität unten auf dem Stapel steigt nach oben und wird aktiv. Abbildung 14 veranschaulicht diesen Prozess.[9]

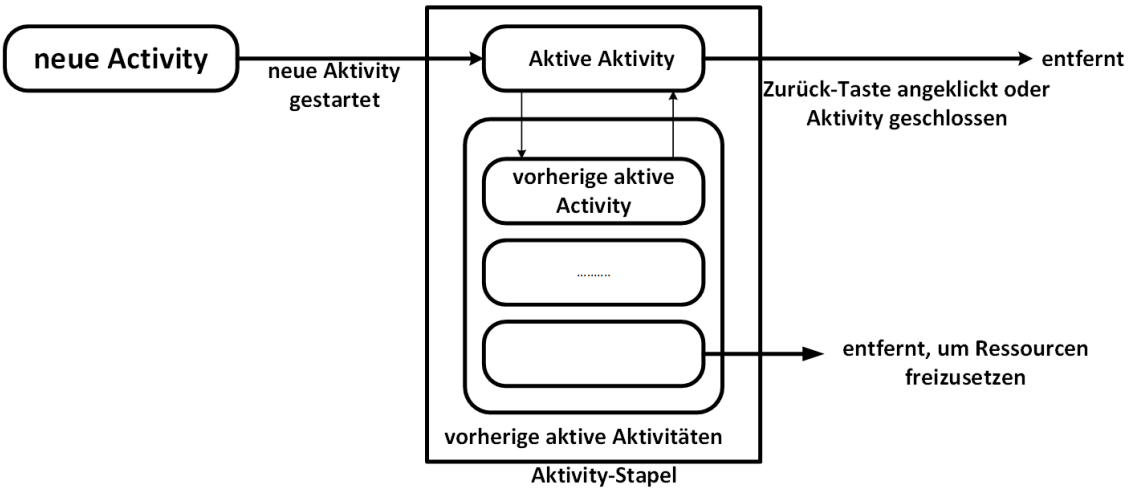


Abbildung 14: Activity Stapel [9]

c. Fragment:

In diesem Abschnitt stellen wir einen weiteren Grundbaustein für Android Apps vor. Fragmente sind Komponente mit eigener Benutzeroberfläche und eigenem Lebenszyklus. Sie sind Bausteine innerhalb von Activities. Sie werden wie Views und View-Groups entweder in Layoutdateien definiert oder per Code erzeugt. Eine Aktivity kann ein, zwei oder mehrere Fragmente aufnehmen. Beispielsweise ermöglichen sie die Wiederverwendung von Teilfunktionen einer Activity und zum Aufbau einer mehrschichtigen UI. UI kann ein ganzer Bildschirm oder nur ein Teil des Bildschirms sein.[2]

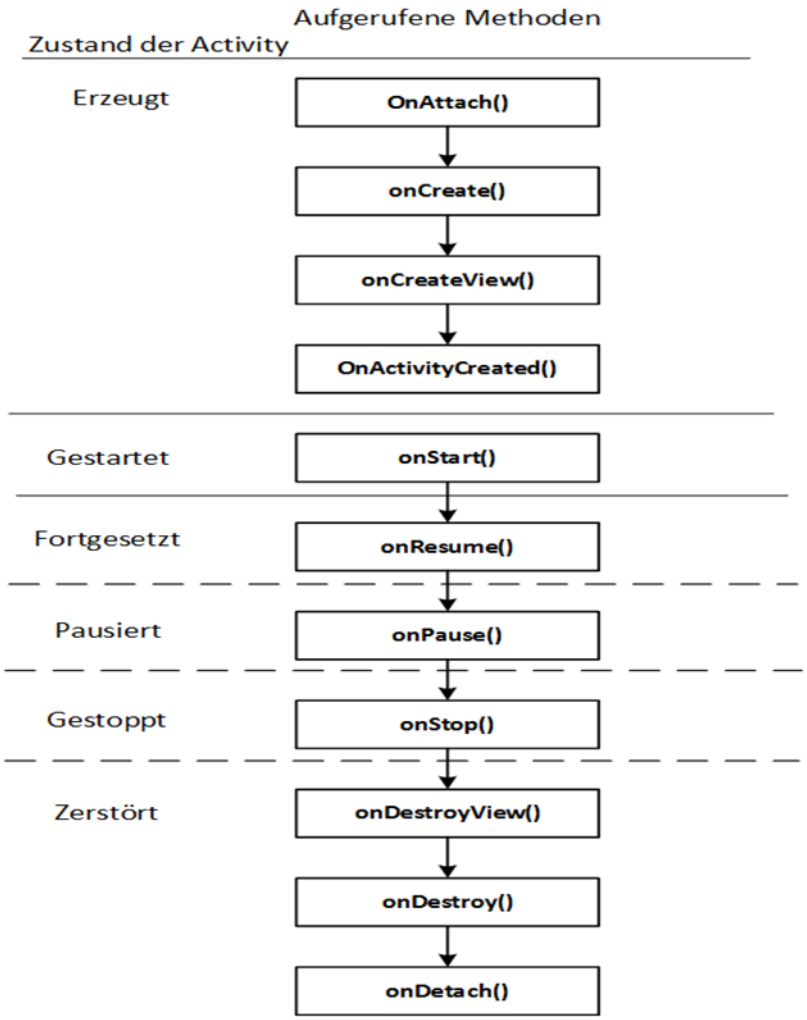


Abbildung 15: Fragment Lebenszyklus [2]

d. Service

Service ist eine Komponente, die im Hintergrund läuft, um langfristige Operationen durchzuführen, ohne dass eine Interaktion mit dem Benutzer erforderlich ist. Service wird zum Beispiel zur Wiedergabe von Musik und zur Abwicklung von Netzwerk-Transaktionen usw. verwendet. Darüber hinaus kann sich eine Komponente an einen Service binden, um mit ihm zu interagieren und sogar Interprozesskommunikation (IPC) durchzuführen. Damit gilt er als ein üblicher Prozess zur Implementierung der Android-Anwendungsframework-API.

Android unterscheidet zwischen gestarteten und gebundenen Services. Vereinfacht ausgedrückt stellen letztere auf diese Weise eine Kommunikationsschnittstelle zur Verfügung. als gestarteter Service. Ein gebundener Service hat einen Lebenszyklus, der untrennbar mit einem oder mehreren ServiceConnection-Objekten verbunden ist. Ein Service wird gebunden, wenn sich eine Anwendungskomponente durch den Aufruf von **`bindService()`** an ihn bindet. Dieser Service bietet eine Client-Server-Schnittstelle, die es den Komponenten ermöglicht, mit dem Service zu interagieren, Anfragen zu

senden und Ergebnisse zu erhalten. Ein gebundener Service bleibt so lange aktiv, bis sich alle seine Clients von ihm lösen (gebundene Service Lebenszyklus befindet sich im (Anhang A8) [8]

e. Broadcast Receiver

Broadcast Receiver ist eine Android-Komponente, mit dem wir Android-Systeme oder Anwendungsereignisse senden oder empfangen können. Boradcast Receiver übernimmt die Kommunikation zwischen dem Android-Betriebssystem und der Anwendung. Er wird verwendet, um Broadcast-Intents zu empfangen, so dass die Anwendung nach Intents Ausschau halten kann, die den gewünschten Kriterien entsprechen. [3]

f. Intents

Intent ist ein leistungsstarkes, anwendungsübergreifendes Nachrichtenübermittlungs-Framework; Intents werden in Android intensiv genutzt. Nachrichten, die zwischen Komponenten verschickt werden, sie erlauben bestimmte Komponenten miteinander zu verbinden, indem Intents ausgesendet und empfanget werden. Android Intents werden auch verwendet, um ein Service, Activity zu starten eine Nachricht zu senden oder eine Webseite anzuzeigen. Es gibt zwei Arten von Intents:
Explicit Intent: explicit Intent spezifiziert die Komponte, in einem solchen Fall Intent die aufzurufende externe Klasse zur Verfügung stellt.
Implizit Intent: Implizit Intent spezifiziert nicht die Komponente. In einem solchen Fall findet das System einen geeigneten Empfänger basierend auf Eigenschaften des Intents. [1]

2.8.4 Die Benutzeroberfläche:

In Android werden alle Objekte der grafischen Benutzeroberfläche (GUI) (Widgets, Layouts, Bild Objekte usw.) von der View-Klasse der GUI-Bibliothek abgeleitet. ViewGroup ist die Basis Klasse für Layouts. Layout definiert die visuelle Struktur für eine Benutzerschnittstelle. ViewGroup ist eine Sammlung von views und anderen untergeordneten views.
Linear Layout z.B. ist der View Group und ordnet einen Button (view) und auch andere Layouts zu.
- LinearLayout: ordnet alle views in einer einzelnen Richtung entweder (horizontal oder vertikal).
-RelativeLayout: ordnet die views relativ zu verwandten Elementen oder relativ zu ViewGroup RelativLayout (Rand)

Android UI Widgets

- TextView: wird verwendet, um dem Benutzer Text anzuzeigen.
- EditText: wird verwendet, um Texteingaben von Benutzern zu erhalten.
- Button: kann angeklickt werden, um eine Aktion auszuführen.
- CheckBox: wird verwendet, wenn den Benutzern eine Gruppe von wählbaren Optionen präsentiert werden, die sich gegenseitig ausschließen.

- Toast: eine Benachrichtigung, die für eine bestimmte Dauer erscheint.
- Listview: android Listview ist eine View, die mehrere Elemente gruppiert und in einer vertikal scrollbaren Liste anzeigt: Listenelemente werden automatisch mit Hilfe eines Adapters in die Liste eingefügt. [13]

2.8.5 Model-View-Controller (MVC)

MVC bezieht sich auf die Aufteilung der verschiedenen Aspekte unserer Anwendung in verschiedene Teile oder Schichten. Android-Anwendungen verwenden üblicherweise das Model-View-Controller-Muster. Das Modell enthalt Logik und den Status der Anwendung sowie Daten, die sie steuern.
View stellt die Benutzungsoberfläche dar. Sie bezieht sich auf alle Widgets und verschiedene Layouts. Alles, was der Benutzer auf dem Bildschirm sehen oder damit interagieren kann, ist normalerweise Teil der View.
Der Controller, der die Synchronisierung zwischen der View und dem Modell verwaltet, interagiert mit beiden und hält sie auch getrennt.
Activity verwaltet die View und verarbeitet Ereignisse.
View wird durch Layout in XML definiert.
Model besteht aus Java Klassen kann auch ein Service sein.[13]

3 Anforderungen

In diesem Kapitel werden alle Anforderungen der Android App erläutert.
Zuerst werden die Anwendungsfälle definiert und in einem Diagramm dargestellt Anschließend werden die Anforderungen im Hinblick auf BLE GAP Rollen, GATT Rollen und ATT Protokoll genauer betrachtet um sie abschließend in einem konzeptuellen Design des User Interface zu verwirklichen. Um die Funktionalität des Gesamtsystems zu beschreiben und Anforderungen zu definieren, müssen zunächst Anwendungsfälle definiert werden: Da unsere App an Kinder und Entwickler (Studenten) gerichtet werden, sind die Anforderungen und die Anwendungsfälle in zwei Benutzer-Modi aufgeteilt.

Anwendungsfalldiagramme sind nützliche Kommunikationswerkzeuge auf hoher Ebene, um die Anforderungen des Systems darzustellen. Die Diagramme zeigen die Interaktion von Benutzern mit dem System, das entwickelt werden wird.

3.1 Kinder-Modus

Kinder-Modus: Das System soll folgende Anwendungsfälle (Use Cases) implementieren:

- Der Benutzer soll nach dem Start der App Bluetooth aktivieren.
- Der Benutzer soll nach verfügbaren Bluetooth Geräten suchen.
- Mit dem gewählten Gerät eine Verbindung herstellen.
- Das Fahrzeug des ATLAS Modells vorwärtsfahren können.
- Das Fahrzeug des ATLAS Modells rückwärtsfahren können.
- Der Benutzer soll die Geschwindigkeit einstellen können.
- Der Benutzer soll das Fahrzeug abbremsen können.
- Der Benutzer soll in der Lage sein akustisches Signal des Fahrzeugs auszulösen.
- Der Benutzer soll Fahrstraßen steuern können.
- Der Benutzer soll LED´s ein/ausschalten können.
- Die App soll Responses (Rückmeldung) von ATLAS Modells als ASCII Format erhalten werden.
- Die Verbindung soll getrennt werden.

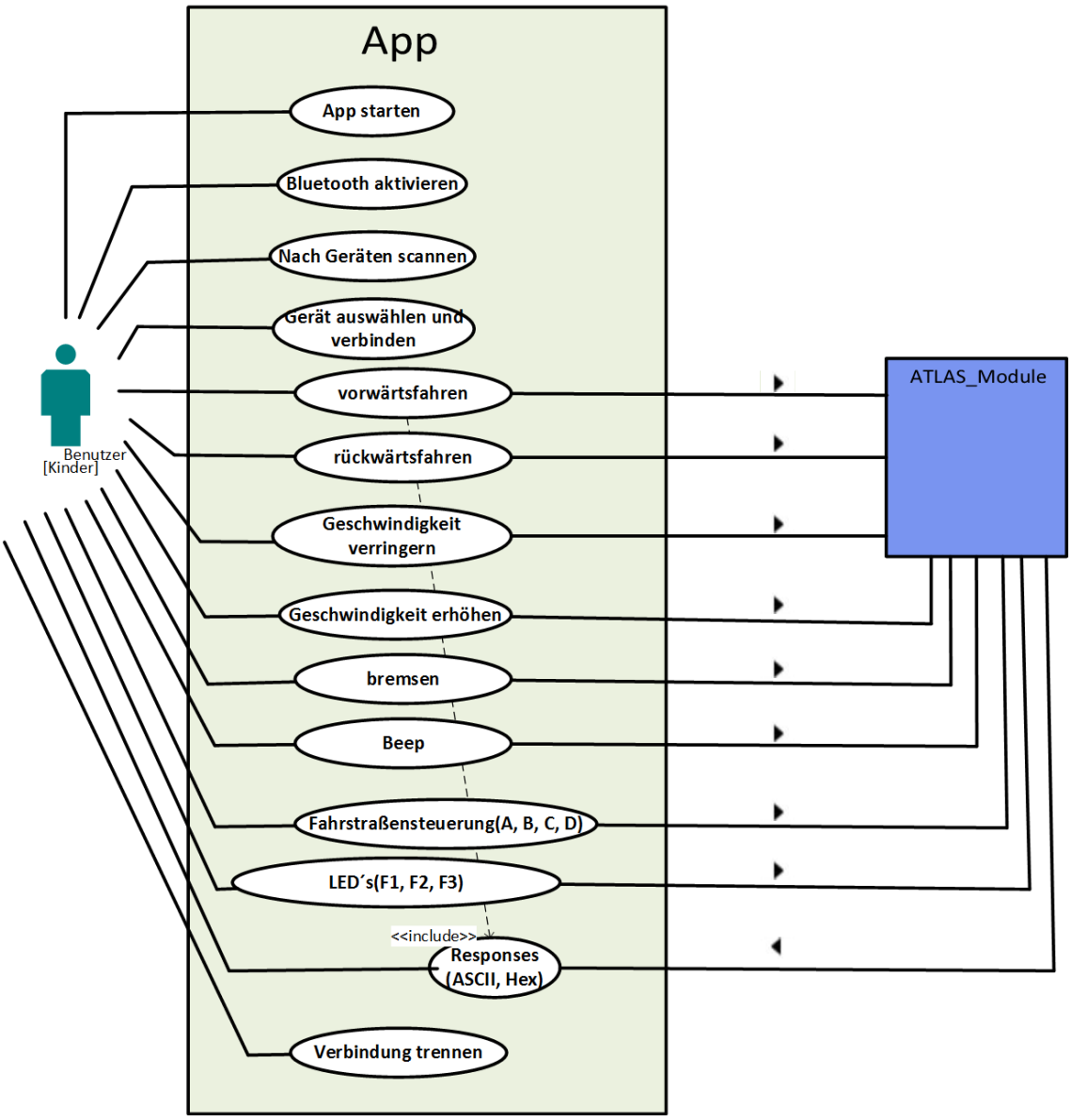


Abbildung 16: Anwendungsfalldiagramm (Kinder_Modus)

3.2 Entwickler-Modus

Entwickler-Modus: zusätzlich zu dem bereits erwähnten Anwendungsfälle vom Kinder-Modus ist dieser Modus erweiterbar und soll folgende Use Cases implementieren:

- Der Benutzer soll beliebige Befehle senden.
- Die Position des Fahrzeugs kann durch Balise ausgelesen werden.
- Der Benutzer soll in der Lage sein, die aktuelle Position speichern können.
- Der Benutzer kann die gespeicherte Position wieder laden.
- Die App soll Responses (Rückmeldung) von ATLAS Modells als ASCII und Hex Format erhalten werden.

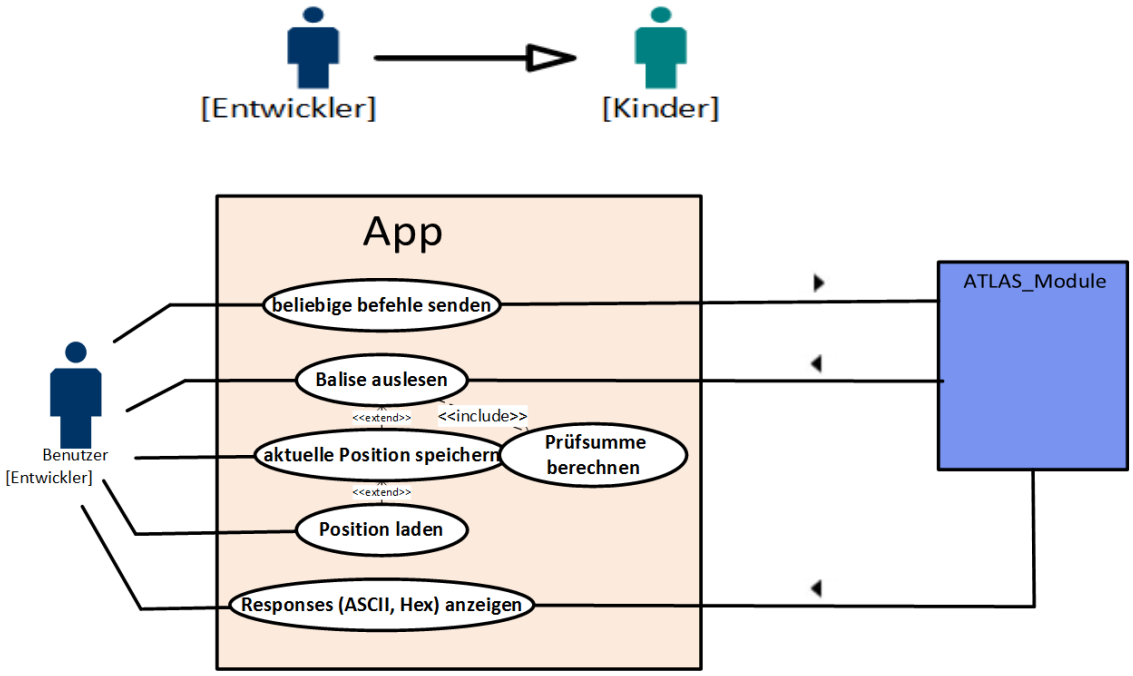


Abbildung 17: Anwendungsfalldiagramme (Entwickler_Modus)

3.3 Verfahren

Unsere App soll folgende Anforderungen erfüllen:

Die fundamentale Kommunikationstechnik, die bei Bluetooth Low Energy (BLE) verwendet wird, ist zuerst eine zentrale und periphere Kommunikation und dann gefolgt von einer Client-Server Architektur GATT Client, der die Daten unter ATT Protokoll vom GATT Server abruft.

Es ist auch wichtig zu wissen, dass das Attributprotokoll atomar und sequenziell im Betrieb ist. Man sollte immer warten, bis die einzelne Operation erfolgreich durchgeführt wurde.

4 Konzept und Design

4.1 Überblick

Nachdem wir Grundlagen erläutert haben und die spezifischen Anforderungen festgelegt haben, wird jetzt das entwickelte Konzept vorgestellt.
Dieses Konzept gilt als Grundlage der Implementierung. Das Ziel aus Kapitel 1 wird spezifiziert. Außerdem werden wir auf technische Details eingehen. Es wird eine BLE Android App entwickelt. Diese App stellt Use Cases auf Basis der Implementierung des ATLAS_BT Moduls dar.

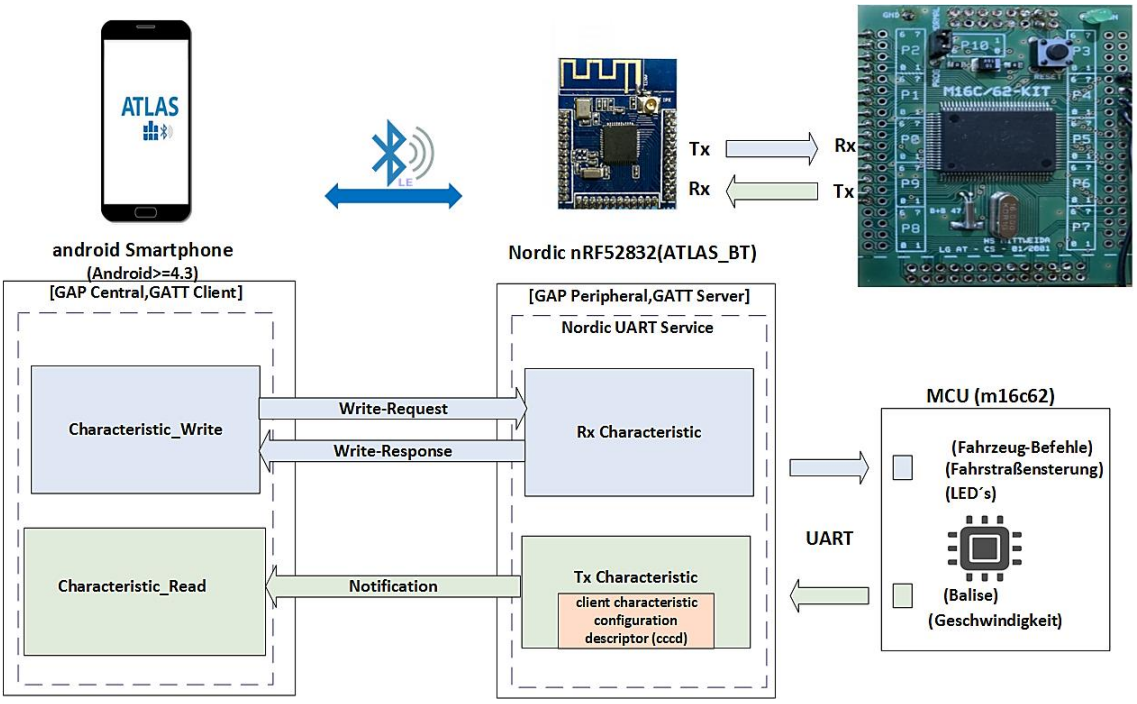


Abbildung 18:ATLAS Konzept: Die App sendet per BLE die Fahrzeugs Befehle und empfangt die Position des Fahrzeugs (Balise)

GATT-Server stellt Services und Characteristics zur Verfügung um eine Kommunikation zu gewährleisten. Hierbei ist das Smartphone der Client und das ATLAS Modul der Server. im Falle der TX Characteristic folgt auf ihre Characteristic-wert-deklaration ein Deskriptor. Der Deskriptor ist ein Attribute vom Typ Client Characteristic Configuration (CCCD). Die UUID sind eindeutige Nummern zur Identifizierung von Services, Characteristic und Deskriptoren. Diese IDs werden über die Luft übertragen, so dass ein Peripheral dem Centrale mitteilen kann, welche Services es zur Verfügung stellt.

Service Name	UUID
Nordic UART Service	6E400001-B5A3-F393-E0A9-E50E24DCCA9E

Characterisctic Name	UUID	Properties
Rx Characteristic	6E400002-B5A3-F393-E0A9-E50E24DCCA9E	Write
Tx Characteristic	6E400003-B5A3-F393-E0A9-E50E24DCCA9E	Notify

Attribute type	UUID	Beschreibung
Client Characteristic Configuration	0x2902	ClientCharacteristicDescriptor(CCCD)

Tabelle 1: ATLAS-Module UUIDs

- Die Daten (Fahrzeug-Befehle) ,(Fahrstraßensteuerung) und (LED`s steuerung) werden von Android an ATLAS-BLE Module gesendet indem mit „Write Request“ jede einzelne Werte in Characteristic_Write geschrieben wird.
-Die Daten (Balise) , (Geschwindigkeit Callbacks) werden von ATLAS_BLE Module an Android gesendet, indem mit „Notification“ Characteristic_Read aktiviert.

4.2 Verbindungsprozedur vom Beginn an

Die Verbindung zwischen beiden Geräten soll über eine BLE Verbindung realisiert werden. Da die Kommunikation mit unserem ATLAS Modul nur über Bluetooth-LE möglich ist, sind hier einige Einschränkungen gegeben. Wir müssen also GAP Rollen definieren und ATT Protokoll befolgen. Der Datenaustausch findet zwischen GATT-Client und GATT-Server statt. Das ATLAS Modul (Peripheral) startet beim Einschalten immer mit dem Advertising, um für alle Geräte in der Umgebung sichtbar zu sein.
Peripheriegeräte werden bei Punkt-zu-Punkt-Verbindungen immer Slave.
Geräte mit der Central Rolle operieren nicht im Advertising Zustand wie beschrieben im Link Layer. Bei einer Punkt-zu-Punkt-Verbindung wird Central immer Master nachdem er die Verbindung mit einem Connection Request imitiert hat.
Nachdem die Verbindung etabliert wird, startet das Senden bzw. Empfangen von Data PDU über die Datenkanäle. (je nach Property oder ATT Methode) (Paket-Format befindet sich im Anhang A5)

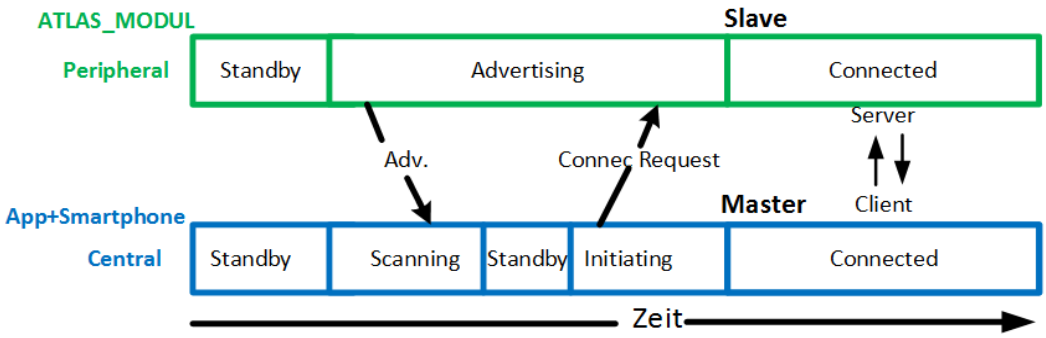


Abbildung 19: Verbindungsprozedur vom Beginn an

4.3 BLE-UART-Kommandos

Für unsere App müssen wir also zwei Characteristic UUIDs definieren:
UUID_CHARACTERISTIC_WRITE: 6E400002-B5A3-F393-E0A9-E50E24DCCA9E
UUID_CHARACTERISTIC_READ: 6E400003-B5A3-F393-E0A9-E50E24DCCA9E
um eine Synchronisation in Reihenfolge mit den Characteristics: Rx Characteristic, Tx Characteristic vom ATLAS_Modul durchzuführen.
Die Rx-Characteristic kann unter die UUID_CHARACTERISTIC_WRITE angesprochen werden. Diese Characteristic kann geschrieben werden (Property Write).
Mit der write- request muss sie zuerst geschrieben werden und dementsprechend wird diese Response ausgelesen. Der Client (App) kann durch diese Characteristic je nach Wunsch einzeln Befehle senden. (einzelne Response als Rückgabe) Anhand UUID_CHARACTERISTIC_WRITE definieren wir also diese Fahrzeugs Befehle und diese Fahrstraßensteuerung (hex Format)

4.3.1 Fahrzeug-Befehle

- VORWAERTS (0x10)
- RUECKWAERTS (0x20)
- SPEED (−) (0x30)
- SPEED (+) (0x40)
- BREMSEN (0x50)
- Beep (0x90)

4.3.2 Fahrstraßensteuerung

- Fahrstraßensteuerung-A (0xF0)
- Fahrstraßensteuerung-B (0xF1)
- Fahrstraßensteuerung-C (0xF2)
- Fahrstraßensteuerung-D (0xF3)

4.3.3 LEDs-Steuerung

- Rote LED-F1 (0x60)
- Orange LED-F2 (0x70)
- Grüne LED-F3 (0x80)

oder beliebige Data bis 20 bytes(Hex oder ASCII) senden.

4.3.4 Balise

Analog die Tx Characteristic kann unter UUID_CHARACTERISTIC_READ angesprochen werden sie enthält einen Descriptor. (Property Notify) durch diese Characteristic sendet das Atlas_Modul„Notification“ an die App (Client) .Die App empfängt automatisch diese Daten ohne Bestätigung. Vorher muss man Client Characteristic Configuration Descriptor (CCCD) aktivieren Diese Daten sind Position des Fahrzeugs (Balise vorne und hinten) sie werden als ASCII im Littel_Endian_Format gelesen (LSB zuerst) (siehe Abb 20)

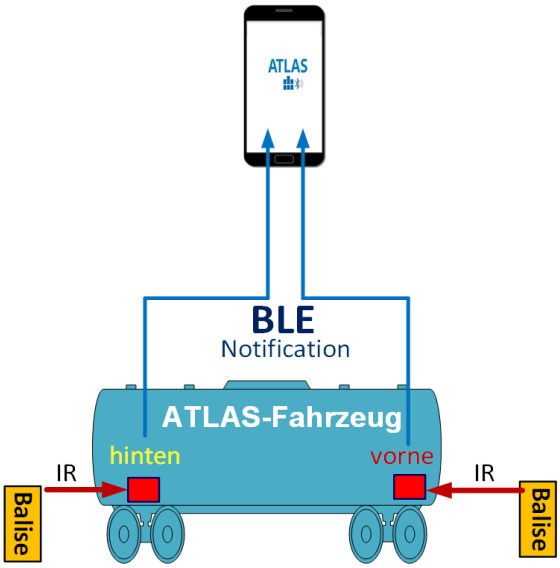


Abbildung 20: Schema des Kommunikationprotokolls(Balise —>App)

4.4 App Design

In diesem Abschnitt wird eine vereinfachte Darstellung der App vorgestellt.
Die App (Kinder-Modus) soll eine kinderleichte Bedienung haben.
Die App soll den Benutzer informieren, falls einen Verbindungsverlust mit dem Atlas-Modul gibt.
Die App soll Name, bzw. Mac Adresse und RSSI des Atlas-Moduls anzeigen.
Die App soll eine Authentifizierung (Username und Passwort) verlangen beim Auswählen des Entwickler Modus.
Im Folgenden wird die Struktur der Benutzeroberfläche (GUI) durch dieses Flussdiagramm gezeigt:

Abbildung 21: Flussdiagramm der App (UI)

In den folgenden Abschnitten werden Entwicklungsumgebung und grundlegender Aufbau einer App Implementierung von Benutzeroberflächen vorgestellt.

4.4.1 Android Studio

Android-Anwendungen wurden in der Vergangenheit mit der integrierten Entwicklungsumgebung (IDE) Eclipse mit dem Android Development Tools (ADT)-Plugin entwickelt. Google führte jedoch im Jahr 2014 Android Studio als offizielle IDE für die Entwicklung von Android-Anwendungen ein, und diese IDE wurde zum offiziellen Standard. Android Studio basiert auf der IntelliJ IDEA-Plattform und erbt die meisten Funktionen von Intellij´s core.

Java neben Kotlin sind als Standard für die Entwicklung von Android-Apps gewählt worden. Sie basieren aber auf einer abweichenden Klassenbibliotheks-API. Java wird in Kombination mit dem Android Software Development Kit (SDK) in der Android Studio-

Umgebung zur Entwicklung von Anwendungen verwendet.
Android-Studio ermöglicht eine optimierte eigene Benutzeroberfläche zu entwerfen und paar Funktionen einzubauen, damit die App die Aufgaben ausführen kann.

4.4.2 Vorbereitung der Android Studio-Umgebung

Damit die Android Studio-Entwicklungsumgebung ordnungsgemäß funktionieren kann, muss das Referenzdesign von ATLAS-CAR App mit dem Ziel -API-Ebene von Android, auf der die App ausgeführt werden soll, in der gleichen Konfiguration sein. Dies erfordert die Installation der richtigen API-Tools und anderer unterstützender Dateien mit dem SDK Manager. Man muss auch in Betracht ziehen, dass die minimale Android-API-Ebene, die ein Android-Gerät zum Ausführen unserer BLE App benötigt größer als Version 18 sein muss.

4.4.3 Anschließen des Android Smartphone an den PC

Bevor das Smartphone an den PC angeschlossen wird, muss als erstes der USB-Debugging-Modus im Menü "Entwickler-Optionen" des Telefons aktiviert werden.
Der USB-Debugging-Modus ermöglicht es, die 'App'-Datei von der Android Studios IDE auf das Smartphone zu übertragen.
Sobald das Gerät für den USB-Debugging-Modus eingerichtet ist, installieren wir auch mit dem SDK Manager den Google USB-Treiber, welcher für Windows benötigt wird, wenn ein adb-Debugging erwünscht. Android Debug Bridge (ADB) ist eine Client-Server-Anwendung, die eine Verbindung zwischen dem Computer und virtuellen und physischen Android-Geräten herstellt. Mit der können wir Dateien kopieren, kompilierte Anwendungspakete (.apk) installieren und Shell-Befehle ausführen. Anschließend kann unser Smartphone über das USB-Kabel an den PC angeschlossen und verbunden werden. [13]

4.5 Struktur und Aufbau der App

Eine Anwendung besteht aus mehr als nur Java-Datei:
src/main/AndroidManifest.xml: Diese Datei enthält die Konfigurationsparameter des Projekts wie Berechtigungen (Permissions), Services und zusätzliche Bibliotheken. **src/main/java**: In diesem Ordner befinden sich in der Java geschriebenen Quellcodedateien.
src/res/: Unter Ressourcen versteht man grundsätzlich alle benötigten Dateien mit Ausnahme des Quellcodes. Medien-, Bild- und Layoutdateien: Layoutdateien in Android bestehen aus UI-Elementen/Views, Values, Colors, logo … die im XML-Format definiert sind, die sich im Ressourcen-Ordner befinden, werden über Java-Code in Quellcode zugegriffen.

In Android werden alle Objekte der grafischen Benutzerschnittstelle (GUI) (Widgets, Layouts, Bild Objekte usw.) von der View-Klasse der GUI-Bibliothek abgeleitet. Die Komponenten der GUI einer Android-App haben die folgenden Basiseigenschaften: Jede View belegt einen rechteckigen Bereich des Bildschirms. Ihre Position und Größe werden durch Layouts bestimmt. Diese wiederum erben von android. view.ViewGroup, die

ebenfalls ein Kind von View ist. Die Unterklassen RelativeLayout, LinearLayout von ViewGroup sind spezielle views, die andere views und Komponente enthalten können.

Android-Apps werden vom Gradle und Android SDK kompiliert und in einer Apk-Datei verpackt (Android Applikation Package) und kann auf einem Android-Gerät installiert werden kann.
Um eine Android-Anwendung zu entwickeln, brauchen wir sogenannte Android Platform APIs. Diese API sind fest definierte Methoden (Funktionsaufrufe). Über sie kann unsere App Zugriff auf verschiedene Betriebsystemresourcen nehmen. Da die Android Plattform weiterentwickelt wird und immer neue Android-Versionen erscheinen werden, entspricht jede Android-Version einer einzelnen Android-API-Ebene.

4.5.1 Java-Dateien

Die Java-Dateien liefern die logischen Prozesse, in denen die Aktionen stattfinden. Diese Dateien werden zusammen mit den XML-Dateien verwendet, um die Benutzerschnittstelle (GUI) auf den Bildschirm zu bringen. Das Java-Paket mit dem Quellcode der Anwendung enthält folgende Klassen:

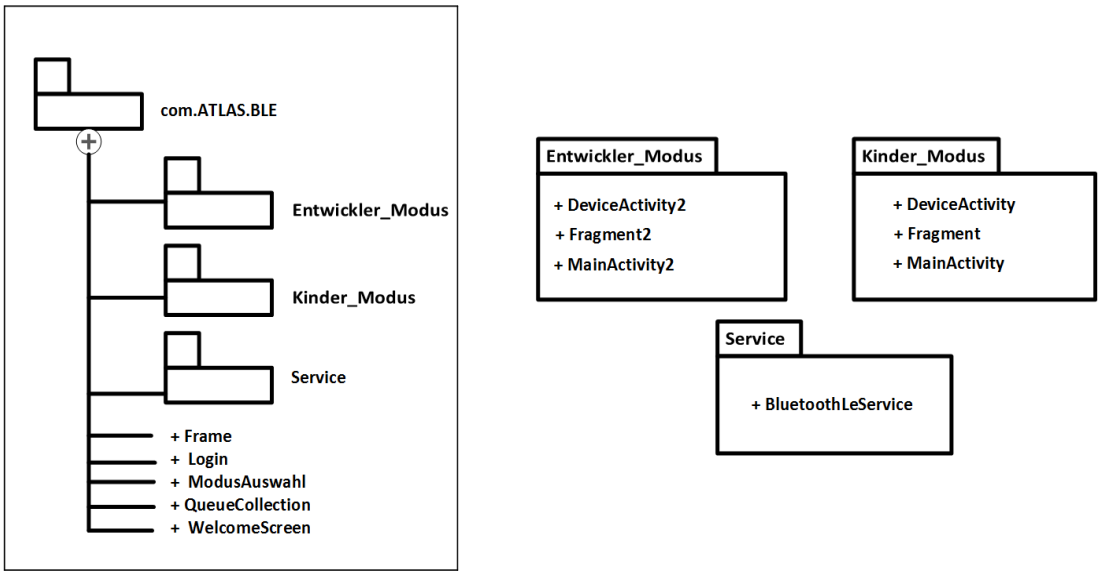


Abbildung 22: Pakethierarchie: com.ATLAS.BLE

(Paketdiagramm UML befindet sich im Anhang A6)

Datei Name	Aufgabe
WelcomeScreen.java	Activity zeigt ATLAS Logos und verschwindet in vier Sekunden.
ModusAuswahl.java	Activity zum Modusauswahl und zum Prüfen von Dialog Fenster Data (Benutzername und Passwort)
Login.java	Dialog Fenster wird aufgerufen beim Anklicken der Entwickler_Modus Button zum Eingeben der Login Data
DevicesActivity2.java	Activity zum Scannen und zum Verbindungsherstellen
MainActivity2.java	Activity zum Empfangen bzw Lesen und Verarbeiten von Data.
BluetoothLeService.java	Service, der über die Android BLE-API mit dem BLE-Gerät interagiert.
Fragment2.java	Fragment zum Senden von Data bzw. Fahrzeugs Befehle
QueueCollection.java	Klasse, die LinkedList Objekt zu der seriellen Bearbeitung der Daten erzeugt.
Frame.java	Klasse wo Enum Fahrzeug, Fahrstraße und LEDs als byte Array definiert worden.

Tabelle 2: App Java Klassen (Entwickler)

(Klassendiagramm UML_Entwickler_Modus befindet sich im Anhang A7)

4.5.2 xml-Dateien:

Die XML-Dateien behandeln die grafische Benutzeroberfläche (GUI). Diese Dateien bestimmen das Design und das Aussehen der Bedienoberfläche und die Platzierung der Daten auf dem Bildschirm, wenn die "App" auf dem Smartphone läuft.
Die Tabelle unten ist eine Liste der in diesem Projekt wichtigsten verwendeten XML-Dateien für (Entwickler-Modus).

Datei Name	Aufgabe
AndroidManifest.xml	Präsentiert wesentliche Informationen über die App für das Android-System.
activity_devices2.xml	LinearLayout mit Listview und Button für (DevicesActivity2)
activity_welcome.xml	LinearLayout mit ImageView + TextView für (WelcomeScreen)
activity_modusaus-wahl.xml	LinearLayout mit Zwei Button für (ModusAuswahl)
activity_main2.xml	Viewpager für Fragment
fragment_2.xml	Mehrere LinearLayout mit Buttons und Textview für (Fragment2)
layout_dialog.xml	Relativelayout mit Zwei Textview für (Login)
listitem_device.xml	Linearlayout mit 3 Textview und eine Checkbox.
menu_connected.xml	Menu mit connect und disconnect Button
menu_devices.xml	Menu mit scan und Stop Scan Button
Colors.xml	Ressource, die Farbwerte besitzt.

Tabelle 3: App XML-Dateien (Entwickler)

4.5.3 Sequenzdiagramm

Die Sequenzdiagramme zeigen die Interaktion zwischen den oben genannten Komponenten (Entwickler_Modus)

Welcome Screen:

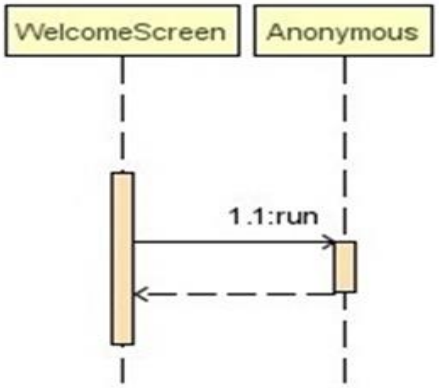


Abbildung 23: Sequenzdiagramm (WelcomeScreen)

Entwickler_Modus + Login:

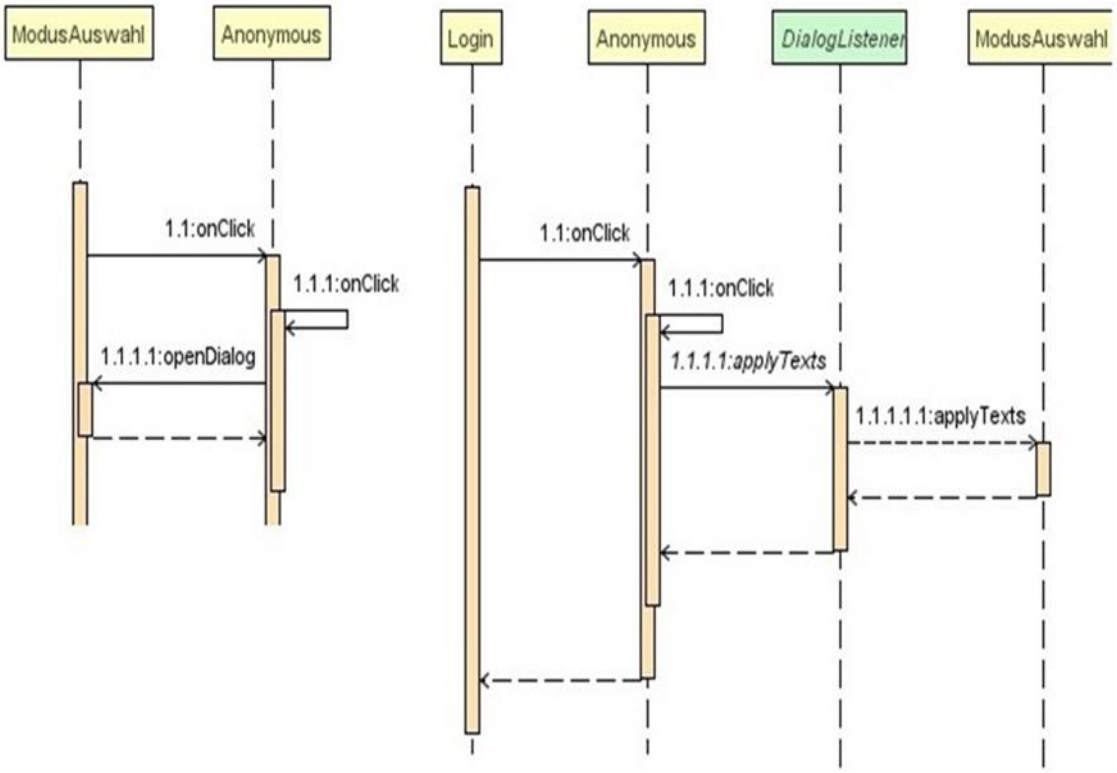


Abbildung 24: Sequenzdiagramm (Entwickler-Modus + Login)

BLE Geräte suchen:

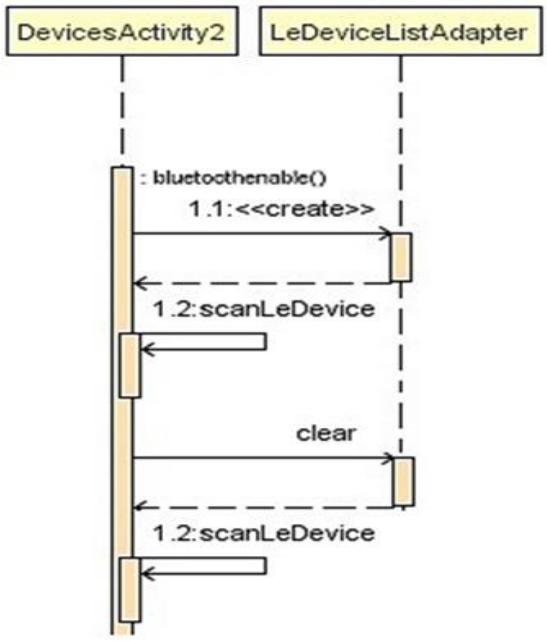


Abbildung 25: Sequenzdiagramm (Scan)

Verbindung-Initialisieren:

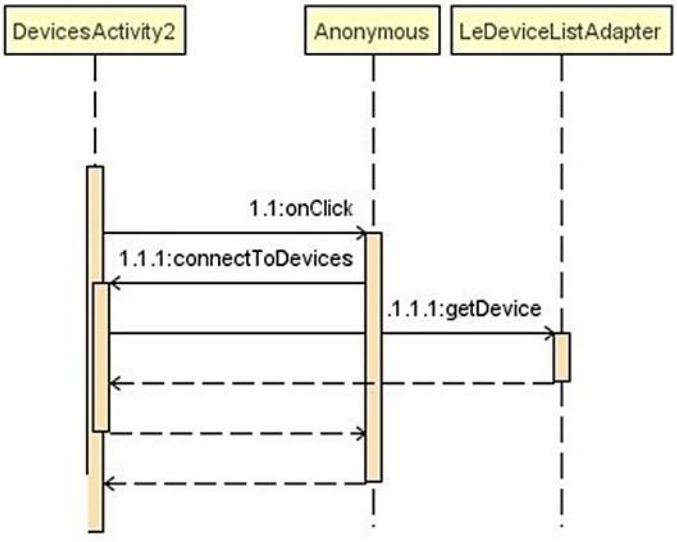


Abbildung 26: Sequenzdiagramm (Verbindung-Initialisieren)

Verbinden mit dem BluetoothLeService:

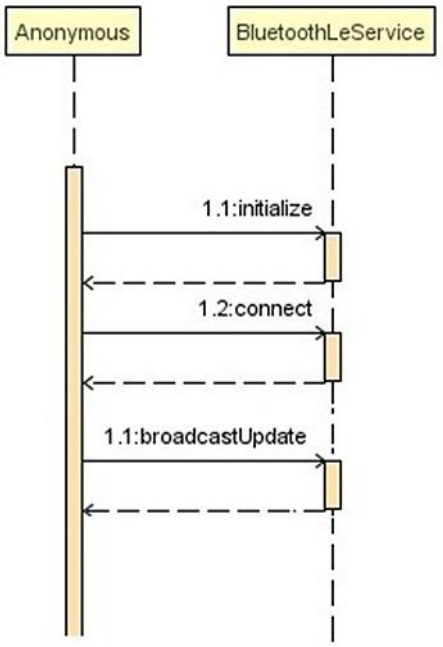


Abbildung 27: Sequenzdiagramm (Verbindung mit BluetoothLeService)

Data senden:

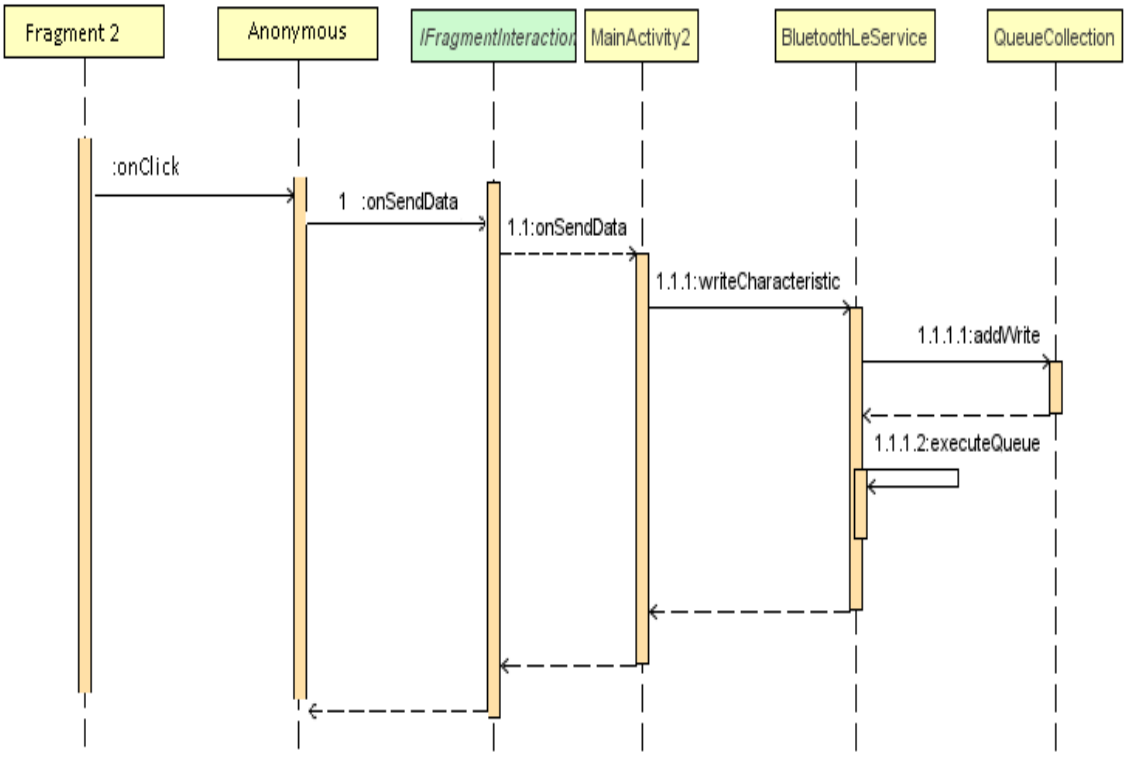


Abbildung 28: Sequenzdiagramm (Senden von Data)

Notification erhalten:

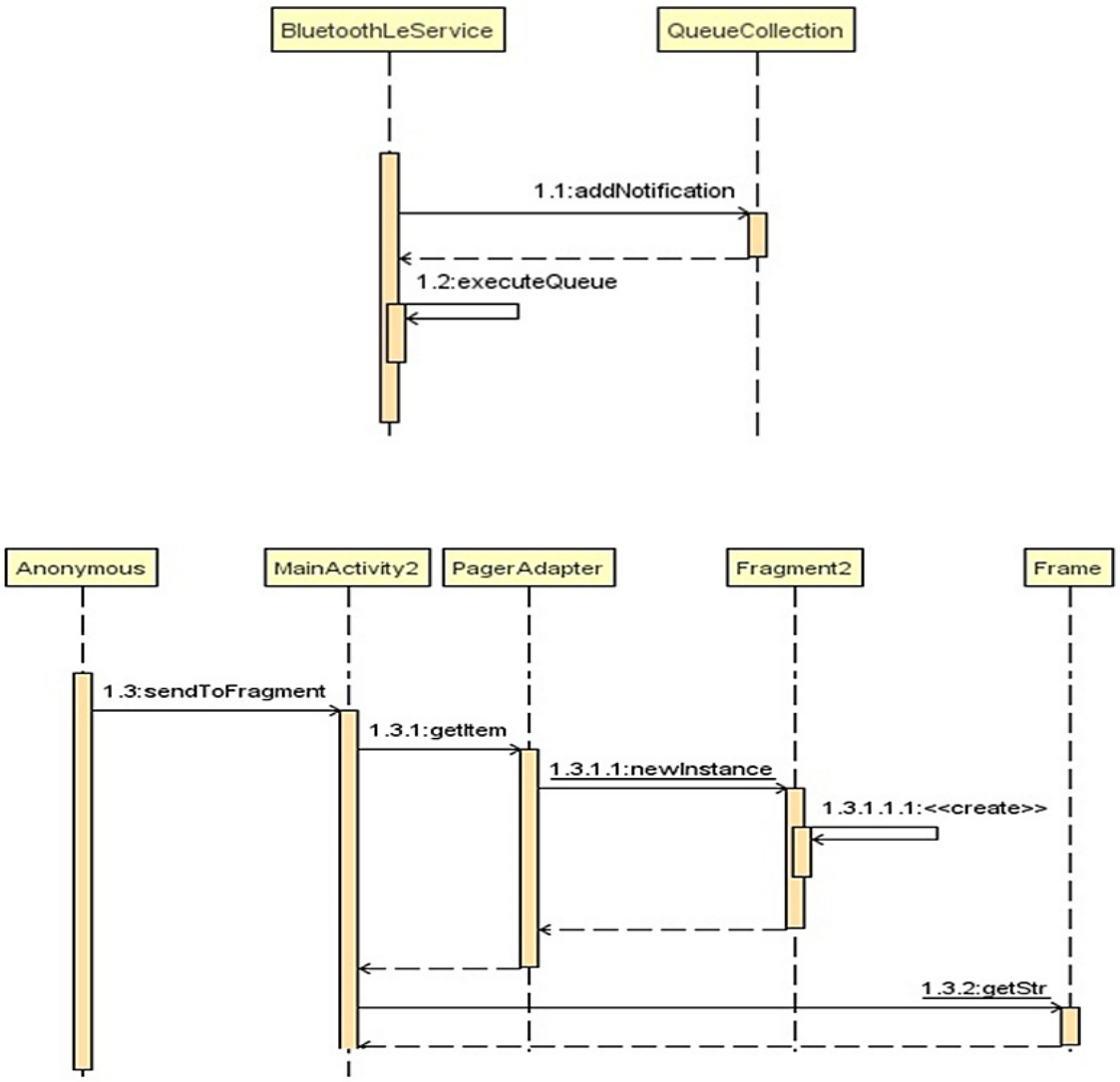


Abbildung 29: Sequenzdiagramm (Erhalten von Notification)

Verbindung trennen:

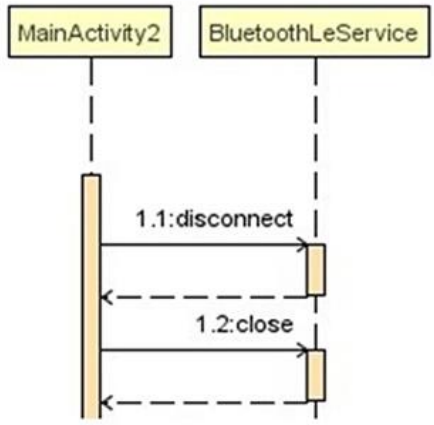


Abbildung 30: Sequenzdiagramm (Trennen von Verbindung)

4.5.4 Wechsel zwischen Aktivitäten

Unsere App besteht aus mehreren Aktivitäten, um von einer Activity zu eine zu wechseln müssen wir explizit-Intent implementieren. z.B. von einem Wechsel von WelcomeScreen zu ModusAuswahl :

```
Intent homeIntent = new Intent(WelcomeScreen.this, ModusAuwahl.class);
startActivity(homeIntent);
```

Wir initialisieren Intent-Objekt homeIntent die Intent-Klasse hat einen Konstruktor, der zwei Argument benötigt. Das erste ist ein Verweis auf die aktuelle Aktivity, das zweite Argument ist der Name der Aktivity, die wir öffnen wollen, ModusAuswahl.class Die class am Ende von ModusAuswahl entspricht dem vollständigen Namen der Aktivity, wie er in der Datei AndroidManifest.xml deklariert ist.

5 Umsetzung

In diesem Kapitel werden die notwendigen Arbeitsschritte zur Umsetzung vorgestellt. Diese App wurde mit der Android Studio und mit der Programmiersprache JAVA entwickelt. In diesem Kapitel werden wir auch das Bluetooth API verwenden und die Interaktion zwischen App Komponenten beschreiben.
Es wurde eine Applikation unter dem Namen **„ATLAS-Car**“ für das Android entworfen und umgesetzt.
Wie bereits erläutert muss für diese BLE Kommunikation zwischen dem Smartphone und ATLAS_BT Modul, Bluetooth-Protokoll‘ implementiert werden. Zum Einsatz kam das Bluetooth-Application Programming Interface (API) aus dem Android-Software Development Kit (SDK) Packet:

5.1 android.bluetooth

Mithilfe der Bluetooth-API können Anwendungen nach anderen Bluetooth-Geräten in eine Reichweite von 50 m im Außenbereich und 10 m im Innenbereich. suchen, eine Verbindung zu diesen Geräten herstellen und die Verwaltung der Datenübertragung. Die Bluetooth-APIs lassen auch App als GATT-Client oder als GATT-Server BLE fungieren. Diese API bietet Klassen, die die vorher genannten Bluetooth-Funktionalität verwalten Es gibt Basis Klassen und Interfaces, die für jede BLE Basis Implementierung bei jeder BLE App erforderlich sind.
Für die Entwicklung unsere BLE App, benötigen wir diese Klassen bzw. Interface aus dem **android.bluetooth** APIs Package:

1. **BluetoothManager**

Ein High Level Manager wurde für die Instanz eines BluetoothAdapters und für das gesamte Bluetooth-Management eingesetzt. Die BluetoothManager Klasse ist eine Instanz des BluetoothServices, der auf Android-Geräten läuft.

2. **BluetoothAdapter**

Diese Klasse repräsentiert den Bluetooth-Adapter des lokalen Geräts. Sie ist für die Interaktion mit dem Radio zuständig.
BluetoothAdapter.LeScanCallback: Schnittstelle, die verwendet wird, um LE-Scanergebnisse zu liefern.

3. **BluetoothGatt**:

Diese Klasse bietet Bluetooth-GATT-Funktionalität, um die Kommunikation mit Bluetooth-Modul zu ermöglichen. Die BluetoothGatt-Instanz wird verwendet, um GATT-Client-Operationen durchzuführen (Service discovery, lesen und schreiben der

Characteristic), und das Ergebnis dieser Operationen wird als BluetoothGattCallback zurückgegeben.

4. BluetoothGattCallback

Diese abstrakte Klasse wird verwendet, um BluetoothGatt-Rückrufe zu implementieren.
-onConnectionStateChange: Callback Methode, die anzeigt, wenn ein GATT-Client eine Verbindung/Trennung zu/von einem entfernten GATT-Server erfolgt ist.
-Diese Vier Callback Methoden werden auch aus dem BluetoothCallback-Klasse generiert werden:
-onCharacteristicWrite: Ereignet sich, wenn GATT-Server eine Anfrage von einem Client-Gerät zur Bearbeitung der Daten einer Charcteristic erhält. Dieser Callback liefert das Ergebnis einer Characteristic Schreiboperation. Der Handler von onCharacteristicWrite erhält die folgenden Parameter:
+ Der Sender ist der GATT-Server
+ Characteristic die abgefragt wurde
+ Status: Der Ausgangstatus der Schreibanfrage.
-onServicesDiscovered: Ereignet sich, wenn die Liste der Services, die ein Gerät bietet, entdeckt wird und wenn Characteritic und Deskriptoren für das Server aktualisiert wurde.
-onCharacteristicChanged: ereignet sich, wenn die Notification auf einem Characteristic des Servers Entstand.
-onDescriptorWrite: ereignet sich, wenn die Daten eines Deskriptors von dem Server auf ihn geschrieben werden. Wir verwenden writeDescriptor, um den Schreibprozess zu starten. Der Ereignis Handler von onDeskriptorWrite erhält die folgenden Parameter:
+Sender ist das Remote-Gerät (ATLAS Module), das den Deskriptor bereitstellt.
+Der Deskriptor, der geschrieben wurde.
+Status ist der Ausgangsstatus des Schreibvorgangs.

5. BluetoothGattService

Gatt Service enthält eine Sammlung von BluetoothGattCharacteristic. Diese Klasse gibt dem Entwickler Flexibilität, um die Characteristics des Services zu erhalten.

6. BluetoothGattCharacteristic

Diese Klasse repräsentiert Bluetooth GATT-Characteristic sie enthält Funktionalität und Parameter eines Characteristics und auch die Verfahrenzum Lesen und Schreiben der Daten.

7. BluetoothGattDescriptor

GATT-Deskriptoren enthalten zusätzliche Informationen und Attribute eines GATT-Characteritics in unserem Fall für das CCCD um Characteristic Notification zu aktivieren.

5.2 Ausführungsprozess der Klassen

Da unsere Android-Programmierung auf Java aufgebaut und damit eine objektorientierte Programmierung ist. wird in diesem Abschnitt mit Klassen, Methoden und Objekten umgegangen. Der Ausführungsprozess der Klassen für unsere BLE-Entwicklung in Android

siehst wie folgt aus:
Bluetooth Permissions:
Für das Scannen benötigt die App eine Berechtigung vom User, um auf Bluetooth Hardware zuzugreifen und diese nutzen zu können. Diese erforderlichen Permission wurden in Android Manifest.xml Datei hinzugefügt.

Mit BluetoothAdapter.startLeScan: starten wir einen Scan für Bluetooth LE-Geräte.
In der Rückruffunktion (Callback) vom Scannen, erhalten wir das BluetoothGeräte-Objekt, und verwenden BluetoothAdapter.stopLeScan, um das Scannen zu beenden. Mit dem Öffnen eines Threads, können wir gescannte Bluetooth-Gerät in der ListView durch Adapter anzeigen lassen. Sobald wir Peripheriegeräte gefunden haben, werden Verbindungen und Kommunikation durch die Verwendung von Generic Attribute Profiles (GATT) koordiniert. Um eine Verbindung mit dem Peripheral herzustellen, erzeugen wir einen BluetoothGattCallback Instanz und rufen BluetoothDevice.connectGatt auf, um BluetoothGatt Objekt zu erhalten.

```
private BluetoothGatt mBluetoothGatt;
mBluetoothGatt = device.connectGatt(this, false, mGattCallback);
```

Die zurückgegebene BluetoothGatt-Instanz kann zur Ausführung von GATT-Operationen auf dem Peripheriegerät verwendet werden. Beim Überschreiben des Handlers **onConnectionStateChanged** können wir verfolgen, wann die Verbindung erfolgreich hergestellt wurde.

```
        @Override
public void onConnectionStateChanged(BluetoothGatt gatt, int status,
int newState) {

String intentAction;
   if (newState == BluetoothProfile.STATE_CONNECTED) {

        intentAction = ACTION_GATT_CONNECTED;
        mConnectionState = STATE_CONNECTED;
        broadcastUpdate(intentAction);

// Nach erfolgreicher Verbindung versucht, Service zu finden.

        mBluetoothGatt.discoverServices();
} else if (newState == BluetoothProfile.STATE_DISCONNECTED) {
        intentAction = ACTION_GATT_DISCONNECTED;
        mConnectionState = STATE_DISCONNECTED;
        broadcastUpdate(intentAction);
}
```

Bei der Implementierung von **BluetoothGatt.discoverServices** als asynchrone Operation (boolean), erhalten wir den Status in der Callback-Funktion **onServicesDiscovered,** indem wir bestimmen, ob der Status gleich **BluetoothGatt.GATT_SUCCESS** ist, um festzustellen, ob das Auffinden des Services erfolgreich

ist wenn dies erfolgreich ist, erhalten wir BluetoothGattService über **BluetoothGatt.getService.** dann BluetoothGattCharakteristic beim Abrufen **BluetoothGattService.getCharacteristic**. Anschließen holen wir BluetoothGattDescriptor über **BluetoothGattCharacteristic.getDescriptor.**

```
 @Override
public void onServicesDiscovered(BluetoothGatt gatt, int status) {
BluetoothGattCharacteristic characteristic = gatt.get-
Service(UUID.fromString(UUID_SERVICE)).getCharacteristic(UUID.from-
String(UUID_CHARACTERISTIC_READ));
gatt.setCharacteristicNotification(characteristic, true);
BluetoothGattDescriptor descriptor = characteris-
tic.getDescriptor(UUID.fromString(UUID_DESCRIPTOR));
Descriptor.setValue(BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE)
;
mBluetoothGatt.writeDescriptor(descriptor);

    if (status == BluetoothGatt.GATT_SUCCESS)
    broadcastUpdate(ACTION_GATT_SERVICES_DISCOVERED);

}
```

(Programmablaufplan von GATT Discovery befindet sich im Anhang A9)

Um die Notification zu erhalten, müssen wir die **setCharacteristicNotification** Methode auf dem BluetoothGatt objek aufrufen. Die Notification für geänderte Werte werden an die **onCharakteristicChanged** callback innerhalb des BluetoothGattCallbacks gesendet:

```
 @Override
public void onCharacteristicChanged(BluetoothGatt gatt, BluetoothGat-
tCharacteristic characteristic) {
broadcastUpdate(ACTION_DATA_AVAILABLE, characteristic, QueueCollec-
tion.TYPE_NOTIFICATION);
executeQueue();
```

5.3 Verbindung Layout zum Code

In diesem Abschnitt werden die Anwendungsfälle aus Kapitel 3 erneut aufgegriffen und mit den Widgets in einer Tabelle zuordnet. Außerdem erläutern wir, wie dies Interaktion zwischen unserem Code und unserer Benutzeroberfläche funktioniert.
Das Funktionsprinzip ist folgendes: Wenn wir jedes Mal z.B. ein Widget in unserer UI hin fügen, fügen wir tatsachlich ein Java Objekt hinzu, auf das wir mit einer entsprechenden Referenz in unserem Code zugreifen können.

In der Methode **`onCreate()`** wird die Benutzeroberfläche mit **`setContentView()`** erzeugt und angezeigt. Widgets und Layouts werden in echte Java-Objekte auf dem Heap verwandelt.
Für jedes Element mit android:id-Attribut wird eine Konstante R. id.name erstellt. Jedes UI-Element, dessen id-Eigenschaft festgelegt ist, kann seine Referenz mit der Methode **`findViewById()`** aus dem Heap abgerufen werden.

Use Case	Widgets type	android:id
Entwickler Modus auswählen	Button	buttonEntwickler
Kinder Modus auswählen	Button	buttonKinder
UserName+Passwort	EditText	edit_username edit_password
Scannen + Verbindung stoppen trennen	Menu mit Zwei Items Menu mit Zwei Items	menu_scan +menu_stop menu_connect +menu_disconnect
erzeugen einer Liste zum Anzeigen von BLE Geräten	ListView	lvDevices
Geräte Name + RSSI +Adresse	TextView	device_name+device_rssi
Gerät auswählen	Checkbox	cbSelect
Verbinden	Button	btnConnect
vorwärtsfahren	Button	btnVor
rückwärtsfahren	Button	btnRueck
Geschwindigkeit verringern	Button	btnDec
Geschwindigkeit erhöhen	Button	btnInc
bremsen	Button	btnBremsen
hupen	Button	btnHupen
Fahrstraßensteuerung (A, B, C, D)	Button	buttonA buttonB buttonC buttonD

LED´s Steuerung (F1, F2, F3)	Button	buttonF1 buttonF2 buttonF3
Responses anzeigen	TextView	textView
Balise auslesen	TextView	textViewUART1 textViewUART2
Aktuelle Position speichern	Button	btnSave
Position Laden	Button	btnLoad
Geschwindigkeit fort-schrittsanzeiger (wird nicht mehr benötigt)	Progressbar + TextView	circularProgressbar + tv1

Tabelle 4: Verhältnis Use Case-Widgets

Nachdem wir eine Referenz auf alle unsere Buttons-Objekte, Listview und TextViews haben, können wir damit beginnen, ihre Methoden zu verwenden.

Button:

Wir verwenden die **`setOnClickListener`**-Methode für jeden der Button-Referenzen.Durch das Registrieren eines **`View.OnClickListeners`** (anonyme Klasse) mit **`setOnClickListener()`** können wir Klicks des Buttons an unsere **`onClick`**-Methode weiterleiten.

ListView:

Wir können ein einzelnes ListView-Widget zu unserem UI-Layout hinzufügen und dann über einen Adapter mit ihm interagieren.: Wir werden die Klasse BaseAdapter verwenden, sie erweitern, anpassen und dann zur Steuerung der Daten von ArrayList und zur Wiedergabe in ListView verwenden: Wir erzeugen eine Instanz von BaseAdapter Klasse und rufen eine Referenz auf ListView ab und binden wir diese an unsere Adapterinstanz.

```
private void setListAdapter(BaseAdapter baseAdapter) {
        ListView lvDevices = (ListView) findViewById(R.id.lvDevices);
        lvDevices.setAdapter(baseAdapter);
        lvDevices.setOnItemClickListener(this);
    }
```

Wir müssen BaseAdapter erweitern und dafür sorgen, dass er mit unseren Daten und ListView funktioniert.Adapter für Geräte, die durch Scannen gefunden werden können.

```
private class LeDeviceListAdapter extends BaseAdapter {
        private ArrayList<BluetoothDevice> mLeDevices;
        private LayoutInflater mInflator;
    public LeDeviceListAdapter() {
        super();
```

```
        mLeDevices = new ArrayList<BluetoothDevice>();
        mInflator = DeviceScanActivity.this.getLayoutInflater();
        }
.. .. ..
// Wir müssen vier Methoden überschreiben und ein wenig eigenen Code hinzufügen.
@Override
    public int getCount() {
          }
        @Override
        public Object getItem(int i) {
        }
        @Override
        public long getItemId(int i) {
        }
        @Override
        public View getView(final int i, View view, ViewGroup viewGroup)
{
…..
}
```

Schließlich können wir ListView und Array List miteinander verbinden und automatisch Bluetooth Geräten beim Scannen anzeigen.

5.4 Bind MainActivty mit dem Service

In diesem Abschnitt werden die vorhandenen Komponenten und deren Beziehungen zueinander beschrieben

Um einen Service zu erstellen, erstellen wir eine Java-Klasse, die die Service-Basisklasse erweitert. Die Service-Basisklasse definiert verschiedene Callback-Methoden wie **`onBind()`** und **`onUnbind()`**. Um das Binden für unseren Service zu unterstützen, implementieren wir die onBind-Methode, die die aktuelle Instanz des zu bindenden Service zurückgibt:

```
public class BluetoothLeService extends Service {
    private final IBinder mBinder = new LocalBinder();
    @Override
    public IBinder onBind(Intent intent) {
    return mBinder;
    }
public class LocalBinder extends Binder {
    public BluetoothLeService getService() {
    return BluetoothLeService.this;
        }
    }
```

Die Verbindung zwischen dem Service und einer anderen Komponente (MainActivity2) wird als ServiceConnection dargestellt. Um **`BluetoothLeService`** mit MainActivity2 zu binden, müssen wir eine neue **`ServiceConnection`** implementieren und dabei die Methoden **`onServiceConnected`** und **`onServiceDisconnected`** überschreiben, um

eine Referenz auf die Service-Instanz zu erhalten, nachdem eine Verbindung hergestellt wurde.

Erstellen einer Service Connection für Service-Bindung

```
//Referenz auf den Service
private BluetoothLeService mBluetoothLeService;

@Override
public void onServiceConnected(ComponentName componentName, IBinder
service) {

//wird aufgerufen, wenn die Verbindung hergestellt wird.

 mBluetoothLeService = ((BluetoothLeService.LocalBinder) service).get-
Service();
if (!mBluetoothLeService.initialize()) {
finish();
}
//Verbindet sich automatisch mit dem Gerät nach erfolgreicher Start-Initialisierung.

mBluetoothLeService.connect(mDevices.get(currentDevice).getAddress());
}

@Override
public void onServiceDisconnected(ComponentName componentName) {
mBluetoothLeService = null;
}
};
```

Bindung Mainactivity2 an BluetoothLeService:

Um die Bindung durchzuführen, rufen wir bindService innerhalb userer MainActivity2 auf und geben eine explizite Intent ein, die den zu bindenden Service und eine Instanz einer ServiceConnection-Implementierung bestimmt: Wir geben Bindungsflags an, was bedeutet, dass das Ziel Service erstellt werden soll. Wenn die Bindung initiiert wurde, erstellen wir sie im **`onCreate`** unserer MainActivity2 und der entsprechende unbind-Service wird in **`onDestroy`** erstellt.

```
Intent gattServiceIntent = new Intent(this, BluetoothLeService.class);
bindService(gattServiceIntent, mServiceConnection, BIND_AUTO_CREATE);
```

Wenn der Service gebunden ist, sind alle seine Public-Methoden und Eigenschaften über das serviceBinder-Objekt verfügbar, das vom onServiceConnected-Handler abgerufen wird. (**`mBluetoothLeService`**).

5.5 Fragment-Mainactivity

Der im View Pager angezeigte Inhalt wird mit einem FragmentPagerAdapter erstellt. Fragment wird zur Darstellung jeder ViewPager verwendet. Um das zu ermöglichen, erweitern wir FragmentPagerAdapter im Mainactivity2 und überschreiben **`getCount()`**, um die Anzahl der Seiten zurückzugeben und **`getItem()`**, um das entsprechende Fragment für eine gegebene Position zurückzugeben.

```
public class PagerAdapter extends FragmentPagerAdapter {

public PagerAdapter(FragmentManager fm) {
       super(fm);
private Fragment2 mFragment = null;
       }
. . . . .
@Override
public int getCount() {
        return 1;
       }
. . . . .
@Override
public Fragment getItem(int position) {
      if (position == 0) {
          if (mFragment == null) mFragment = Fragment2.newInstance();
```

Fragment2 –Mainactivty2:

Wir erstellen Fragment2 Klasse, die Fragment erweitert und den onCreateView Handler überschreibt, um das fragment_2 Layout im XML-Datei anzuzeigen:

```
 public class Fragment2 extends Fragment{

 @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup con-
tainer, Bundle savedInstanceState) {
  return inflater.inflate(R.layout.fragment_2, container, false);
    }
}
```

Wir definieren ein Interface innerhalb Fragment2. Es sollte eine onSendData-Methode enthalten, die aufgerufen wird, wenn wir es implementieren.

```
public interface IFragmentInteraction {
void onSendData(BluetoothGattCharacteristic characteristic, byte[]
data);
 }
private IFragmentInteraction mListener;
 @Override
     public void onAttach(Activity activity) {
     super.onAttach(activity);
     try {
```

```
        mListener = (IFragmentInteraction) activity;
        }catch (ClassCastException e) {
        throw new ClassCastException(activity.toString()
        + " must implement IFragmentInteraction");
        }
    }
    @Override
    public void onDetach() {
        super.onDetach();
        mListener = null;
    }
```

Kehren wir zur MainActivity2 zurück und lassen wir die im Fragment2 definierte Schnittstelle implementieren und verwenden wir das View Model.

```
public class MainActivity2 extends Activity implements Frag-
ment2.IFragmentInteraction {
private BluetoothLeService mBluetoothLeService;
. . . . . .
    @Override
public void onSendData(BluetoothGattCharacteristic characteristic,
byte[] data)
mBluetoothLeService.setCharacteristicNotification(characteristic,
true);
mBluetoothLeService.writeCharacteristic(characteristic, data);
    }
```

Um Data mit Button zu senden benutzen wir unseren Fragment2:

```
btnVorwaerts.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
        b[0] = Frame.Fahrzeug.VORWAERTS.getValue();
        mListener.onSendData(characteristic, b);
```

```
btnA.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
        b[0] = Frame.Fahrstraßensteuerung.A.getValue();
        mListener.onSendData(characteristic, b);
    }
});
```

Um Data anzuzeigen benutzen wir unsere Mainactivity2:

-Balise auslesen:

```
...
If (uuid.equals(UUID_CHARACTERISTIC_READ)) {
   if (type == QueueCollection.TYPE_NOTIFICATION) {
        byte[] items = data;
```

```
 ((TextView) v.findViewById(R.id.textViewUART1)).setText(new
String(items))
. . . . . .
```

Im **textViewUART1** werden diese Daten als ASCII Code angezeigt.
Mit der Methode **getStr(byte[] items, int i)** im Frame Klasse definiert können wir items als Hex array anzeigen Mit der Klasse **CRC32** berechnen wir die Prüfsumme für das Feld.

```
. . .
CRC32 x = new CRC32();
x.update(data, 0, 9);
long checksum = x.getValue();
String s0 = getStr(data, 0);
. . . . . .

((TextView) v.findViewById(R.id.textViewUART2)).setText("Frame:" + s0
+ ":" + s1 + ":" + s2 + ":" + s3 + ":" + s4 + ":" + s5 + ":" + s6 +
":" + s7 + ":" + s8 + "\n" + "<checksum:" + checksum + ">");
```

Sobald die Smartphone-Anwendung Notification Daten von der Chracheristic Read empfängt, kann der User sie mit einem Zeitstempel speichern. Diese Daten werden mit shared preference in der App bleiben und dann gespeichert auch wenn die Anwendung beendet wird.

-Fahrzeugs- Responses (ASCII und hex Format)

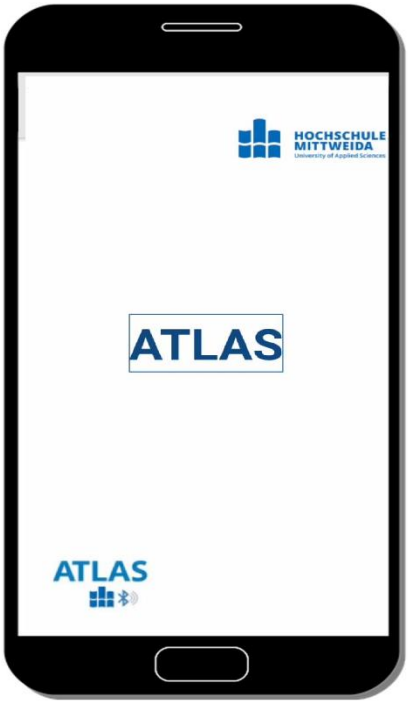
```
if (Fragment2.class.isInstance(fragment)) {
   if (uuid.equals(UUID_CHARACTERISTIC_WRITE)) {
     if (type == QueueCollection.TYPE_READ) {
           ((TextView) v.findViewById(R.id.textView)).setText((""));
           StringBuffer stringBuffer = new StringBuffer(data.length);
        for (byte byteChar : data)
        stringBuffer.append(String.format("%02X ", byteChar));
((TextView) v.findViewById(R.id.textView)).setText(new String(data) +
"\n" + "<" + stringBuffer.toString() + ">");
```

(Programmablaufplan von GATT Transaktionen befindet sich im Anhang A10)

6 Test

Dieses Kapitel stellt die ATLAS-CAR App aus Sicht des Benutzers vor. Dabei werden die einzelnen Funktionen und UI gezeigt, die beim Ausführen der App bereits verfügbar sind. Es wurden auch ein paare Erweiterungsmöglichkeiten vorgeschlagen und getestet.

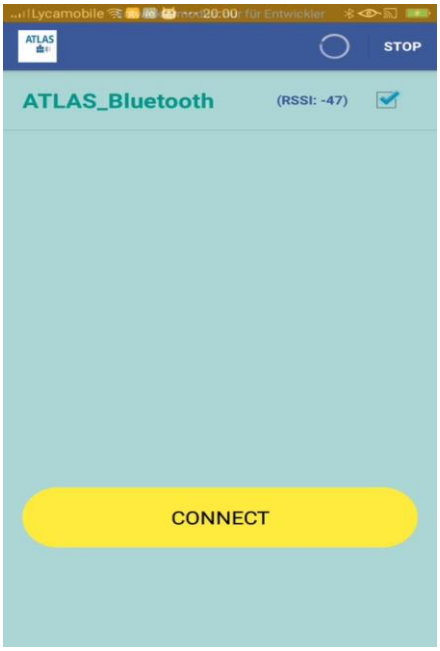
Ein Terminal-Emulator-Programm kann verwendet werden, um zu überprüfen, ob alles so funktioniert, wie es sollte. Es ist notwendig, dem Terminal-Emulator mitzuteilen, welcher COM-Anschluss dem nRF zugewiesen ist, und seine Kommunikationsgeschwindigkeit auf 9600 Baud einzustellen.



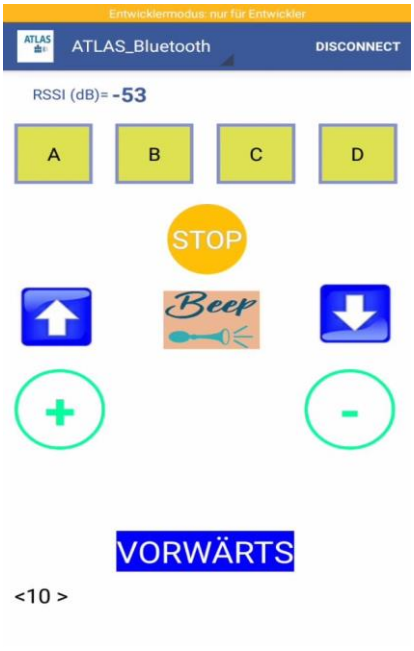
WelcomeScreen

Modus Auswahl

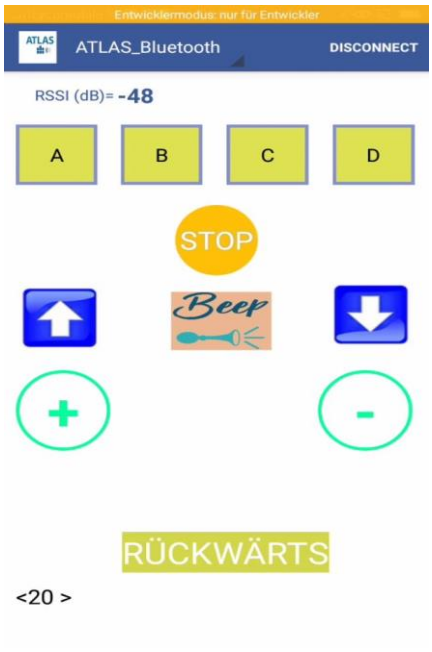
6.1 Kinder-Modus



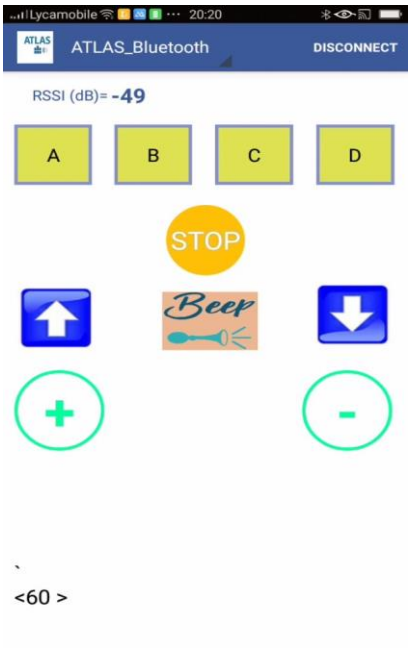
automatisch nach BLE Geräte scannen. Wir wählen mit dem Checkbox das ATLAS Modul und stellen mit Connect-Button eine Verbindung her.



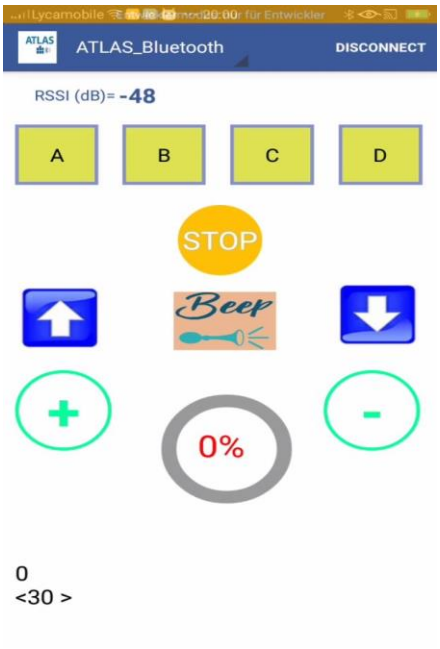
vorwärts Fahren, der Benutzer wird durch einen Toast benachrichtigt



rückwärts Fahren

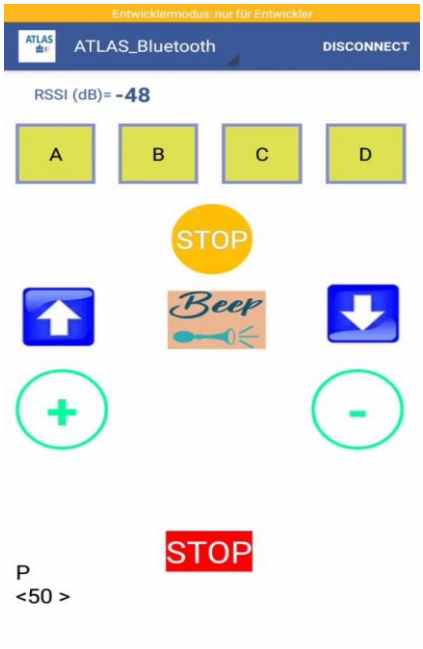


akustisches Signal (Beep)

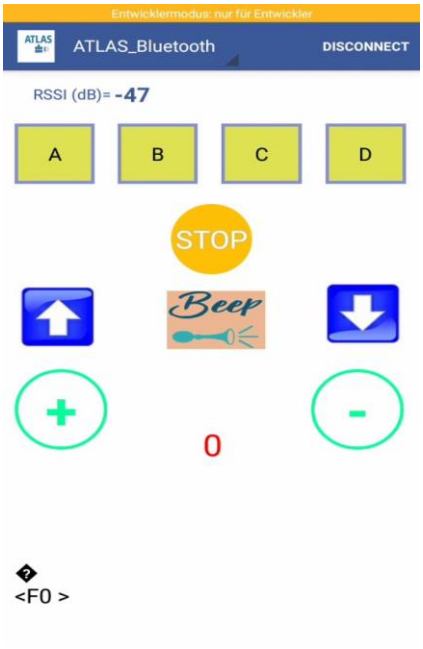


minimale Geschwindigkeit beim Dekrementieren

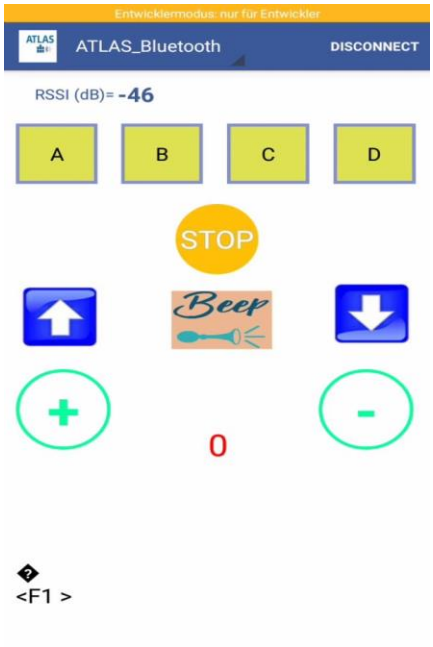
maximale Geschwindigkeit beim Inkrementieren



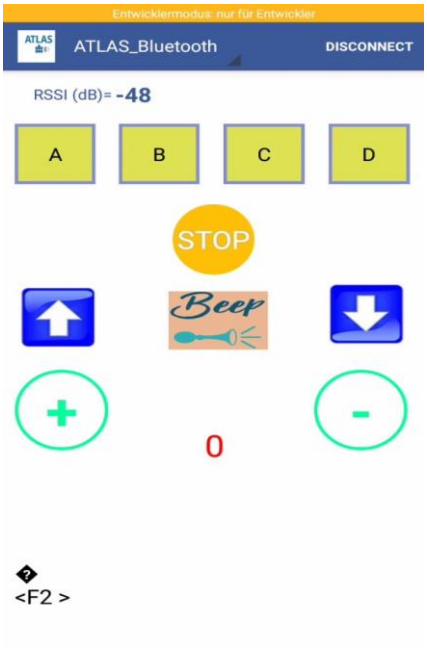
Fahrzeug stoppen, der Benutzer wird durch einen Toast benachrichtigt



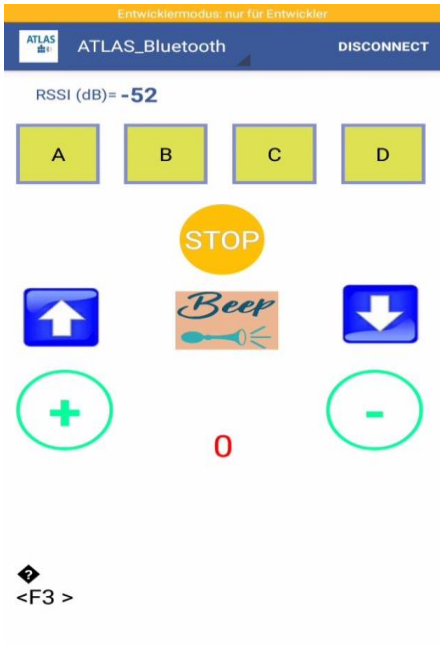
Fahrstraßensteuerung A



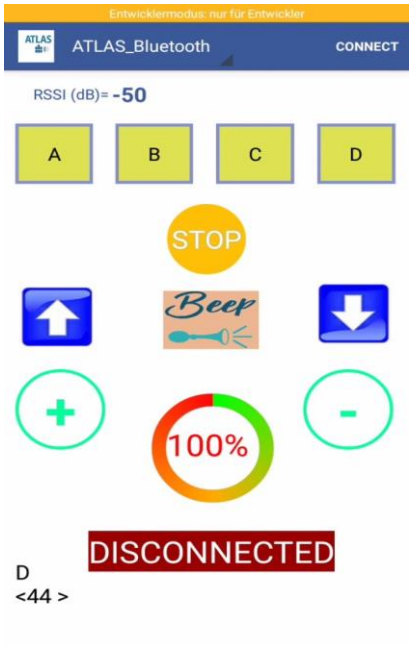
Fahrstraßensteuerung B



Fahrstraßensteuerung C

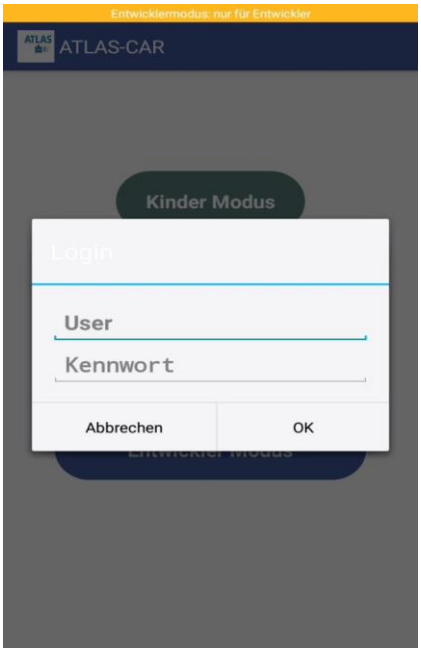


Fahrstraßensteuerung D

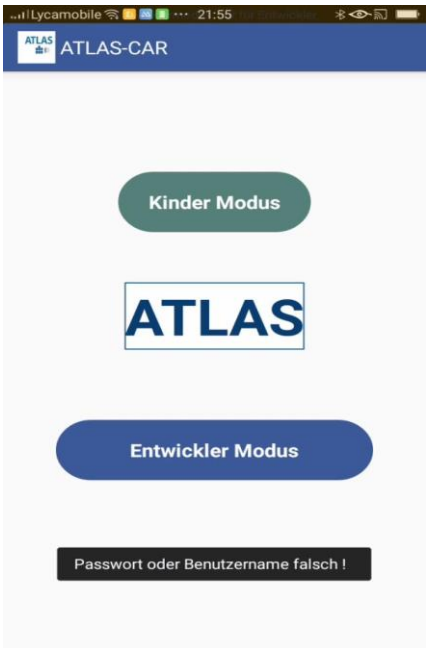


Verbindung trennen

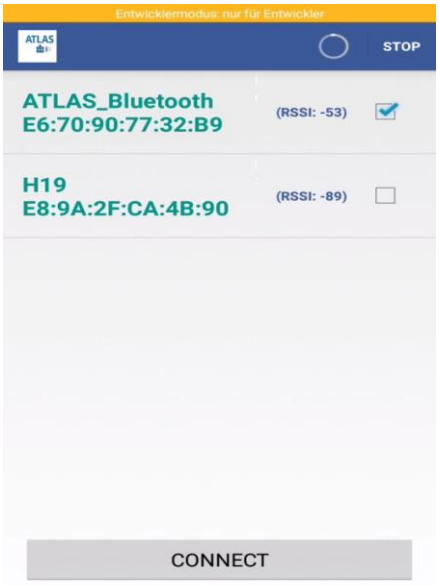
6.2 Entwickler-Modus



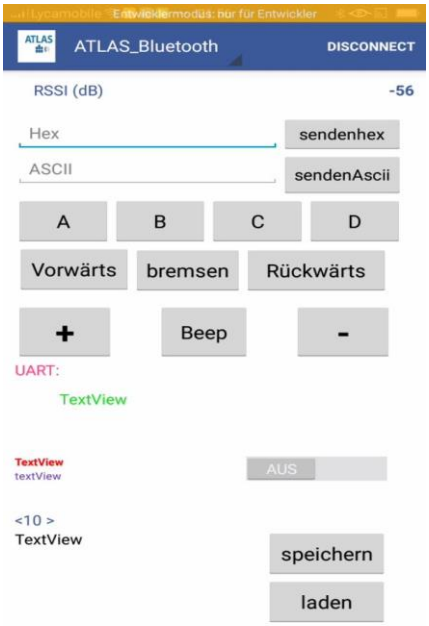
Wird Entwickler_Modus ausgewählt, so befindet man sich auf dem Login-Screen



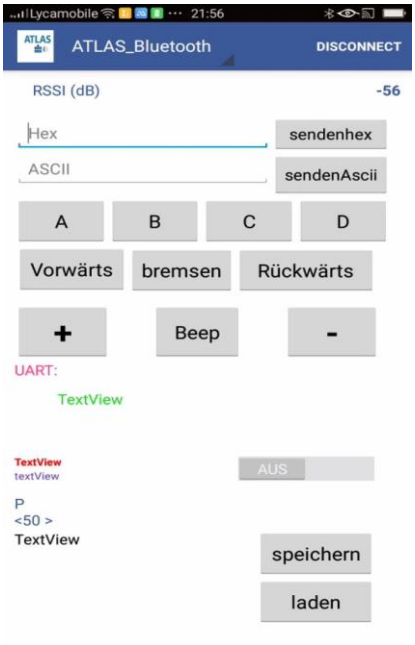
falsche Login Daten eingegeben



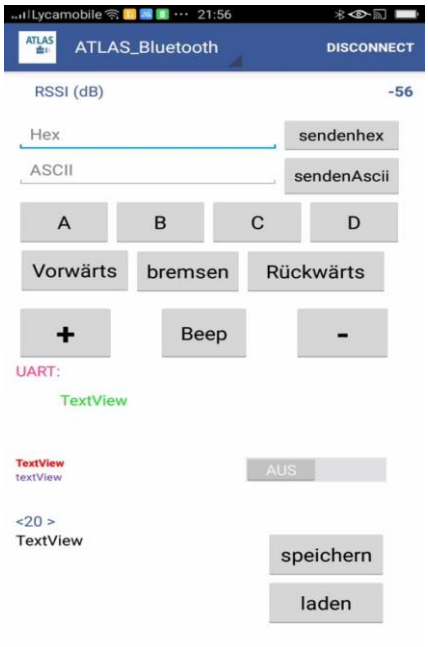
automatisch nach BLE Geräte scannen und ATLAS Modul auswählen (DeviceActivity2)



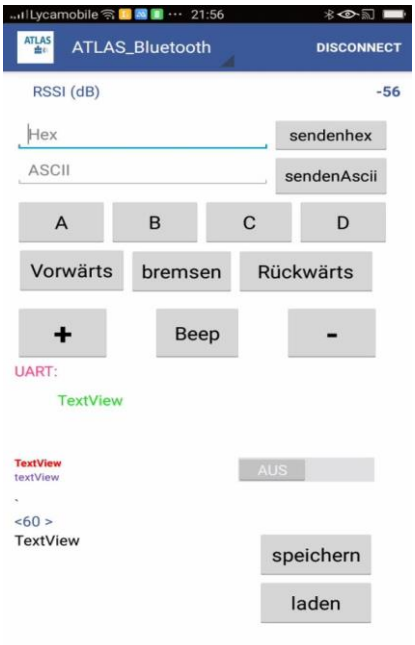
Vorwärts Fahren



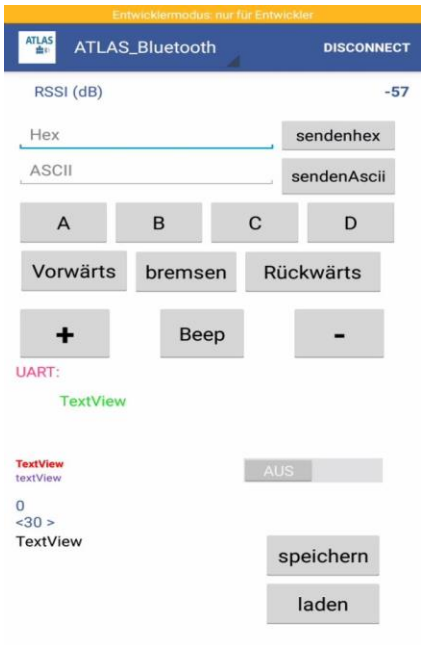
Fahrzeug stoppen



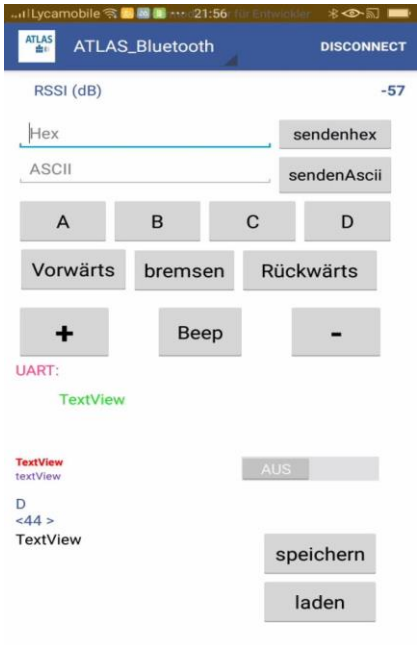
Rückwärts Fahren



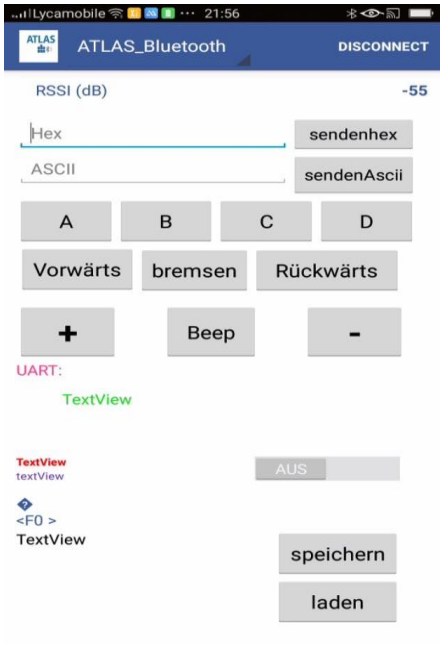
Akustisches Signal (Beep)



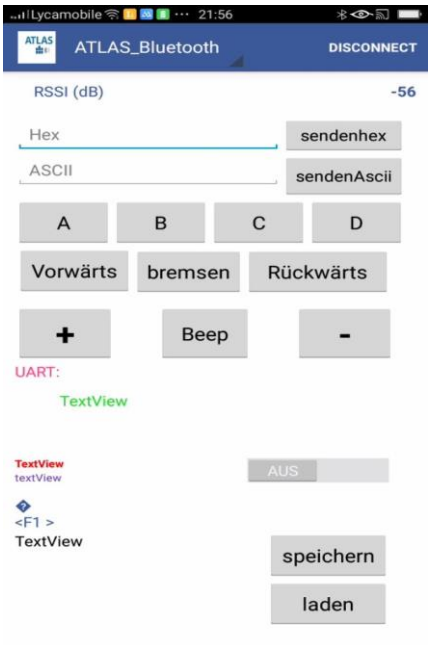
Minimale Geschwindigkeit



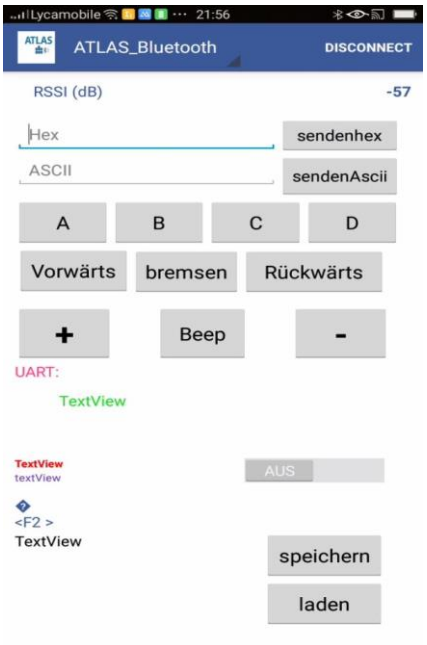
Maximale Geschwindigkeit



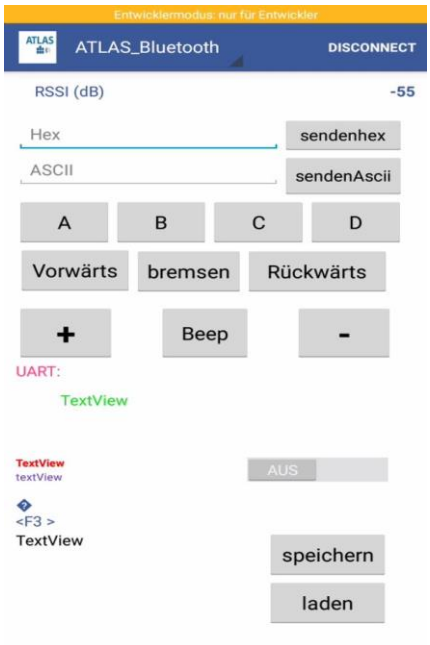
Fahrstraßensteuerung A



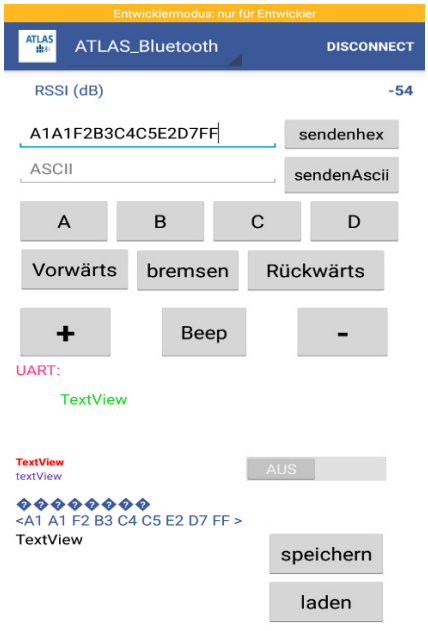
Fahrstraßensteuerung B



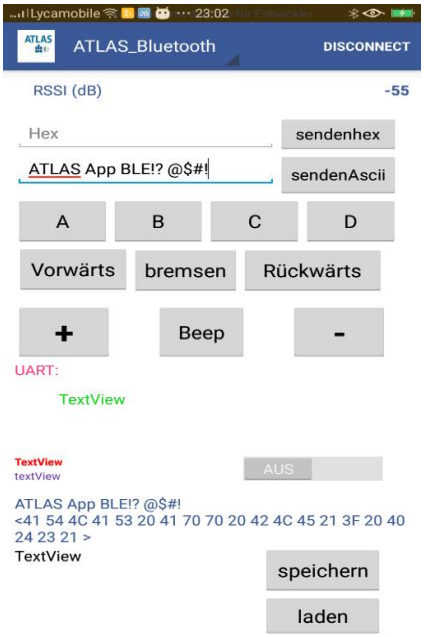
Fahrstraßensteuerung C



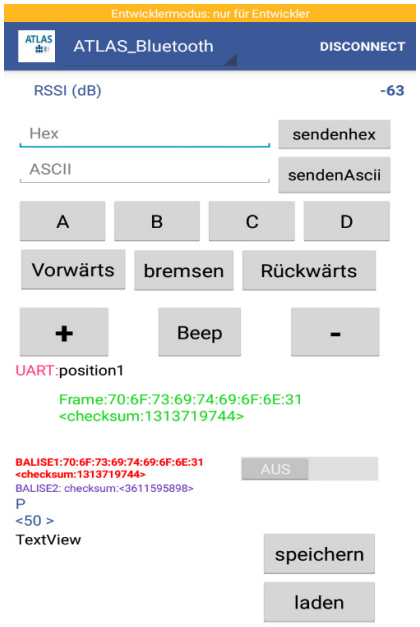
Fahrstraßensteuerung D



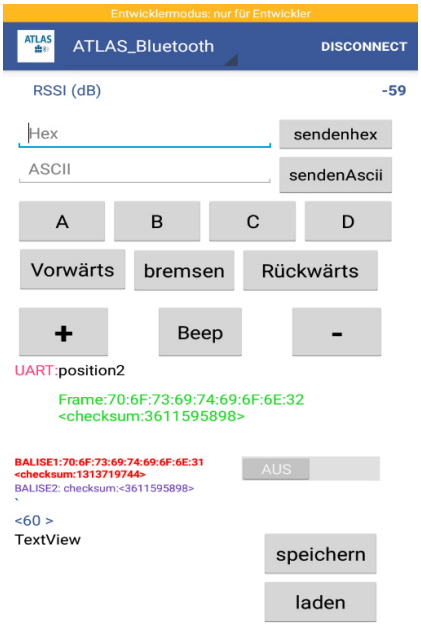
Senden beliebige Hex
Bsp.: A1A1F2…
Response erfolgreich vom GATT Server der blaue Text(<Hex>)



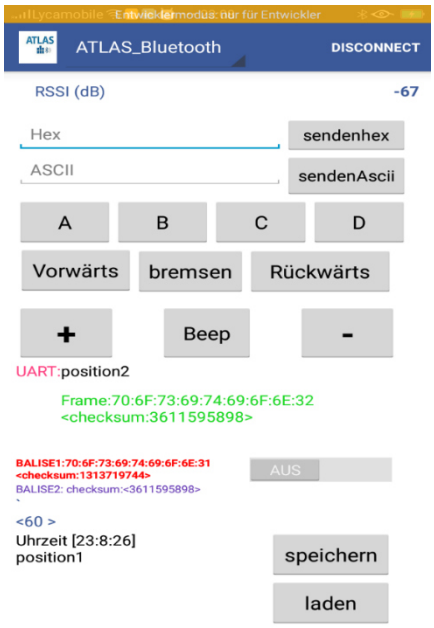
Senden beliebige ASCII
Bsp.: ATLAS App BLE…
Response erfolgreich vom GATT Server der blaue Text (ASCII <Hex>)



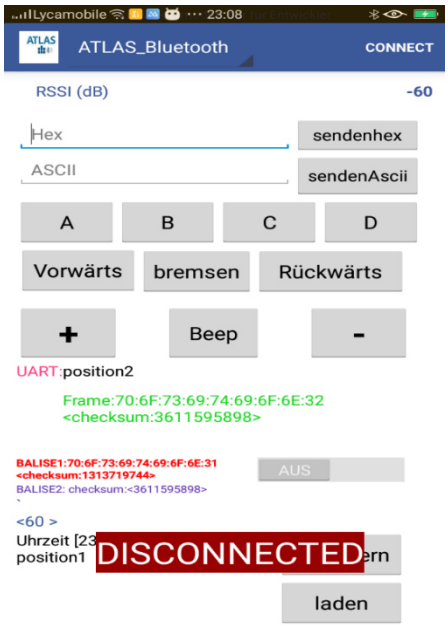
-Über UART erhält der User “position1” (Der schwarze Text) und ihre Hex Werte plus die Prüfsumme (Der grüne Text)



-Über UART erhält der User “position2” (Der schwarze Text) und ihre Hex Werte plus die Prüfsumme (Der grüne Text)

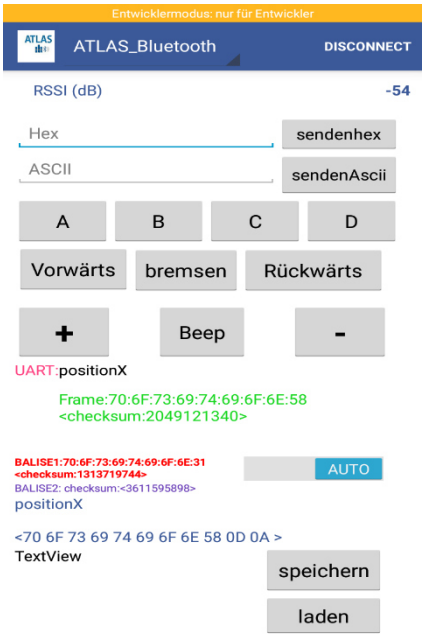


-Mit dem Speicher Button gespeicherte “position1” über UART wird mit Zeitstempel wieder geladen (Der letzte schwarze Text)



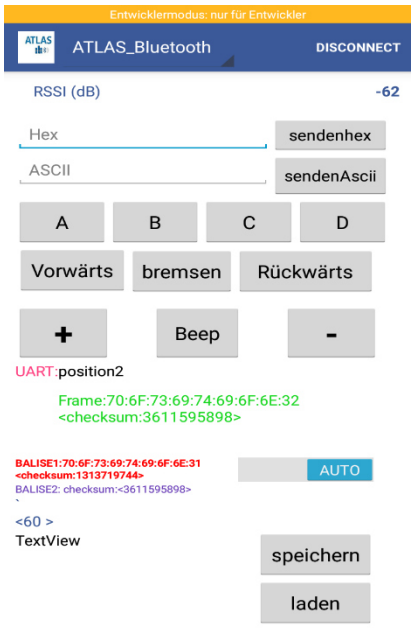
Beim Verbindungsverlust oder beim Anklicken disconnect Button erscheint sofort ein Toast (Disconnected)

6.3 Erweiterungsmöglichkeit



Die App kann über UART erhaltene Daten automatisch wieder an den ATLAS senden: Wenn der User Switch Button aktiviert hat. Bsp.: über UART wird positionX erhalten, automatisch wird diese position wieder an den ATLAS Modul gesendet.

Die App kann über UART erhaltene Daten die errechnete Prüfsumme mit der Prüfsumme von Balise1 vergleichen und wenn die Gleichung stimmt, wird Fahrstraßensteuerung A Befehl automatisch im App ausgelöst.



Die App kann über UART erhaltene Daten, die errechnete Prüfsumme mit der Prüfsumme von Balise2 vergleichen und wenn die Gleichung stimmt, wird ein Beep Befehl automatisch im App ausgelöst.

6.4 Der neue Testaufbau

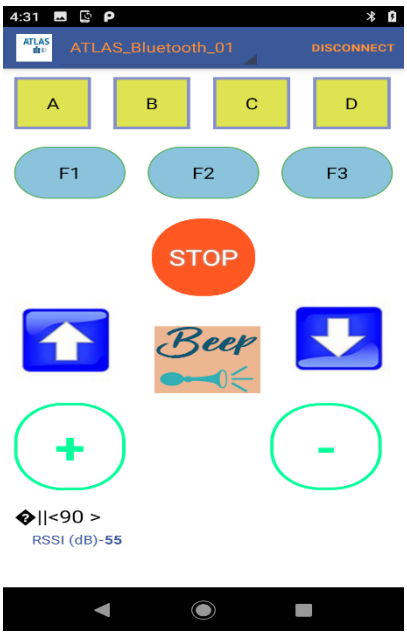
Um die Kommunikation zwischen der entwickelten App und den M16c62 über nRf 52832 via BLE zu prüfen, wird ein Testaufbau mit diesem MCU und dem Modul als Prototyp angefertigt und getestet. Hardwareseite werden wir auf DC Motor und Balise verzichten. Dieser Abschnitt veranschaulicht den Versuchaufbau, der für die Auswertung verwendet wurde. Die erhaltenen Ergebnisse werden mit den Anforderungen gemäß Kapitel 3 im Kapitel 7 verglichen. Im Rahmen dieser Masterarbeit wurde ein android Smartphone "motoe[6]"(Android Version 9) für die Programmierung der Applikation und der Test des Prototyps verwendet. Jede einzelne Funktion der App wurde vom Autor dieser Masterarbeit getestet und überprüft.

Abbildung 31: Testaufbau MCU M16c62 und nRF 52832

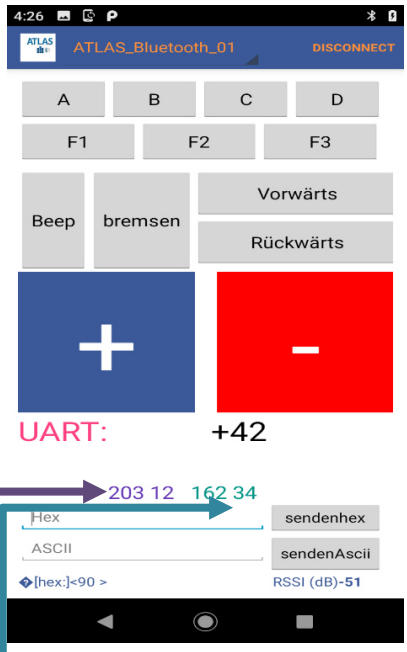
Das BLE Modul wird mit 3.3 Volt externen Spannung versorgt. Für die Pinbelegung MCU mit dem nRf Modul (siehe [26]).

1) Nach dem Einschalten des Boards beleuchten alle LED´s und die Geschwindigkeit ist immer auf null gesetzt.
2) Nach erfolgreicher Verbindung mit dem BLE Modul:
   - Im kinder-Modus: funktionieren alle Buttons: alle LED´s können Ein/Aus geschaltet werden. Im LCD-Display werden die Ergebnisse je nach dem Button Klick angezeigt. Die Geschwindigkeit kann in einem Intervall Vorwärts [0,+100] und Rückwärts [-100, 0] Inkrementiert und Dekrementiert werden (nach dem Drücken und Halten Prinzip mit dem Loslassen bleibt sie unverändert).

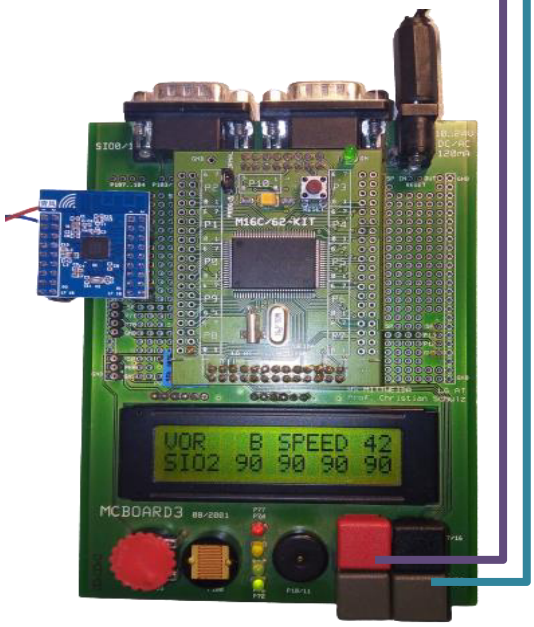
- Im Entwickler-Modus: wird auf der App zusätzlich die Geschwindigkeit-Callbacks als Notification gelesen und angezeigt. Beim Drücken eines Buttons des Boards wird auch dieses Event je nachdem in zwei verschieden Textviews angezeigt. Diese Funktion ersetzt also die Balisen.



Kinder-Modus mit einer neuen Benutzeroberfläche



Entwickler-Modus
Die Geschwindigkeit +42 als Notification gelesen. Zwei unterschiedliche Notification Werte nach dem Button Druck am Board.



Alle LED´s werden durch Buttons ausgeschaltet.
LCD: zeigt die Geschwindigkeit 42 an.
Button B Event Klick
Button Vorwärts Event Klick

7 Zusammenfassung und Ausblick

In diesem Projekt wird ein BLE-Modul vom Typ nRF52832 mit dem MCU M16C62 verbunden, so dass er über eine BLE-Anwendung kommunizieren kann.
Innerhalb dieser Masterarbeit haben wir die wichtigsten Konzepte und Ereignisse im Zusammenhang mit der Implementierung einer Central-Peripheral Kommunikation kennen gelernt. Dazu gehört:
- Scanning
- Verbindungsherstellen
- Service Discovery
- Senden und Empfangen von Daten

Außerdem haben wir im Zuge dieser Arbeit das Kommunikationsprotokoll und die verwendeten App-Komponente betrachtet.
Alle Transaktionen werden durch den GATT-Client gestartet und die ATT Methoden, die im Kapitel 4 benannt wurden, verwendet worden. Während GATT-Server verfügt über (Services bzw. Characteristics) und kann Anforderungen vom GATT Client annehmen. Das ATT Protokoll definiert also diese Client-Server Architektur.
Um die serielle Kommunikation über BLE zu erzielen, benötigen wir also diesen Nordic UART Service, der ein Rx characteristic mit Notify Permission und ein Tx Characteric mit Write Permission enthält. Die wichtigste Voraussetzung bei dem Entwurf dieser BLE App ist einen Android Service (gebundenen Service) zu erstellen und BLE-API zu implementieren. Es soll auch betrachtet, dass Alle ATT Operationen nacheinander durchgeführt werden. Aufgrund der Tatsache, dass BLE-API in Android asynchron ist, müssen wir als Lösung die QueueCollection klasse implementieren. Infolge dessen war die Realisierung der App erfolgreich. Die Verbindung kann schnell aufgebaut werden und alle ATT Methoden oder GATT Transaktion können problemlos durchgeführt werden. Somit erfüllt unsere BLE App alle Anforderungen.
Es wurden auch eine paare Erweiterung vorgeschlagen sowie zukünftige Funktionalitäten erfolgreich im Kapitel 6 getestet.
Abschließend zeigte sich bei der Entwicklung der **„ATLAS-CAR "**, dass es möglich war mit Android eine BLE Daten Kommunikation über UART im Vollduplex Modus herzustellen. Somit können Motorsteuerungsbefehle und Kommandos an das Fahrzeug gesendet werden sowie an die Fahrstraßen und die aktuellen Status von Balisen und Geschwindigkeit-Callbacks ausgelesen werden und anzeigen lassen. Mit den Erweiterungen von Kapitel 6 wird die Möglichkeit zur Entwicklung eines automatischen ATLAS-Verkehrsystems angeboten.

Nachdem die Applikation ATLAS-CAR fertiggestellt war, wurden alle festgelegten Funktionen und Anwendungsfälle aus dem Kapitel 3 auf Android Version 5, Android Version 8.1.0 und Version 9.0.0 Smartphones getestet. Mit Hilfe einer Terminal Software (Hterm)

wurde das nRF Modul an den Computer angeschlossen und geprüft. Wir können feststellen, dass Senden und Empfangen von Daten reibungslos funktioniert und alles läuft einwandfrei. Was bedeutet, dass alle BLE UART Kommandos erfolgreich durchgeführt wurden und die BLE Verbindung stabil geblieben. Somit ist unsere App einsatzbereit. Aufgrund Berufsbedingte Kontakte und Schutzmaßnahmen gegen Corona-Virus (SARS-CoV-2) bleibt Laborraum der Hochschule Mittweida geschlossen. Daher wurde die App noch nicht für den ATLAS-Fahrzeug Prototyp getestet. (update siehe 6.4)

Als Verbesserungsvorschlag und Erweiterung könnte man Batterie Service mit dem Batteriestand Characteristic beim GATT Server implementieren so kann die App Batteriestatus des Moduls abfragen und lesen. Die App kann auch gleichzeitig zwei Fahrzeugs (im Entwickler-Modus) ansteuern.
Die Anwendung existiert nun für Android es wäre auch machbar für IOS. Somit können fast alle Smartphone Betriebssysteme abgedeckt werden.

Literaturverzeichnis

[1] android1. (27. 12 2019). Abgerufen am 13. 04 2020 von https://developer.android.com/guide/components/intents-filters

[2] android2. (27. 12 2019). Abgerufen am 10. 05 2020 von https://developer.android.com/guide/components/fragments

[3] android3. (27. 12 2019). Abgerufen am 03. 04 2020 von https://developer.android.com/guide/components/broadcasts

[4] android4. (27. 12 2019). Abgerufen am 01. 04 2020 von https://developer.android.com/reference/android/bluetooth/package-summary

[5] android5. (2020, 01 06). Retrieved 03 10, 2020, from https://source.android.com/devices/bluetooth

[6] android6. (2020, 05 07). Retrieved 05 12, 2020, from https://developer.android.com/guide/platform

[7] android7. (05. 05 2020). Abgerufen am 11. 05 2020 von https://developer.android.com/reference/android/app/Activity

[8] android8. (19. 03 2020). Abgerufen am 10. 05 2020 von https://developer.android.com/guide/components/services

[9] android9. (2019, 12 27). Retrieved from https://developer.android.com/guide/components/activities/tasks-and-back-stack

[10] android10. (27. 12 2019). Von https://developer.android.com/things/sdk/pio/uart abgerufen

[11] Berns, K. (1. Auflage 2010). Eingebettete Systeme Systemgrundlagen und Entwicklung. Wiesbaden: Vieweg+Teubner Verlag , ISBN 978-3-8348-0422-8.

[12] bin Aftab, M. (2017). Building Bluetooth Low Energy Systems. Birmingham: Packt Publishing Ltd, ISBN 978-1-78646-108-7.

[13] Gerber, A. (2015). Learn Android Studio ,Build Android Apps Quickly and Effectively. berkeley ca: Apress, ISBN-13 (pbk): 978-1-4302-6601-3.

[14] Gupta, N. (2016). In Inside Bluetooth Low Energy Second Edition (S. 10). Boston: Artech House Publishers, ISBN-13: 978-1630810894.

[15] Heydon, R. (2013). Bluetooth Low Energy The Developer's Handbook. Pearson Education, Inc -ISBN-13: 978-0-13-288836-3.

[16] infineon. (03. 07 2007). Abgerufen am 06. 02 2020 von http://dtsheet.com/doc/194905/infineon-tle5205-2g

[17] Kevin Townsend, C. C. (2014). In Getting Started with Bluetooth Low Energy (S. 10). O'Reilly and Associates, ISBN: 978-1-491-94951-1.

[18] NORDIC1, S. (20. 10 2017). Abgerufen am 15. 12 2019 von https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk5.v14.0.0%2Fble_sdk_app_nus_eval.html

[19] NORDIC2, S. (13. 03 2017). SoftDevice SpecificationS132 v4.0. Abgerufen am 17. 12 2019 von https://infocenter.nordicsemi.com/pdf/S132_SDS_v4.0.pdf

[20] NORDIC3, S. (04. 05 2020). Von https://infocenter.nordicsemi.com/index.jsp?topic=%2Fstruct_nrf52%2Fstruct%2Fnrf52832.html abgerufen

[21] Pack, S. F. (2008). Atmel AVR Microcontroller Primer: Programming and Interfacing. San Rafael, CA: Morgan & Claypool , ISBN: 1598295411.

[22] Renesas, E. (2010, April 1). Retrieved 01 06, 2020, from https://www.renesas.com/in/en/doc/products/mpumcu/apn/003/reu05b0025_m16cap.pdf

[23] SIG. (2014, December 02). CS – Core Specification version 4.2. Retrieved 11 05, 2019, from https://www.bluetooth.com/de/specifications/archived-specifications/

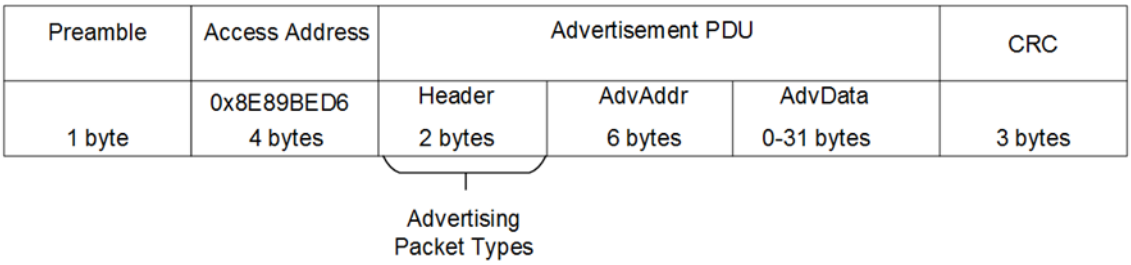
[24] Renesas, E. (2010, April 1). Retrieved 01 06, 2020, from https://www.google.com/url?client=internal-element-cse&cx=partner-pub-9634067433254658:9653363797&q=https://www.renesas.com/us/en/doc/products/mpumcu/001/rej03b0001_16c62pds.pdf&sa=U&ved=2ahUKEwj9g6bC94frAhWMC-wKHcsYBscQFjAAegQIABAB&usg=AOvVaw0CNzEhI1rEvyMfIptb7ZRg

[25] Yujing,Wang, Prototyp für ATLAS-Weichensteuerung über Infrarot-Schnittstelle, Bachelorarbeit, Hochschule Mittweida, 2019

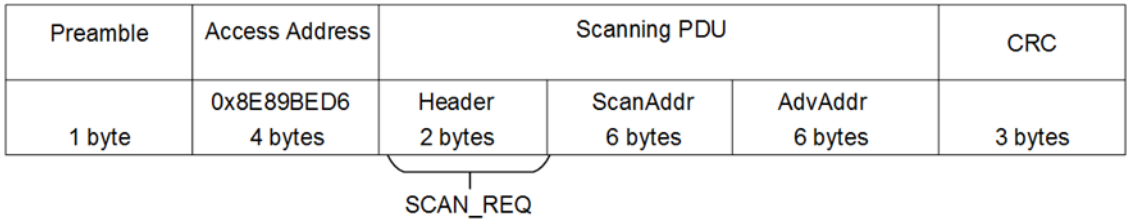
[26] Liu, Yan, ATLAS-Fahrzeugansteuerung über Bluetooth, Bachelorarbeit, Hochschule Mittweida, 2020

Anhang

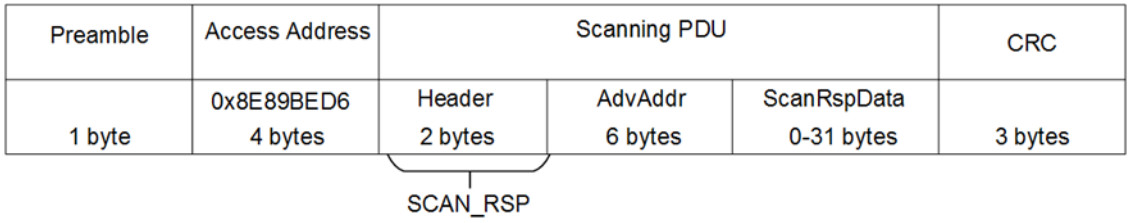
A1: advertising packet



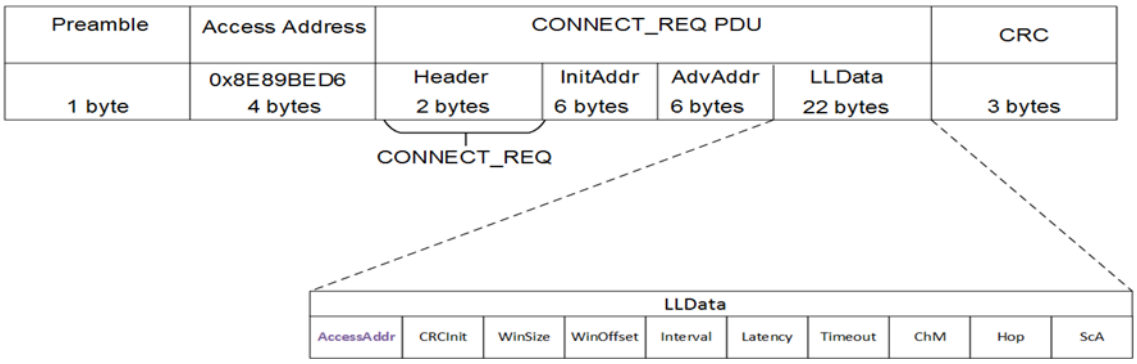
A2: scan request packet



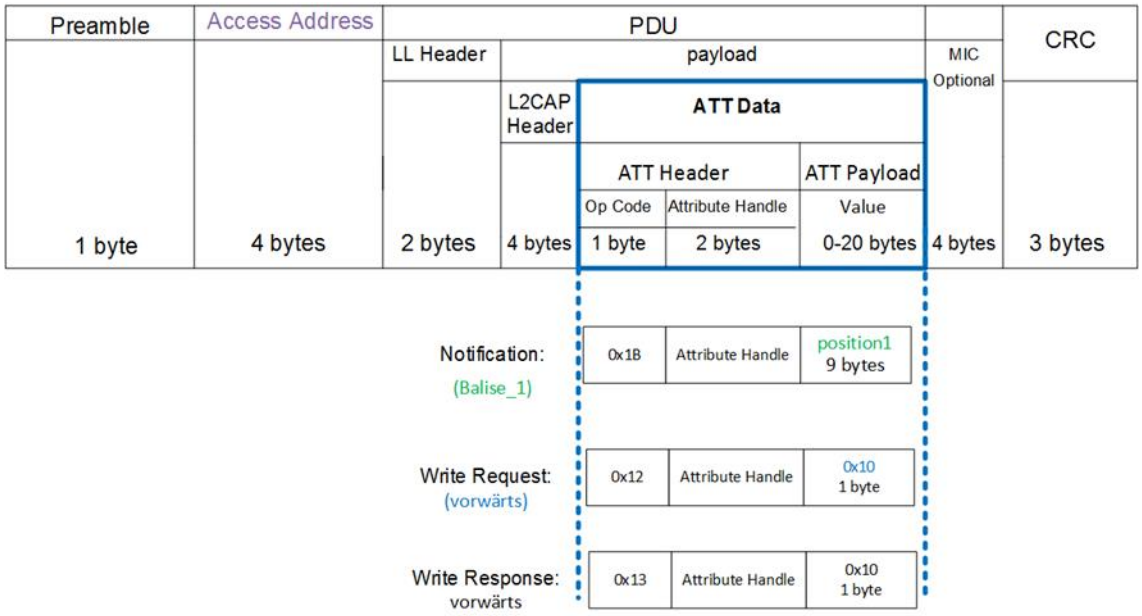
A3: scan response packet



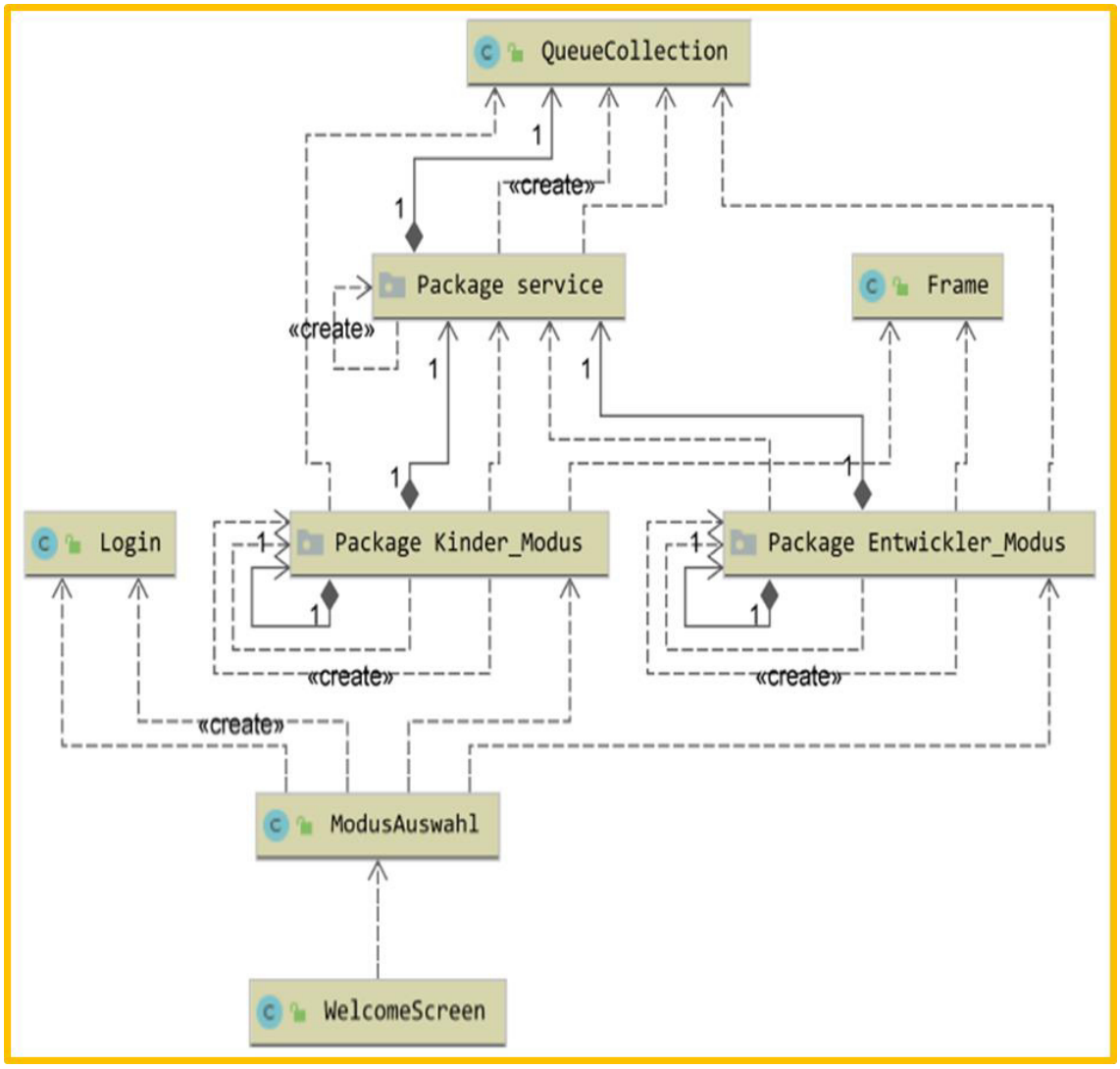
A4: Initiating PDU (CONNECT_REQ packet)



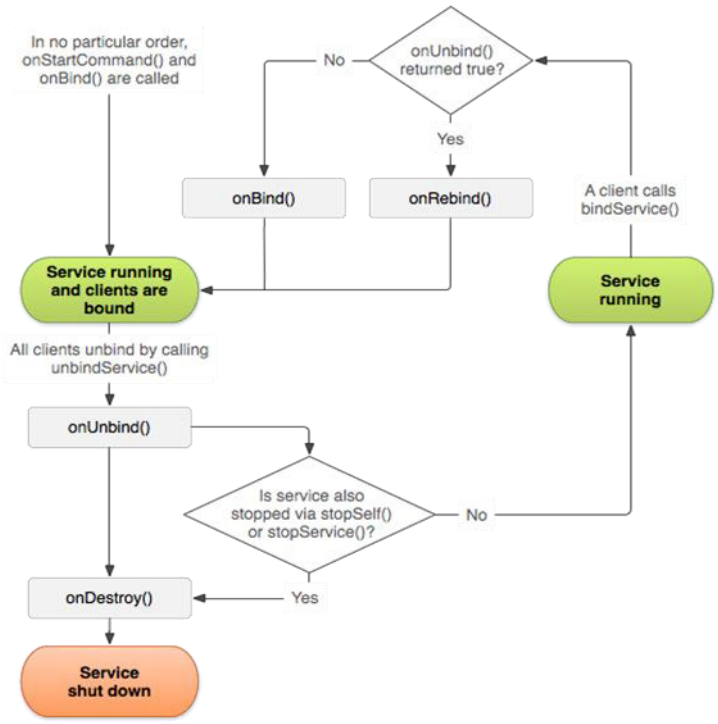
A5: Data Packet



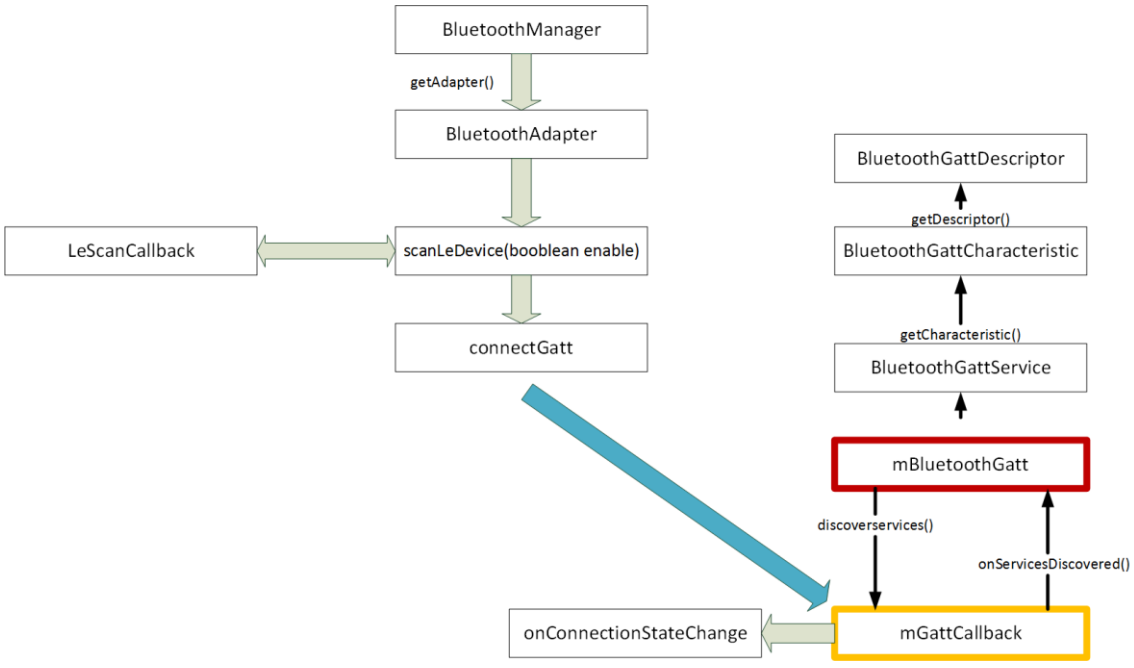
A6: Paketdiagramm



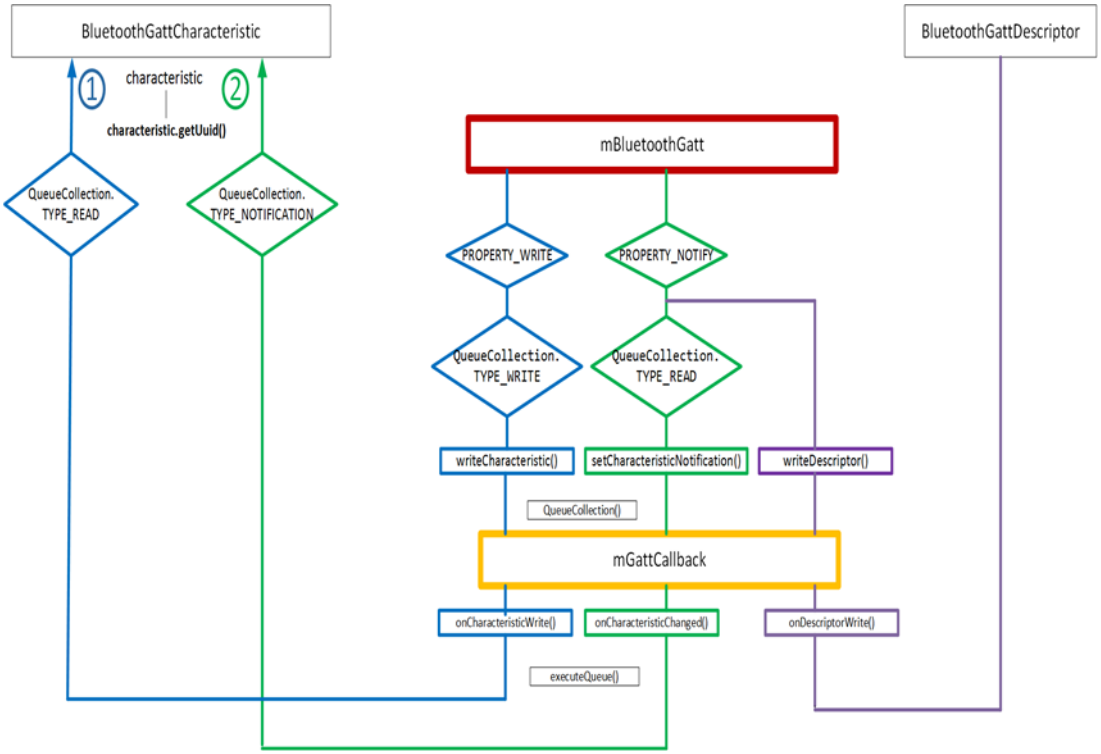
A8: bound service Lebenszyklus



A9: GATT Discovery



A10: GATT Transaktionen

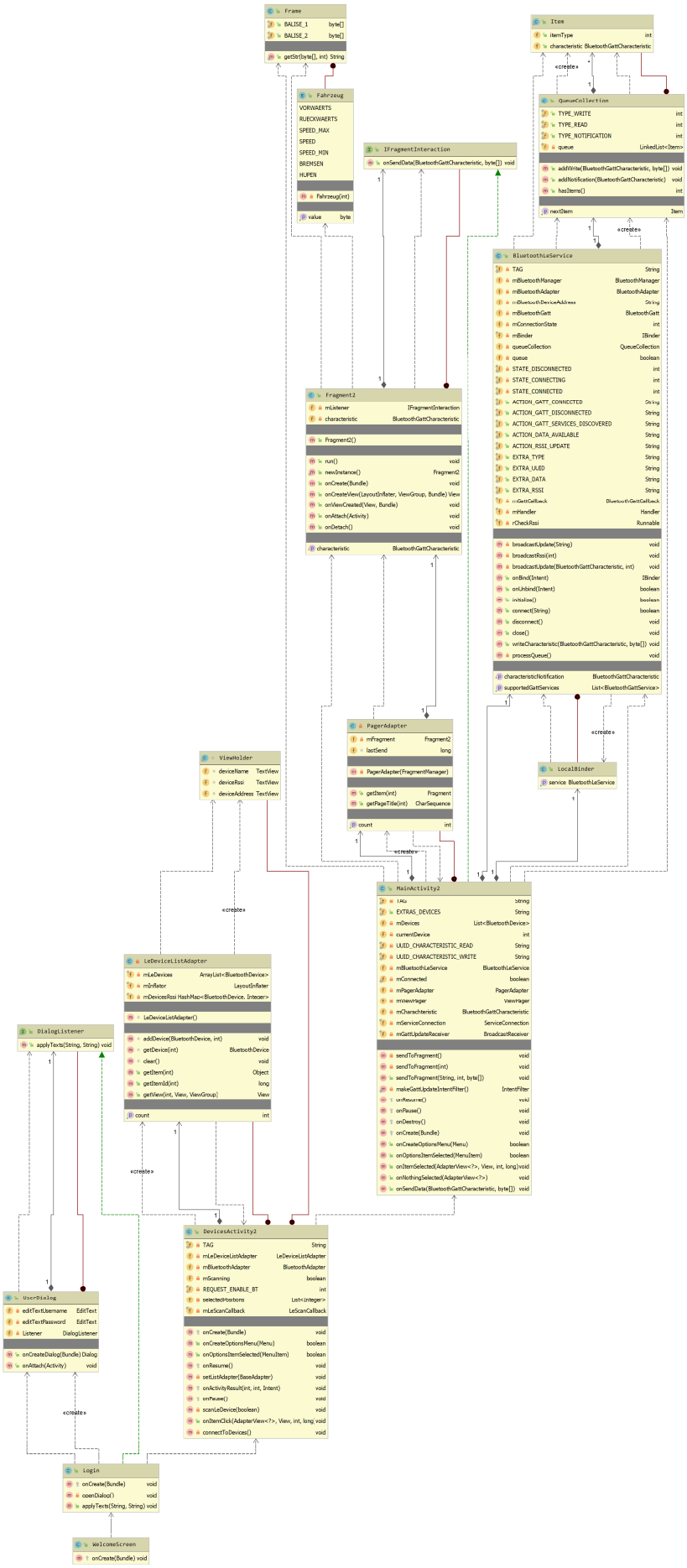


A11: Überblick UART-Kommandos

Befehle	Hex-Wert
VORWAERTS	0x10
RUECKWAERTS	0x20
SPEED (-) (Dekrement)	0x30
SPEED (+) (Inkrement)	0x40
BREMSEN	0x50
Beep	0x90
Fahrstraßensteuerung-A	0xF0
Fahrstraßensteuerung-B	0xF1
Fahrstraßensteuerung-C	0xF2
Fahrstraßensteuerung-D	0xF3

F1	0x60
F2	0x70
F3	0x80

A7: Klassendiagramm (Entwickler_Modus)


Frame
BALISE_1 byte[]
BALISE_2 byte[]
getStr(byte[], int) String
Item
itemType int
characteristic BluetoothGattCharacteristic
«create»
Fahrzeug
VORWAERTS
RUECKWAERTS
SPEED_MAX
SPEED
SPEED_MIN
BREMSEN
HUPEN
Fahrzeug(int)
value byte
QueueCollection
TYPE_WRITE int
TYPE_READ int
TYPE_NOTIFICATION int
queue LinkedList<Item>
addWrite(BluetoothGattCharacteristic, byte[]) void
addNotification(BluetoothGattCharacteristic) void
hasItems() int
nextItem Item
IFragmentInteraction
onSendData(BluetoothGattCharacteristic, byte[]) void
BluetoothLeService
TAG String
mBluetoothManager BluetoothManager
mBluetoothAdapter BluetoothAdapter
mBluetoothDeviceAddress String
mBluetoothGatt BluetoothGatt
mConnectionState int
mBinder IBinder
queueCollection QueueCollection
queue boolean
STATE_DISCONNECTED int
STATE_CONNECTING int
STATE_CONNECTED int
ACTION_GATT_CONNECTED String
ACTION_GATT_DISCONNECTED String
ACTION_GATT_SERVICES_DISCOVERED String
ACTION_DATA_AVAILABLE String
ACTION_RSSI_UPDATE String
EXTRA_TYPE String
EXTRA_UUID String
EXTRA_DATA String
EXTRA_RSSI String
mGattCallback BluetoothGattCallback
mHandler Handler
rCheckRssi Runnable
broadcastUpdate(String) void
broadcastRssi(int) void
broadcastUpdate(BluetoothGattCharacteristic, int) void
onBind(Intent) IBinder
onUnbind(Intent) boolean
initialize() boolean
connect(String) boolean
disconnect() void
close() void
writeCharacteristic(BluetoothGattCharacteristic, byte[]) void
processQueue() void
characteristicNotification BluetoothGattCharacteristic
supportedGattServices List<BluetoothGattService>
Fragment2
mListener IFragmentInteraction
characteristic BluetoothGattCharacteristic
Fragment2()
run() void
newInstance() Fragment2
onCreate(Bundle) void
onCreateView(LayoutInflater, ViewGroup, Bundle) View
onViewCreated(View, Bundle) void
onAttach(Activity) void
onDetach() void
characteristic BluetoothGattCharacteristic
PagerAdapter
mFragment Fragment2
lastSend long
PagerAdapter(FragmentManager)
getItem(int) Fragment
getPageTitle(int) CharSequence
count int
«create»
LocalBinder
service BluetoothLeService
ViewHolder
deviceName TextView
deviceRssi TextView
deviceAddress TextView
MainActivity2
TAG String
EXTRAS_DEVICES String
mDevices List<BluetoothDevice>
currentDevice int
UUID_CHARACTERISTIC_READ String
UUID_CHARACTERISTIC_WRITE String
mBluetoothLeService BluetoothLeService
mConnected boolean
mPagerAdapter PagerAdapter
mViewPager ViewPager
mCharacteristic BluetoothGattCharacteristic
mServiceConnection ServiceConnection
mGattUpdateReceiver BroadcastReceiver
sendToFragment() void
sendToFragment(int) void
sendToFragment(String, int, byte[]) void
makeGattUpdateIntentFilter() IntentFilter
onResume() void
onPause() void
onDestroy() void
onCreate(Bundle) void
onCreateOptionsMenu(Menu) boolean
onOptionsItemSelected(MenuItem) boolean
onItemSelected(AdapterView<?>, View, int, long) void
onNothingSelected(AdapterView<?>) void
onSendData(BluetoothGattCharacteristic, byte[]) void
LeDeviceListAdapter
mLeDevices ArrayList<BluetoothDevice>
mInflator LayoutInflater
mDevicesRssi HashMap<BluetoothDevice, Integer>
LeDeviceListAdapter()
addDevice(BluetoothDevice, int) void
getDevice(int) BluetoothDevice
clear() void
getItem(int) Object
getItemId(int) long
getView(int, View, ViewGroup) View
count int
DialogListener
applyTexts(String, String) void
DevicesActivity2
TAG String
mLeDeviceListAdapter LeDeviceListAdapter
mBluetoothAdapter BluetoothAdapter
mScanning boolean
REQUEST_ENABLE_BT int
selectedPositions List<Integer>
mLeScanCallback LeScanCallback
onCreate(Bundle) void
onCreateOptionsMenu(Menu) boolean
onOptionsItemSelected(MenuItem) boolean
onResume() void
setListAdapter(BaseAdapter) void
onActivityResult(int, int, Intent) void
onPause() void
scanLeDevice(boolean) void
onItemClick(AdapterView<?>, View, int, long) void
connectToDevices() void
UserDialog
editTextUsername EditText
editTextPassword EditText
Listener DialogListener
onCreateDialog(Bundle) Dialog
onAttach(Activity) void
Login
onCreate(Bundle) void
openDialog() void
applyTexts(String, String) void
WelcomeScreen
onCreate(Bundle) void

Erklärung

Hiermit versichere ich, dass ich meine Abschlussarbeit selbständig verfasst und keine anderen als die angegeben Quellen und Hilfsmittel benutzt habe.

Datum: ……

(Unterschrift)