UNIVERSITÄT DUISBURG-ESSEN
FAKULTÄT FÜR INGENIEURWISSENSCHAFTEN
ABTEILUNG INFORMATIK UND ANGEWANDTE KOGNITIONSWISSENSCHAFT

Bachelorarbeit

Entwicklung einer Augmented Reality App für iOS-Geräte zur Bedienung von IoT-Geräten

Malte Josten
Matrikelnummer: 3066184
Angewandte Informatik (Bachelor)

UNIVERSITÄT DUISBURG ESSEN

Fachgebiet Verteilte Systeme, Abteilung Informatik
Fakultät für Ingenieurwissenschaften
Universität Duisburg-Essen

22. Dezember 2020

Erstgutachter: Prof. Dr-Ing. Torben Weis
Zweitgutachter: Prof. Dr. Gregor Schiele
Zeitraum: 28. September 2020 - 27. Dezember 2020

Abstrakt

In dieser Arbeit wird eine App für iOS-Geräte entwickelt, die eine Benutzerschnittstelle für IoT-Geräte durch das Einbinden von Augmented Reality zur Verfügung stellt. Dabei baut die App nach Einscannen eines visuellen Markers eine Verbindung zum entsprechenden IoT-Gerät auf und stellt nach der Ermittlung der zur Verfügung stehenden Dienste dynamisch eine Benutzeroberfläche zur Verfügung. Jegliche Interaktion mit dem Interface führt zu einer Aktualisierung auf dem Zielgerät. Die Umsetzung hat gezeigt, dass Augmented Reality einen guten Ansatz darstellt, um eine intuitive und direkte Interaktion mit Internet Of Things-Geräten zu ermöglichen.

Inhaltsverzeichnis

1 Einleitung **1**

2 Technologiestand **3**

3 Ähnliche Anwendungen und Arbeiten **5**
3.1 Anwendungen 5
3.2 Arbeiten 6

4 Netzwerkkommunikation **9**
4.1 Service Discovery 10
4.1.1 mDNS-SD 13
4.2 TCP-Verbindung 14
4.3 Sicherheit 15
4.3.1 Bonjour 15
4.3.2 Multicast DNS(-SD) 16
4.3.3 Avahi 17
4.3.4 TCP-Pakete 17

5 App Implementierung **19**
5.1 Model-View-ViewModel 19
5.2 Anwendungszyklus 20
5.3 User Interface 23

6 IoT-Gerät Implementierung **25**
6.1 Starten des Programms 25
6.2 Paket-Empfang und -Validierung 27
6.3 Service Datei 28

7 Fazit **31**

8 Zukünftige Arbeiten **33**

Literatur **35**

Kapitel 1

Einleitung

Im Zuge des digitalen Wandels und der stetig anwachsenden Informationsflut muss eine Möglichkeit der Vermittlung und Darstellung von Daten gefunden werden. Der Bereich “Internet Of Things“ (IoT) ermöglicht durch seine andauernde Weiterentwicklung von Soft- und Hardware eine wachsende Vielfalt an Möglichkeiten der Datenerfassung und -verarbeitung. Das populäre Einsatzgebiet “Smart Homes“ nutzt IoT-Geräte, um Abläufe im Haushalt zu vereinfachen und Nutzer mit der Bedienung von Haushaltsgeräten zu unterstützen. Da viele IoT-Geräte kein natives User Interface vorweisen, sind Interaktionen ein komplizierter Bereich der Entwicklung und der Zugriff über APIs oder Web-Schnittstellen gestaltet sich umständlich. Das Darstellen von Informationen und die Interaktion mit digitalen Schaltflächen durch die Nutzung einer Augmented Reality (AR) Umgebung ist ein möglicher Ansatz, um ebendieses Problem zu lösen. Zudem wird die Geräteselektion durch das simple Abscannen eines Markers statt der Suche des entsprechenden Geräts (beispielsweise in einer Liste) intuitiver und schneller. Die Verwendung von AR-Technologien bietet sich an, da sie zum einen eine höhere Immersion und zum anderen eine stärkere Verbindung zwischen Aktion, Interface und Auswirkung zwischen Nutzer und (IoT-)Gerät erzeugt.

Im Kontext von verteilten Systemen - insbesondere Smart Home Technologien - wäre eine mit Augmented Reality als Schwerpunkt entwickelte App eine Neuheit. Daher wird im Zuge dieser Arbeit eine Augmented Reality App für iOS-Geräte entwickelt, mit welcher IoT-Geräte bedient werden können. Diese löst das Problem der o.g. Steuerung und bietet einen intuitiven Ansatz zur Smart Home Integration. Um die Funktionsweise der Anwendung zu gewährleisten, ist die Erstellung einer Testumgebung mit passender IoT-Hardware unverzichtbar. Während des Entwicklungsprozesses wird ein iPad Air 3. Generation und ein Raspberry Pi 4 Model B 2GB verwendet.

Nach Abschluss der Arbeit soll eine funktionsfähige mobile Applikation vorhanden sein, welche die Interaktion des Nutzers mit IoT-Geräten über ein AR-Interface ermöglicht. Dabei sollte sich das User Interface der Anwendung nach Einscannen eines Geräte-Markers durch eine Service Discovery-Suche innerhalb des Netzwerks entsprechend anpassen. Zudem sollte das IoT-Gerät über Nutzerinteraktionen mit dem User Interface informiert und passend modifiziert werden.

Kapitel 2

Technologiestand

Die in dieser Ausarbeitung verwendeten Technologien unterliegen im Zuge der Digitalisierung und dem daraus resultierenden Potenzial einem stetigen Wandel, welcher anhand der regen Arbeit an neuer sowie verbesserter Hard- und Software zu erkennen ist. Da sich die Möglichkeiten und Umstände je nach Stand der Technik ändern, ist es von Bedeutung, den zu diesem Zeitpunkt akutellen Technologiestand im Folgenden aufzugreifen und kurz zu erläutern.

Die mitunter populärsten Frameworks bzw. APIs wie WebXR (Mozilla), ARCore (Google) und ARKit (Apple) bieten die Möglichkeit des Einbindens von Augemented Reality Inhalten in ein Projekt.

WebXR ist eine von Mozilla für Webbrowser entwickelte API, wobei nur Chrome und Edge sowie Samsung Internet unterstützt werden. Außerdem ist es die "jüngste" API und besitzt weniger Features als die im Folgenden beschriebenen Frameworks [7].

ARCore unterstützt eine Vielzahl von Endgeräten mit einer Betriebsystemsversion von mindestens Android 8.1 bzw. iOS 11.0 (siehe [2] für die gesamte Auflistung), was die Zielgruppe einer Augmented Reality App deutlich vergrößert. ARCore bietet unter anderem die Funktionen zur Bewegungsverfolgung, zur Tiefenwahrnehmung sowie zum Umgebungsverständnis [11].

Zusätzlich zu den von ARCore bereitgestellten Features beinhaltet ARKit Funktionen zum Verfolgen und Erkennen von Objekten sowie von Personen [22].

Durch den stetig voranschreitenden Stand der Technik, insbesondere im Bereich Rechenkapazität und Kameraleistung mobiler Geräte, werden die Stabilität und Robustheit der Augmented Reality Applikationen verbessert und das Spektrum an Möglichkeiten erweitert. Hierzu zählen beispielsweise die von Apple verzeichneten Forschritte mit dem iPad Pro 2020 durch eine verbesserte Sensorik, die sogenannten "Pro Kameras" und LiDAR Scanner [24, 21].

SwiftUI ermöglicht Apple-Entwicklern das Erstellen und Designen von User Interfaces mittels kompaktem, deklaritivem Code sowie eines grafischen Editors. Zudem wird das Toolkit von allen Apple-Plattformen nativ unterstützt, sodass der Code nur in geringem Maße manuell angepasst werden muss und das UI wiederverwendet werden kann [20, 25]. Folglich ist SwiftUI ein geeignetes Mittel, um moderne und fortschrittliche Anwendungen für Apple Systeme zu entwickeln.

Auch im Bereich Internet Of Things findet eine andauernde und vor allem rasante Entwicklung der Soft- und Hardware statt, was dessen Möglichkeiten laufend verbessert und erweitert. Der Erfolg und das Potenzial von IoT-Geräten ist, besonders im Zusammenhang mit smarten Systemen, vielversprechend, weshalb voraussichtlich immer weitere Innovationen sowie verbesserte und neue Technologien entwickelt werden[34, 6].

Kapitel 3

Ähnliche Anwendungen und Arbeiten

Die Variation von SmartHome-Apps sowie die Anzahl der SmartHome-kompatiblen Geräte steigen durch die fortschreitende Digitalisierung stetig an. Ebenso liegt die Technologie der Augmented Reality im Trend, was zum einen durch die regelmäßige Verbesserung von Hardware ermöglicht und zum anderen durch die Anzahl der Arbeiten sowie Anwendungen bestätigt wird.

3.1 Anwendungen

Die auf allen iOS-Geräten vorinstallierte Anwendung "*Home*" basiert auf dem *HomeKit* Framework und ermöglicht das Einrichten und Steuern eines SmartHome-Systems. Dabei wird eine Vielzahl von IoT-Geräten unterstützt, wobei der Umfang bisher noch nicht an den von "*Google Home*" oder "*Amazon Alexa*" heranreicht [29]. Die App ist für Mobilgeräte (iPhone, iPad und iPod mit iOS 10.0+ sowie Apple Watch mit watchOS 2.2+) als auch für Computer mit macOS (mindestens Mojave) verfügbar. Zum einen bietet ein mit *Apple Home* gesteuertes SmartHome-System ein gewisses Sicherheitsniveau, indem nur von Apple verifizierte Geräte eingebunden werden können, andererseits wird dadurch jedoch die Möglichkeit, unbekannte und/oder neue Geräte mit einzubeziehen, erheblich erschwert. Hinsichtlich einer Augmented Reality-Einbindung bietet die App bisher keine Inhalte [23].

Auch die ebenfalls populäre SmartHome App *Google Home* beinhaltet bisher keine Augmented Reality-Inhalte. Sie ist allerdings im Gegensatz zu *Apple Home* nicht iOS-exklusiv, sondern wird auch für Android bereitgestellt, was ihre Verfügbarkeit deutlich erweitert. Die Anzahl an kompatiblen Geräten umfasst eine größere Auswahl als die von *Apple Home* [35, 46].

"*Devices – Control for HomeKit*" ist eine auf *Apple Home* aufbauende App von *Link-Desk*, welche ebenfalls SmartHome-Funktionalitäten mit der gleichen Auswahl an kompatiblen IoT-Geräten bereitstellt. Sie ist zum jetzigen Stand mitunter die einzige SmartHome-Anwendung, durch die der Nutzer in der Lage ist, seine Geräte in einer Augmented Reality-Umgebung zu konfigurieren und zu steuern. Allerdings ist die App nur für Apple Mobilgeräte (iPhone, iPad und iPod mit iOS 11.0+ sowie Apple Watch mit watchOS 4.0+) sowie Computer mit macOS (mindestens Mojave) verfügbar [32, 31].
Im Gegensatz zu dieser Arbeit werden die in der Augmented Reality-Umgebung festgelegten Geräte nicht per Bild- oder Objekterkennung, sondern mittels AR-Ankern erstellt und gespeichert. Das heißt, dass die App beim Abscannen der Umgebung anhand von erkannten Flächen und Kanten Punktewolken erzeugt, an denen der Nutzer Referenzpunkte für IoT-Geräte platzieren kann. Ist ein Geräteverweis erzeugt und entsprechend platziert worden, "schwebt" das Interaktionsfeld im Augmented Reality-Raum und ist auch nach einem Neustart der App an der gleichen Stelle vorhanden (sofern die Umgebung abermals abgescannt wurde). Da die Anwendung auf *Apple Home* basiert, ist die Kommunikation zwischen Mobil- und IoT-Gerät gleich der Basis App. Demnach findet die Netzwerkkommunikation, je nach IoT-Gerät, über WiFi oder Bluetooth statt [33].

3.2 Arbeiten

In dem Artikel *"ARIoT: scalable augmented reality framework for interacting with Internet of Things appliances everywhere"* [27] von Jo Dongisk, *et al.* wird das Augmented Reality Framework *ARIoT* vorgestellt, welches Nutzern durch serverseitige Bild- und Objekterkennung die Kommunikation mit in der Umgebung befindlichen IoT-Geräten ermöglicht. Das Zusammentragen der verfügbaren Dienste geschieht durch lokales Service Discovery per Bluetooth, wobei anschließend ein entsprechendes User Interface in einer Augmented Reality-Umgebung angezeigt wird [27].

Boris Pokric *et al.* stellt in der Arbeit *"Augmented Reality Enabled IoT Services for Environmental Monitoring Utilising Serious Gaming Concept"* [38] eine Möglichkeit vor, mit der umweltbezogene Informationen (insbesondere über die Luftqualität/-verschmutzung) mit Hilfe von IoT-Geräten als Mess- und Datenpunktstationen sowie Augmented Reality als Ausgabemedien dargestellt werden können. Hierbei speichern Backend-Server die Messdaten der IoT-Geräte und mobile sowie Web-Anwendungen rufen diese ab. Dadurch werden eine benutzerfreundliche Darstellung und die Interaktion mit dieser ermöglicht [38].

Um älteren oder behinderten Menschen alltägliche Interaktionen mit Haushaltsgegenständen in einem Smart Home zu erleichtern, hat Ahmed Mohmmad Ullah *et al.* das System *"Remote-touch: Augmented reality based marker tracking for smart home control"* [45] entwickelt. Die Interaktionen mit einer intuitiven Benutzeroberfläche am Smartphone werden nach dem Einscannen eines QR-Codes ermöglicht. Dabei werden die aufgenommenen Bilder, je nach Verfügbarkeit, lokal auf dem Smartphone oder auf einem Server verarbeitet. Bei diesem Ansatz dient ein Server als Knotenpunkt jeglicher Kommunikation, da dieser

1. ggf. den Prozess der Bildverarbeitung und QR-Code Detektion bereitstellt,
2. jegliche Informationen über die verfügbaren Geräte verwaltet sowie aktualisiert und
3. für die Synchronisation zwischen Nutzer- als auch IoT-Geräten zuständig ist [45].

Das User Interface einer SmartHome-Anwendung wurde bereits mit verschiedenen Ansätzen umgesetzt: von Geräten ähnlich einer Fernbedienung (*Universal Remote Console*s) [42], über herkömmliche User Interfaces auf Mobilgeräten (siehe *Apple Home*) bis hin zu 3D-Ansichten des gesamten Haushalts [4] oder der Visualisierung mit Hilfe von XR [32]. Da der Ansatz mit Augmented/Virtual Reality bisher noch nicht umfassend umgesetzt wurde, entspricht die Entwicklung einer SmartHome-Anwendung mit AR-Inhalten dem derzeitigen Technologiestand und bietet neue sowie intuitive Interaktionsmöglichkeiten.

Kapitel 4

Netzwerkkommunikation

Die Netzwerkkommunikation ist aufgrund der Gegebenheiten in zwei Abschnitte zu teilen:

1. Das Suchen eines passenden Geräts sowie seiner Dienste im lokalen Netzwerk nach Einscannen eines Markers durch zeroconf.

2. Das Übertragen von abgeänderten Werten durch eine Nutzerinteraktion vom Tablet zum Raspberry Pi nach erfolgreichem Verbindungsaufbau über TCP-Sockets.

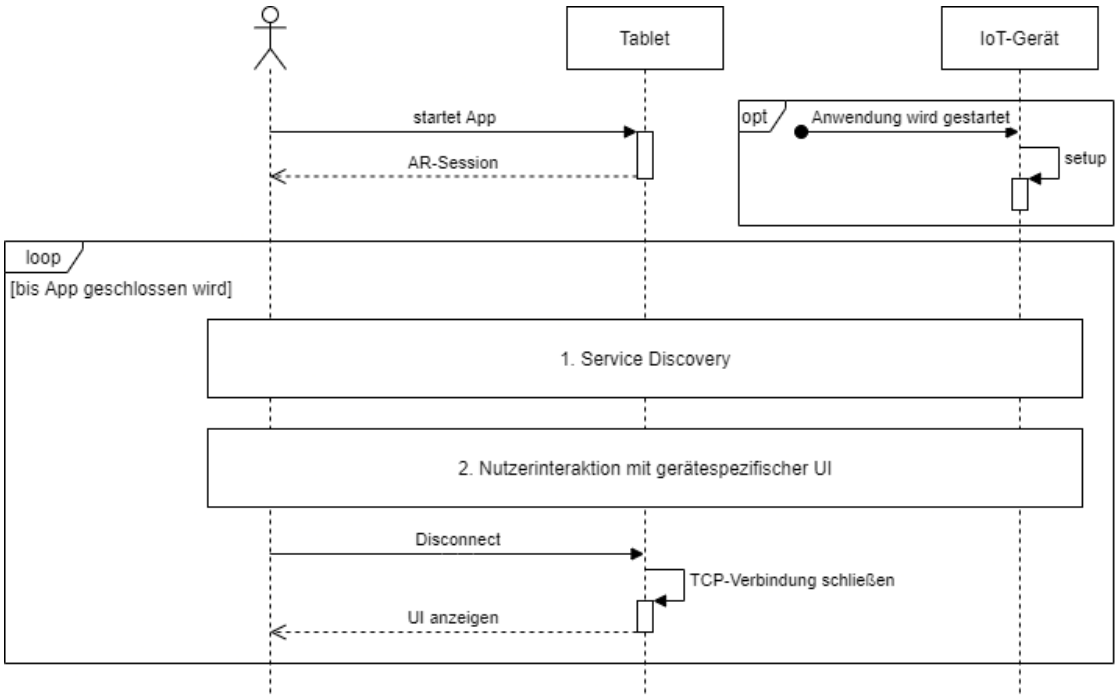

Abbildung 4.1: Prozess- und Netzwerkkommunikationsablauf

4.1 Service Discovery

Die bei der Tablet-seitigen Entwicklung eingesetzte Service Discovery API ist das von Apple entwickelte "Apple Bonjour"-Framework. Dieses implementiert eine Vielzahl von Protokollen, wobei DNS-SD für die Ausarbeitung essenziell ist [18]. Bonjour bietet zudem durch das automatische Zuweisen von IP-Adressen und Host Namen einen direkten Zugriff auf Services im lokalen Netzwerk. Außerdem ist Bonjour standardmäßig seit 2002 (damals unter dem Namen "Rendevouz") bei allen macOS und iOS-Geräten vorhanden - es benötigt also keine Installation [17].

Da die Nutzung von Bonjour für den Raspberry Pi aufgrund des Betriebssystems nicht möglich ist, wird das Bonjour-kompatible Framework "Avahi" genutzt, welches das Erstellen, Verwalten und Verbreiten von Services durch DNS-SD im lokalen Netzwerk ermöglicht. Außerdem wird mDNS unterstützt, wodurch sich Hosts im Netzwerk ohne einen zentralen DNS-Server über *<Gerätenamen>.local* erreichen lassen [41]. Die sogenannten Services sind durch XML-Dateien repräsentierte Dienste, welche nach Starten eines Avahi-daemon Prozesses im lokalen Netzwerk regelmäßig publiziert werden [44].

```
<?xml version="1.0" standalone='no'?><!--*-nxml-*-->
<!DOCTYPE service-group SYSTEM "avahi-service.dtd">
<service-group>
    <name>raspberrypi</name>
    <service>
        <type>_http._tcp</type>
        <port>36137</port>
        <txt-record>running=true</txt-record>
        <txt-record>toggle=false</txt-record>
        <txt-record>colorpicker=00387A</txt-record>
    </service>
</service-group>
```

Abbildung 4.2: Avahi Service Datei

Das Beispiel in Abbildung 4.2 wird mit dem *<service-group>*-Tag initiiert, welches das Dokumenten-Tag einer jeden Avahi Service Datei ist. Enthalten sein sollte jeweils ein *<name>*- sowie mindestens ein *<service>*-Tag. Innerhalb des *<name>*-Tags spezifiziert man den zu publizierenden Namen des Services (hier: *raspberrypi*). Das *<service>*-Tag beinhaltet die Service-Informationen, wie den Typen des Services (*<type>*, hier: *_http._tcp.*) oder des Ports (*<port>*, hier: *36137*). Außerdem können zusätzliche Subtypen (*<subtype>*) und Domain- sowie Host-Namen (*<domain-name>*, *<host-name>*) angegeben werden. *<txt-record>*-Tags enthalten DNS-SD TXT Record Data Einträge, die eine Struktur ähnlich einem Wörterbuch aufweisen (hier: *running=true*, *toggle=false* und *colorpicker=00387A*) [9].

Sofern der installierte Avahi-daemon korrekt konfiguriert ist, wird dieser beim Starten des Pis ausgeführt und publiziert bzw. registriert den erstellten Service mittels DNS-SD im lokalen Netzwerk [10].

Je nach Betriebssystem kann man sich mit sogenannten 'Bonjour Browsern' die verfügbaren Services im jeweiligen Netzwerk anzeigen lassen. Im Falle von macOS reicht ein einfacher Terminal Command, um eine gute Übersicht der jeweiligen Dienste zu erhalten. Folgend werden alle Services vom Typen *_http._tcp* gesucht und im Zone-File-Format angezeigt (siehe Abbildung 4.3) [8].

```
user@mac ~ % dns-sd -Z _http._tcp
Browsing for _http._tcp
DATE: ---Sat 24 Mar 2001---
12:34:56.789  ...STARTING...

; To direct clients to browse a different domain, substitute that domain in place of '@'
lb._dns-sd._udp                                 PTR     @

; In the list of services below, the SRV records will typically reference dot-local Multicast DNS names.
; When transferring this zone file data to your unicast DNS server, you'll need to replace those dot-local
; names with the correct fully-qualified (unicast) domain name of the target host offering the service.

_http._tcp                                      PTR     raspberrypi._http._tcp
raspberrypi._http._tcp                          SRV     0 0 36137 raspberrypi.local. ; Replace with unicast FQDN of target host
raspberrypi._http._tcp                          TXT     "running=true"  "toggle=false" "colorpicker=003B7A"
```

Abbildung 4.3: Terminalausgabe nach Service-Suchlauf

Ausgaben im Zone-File-Format geben Aufschluss über den *PTR*-, *SRV*- und *TXT*-Eintrag eines Dienstes. Aus dem *PTR*-Eintrag geht die Instanz des Dienstes hervor, welche dessen Namen sowie den Typen enthält. Der *SRV*-Eintrag gibt Auskunft über die Priorität, die Gewichtung des Servers, den Port und den Domänennamen des Zielhosts (von links), wobei die Priorität sowie die Gewichtung für die Ausführung dieser Arbeit irrelevant sind [1]. Durch den *TXT*-Eintrag erhält man, sofern welche vorhanden sind, die <txt-records> des Dienstes.
Im oben aufgeführten Beispiel ist demnach eine Instanz des Services *raspberrypi* des Typen *_http._tcp* gefunden worden, welcher über den Domänennamen *raspberrypi.local.* sowie den Port *36137* erreichbar ist. Zudem sind die <txt-records> zu *"running"* , *"toggle"* und *"colorpicker"* aufgelistet.

Der Ablauf des Service Discovery-Prozesses findet wie folgt statt:

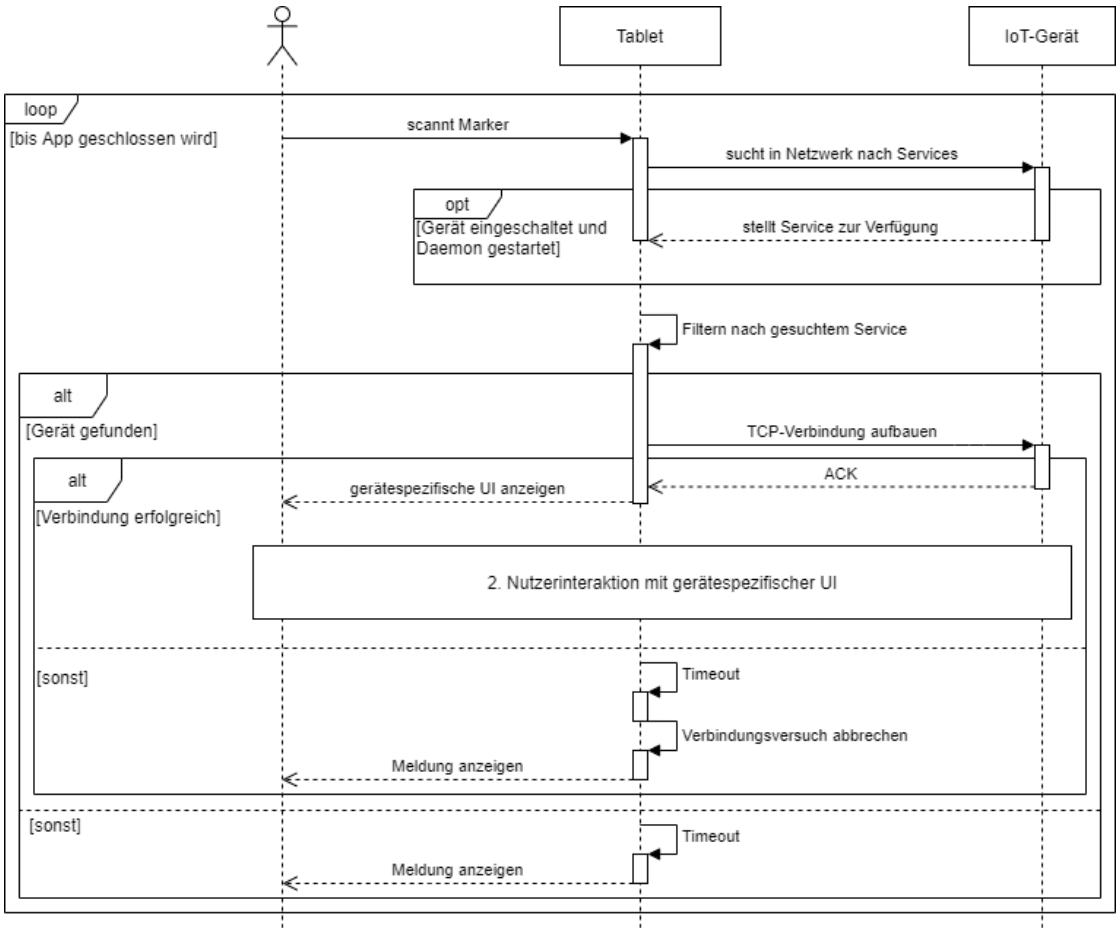

Abbildung 4.4: Ablauf des Service Discovery-Prozesses

4.1.1 mDNS-SD

Bei Multicast DNS (mDNS) handelt es sich um eine DNS-Technologie, welche sich von herkömmlichen Unicast DNS-Systemen insofern unterscheidet, als auch ohne einen zentralen DNS Server die Möglichkeit besteht, DNS-ähnliche Operationen auf lokaler Ebene durchzuführen. Ein Client des Netzwerks ist in der Lage seine DNS-Anfragen an *224.0.0.251:5353* zu senden, wodurch diese als mDNS-Anfragen gesehen und von den entsprechenden sich im Netzwerk befindlichen, mDNS-kompatiblen Hosts beantwortet werden. Dabei ist jedoch zwischen 'One-Shot mDNS'- und 'Continuous mDNS'-Anfragen zu unterscheiden. Bei 'One-Shot mDNS'-Queries gibt sich der Client mit dem erstbesten Ergebnis zufrieden, wohingegen bei 'Continuous mDNS' das Netzwerk solange durchsucht wird, bis der Nutzer den Suchvorgang manuell unterbricht oder keine weiteren Ergebnisse gebraucht werden [41]. Ein grundlegender Vorteil eines Systems, welches mDNS implementiert, besteht darin, dass nur geringe bis gar keine Administration und Konfiguration beim Aufsetzen notwendig ist [41].

DNS Service Discovery (DNS-SD) ist eine DNS-Spezifikation, die es den Nutzern eines Netzwerks erlaubt, eine Liste mit den Namen aller im Netzwerk verfügbaren Services abzurufen und diese mit DNS-Anfragen aufzulösen. Dies ist sowohl mit bereits bestehender DNS-Software als auch mit mDNS kompatibel. [5, 40].

Kombiniert man DNS-SD mit mDNS (mDNS-SD), werden, sofern kompatible Geräte vorhanden sind, deren Services in einer zeroconf-Umgebung publiziert. Die Nutzung von DNS-SD mit Unicast DNS benötigt hingegen einen gewissen Grad an Konfigurationen, wie beispielsweise das manuelle Konfigurieren der DNS-Domänen [39, 40].

Hinsichtlich dieser Arbeit ist mDNS-SD von großem Interesse, da kein zentraler DNS-Server benötigt wird und keine Konfiguration der einzelnen Geräte vorzunehmen ist. Des Weiteren ist eine beliebige Erweiterung des Netzwerks durch weitere Geräte nach dem “Plug’n’Play“-Prinzip einfach umsetzbar.

4.2 TCP-Verbindung

Sofern das passende Gerät über Bonjour gefunden und eine TCP-Verbindung hergestellt werden konnte, sind Nutzerinteraktionen mit dem gerätespezifischen User Interface möglich. Verändert eine solche Interaktion den Wert eines zur Verfügung stehenden UI-Elements, wird dessen Veränderung zuerst lokal gespeichert und das UI wird entsprechend angepasst sowie aktualisiert. Danach wird der Raspberry Pi per TCP-Verbindung über die Veränderung informiert. Dieser ändert daraufhin seine Service-Datei, sodass die neuen Werte im Netzwerk verfügbar sind.

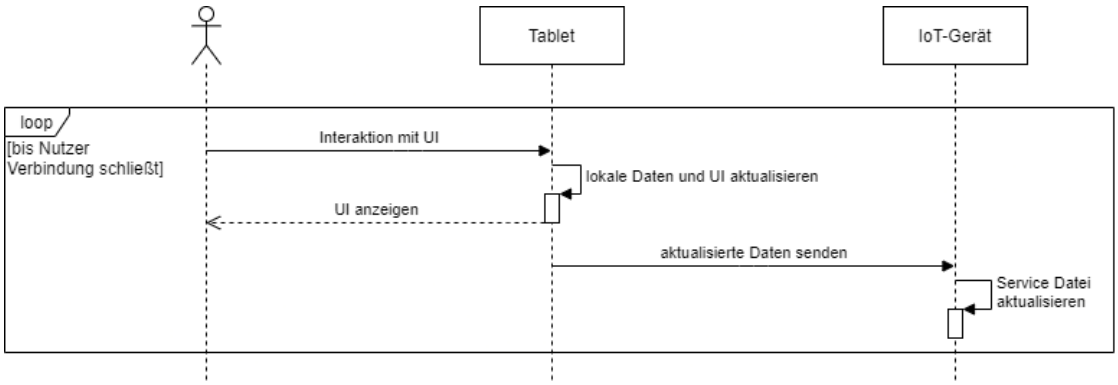

Abbildung 4.5: Netzwerkkommunikation nach Nutzerinteraktion mit UI

Für den Fall, dass während einer aktiven TCP-Verbindung der Raspberry Pi ausfällt oder das Programm geschlossen wird, empfängt das Tablet ein “Disconnection“-Paket, welches über die Verbindungsauflösung informiert.

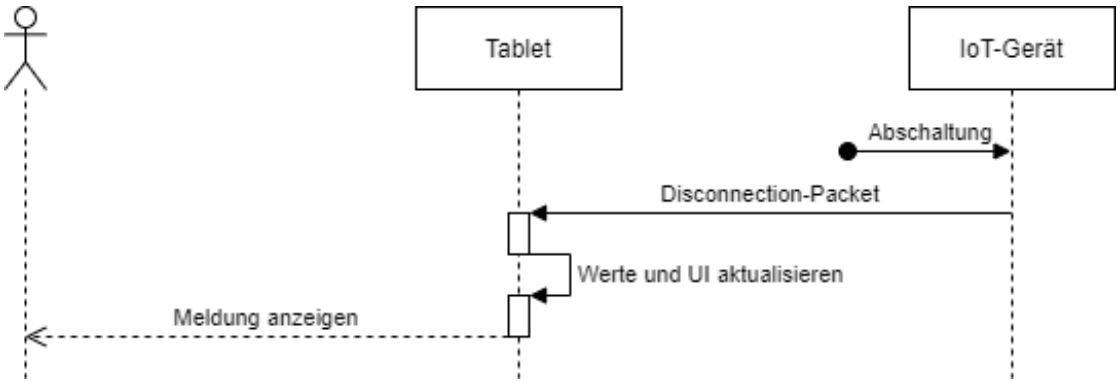

Abbildung 4.6: Netzwerkkommunikation bei einem vom IoT-Gerät eingeleiteten Verbindungsabbruch.

4.3 Sicherheit

Der Sicherheitsaspekt ist im Hinblick auf das kabellose Verbinden von User Interface mit IoT-Geräten insofern relevant, als dass je nach Verwendungszweck (insbesondere im Fall von SmartHome-Systemen) sensible Daten publiziert werden oder ein Eingriff in die Privätsphäre stattfinden könnte. Im Folgenden werden bekannte Schwachstellen von den in der Arbeit genutzten Frameworks und Protokollen, wie auch mögliche Lösungsansätze zum Vermeiden ebendieser aufgezeigt.

Es wurde im Laufe der Ausarbeitung der Versuch unternommen, eine PIN-Abfrage nach dem erstmaligen Verbinden von Tablet und IoT-Gerät während einer Sitzung einzurichten, was jedoch an der in Swift implementierten *NWConnection* scheiterte. Diese Implementierung sieht vor, dass die bestehende TCP-Verbindung nach dem Empfangen eines Pakets abgebrochen wird. Die daraufhin stattfindende Kommunikation zwischen den beiden Geräten wird dadurch stark erschwert. Durch den erheblichen Zeitaufwand und den ausbleibenden Erfolg wurde der Versuch der Implementierung eingestellt.

Da die iOS Anwendung keine und die IoT-Implementierung nur bedingt Sicherheiten aufweist, ist es ratsam, die Anwendung nur in dafür vorgesehenen und vertrauenswürdigen Umgebungen einzusetzen. Das Projekt könnte zudem bei Bedarf durch entsprechende Mechanismen oder zusätzliche Hardware im Zuge weiterer Arbeiten ergänzt werden.

4.3.1 Bonjour

Im Zuge der Ausarbeitung der Arbeit *Staying Secure and Unprepared: Understanding and Mitigating the Security Risks of Apple ZeroConf* [3] wurden mehrere Studien geführt, in dessen Folge sich herausstellte, dass viele zeroconf-nutzende Apps und Services oftmals große Sicherheitslücken aufweisen (dazu zählen beispielsweise Airdrop und Filedrop). Zu den schwerwiegendsten Problemen zählen die Möglichkeit von Man-In-The-Middle Attacken, das Spoofen von Nachrichten und die Imitierung/Nachahmung von legitimen Geräten im Netzwerk. Außerdem ist ein Zugriff auf sensitive Daten des Nutzers möglich [3]. Man-In-The-Middle Attacken sind nur in gewissem Maße durchführbar, da je nach Query-Art (siehe Kapitel 4.1.1 mDNS-SD) der zuerst gefundene (passende) Service ausgewählt wird. Dadurch ist der Erfolg eines solchen Angriffs zufällig.

Spoofing- und Nachahmungsattacken können durch eine Ergänzung im Programmcode verhindert werden, indem ein mehrfach aufkommender Service-Name beim Service Discovery Prozess als nicht legitim angesehen und infolgedessen ein leeres Suchergebnis bzw. eine passende Meldung angezeigt wird. Dies könnte ein Problem darstellen, wenn sich tatsächlich zwei gleiche Geräte (beispielsweise LED-Lampen) im Netzwerk befinden. In diesem Fall würde (unerwünschterweise) eine mögliche Man-In-The-Middle- oder Nachahmungsattacke identifiziert und folglich die gefundenen Geräte verworfen werden. Eine Möglichkeit, um das Problem zu umgehen und weiterhin Imitierungsattacken zu erschweren, können zuvor festgelegten Schlüssel, wie ein PIN-Code auf dem Marker verwendet werden [3].
Zudem empfiehlt Apple bei der Verwendung von Bonjour in größeren und/oder öffentlichen Netzwerken die Einbeziehung von Bonjour Gateway Produkten sowie bei einer kabellosen Kommunikation, die Verwendung eines Verschlüsselungsgrads von WPA2-PSK oder höher [19].

4.3.2 Multicast DNS(-SD)

In Systemen, welche Multicast DNS verwenden, wird die Kooperation aller im Netzwerk beteiligter Geräte vorausgesetzt, da keine zentrale Autorität vorhanden ist. Können jedoch nicht-kooperative Geräte nicht ausgeschlossen werden, ist die Verwendung von IPSec Signaturen und/oder DNSSEC Signaturen empfehlenswert [41]. Sollte sich auf einer im Netzwerk befindlichen Maschine Schadsoftware befinden, besteht je nach Betriebssystem die Möglichkeit, dass der für mDNS reservierte Port 5353 ohne administrative Rechte genutzt und ggf. missbraucht werden kann [41]. Außerdem ist jegliche Kommunikation aller Netzwerkteilnehmer in einem mDNS System abhörbar. Dies kann jedoch durch Verschlüsselungen oder Protokoll-Erweiterungen (siehe Kapitel 4.3.3 Avahi) verhindert werden [28]. Hierzu zählt zum Beispiel das asymmetrische Verschlüsselungsverfahren RSA. Eine Umsetzung bei primitiven Geräten (beispielsweise LED-Lampen) gestaltet sich schwierig, da nur eine sehr geringe bis keine Rechenleistung vorhanden ist.

Da bei DNS-SD lediglich der Umgang mit Namen und Einträgen in einem DNS-System definiert wird, gelten für DNS-SD nutzende Systeme dieselben Sicherheitsempfehlungen/-bestimmungen wie für herkömmliche DNS-Systeme [40]. Demnach sollten für DNS-Anfragen DNSSEC-Signaturen verwendet sowie das Aktualisieren von DNS-Einträgen nur durch zugelassene Nutzer ermöglicht werden [40].

4.3.3 Avahi

Aufgrund der Tatsache, dass Avahi mDNS-SD implementiert, sind die bereits angesprochenen Probleme zu Multicast DNS(-SD) (siehe Kaptiel 4.3.2 Multicast DNS(-SD)) sowie die entsprechenden Lösungen im gleichen Maße auf Avahi anzuwenden [44]. Zudem verhindert die von Daniel Kaiser vorgestellte Erweiterung für den Avahi Daemon die Bekanntgabe von privaten Daten über das Netzwerk, indem die resource-records verschlüsselt werden. Zugleich wird durch eine geringere Anzahl an versendeten Multicast-Paketen eine verminderte Netzwerkauslastung erreicht [28].

4.3.4 TCP-Pakete

Hat die iOS-App bereits eine TCP-Verbindung zum IoT-Gerät aufgebaut, sollte es optimalerweise bereits durch vorrangegangene Sicherheitsmaßnahmen als vertrauenswürdig eingestuft worden sein. Ist dies der Fall, sollten bewusste Manipulationsversuche über TCP-Pakete nicht vorkommen. Werden dennoch nicht-vorhergesehene Pakete empfangen, werden diese entsprechend behandelt und gegebenenfalls verworfen.

Das IoT-Gerät implementiert einen *DataValidator*, welcher eintreffende Pakete auf Fehlerfreiheit und Sicherheit prüft (siehe Kapitel 6 IoT-Gerät Implementierung).

Im Falle der Beendigung des Programms auf dem Raspberry Pi, empfängt das Tablet ein TCP-Paket und aktualisiert daraufhin den Verbindungsstatus sowie das UI (siehe Kapitel 5 AR-App Implementierung). Das Überprüfen des Nachrichteninhalts ist dabei nur begrenzt sicherheitsrelevant, da die vorhandene NWConnection aufgrund ihrer Implementierung nach jedem empfangenen Paket die Verbindung schließt.

Mögliche DOS-Attacken sind mit der derzeitigen Umsetzung nicht auszuschließen. Um die Auswirkungen auf die App nach einem Missbrauchsfall zu verringern, wird auf eine Inhaltsüberprüfung verzichtet, um auch im Angriffsfall den Verbindungsstatus sowie das UI entsprechend anzupassen.

Kapitel 5

App Implementierung

Im Zuge der Implementierung der iOS-Applikation wurde die Anwendung mit Swift 5 in XCode, mit Hilfe von ARKit und dem *SceneKit* Framework für die Augmented Reality Inhalte und *SwiftUI* zur Entwicklung der UI-Elemente verwendet.

Die Entscheidung, *SceneKit* zu nutzen, fiel aufgrund der Tatsache, dass das ebenfalls verbreitete Framework *RealityKit* für eher perfomance-lastige 3D Simulationen und Renderings in Augmented Reality Anwendungen gedacht ist [14].

SwiftUI bietet die Möglichkeit, durch ein deklaratives User Interface sogenannte *SwiftUI Views* zu generieren. Diese *Views* können in der App angezeigt werden. *Views* sind miteinander kompatibel, was eine Verschachtelung ermöglicht. Zudem ist das Erstellen und Bearbeiten von *SwiftUI Views* durch eine knappe und kompakte Codebase durchführbar [15, 16, 12].

5.1 Model-View-ViewModel

Das *Model-View-ViewModel*-Pattern (*MVVM*) ähnelt stark dem populären Ansatz des "*Model-View-Controller*"-Patterns (*MVC*). Sie unterscheiden sich allerdings in der Art der Verknüpfung von View und Model. Während im MVC-Pattern ein *Controller* die logik-enthaltene Brücke zwischen View und Model bildet, besitzt ein MVVM-System einen oder mehrere ViewModel(s), welche keine Kenntnis über den View-Inhalt haben und somit die Abkapselung von Design und Programmierung deutlich wird (siehe Abbildung 5.1 und Abbildung 5.2). Außerdem beinhaltet die *View* bei einem MVVM-Ansatz keine Logik. Diese wird ausschließlich vom ViewModel ausgeführt [37, 43, 13].

Abbildung 5.1: Model-View-ViewModel Aufbau

Abbildung 5.2: Model-View-Controller Aufbau

Eine Anwendung, welche das MVVM-Pattern implementiert, besteht aus Models, Views und ViewModels. Models repräsentieren Daten und enthalten keine Logik. Außerdem haben sie keine Kenntnis über bzw. keine Verbindung mit einer View. Diese wird durch ein ViewModel bereitgestellt, indem es die im Model enthaltenen Daten manipuliert und bei Änderungen derselben die View(s) benachrichtigt. Ebenso werden durch die Interkation mit User Interfaces entstandene Events von der View an das ViewModel weitergeleitet und von diesem verarbeitet. Views sind hingegen ausschließlich für die Darstellung von Daten und die Bereitstellung von Interaktionsmöglichkeiten verantwortlich, weswegen sie, bis auf den Code zum Registrieren von Nutzerinterkationen, keine Programmierlogik beinhalten [43, 30, 26].

Im Zuge meiner Arbeit habe ich mich für das Model-View-ViewModel-Pattern entschieden, da es gegenüber anderen bekannten Ansätzen (hier besonders MVC) eine eindeutige Trennung von User Interface, Code und Daten ermöglicht. Infolgedessen ist das (Unit-)Testen der Programmlogik deutlich einfacher durchzuführen, da keine Einbindung von UI-Elementen innerhalb des ViewModels vorhanden ist. Im Übrigen ist durch die Trennung die unabhängige Entwicklung von UI und Logik deutlich einfacher durchzuführen. Ebenso bewirkt diese strikte Trennung sowohl eine bessere Skalier- als auch Erweiterbarkeit des Projekts [43, 36].

5.2 Anwendungszyklus

Der dargestelle Zyklus (siehe Abbildung 5.3) wird so lange aufrecht erhalten, bis die iOS-Anwendung (manuell) geschlossen wird. Hierbei entsprechen alle Klassen, deren Name auf "*View*" endet, einer View des MVVM-Patterns. Der *DeviceHandler* sowie die Klasse *Bonjour* repräsentieren jeweils ein Viewmodel.

Es ist anzumerken, dass alle SwiftUI Views, welche UI-Elemente wie beispielsweise ein Farbauswahl-Tool beinhalten, von der Klasse *ElementView* erben und somit in den folgenden Diagrammen unter ebendiesem Namen zusammengefasst werden.

Nach dem Starten der App wird die Augmented Reality Umgebung (*ARView/-Handler*) angezeigt und bietet die Möglichkeit, vordefinierte Marker sowie Bilder einzuscannen. Infolge des Einscannens eines bekannten Markers wird der Service Discovery Prozess per Apple Bonjour in der Klasse *Bonjour* angestoßen und liefert bei Erfolg einen entsprechenden Service an den *DeviceHandler*. Dieser baut eine TCP-Verbindung zum dazugehörigen IoT-Gerät auf. Sobald ein Service sowie eine erfolgreich aufgebaute TCP-Verbindung vorliegen, werden in der *DeviceView* die verfügbaren UI-Elemente (*ElementView(s)*) nach entsprechendem Auslesen des Service Discovery Eintrags angezeigt.

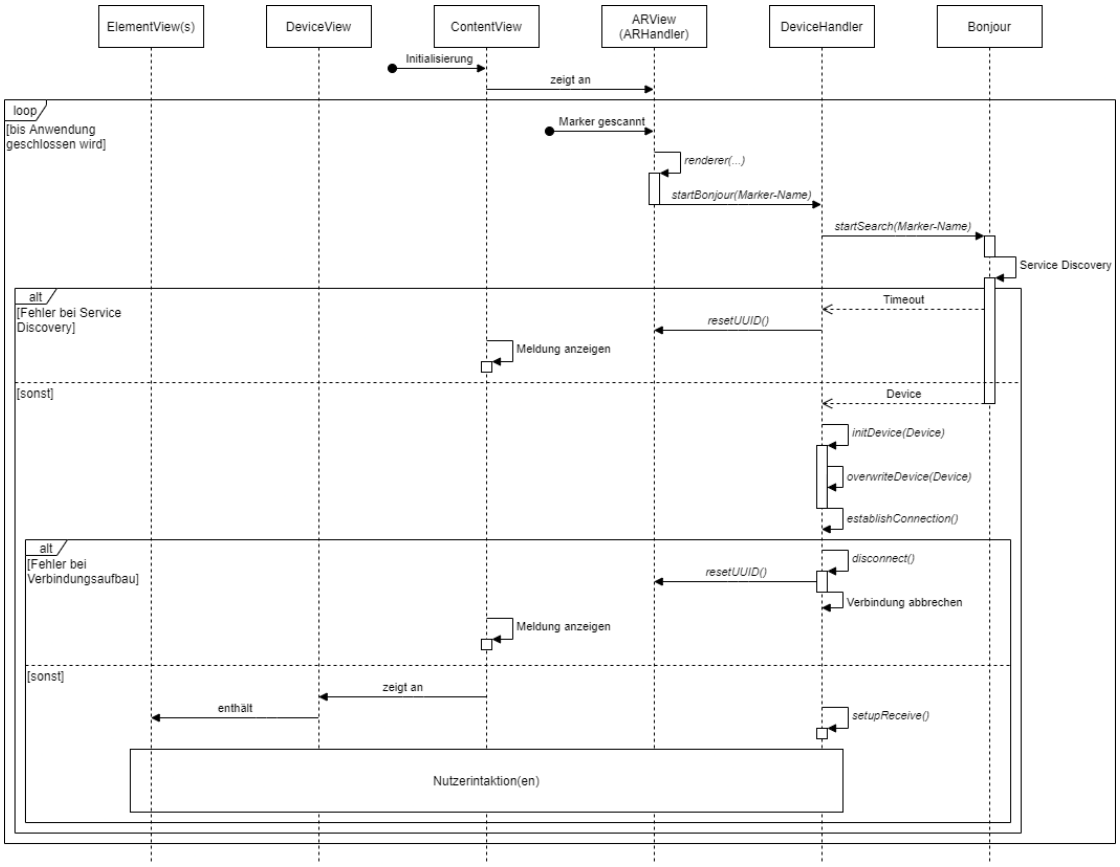

Abbildung 5.3: Anwendungszyklus der iOS-Anwendung

Anschließend kann der Nutzer mit den ihm zur Verfügung stehenden Elementen interagieren und Eigenschaften/Werte des IoT-Geräts manipulieren (siehe Abbildung 5.4). Verändert sich ein Wert eines lokalen UI-Elements, wird der *DeviceHandler* benachrichtigt und übermittelt dem entsprechenden IoT-Gerät die Veränderung. Bricht der Nutzer die Verbindung durch das Minimieren oder Schließen der Applikation sowie durch das Betätigen des Power-Knopfes ab, wird die bestehende TCP-Verbindung unterbrochen und die *DeviceView* ausgeblendet. Dies verhindert eine weitere Interaktion nach einem Verbindungsabbruch. Empfängt das Tablet zu einem beliebigen Zeitpunkt ein TCP-Paket vom IoT-Gerät, wird dieses als sogenanntes "Disconnection"-Paket gedeutet, welches denselben Code wie bei einem manuellen Verbindungsabbruch aufruft. Zudem wird dem Nutzer eine entsprechende Information im Debug-Bereich angezeigt.

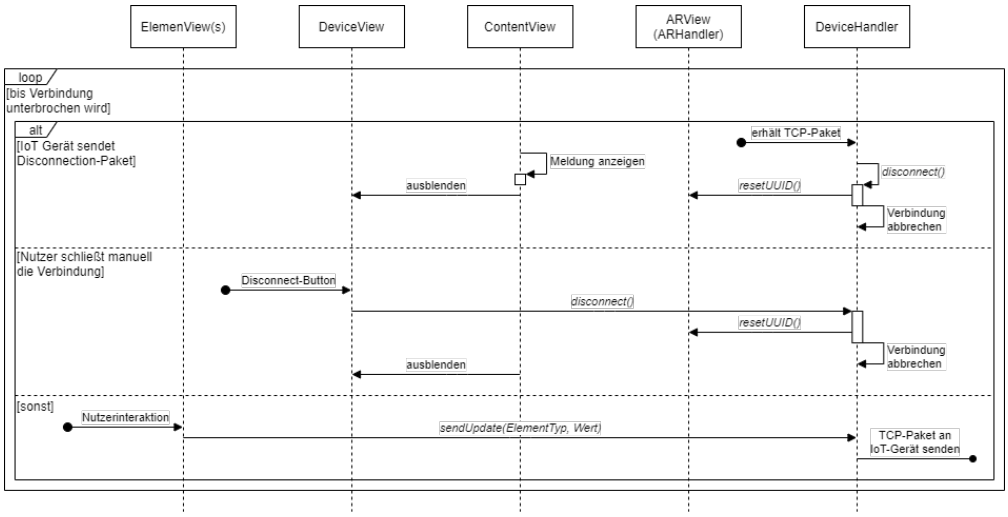

Abbildung 5.4: User Interface Interaktionen im Anwendungszyklus

Tritt im Laufe der Nutzung ein Fehler oder ein vom IoT-Gerät initiierter Verbindungsabbruch auf, zeigt der Debug-Bereich eine entsprechende Meldung. *resetUUID()*-Befehle sind notwendig, da die durch SceneKit implementierte Bilderkennung standardmäßig in jeder Sitzung jedes Bild nur genau einmal registriert. Damit nach Verbindungsabbrüchen ein mehrmaliges Einscanne des Markers möglich ist, werden die Bilderkennungs- bzw. Trackinginformationen in *ARHandler* zurückgesetzt.

5.3 User Interface

Nachdem die App auf dem Tablet gestartet wurde, nimmt die *ContentView* des gesamten Bildschirmbereich ein. Im oberen Teil des Bildschirms befindet sich ein Debug-Bereich, welcher Verbindungsinformationen und -meldungen anzeigt. Der restliche Bildschirm stellt die Augmented Reality Umgebung mit Hilfe der *ARView* dar (siehe Abbildung 5.5).

Nach erfolgreicher Verbindung von Tablet zum IoT-Gerät wird "auf" der *ARView* zusätzlich die *DeviceView* angezeigt, welche die View des IoT-Geräts repräsentiert (siehe Abbildung 5.6). Innerhalb dieser Ansicht befinden sich die verfügbaren ElementViews (hier: Checkbox, Farbauswahl-Tool und ein Button) sowie der "Disconnect"-Button, um die TCP-Verbindung zu unterbrechen. Zusätzlich werden im Debug-Bereich geräte-spezifische Informationen angezeigt.

Abbildung 5.5: Tablet-Bildschirm nach Starten der Anwendung

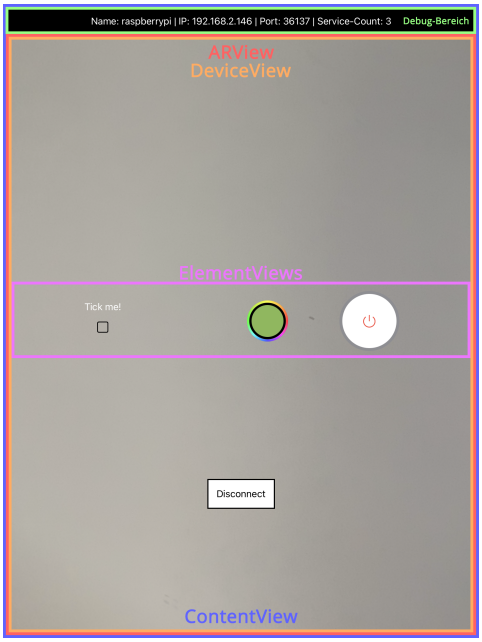

Abbildung 5.6: Tablet-Bildschirm nach Verbinden mit IoT-Gerät

Zusätzlich sollte erwähnt werden, dass das Anzeigen eines SwiftUI Textfelds und Nutzerinteraktionen mit diesem, aufgrund eines Memory Leaks, zu einem App Crash führen können. Ein Grund für den Fehler könnte ein Retain Cycle sein - im Zuge der Ausarbeitung konnte dieser allerdings nicht behoben werden.

Kapitel 6

IoT-Gerät Implementierung

Die Implementierung der IoT-Anwendung beinhaltet *Main.java*, welche zum Starten genutzt wird. Nachdem die Anwendung über die Kommandozeile initialisiert wurde, wird der Parameter-Input geprüft und es werden ggf. Konsolenausgaben bezüglich der Parameter ausgegeben. Zur Bestimmung der passenden Ausgabe wird *promptEnum.java* verwendet. Ist der Ausführungsaufruf fehlerhaft, wird die Anwendung gestoppt und es bedarf eines weiteren Konsolenaufrufs, um das Programm erneut zu starten. In der Datei *Main.java* wird außerdem eine Instanz von *ServiceProvider* erstellt, woraufhin die in den Parametern angegebenen *NetService*s initialisiert und dem *ServiceProvider* zugewiesen werden.

Für die Netzwerkkommunikation ist ein Objekt der *ServiceProvider*-Klasse verantwortlich und verwaltet sowie bearbeitet die Avahi Service Datei. Eine Instanz von *DataValidator.java* wird vom *ServiceProvicer* ausschließlich zur Überprüfung eines empfangenen TCP-Pakets verwendet.

6.1 Starten des Programms

Nachdem *Main.java* kompiliert wurde, kann man mit dem privilegierten Aufruf

```
sudo java Main <Service-Datei Pfad> <Port> [<Inputs>]
```

das Program auf dem Raspberry Pi starten. Da sich alle Avahi Service Dateien im Ordner `/etc/avahi/services` befinden und administrative Rechte zum Schreiben in ebendiesem Pfad benötigt werden, muss das Programm als `root` ausgeführt werden.

Der erste Parameter beinhaltet den absoluten Pfad zur Avahi Service Datei, welche den bereitzustellenden Dienst beinhaltet. Die Angabge einer '0' als Port führt zu einer zufälligen Port-Auswahl. Wird eine Port-Nummer $\neq 0$ angegeben, wird diese im Laufe der Initialisierung auf ihre Verfügbarkeit gerpüft. Sollte der Datei-Pfad nicht valide oder der Port nicht verfügbar sein, erhält der Nutzer eine entsprechende Rückmeldung.

Jede der folgenden Input-Varianten kann jeweils einmal pro Programm-Instanz genutzt werden. Dabei stehen die folgenden `input`-Optionen zur Auswahl:

`-b`, `--button`,
`-p`, `--colorpicker`,
`-t`, `--textfield`,
`-c`, `--checkbox`.

Ist es allerdings zu erwähnen, dass von der Verwendung des TextFields abgesehen werden sollte, da dessen Verwendung auf dem Tablet zu einem Absturz führen kann (siehe Kapitel 5.3 App Implementierung - User Interface).

Zuletzt erhält man durch das Anhängen von `--help` (oder durch `java Main --help`) eine detaillierte Beschreibung zum Ausführen der *.class*-Datei.
Enthält der Programmaufruf Fehler, wird eine entsprechende Konsolenausgabe ausgegeben und die Anwendung wird beendet.

Ein beispielhafter Programmaufruf kann wie in Abbildung 6.1 aussehen: Es wird der Pfad `/etc/avahi/services/iot.service` sowie ein zufälliger (verfügbarer) Port durch die Parameterangabe der `0` festgelegt. Der Dienst soll einen simplen Knopf (`TOGGLE`), ein Farbauswahl-Tool (`COLORPICKER`) und eine Checkbox (`CHECKBOX`) zur Verfügung stellen.

```
pi@raspberrypi $ sudo java dev.Main /etc/avahi/services/iot.service 0 -b -c -p
Using port 43691.
192.168.2.194 has connected.
[192.168.2.194]: Changed value of Service COLORPICKER to EC6C58 (old: FFFFFF).
[192.168.2.194]: Changed value of Service TOGGLE to true (old: false).
[192.168.2.194]: Changed value of Service TOGGLE to false (old: true).
[192.168.2.194]: Changed value of Service CHECKBOX to true (old: false).
[192.168.2.194]: Changed value of Service COLORPICKER to EB512E (old: EC6C58).
[192.168.2.194]: Changed value of Service COLORPICKER to 9CA435 (old: EB512E).
[192.168.2.194]: Changed value of Service COLORPICKER to 591D77 (old: 9CA435).
192.168.2.194 has disconnected.
```

Abbildung 6.1: Beispielhafter Programmaufruf inkl. Nutzerinteraktionen

6.2 Paket-Empfang und -Validierung

Sofern sich ein Gerät über eine TCP-Verbindung mit dem Raspberry Pi verbunden hat, wartet dieser im *ServiceProvider* auf Pakete. Wird ein solches Paket empfangen, erscheint eine Nachricht mit dem neuen sowie dem alten Wert in der Konsole (siehe Abbildung 6.1) und eine Instanz des *DataValidator*s überpüft die Nachricht auf fehlerhafte Angaben und mögliche Manipulationsversuche. Sofern bei einer der im folgenden beschriebenen Überprüfungen ein Fehler festgestellt wird, wird eine passende Nachricht auf der Konsole ausgegeben und der *ServiceProvider* erhält eine entsprechende Rückmeldung.

Im Zuge der Datenkontrolle wird die Nachricht zuerst auf das geforderte Format

`TYP`=`Wert`,

überprüft und das Vorhandensein des angegebenen *ServiceType*s `TYP`

`TYP` $\in$ {`verfügbare Typen`}

wird sichergestellt. Nach dem Feststellen eines validen *ServiceType*s wird der übermittelte `Wert` auf einen Injektionsversuch

`Wert` enthält '<' oder '>'

untersucht, da man mit der Zeichenfolge

```
TOGGLE=true</txt-record>
<service>
<type>_tls._tcp</type>
<port>43152</port>
</service>
```

einen weiteren Service einrichten und verbreiten lassen kann. Das Einfügen von weiteren <txt-records> wäre selbst ohne die Injektionskontrolle nicht möglich, da bei der Überprüfung auf das richtige Format nur Zeichenketten mit genau einem '=' angenommen werden.

Das Einfügen eines weiteren Eintrags mit dem Paket

```
TOGGLE=true</txt-record>
<txt-record>foo=bar</txt-record>
```

wäre somit nicht möglich, da nicht genau ein '=' vorhanden ist.
Des Weiteren wird die korrekte Formatierung bzw. der korrekte Inhalt des Werts mit Rücksicht auf den angegebenen *ServiceType*n geprüft. Hierbei gilt für die verschiedenen Typen:

ServiceType ***T***	**valider Wert** ***W***
TOGGLE	$W \in$ {“true“, “false“}
CHECKBOX	$W \in$ {“true“, “false“}
COLORPICKER	Länge von $W == 6 \wedge$
	W ist eine Hex-Zahl
TEXTFIELD	keine Einschränkungen

Wird die Überprüfung ohne eine Auffälligkeit durchlaufen, erhält der *ServiceProvider* eine entsprechende Rückmeldung und die gespeicherten *NetService*s sowie die Service-Datei werden aktualisiert.

6.3 Service Datei

Die im Projekt verwendete Service Datei entspricht dem in Kapitel 4.1 dargelegten Format. Zusätzlich zu den bereitgestellten Interaktions-Elementen wird ein weiterer <txt-record> mit dem Schlüssel `running` und einem Boolschen Wert gespeichert. Anhand dieses Schlüssels können andere Geräte, welche sich mit dem IoT-Gerät verbinden möchten, im Netzwerk erkennen, ob die Anwendung ausgeführt wird oder nicht. Beim Starten des Programms wird der Wert des Eintrags auf `true` und beim Beenden auf `false` gesetzt. Tritt während der Ausführung der Anwendung eine Exception auf, wird eine passende Konsolenausgabe erzeugt und das Programm wird nach dem Anpassen des `running` <txt-record> beendet. Wird nach einem Programmabbruch/-stop der `running` Eintrag nicht auf `false` gesetzt, ermöglicht die iOS-App Implementierung dennoch die Verbindung mit dem IoT-Gerät. Trotz der Tatsache, dass die benötigte Anwendung nicht läuft, würde dem Tablet-Nutzer das gerätespezifische UI angezeigt werden. Nutzerinteraktionen hätten folglich keine Auswirkungen auf das IoT-Gerät, da kein Programm läuft, welches die TCP-Pakete verarbeitet.

Bei Programmstart wird zunächst überprüft, ob eine Service Datei am angegebenen Pfad existiert bzw. ob der Nutzer einen validen Pfad übergeben hat. Existiert keine Service Datei, so wird eine neue *.service*-Datei erstellt und mit den passenden Einträgen befüllt sowie gespeichert. Ist die Datei bereits vorhanden, wird der Inhalt (teilweise) überschrieben, indem unter anderem der `port` <txt-record> angepasst wird. Außerdem wird der Inhalt der Datei auf das Vorhandensein von den beim Start angegebenen sowie alten Elementen überprüft. Alte bzw. nicht mehr genutzte Elemente werden durch das Löschen des entsprechenden <txt-record> entfernt. Ist ein Eintrag eines Elements vorhanden, wird keine Änderung vorgenommen, da der bereits bestehende Wert nicht zurückgesetzt werden soll. Zudem wird für jedes Element, welches durch den Nutzer angegeben wurde und nicht in der Service Datei verzeichnet ist, ein <txt-record> mit Standardwerten angelegt.

Um den Umgang mit den <txt-record> Einträgen beim Initialisieren des Programms zu veranschaulichen, werden im Folgenden Teile der Service Datei vor und nach dem Start aufgezeigt. Dabei wird davon ausgegangen, dass die in Abbildung 6.2 dargestellten Einträge, dem Inhalt vor der Ausführung des Programms entsprechen. Startet der Nutzer nun das Programm mit

```
sudo java Main /etc/avahi/services/iot.service 42681 -c -p,
```

werden der Eintrag des Ports sowie die <txt-record>s aktualisiert. Im vorliegenden Fall ist der toggle-Eintrag veraltet und wird entfernt. Zudem wurde eine Checkbox angegeben, sodass diese einen <txt-record> mit einem Standardwert erhält. Da der Colorpicker abermals verwendet werden soll, wird der Eintrag weder gelöscht, noch aktualisiert (siehe Abbildung 6.3).

```
<port>36167</port>
...
<txt-record>toggle=true</txt-record>
<txt-record>colorpicker=6CB732</txt-record>
```

Abbildung 6.2: Auszüge der Service Datei vor Start des Programms.

```
<port>42681</port>
...
<txt-record>checkbox=false</txt-record>
<txt-record>colorpicker=6CB732</txt-record>
```

Abbildung 6.3: Auszüge der Service Datei nach der Aktualisierung.

Während der Programmausführung können TCP-Pakete empfangen werden. Wurde das Paket bei Erhalt durch einen *DataValidator* freigegeben, wird der passende <txt-record> aus der Service Datei gefiltert und abgeändert. Somit wird garantiert, dass andere Geräte bei der Suche nach dem Dienst im Netzwerk die aktuellsten Werte erhalten.

Kapitel 7

Fazit

Die Bedienung von IoT-Geräten und Smart Homes über eine intuitive Nutzerschnittstelle ist ein wichtiger Aspekt im Hinblick auf ihren zukünftigen Erfolg. Aus diesem Grund wurde im Zuge dieser Arbeit eine Anwendung für iOS-Geräte entwickelt, die nicht nur durch das Erkennen und das Verbinden über Augmented Reality, sondern auch durch eindeutige User Interface-Elemente eine intuitive Interaktion mit IoT-Geräten ermöglicht.

Damit der Nutzer sich mit einem IoT-Gerät verbinden kann, wird ein dazugehöriger Marker innerhalb der Augmented Reality-Umgebung gescannt. Um den Anforderungen einer Anwendung in einem Netzwerk ohne DNS-Server, also einem mit zeroconf konfiguriertem Netzwerk, gerecht zu werden, wird mittels Apple Bonjour (speziell mit mDNS-SD) das passende Gerät ermittelt. Nach einem erfolgreichen Verbindungsaufbau kann nun der Anwender über ein minimalistisches Interface mit den vom IoT-Gerät bereitgestellten Diensten interagieren und diese auslesen sowie manipulieren.

Die direkte, visuelle Verbindung von Benutzeroberfläche und Gerät durch Augmented Reality weist ein großes Potenzial hinsichtlich Internet Of Things und Smart Home-Systemen auf. Jedoch verdeutlicht Kapitel 4, dass potenzielle Sicherheitsrisiken in zeroconf Netzwerken in umfangreichem Maße vorhanden sind und folglich die Ausarbeitung von Protokollen und Sicherheitsmaßnahmen erforderlich sind. Dann könnte die App auch in öffentlichen Netzwerken gefahrlos angewendet werden.

Aufgrund der vorliegenden Ergebnisse lässt sich sagen, dass ein Ansatz mit Augmented Reality im Gegensatz zu herkömmlichen Smart Home-Apps eine schnelle Verbindung und intuitive Bedienung von Geräten zulässt. So könnte der Mehrwert von Anwendungen wie “Apple Home“ oder “Google Home“ durch die Verwendung von AR-Technologien (ARKit) ansteigen.

Kapitel 8

Zukünftige Arbeiten

Wie aus dem Kapitel 4.3 “Netzwerkkommunikation - Sicherheit“ hervorgeht, wurden im Zuge der Implementierung keine Sicherheitsvorkehrungen hinsichtlich des Service Discovery Prozesses und der Netzwerkkommunikation vorgenommen. Daher wäre die Ausarbeitung von Protokollen und/oder Algorithmen zur Gewährleistung der Integrität sowie Authentizität innerhalb des Netzwerks eine Möglichkeit, die Anwendung für größere und ggf. öffentliche Bereiche nutzbar zu machen. Dabei könnte beispielsweise die Verwendung eines PINs oder einer ähnlichen Authentifizierungsmethode erneut aufgegriffen werden.

Zudem weist die derzeitige Umsetzung des User Interfaces gewisse Lücken bezüglich der Nutzerfreundlichkeit auf. Um diese zu schließen, empfiehlt es sich, eine einheitliche und übersichtliche Benutzeroberfläche zu erstellen, welche der Anwendung einen gewissen Grad an Professionalität verleiht. Hierbei kann man sich durchaus an bereits bestehenden Apps/Anwendungen orientieren und ggf. für den Aspekt der Augmented Reality individuelle User Interface Erweiterungen umsetzen.

Außerdem würde der Prozess der IoT-Geräte-Ermittlung durch Objekterkennung im Gegensatz zum Einscannen eines Markers deutlich simpler ausfallen, da statt des Einscannens lediglich das betreffende Objekt abgefilmt wird. ARKit bietet nativ die Möglichkeit, Objekte in der realen Welt zu erkennen, indem durch das Gerät erkannte Punkte, Flächen und Ecken mit einem zuvor festgelegten 3D-Modell abgeglichen werden. Ein Objekt gilt als erkannt, wenn eine vorgegebene Menge der genannten Referenzen mit denen des 3D-Modells des Objekts übereinstimmt. Bei einem markerlosen Ansatz sinkt zusätzlich der Anteil an physischen und relevanten Objekten durch das Wegfallen der Marker und die Immersion steigt bzw. die Bedienung der App wird intuitiver.

Literatur

[1] L. Esibov A. Gulbrandsen P. Vixie. *RFC 2782 - A DNS RR for specifying the location of services (DNS SRV)*. [Online; abgerufen am 27.10.2020]. 2000. URL: https://tools.ietf.org/html/rfc2782.

[2] *ARCore supported devices — Google Developers*. [Online; abgerufen am 20.11.2020]. 2020. URL: https://developers.google.com/ar/discover/supported-devices.

[3] X. Bai u. a. „Staying Secure and Unprepared: Understanding and Mitigating the Security Risks of Apple ZeroConf“. In: *2016 IEEE Symposium on Security and Privacy (SP)*. 2016, S. 655–674. DOI: 10.1109/SP.2016.45.

[4] L. Borodulkin, H. Ruser und H. -. Trankler. „3D virtual ßmart homeüser interface“. In: *2002 IEEE International Symposium on Virtual and Intelligent Measurement Systems (IEEE Cat. No.02EX545)*. 2002, S. 111–115. DOI: 10.1109/VIMS.2002.1009367.

[5] Stuart Cheshire. *DNS Service Discovery (DNS-SD)*. [Online; abgerufen am 28.10.2020]. URL: http://www.dns-sd.org/.

[6] Louis Columbus. *10 Charts That Will Challenge Your Perspective Of IoT's Growth*. [Online; abgerufen am 25.11.2020]. 2018. URL: https://www.forbes.com/sites/louiscolumbus/2018/06/06/10-charts-that-will-challenge-your-perspective-of-iots-growth/.

[7] MDN contributors. *WebXR Device API*. [Online; abgerufen am 20.11.2020]. 2020. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebXR_Device_API.

[8] Darwin. *dns-sd(1) - NetBSD Manual Pages*. [Online; abgerufen am 27.10.2020]. 2020. URL: https://man.netbsd.org/dns-sd.1.

[9] The Avahi Developers. *avahi-daemon(5) - Linux man page*. [Online; abgerufen am 11.11.2020]. 2019. URL: https://linux.die.net/man/5/avahi.service.

[10] The Avahi Developers. *avahi-daemon(8) - Linux man page*. [Online; abgerufen am 24.10.2020]. 2019. URL: https://linux.die.net/man/8/avahi-daemon.

[11] *Fundamental concepts — ARCore — Google Developers*. [Online; abgerufen am 20.11.2020]. 2020. URL: https://developers.google.com/ar/discover/concepts.

[12] Paul Hudson. *SwiftUI lets us build declarative user interfaces in Swift*. [Online; abgerufen am 10.11.2020]. 2019. URL: https://www.hackingwithswift.com/articles/191/swiftui-lets-us-build-declarative-user-interfaces-in-swift.

[13] Paul Hudson. *What is MVVM?* [Online; abgerufen am 04.11.2020]. 2019. URL: https://www.hackingwithswift.com/example-code/language/what-is-mvvm.

[14] Applce Inc. *Reality Kit — Apple Developer Documentation*. [Online; abgerufen am 06.11.2020]. 2020. URL: https://developer.apple.com/documentation/realitykit.

[15] Applce Inc. *SwiftUI — Apple Developer Documentation*. [Online; abgerufen am 06.11.2020]. 2020. URL: https://developer.apple.com/documentation/swiftui.

[16] Applce Inc. *SwiftUI Tutorials — Apple Developer Documentation*. [Online; abgerufen am 06.11.2020]. 2020. URL: https://developer.apple.com/tutorials/swiftui.

[17] Apple Inc. *Bonjour Network Discovery and Connectivity - WWDC 2011 - Videos - Apple Developer*. [Online; abgerufen am 24.10.2020]. 2011. URL: https://developer.apple.com/videos/play/wwdc2011/211/.

[18] Apple Inc. *Apple Developer Documentation Archive, About Bonjour*. [Online; abgerufen am 12.10.2020]. 2013. URL: https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/NetServices/Introduction.html.

[19] Apple Inc. *Security Implications of Bonjour protocol for developers and administrators*. [Online; abgerufen am 26.10.2020]. 2016. URL: https://support.apple.com/en-us/HT205195.

[20] Apple Inc. *SwiftUI - News - Apple Developer*. [Online; abgerufen am 25.11.2020]. 2019. URL: https://developer.apple.com/news/?id=06032019b.

[21] Apple Inc. *Apple stellt neues iPad Pro mit fortschrittlichem LiDAR Scanner vor und bringt Trackpad-Unterstützung für iPadOS*. [Online; abgerufen am 27.11.2020]. 2020. URL: https://www.apple.com/de/newsroom/2020/03/apple-unveils-new-ipad-pro-with-lidar-scanner-and-trackpad-support-in-ipados/.

[22] Apple Inc. *ARKit — Apple Developer Documentation*. [Online; abgerufen am 20.11.2020]. 2020. URL: https://developer.apple.com/documentation/arkit/.

[23] Apple Inc. *iOS - Home - Apple (DE)*. [Online; abgerufen am 23.11.2020]. 2020. URL: https://www.apple.com/de/ios/home/.

[24] Apple Inc. *iPad Pro*. [Online; abgerufen am 20.11.2020]. 2020. URL: https://www.apple.com/de/ipad-pro/.

[25] Apple Inc. *Xcode - SwiftUI - Apple Developer*. [Online; abgerufen am 25.11.2020]. 2020. URL: https://developer.apple.com/xcode/swiftui/.

[26] Bart Jacobs. *Swift and MVVM in Practice*. [Online; abgerufen am 04.11.2020]. URL: https://cocoacasts.com/swift-and-model-view-viewmodel-in-practice/.

[27] D. Jo und G. J. Kim. „ARIoT: scalable augmented reality framework for interacting with Internet of Things appliances everywhere“. In: *IEEE Transactions on Consumer Electronics* 62.3 (2016), S. 334–340. DOI: 10.1109/TCE.2016.7613201.

[28] Daniel Kaiser u. a. „A Multicast-Avoiding Privacy Extension for the Avahi Zeroconf Daemon“. In: *NetSys 2015, demonstration*. 2015. URL: https://www.netsys2015.com/program/demonstrations/.

[29] Kate Kozuch. *Apple HomeKit: What is it, and how do you use it?* [Online; abgerufen am 26.11.2020]. 2020. URL: https://www.tomsguide.com/us/apple-homekit-faq,review-4195.html.

[30] Andreas Kühnel. *Visual CSharp 2012*. Bd. 6. [Online; abgerufen am 04.11.2020]. Rheinwerk Computing, 2013, Kapitel 28.5 Das MVVM–Pattern. ISBN: ISBN 978-3-8362-1997-6. URL: http://openbook.rheinwerk-verlag.de/visual_csharp_2012/1997_28_005.html.

[31] LinkDesk. *Devices - Control for HomeKit im AppStore*. [Online; abgerufen am 23.11.2020]. 2019. URL: https://apps.apple.com/de/app/devices-control-for-homekit/id966877433.

[32] LinkDesk. *LinkDesk*. [Online; abgerufen am 23.11.2020]. URL: linkdesk.com.

[33] LinkDesk. *LinkDesk*. [Online; abgerufen am 23.11.2020]. URL: https://www.linkdesk.com/help/devices/.

[34] Shanhong Liu. *Internet of Things (IoT) - Statistics and Facts*. [Online; abgerufen am 25.11.2020]. 2020. URL: https://www.statista.com/topics/2637/internet-of-things/.

[35] Google LLC. *Google Home - Apps bei Google Play*. [Online; abgerufen am 23.11.2020]. 2020. URL: https://play.google.com/store/apps/details?id=com.google.android.apps.chromecast.app&hl=de&gl=US.

[36] *Model View ViewModel - Wikipedia*. [Online; abgerufen am 04.11.2020]. 2020. URL: https://de.wikipedia.org/wiki/Model_View_ViewModel.

[37] *MVC Framework - Introduction*. [Online; abgerufen am 06.11.2020]. 2020. URL: https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm.

[38] Boris Pokric u. a. „Augmented Reality Enabled IoT Services for Environmental Monitoring Utilising Serious Gaming Concept“. In: *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.* 6 (2015), S. 37–55.

[39] D. Steinberg S. Cheshire. *Zero Configuration Networking: The Definitive Guide*. O'Reilly Media, 2005. ISBN: ISBN 0-596-10100-7.

[40] M. Krochmal S. Cheshire. *RFC 6762 - Multicast DNS*. [Online; abgerufen am 26.10.2020]. 2013. URL: https://tools.ietf.org/html/rfc6762.

[41] M. Krochmal S. Cheshire. *RFC 6763 - DNS-Based Service Discovery*. [Online; abgerufen am 26.10.2020]. 2013. URL: https://tools.ietf.org/html/rfc6763.

[42] Lukas Smirek, Gottfried Zimmermann und Michael Beigl. „Just a Smart Home or Your Smart Home – A Framework for Personalized User Interfaces Based on Eclipse Smart Home and Universal Remote Console". In: *Procedia Computer Science* 98 (2016), S. 107–116. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2016.09.018. URL: http://www.sciencedirect.com/science/article/pii/S1877050916321391.

[43] *The Model-View-ViewModel Pattern*. [Online; abgerufen am 04.11.2020]. 2017. URL: https://docs.microsoft.com/de-de/xamarin/xamarin-forms/enterprise-application-patterns/mvvm.

[44] ubot. *Ubuntuusers, Avahi*. [Online; abgerufen am 12.10.2020]. 2019. URL: https://wiki.ubuntuusers.de/Avahi.

[45] A. M. Ullah u. a. „Remote-touch: Augmented reality based marker tracking for smart home control". In: *2012 15th International Conference on Computer and Information Technology (ICCIT)*. 2012, S. 473–477. DOI: 10.1109/ICCITechn.2012.6509774.

[46] Mariella Wendel. *Diese Geräte sind mit Google Home steuerbar*. [Online; abgerufen am 23.11.2020]. 2019. URL: https://www.homeandsmart.de/google-home-kompatible-geraete.

Versicherung an Eides Statt

Ich versichere an Eides statt durch meine untenstehende Unterschrift,

- dass ich die vorliegende Arbeit - mit Ausnahme der Anleitung durch die Betreuer - selbstständig ohne fremde Hilfe angefertigt habe und
- dass ich alle Stellen, die wörtlich oder annähernd wörtlich aus fremden Quellen entnommen sind, entsprechend als Zitate gekennzeichnet habe und
- dass ich ausschließlich die angegebenen Quellen (Literatur, Internetseiten, sonstige Hilfsmittel) verwendet habe und
- dass ich alle entsprechenden Angaben nach bestem Wissen und Gewissen vorgenommen habe, dass sie der Wahrheit entsprechen und dass ich nichts verschwiegen habe.

Mir ist bekannt, dass eine falsche Versicherung an Eides Statt nach §156 und §163 Abs. 1 des Strafgesetzbuches mit Freiheitsstrafe oder Geldstrafe bestraft wird.

Duisburg, 22. Dezember 2020
(Ort, Datum)

(Vorname Nachname)