

Konzeption und Implementierung eines Sachwarmverhaltens von mobilen Kleinrobotern anhand eines Verfolgungsszenarios

STUDIENARBEIT

für die Prüfung zum
Bachelor of Science
des Studiengangs Informatik
Studienrichtung Angewandte Informatik
an der
Dualen Hochschule Baden-Württemberg Karlsruhe

10. April 2017

Name	Manuel Bothner	Simon Lang
Matrikelnummer	8359139	6794837
Kurs	TINF14B2	TINF14B2
Ausbildungsfirma	1&1 Internet SE Brauerstr. 48 76135 Karlsruhe	ifm ecomatic GmbH Im Heidach 18 88079 Kressbronn am Bodensee
Betreuer	Prof. Hans-Jörg Haubner	

Erklärung

(gemäß §5(3) der „Studien- und Prüfungsordnung DHBW Technik“ vom 29. 9. 2015)

Ich versichere hiermit, dass ich die Studienarbeit meiner Studienarbeit mit dem Thema: „Konzeption und Implementierung eines Sachwarmverhaltens von mobilen Kleinrobotern anhand eines Verfolgungsszenarios“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort, Datum

Unterschrift

Ort, Datum

Unterschrift

Abstract

Zusammenfassung

Inhaltsverzeichnis

1	Einleitung	1
1.1	Ausgangslage	1
1.2	Zielsetzung	1
1.3	Erwartetes Ergebnis	1
2	Technische Grundlagen	2
2.1	Robotik	2
2.1.1	Mobile Roboter	2
2.1.2	Sensorik	2
2.1.3	Antriebsarten	2
2.2	LEGO MINDSTORMS	2
2.2.1	Das EV3-System	3
2.2.2	Der EV3-Stein (Steuereinheit)	3
2.2.3	Motoren	4
2.2.4	Sensoren	5
2.2.5	Programmierung	7
2.3	Application (App) Entwicklung	9
2.3.1	Native Apps	9
2.3.2	Web Apps	9
2.3.3	Hybride Apps	10
2.3.4	Plattformübergreifende Entwicklung	10
2.3.5	Xamarin	11
2.3.6	Mono	13
2.3.7	.NET Framework	13
3	Theoretische Grundlagen	15
3.1	Schwarmverhalten	15
3.1.1	Allgemein	15
3.1.2	Vorbilder aus dem Tierreich	15
3.1.3	Szenarien	15
3.1.4	Algorithmen	15
3.2	Kommunikation	15
3.2.1	Grundlagen	15
3.2.2	TCP/IP	15
3.2.3	Wifi	15
3.2.4	Serialisierung	15

4	Projektorganisation	16
4.1	Projektablaufplan	16
5	Konzeption	17
5.1	Anforderungsdefinitionen	17
5.2	Softwarearchitektur	18
5.3	Steuerung	20
5.4	Szenarien	20
5.5	Use Cases	20
5.5.1	Connect	20
5.5.2	Synchronization	21
5.5.3	Szenario	21
5.5.4	Exception	21
5.6	Kommunikation	21
6	Implementierung	31
6.0.1	Kommunikation	31
6.1	App	31
6.2	Backend	31
6.3	Robot	31
7	Evaluation	32
8	Ausblick	33

Abkürzungsverzeichnis

AOT Ahead of Time.

API Application Programming Interface.

App Application.

ARM Acorn RISC Machines.

CLI Common Language Infrastructure.

CLR Common Language Runtime.

CSS Cascading Style Sheets.

DLL Dynamic Linked Library.

HTML Hypertext Markup Language.

IL Intermediate Language.

JIT Just-in-Time.

JSON JavaScript Object Notation.

MVVM Model View ViewModel.

TCP Transmission Control Protocol.

UI User Interface.

XML Extensible Markup Language.

Glossar

Application Programming Interface Eine API ist eine Programmierschnittstelle, die die Anbindung von Software ermöglicht.

C# .

Eclipse .

EV3 .

Framework Ein Framework ist ein Rahmen zur Programmierung mit verschiedenen Softwarekomponenten.

Java .

JavaScript JavaScript ist eine Skriptsprache zur Entwicklung dynamischer Internetseiten.

leJos .

Objective-C .

Template .

TypeScript TypeScript ist eine objektorientierte Programmiersprache von Microsoft basierend auf den ECMA-Script-6 Standards.

Abbildungsverzeichnis

1	Zentrale Komponenten des EV3-Systems	3
2	App Entwicklung	9
3	Xamarin	11
4	Shared Project	11
5	Portable Class Library	12
6	Unterstützung Portable Class Library	12
7	Mono	13
8	.NET Framework Ausführung	14
9	Common Language Runtime	14
10	Softwarearchitektur	19
11	Steuerung	20
12	Connect	21
13	Connection	22
14	Synchronization	23
15	Robot list	24
16	Spectator	25
17	Spectator	26
18	Control	27
19	Control	28
20	Synchron	29
21	Follow	30

Tabellenverzeichnis

1	Eigenschaften der EV3-Motortypen	5
2	Eigenschaften der EV3-Motortypen	7

1 Einleitung

Heutzutage werden viele Arbeitsschritte in der Produktion, als auch Dienstleistungen von Maschinen verrichtet, da diese effizienter Arbeiten und weniger Kosten als Menschen verursachen. Da jede Maschine auf einen spezifischen Arbeitsschritt konfiguriert ist, müssen die verschiedenen Maschinen untereinander wie ein Schwarm agieren. Diese Verhaltensstrukturen kommen ursprünglich aus dem Tierreich, wie Fischeschwärme, Ameisen oder Bienen. Hierbei erledigt jedes Individuum seine zugewiesenen Aufgaben und hält die anderen Parteien auf dem aktuellen Stand.

In diesem Projekt werden diese Verhaltensmuster aus dem Tierreich aufgegriffen und anhand eines Verhaltensszenarios mit Kleinrobotern verwirklicht, die autonom agieren und kommunizieren, um zusammen ihr Ziel zu erreichen. Dabei sollen Konzepte, sowie Algorithmen für Schwarmroboter entstehen, die auch auf andere Szenarien angewendet werden können.

1.1 Ausgangslage

1.2 Zielsetzung

1.3 Erwartetes Ergebnis

2 Technische Grundlagen

2.1 Robotik

Tatsächlich gibt es Die VDI-Richtlinie 2860 von 1990 definiert einen Roboter wie folgt:

»Ein Roboter ist ein frei und wieder programmierbarer, multifunktio- naler Manipula- tor mit mindestens drei unabhängigen Achsen, um Ma- terialien, Teile, Werkzeuge oder spezielle Geräte auf programmierten, variablen Bahnen zu bewegen zur Erfüllung der ver- schiedensten Auf- gaben.«

Auch wenn die Definition einige Eigenschaften eines Roboters ... So beschreibt diese Defi- nition hauptsächlich stationäre Industrieroboter, wie sie in der Automatisierungstechnik verwendet werden, wie beispielsweise Schweiß- oder Lackierroboter in der Automobilferti- gung oder Kommissionierroboter in der Logistik. Die genannten programmierten Bahnen sind dort möglich und sinn- voll, weil der Arbeitsprozess, dessen Teil der Roboter ist, gemeinsam mit dem Roboter und seiner Programmierung gestaltet wird:

2.1.1 Mobile Roboter

Diese Kapitel ... Anders

2.1.2 Sensorik

Diese Kapitel ... Sensoren lassen sich hinsichtliche ihrer Arbeitsweise und ... wie folgt klassifizieren:

- A
-
-
-

2.1.3 Antriebsarten

2.2 LEGO MINDSTORMS

LEGO MINDSTORMS ist eine seit 1988 existierende Produktserie des Spielwarenher- stellers LEGO [vgl. 5, 21]. LEGO MINDSTORMS ermöglicht das Bauen, Programmieren und Steuern verschiedener LEGO Roboter. Dies Roboter bestehen dabei aus gängigen LEGO Teilen die auch in anderen LEGO-Produkten Verwendung finden, sowie speziellen LEGO-Komponenten wie einer zentralen Steuereinheit, Motoren und Sensoren.

2.2.1 Das EV3-System

Der 2013 erschienene EV3 ist das dritte System der LEGO MINDSTORMS Reihe. Die Bezeichnung setzt sich aus EV für Evolution und 3 für die 3 Stufe der LEGO MINDSTORMS-Serie zusammen [vgl. 5, Seite 21].

Im Vergleich zu den Vorgängersystemen verfügt das EV3-System über eine modernere und leistungsfähigere Steuereinheit und auch die anderen elektronischen Komponenten des System wurden an den heutigen Stand der Technik angepasst [vgl. 5, Seite 22].

Die folgende Abbildung X.X zeigt einige der zentralen Komoneten des EV3-Systems, wie die Steuereinheit (EV3-Stein), Motoren und vier Sensoren.

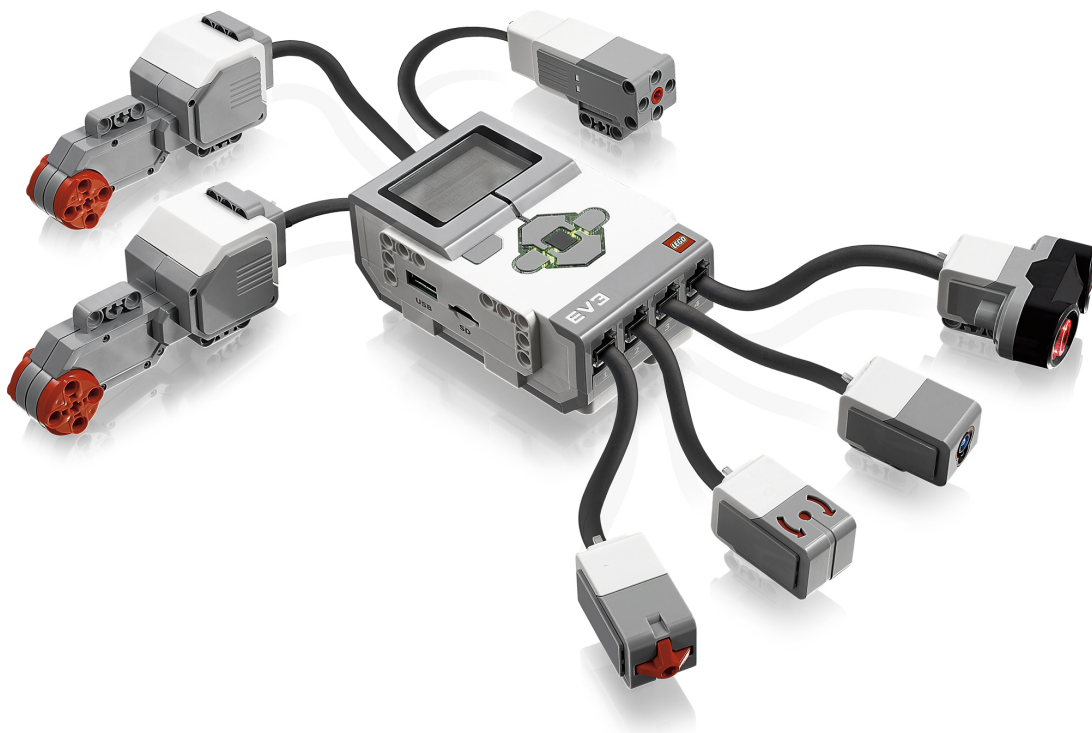


Abbildung 1: Zentrale Komponenten des EV3-Systems

Neben den elektronischen Komoneten gehören auch nicht elektronische Teile wie Verbindungsstücke, Balken und Zahnräder wie sie aus gängigen LEGO Produkten bekannt sind, zum EV3-System. Sie bilden die strukturelle und meschanische Grundlage der Roboter.

Im Folgenden wird auf die elektronischen Komponenten des EV3-Systems näher eingegangen. dieses Projekt eine deutlich größere Relevanz aufweisen.

2.2.2 Der EV3-Stein (Steuereinheit)

Die zentrale Komponenten und das Gehirn des LEGO MINDSTORMS EV3-Systems ist die zentrale Steuereinheit kurz (EV3-)Stein oder auch Brick genannt. Bei ihm handelt es sich um eine Computer welcher selbständig Programme ausführen kann. Dazu verfügt

der EV3-Stein über ein Linux Betriebssystem und eine spezielle Firmware, die wie die auszuführenden Programme auf einem Flash-Speicher liegen [vgl. 5, 21].

Zur Kommunikation mit dem PC verfügt der EV3-Stein über eine USB- sowie Bluetooth-Schnittstelle. Neben der Kommunikation zu einem Computer kann die USB-Schnittstelle auch für den Zusammenschluss mit einem weiteren EV3-Stein (genannt Daisy Chain) genutzt werden [vgl. 5, Seite 21].

Für den Anschluss von Motoren und Sensoren verfügt der EV3-Stein über 8 Ports, an welche die anderen System-Komponenten müber Kabel mit RJ12-Steckern angeschlossen werden. 4 der Ports dienen für den Anschluss von Motoren, die restlichen 4 Ports für die Abfrage von Sensorwerte [vgl. 5, 21].

Der EV3-Stein besitzt an der Vorderseite ein LCD-Display zur Anzeige von Texten und Grafiken sowie 6 Knöpfe für die Bedienung durch den Benutzer. Display und Knöpfe dienen zur Bedienung der Firmware sowie zur Tätigkeit von Einstellungen, können aber ebenso durch Programmen angesprochen und ausgewertet werden Start Mein Kanal Trends Abos BIBLIOTHEK [vgl. 5, 21].

Die folgende Auflistung zeigt einige Leistungsmerkmale des EV3-Steins [vgl. 5, 4, Seite 23 f., Seite 32].

- Prozessor: ARM9 32Bit, 300 MHz, 16 MB Flash 64MB RAM
- Betriebssystem: Linux
- Sensoranschlüsse: 4x, Analog / Digital bis zu 460,8 Kbit/s
- USB-Schnittstellen: 2x, für Kommunikation zum PC, Daisy Chain, WiFi-Stick, USB-Speichermedium
- SD-Karten-Lesegerät: 1x, für MicroSD-Karte bis 32 GB
- User-Interface: 6 Knöpfe inkl. Beleuchtung
- Display: LCD Matrix, monochrom, 178 x 128 Pixel
- Kommunikation: Bluetooth v2.1, USB 2.0 (Kommunikation zum PC), USB 1.1 (Daisy Chain)

2.2.3 Motoren

Das EV3-System verfügt über zwei unterschiedliche Motoren, einen großen Motor und einen mittleren Motor. Bei beiden handelt es sich um Servomotoren mit integriertem Rotationssensor, welche von außen angesteuert und abgefragt werden können [vgl. 4, 92]. Die Motoren lassen sich sehr exakt steuern und ermöglichen so einen synchronen Betrieb mehrerer Motoren [vgl. 5, Seite 29 f.].

Die folgende Tabelle zeigt die wichtigsten Eigenschaften der beiden Motoren.

Eigenschaft / Motortyp	Großer Motor	Mittlerer Motor
Winkelgenauigkeit	1 °	1 °
Umdrehungen	160 bis 170 U/min	240 bis 250 U/min
Drehmoment Rotation	20 Ncm	8 Ncm
Drehmoment Stillstand	40 Ncm	12 Ncm
Gewicht	76g	36g

Tabelle 1: Eigenschaften der EV3-Motoren

2.2.4 Sensoren

Zum EV3-System gehören eine Reihe von verschiedenen Sensoren die es den Robotern ermöglichen Informationen über ihre Umwelt zu sammeln sowie ihre Eigenbewegungen zu erfassen. Im folgenden Abschnitt werden die wichtigsten Sensoren mit ihren Leistungsmerkmalen beschrieben.

Farbsensor Der Farbsensor ist ein digitaler Sensor der dazu dient die Lichtintensität sowie verschiedener Farben zu erkennen. Der Sensor kann sowohl aktiv als auch passiv betrieben werden und verfügt dafür über vier unterschiedliche Betriebsmodi [vgl. 4, 101]:

- Farbmodus (passiv) - In diesem Modus erkennt der Sensor 7 verschiedenen Farben.
- RGB-Modus (aktiv) - In diesem Modus sendet der Sensor nacheinander rotes, grünes und blaues Licht aus, je nachdem zu welchem Anteil ein Gegenstand die einzelnen Farben reflektiert wird die Farbe des Gegenstands ermittelt.
- Rotlicht-Modus (aktiv) - Bei diesem Modus wird Rotlicht ausgesendet und die Intensität des reflektierten Lichts gemessen.
- Umgebungslicht-Modus (passiv) - Bei diesem Modus wird die Intensität des in das Sensorfenster eindringende Umgebungslichts gemessen.

Eigenschaften:

- Erkennung der Farben: keine Farbe, Schwarz, Blau, Grün, Gelb, Rot, Weiß, Braun
- Abtastrate: 1.000 Hz
- Entfernungsreichweite: 15 bis 50 mm

Durch diesen Sensor wird es beispielsweise möglich den Roboter einer farbigen Linie auf dem Boden zu folgen.

Ultraschallsensor Diese aktive Sensor verwendet für den Menschen unhörbaren Ultraschall um die Entfernung von Objekten zu ermitteln. Der Sensor emittiert dazu Ultraschall und misst die Laufzeit der Schallwellen, wenn diese von einem Objekt reflektiert werden, aus der Laufzeit kann dann die Entfernung ermittelt werden. Der Sensor verfügt über zwei unterschiedliche Betriebsmodi [vgl. 5, 32 f.]:

- Messen - In diesem Modus sendet der Sensor Ultraschall aus um die Entfernung von Objekten zu ermitteln.
- Scannen - In diesem passiven Modus emittiert der Sensor selbst keinen Ultraschall, sondern er reagiert auf »fremden« Ultraschall und kann so einen anderen aktiven Ultraschallsensor erkennen.

Eigenschaften:

- Genauigkeit: ± 1 cm
- Messbereich: 3 cm bis 250 cm

Berührungssensor Der Berührungssensor ist ein einfacher mechanischer Sensor. Wird der Knopf am Ende des Sensors gedrückt wird dies registriert. Trotz der Einfachheit dieses Sensors ist dieser dennoch sehr nützlich, da er beispielsweise die Kollision des Roboters mit einem Hindernis erkennen kann [vgl. 5, 33].

Kreiselsensor (Gyroskop) Der Kreiselsensor ermöglicht es Drehbewegungen um eine Achse über Rotationsgeschwindigkeit und Drehwinkel zu messen. Dadurch wird es möglich die Eigenbewegung des Roboters oder einer Roboterkomponente zu registrieren [vgl. 5, 33].

Eigenschaften:

- Genauigkeit: $\pm 3^\circ$ (bei einer 90° Drehung)
- Geschwindigkeit: maximal 440 Grad/Sekunde
- Abtastrate: 1.000 Hz

Rotationssensor (Integriert) Wie bereits im Abschnitt X.X dargelegt verfügen die beiden Motortypen über integrierte Rotationssensoren die es ermöglichen, die Umdrehungen der Motoren auszulesen. Durch diese Sensoren ist es möglich durch Odometrie Rückschlüsse über die Bewegung bzw. Position des Roboters zu schließen.

Eigenschaften:

- Genauigkeit: 1°

- Umdrehungen: Motorabhängig

Neben den hier vorgestellten Sensoren existiert noch ein Infrarotsensor, welcher in Verbindung mit einer Infrarotfernsteuerung dazu dient einen EV3-Roboter fernzusteuern.

2.2.5 Programmierung

Für die Programmierung der LEGO MINDSTORMS Produkte gibt es eine Reihe unterschiedlicher Programmiersprachen und -umgebungen. Die hauseigene LEGO-Software zur Programmierung des EV3 richtet sich an Einsteiger. Sie ermöglicht es über eine grafische Oberfläche via vorgefertigter Programmabläufe welche durch grafische Blöcke repräsentiert werden den EV3 zu programmieren.¹

Die Abbildung X.X gibt einen Überblick über verschiedene für den EV3 verfügbare Programmiersprachen sowie ihre Vor- und Nachteile.

leJOS Das LEGO Java Operating System abgekürzt leJOS ist ein Framework, das es ermöglicht den EV3 mit der Programmiersprache Java zu programmieren. Das leJOS-Projekt wurde 1999 gegründet und sämtliche Komponenten (wie auch Java) sind kostenlos verfügbar [vgl. 4, 21 f.].

leJOS bietet eine schlanke Java Virtual Machine (JVM) für den EV3-Stein sowie eine Klassenbibliothek mit welcher die Komponenten des EV3 (Motoren, Sensoren etc.) angesprochen werden können. Installiert wird leJOS auf einer bootbaren microSD-Karte und kann anschließend davon gestartet werden, ohne die auf dem EV3 vorhandene LEGO-Software zu löschen oder zu verändern [vgl. 4, 23 f.].

Durch leJOS ist es möglich den EV3 mit Hilfe der Hochsprache Java zu programmieren womit eine mächtige Programmiersprache zur Verfügung steht und die Vorteile der Objektorientierung für den EV3 genutzt werden können. leJOS bietet eine umfangreiche

Eigenschaft / Programmiersprache	leJOS	EV3-Software	RobotC	NEPO
Installation	+	++	+	+++
Handhabung	+	++	+	++
Kosten	kostenlos	kostenlos	49\$	kostenlos
Einstieg	0	++	+	+++
Funktionsumfang	++	+	++	++

0 = neutral; + = gut; ++ = sehr gut; +++ = hervorragend

Tabelle 2: Eigenschaften der EV3-Motoren

Klassenbibliothek sowie gut dokumentierte API was unter anderem die Integration von weiteren Sensoren etc. erleichtert [vgl. 4, 23 f.]. Im folgenden sind einige Features die leJOS bietet aufgelistet:

¹[vgl. 4, 25 f.]

- Objektorientierte Programmierung mit Java
- Die meisten Klassen der Pakete `java.lang`, `java.util` und `java.io`
- Rekursion
- Synchronisation
- Multithreading
- Exceptions
- Vollständige Bluetooth unterstützung
- Umfangreiche Klassenbibliothek zum Steuern und Auslesen der EV3-Komponenten
- High-Level-Robotik-Tasks (Navigation, Localization etc.)

2.3 App Entwicklung

Eine **App** ist ein ausführbares Programm für mobile Geräte, wie Smartphones oder Tablets. Um eine **App** für ein mobiles Gerät zu entwickeln, müssen wie für andere Anwendungen im Voraus Anforderungen definiert werden, die diese erfüllen soll. Je nach festgelegten Anforderungen, die an das System gestellt werden, besteht eine bestimmte Anzahl von Möglichkeiten der Entwicklung. Allgemein kennt die **App** Entwicklung drei verschiedene Arten, die native, web und hybride Entwicklung, siehe (2.3.1), (2.3.2) und (2.3.3). Dabei werden verschiedene **Frameworks** verwendet, um mit unterschiedlichsten Programmiersprachen den Aufbau der Logik zu beschreiben. Eine **App** besteht immer aus zwei Teile, dem **User Interface (UI)**, das meist mit einer **Extensible Markup Language (XML)** ähnlichen Sprache beschrieben wird und dem Programmcode, der sich auf viele Klassen verteilt und die Funktionalitäten der **App** beschreiben.



Abbildung 2: App Entwicklung

2.3.1 Native Apps

In der Entwicklung von nativen **Apps** werden die direkten Ressourcen des Gerätes verwendet. Dazu gehört die Laufzeitumgebung des Betriebssystems, Bibliotheken und Hardware-schnittstellen. Der Vorteil von einer nativen Entwicklung liegt hauptsächlich darin, dass diese für das Betriebssystem optimiert ist und die vorhandenen Schnittstellen genutzt werden können, um komplexe und rechenintensive Anwendungen zu ermöglichen.²

Vertreter diese Entwicklung finden sich für verschiedene Betriebssysteme. Der populärste unter ihnen ist bei weitem Android mit einer nativen Java Entwicklung über Android Studio von Google. Sie besitzt aktuell den höchsten Marktanteil und eine entsprechende Popularität unter Entwickler und Nutzer.

2.3.2 Web Apps

Die Entwicklung von web **Apps** arbeitet mit systemübergreifenden Ressourcen und greift auf gängige Webtechnologien, wie Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) und JavaScript zurück. Die **App** wird hierbei nicht wie normale Anwendungen direkt auf dem System des Gerätes ausgeführt, sondern kommt in dessen Browser zur Ausführung. Der Vorteil hierbei ist vor allem, dass diese Art von **App** auf allen Betriebssystemen lauffähig ist und direkt über das Internet veröffentlicht und aktualisiert werden

²[vgl. 1, Unterschiede und Vergleich native Apps vs. Web Apps]

kann, jedoch wird eine stabile Internetverbindung vorausgesetzt.²

Von dieser Entwicklung finden sich viele Vertreter mit der Unterstützung diverser Frameworks. Das populärste unter ihnen ist aktuell AngularJS von Google, was auf JavaScript basiert. In Kombination mit anderen Webtechnologien, wie HTML und CSS lassen sich performante web Apps entwickeln.

2.3.3 Hybride Apps

Die Entwicklung von hybride Apps vereinigt die beiden Entwicklungen von native und web. Sie besteht dabei aus einem nativen Rahmen, in der eine web App zur Ausführung kommt, diese besitzt entsprechende Zugriffsrechte auf Hardwareschnittstellen, um diese mit Application Programming Interfaces (APIs) anzusprechen.³

Diese Entwicklung ist aktuell noch sehr jung, jedoch stechen hier bereits verschiedene Vertreter hervor. Der populärste unter ihnen ist Ionic von Drifty, welches auf Apache Cordova als Basis zurückgreift. In Kombination mit AngularJS, TypeScript und anderen Webtechnologien lässt sich die web App entwickeln und auf einem beliebigen Gerät unter einem nativen Browser ausführen. Es unterstützt dabei verschiedenste Betriebssystem, wie Android, iOS und Windows. Diese Entwicklungen können dabei meist nicht nur mobil, sondern unter anderem auf weiteren Systemen, wie stationäre bereitgestellt werden.

2.3.4 Plattformübergreifende Entwicklung

Um die Entwicklung von Apps einfach zu halten, verwenden immer mehr Entwickler die Form der plattformübergreifenden Entwicklung. Dadurch lässt sich die App unabhängig des Betriebssystems entwickeln und kann somit eine größere Menge von Nutzern erreichen. Diese Entwicklung greift dabei meist auf plattformübergreifende Konzepte, wie eine native Laufzeitumgebung, oder Browser zurück, um darin die App auszuführen. Der große Vorteil in dieser Entwicklung, liegt in der Wiederverwendbarkeit des Quellcodes und der verbesserten Wartbarkeit, da hier lediglich ein Projekt gewartet werden muss und der Quellcode für viele Betriebssysteme übernommen werden kann. Zur plattformübergreifenden Entwicklung wurden die letzten Jahre viele Ansätze mit verschiedenen Frameworks entwickelt. Beispiele hierfür sind Ionic, Unity, Qt oder Xamarin.

³[vgl. 3, Native App, Web App und Hybrid App im Überblick]

2.3.5 Xamarin

Xamarin ist ein **Framework** zur Entwicklung von nativen plattformübergreifenden Apps, welches auf Mono basiert, siehe (2.3.6). Um nativen Quellcode auf den verschiedenen Systemen auszuführen, setzt Xamarin auf verschiedene Softwarekomponenten, um aus einem mit .NET entwickelten Projekt nativen Quellcode zu erzeugen.

Für iOS Systeme verwendet Xamarin den **Ahead of Time (AOT) Compiler**, um aus einem Xamarin.iOS Projekt **Acorn RISC Machines (ARM) Maschinencode** zur erzeugen, der entsprechend schnell auf dem System ausgeführt werden kann.⁴ Bei Android hingegen wird der Quellcode in **Intermediate Language (IL)** übersetzt, welches **Just-in-Time (JIT)** nutzt um zur Laufzeit Maschinencode für das entsprechende Gerät zu erzeugen.⁴ Dazu nutzt Xamarin Softwarekomponenten, während der Laufzeit, um bestimmte Prozesse, wie Speicherverwaltung und Plattformoperationen. Zur Entwicklung bringt Xamarin eine große Bandbreite von Funktionalitäten für den Entwickler, wie Bibliotheken, eine Test Cloud, sowie Unterstützung von nativen Bibliotheken, wie für **Java** oder **Objective-C**. Xamarin bietet Unterstützung für diverse Betriebssysteme, wie zum Beispiel Android, iOS, Windows und Windows Phone. Um mit Xamarin zu entwickeln, gibt es aktuell verschiedene Möglichkeiten mit Unterstützung auf unterschiedlichen Betriebssystemen. Einerseits kann mit Xamarin Studio auf einem OSX System, oder mit Visual Studio auf Windows und Linux entwickelt werden. Wie viele andere **Frameworks**, bietet auch Xamarin verschiedene nützliche **Templates**, die jeweils andere Nutzen besitzen. Diese bauen dabei auf zwei Hauptkomponenten von Bibliotheken, einerseits **Shared Projects** und **Portable Class Libraries**.



Abbildung 3: Xamarin

Shared Projects ermöglichen dem Entwickler Quellcode für verschiedene Plattformen zu entwickeln, wobei die plattformspezifischen Projekte das entsprechende shared Project referenzieren. Somit besitzt diese Projektart keinen direkten Output, sondern kopiert den Quellcode in das zu entsprechend bauende Projekt, siehe Abbildung (4).⁵ Der Hauptunterschied zu Standardprojekten liegt vor allem dar-

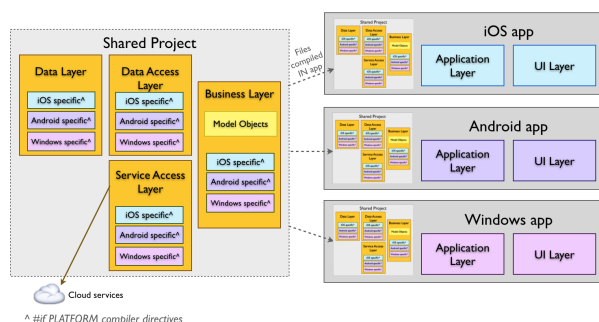


Abbildung 4: Shared Project

⁴[vgl. 6, Introduction to Mobile Development - Xamarin]

⁵[vgl. 8, Shared Projects - Xamarin]

in, dass ein shared Project keine Abhängigkeiten haben darf und daher lediglich als Referenz für andere Projekte dienen kann.

Portable Class Libraries ermöglichen dem Entwickler die Implementierung von plattformübergreifende Bibliotheken, aus denen **Dynamic Linked Libraries (DLLs)** erzeugt werden können. Das Besondere an Portable Class Libraries ist dabei, dass die Plattformen spezifisch ausgewählt werden können, wobei auf die Unterstützung verschiedener Betriebssysteme zu achten ist, siehe Abbildung (6).⁶ Eine Portable Class Library besitzt darüber hinaus verschiedene Vor- bzw. Nachteile, die für oder gegen ihre Nutzung sprechen.

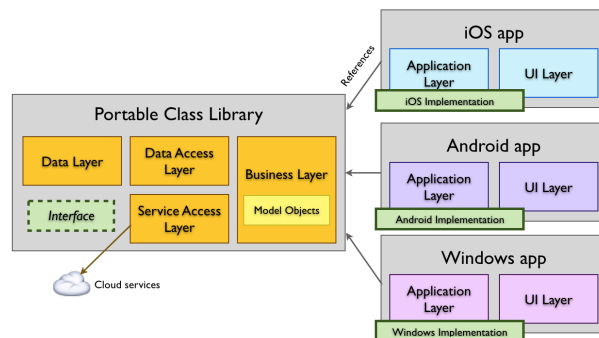


Abbildung 5: Portable Class Library

Vorteile:

- Implementierung von zentralem Quellcode
- Einfaches Refactoring
- Referenzierung von Anwendungen

Feature	.NET Framework	Windows Store Apps	Silverlight	Windows Phone	Xamarin
Core	Y	Y	Y	Y	Y
LINQ	Y	Y	Y	Y	Y
IQueryable	Y	Y	Y	7.5 +	Y
Serialization	Y	Y	Y	Y	Y
Data Annotations	4.0.3 +	Y	Y		Y

Nachteile:

- Keine Referenzierung von Plattform spezifischem Quellcode
- Keine Standardbibliotheken vorhanden

Abbildung 6: Unterstützung Portable Class Library

⁶[vgl. 7, Introduction to Portable Class Libraries - Xamarin]

2.3.6 Mono

Mono ist ein opensource Framework, das auf dem .NET Framework von Microsoft basiert. Die Implementierung von Mono greift dabei auf die Standards von .NET für die Programmiersprache C#, sowie die Common Language Infrastructure (CLI) zurück.⁷ Dies ermöglicht Entwicklern die Erstellung von plattformübergreifenden Anwendungen, welche mittels einer zur Verfügung gestellten Laufzeitumgebung auf verschiedenen Systemen ausgeführt werden können.

Um Anwendungen auf verschiedenen Systemen auszuführen, nutzt Mono verschiedene Komponenten. Dazu gehört an vorderster Stelle ein Compiler, um den erstellten Quellcode in die jeweilige Maschinsprache zu übersetzen. Die Übersetzung findet dabei in Kooperation der Mono Runtime statt, die die entsprechende Infrastruktur zur Ausführung der Anwendung bereitstellt. Für eine effiziente Entwicklung stellt Mono zwei Bibliotheken zur Verfügung, einerseits die .NET Class Library, die die Grundelemente von .NET enthält, sowie die Mono Class Library mit zusätzlichen Funktionen für plattformübergreifende Anwendungen.

Für die Nutzung von Mono stehen verschiedene Vorteile im Vergleich zu anderen Framework. Der Hauptgrund für die Nutzung liegt vor allem in der Popularität von .NET, da dies auf den meisten Computern zur Verfügung steht, oder installiert werden kann. Ein großer Nutzen stellt die High-Level Programmierung dar, die eine Implementierung mit einer Laufzeitumgebung ermöglicht, die Funktionen wie Speicherverwaltung selbst organisiert. Durch Verwendung der Common Language Runtime (CLR) kann der Entwickler seine übliche Programmiersprache verwenden und ist unabhängig vom bestehenden System.



Abbildung 7: Mono

2.3.7 .NET Framework

Das .NET Framework dient zur Entwicklung, sowie Ausführung von Anwendungen, die mit Programmiersprachen implementiert werden, die auf den Standards von .NET basieren. Es besteht aus verschiedenen Komponenten, wobei der Kern des Frameworks in der CLR liegt. Diese ist verantwortlich für die Laufzeitumgebung und entsprechend für die Ausführung der Anwendungen, indem es die bereitgestellten Ressourcen des Systems nutzt.

⁷[vgl. 2, About Mono]

Die CLR führt zur Laufzeit je nach System verschiedene Aktionen aus, um die entsprechende Anwendung auszuführen. Der allgemeine Ablauf ist dabei folgender. Der Quellcode wird in die CLR geladen und nach Sicherheitsanforderungen entsprechend des Systems überprüft. Anschließend wird er durch eine JIT Kompilierung in einen IL Quellcode konvertiert, um diesen nativ auf dem System ausführen zu können. Der IL Quellcode setzt dabei auf die gesetzten Standards der CLI auf, die eine sprach- und plattformunabhängige Entwicklung von Anwendungen ermöglicht.

Das .NET Framework bietet zusätzlich zur unabhängigen Entwicklung verschiedene unterstützende Komponenten. Die wichtigste unter ihnen ist die .NET Class Library. Diese unterstützt den Entwickler mit einer Sammlung bereits implementierten Quellcode, wie Klassen und entsprechenden Zugang zu systemnahen Schnittstellen. Mit dem .NET Framework lässt sich eine große Bandbreite von Anwendungen entwickeln, von Konsolenanwendungen, grafischen Oberflächen, bis hin zu Webanwendungen.

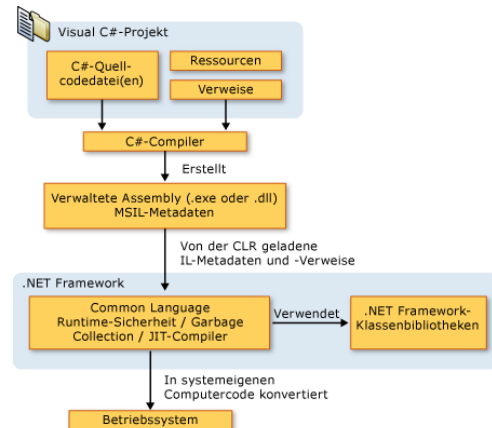


Abbildung 8: .NET Framework Ausführung

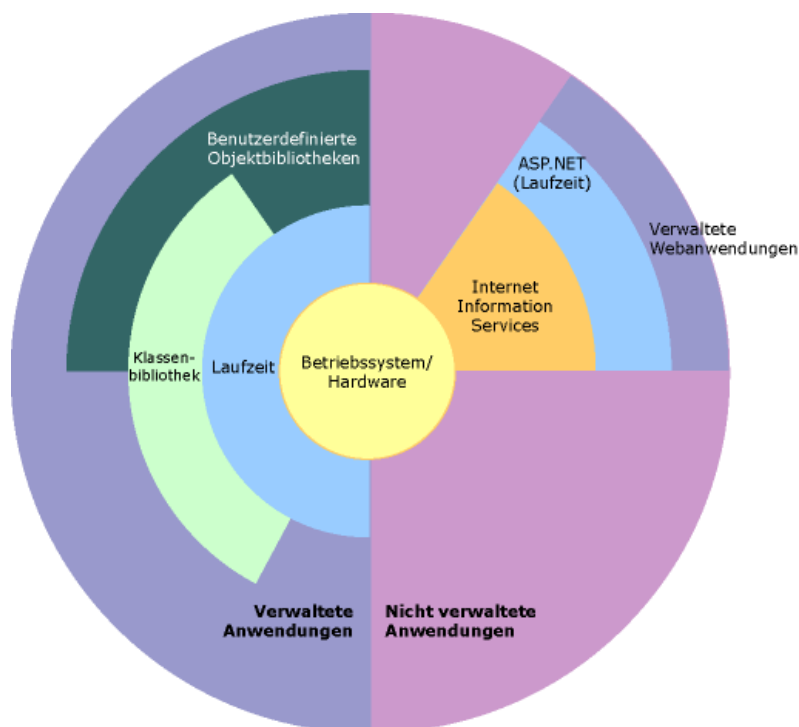


Abbildung 9: Common Language Runtime

3 Theoretische Grundlagen

3.1 Schwarmverhalten

3.1.1 Allgemein

3.1.2 Vorbilder aus dem Tierreich

3.1.3 Szenarien

3.1.4 Algorithmen

3.2 Kommunikation

3.2.1 Grundlagen

3.2.2 TCP/IP

3.2.3 Wifi

3.2.4 Serialisierung

4 Projektorganisation

4.1 Projektablaufplan

5 Konzeption

In diesem Kapitel werden die Anforderungsdefinitionen des Projektes, mit Spezialisierung auf die verschiedenen Use Cases beschrieben.

5.1 Anforderungsdefinitionen

Ein Schwarmverhalten zur Interaktion von Kleinroboter benötigt verschiedene Anforderungen, um korrekt untereinander agieren zu können. Die Basis hierbei bildet das Kommunikationssystem zwischen den einzelnen Komponenten, um erfasste Daten zuverlässig zu synchronisieren. Um die Daten entsprechend zu interpretieren benötigt jede Komponente den jeweiligen Aufbau der Kommunikation, damit diese verwertet und Aktionen ausgeführt werden können.

Diese Aktionen repräsentieren den Grundbestandteil des Schwarmverhaltens und sind auf die verschiedenen Systeme verteilt. Die Roboter benötigen hierbei implementierte Funktionen, wie das Ansteuern von Motoren, Sensorik, sowie die Aktualisierung, um erfasste Daten an Nutzer weiterzuleiten. Zur Steuerung dient eine App für mobile Smartphones mit einem UI um verschiedene Szenarien zu starten, sowie die Roboter kontrollieren zu können. Die Kontrollschnittstelle stellt dabei eine Desktopanwendung dar, über die der Nutzer mit den Robotern kommuniziert und Daten zur Steuerung abgreifen kann, wobei mehrere Nutzer zur selben Zeit mit verschiedenen Szenarien unterstützt werden sollen.

Damit bestehen folgende Anforderungsdefinitionen an die zu erstellenden Softwarekomponenten:

- Kommunikationssystem
- Interpretation
- UI
- Steuerungsfunktionen

5.2 Softwarearchitektur

Die Architektur des Schwarmverhaltens besteht aus drei Hauptkomponente, den Robotern, einer Desktopanwendung, sowie einer mobilen App, siehe Abbildung 10.

Diese Komponenten kommunizieren über ein drahtloses Netzwerk mittels **Transmission Control Protocol (TCP)** untereinander, indem diese Zeichenketten als **JavaScript Object Notation (JSON)** versenden. Dadurch lassen sich gesammelte Daten als Objekte kapseln und auf den verschiedenen Systemen entsprechend synchronisieren. Dies geschieht über eine Klassenstruktur, die Kommandos abbildet, durch die die kommunizierten Daten serialisiert und als Objekte dargestellt werden können.

Die Roboter basieren auf dem Java System **leJos**, da durch die bereitgestellten Bibliotheken für **EV3** Systeme eine unkomplizierte Implementierung von Logik möglich ist, sowie eine direkte Unterstützung von **Eclipse** gegeben ist, um erstellte Software zu debuggen. Die Roboter unterstützen für ein Schwarmverhalten klassische Funktionen, um die Daten der vorhandenen Sensorik auszulesen, sowie Motoren anzusteuern. Diese Funktionen werden einerseits durch Kommandos ausgeführt um den entsprechenden Roboter zu steuern. Andererseits werden regelmäßig Daten durch einen Prozess erfasst, um diese auf dem Backend zu aktualisieren.

Die App beruht auf der plattformübergreifenden Implementierung mittels des Frameworks **Xamarin** um möglichst viele Systeme zu erreichen. Sie baut dabei auf ein einfaches UI mit dem Design Pattern **Model View ViewModel (MVVM)** auf, um diese von der eigentlichen Logik zu trennen und einen qualitativ hochwertigen Quellcode zu schaffen, der einfach gewartet werden kann. Die App besitzt Basisfunktionen zur Erstellung von Kommandos, die die Verwaltung von Szenarien veranlassen und steuert somit den Schwarm.

Das Backend dient als Kommunikationsschnittstelle des gesamten Systems und steuert die Kommandos für den Ablauf der Szenarien. Es besitzt ein UI mit **Eclipse** Realisierung zur Anzeige von erfassten Daten der einzelnen Komponenten und stellt diese anhand einer auswählbaren Hierarchie dar.



Abbildung 10: Softwarearchitektur

5.3 Steuerung



Abbildung 11: Steuerung

5.4 Szenarien

Zur Steuerung der verschiedenen Roboter wurden Szenarien entwickelt. Diese repräsentieren, in welchem Kontext die teilnehmenden Roboter gesteuert werden. Dabei gibt es verschiedene Unterschiede, von Anzahl der Roboter, Nutzer, sowie des Ziels des jeweiligen Szenario. Die Steuerung der Szenarien läuft dabei jeweils gleich ab, durch die im mobilen Gerät vorhandenen Gleichgewichtssensoren, werden Parameter zur Steuerung berechnet, die für die Ausrichtung und die Geschwindigkeit des entsprechenden Roboters zuständig ist.

Control

Synchron

Follow

Flee

Catch

5.5 Use Cases

5.5.1 Connect

Im Use Case Connect wird eine erste Verbindung durch die Eingabe der IP-Adresse zum Backend aufgebaut. Dabei sendet die Komponente, ob Roboter oder App eine Abbildung seiner selbst als Objekt dem Backend. Daraufhin startet das Backend die Verbindung

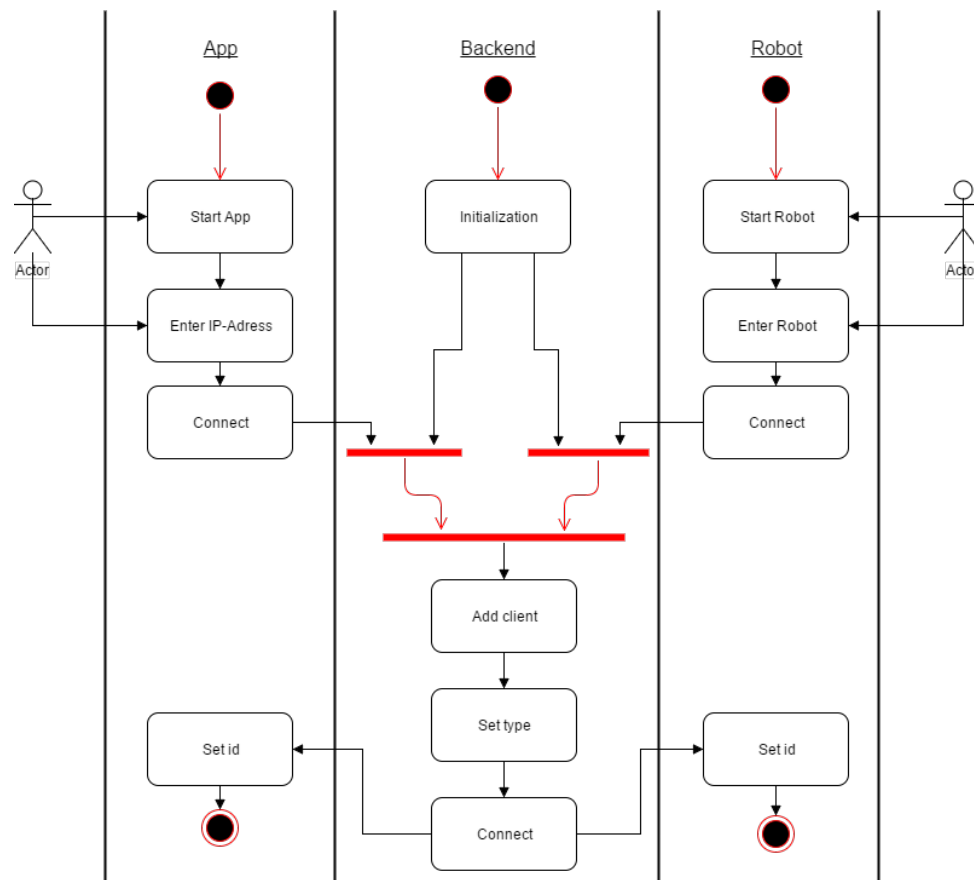


Abbildung 12: Connect

indem es der Komponente entsprechende Verbindungskommandos zusendet. Sobald eine Reaktion in einer festgelegten Zeit erfolgt, akzeptiert das Backend die Verbindung und sendet die entsprechende Id für die Komponente. Ab diesem Moment ist die Komponente verbunden und ein Robot für Aktionen entsprechend verfügbar. Die Verbindungsinitialisierung dient hierbei der Verkürzung der Reaktionszeit, die bei einem Roboter sonst entsprechend hoch wäre.

5.5.2 Synchronization

Im Use Case Synchronization werden Daten entsprechend des gesetzten Typen zwischen den Komponenten übertragen. Dabei können einerseits die Roboter als Objekte, oder ganze Szenarien übertragen werden. Dies dient zur Gegenseitigen Synchronisierung der Daten.

5.5.3 Szenario

5.5.4 Exception

5.6 Kommunikation



(a) Connect



(b) Home

Abbildung 13: Connection

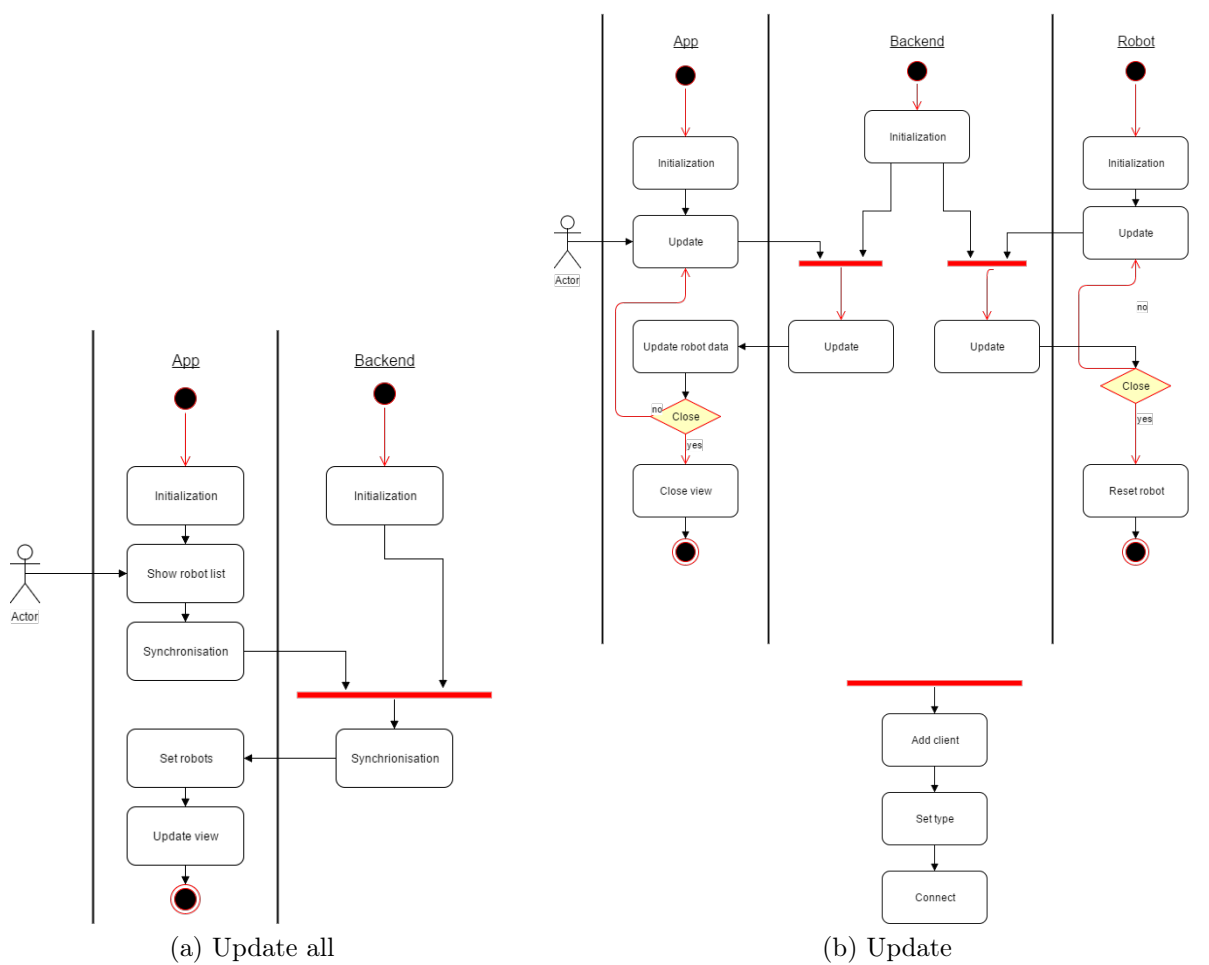


Abbildung 14: Synchronization

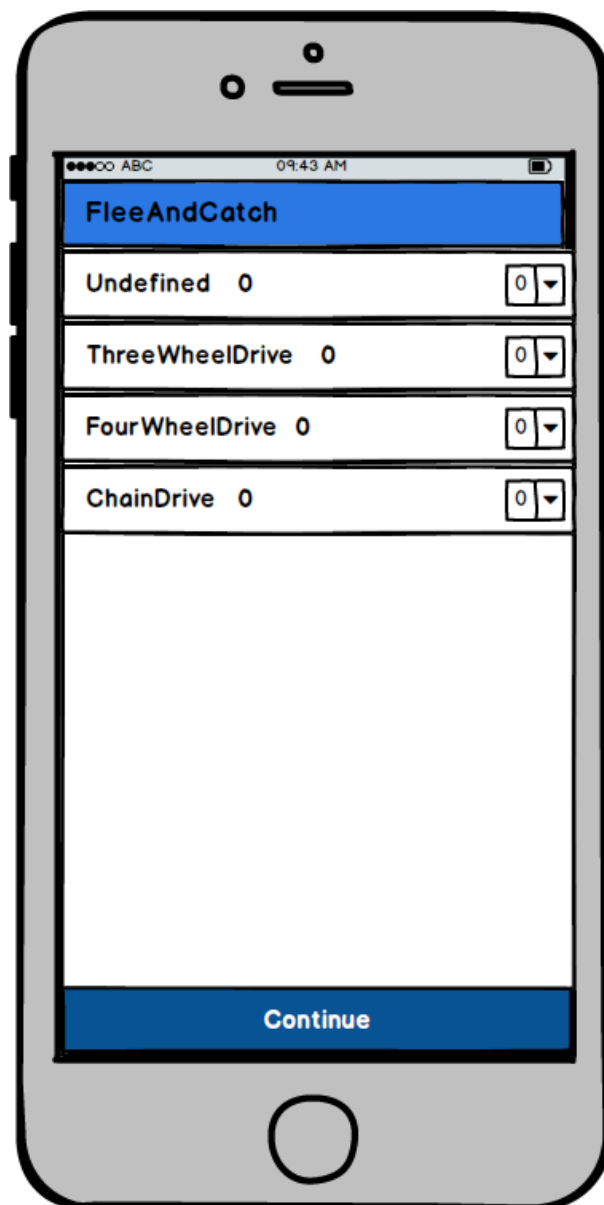


Abbildung 15: Robot list

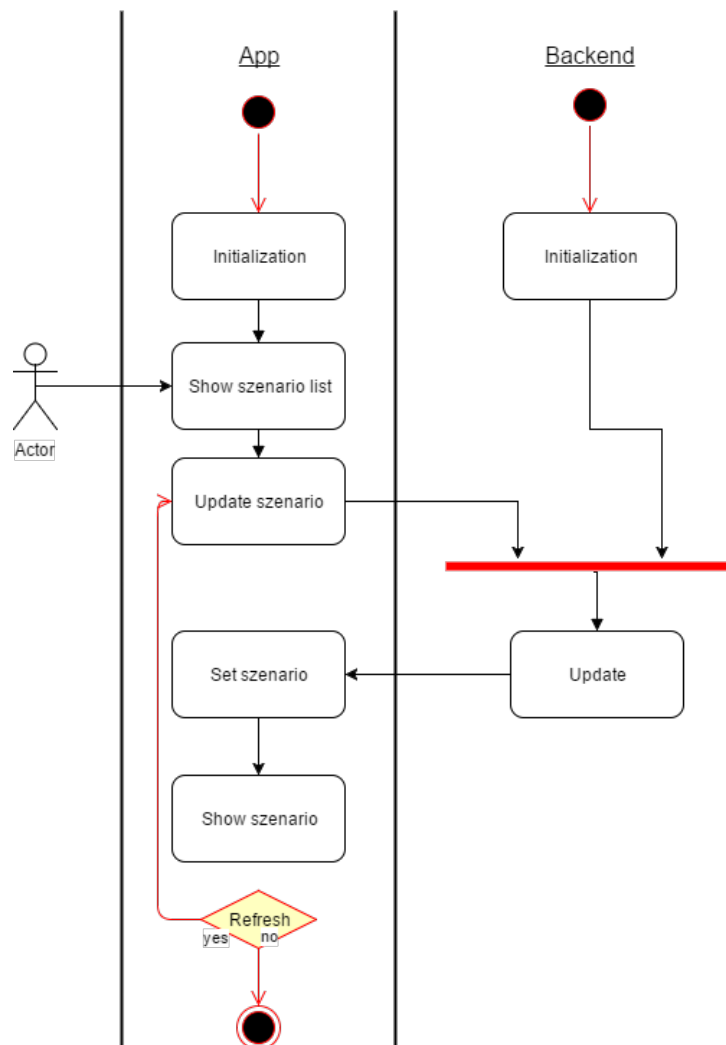


Abbildung 16: Spectator

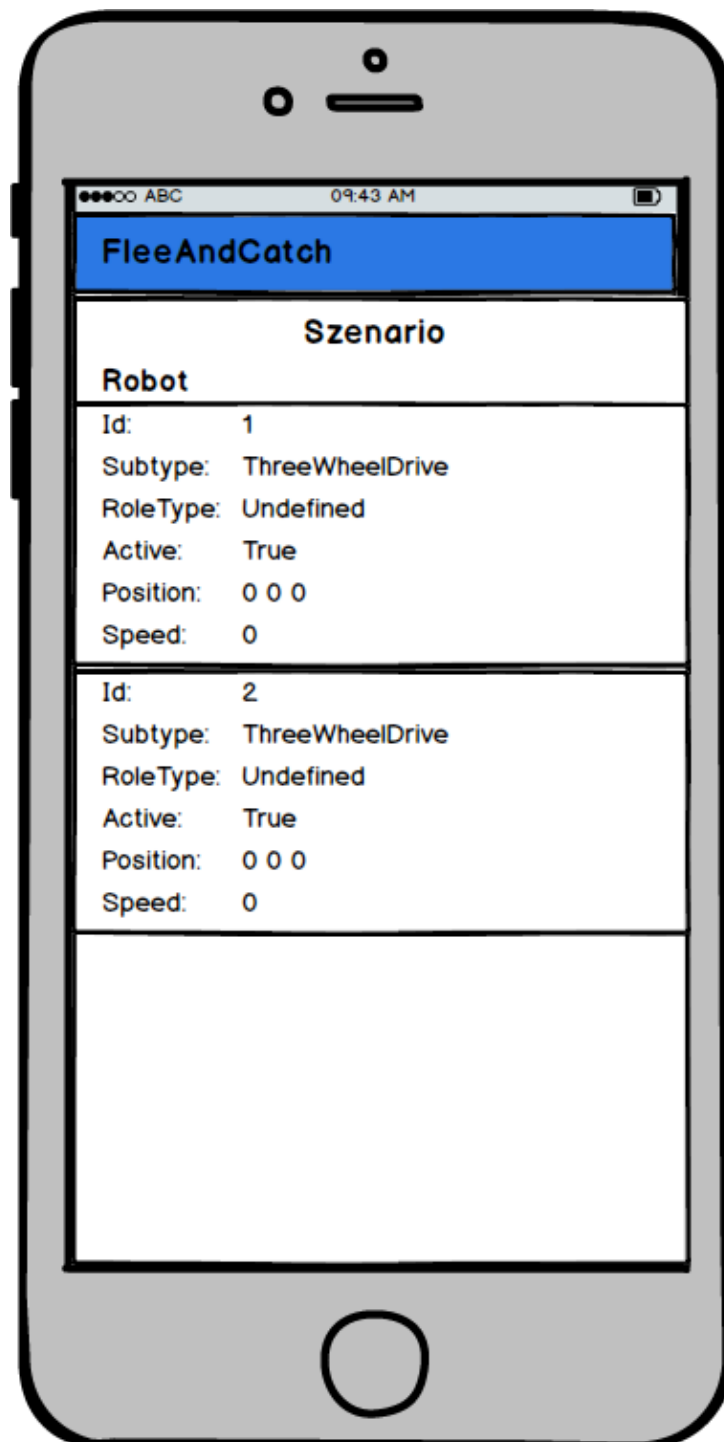


Abbildung 17: Spectator

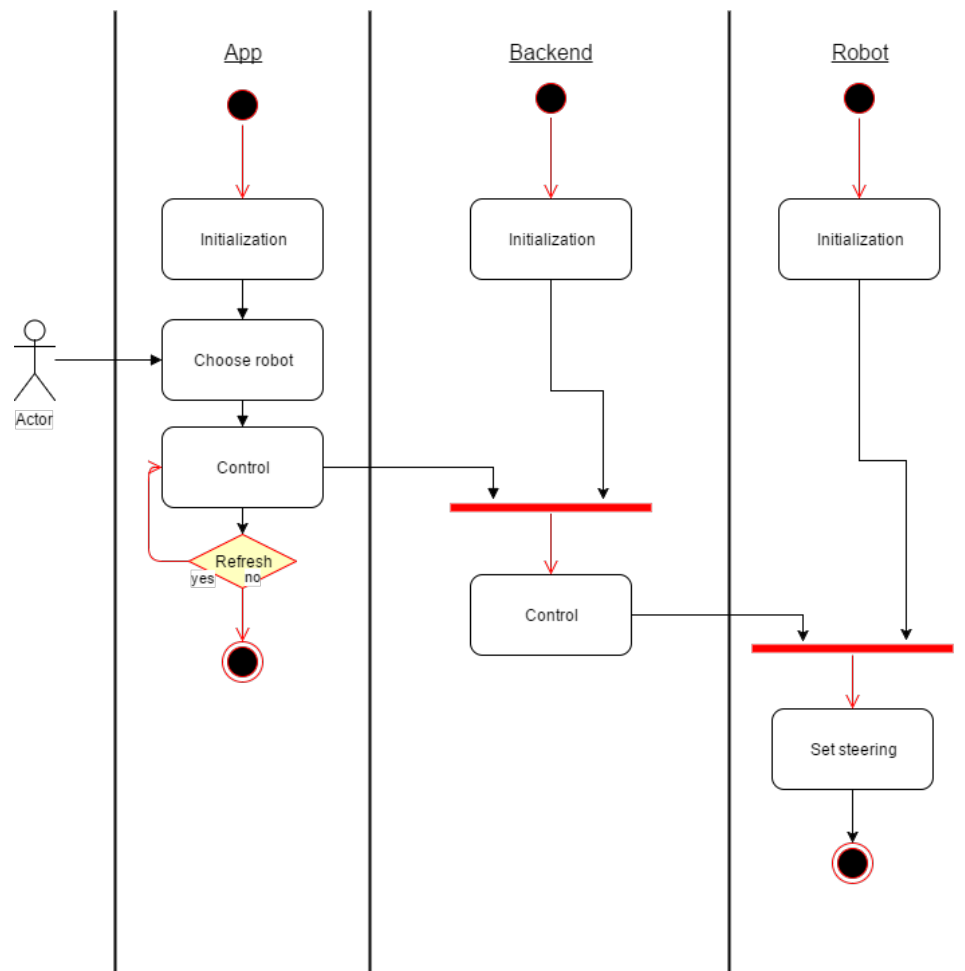


Abbildung 18: Control

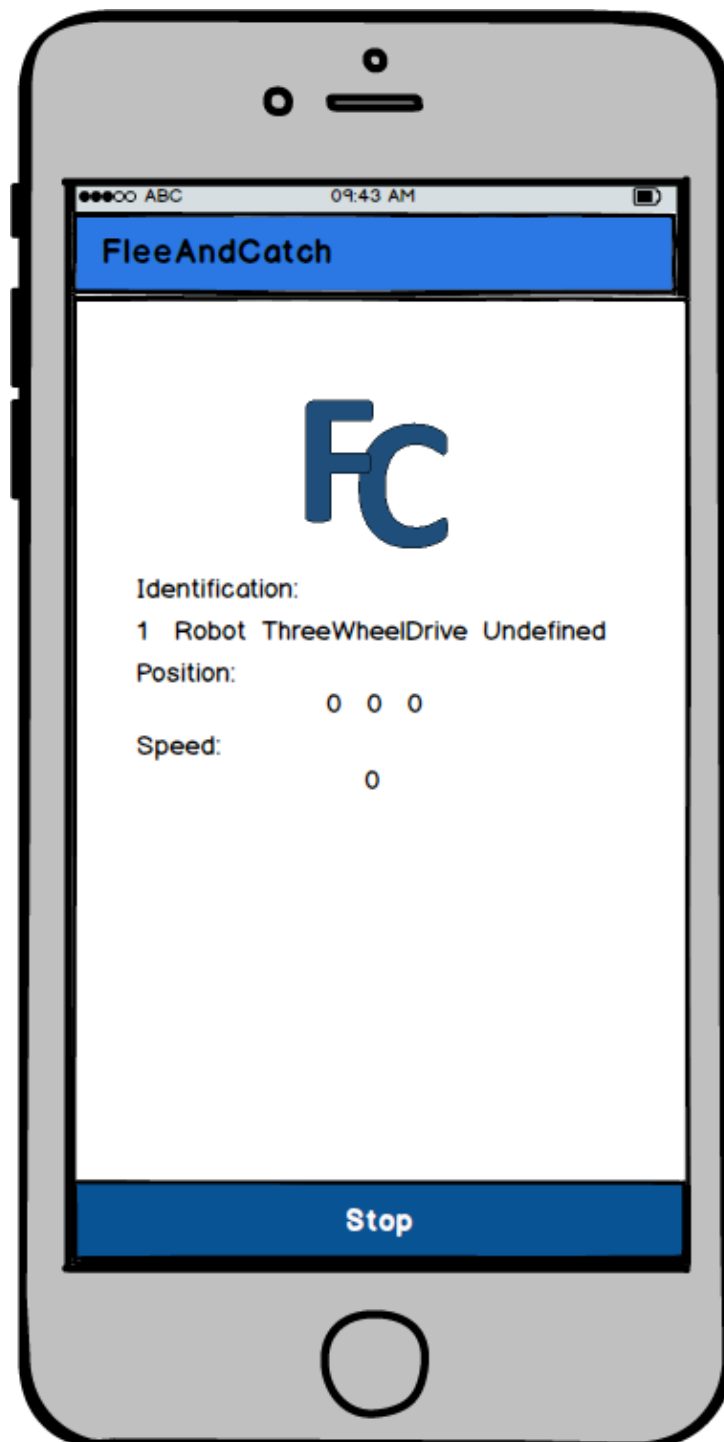


Abbildung 19: Control

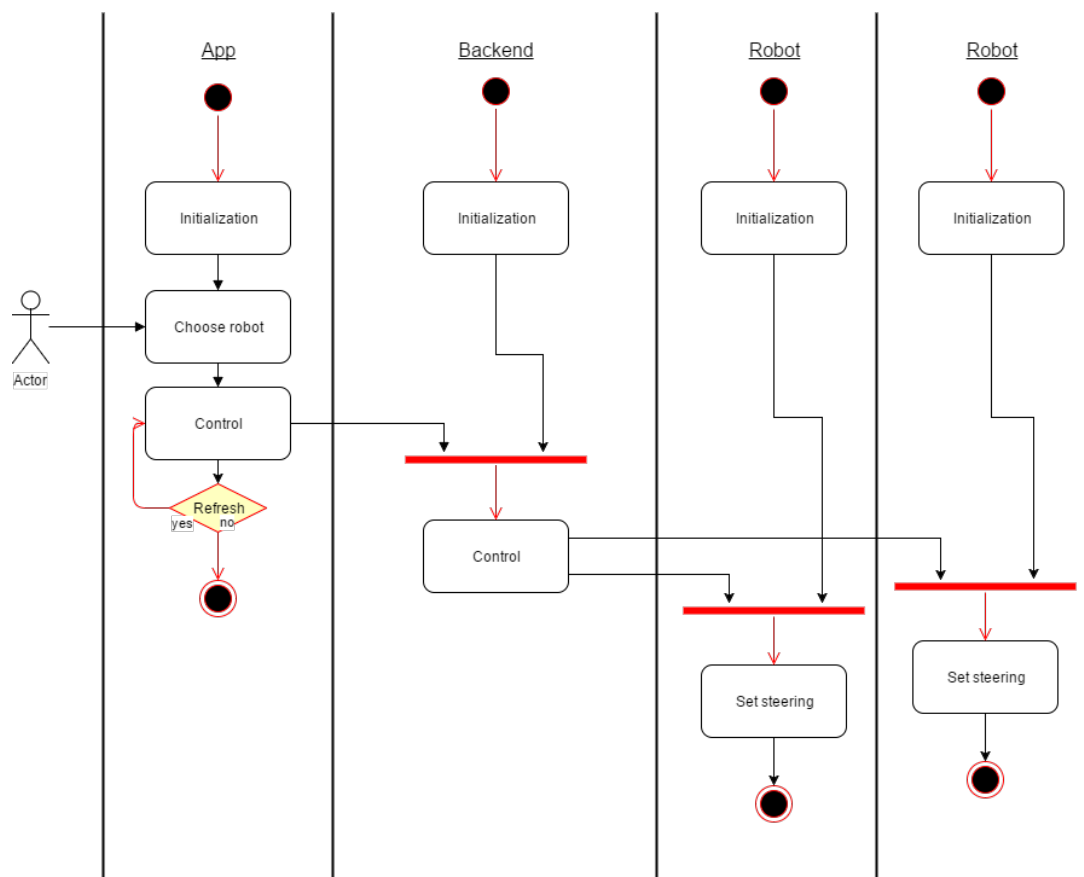


Abbildung 20: Synchron

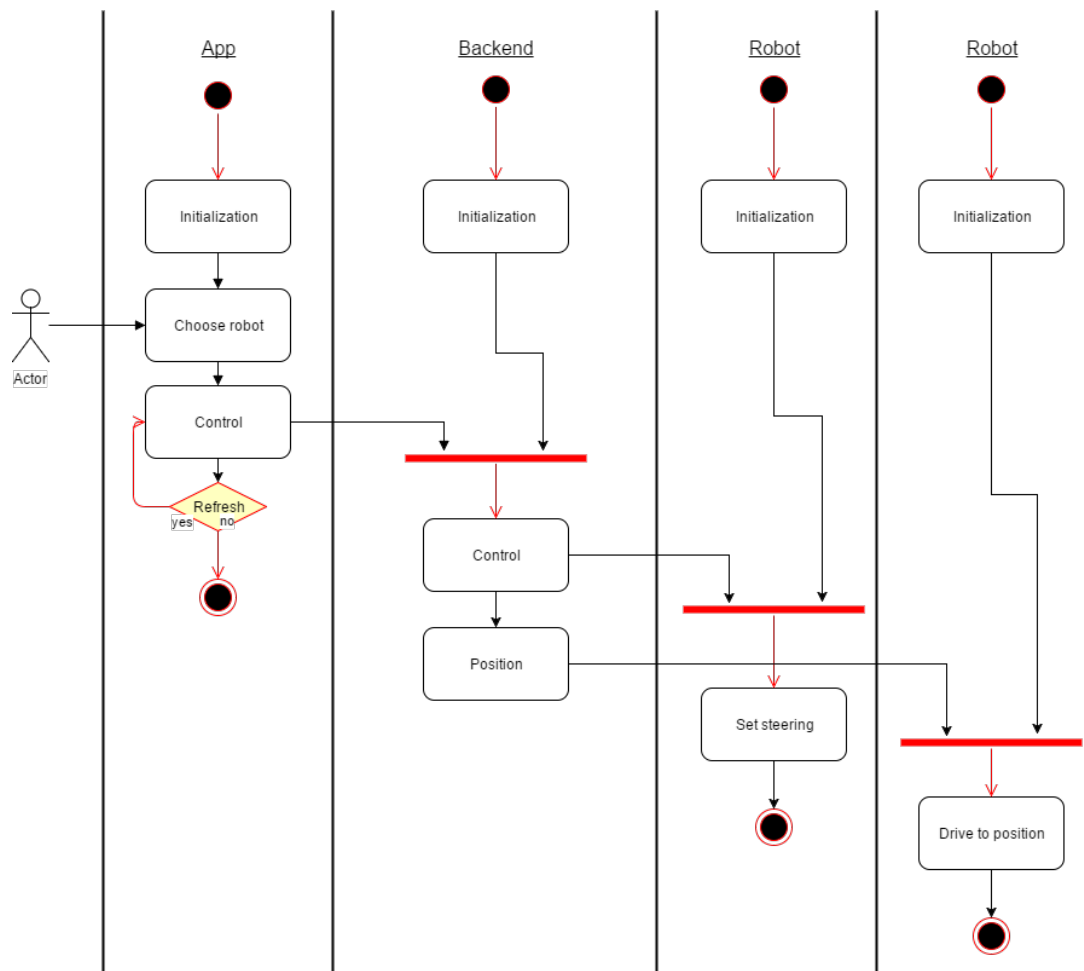


Abbildung 21: Follow

6 Implementierung

6.0.1 Kommunikation

6.1 App

6.2 Backend

6.3 Robot

7 Evaluation

8 Ausblick

Literatur

- [1] Daniel Würstl. Unterschiede und vergleich native apps vs. web apps. URL <http://www.app-entwickler-verzeichnis.de/faq-app-entwicklung/11-definitionen/107-unterschiede-und-vergleich-native-apps-vs-web-apps>.
- [2] Mono project. About mono | mono, 17.03.2017. URL <https://www.mono-project.com/docs/about-mono/>.
- [3] Petra Riepe. Native app, web app und hybrid app im überblick: Warum native wenn es auch hybrid geht? URL <http://www.computerwoche.de/a/warum-native-wenn-es-auch-hybrid-geht,3096411>.
- [4] Maximilian Schöbel, Thorsten Leimbach, and Beate Jost. *Roberta - EV3-Programmieren mit Java*. Roberta - Lernen mit Robotern. Fraunhofer-Verl., Stuttgart, 2015. ISBN 9783839608401.
- [5] Matthias Paul Scholz, Beate Jost, and Thorsten Leimbach. *Das EV3 Roboter Universum: Ein umfassender Einstieg in LEGO MINDSTORMS mit 8 spannenden Roboterprojekten*. 1. auflage edition, 2014. ISBN 3-8266-9473-2.
- [6] Xamarin. Introduction to mobile development - xamarin, . URL https://developer.xamarin.com/guides/cross-platform/getting_started/introduction_to_mobile_development/.
- [7] Xamarin. Introduction to portable class libraries - xamarin, . URL https://developer.xamarin.com/guides/cross-platform/application_fundamentals/pcl/introduction_to_portable_class_libraries/.
- [8] Xamarin. Shared projects - xamarin, . URL https://developer.xamarin.com/guides/cross-platform/application_fundamentals/shared_projects/.

Anhang