DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

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

30. 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	

DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

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

DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

Abstract

DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

Zusammenfassung

DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

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 Fischschwärme, 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 Theoretische und Technische Grundlagen

In diesem Kapitel werden die technischen Grundlagen des Projektes, die zur Umsetzung nötig sind beschrieben.

2.1 Robotik

Die Robotik beschäftigt sich mit dem Entwurf, der Konstruktion sowie der Programmierung, Steuerung und dem Betrieb von Robotern
[http://wirtschaftslexikon.gabler.de/Definition/robotik.html]. Dabei umfasst die Robotik eine Vielzahl von Fachgebieten wie der Elektrotechnik, dem Maschienenbau, der Informatik sowie der Biologie und der Medizin.

2.1.1 Roboter

Was im Kontext der Robotik unter einem Roboter zu verstehen ist, ist gar nicht so einfach darzulegen, da es in Tat keine allgemein anerkannte Definition dieses Begriffs gibt, die seiner üblichen Verwendung entspricht [Mobile Roboter - 2].
Auch ohne eine vollkommen allgemeingültige und präzise Beschreibung eines Roboters zu sein, soll hier die VDI-Richtline 2860 einen Eindruck darüber vermitteln, was in der Robotik mit Roboter gemeint ist.
Die VDI-Richtline 2860 von 1990 definiert einen Roboter wie folgt:

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

Auch wenn diese Definition einige grundlegenden Eigenschaften eines Roboters darlegt, beschreibt diese Definition den Begriff des Roboters im industriellen Kontext und trifft hauptsächlich auf stationäre Industrieroboter zu, wie sie in der Automobilfertigung beispielsweise als Schweiß- oder Lackierroboter verwendet werden. Die genannten programmierten Bahnen sind dort möglich, weil die Arbeitsprozess und die Umgebung auf den Roboter zugeschnitten und vollständig bekannt sind [MR 2].
Im Gegensatz dazu trifft dieser Aspekt auf mobile Roboter nicht zu, sich diese meist in einer unbekannten, unsttrukturierten und dynamischen Umgebung bewegen.

2.1.2 Mobile Roboter

Mobile Roboter bewegen sich selbständig duch eine sich meist ständig ändernde Umwelt, so dass alle ihre Aktionen von ihrer aktuellen Umgebung abhängen und diese erst zum Zeitpunkt der Ausführung dieser Aktionen bekannt ist im Detail. Dieser grundsätzliche

Unterschied zu stationären Robotern macht es unerlässlich das mobile Roboter ihre Umgebung mittels Sensoren erfassen, diese Sensordaten auswerten und auf Grundlage dessen ihre nächsten Aktionen planen.

der Hinsicht, Diese macht es notwendig das die Roboter ihre Umgebung erfassen und wahrnehmen können um auf Veränderungen wie bespielsweise Hindernisse reagieren zu können.

...

Zu den Vertretern mobiler Roboter zählen zum Beispiel Service-Roboter, Erkundungsroboter und Humanoide Roboter. Im Folgenden werden einige Beispielefür mobile Roboter dargelegt:

- **Shakey:** Shakey war ein mobiler Roboter der von 1966 bis 1972 an der am Stanford Research Institute entwickelt wurde. Seine Entwicklung leistete wichtige Beiträge für die Robotik sowie in der KI-Forschung im Bereich der Handlungsplanung und dem selbständigen Lernen [MR 5f].

- **Spirit & Opportunity:** Spirit & Opportunity sind zwei baugleiche Roboter die im Jahr 2003 von der NASA zum Mars geschickt wurden um den Himmelskörper zu erkunden. Die beiden Erkundungsroboter sind Radfahrzeuge mit flexiblem Fahrgestell, verfügen über eine Panoramakamera sowie Sensoren zur Untersuchung des Erdbodens und Gesteins. Obwohl die Roboter in Bezug auf ihrer gundsätzlichen Aktionen von der Erde aus ferngesteuert werden, ist eine autonome Steuerung welche auf kurzfristige, unerwartet Ereignisse wie das Wegrutschen von Rädern reagiert aufgrund der langen Signallaufzeiten unverzichtbar. Die Roboter waren für eine Lebensdauer von 90 Marstagen ausgelegt, übertrafen diese aber bei weitem (mehr als das 30fache) [MR 8f]

- **Stanley:** Stanley ist ein vollständig autonomer Roboter der 2005 am Grand Challenge Wetbewerb teilnahm und diesem gewann. Bei diesem Wettbewerb mussten Fahrzeuge ohne Eingriff von Menschen eine festgelegte, jedoch nicht markierte Strecke von rund 213 km von einem definierten Start- zu einem definierten Zielpunkt zurücklegen. Die Strecke führte durch die Mojave-Wüste in den USA. Bei Stanley handelt es sich um einen modifizierter VW Touareg, dem Sensoren zur Umgebungswahrnehmung und Bordrechner zur Bearbeitung des Kontrollprogramms eingebaut wurden. Stanleys wichtigste Umgebungssensoren waren mehrere Laserscanner und eine Kamera.Stanley meisterte die 213 km lange Strecke welche unter anderem durch felsige oder sandige Bereiche sowie durch Wasserläufe führte in knapp unter 7 Stunden.

- **Tribot** D

Die Beispiele zeigen wie vielfältig ... Neben der Forschung -> Marktpotenzial erwähnen

2.1.3 Sensorik

Um mit der Umgebung interagieren zu können müssen mobile Roboter diese wahrnehmen, dazu dienen Sensoren. Sensoren ermöglichen es dem Roboter Informationen über seine Umwelt und über seinen Zustand zu sammeln. Sensoren lassen sich hinsichtliche ihrer Arbeitsweise und ... wie folgt klassifizieren:

- **Propriozeptive Sensoren** – Diese Art der Sensoren bestimmen eine Messgröße des Roboters selbst und haben keine „Kontakt“ zur Umwelt z.B Bestimmung der Lage Aufgrund eines Neigungssensors.
- **Exterozeptive Sensoren** – Im Gegensatz zu den propriozeptive Sensoren gewinnen diese Sensoren Informationen aus Messgrößen der Umwelt beispielsweise die Bestimmung der Orientierung in Bezug auf die Umwelt.
- **Aktive Sensoren** – Aktive Sensoren senden aktive Energie in ihre Umwellt aus und Erfassen anschließend die zurückkehrenden Signale wie dies beispielsweiße ein Ultraschallsensor tut.
- **Passive Sensoren** – Diese Sensoren senden nicht aktiv aus sondern erfassen ausschließlich die von Natur aus vorhandenen Signale wie z.B. das einfallende Licht durch eine Kamera.

Die folgenden Tabelle zeigt beispielhaft die Einordnung einiger Sensoren:

	Aktive Sensoren	Passive Sensoren
Propriozeptive Sensoren	(Inkrementalgeber (Photoelektrische Abtastung))	Inkrementalgeber, Neigungssensor, Gyroskop
Exterozeptive Sensoren	Ultraschallsensor, Laserscanner, Infrarotsensor, Radar	Kontaktsensor, Kompass, Kamera, GPS

Tabelle 1: Einordnung von Sensoren

2.1.4 Sensordatenverarbeitung

2.1.5 Antriebsarten

2.2 LEGO MINDSTORMS

LEGO MINDSTORMS ist eine seit 1988 existierende Produktserie des Spielwarenherstellers LEGO [vgl. 10, 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. 10, 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. 10, Seite 22].

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

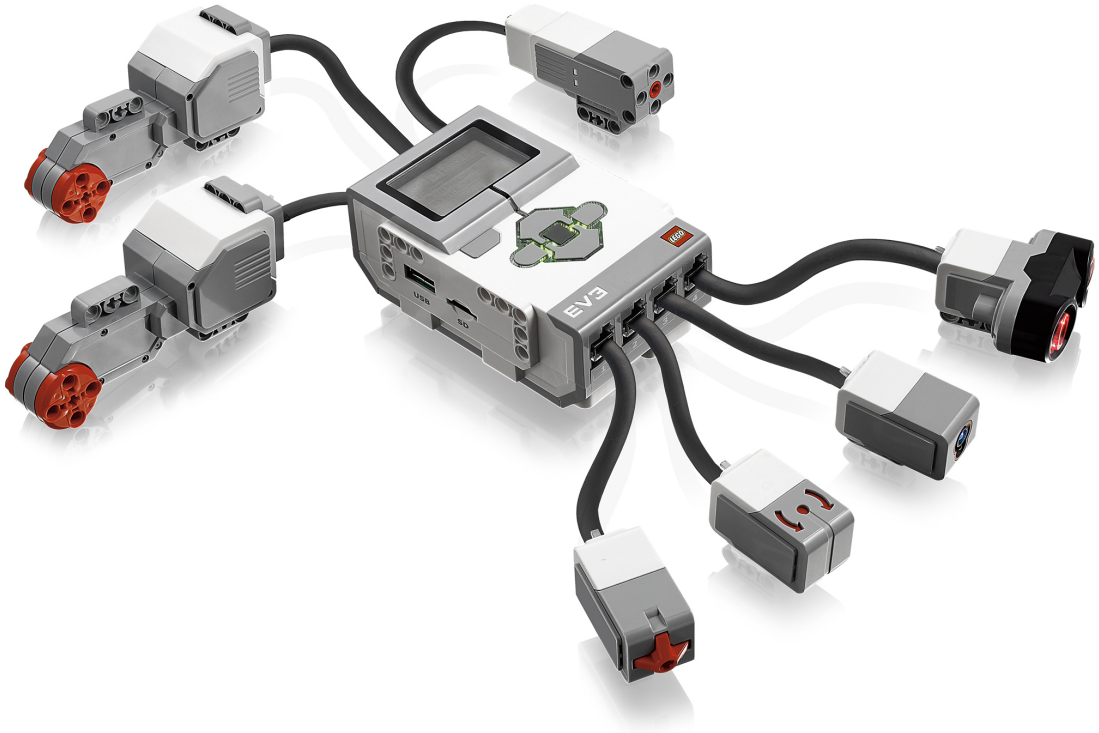


Abbildung 1: Zentrale Komponenten des EV3-Systems

Neben den elektronischen Komponeten 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 Relevanzu 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. 10, 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. 10, 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. 10, 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 Tätigung von Einstellungen, können aber ebenso durch Programmen angesprochen und ausgewertet werden Start Mein Kanal Trends Abos BIBLIOTHEK [vgl. 10, 21].

Die folgende Auflistung zeigt einige Leistungsmerkmale des EV3-Steins [vgl. 10, 9, 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 Servormotoren mit integriertem

Rotationssensor, welche von außen angesteuert und abgefragt werden können [vgl. 9, 92]. Die Motoren lassen sich sehr exakt steuern und ermöglichen so einen synchronen Betrieb mehrerer Motoren [vgl. 10, 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 2: 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 Frabsensor 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. 9, 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 balues Licht aus, je nachdem zu welchem Anteil ein Gegenstand die einzelnen Farben reflektiert wird die Frabe 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 Intesnsität des in das Sensorfenster eindringende Umgebungslichts gemessen.

Eigenschaften:

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

Durch diesen Sensor wird es beispielsweise möglich den Roboter einer frabigen 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 emmitiert dazu Untraschall und misst die Laufzeit der Schallwellen, wenn diese von einem Objekt reflektiert werden, aus der Laufzeit kann dann die Entfernung ermittelt werden. Der Senors verfügt über zwei unterschiedliche Betriebsmodi [vgl. 10, 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 Untraschall, 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 Senors 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. 10, 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. 10, 33].

Eigenschaften:

- Genauigkeit: +/- 3° (bei einer 90° Drehung)
- Geschwindigkeit: maximal 440 Grad/Sekunde
- Abtastrate: 1.000 Hz

Rotationssensor (Integiert) Wie bereits im Abschnitt X.X dargelegt verfügen die beiden Motortypen über initgrierte 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 Infrarotfernsteruerung 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 Programmiersprechen 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. 9, 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. 9, 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 3: Eigenschaften der EV3-Motoren

Klassenbibliothek sowie gut dokumentierte API was unter anderem die Integration von weiteren Sensoren etc. erleichter [vgl. 9, 23 f.]. Im folgenden sind einige Features die leJOS bietet aufgelistet:

[1] [vgl. 9, 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
- Unfangreiche Klassenbibliothek zum Steuern und Auslesen der EV3-Komponeten
- 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 vor Start der Entwicklung Anforderungen definiert, damit die Software spezifisch angepasst werden kann. Je nach Art der Anforderungen die an das System gestellt werden, bestehen verschiedene Möglichkeiten der Entwicklung. Allgemein kennt die App Entwicklung drei verschiedene Arten, die Native-, Web- und Hybride-Entwicklung, siehe Abschnitt (2.3.1), (2.3.2) und (2.3.3). Dabei werden verschiedene Frameworks verwendet, um mit unterschiedlichen 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 [1]

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 Hardwareschnittstellen. 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. Native Apps besitzt aktuellen 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. 2, Unterschiede und Vergleich native Apps vs. Web Apps]

kann, jedoch wird eine stabile Internetverbindung vorausgesetzt.[2]
Diese Entwicklung besitzt 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 perfomante Web-Apps entwickeln.

2.3.3 Hybride Apps

Die Entwicklung von hybriden Apps vereinigt die native- und webbasierte Entwicklung. 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 treten 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 hybride App entwickeln und auf einem beliebigen Gerät unter einem nativen Browser ausführen. Hybride Apps unterstützten dabei verschiedene Betriebssystem, wie Android, iOS und Windows. Diese Apps können dabei meist nicht nur mobil, sondern unter anderem auf weiteren Systemen, wie stationäre zum Beispiel Desktop Rechner bereitgestellt werden.

2.3.4 Plattformübergreifende Apps

Um die Entwicklung von Apps einfach zu gestalten, verwenden immer mehr Entwickler die Form der plattformübergreifenden Entwicklung. Dadurch lässt sich die App unabhängig des Betriebssystem entwickeln und kann somit eine größere Menge von Nutzern erreichen. Diese Entwicklung greift dabei meist auf plattformübergreifende Konzepte, wie eine native Laufzeitumgebung, oder einen nativen Browser zurück, um darin die App auszuführen. Der große Vorteil dieser Apps, 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 in den letzten Jahre viele Ansätze mit verschiedenen Frameworks entwickelt. Beispiele hierfür sind Ionic, Unity, Qt oder Xamarin.

[3][vgl. 8, Native App, Web App und Hybrid App im Überblick]

2.3.5 Xamarin

Xamarin ist ein Framework zur Entwicklung von nativen, plattformübergreifenden Apps. Es basiert auf dem Mono Projekt, siehe Abschnitt (2.3.6), um damit auf verschiedenen Betriebssysteme, wie Android, iOS, Windows und Windows Phone ausgeführt werden kann. Um nativen Quellcode auf den verschiedenen Systemen auszuführen, setzt Xamarin auf verschiedene Softwarekomponenten, um aus einem mit .NET entwickelten Projekt nativen Quellcode zu erzeugen.

Abbildung 3: Xamarin [11]

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, eine plattformübergreifende Assemblersprache, die durch das .NET Framework zur Ausführung gebracht wird. Die Übersetzung in IL geschieht mittels Just-in-Time (JIT), um zur Laufzeit Maschinencode für das entsprechende Gerät zu erzeugen.[4] Damit dies bewerkstelligt werden kann, nutzt Xamarin zur Laufzeit Softwarekomponenten, die bestimmte Prozesse, wie Speicherverwaltung und Plattformoperationen verwalten.

Für eine effiziente Entwicklung bringt Xamarin eine große Bandbreite von Funktionalitäten für einen Entwickler. Beispiele sind dafür Bibliotheken, eine Test Cloud, sowie eine Unterstützung von nativen Bibliotheken, beispielsweise für Java oder Objective-C. Um mit Xamarin zu entwickeln, gibt es aktuell verschiedene Möglichkeiten 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 für verschiedene Zwecke nützliche Templates, die jeweils andere Nutzen besitzen. Die Entwicklung der App baut dabei vor allem auf zwei Hauptkomponenten von Bibliotheken, den Shared Projects und Protable Class Libraries.

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 ko-

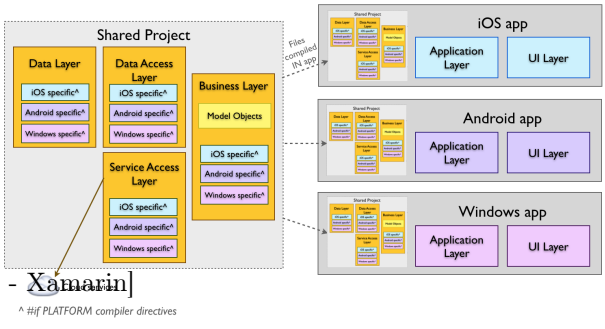


Abbildung 4: Shared Project [14]

[4] [vgl. 12, Introduction to Mobile Development - Xamarin]

piert den Quellcode entsprechend in das zu bauende Projekt, siehe Abbildung (4).[5] Der Hauptunterschied zu Standardprojekten liegt vor allem darin, dass ein Shared Project keine Abhängigkeiten haben darf und daher lediglich als Referenz für andere Projekte dienen kann.

Protable Class Libraries ermöglichen dem Entwickler die Implementierung von plattformübergreifenden Bibliotheken, aus denen Dynamic Linked Librarys (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] Eine Portable Class Library besitzt darüber hinaus verschiedene Vor- bzw. Nachteile, die für oder gegen ihre Nutzung sprechen.

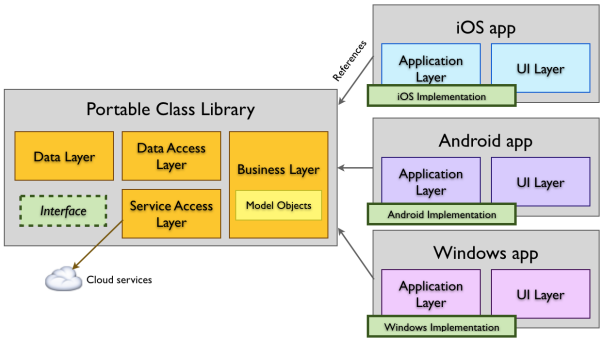


Abbildung 5: Portable Class Library [13]

Vorteile:

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

Nachteile:

- Keine Referenzierung von plattformspezifischen Quellcode
- Keine Standardbibliotheken vorhanden

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

Abbildung 6: Unterstützung Portable Class Library [13]

[5] [vgl. 14, Shared Projects - Xamarin]
[6] [vgl. 13, Introduction to Portable Class Libraries - Xamarin]

2.3.6 Mono

Mono ist ein Open-Source-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.

Abbildung 7: Mono [6]

Um Anwendungen auf verschieden Systemen auszuführen, nutzt Mono verschiedene Komponenten. Dazu gehört an vorderster Stelle ein Compiler, um den erstellten Quellcode in die jeweilige Maschinensprache zu übersetzen. Die Übersetzung findet dabei in Kooperation mit der Mono Runtime statt, welche 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.
Im Vergleich mit anderen Frameworks sprechen verschiedenen Vorteile für die Nutzung von Mono. Der Hauptgrund für die Nutzung liegt vor allem in der Popularität von .NET, da dies auf den meisten Rechnern 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 unhabhängig vom bestehenden System.

2.3.7 .NET Framework

Das .NET Framework dient zur Entwicklung sowie Ausführung von Anwendungen, die mit Programmiersprachen implementiert werden und auf den Standards von .NET basieren. Es besteht aus verschiedenen Komponenten, wobei der Kern des Frameworks in der CLR liegt.[8] Diese ist verantwortlich für die Laufzeitumgebung und somit für die Ausführung der Anwendungen, indem es die bereitgestellten Ressourcen des Systems nutzt.

[7] [vgl. 7, About Mono]
[8] [vgl. 5, Overview of the .NET Framework]

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 entsprechenden Sicherheitsanforderungen des Systems überprüft.[8] Anschließend wird er durch eine JIT Kompilierung in einen IL-Quellcode konvertiert, um diesen nativ auf dem System ausführen zu können.[8] Der IL-Quellcode setzt dabei auf die gesetzten Standards der CLI auf, die eine sprach- und plattformunabhängige Entwicklung von Anwendungen ermöglicht.[8]

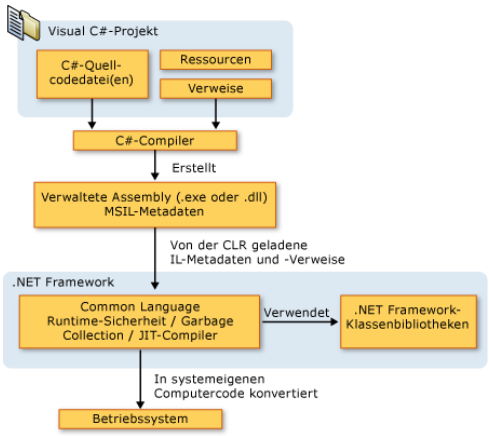


Abbildung 8: .NET Framework Ausführung [4]

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 Quellcodes, wie Klassen und entsprechenden Zugang zu systemnahen Schnittstellen. Mit dem .NET Framework lässt sich eine große Bandbreite von Anwendungen entwickeln, von Konsolenanwendungen, über grafischen Oberflächen, bis hin zu Webanwendungen.

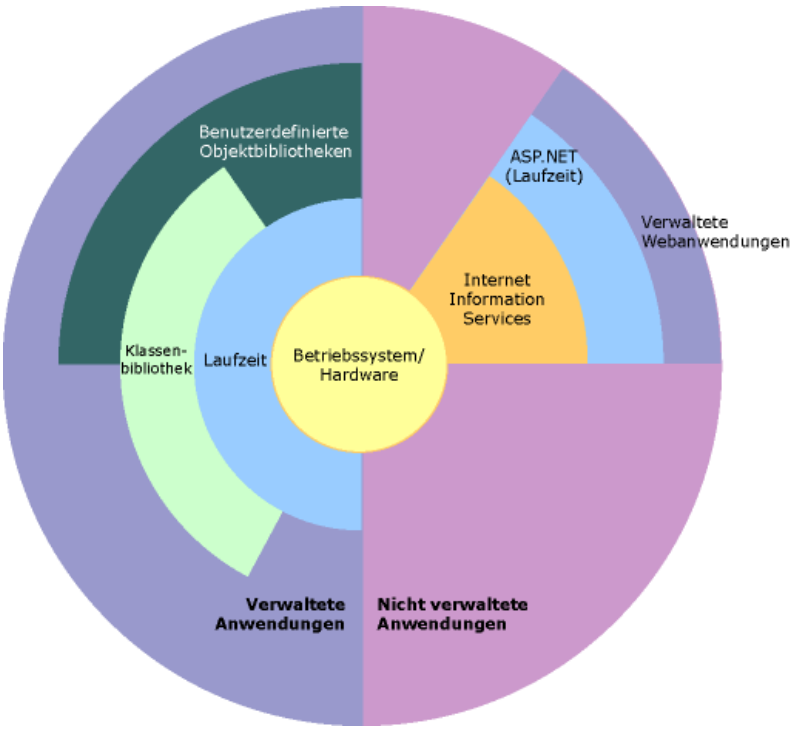


Abbildung 9: Common Language Runtime [5]

2.4 TCP-Kommunikation

Das Transmission Control Protocol (TCP) ist ein Transportprotokoll und ermöglicht eine Datenaustausch zwischen kommunizierenden Anwendungsinstanzen in einer Ende-zu-Ende-Beziehung zwischen. TCP ist als Transportprotokoll auf in der 4 Schicht des OSI-Modells angesiedelt und basiert auf dem Internetprotokoll (IP) mit dem zusammen es als Nahmensgeber der TCP/IP-Protokollfamilie (Internetprotokollfamilie) dient. TCP ist ein offenes, frei verfügbares und weit verbreitetes Protokoll. Als Mitglied der Internetprotokollfamilie ist TCP neben UDP das Transportprotokoll, auf dem die meisten Anwendungen im Internet basieren [GD189].

2.4.1 Gundlegendes

Als verbindungsorientiertes Protokoll sorgt TCP für die Erzeugung und Erhaltung einer gesicherten Ende-zu-Ende-Verbindung zwischen zwei Anwendungsprozessen. TCP arbeitet paketvermittelt d.h. überträgt Daten Paketweise und ist ein zuverlässiges Protokoll. Durch diese Eigenschaften stellt TCP sicher dass, Daten

- nicht verloren gehen
- nicht verändert werden
- nicht dupliziert werden
- in der richtigen Reihenfolge eintreffen

Zur Gewährleistung einer vollständingen Übertragung sowie der Integrität der gesendeten Daten nutzt TCP Prüfsummen,Bestätigungen, Zeitüberwachungs- und Nachrichtenwiederholungsmechanismen sowie Sequenznummern für die Reihenfolgeüberwachung und das Sliding Windows Prinzip zur Flusskontrolle [GD].
TCP nutzt prinzipiell folgende Protokollmechanismen:

- Drei-Wege-Handshake-Verbindungsauf- und -abbau
- Positives, kumulatives Bestätigungsverfahren mit Timerüberwachung für jede Nachricht
- Implizites negatives Bestätigungsverfahren (NAK-Mechanismus): Bei drei ankommenden Duplikat-ACK-PDUs wird beim Sender das Fehlen des folgenden Segments angenommen. Ein sog. Fast-Retransmit-Mechanismus führt zur Neuübertragung des Segments, bevor der Timer abläuft.
- Pipelining
- Go-Back-N zur Übertragungswiederholung
- Fluss- und Staukontrolle

2.4.2 Nagle-Algorithmus

Nagle-Algorithmus (RFC 896 und RFC 1122) ist ein Algorithmus der der Optimierung dient und der bei allen TCP-Implementierungen verwendet wird. Der versuchte Nagle-Algorithmus aus Optimierungsgründen zu verhindern, dass viele kleine Nachrichten gesendet werden, da dies schlecht für die Netzauslastung ist [GD198].
Dazu werden mehrere Nachrichten zusammengefasst und gebündelt versendent, dies geschieht nach folgendem Prinzip:

- Erhält der TCP-Endpunkt Daten vom Anwendungsprozess wird zunächst nur das erste Datenpaket gesendet und die restlichen Daten werden im Sendepuffer gesammelt.
- Danach werden weiteren Daten so lange im Sendepuffer gesammelt bis alle zuvor gesendeten Datenpakete vom Empfänger bestätigt wurden oder so viele Daten im Sendepuffer liegen das die eingstellte Segmentgröße erreicht ist und ein volles Datenpaket gesendet werden kann.

Dieses Verfahren sorgt zwar für eine gute Netzauslastung da das Verhältnis von Nutzdaten zu Overhead (TCP-Header etc.) steigt jedoch ist dies allerdings nicht nicht für alle Anwendungsszenarien optimal da es die Latenz erhöt. Insbesondere bei Anwendungen die eine ummittelbare Antwort der Gegenstelle benötigen wie SSH- oder Telnet-Anwendung sorgt dies für Verzögerungen. In diesem Fall ist es besser den Nagle-Algorithmus auszuschalten. [GD198 f.]

2.4.3 Kommunikationsablauf

Da es sich bei TCP um ein verbindungsorientiertes Protokoll handelt gliedert sich die Kommunikation in die drei Phasen Verbindungsaufbau, Datenaustausch und Verbindungsabbau. Bevor Daten übertragen werden können muss die Verbindung durch den Verbindungsaufbau initiiert und nach Beendinung der Datenbertragung wieder abgebaut werden.

Client & Server Der Verbindungsaufbau einer Kommunikation erfolgt bei TCP nach dem Client-/Server-Paradigma, d.h. einer der Teilnehmern aggiert als Server und wartet auf einen Verbindungsaufbau durch den Client welchen der andere Teilnehmern darstellt. Nach dem Verbindungsauffbau haben die beiden Rollen jedoch keine Bedeutung mehr und die beide Teilnehmern sind sowohl bei der Datenübertragung als auch beim Verbindungsabbau gleichberechtigt.

Verbindungsaufbau
Der Verbindungsaufbau bei TCP basiert auf dem Three-Way-Handshake. Dabei schickt

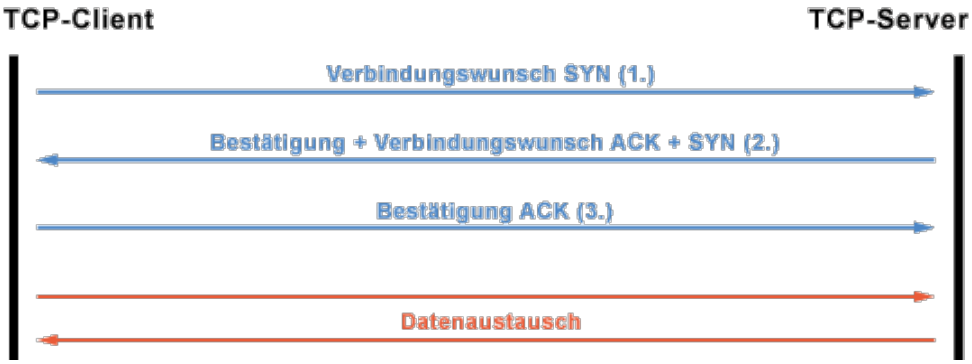


Abbildung 10: TCP Verbindungsaufbau

der schickt der Client einen Verbindungswunsch (SYN) an den Server. Der Server bestätigt den Erhalt der Nachricht (ACK) und äußert seinerseits einen Verbindungswunsch (SYN) welchen der Client nach Erhalt der Nachricht bestätigt (ACK). Nach Abblauf dieses gegenseitigen Anfrage- und Bestätigugsvorgangs ist die Verbindung initiiert und der Datenaustausch zwischen den Teilnehmern kann beginnen [EK].

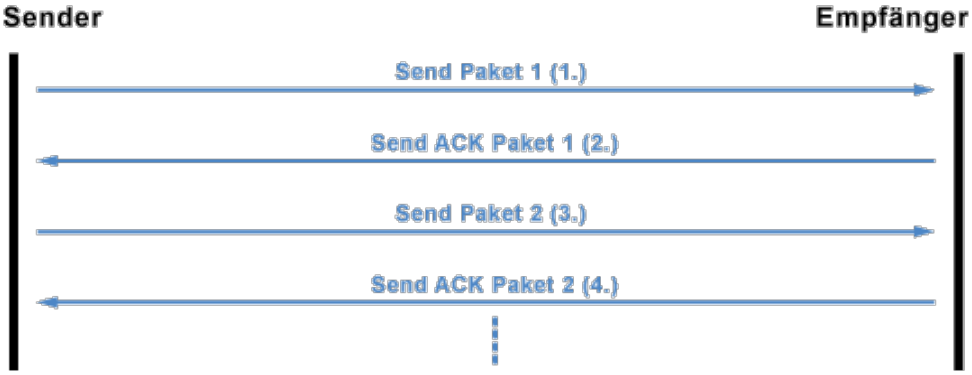


Abbildung 11: TCP Datenaustausch

Datenaustausch

Abbildung 12: TCP Verbindungsabbau

Verbindungsabbau Nach Abschluss der Datenübertragung wird von einer Seite (egal von welcher) ein Verbindungsabbau initiiert. Dazu dient einen etwas modifizierten Drei-

Wege-Handshake-Mechanismus. Jede der beiden Verbindungsrichtungen der Vollduplex-Verbindung wird abgebaut, d.h. beide Seiten bauen ihre „Senderichtung“ ab. Die initiierende Seite schickt zuerst einen Verbindungsabbauwunsch (FIN). Die Gegenstelle bestätigt den Erhalt der Nachricht (ACK) und schickt ebenfalls einen Verbindungsabbauwunsch (FIN) woraufhin sie von der Gegenstelle noch mitgeteilt bekommt, dass die Verbindung abgebaut ist (ACK).

2.4.4 Socket-Programmierung

Als Transportzugriffsschnittstelle für die TCP-basierte Kommunikation dient die Socket-Schnittstelle. Obwohl es sich bei TCP ein paketvermitteltes Protokoll handlet der Anwendung eine Strom-orientierte Kommunikation, die Daten werden also von einem Anwendungsprozess Byte für Byte in einem Bytestrom geschrieben und TCP sorgt anschließend um den Aufbau von Segmenten, die dann übertragen werden. Andere Transportdienste erwarten ihre Daten in festen Blöcken [GD].

2.5 Schwarmverhalten

Dieser Abschnitt beschäftigt sich mit dem Schwarmverhalten, welches mit Roboter imitiert werden soll.

2.5.1 Allgemein

Das Schwarmverhalten beschreibt die Verhaltensweise eines Schwarmes, welcher aus einer einheitlich formierten Tierart besteht. Diese interagieren untereinander, um eine evulotionstechnische Überlegenheit durch das Bilden eines Schwarmes zu erhalten. Der Vorteil liegt dabei hauptsächlich im Schutz vor Fressfeinden des einzelnen Individuums im Schwarm, indem diese nicht als Beutetier identifiziert werden können. Andernfalls kann sich der Schwarm als solches effizienter gegenüber Fressfeinden behaupten und Jungtiere schützen. Zudem dient es einer Erleichterung der Fortpflanzung, da in einem Schwarm eine entsprechende Auswahl an Partnern besteht, um eine genetische Vielfalt zu erreichen.

2.5.2 Vorbilder aus dem Tierreich

Zur Erstellung eines Schwarmverhaltens existieren verschiedene Ideale, die aus dem Tierreich übernommen werden können. Dabei verwendet jede Tierart ihre eigenen Überlebensstrategien, sowie Regeln die in dem anzutreffenden Schwarm vorherrschen.

Ein Fischschwarm ist hierfür eines der bekanntesten Beispiele, indem die einzelnen Individuen des Schwarmes sich so verhalten, als würden sie wie ein einzelnes Tier agieren. Dabei setzt jeder Fisch feste Regeln um, die zur Umsetzung eines Schwarmverhaltens führen.

Abbildung 13: Fischschwarm

1. Folge deinem Vordermann
2. Behalte die Geschwindigkeit deines Nachbarn bei

Die wichtigste Rolle spielt hierbei der Schwellenwert zur Steuerung des Schwarms durch einzelne Teilnehmer. Da jedes Individuum den Schwarm durch seine Bewegung beeinflusst, existiert ein Schwellenwert, der sich einer Mindestanzahl von etwa fünf Prozent richtet, durch die der Schwarm gesteuert werden kann. Somit reagieren die Fische ausschließlich auf die Mehrheit und der Schwarm lässt sich nicht durch

eine Minderheit Fehlsteuern. Dieses Szenario lässt im Sinne von Veranstaltungen auf Menschen übertragen, um zu festzustellen an welchen Positionen Sicherheitspersonal positioniert werden muss, um einen geregelten Ablauf zu gewährleisten.

Ein Termitenschwarm geht nach dem Prinzip der Stigmergie vor, wie andere Insekten, die vorwiegend in einem Staat leben. Dabei kommunizieren die einzelnen Individuen indirekt über die Beeinflussung ihrer direkten Umgebung, wie dem Hinterlassen von Spuren als Merkmale. Dieses Prinzip ermöglicht auch den kleinsten und primitivsten Organismen einen evolutionären Vorteil zu erlagen, indem sie sich nicht allein den Gefahren stellen, sondern in einer großen Masse zusammenarbeiten.

Abbildung 14: Termitenschwarm

Dies ermöglicht das Bauen riesiger Nester, indem die Insekten ihren inneren Plan, mit ihrer aktuellen Umgebung vergleichen und dadurch intuitiv erkennen, welche Arbeit sie zu erledigen haben.Der Ablauf ist dabei folgender:

1. Erkennen
2. Analysieren
3. Reagieren

Dieses Prinzip lässt sich durch festlegen von Regeln auf Roboterschwärme abbilden, um wie Insekten vollkommen autonom Gebäude und andere Objekte zu errichten. Zudem lässt sich dadurch eine indirekte Kommunikation und Zusammenarbeit zwischen Verschiedenartigen Robotern realisieren.

2.5.3 Szenarien

Zur Umsetzung verschiedener Schwarmverhalten lassen sich nützliche Teile der Tierwelt auf Roboter abstrahieren, um eine Zusammenarbeit der einzelnen Individuen effizienter zu gestalten. Hierbei werden meist mehrere Basisszenarien eingesetzt, die einerseits die Kommunikation, oder Abläufe verbessern.
Für die Umsetzung der Kommunikation der einzelnen Teilnehmer existieren zwei verschiedene Möglichkeiten, einerseits eine direkt, oder indirekte Kommunikation. Dabei kann bei einer direkten Kommunikation auf die bekannten Kommunikationsmittel zurückgegriffen werden und über ein existierendes Netzwerk kommuniziert werden. Bei einer indirekte

Kommunikation analysiert der Roboter dagegen seine direkte Umgebung, um daraus Veränderungen zu registrieren und entsprechend darauf zu reagieren. Dabei wird bei einer Umsetzung zur Implementierung folgend vorgegangen:

1. Datenerfassung durch Sensorik
2. Datenauswertung
3. Vergleich der Datensätze
4. Reaktion und Ausführung

1. Separation
2. Aligment
3. Cohesion
4. Ausseneinflüsse

3 Konzeption

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

3.1 Anforderunsdefinitionen

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

3.2 Softwarearchitektur

Die Architektur des Schwarmverhaltens besteht aus drei Hauptkomponente, den Robotern, einer Desktopanwendung, sowie einer mobilen App, siehe Abbildung 15.
Diese Komponenten kommunizieren über ein drahtloses Netzwerk mittels Transmission Control Protocol (TCP) untereinander, indem diese Zeichenketten als 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, siehe 3.6.
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. Anderseits 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 JavaFX Realisierung zur Anzeige von erfassten Daten der einzelnen Komponenten und stellt diese anhand einer auswählbaren Hierarchie dar.

Abbildung 15: Softwarearchitektur

3.3 Steuerung

Die Steuerung des Roboterschwarms greift in sämtlich implementierten Szenarien auf die Sensorik des Smartphones als Basis zurück. Verwendet werden hierbei die Bewegungssensoren um eine Steuerung durch das Hin- und Herschwenken des Smartphones zu ermöglichen. Dies stellt eine intuitive Steuerung dar und ist für jeden neuen Nutzer schnell begreiflich. In Abbildung 16 ist die entsprechende Steuerung zur Bewegung des Roboters dargestellt. Um die Roboter möglichst genau zu steuern, erfasst die Sensorik laufend Daten, welche im UI angezeigt werden. Dadurch lässt sich eine Veränderung der Daten darstellen, die zu einer signifikant verbesserten Steuerung führen.

Abbildung 16: Steuerung

3.4 Szenarien

Zum Ablauf der Software greift der Roboterschwarm auf verschieden definierte Szenarien als Kontext zurück. Diese sind in Control, Synchron, Follow, Flee und Catch untergliedert, wobei ein Single mit einem Nutzer oder einem Mehrnutzersystem als Multi unterschieden wird. Der Multi Mode dient hierbei als Erweiterung zur Software und ist im vorliegenden System nicht implementiert und kann daher nicht genutzt werden. Folgend werden die einzelnen Szenarien beschrieben, die als Kontext für ein Schwarmverhalten genutzt werden können.

Control stellt eine direkte Steuerung eines einzelnen Roboters und fällt somit nicht unter die Kategorie Schwarmverherhalten. Dieses Szenario dient zur Entwicklung der grundlegenden Funktionen, auf denen das Schwarmverhalten und damit weitere Szenarien aufbauen.

Synchron stellt eine synchrone Steuerung von mehreren Robotern dar, in dessen Kontext jeder beteiligte Roboter identische Kommandos erhält. Durch eine entsprechende Aufstellung der Roboter lassen sich Schwärme aus dem Tierreich, wie Fische oder Vögel nachahmen.

Follow stellt eine Reihe von Robotern dar, indem der vorderste vom Nutzer gesteuert werden kann. Die restlichen Roboter erhalten ihrer Position in der Schlange entsprechend der Position ihres Vordermannes, zu dem diese vollkommen autonom fahren. Da diese Ablauf laufend wiederholt wird, stellen alle Roboter gesamt eine Schlange dar, wobei die einzelnen die Muskeln und der vorderste Roboter den Kopf repräsentiert.

Flee stellt ein Verfolgungsszenario dar, indem der Nutzer mit seinem Roboter vor anderen flieht. Dabei erhalten die restlichen Roboter laufend eine Position um immer näher an diesen heranzufahren. Sollte der Nutzer durch einen Roboter erwischt werden, ertönt ein Endsignal , wobei anschließend das Szenario beendet wird.

Catch stellt ein Verfolgungsszenario dar, indem der Nutzer die restlichen Roboter fängt. Diese fahren zufällig in verschiedene Richtungen davon. Sollte der Nutzer alle gefangen haben, ertönt ein Signal und beendet damit das Szenario.

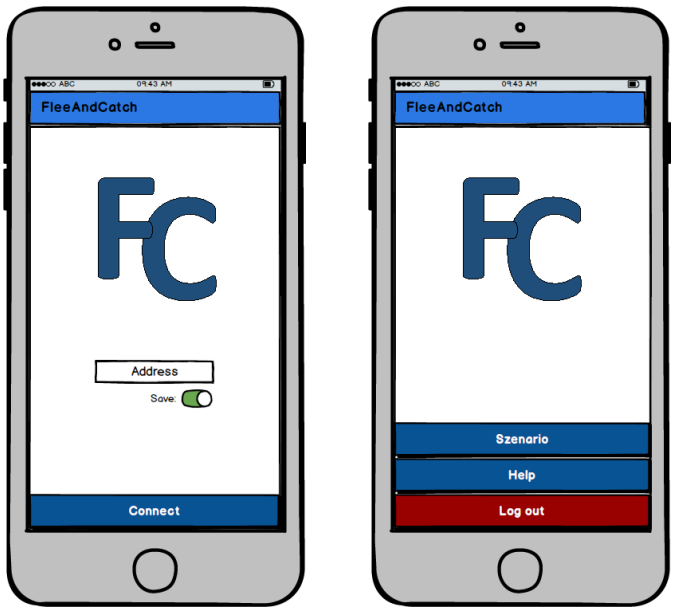
Abbildung 17: Logo

3.5 Use Cases

In diesem Abschnitt werden die Use Cases des Schwarmverhaltens beschrieben. Dabei wird insbesondere auf den Ablauf in Form von UMnified Modeling Language (UML) Diagrammen eingegangen.

3.5.1 Connection

Der Use Case Connection stellt den Ablauf eines Verbindungsaufbaus zwischen den Komponenten und der Desktopanwendung dar. Dies soll über die Nutzung eines drahtlosen Netzwerks mittels TCP Schnittstelle des Smartphones realisiert werden. Die IP-Adresse kann dabei durch die Verwendung einer SQLite Datenbank lokal gespeichert werden, um diese später bei einem erneuten Aufruf automatisch eintragen zu lassen. Zur Unterscheidung der verschiedenen Komponenten soll eine individuelle Identifikation erstellt werden, wobei der Typ der Komponente, sowie weitere Merkmale ersichtlich werden sollen.
Der Use Case soll dabei in zwei unterschiedliche Typen untergliedert werden, womit ein Verbindungsaufbau von einem Verbindungsabbau unterschieden werden kann. Der Ablauf eines Verbindungsaufbaus soll dabei für jede Komponente identisch abgewickelt werden, siehe Abbildung 19. Um eine entsprechend stabile Reaktionszeit der teilnehmenden Roboter zu garantieren, sollen zum Start des Verbindungsaufbaus wiederholt Kommandos versendet werden. Dies soll eine erhöhte Central Processing Unit (CPU) Laufzeit erreichen, um eine Zeitverzögerung zur Laufzeit der Szenarios zu verhindern.



(a) Log in (b) Home

Abbildung 18: Mockup Connection

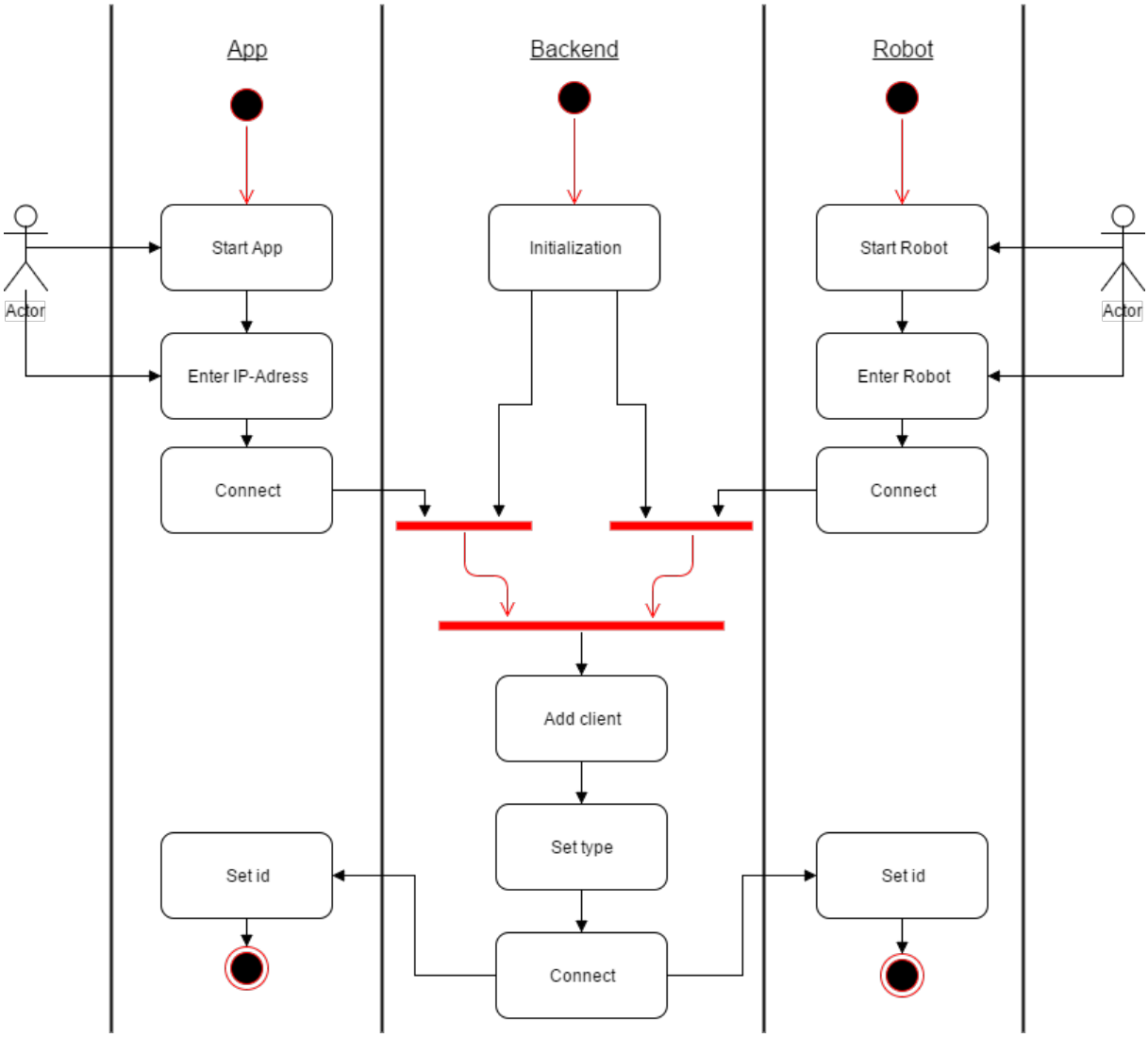


Abbildung 19: Use Case Connection

3.5.2 Synchronization

Der Use Case Synchronization stellt den Datenaustausch der beteiligten Komponenten dar, siehe Abbildung 21 und 22. Hierbei sollen verschiedene Typen unterschieden werden, wobei der komplette Datensatz in Form von allen Szenarien, Robotern, oder eines einzelnen Szenarios, sowie Roboters übertragen werden soll. Dies wird einerseits der Erstellung eines Szenarios dienen, als auch dessen Beobachtung durch den Spectator Modus. Die Übertragung eines einzelnen Roboter dient der Aktualisierung der jeweiligen Daten der Desktopanwendung, sowie der App, um diese laufend aktuell zu halten. Die synchronisierten Daten sollen des Weiteren auf den Komponenten über eine UI dargestellt werden können, um die Veränderung der aktuellen Daten zu verdeutlichen, sowie eine verbesserte Steuerung schaffen.

(a) Log in (b) Home

Abbildung 20: Mockup Synchronization

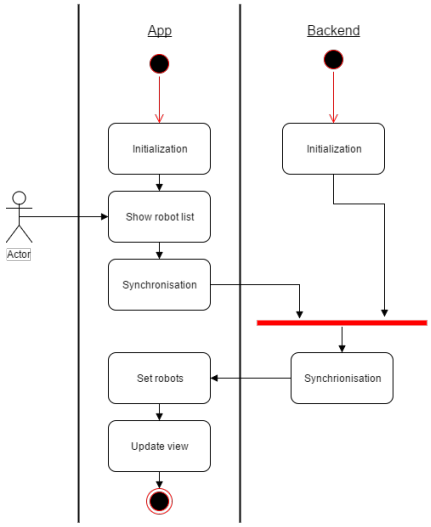


Abbildung 21: Use Case Synchronization

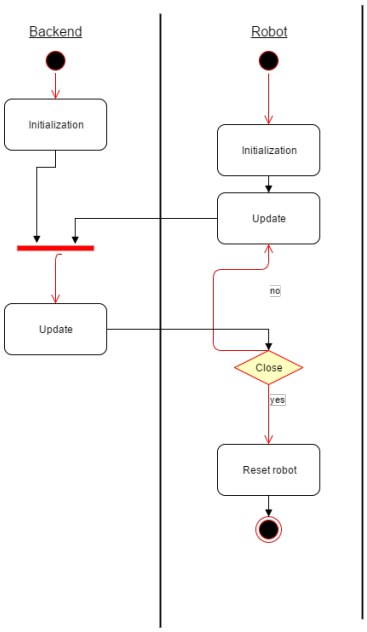


Abbildung 22: Use Case Update

3.5.3 Szenario

Der Use Case Szenario stellt den Ablauf der definierten Szenarien, zur Steuerung der Roboter dar, siehe Abbildung 24. Der Nutzer soll dabei zu Beginn das gewünschte Szenario, sowie die teilnehmenden Roboter festlegen. Anschließend wird das Szenario durch das Senden eines Kommandos gestartet, welches von der Desktopanwendung initialisiert wird. Dieses Kommando soll laufend wiederholt werden, um aktuelle Daten, wie Steuerungsinformationen an die Roboter weiterzuleiten. Zur Steuerung sollen hierbei eine direkte von einer positionsorientierten unterschieden werden, wobei die Roboter je nach Kommando die direkten Steuerungsinformationen oder eine anzufahrende Position erhalten.
Diese Implementierung soll den Vorteil einer Auslagerung der Programmlogik schaffen, indem die Ressourcen der Roboter geschont werden und die Logik zentral in der Desktopanwendung ausgeführt werden kann.

(a) Szenario (b) Spectator

Abbildung 23: Mockup Szenario

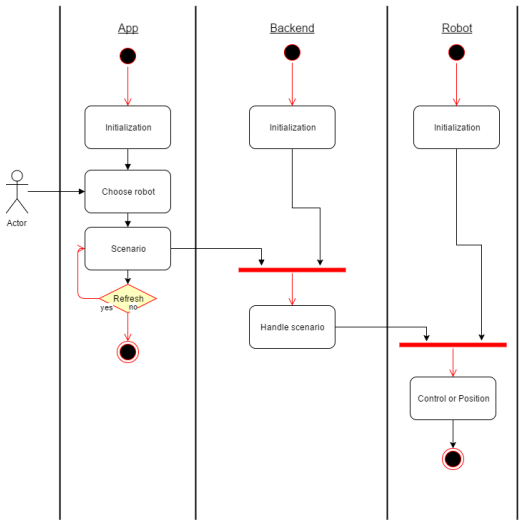


Abbildung 24: Use Case Szenario

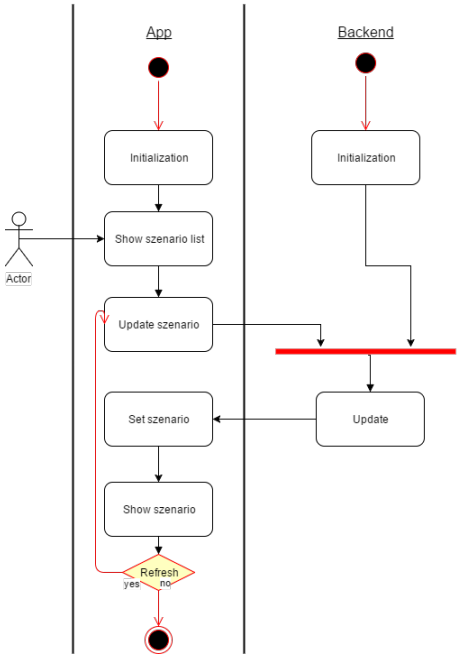


Abbildung 25: Use Case Spectator

3.5.4 Exception

Der Use Case Exception stellt den Ablauf einer auftretenden Exception in einem Roboter zum Kontext eines Verbindungsverlustes dar, siehe Abbildung 26. Dabei soll die auftretenden Nachricht, sowie die beteiligte Komponente der Desktopanwendung zugesendet werden. Dies soll ein kontrolliertes Schließen eines Szenarios ermöglichen, welches von einer solchen Exception betroffen ist. Die dabei teilnehmenden Komponenten, wie Nutzer und andere Roboter sollen entsprechend zurückgesetzt werden, damit diese für ein neues Szenario zur Verfügung stehen.

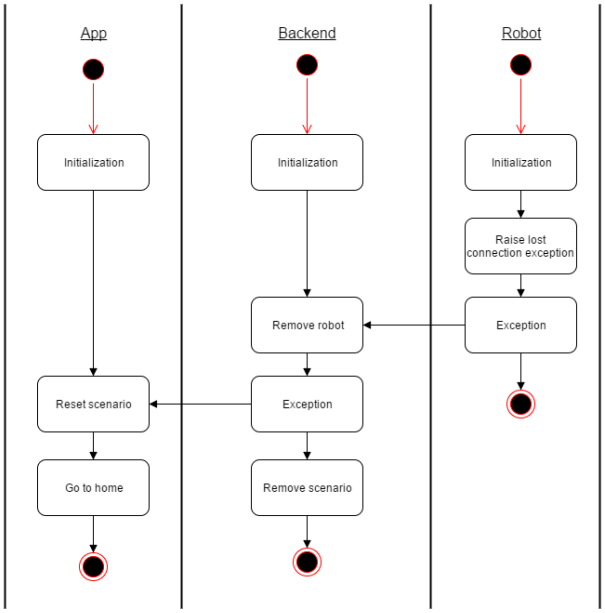


Abbildung 26: Use Case Exception

3.6 Kommunikation

Die Kommunikation des Schwarmverhalten basiert auf dem Standard für Netzwerkkommunikation, indem über eine TCP Schnittstelle Zeichenketten binär versendet werden. Dabei sollen die zu übertragenen Daten als Zeichenkette serialisiert und binär versendet werden, wobei die Gegenstelle die Daten über eine vorliegende Kommandostruktur deserialisiert. Um auf die Daten entsprechend zu reagieren werden diese nach Abhängigkeiten interpretiert. Dies soll über einen Typen realisiert werden, der der jeweiligen Struktur des Kommandos entspricht. Damit die Kommandostruktur als solche von den Komponenten von anderen unterschieden werden kann, sollen diese eine zusätzliche Grundstruktur zur Identifizierung der Schnittstelle, als auch den Kommandos erhalten.

Connection stellt das Kommando zur allgemeinen Verbindung dar, wobei eine Verbindung sowohl aufgebaut, als auch abgebaut werden kann. Dazu werden Parameter zur Identifizierung von Komponenten genutzt, die entsprechend auch in weiteren Kommandos ihre Anwendung finden. Die Tabelle 4 stellt den allgemeinen Aufbau eines solchen Kommandos dar.

Connection	Definition
String id	Stellt den Identifikationsstring dar
String type	Stellte den Typen dar
String apiid	Stellt die verwendete API dar
ClientIdentification identification	Stellt das Identifikationsobjekt der Komponente dar

Tabelle 4: JSON Kommando Connection

Synchronization stellt das Kommando zur Synchronization der erfassten Daten zwischen den verschiedenen Komponenten dar. Dabei werden die Daten als Listen mit entsprechenden Szenarien, wie Robotern übertragen. Je nach vorliegendem Typ des Kommandos, können verschiedene Daten übertragen werden, wobei Current einem einzelnen Objekten und die entsprechende Mehrzahl kompletten Datensätzen entspricht.

Synchronization	Definition
String id	Stellt den Identifikationsstring dar
String type	Stellte den Typen dar
String apiid	Stellt die verwendete API dar
ClientIdentification identification	Stellt das Identifikationsobjekt der Komponente dar
List scenarios	Stellt die aktuellen Szenarien dar, die übertragen werden
List robots	Stellt die aktuellen Roboter dar, die übertragen werden

Tabelle 5: JSON Kommando Synchronization

Scenario stellt das aktuell ablaufende Szenario dar, dass als Schwarmverhalten nachgeahmt wird. Darin enthalten sind die teilnehmenden Komponenten, sowie deren aktuell erfassten Daten. Zudem sind Steuerungsinformationen enthalten, die zur Orientierung des Schwarms entsprechend dem Kommando dienen. Diese verändern sich je nach Typen des Szenarios und können in verschiedenen Kontexten kommuniziert werden, um ein unterschiedliches Ergebnis zu erreichen.

Scenario	Definition
String id	Stellt den Identifikationsstring dar
String type	Stellte den Typen dar
String apiid	Stellt die verwendete API dar
ClientIdentification identification	Stellt das Identifikationsobjekt der Komponente dar
Scenario scenarios	Stellt das aktuell ablaufende Szenario dar

Tabelle 6: JSON Kommando Scenario

Exception stellt das Kommando zur Abhandlung eines Fehlverhaltens einer Komponente dar, die anderen Komponenten mitgeteilt werden kann. Sie enthält Informationen zur verursachenden Komponente, sowie des auftretenden Fehlers, der behandelt wird. Verwendung findet die Fehlerweiterleitung im Verbindungsverlust des Roboters, damit andere Teilnehmer über das Abmelden des entsprechenden Roboters in Kenntnis gesetzt werden und das Szenario beenden.

Exception	Definition
String id	Stellt den Identifikationsstring dar
String type	Stellte den Typen dar
String apiid	Stellt die verwendete API dar
ClientIdentification identification	Stellt das Identifikationsobjekt der Komponente dar
Exception exception	Stellt die aktuelle Execption dar, die im Roboter auftritt

Tabelle 7: JSON Kommando Exception

Control stellt das Kommando zur direkten Steuerung eines Roboters dar. Hierbei werden Steuerungsinformationen, wie Geschwindigkeit und Ausrichtung übertragen, um den Roboter entsprechend seiner aktuellen Position relativ zu steuern. Dies findet Anwendung in einer direkten Steuerung, als auch im Schwarmverhalten zur Steuerung des Nutzer abhängigen Roboters.

Control	Definition
String id	Stellt den Identifikationsstring dar
String type	Stellte den Typen dar
String apiid	Stellt die verwendete API dar
ClientIdentification identification	Stellt das Identifikationsobjekt der Komponente dar
Robot robot	Stellt den aktuellen Roboter dar
Steering steering	Stellt die aktuellen Steuerungsinformationen dar

Tabelle 8: JSON Kommando Control

Position stellt das Kommando zur indirekten Steuerung eines Roboters dar. Hierbei werden Positionsdaten übertragen, die der autonomen Navigation des Roboters dienen. Um eine entsprechende Geschwindigkeit zu erhalten, kann zudem ein Geschwindigkeitswert angefügt werden, der abhängig des Roboters gesetzt wird.

Position	Definition
String id	Stellt den Identifikationsstring dar
String type	Stellte den Typen dar
String apiid	Stellt die verwendete API dar
ClientIdentification identification	Stellt das Identifikationsobjekt der Komponente dar
Robot robot	Stellt den aktuellen Roboter dar
Position position	Stellt die anzufahrende Position dar
Speed speed	Stellt die einzustellende Geschwindigkeit dar

Tabelle 9: JSON Kommando Position

4 Implementierung

In diesem Kapitel wird die Implementierung des Projektes mit Fokussierung auf die einzelnen Komponenten beschrieben.

4.1 Kommunikation

Die Kommunikation der einzelnen Komponenten des Schwarmverhaltens baut auf einer klar definierten Struktur, um ein verteiltes System zu ermöglichen, siehe Abbildung 27. Die Daten werden dabei als JSON Objekte zur optimalen plattformübergreifenden Interpretation versendet, wobei jeweils die entsprechende Bibliothek zur Serialisierung verwendet wird.

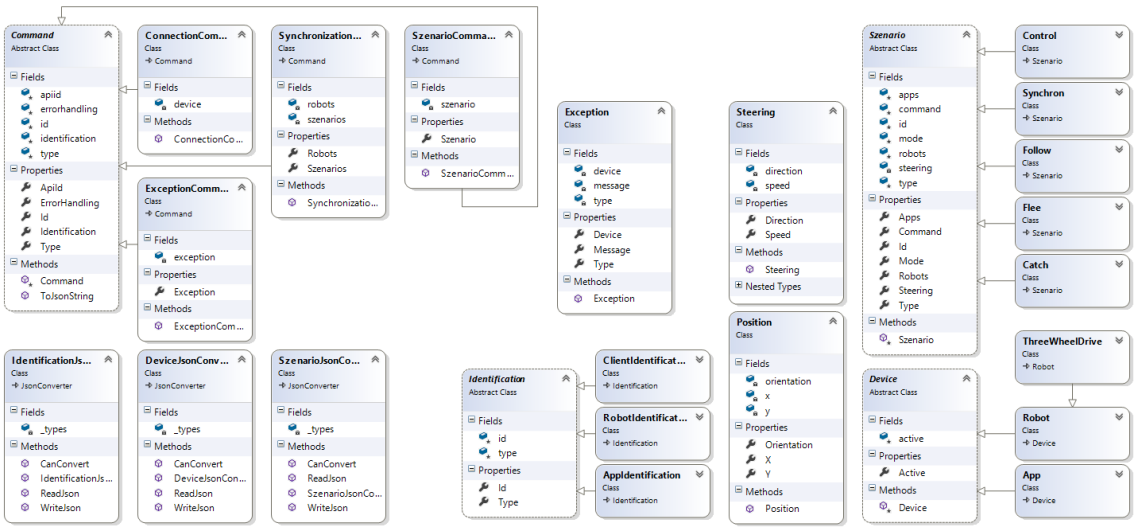


Abbildung 27: Aufbau Commands

Der Kern zur Implementierung der Kommunikation erfolgt in zwei Methoden, die auf jeder Komponente zur Verfügung stehen. Diese dienen zum Versenden, sowie Empfangen von Daten, wobei diese als Zeichenkette serialisiert und in Bytes aufgeteilt werden, siehe Abbildung 28a. Um den vollständigen Umfang der Daten zu erfassen, wird die Größe ermittelt und standardmäßig mittels vier Bytes übertragen. Dadurch ist eine maximale Paketgröße von 32 Byte möglich, was einer Länge von etwa 4 Milliarden Zeichen entspricht. Die Interpretation zum Empfangen erfolgt mit ähnlichem Muster, indem zunächst die Größe der Daten festgestellt wird und die Daten deserialiert werden, siehe Abbildung 28b.

```
12 references | ThunderSL94, 61 days ago | 1 author, 2 changes
public static async void SendCmd(string pCommand)
{
    if (!connected) throw new System.Exception("There is no connection to the server");
    checkCmd(pCommand);

    var command = Encoding.UTF8.GetBytes(pCommand);
    var size = new byte[4];
    var rest = pCommand.Length;

    //Calculate the length of the data
    for (var i = 0; i < size.Length; i++)
    {
        size[size.Length - (i + 1)] = (byte)(rest / Math.Pow(128, size.Length - (i + 1)));
        rest = (int)(rest % Math.Pow(128, size.Length - (i + 1)));
    }

    try
    {
        //Send the length of the data
        tcpSocketClient.WriteStream.Write(size, 0, size.Length);
        await tcpSocketClient.WriteStream.FlushAsync();

        //Send the full data
        tcpSocketClient.WriteStream.Write(command, 0, command.Length);
        await tcpSocketClient.WriteStream.FlushAsync();
    }
    catch (Exception e)
    {
        throw new Exception(304, "The json command could not send");
    }
}
```

(a) Versende Kommando

```
1 reference | Simon Lang, 18 days ago | 2 authors, 3 changes
private static string ReceiveCmd()
{
    var size = new byte[4];
    byte[] data = null;

    try
    {
        //Read 4 bytes
        tcpSocketClient.ReadStream.Read(size, 0, size.Length);

        //Calculate the full length
        var length = size.Select((t, i) => (int)(t * Math.Pow(128, i))).Sum();
        data = new byte[length];

        //Read the full data
        tcpSocketClient.ReadStream.Read(data, 0, data.Length);
    }
    catch (Exception e)
    {
        throw new Exception(303, "The json command could not receive");
    }

    //Get the string of the data
    var dataString = Encoding.UTF8.GetString(data, 0, data.Length);

    if (!checkCmd(dataString))
        return null;

    return dataString;
}
```

(b) Empfange Kommando

Abbildung 28: Kommunikation

Kommandos stellen die Basis der Kommunikationsstruktur sowie den aktuellen Kontext dar, indem sich die Software befindet, siehe Abbildung 29. Sie enthalten grundlegende Attribute zur allgemeinen Identifikation des Kommandos, die zur Interpretation verwendet, welche über definierte Enums ausgewählt werden. Je nach Kommando sind zusätzliche Objekte enthalten, die durch die jeweilige Id vordefiniert sind.

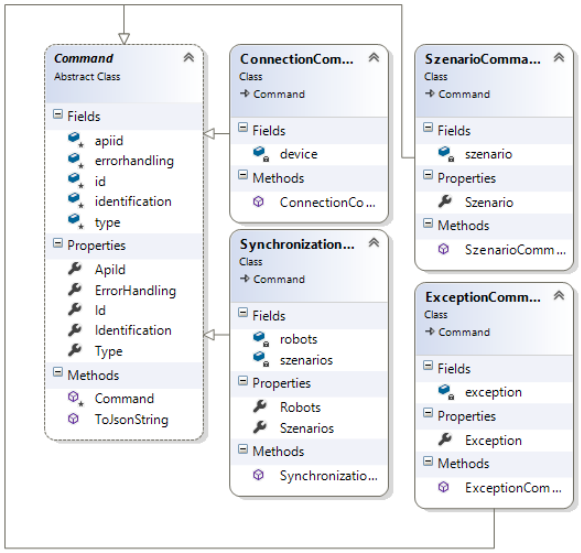


Abbildung 29: Kommandos

Identifikationen stellt die individuelle Identität der einzelnen Komponente dar, siehe Abbildung 30. Diese wird durch eine fortlaufende Identifikationsnummer, Typen und je nach Ableitung weiteren Attributen erreicht. Um die jeweiligen Kommandos entsprechend zuzuordnen, sind diese in jedem Kommando vorhanden und bilden die Basisobjekte. Die unterschiedlichen Typen sind dabei für verschiedene Kontexte der Software zuständig. Die ClientIdentification stellt einerseits die Verbindung einer allgemeinen Komponente zur Desktopanwendung dar, wogegen die Robot- bzw. AppIdentification die spezifische Identifikation der Komponente darstellt. Die Erstellung der Identifikation erfolgt wiederholt zur Anmeldung der Komponente am System. Zunächst wird ein leeres Objekt erzeugt, dass anschließend durch abfragende Kommandos an die entsprechende Komponente befüllt wird, welche hinterher eine berechnete Identifikationsnummer erhält.

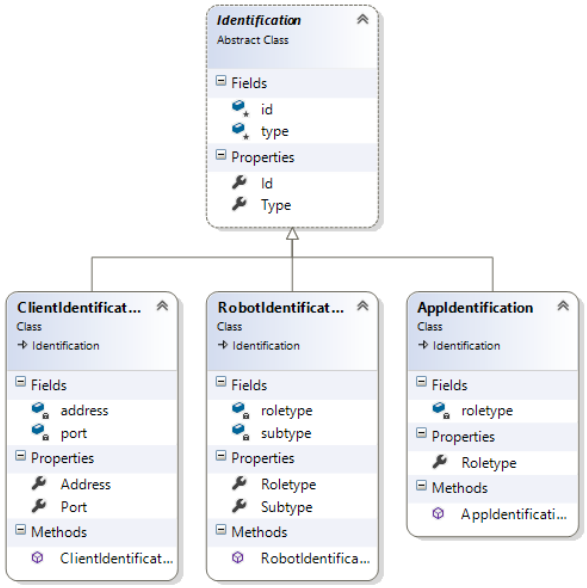


Abbildung 30: Identifikation

Geräte stellen die Komponenten dar, die an einem Szenario eines Schwarmverhaltens teilnehmen, siehe Abbildung 31. Sie enthalten jeweils spezifische Identifikations Objekte, zur gegenseitigen Zuordnung, sowie die erfassten Daten der entsprechenden Systeme. Die Unterscheidung erfolgt in zwei Komponenten, dem Robot und der App, wobei der Roboter in die jeweiligen Untertypen gegliedert werden kann.

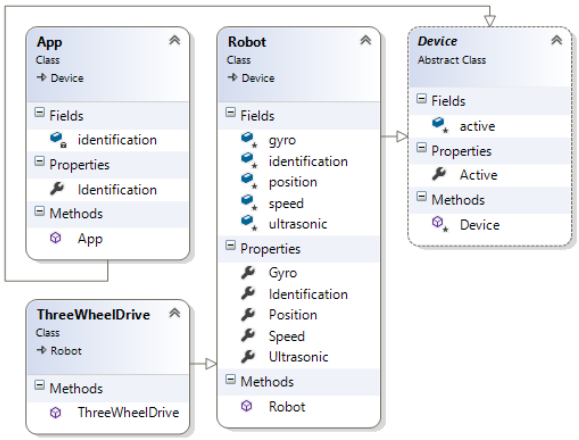


Abbildung 31: Devices

Szenarios stellen den Ablauf des Schwarmverhaltens dar, in dem sich der Nutzer befindet, siehe Abbildung 32. Sie enthalten die jeweiligen Teilnehmer des Szenarios, sowie die Steuerungsinformationen und damit die gesamten Daten des aktuellen Kontextes. Diese Objekte werden laufend aktualisiert und besitzen lediglich zur Laufzeit des Szenarios ihre Gültigkeit. Dabei existieren verschiedene Kategorien von Szenarien, siehe Abschnitt 3.4. Diese definieren jeweils einen unterschiedlichen Kontext und besitzen daher je nach Szenario zusätzliche Attribute.

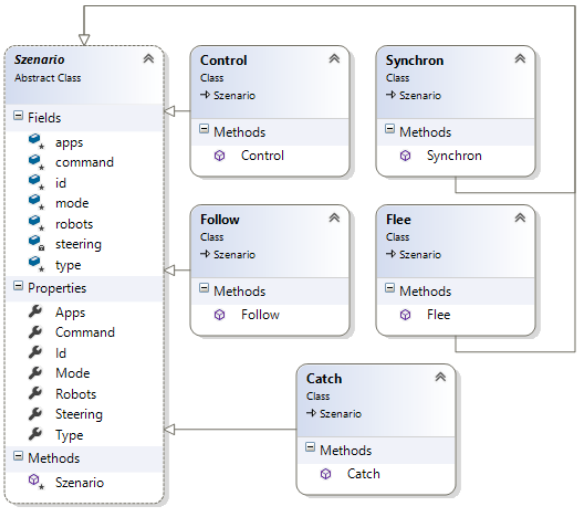


Abbildung 32: Scenarios

Konverter dienen der Deserialisierung von abstrakten JSON Objekten, welche nicht direkt identifiziert werden können, siehe Abbildung 33. Dazu gehören abstrakte Klassen, sowie Schnittstellen, welche keinem spezifischen Objekt zugeordnet werden kann. Die Implementierung erfolgt durch die Überschreibung der entsprechenden Methoden zur Deserialisierung und Serialisierung, siehe Abbildung 34a und 34b. Je nach Anwendung, wird ein Parameter übergeben, der das Objekt als Zeichenkette beinhaltet. Dieses wird durch eine Abfolge von Bedingungen auf den Typen geprüft wird, um das Objekt zu erstellen.

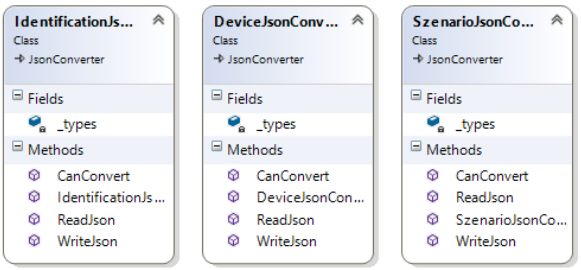


Abbildung 33: JsonConverter

```
2 references | ThunderSL94, 74 days ago | 1 author, 2 changes
public override object ReadJson(JsonReader reader, Type objectType, object existingValue, JsonSerializer serializer)
{
    switch (reader.TokenType)
    {
        case JsonToken.StartArray:      //Parse an array
            List<Device> devices = null;
            var jsonArray = JArray.Load(reader);

            foreach (var t in jsonArray)
            {
                if (jsonArray["identification"]["type"] == null) throw new System.Exception("Szenario is not implemented");
                switch (jsonArray["identification"]["type"].ToString()) //Check the identification
                {
                    case "App":
                        devices.Add(t.ToObject<App>());
                        break;
                    case "Robot":
                        devices.Add(t.ToObject<Robot>());
                        break;
                }
            }

            return devices;
        case JsonToken.StartObject:     //Parse an object
            Device device = null;
            var jsonObject = JObject.Load(reader);

            if (jsonObject["identification"]["type"] == null) throw new System.Exception("Devie is not implemented");
            switch (jsonObject["identification"]["type"].ToString()) //Check the identification
            {
                case "App":
                    device = jsonObject.ToObject<App>();
                    break;
                case "Robot":
                    device = jsonObject.ToObject<Robot>();
                    break;
            }
            return device;
    }
    throw new System.Exception("Not defined JsonToken");
}
```

(a) ReadJson

```
2 references | Simon Lang, 24 days ago | 2 authors, 3 changes
public override void WriteJson(JsonWriter writer, object value, JsonSerializer serializer)
{
    var t = JToken.FromObject(value);
    if(t.Type != JTokenType.Object)
        t.WriteTo(writer);
    else
    {
        var o = (JObject)t;
        o.WriteTo(writer);
    }
}
```

(b) WriteJson

Abbildung 34: Device JsonConverter

4.2 App

Die App erfolgt in einer plattformübergreifenden Implementierung durch Xamarin in C#. Kernelemente sind hierbei die Struktur, Oberfläche, Buisnesslogik und die Kommunikation. Dabei sind diese in zwei Projekten unterteilt. Die Kommunikation, mite den benötigten Strukturen zur Kommunikation bildet eine einbindbare Bibliothek (PCL), wie sie auch in den jeweiligen anderen Komponenten zu finden ist. Somit können die einzelnen ALgorithmen übertragen werden und müssen lediglich einmalig implementiert werden. Die eigentliche App besteht aus einer pcl, welches die jeweiligen plattformen, einbindet und daraus den plattformspezifischen Quellcode erzeugt. Der allgemeine Quellcode ist dabei im portable projekt und kann plattformübergreifend implementiert werden. Anpassungen können dabei in den plattformspezifischen Projekten unternommen werden.

4.2.1 Workflow

Die Struktur der App basiert auf dem in Xamarin weit verbreiteten ModelViewModelView Design Pattern, wobei die View durch Bindings aktualisiert werden kann.

Model View ViewModel (MVVM) ist ein Design Pattern, welches eine grundlegende Struktur in den Quellcode bringt. Dabei wird die erstellte Benutzeroberfläche von der Logik, sowie den Daten getrennt, um Änderungen unabhängig voneinander durchführen zu können. Um die Teile des Quellcodes zu verbinden wird auf Bindings gesetzt, die dafür zuständig sind, die jeweiligen Objekte gegenseitig zu aktualisieren.

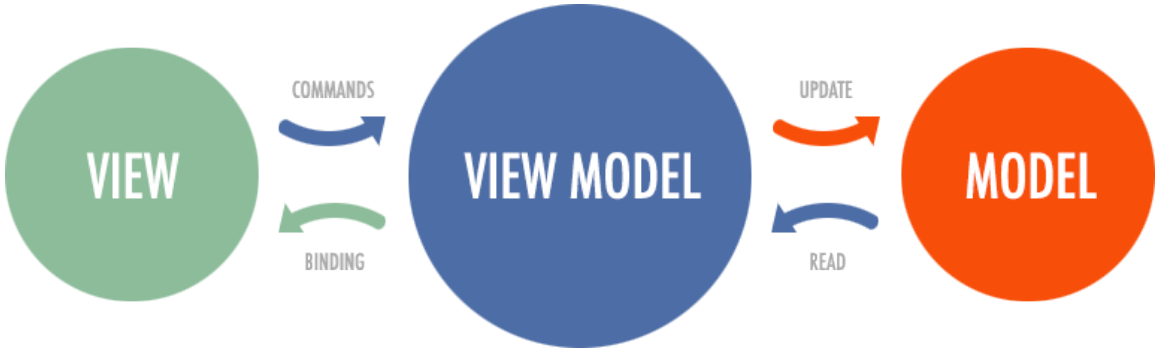


Abbildung 35: ModelViewModelView [3]

Bindings sind für die gegenseitige aktualisierung von Daten der jeweiligen Objekte zuständig. SIe entsprechen ihrem Sinn nach dem Observer Design Pattern und knüpfen an die VIewModels des MVVM an, wobei die dort definierten Objekte mit die der gleichnamigen View und den MOdel Objekten verbunden sind. Durch eine aktualisierung der Daten werden dabei die jeweilig verbundennen aktualisiert. DIes kann auf zwei verschiedenen wegen von statten gehen. Manuell, durch das aufrufen der entsprechenden Methode

im ViewModel, andererseits automatisch, durch Attribute, wobei bei einer Änderung, die Methode automatisch angestoßen wird.

4.2.2 Graphical User Interface (GUI)

Die Oberfläche der App wird mittels XAML erstellt, welche auf der Basis von XML ist. Dabei werden die einzelnen ELemente untereinnander Strukturuert, wobei der ENtwickler in Xamarin verschiedene Möglichkeiten des Designs besitzt. Grundlegend ist die entscheidung, ob eine plattformspezifisches Design sinnn macht, oder aber ein plattformübergreifendes, wobei dieses weniger Freiheiten bietet. Die plattformübergreifendes Design besitzt eine Reihe von grundlegeneden Layouts, die für Xamamrin eingesetzt werden können, sowie unterschiedliche Seitenarten. Views Seiten (ContentPage, MasterDetailPage, NavigationPage, TabbedPage, TemplatedPage, CaruselPage) View (ContentPresenter, ContentVIew, ScrollVIew, Frame, TemplatedVIew) Layout (Stacklayout, absolutLayout, RelativeLayout, GridLayout)

Weitere standardelemente

Durch verschiednene Packages könenn dabei viele Zusätzliche DInge erstellt werden

LogIn Page

Home Page

Option Page

List Page

Szenario Page

4.2.3 Buisnesslogic

4.2.4 Deployment

4.3 Backend

4.4 Robot

5 Evaluation

- bluetooth
- delay
- struktur (kommunikation, design pattern)
- ev3 (Verhalten,installation, konfiguration)
- vergleich ziele, ergebnis

6 Ausblick

Literatur

[1] Web & app development. URL http://www.nethority.com/web-app-development/.

[2] 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.

[3] Erazer Brecht. Mvvm entity framework | erazerbrecht's blog on wordpress.com. URL https://erazerbrecht.wordpress.com/2015/10/13/mvvm-entityframework/.

[4] Microsoft. Introduction to the c# language and the .net framework, . URL https://msdn.microsoft.com/de-de/library/z1zx9t92.aspx.

[5] Microsoft. Overview of the .net framework, . URL https://msdn.microsoft.com/en-us/library/zw4w595w(v=vs.110).aspx.

[6] Mono project. Mono project. URL https://github.com/mono.

[7] Mono project. About mono | mono, 17.03.2017. URL https://www.mono-project.com/docs/about-mono/.

[8] 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.

[9] Maximilian Schöbel, Thorsten Leimbach, and Beate Jost. *Roberta - EV3-Programmieren mit Java*. Roberta - Lernen mit Robotern. Fraunhofer-Verl., Stuttgart, 2015. ISBN 9783839608401.

[10] 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.

[11] Xamarin. Xamarin, . URL https://github.com/xamarin.

[12] Xamarin. Introduction to mobile development - xamarin, . URL https://developer.xamarin.com/guides/cross-platform/getting_started/introduction_to_mobile_development/.

[13] Xamarin. Introduction to portable class libraries - xamarin, . URL https://developer.xamarin.com/guides/cross-platform/application_fundamentals/pcl/introduction_to_portable_class_libraries/.

[14] Xamarin. Shared projects - xamarin, . URL https://developer.xamarin.com/guides/cross-platform/application_fundamentals/shared_projects/.

Anhang