Masterarbeit im Studiengang

Information Systems Engineering

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Entwicklung eines Systems zur Erzeugung, Anbindung und Simulation von virtuellen Gegenständen

- am Beispiel einer realen Smart-Living-Modelllandschaft -

Fachhochschule Aachen

University of Applied Sciences

Fachbereich

Elektrotechnik und Informationstechnik

von Mathias Kohs

845781

Erstprüfer: Prof. Dr.-Ing. Martin R. Wolf

Zweitprüfer: Johannes König M. Eng.

Eigenständigkeitserklärung

Hiermit versichere ich, Mathias Kohs (Matrikel Nr. 845781), dass ich diese Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind.

06.09.2019 Aachen

_____

Mathias Kohs

Inhaltsverzeichnis

1 EINLEITUNG 1

1.1 Anlass 1
1.2 Technologie 1
1.2.1 Digitaler Zwilling 1
1.2.2 MQTT 2
1.2.3 Regelzentrierte Logiken 4
1.3 Herausforderungen mit Forschungsprototypen 4
1.3.1 Wiederverwendbarkeit 5
1.3.2 Skalierbarkeit 5
1.3.3 Modularität 5
1.4 Ziel der Entwicklung 6
1.5 Stand der Forschung 6
1.6 Forschungsfrage 7
1.7 Methodische Konzeption 7

2 ANFORDERUNGSANALYSE 9

2.1 Stakeholder 9
2.2 Soll-Situation 14
2.2.1 Nicht direkt aufgezeigte Stakeholder 14
2.3 Ist-Situation 15
2.3.1 Im Projekt verwendete Programmiersprachen 15
2.4 Anforderungen generisches Metamodel 16
2.4.1 Modell - Zentrales System 16
2.4.2 Simleiter - Zentrales-System 18
2.4.3 3rd-Party-System - Zentrales-System 21
2.4.4 Administrator - Zentrales-System 21
2.4.5 Proband - Modell 21
2.5 Nichtfunktionale Anforderungen 22
2.5.1 Qualitätsanforderungen 23
2.5.2 Technologische Anforderungen 23
2.5.2.1 3D- und VR-Umsetzung 24
2.5.2.2 Mittelschicht Umsetzung 25
2.5.2.3 Anforderungen an sonstige Lieferbestandteile 25
2.6 Prüfen der Anforderungen 26
2.7 Software Designprinzipien 27
2.7.1 SOLID-SRP: Single Responsibility-Prinzip 28
2.7.2 SOLID-OCP: Open Closed-Prinzip 28
2.7.3 SOLID-LSP: Das Liskov'sche Substitutionsprinzip 28
2.7.4 SOLID-ISP: Interface Segregation-Prinzip 28
2.7.5 SOLID-DIP: Dependency Inversion-Prinzip 29

3 TECHNOLOGIE WAHL 30

3.1 Modelllandschaft 30
3.2 3D- und VR-Welt 30
3.2.1 Game Engine 30
3.2.2 VR-Toolkit 31
3.2.2.1 Auswahl 32
3.2.3 MQTT-Binding an Unity3D (vorhandene Lösungen) 33
3.2.3.1 Kandidat 1 33
3.2.3.2 Kandidat 2 33

3.2.3.3 Kandidat 3 ... 34
3.2.3.4 Kandidat 4 ... 34
3.2.4 MQTT-Binding an Unity3D über angepasste Libraries ... 34
3.2.4.1 Vorgehen ... 35
3.2.4.2 Auswahl ... 36
3.3 Mittelschicht ... 37
3.3.1 Programmiersprache ... 37
3.3.1.1 Stufe 1 Vorauswahl ... 37
3.3.1.2 Stufe 2 Auswahl ... 38
3.3.2 Software Plattform / Framework ... 39
3.3.3 MQTT-Client ... 39
3.3.4 Trennung der Oberfläche vom Backend ... 40
3.4 MQTT 5.0 oder MQTT 3.1.1 ... 40
3.5 Bezug zur Forschungsfrage ... 41
4 SYSTEMENTWICKLUNG ... 42
4.1 MQTT Topic Richtlinien ... 42
4.2 MQTT Unity3D Anbindung ... 43
4.2.1 Versuche mit der Bindung ... 43
4.2.1.1 Versuch 1 dry Metall ... 43
4.2.1.2 Versuch 2 Blocking Queue ... 45
4.2.1.3 Versuch 3 Active Waiting Queue ... 45
4.2.2 Details der Umsetzung ... 45
4.2.2.1 Zentraler-MQTT-Connector (ZMC) ... 46
4.2.2.2 Singleton-Like Ansatz ... 46
4.2.2.3 Observer Pattern / Publish-subscribe Pattern ... 50
4.2.2.4 UnityEvent Baustein ... 50
4.2.2.5 MQTT-Konfiguration ... 51
4.2.3 Nachträglicher Vergleich ... 52
4.2.3.1 Vergleich Grobdesign ... 53
4.2.3.2 Vergleich Lösung der Nebenläufigkeit ... 54
4.2.3.3 Beispielbenutzung ... 55
4.2.3.4 Wahl der Lösung für diese Anwendung ... 55
4.3 MQTT-Mittelschicht ... 55
4.3.1 MQTT-Mittelschicht-Varianten ... 55
4.3.1.1 Regelung je Endgerät / Micro-Controller ... 55
4.3.1.2 Topic Trenner ... 56
4.3.1.3 MQTT-Broker neu schreiben ... 56
4.3.1.4 MQTT-Broker-Plugin schreiben ... 57
4.3.1.5 Wahl der Entwicklungsvariante ... 57
4.3.2 Bausteinsicht der Mittelschicht ... 59
4.3.2.1 Grundstruktur ... 59
4.3.2.2 Services der Bausteine ... 60
4.4 Verfahrensansicht Mittelschicht ... 61
4.4.1 Verfahren des Topic Trenners ... 61
4.4.2 Mittelschicht Topic Regeln ... 62
4.4.2.1 Access Regelauswertung ... 62
4.4.2.2 Deny Regelauswertung ... 64
4.4.2.3 Anlehnung an das Subscriben der Topics ... 65
4.4.3 Verarbeitungsansatz der Topics ... 65
4.4.3.1 Analyse einer Bestehenden Lösung aus einem Broker ... 65
4.4.4 Abbilden der Regel über Knoten ... 66
4.4.5 Abbilden der Daten in Datenbank ... 69
4.4.6 Service Start über Controller ... 70
4.5 Strukturansicht der Mittelschicht ... 71

4.5.1 Dependency Injection ... *71*
4.5.2 Services als Singleton ... *72*
4.5.3 Dependency Inversion ... *72*
4.5.4 Controller Manager Service ... *74*
4.5.4.1 Controller Manager Service Muster ... 75
4.6 Herausforderungen in der Anwendung ... 76
4.6.1 Kreisschlüsse im MQTT-Netzwerk ... *76*
4.6.1.1 Lösungsvorschlag Kreiserkennungskomponente ... 76
4.6.1.2 Lösungsvorschlag Kreiserkennung durch Heuristiken ... 76
4.6.2 Regeln mit Einfluss auf mehrere Devices und Welten ... *76*
4.6.3 Sonderfall Geräte ohne MQTT-Anbindung ... *76*
4.6.3.1 Versuchsvorschlag Implementieren der Protokolle ... 77
4.6.3.2 Lösungsvorschlag Abbilden der Protokolle durch OpenHAB Regeln ... 77
4.7 Bezug zur Forschungsfrage ... 77
5 EVALUATION DES SYSTEMS ... 78
5.1 Empirische Studie mit Probanden ... 78
5.1.1 Aussage: Darstellen von Virtuellen Gegenständen ... *78*
5.1.1.1 Konzept zum empirischen Evaluieren der Aussage ... 79
5.1.1.2 Einordnung in die Gesamtarchitektur ... 79
5.1.2 Aussage: SLE-Simulation auf Laborhardware ... *80*
5.1.2.1 Konzept zum empirischen Evaluieren der Aussage ... 80
5.1.2.2 Einordnung in die Gesamtarchitektur ... 80
5.1.3 Aussage: Entwickeln neuer Anwendungen ... *81*
5.1.3.1 Konzept zum empirischen Evaluieren der Aussage ... 81
5.1.3.2 Einordnung in die Gesamtarchitektur ... 82
5.1.4 Aussage: Mittelschicht erweiterbar durch API Controller ... *82*
5.1.4.1 Konzept zum empirischen Evaluieren der Aussage ... 82
5.1.4.2 Einordnung in die Gesamtarchitektur ... 83
5.1.5 Aussage: Mittelschicht erweiterbar durch Logger ... *84*
5.1.5.1 Konzept zum empirischen Evaluieren der Aussage ... 84
5.1.5.2 Einordnung in die Gesamtarchitektur ... 85
5.1.6 Aussage: Unity3D Endpunkt erweiterbar ... *85*
5.1.6.1 Konzept zum empirischen Evaluieren der Aussage ... 85
5.1.6.2 Einordnung in die Gesamtarchitektur ... 86
5.1.7 Feedback-Interview ... *86*
5.1.7.1 Fragen des Interviewleitfadens ... 87
5.2 Auswertung der empirischen Studie mit Probanden ... 87
5.2.1 Auswertung zur Aussage 5.2.2 ... *87*
5.2.1.1 Verbesserungen der Anleitung ... 88
5.2.1.2 Verbesserungen der GUI ... 88
5.2.1.3 Verbesserung des Systems ... 89
5.2.2 Auswertung zur Aussage 5.2.3 ... *89*
5.2.2.1 Verbesserungen der Dokumentation ... 90
5.2.3 Auswertung zur Aussage 5.2.4 ... *90*
5.2.3.1 Verbesserungen der Anleitung ... 90
5.3 Auswertung der Experteninterviews ... 91
5.4 Empirischer Belastungstest ... 92
5.4.1 Benchmarks zur Optimierung des Systems ... *93*
5.4.1.1 Versuchsaufbau ... 93
5.4.1.2 Versuch auf H1 ... 93
5.4.1.3 Versuch auf H3 ... 96
5.4.1.4 Abschluss des Benchmarks ... 98
6 FAZIT UND AUSBLICK ... 99

6.1 Zusammenfassung der Ergebnisse 99
6.2 Fazit 100
6.3 Weiterer Forschungsbedarf 101

7 ANHANG 102

7.1 Tools 102
7.2 Anforderungsliste 103
7.2.1 Studien Anforderungen 107
7.3 Anforderungsabdeckung 107
7.4 UML 109
7.5 Diagramme 111
7.6 Screenshots 112
7.7 Technologie Auswahl 140
7.8 Studie mit Probanden und Interviews 143
7.8.1 Proband B12 143
7.8.1.1 Beobachtung 5.2.2 143
7.8.1.2 Feedback Interview 5.2.2 143
7.8.1.3 Beobachtung 5.2.4 143
7.8.1.4 Feedback Interview 5.2.4 144
7.8.1.5 Beobachtung 5.2.5 144
7.8.1.6 Feedback Interview 5.2.5 144
7.8.2 Proband T13 145
7.8.2.1 Beobachtung 5.2.2 145
7.8.2.2 Feedback Interview 5.2.2 145
7.8.2.3 Beobachtung 5.2.4 146
7.8.2.4 Feedback Interview 5.2.4 146
7.8.3 Proband L14 147
7.8.3.1 Beobachtung 5.2.2 147
7.8.3.2 Feedback Interview 5.2.2 147
7.8.3.3 Beobachtung 5.2.3 147
7.8.3.4 Feedback Interview 5.2.3 148
7.8.3.5 Beobachtung 5.2.4 148
7.8.3.6 Feedback Interview 5.2.4 148
7.8.4 Proband I16 149
7.8.4.1 Beobachtung 5.2.2 149
7.8.4.2 Feedback Interview 5.2.2 149
7.8.4.3 Beobachtung 5.2.4 149
7.8.4.4 Feedback Interview 5.2.4 149
7.8.5 Proband T19 150
7.8.5.1 Beobachtung 5.2.2 150
7.8.5.2 Feedback Interview 5.2.2 150
7.8.5.3 Beobachtung 5.2.4 150
7.8.5.4 Feedback Interview 5.2.4 151
7.8.5.5 Beobachtung 5.2.3 151
7.8.5.6 Feedback Interview 5.2.3 151
7.8.6 Proband / Expertin S15 152
7.9 Experteninterview 153
7.9.1 Experte B12 153
7.9.2 Experte K12 155
7.10 Fotos SMILE Exponat 156

8 ABBILDUNGSVERZEICHNIS 161

9 TABELLENVERZEICHNIS 163

10 LITERATURVERZEICHNIS 164

1 Einleitung

1.1 Anlass

Die Masterarbeit nimmt Bezug auf das interdisziplinäre Forschungsprojekt SMART INFRASTRUCTURE FOR LIVING ENVIRONMENTS ((Jacobas et al. 2016)) der FH Aachen. In diesem Projekt geht es darum unter Einbeziehung von möglichst breiten Kompetenz- und Wissensbereichen neue smarte Services zu entwickeln, prototypisch umzusetzen und zu bewerten.

Als Teil eines mehrstufigen Prozesses werden die Ideen in eine Modelllandschaft im Maßstab 1:1 und 1:18 übertragen. Dabei sind normale Gegenstände mit Sensorik und Aktorik ausgestattet, die dadurch zu SLE-Bausteinen werden, die die Werte und Informationen mit einer zentralen Einheit tauschen. Die zentrale Einheit hält die Regeln (If-than-that Aufbau), aus denen die regelbasierten Services aufgebaut werden. Aus der Kombination aus den regelbasierten Services und den SLE-Bausteinen entstehen SLE-Bricks, die die Services anfassbar und erlebbar machen. Die Rückkopplung durch das direkte Ausprobieren der Services und die schnelle Anpassbarkeit sorgt für eine beschleunigte Verbesserung, der entstehenden Services.

Um auch den Ausfall von bestimmten Sensoren oder Aktoren simulieren zu können, wird eine Mittelschicht gebaut, die die Signale bestimmter Sensoren und Aktoren trennen kann. Zusätzlich soll die Mittelschicht in der Lage sein, Digitale Zwillinge (Erklärung *K. 1.2.1*) mit Daten zu versorgen und zwischen den Daten der Modellwelten umzuschalten. Um bestimmte Ideen ausprobieren zu können, die in der Modelllandschaft zu schwer darzustellen sind, wird eine Lösung geschaffen mit der die MQTT-Nachrichten (Erklärung *K. 1.2.2*) in der 3D-Engine Unity3D entgegengenommen und darauf reagiert sowie gesendet werden können.

1.2 Technologie

1.2.1 Digitaler Zwilling

Ein Digitaler Zwilling (DZ.) ist eine digitale oder virtuelle Repräsentanz eines Gegenstands (Produkt, Fertigungsanlage, o.ä.) aus der realen Welt, der die meisten aller Eigenschaften teilt. Der Zwilling ist zeitlich entkoppelt, damit ist es für den Zwilling egal, ob dessen Gegenstück in der realen Welt existiert, nicht existiert oder noch nicht existiert. Digitale Zwillinge sind mehr als nur Daten und können Modelle, Simulationen und Algorithmen enthalten, die ihr Gegenstück aus der realen Welt und dessen Eigenschaften und Verhalten genauer beschreiben (vgl. (Grösser Stefan (Berner Fachhochschule) 2019), (Kaufmann, Pühringer, and Rauscher 2016), (Bracht, Geckler, and Wenzel 2018), (Monostori et al. 2016)).

Der Begriff DZ. wurde primär durch die Industrie 4.0-Bewegung geprägt, was an den Referenzen zu sehen ist (vgl. (Kunze 2016; Kuhn 2017)). Dennoch bieten viele Definitionen die Möglichkeit auch außerhalb von Industrie 4.0 von DZ. zu sprechen (vgl. (Kuhn 2017)). In dieser Arbeit wird von DZ. gesprochen, wenn mehrere Instanzen desselben Systems existieren. Speziell im Projekt SMILE liegen drei bis vier Welten vor, in denen diese Instanzen existieren können: in der 1:18-Modelllandschaft, im 1:1 Modell, in der VR-Welt und ggf. in der realen Welt als fertiges Produkt. Diese Instanzen können auch in allen Welten existieren und eine virtuelle Verbindung über ihre gemeinsamen digitalen Daten haben. Die gemeinsamen Daten machen diese Instanzen zu Zwillingen, Drillingen oder Vierlingen, trotzdem wird in der Arbeit o.B.d.A. von DZ. gesprochen.

1.2.2 MQTT

MQTT ist ein Nachrichtenprotokoll für die Kommunikation zwischen Maschinen und ermöglicht den Austausch zwischen Geräten auch bei instabilen Verbindungen. MQTT wurde in der Vergangenheit stark bei Scada-Systemen, sowie anderen Systemen in der produzierenden Industrie verwendet und findet jetzt bei IoT-Geräten Einsatz (vgl. ((HiveMQ) 2019c), ((mqtt.org) 2019), (Sand and Reiff-Stephan 2018)). Dieses Protokoll ist durch OASIS und ISO standardisiert ((ISO 2016), (OASIS 2014)), die letzte fertige Version 3.1.1 ist 2014 bestätigt worden.

Die Nachrichten werden nach dem Beobachterverfahren (Observer Pattern (Freeman Eric, Freeman Elisabeth, Sierra Kathy 2008), (Gamma et al. 2001)) verteilt und durch einen Broker verwaltet. Die Nachrichten sind nicht an Clients gerichtet, sondern an Topics, die eine hierarchische Struktur, wie URLs, erlauben. Durch die Brokerzentrierung entfällt die Notwendigkeit, dass die Geräte sich direkt kennen müssen und eine geringe Kopplung entsteht.

In dem Projekt SMILE wurde das Nachrichtenprotokoll ausgewählt und bereits ausgiebig genutzt. Diese Technologie wird weiterverwendet, um die bestehenden Systeme zu unterstützen. Trotzdem ist die Frage zu stellen, ob MQTT aktueller Stand ist, denn alte Berichte zu dem Thema zeigen, dass dieser Markt durchaus umkämpft ist, wie ein Artikel der informatik.aktuell.de zeigt (vgl. (Obermaier 2015)). Aus der *Abbildung 1* lässt sich ableiten das schon 2015 in den Suchen anstieg und damit in der Bekanntheit MQTT einen positiven Trend aufwies. In dem Artikel wurden die IoT-Protokolle mqtt, coap und xmapp verglichen, die zu dem Zeitpunkt vorherrschend waren (vgl. (Rowe 2019), (Stansberry 2015)).

Abbildung 1: Google-Trends Mqtt coap xmpp von informatik-aktuell.de (Y-Achse zeigt das Interesse, dient aber nur als Richtwert)

Aktuelle Infos zu dem Thema MQTT zeigen, dass sich der Trend bis heute weiter zieht, wie in *Abbildung 2* zu sehen (vgl. dynamische URL (Google 2019a)).

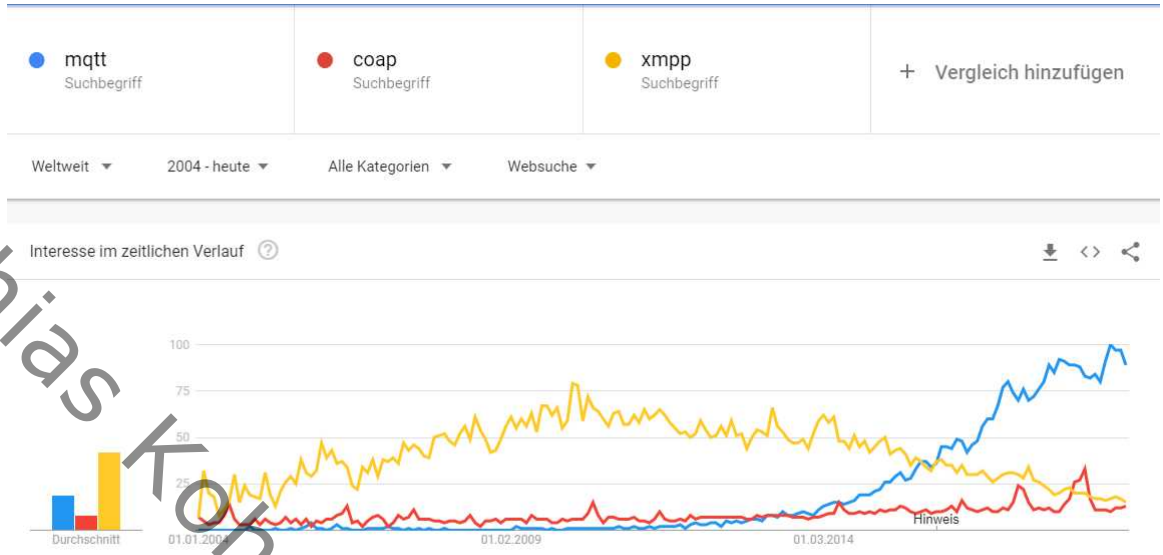


Abbildung 2: Google-Trends Mqtt coap xmpp Q1 2019

Die Standards erfahren ebenfalls Updates; OASIS hat den nächsten Standard fast fertig, denn seit 2018-10-31 gibt es mqtt5.0 als „Candidate OASIS Standard“ (vgl. (A. Banks et al. 2018)), der damit fast abgeschlossen ist.

MQTT ist als Nachrichten-Technologie für das Forschungsprojekt gesetzt. Für neue Projekte wäre eine Betrachtung des Marktes relevant, denn alternative Technologien lassen sich finden, speziell wenn nach SmartHome-Protokollen (statt nach IoT-Protokollen) gesucht wird. Dazu zeigt *Abbildung 3* nur eine Auswahl.

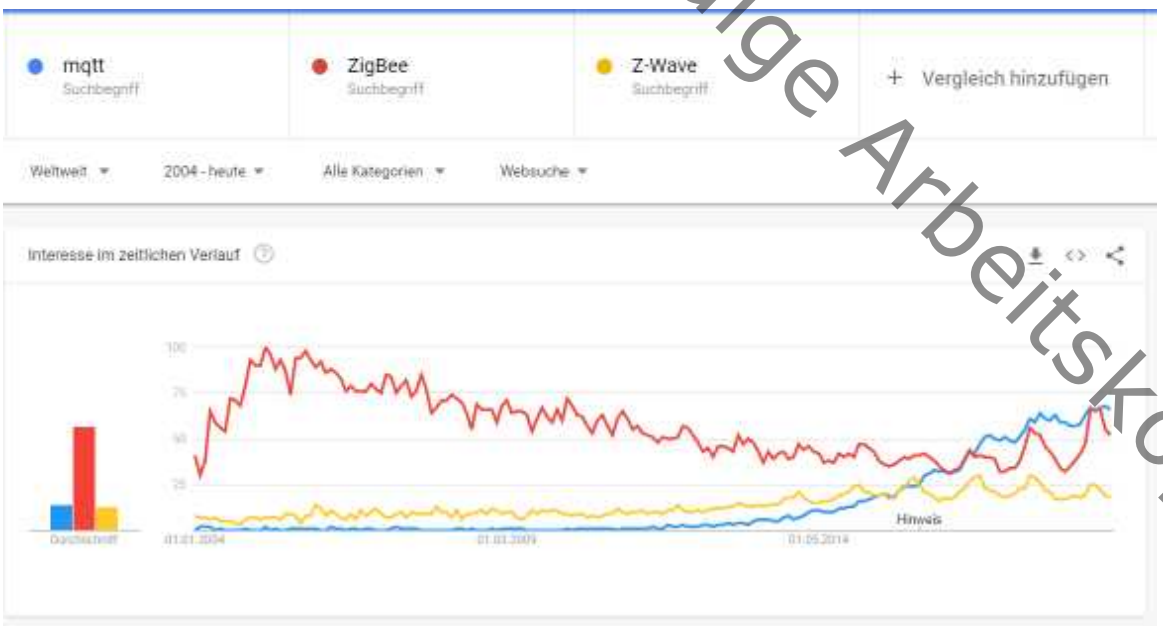


Abbildung 3: Google Trends Mqtt ZigBee Z-Wave Q1 2019

1.2.3 Regelzentrierte Logiken

In dem SMILE-Projekt hat relativ früh OpenHAB2, ein System für regelbasierte Logik, den Einsatz gefunden (vgl. (Kaiser 2018)). Für das System in SMILE, welches die Mittelschicht bildet, ist es wichtig, dass die Verwendung von regelzentrierten Auswertungssystemen weiterhin bestehen bleibt und die Systeme, wie sie bereits eingesetzt werden, weiter genutzt werden können. Auf Basis dieses Regelsystems wurden weitere Tools wie Regeleditoren nach aktuellen Usability-Richtlinien gebaut (vgl. (Kaiser 2018)).

1.3 Herausforderungen mit Forschungsprototypen

Prototypen werden zunächst gebaut, um eine These zu bestätigen oder zu widerlegen. Diese Prototypen spalten sich damit in mindestens zwei Gruppen auf: Diejenigen, die nach Bestätigung der These nicht weiter beachtet werden, und jene, die als Exponat oder Basis für weitere Forschung wiederverwendet werden. Damit ergeben sich weniger sichtbare Herausforderungen für Forschungsprototypen, die als Nicht-Funktionale Anforderungen betrachtet werden müssen. Die Basis für diese Arbeit ist das Forschungsprojekt SMILE und dieses arbeitet mit den Ergebnissen dieser Arbeit weiter, darunter dem Prototyp.

Daraus erwächst die Herausforderung der Wiederverwendbarkeit. Die Wiederverwendbarkeit ist ein Qualitätskriterium, was frühzeitig betrachtet werden muss. Aus der (Weiter- oder) Wiederverwendbarkeit ergeben sich andere Anforderungen an die Prototypen. Es sind Teilmengen jener Anforderungen, die auch die Software oder die Produkte der Industrie erfüllen müssen. Es ist wahrscheinlich, dass sich nicht alle Anforderungen in der echten Umsetzung dieser Arbeit wiederfinden, jedoch soll die Forschungsfrage die Entwicklung in Architektur und Software in die Richtung lenken, die dafür sorgt, dass die Ergebnisse der Arbeit wiederverwendet werden können.

Eine Teilmenge der Nicht-Funktionalen-Anforderungen, die daraus erwachsen, sind Qualitätsanforderungen (Dienstgüte). Qualitätsanforderungen für Softwaresysteme wurden in der ISO/IEC 9126 und der Nachfolger-Norm ISO/IEC 25000 „Software Engineering – Software product Quality Requirements and Evaluation“ definiert.

Die ISO/IEC 25000 unterscheidet sechs Merkmale der Dienstgüte/Qualitätsanforderungen: Änderbarkeit, Benutzbarkeit, Effizienz, Funktionalität, Zuverlässigkeit und Übertragbarkeit.

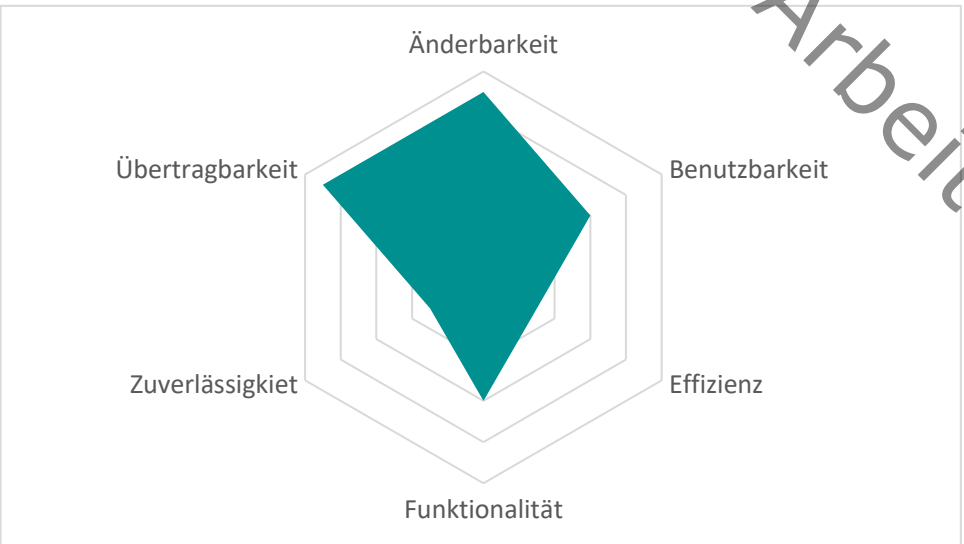


Abbildung 4: Netzdiagramm mit 6 Merkmalen der Dienstgüte für den Prototypen (Richtwert)

Die Verteilung der Wichtigkeit der Merkmale lässt sich wie in *Abbildung 4* aufschlüsseln, wobei sich aus der Abbildung ein Richtwert aber keine Zahlenwerte ablesen lassen sollen. Da es sich um einen Prototyp handelt, ist die Zuverlässigkeit und die Effizienz niedrig bewertet im Vergleich zu den anderen vier Merkmalen. Speziell in dieser Arbeit spielen die Übertragbarkeit und die Änderbarkeit eine wichtige Rolle und münden in der Wiederverwendbarkeit des Systems, mit der Berücksichtigung, dass in der Wiederverwendbarkeit auch die weitere Anpassung, also Änderbarkeit steckt. Die Benutzbarkeit ist mittig angesetzt, da die Benutzbarkeit (zusammen mit der Änderbarkeit) für den Entwickler genauer betrachtet wird, jedoch die Benutzbarkeit für den Endbenutzer speziell zusammen mit der damit verbundenen Oberflächengestaltung (GUI) nicht. Ähnlich verhält es sich mit der Funktionalität, da die Funktionalität des Systems in der Anpassbarkeit liegt. In dieser Arbeit bietet der Prototyp lediglich ein Werkzeug mitsamt eines Beispiels, um konkretere Funktionalitäten abzubilden.

Es stellt sich nun die Frage, ob es sinnvoll ist, im Sinne der Verhältnismäßigkeit für die Wiederverwendbarkeit Aufwand zu investieren. Angenommen, dass das Projekt SMILE in die Phase der Nutzung bzw. Erprobung geht (vgl. Kap2. (Jacobas et al. 2016)), ist damit zu rechnen, dass noch Änderungen gemacht werden und dieser Prototyp weiter verwendet und entwickelt wird. Somit lässt sich schlussfolgern, dass es wichtig ist, dass das Thema der Wiederverwendbarkeit und der damit verbundenen Größen wichtig ist und Aufmerksamkeit bedarf.

1.3.1 Wiederverwendbarkeit

Die Wiederverwendbarkeit gipfelt in der generischen Verwendbarkeit, die häufig in Softwareentwicklung angestrebt wird. Aus der generischen Verwendbarkeit leitet sich ab, dass möglichst viele Anwendungsfälle abgedeckt werden können und neue Anwendungsfälle mit keinem, bis geringem Aufwand ebenfalls erfüllt werden können. Im Extremum sorgt die generische Verwendbarkeit dafür, dass dieses System auch außerhalb der SLE Einsatz findet.

1.3.2 Skalierbarkeit

Nach der Wiederverwendbarkeit ergibt sich die Frage der Skalierbarkeit, da durch hohe Wiederverwendbarkeit auch die Anzahl der Geräte, Anwendungen und Nutzer steigt, die das System verwenden können. Die Skalierbarkeit hat zwei Disziplinen, die für diese Arbeit betrachtet werden: Die Skalierbarkeit in der Administration und die Skalierbarkeit in der Last.

Die Skalierbarkeit in der Administration beschreibt den Aufwand der Administration bei ansteigender Geräte-, User- und Detailzahl. Diese Betrachtung ist wichtig, da der Forschungsantrag von SMILE offen genug gestaltet ist, dass weitere Forscher Anwendungen an das SMILE-Modell anbauen können. Dabei ist es wichtig, dass dieses System immer noch durch Menschen überblickt, konfiguriert und analysiert werden kann.

Skalierbarkeit in der Last beschreibt die Möglichkeit, das System bei wachsenden Anforderungen weiter auszubauen, damit dieses System der wachsenden Zahlen von Geräte- und Useranfragen immer noch zuverlässig funktioniert. Diese Disziplin der Skalierbarkeit hat Ähnlichkeit mit dem Merkmal der Effizienz, jedoch ohne den breiten Anspruch und die Dringlichkeit dessen.

1.3.3 Modularität

Für die Wiederverwendbarkeit der Komponenten des Systems ist die Modularität der Punkt, der angibt, ob diese Komponenten auch isoliert funktionieren oder in anderen Systemen genutzt werden können. Dieser Punkt erhöht die Wahrscheinlichkeit, dass eine Teilmenge des Prototyps

weiterverwendet werden kann. Die weitere Verwendung kann auch in ganz anderen Aufgaben liegen, bei denen nur Teile der Anforderungen mit den aktuellen übereinstimmen. Ebenfalls unterstützt die Modularität den Forschungsalltag, denn wenn an einem Teil des Systems gearbeitet wird und dieses damit nicht nutzbar (oder vorzeigbar) ist, sorgt die Modularität und die geringere Abhängigkeit dafür, dass die anderen Teile des Systems genutzt oder vorgezeigt werden können.

1.4 Ziel der Entwicklung

Ziel der Entwicklung ist es, das aktuelle System dahin weiter zu entwickeln, dass es möglich wird, in der Benutzung zwischen den Modellen zu wechseln und in dem 1:1-Modell dieselben Zustände wiederzufinden wie in dem 1:18-Modell oder in der VR-Welt. Ebenso soll es möglich sein diese Modelle parallel zu nutzen und die Zustände nicht modellübergreifend zu teilen, damit die Modelle autark arbeiten. Damit die Simulation der Störungen erfolgen kann, soll darüber hinaus das Modell dem Spielleiter die Möglichkeit geben Störfälle auszulösen, um unter anderem den Ausfall von Geräten zu simulieren. Die Schnittstellen an das MQTT-Protokoll sind bereits für das 1:1- und 1:18 Modell implementiert, für die VR-Welt dagegen nicht. Die Anbindung an die VR-Welt wird für die Engine Unity3D umgesetzt.

1.5 Stand der Forschung

Da diese Arbeit vordringlich von konstruktiver Art ist, muss der Stand der Forschung beleuchtet werden. Falls die Forschung bereits einen Lösungsvorschlag für das Ziel der Entwicklung inklusive der Herausforderungen dieses Forschungsprototypen bereithält, müsste man diese Lösung nur implementieren und auf Tauglichkeit untersuchen.

Um den Stand der Forschung zu begreifen, wurde generell zu dem Thema recherchiert und speziell folgende Stichwort-Tupel untersucht:

- MQTT Simulation
- MQTT Digital Twin
- Smart Living Simulation

Aus der Recherche lässt sich entnehmen, dass bereits andere Forschungsgruppen MQTT-Modelllandschaften gebaut haben. Unter den gefunden Modelllandschaften lassen sich vermehrt Industrie 4.0-Anwendungen finden. Darüber hinaus lassen sich Anwendungen finden, die ähnlich unserem Ziel sind, aber keine Modelllandschaften verbinden, wie z.B. in dem Journal-Artikel „Vernetzung von physischen und virtuellen Entitäten im cyberphysischen Produktionssystem“ (vgl. (Van de Sand et al. 2019)). Dieses Team hat unter anderem eine AR-Brille über MQTT an das Produktionssystem angeschlossen, was nah unserem Ziel ist, eine VR-Welt an die Modelllandschaft anzubinden.

In der Recherche ließ sich kein direkter Lösungsvorschlag oder eine Kombination von Lösungsvorschlägen finden. Damit reicht es nicht aus, die Lösungsvorschläge, zu implementieren, sondern eine eigene Entwicklung muss durchgeführt und beleuchtet werden.

1.6 Forschungsfrage

Aus dem aktuellen Anlass des SMILE-Projekts und den Herausforderungen der Forschungsprototypen sowie der bisherigen Technologiewahl ergibt sich folgende Forschungsfrage:

Welcher technische Ansatz ermöglicht es, Gegenstände und digitale Zwillinge in eine mit OpenHAB und MQTT betriebene SLE-Modellandschaft unter Beachtung der Aspekte

- generische Verwendbarkeit,
- Skalierbarkeit,
- Modularität und
- regelzentrierte Logiken

einbinden und simulieren zu können?

1.7 Methodische Konzeption

Das Vorgehen der Forschungsphase ist stark anforderungsgetrieben. Die Anforderungen sind aus der Forschungsfrage abgeleitet oder sind durch die Analyse mit dem Projektteam entstanden. Damit die Entwicklung anforderungsgetrieben funktioniert, ist die Entwicklung iterativ, inkrementell erfolgt, um in jeder Iteration die Möglichkeit zu haben die Ergebnisse mit den Anforderungen zu vergleichen. Ebenfalls hat sich das Anforderungsprofil jeweils weiterentwickelt.

Die Art, wie die Ergebnisse mit den Anforderungen verglichen werden, ähnelt stark dem Modell der Regelung, wie man es schon in den 1960er-Jahren modellhaft abgebildet hat (vgl. (Wiener 1961)). Dieses Modell findet in vielen Anwendungen von der Naturwissenschaft über die Regelungstechnik bis hin zum Unternehmensaufbau oder Produktdesign Nutzen (vgl. (Ries 2014)). Die ersten Modelle, die den Weg in die Softwareentwicklung gefunden haben, findet man in den 1975er-Jahren (vgl. (Basili and Turner 1975)). Heute findet man diese Modelle in kombinierter Form im Agilen Manifest (vgl. (Cunningham et al. 2001)) oder in Scrum (vgl. (Scrum.org 2019)).

Das **erste Kapitel** leitet das Thema ein und zeigt ein Ausblick auf die methodische Konzeption.

Im **zweiten Kapitel** werden die Anforderungen dargestellt, auf Basis dessen die Entscheidungen in der Technologiewahl und Entwicklung ausgerichtet werden. Dabei werden die Stakeholder sowie dessen Funktionale- und nicht Funktionale Anforderungen erläutert.

Im **dritten Kapitel** wird die Technologiewahl dargestellt und begründet. Die Wahl für die Bereiche VR-Welt und Mittelschicht werden fokussiert, da in dem Forschungsprojekt dafür noch keine Umsetzung existiert.

Im **vierten Kapitel** werden Software und Systementwicklung auf Modellebene und Umsetzungsdetails betrachtet mit Hinblick auf die Softwarearchitektur, die durch Anforderungen gestützt wird. Wie auch in dem Kapitel Technologiewahl liegt der Fokus auf der Mittelschicht und der Anbindung der 3D-Welt, hinzu kommt die Betrachtung des Gesamtsystems.

Um in der Mittelschicht zu der richten Wahl der Architektur zu kommen, werden mehrere Ansätze miteinander verglichen und das Ergebnis in einem Entwurf umgesetzt und einem Test unterzogen.

Für die Anbindung der 3D-Engine wird auf Basis der Technologiewahl Software entworfen, um die gewählten Technologien zu verbinden. Dazu werden Versuche durchgeführt und die Ergebnisse verglichen.

Im letzten Teil dieses Kapitels wird das Gesamtsystem betrachtet, denn nicht nur die Teile müssen für das Zusammenspiel vieler Komponenten betrachtet werden, sondern auch das Ganze selbst. Dabei werden Regeln, Schnittstellen und Verhalten untersucht und aufgestellt, damit die Systeme unter Berücksichtigung der Anforderungen zusammenarbeiten.

In dem **fünften Kapitel** wird das entstandene System anhand der Anforderungen evaluiert. Die Evaluation ermöglicht neben dem Test der Funktion das Erfassen der Tätigkeit im Hinblick darauf, wie gut dieses System den Nicht Funktionalen Anforderungen gerecht wird.

Im **sechsten Kapitel** werden die gesammelten Erkenntnisse in eine Schlussbemerkung gefasst und bewertet, sowie der weitere Forschungsbedarf aufgezeigt.

2 Anforderungsanalyse

Die Anforderungsanalyse ist Basis für die Entscheidungen der folgenden Kapitel, wie Technologie Wahl oder Systementwicklung. Um genügend Informationen dafür zu schaffen, werden die Stakeholder, die Funktionalen Anforderungen und die Nichtfunktionalen Anforderungen erhoben und analysiert. Alle Anforderungen wurden durch nicht-formale Interviews mit dem Projektteam erhoben und nach Verschriftlichung vorgelegt.

Für das Dokumentieren in Grafischer Form werden Diagramme aus dem UML-Standard verwendet ((Object Management Group (OMG) 2017), (Rupp, Queins, and die SOPHISTen 2012)). Für das Dokumentieren der Anforderungen in sprachlicher Form, werden die Anforderungsschablonen von RUPP und den SOPHISTen herangezogen (vgl. Kap.10 (Rupp and SOPHISTen 2014)).

In Anlehnung an die Anforderungsschablonen und deren Schlüsselwörter für rechtliche Verbindlichkeiten möchte ich auch diese nutzen (vgl. Kap.10.4.1 (Rupp and SOPHISTen 2014)). Um diese in dem Kontext der Arbeit nutzen zu können, müssen diese Schlüsselwörter etwas uminterpretiert werden.

- Die Anforderungen, die das Schlüsselwort „**muss**“ enthalten, gehen verpflichtend in die *Technologie Wahl* mit ein und haben den Hauptfokus in der *Systementwicklung*.
- Die Anforderungen, die das Schlüsselwort „**sollte**“ enthalten, werden in der Entwicklung berücksichtigt und bilden erweiterten Input, sind aber nicht verpflichtend.
- Die Anforderungen, die das Schlüsselwort „**wird**“ enthalten, werden in der Entwicklung lose betrachtet, werden aber nicht näher beleuchtet oder umgesetzt. Diese Anforderungen, dienen den nachfolgenden Entwicklern, um das Gesamtsystem zu verstehen und Empfehlungen zu geben, was in der nächsten Iteration ansteht.

2.1 Stakeholder

„Bevor man versucht, die gewünschte Funktionalität eines Systems zu bestimmen, muss man Antworten auf zwei zentrale Fragen finden“ ((Kleuker 2009) S.52).

1. Die Anforderungen welcher Personenkreise müssen berücksichtigt werden? (Kleuker 2009)
2. Welche zentralen Ziele stecken hinter dem Software Projekt? (Kleuker 2009)
   (Das Ziel wurde in unter *1.4 Ziel der Entwicklung* definiert)

Im Englischen werden die Personen, die Anforderungen formulieren können, als Stakeholder bezeichnet (vgl. (Kleuker 2009) S.52). Ich werde diesen Begriff übernehmen, da die Übersetzung missverständlich sein kann.

Da Stakeholder Personengruppen sind, die über die klassischen Endanwender hinausgehen, werden die Stakeholder einzeln aufgeführt. Die Darstellung der Stakeholder erfolgt unter Zuhilfenahme der Unified Modeling Language (UML), um die Ableitung der Stakeholder darzustellen (vgl. Kap. 7.7.3.1 (Object Management Group (OMG) 2017)).

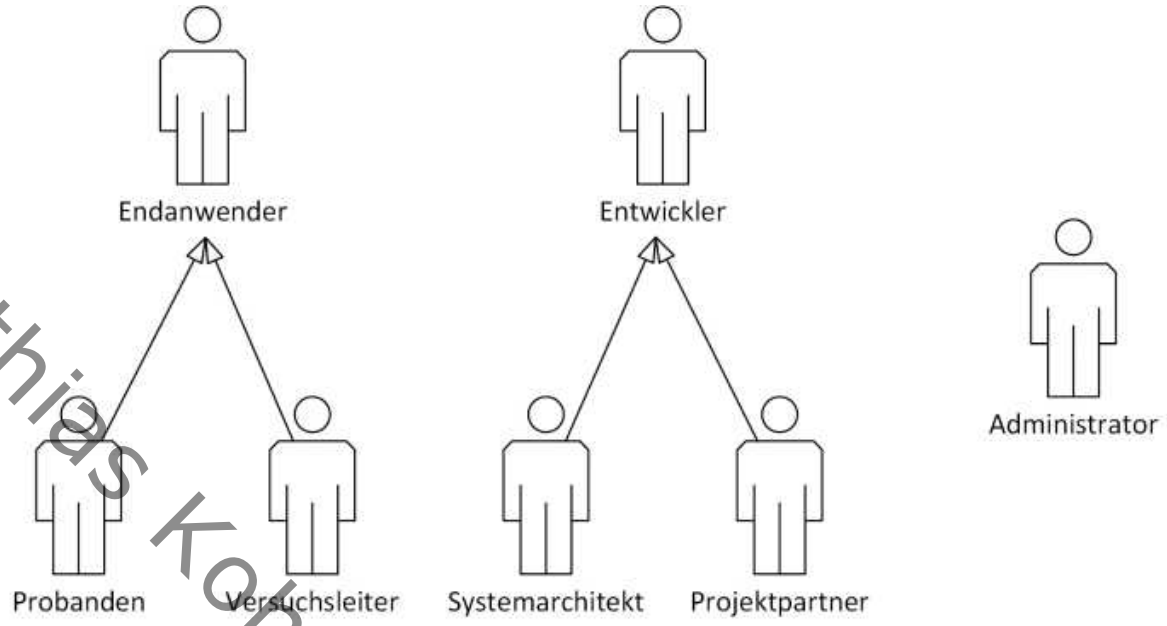


Abbildung 5: Ableitung (Generalisierung / Spezialisierung) von Stakeholdern

Die Ableitung bzw. die Spezialisierung erfolgt, um zu unterstreichen, dass verschiedene Stakeholder gemeinsame Attribute besitzen [1]. Ein Beispiel ist, dass Probanden eine Spezialisierung der Endanwender sind und damit die Attribute der Endanwender besitzen und diese um die eigenen Attributausprägungen erweitern. In anderen Worten, generalisiert der Endanwender die gemeinsamen Attribute der Probanden und Versuchsleiter.

Es folgen nun Tabellen mit Stakeholder-Definitionen, um die Attribute der Stakeholder weiter zu spezifizieren (vgl. (Kleuker 2011) S.51f).

Tabelle 1: Stakeholder Projektleiter

Projektleiter (des Forschungsprojekts)	
Wer ist das genau?	Zu den Projektleitern gehören die aktuellen Projektleiter und in der Unternehmenshierarchie darüberliegende Aktoren, wie z.B. Professoren oder Portfoliomanager.
Was sind die Tätigkeiten?	Projektleiter haben Einblick in die Forschungstätigkeit, den Lehrbetrieb und das Forschungsprojekt an sich. In ihren Spezialgebieten sind sie häufig Ikonen, aber in die Entwicklung der Systeme nicht direkt eingebunden. Professoren haben den Blick über die Weiterentwicklung, Präsentation und Verschmelzung des Projekts.

[1] Der Ableitungsbegriff ist im Sinn der Generalisierung und Spezialisierung, wie innerhalb eines Klassendiagramms, zu lesen.

	Im Verhältnis zu anderen Stakeholdern benutzen diese das Gesamtsystem selten. Deshalb fallen diese Stakeholder in die Rolle der Probanden oder Versuchsleiter.
Abgeleitet von:	Endanwender

Tabelle 2: Stakeholder Endanwender

Endanwender	
Wer ist das genau?	Grundsätzlich schwer abzuschätzen und daher weiter zu untergliedern.
Was sind die Tätigkeiten?	Benutzung des Systems mit den dafür vorgesehen oder durch Dritte entwickelte Schnittstellen.

Tabelle 3: Stakeholder Probanden

Probanden	
Wer ist das genau?	Menschen unterschiedlicher fachlicher Herkunft und Ausbildungsgrad. Probanden nutzen nur die im weiteren festgelegten Probandenschnittstellen.
Was sind die Tätigkeiten?	Probanden werden durch die Simulationen und Welten geführt und werden mit Schnittstellen, die für Probanden vorgesehen sind, interagieren. Diese Interaktion, kann auch nicht sachgemäß ausfallen, da sie nicht kontinuierlich geschult sein werden. Aus der Interaktion der Probanden mit den Welten werden Forschungsdaten erhoben.
Abgeleitet von:	Endanwender
Risiko	Das Risiko von Probanden besteht in ihrer Aufgabe, die Simulation zu durchleben und ggf. Interaktionen nicht sachgemäß, erledigen.

Tabelle 4: Stakeholder Versuchsleiter

Versuchsleiter (Simulationsleiter, SimLeiter)	
Wer ist das genau?	Versuchsleiter sind meist Endanwender, die das Projekt (mindestens aus Schulungen) kennen. Versuchsleiter können sowohl Mitarbeiter, Doktoranten oder Studenten sein.

Was sind die Tätigkeiten?	Versuchsleiter sind Endanwender des Systems, die die Probanden durch die Simulationen und Welten führen und dabei ggf. Daten erheben. Versuchsleiter nutzen spezielle Schnittstellen, mit denen Einstellungen am System vorgenommen werden, um die Probanden optimal zu begleiten.
Abgeleitet von:	Endanwender
Risiko	Das Risiko durch den Versuchsleiter ist verringert, da dieser geschult ist. Das Restrisiko durch den Versuchsleiter kann direkt als Feedback an die Entwickler verstanden werden.

Tabelle 5: Stakeholder Entwickler

Entwickler	
Wer ist das genau?	Entwickler im universitären Umfeld sind häufig Studenten aus den Bachelorsemestern 2-5. In selteneren Fällen sind die Entwickler ausgebildete Kräfte oder Entwickler mit abgeschlossenem Studium. Es ist anzunehmen, dass die gewählten Studenten durch private Projekte oder Vorlesungen bereits Kontakt mit Teilbereichen der Materie hatten.
Was sind die Tätigkeiten?	Entwickler programmieren, verbessern und erweitern (und konzeptionieren) das System und die Software dafür.
Risiko	Das Risiko besteht darin, dass studentische Entwickler einen noch nicht praxiserprobten Wissensstand haben und das Know-how durch potenzielle häufige personelle Wechsel stark schwankt.

Tabelle 6: Stakeholder Softwarearchitekt

Systemarchitekt / Softwarearchitekt	
Wer ist das genau?	Systemarchitekten in dem Forschungsprojekt gibt es kaum, aber wenn, dann werden diese Rollen durch Studenten im Abschlusssemester oder ausgebildete Kräfte bekleidet.
Was sind die Tätigkeiten?	Systemarchitekten entwickeln die Struktur, Architektur und die Richtlinien für das Bauen und Betreiben einer Software- und Systemstruktur. Nebenläufig zur Entwicklung der Struktur gehört das Weitergeben der o.g. Informationen und das Schulen der Entwickler und Administratoren.
Abgeleitet von:	Entwickler

Risiko	Das Risiko besteht in der hohen Know-how-Bindung an eine Person. In Projekten an der Hochschule kann es vorkommen, dass mit Beendigung der Abschlussarbeit (Promotion, Masterarbeit, Bachelorarbeit, Praxisprojekt) sich diese Person aus der Reichweite des Forschungsteams bewegt und damit Know-how-Einbrüche entstehen können.

Tabelle 7: Stakeholder Administrator

Administrator	
Wer ist das genau?	Administratoren von Prototypen oder Exponenten im universitären Umfeld sind zunächst die Entwickler, des Prototyps und später Studenten aus den Bachelorsemestern 2-5. In selteneren Fällen sind die Administratoren ausgebildete Kräfte oder Entwickler mit abgeschlossenem Studium.
Was sind die Tätigkeiten?	Der Administrator dieser Systeme sorgt für die sachgemäße Konfiguration und das Updaten der Systeme, sowie Instandhaltung des Netzwerkes. Aus den bisherigen Erfahrungen sorgt der Administrator auch für das Anlegen von Geräten in OpenHAB.
Risiko	siehe Entwickler

Tabelle 8: Stakeholder Projektpartner

Projektpartner	
Wer ist das genau?	Projektpartner können sowohl Mitarbeiter von Firmen sein als auch Mitarbeiter der Hochschule, die nicht direkt mit dem Projekt zu tun haben. In dieser Masterarbeit wird das Erstellen der verbundenen Welten/Modelllandschaft als eigenes Teilprojekt betrachtet. Alle Mitarbeiter des Projekts, die sich nicht mit dem Aufbau dieser Welten beschäftigen, gehören zu den Stakeholder-Projektpartnern, da diese einen ähnlichen Wissensstand teilen.
Was sind die Tätigkeiten?	Entwickeln von Services oder Systemen, die mit diesem System interagieren müssen. Nimmt Einfluss auf die Gestaltung der Schnittstellen.
Risiko	Das Risiko (für das System) durch Projektpartner wird erst in der Integration von Systemen oder im Betrieb aktiv, da Projektpartner aufgrund von unvollständiger Dokumentation oder ganz neuen Anforderungen das Grundsystem überfordern (bis zerstören).

2.2 Soll-Situation

Vor Beginn der Entwicklung ergab sich aus Interviews mit den Projektbeteiligten (durch alle Stakeholder) folgendes Bild, was die Meta-Ansicht des zu erstellenden Systems mit Endbenutzerinteraktion skizziert.

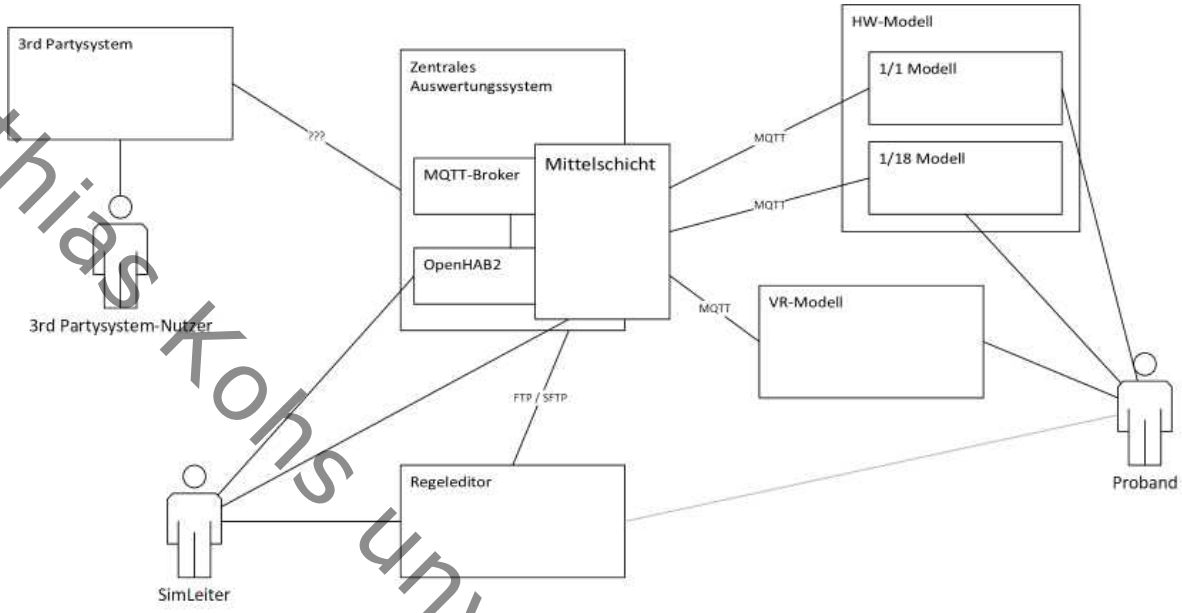


Abbildung 6: Meta-Modell des zu erstellenden Gesamtsystems (SOLL-Situation)

Den Kern des Gesamtsystems bildet das Zentrale Auswertungssystem (ZAS), dass über den Regeleditor mit Regeln versorgt wird und durch den Versuchsleiter genutzt wird. An das ZAS sind Hardware-Modelle (HW-Modelle) im Maßstab 1:1 und 1:18 angeschlossen, sowie das VR-Modell. Die Modelle sowie der Regeleditor werden in einer Untersuchung durch Probanden verwendet. Das Gesamtsystem soll 3rd-Partysysteme anbinden können, was bereits in der Meta-Ansicht als Kommunikation mit dem ZAS interpretiert wurde. Das 3rd-Partysystem wird seinerseits durch 3rd-Partysystem-Nutzer genutzt.

Da die Ausgestaltung zu dem Zeitpunkt, des Erstellens dieses Meta-Modells noch nicht bekannt war, ist die Mittelschicht zum Teil in dem ZAS und zum Teil außerhalb des ZAS gezeichnet.

2.2.1 Nicht direkt aufgezeigte Stakeholder

Endanwender leiten sich in zwei Stakeholder ab, die mit dem System direkt interagieren können: Probanden und Versuchsleiter.

Professoren und Projektleiter sind Stakeholder, die von Endanwender abgeleitet sind und für den Fortbestand des Systems unter weiterer Sub- oder Obersysteme wichtig sind. Diese Stakeholder werden während der Benutzung des Systems zu Probanden, Versuchsleitern oder selten zu Administratoren des Systems.

Entwickler und (damit auch Systemarchitekten) sind Stakeholder, die bei der Benutzung ebenfalls in die Rollen eines Probanden, Versuchsleitern oder Administrators fallen, jedoch nicht von Benutzer abgeleitet sind. Entwickler verfügen, respektive können über essentielles internes Wissen verfügen, sodass diese als klassische Endbenutzer ausgeschlossen sind.

2.3 Ist-Situation

Reduziert man nun die SOLL-Situation, wie sie in Abbildung 6 zu sehen ist, auf die aktuelle Situation, erhält man die IST-Situation, wie sie in Abbildung 7 skizziert ist.

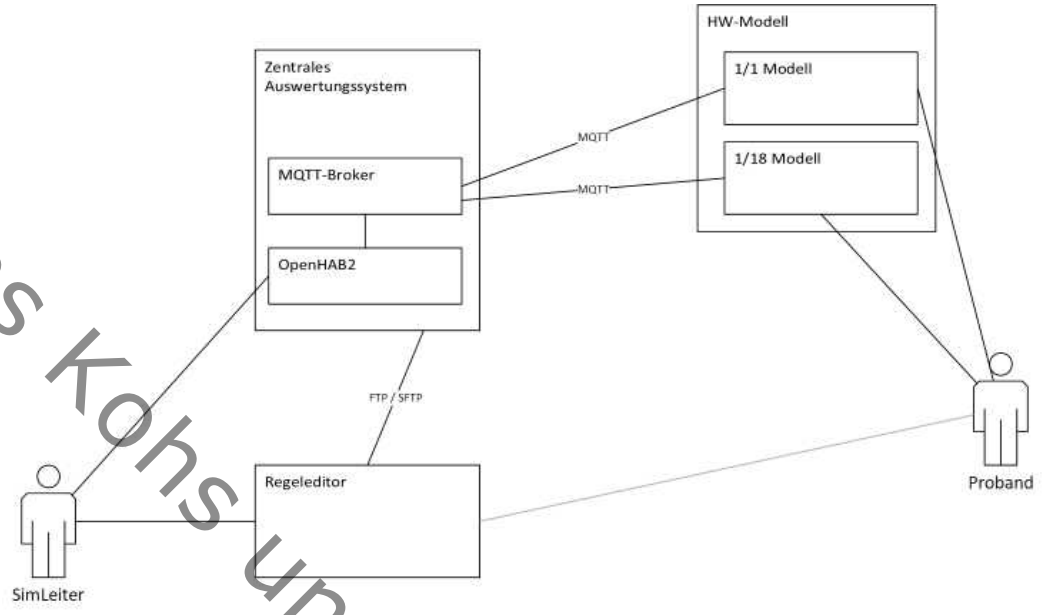


Abbildung 7: Meta-Modell der IST-Situation

Wie kurz angedeutet existiert noch keine Anbindung eines VR- oder 3D-Modells. Genauso fehlt eine Betrachtung, wie 3rd-Partysysteme angebunden werden können. Ebenfalls fehlt die Mittelschicht, die die unterschiedlichen Modelle zu Zwillingen macht und Fehlerzustände auslöst.

Die Kommunikation mit den Modellen existiert aktuell, unterliegt aber noch keiner Konvention, - an dieser Stelle entspricht die IST-Situation ebenfalls noch nicht der SOLL-Situation.

2.3.1 Im Projekt verwendete Programmiersprachen

In der IST-Situation des Projekts gibt es drei verwendete Programmiersprachen, in dieser wird jedoch nicht mehr durch aktive Entwickler abgedeckt.

Tabelle 9: Programmiersprachen im Projekt

Programmiersprache	Teilprodukt	Sprache abgedeckt
Java for Android	Regeleditor	Nein
Arduino-CC	HW-Modell auf Micro-Controller-basis	Ja
Python	HW-Modell auf Raspberry-PI-basis	Nein

2.4 Anforderungen generisches Metamodel

Aus der Betrachtung des Metamodells gegeben sich Use-Cases zwischen folgenden Aktoren:

- 1:1-Modell - Zentrales-System (MQTT Anbindung)
- 1:16-Modell - Zentrales-System (MQTT Anbindung)
- VR-Modell - Zentrales-System (MQTT Anbindung)
- 3rd Party-System - Zentrales-System
- Versuchsleiter - Zentrales-System
- Proband - 1:1-Modell (Proband - Modell)
- Proband - 1:16-Modell (Proband - Modell)
- Proband - VR-Modell (Proband - Modell)
- Versuchsleiter - Regeleditor (nicht weiter betrachtet)
- Proband - Regeleditor (nicht weiter betrachtet)
- Regeleditor - Zentrales -System (nicht weiter betrachtet)

In der folgenden Bearbeitung werden nicht alle Use-Cases betrachtet. Der Use-Case des Regeleditors wird nicht betrachtet, da es bereits in einer anderen Arbeit betrachtet wurde ((Kaiser 2018)). Die Use-Cases mit 1:1-, 1:18 oder VR-Modellen werden in Modell (MQTT Anbindung) zusammengefasst. Die Interaktion mit den Probanden lässt sich nicht in Use-Cases in inhaltsvolle Use-Case-Diagramme abbilden, da die Interaktion der Probanden von der aktuellen Ausgestaltung des Modells abhängt.

2.4.1 Modell - Zentrales System

Die Use-Cases der Modelle gegenüber dem Zentralen System sind für alle identisch. Die Schnittstelle, die alle Modelle teilen, ist die TCP/MQTT-Schnittstelle zum Broker. Es folgt eine Liste von Anforderungen zwischen dem Modell und dem Zentralen System in der Form, die durch die Anforderungsschablonen nach RUPP geregelt ist (vgl. (Rupp and SOPHISTen 2014)):

Das Zentrale Auswertungssystem muss fähig sein,

- die oben genannten Modelle (über ihre Welten) als Zwillinge zu verbinden.
- die verbundenen Modelle wieder zu trennen.
- die sich in den Modellen befindenden Geräte zu verbinden.
- die verbundenen Geräte wieder zu trennen.
- die Systemuhrzeit auszugeben.
- die Systemuhrzeit zu manipulieren.
    - die Systemuhrzeit zu setzten.
- die sich in den Modellen befindenden Geräte einzuschalten[2].

[2] sofern geräteseitig unterstützt

- die sich in den Modellen befindenden Geräte auszuschalten[3].
- manuelle Werte zu senden.
- manuelle Werte zu forcieren.
- ein geplantes konstant halten von Werten durchzusetzen.
- ein geplantes konstant halten von Werten wieder aufzuheben.
- einen geplanten, manuellen Werteverlauf durchzusetzen.

[3] sofern geräteseitig unterstützt

2.4.2 Simleiter - Zentrales-System

Um für unterschiedliche Umsetzungen offen zu sein, werden die Use-Cases des ZAS inklusive der Mittelschicht wie ein System betrachtet. Die Darstellung sieht dann wie folgt aus (Abbildung 8):

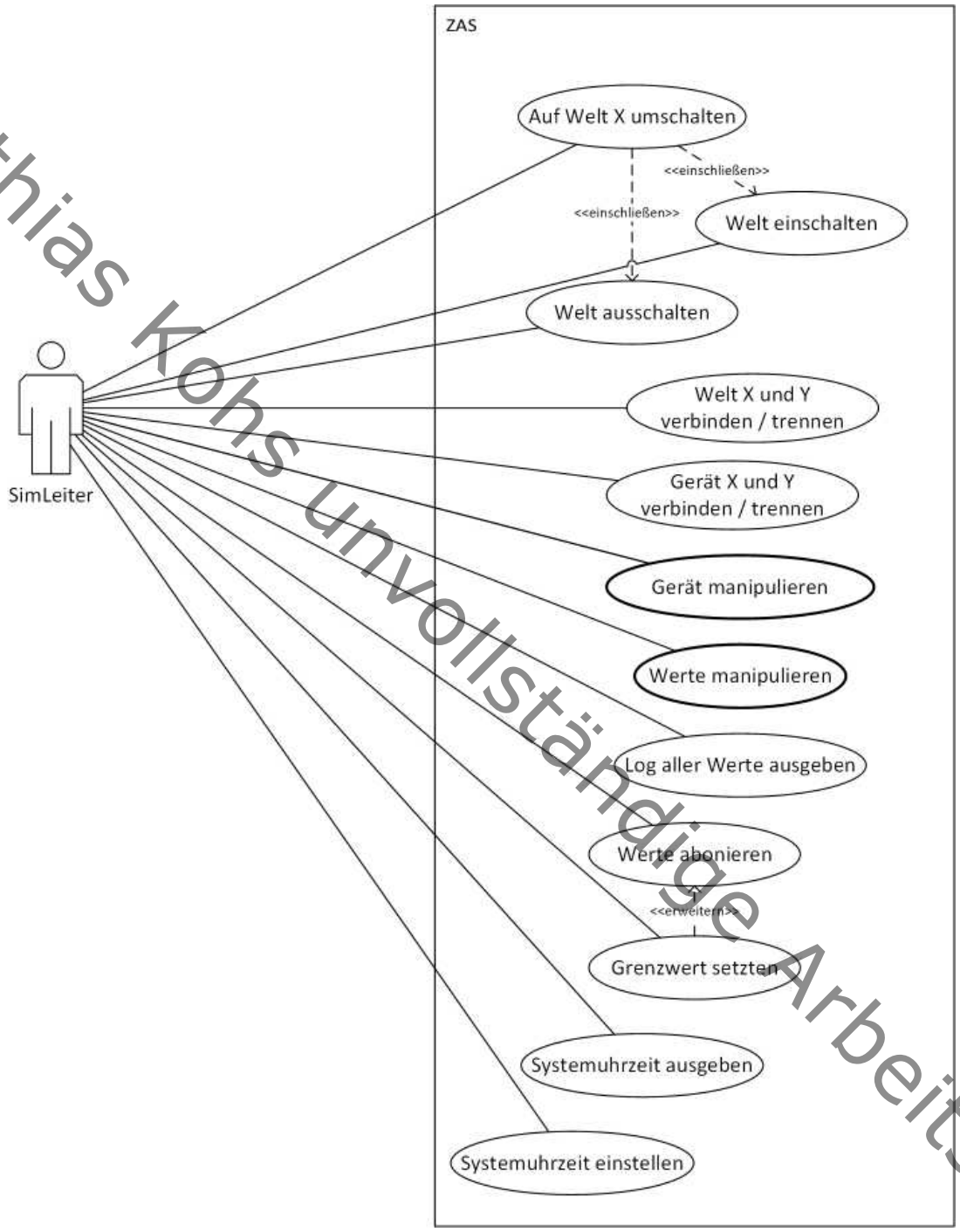


Abbildung 8: Use Case-Diagramm SimLeiter nutzt ZAS

Das Ergebnis aus dem Use Case-Diagramm lässt sich (durch Anforderungsschablonen, vgl. (Rupp and SOPHISTen 2014)) in abzählbare Anforderungen überführen:

Das ZAS muss dem Versuchsleiter die Möglichkeit geben,

- eine Modell-Welt einzuschalten.
- eine Modell-Welt auszuschalten.
- auf Welt X umzuschalten.
- die Welt X und die Welt Y zu verbinden.
- die Welt X und die Welt Y zu trennen.
- das Gerät X und das Gerät Y zu verbinden.
- das Gerät X und das Gerät Y zu trennen.
- das Ausschalten von Geräten zu simulieren.
- das Einschalten von Geräten zu simulieren.
- Geräte mit dem ZAS zu verbinden.
- Geräte mit dem ZAS zu trennen.
- Werte zu manipulieren.
    - einen manuellen Wert zu forcieren.
    - manuell Werte zu senden.
    - einen manuellen Wert mit zeitlicher Beschränkung Beginn und Ende zu planen.
    - einen geplanten Wert zu stornieren.
    - (einen manuellen Wert X Mal in einem Zeitraum senden zu lassen).
    - einen manuellen Wert zu deaktivieren.
- ein Log aller Werte auszugeben.
- Topics zu abonnieren.
- bei abonnierten Topics Grenzwerte einzugeben.
- die Systemuhrzeit einzustellen.
- die Systemuhrzeit auszugeben.

Das Manipulieren von Geräten (siehe Anforderungsliste) ist weiter konkretisierbar, wie in Abbildung 9 zu sehen ist.

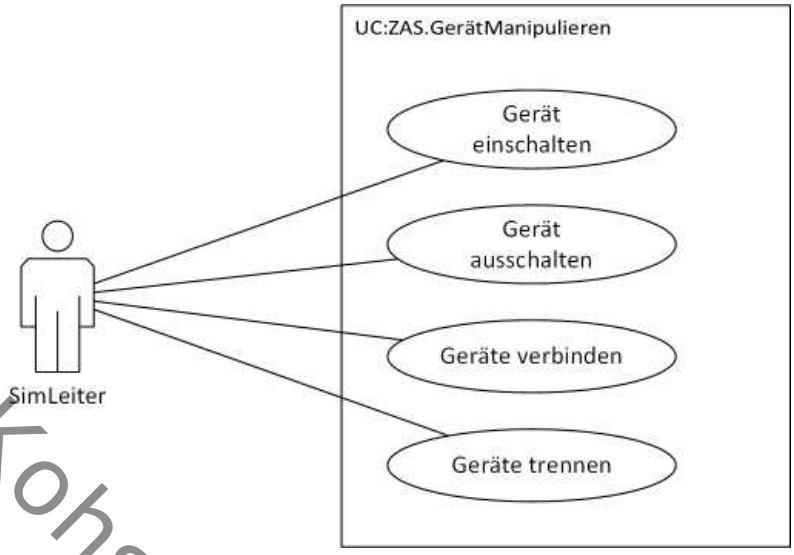


Abbildung 9: Use-Case-Diagramm SimLeiter GerätManipulieren

Das Manipulieren von Werten ist weiter konkretisierbar, wie in Abbildung 9 zu sehen ist.

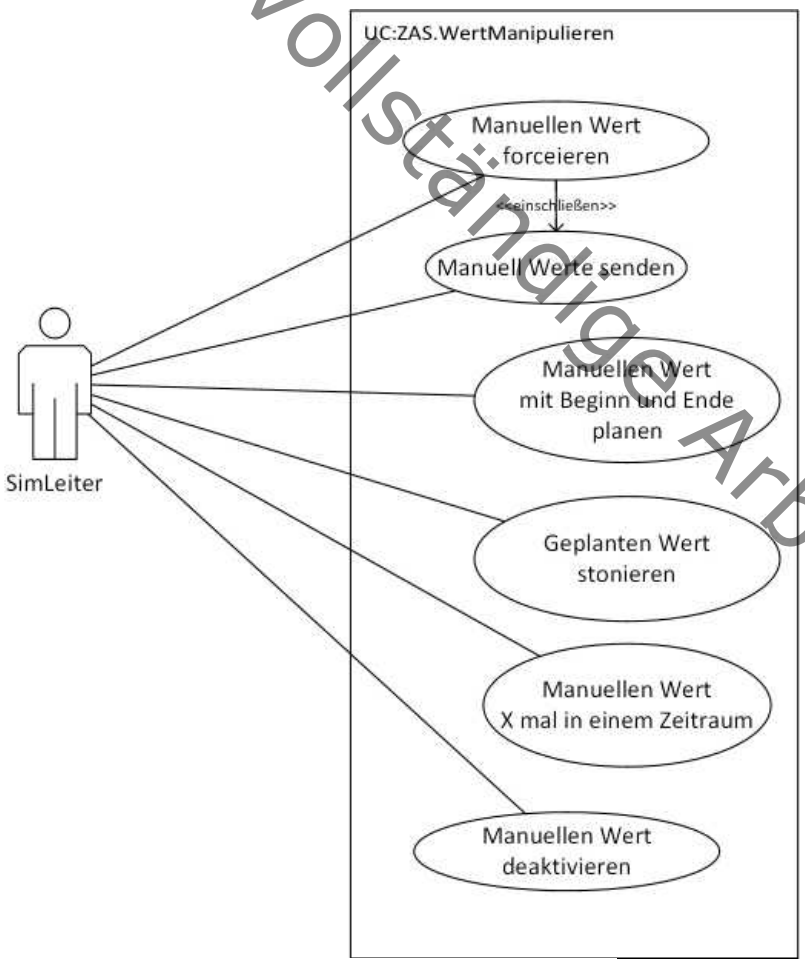


Abbildung 10: Use-Case-Diagramm SimLeiter WerteManipulieren

2.4.3 3rd-Party-System - Zentrales-System

Da noch kein System von einem Drittanbieter oder Projektpartner vorliegt, kann kein sinnvolles Use-Case-Diagramm gezeichnet werden. Jedoch gibt es Anforderungen aus dem Projektteam, die von Mitwissenschaftlern und Projektleitern gewünscht wurden:

Das Zentrale-System sollte 3rd-Party-Systemen die Möglichkeit bieten

- alle Wert-Manipulationen durchzuführen.
- alle Geräte-Manipulationen durchzuführen.

Indirekt ergibt sich die Anforderung für den weiteren Betrieb:

Das Zentrale-System wird 3rd-Party-Systemen die Möglichkeit bieten

- sich zu authentifizieren.
- autorisierte Handlungen durchzuführen.

2.4.4 Administrator - Zentrales-System

Der Administrator spielt in der Entwicklung, während dieser Arbeit, noch keine Rolle, jedoch sollten die generellen Anforderungen dieses Stakeholders betrachtet werden, um abzuschätzen, ob das entwickelte System wartbar bleibt.

Das Zentrale-System wird dem Administrator die Möglichkeit bieten

- andere Systeme für den Zugang zu autorisieren.
- Zugänge mit Authentifizierungen zu verbinden.
- Geräte in die Modelllandschaft hinzuzufügen.
- Geräte aus der Modelllandschaft zu entfernen.

2.4.5 Proband - Modell

Die Use-Cases für die Interaction zwischen Probanden und Modell werden in dieser Arbeit nicht genauer betrachtet, da die Untersuchung ein Teil der Workshops ist, die sich dieser Arbeit anschließen. Diese Arbeit bietet mit dem Bau der Mittelschicht und der MQTT-Unity3D-Anbindung ein Teil der Werkzeuge, um diese Untersuchungen in den Workshops durchzuführen (vgl. (Jacobas et al. 2016)).

Generell lassen sich einfache FA ableiten, die das Modell als generisches Interaktionsmedium beschreibt.

Das Modell muss dem Probanden die Möglichkeit bieten,

- einen Wert durch Interaktion mit der Modelllandschaft zu verändern.
- Werteänderungen durch Veränderungen am Modell wahrzunehmen.

2.5 Nichtfunktionale Anforderungen

Neben den Funktionalen Anforderungen, die einheitlich definiert sind, gibt es viele Ansätze die Nicht-Funktionalen Anforderungen (NFA) zu definieren. In dieser Arbeit wird die Definition der SOPHISTen für NFAs angewandt (vgl. Kap 12f (Rupp and SOPHISTen 2014)). Nicht-funktionale Anforderungen sind demnach „alle Anforderungen, die nicht … funktional sind“ (Zitat Kap 12.1 (Rupp and SOPHISTen 2014)). Diese werden in sechs Kategorien unterteilt, siehe Abbildung 11: In

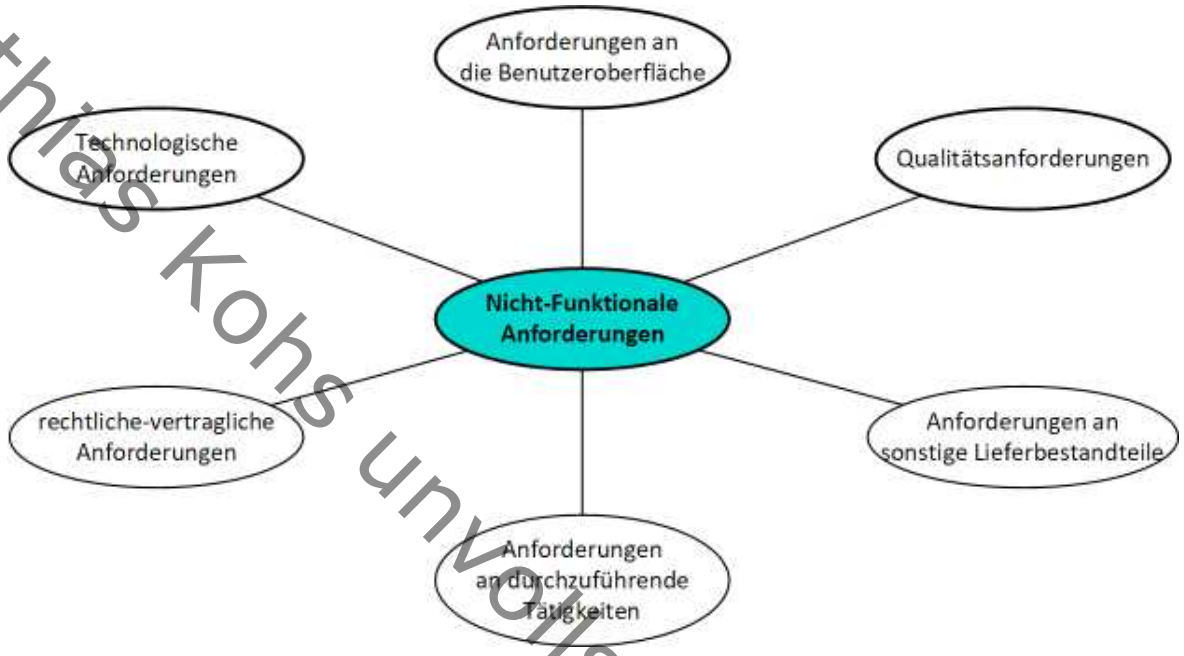


Abbildung 11: Kategorien für nicht-funktionale Anforderungen nach den 'SOPHISTen'

dieser Arbeit gehe ich nur auf zwei der Kategorien ein: Technologische Anforderungen und Qualitätsanforderungen.

Anforderungen an die Benutzeroberfläche sind Anforderungen, die das Erscheinungsbild und die menschliche Interaktion mit den Interfaces des Systems näher beschreiben (vgl. K.12.5 (Rupp and SOPHISTen 2014)). In dieser Arbeit, werden auch Benutzeroberflächen geschaffen, jedoch soll dies nicht Fokus der wissenschaftlichen Arbeit sein. Daher wird auf Anforderungen zur Benutzeroberfläche verzichtet.

In den *Anforderungen an sonstige Lieferbestandteile* fordern Kunden Ersatzhardware, Schulungsunterlagen, Dokumentation oder „andere Dinge“, die nicht direkt zu dem Produkt gehören, welche aber notwendig sind, um den Betrieb des Produkts zu gewährleisten. (vgl. K.12.6 (Rupp and SOPHISTen 2014)). Parallel zum Anfertigen dieser Arbeit wird eine Dokumentation aufgebaut, um entstehende Komponenten weiter zu verwenden. Aus zeitlichen Gründen liegt dieser Punkt nicht im Fokus dieser Arbeit.

Rechtliche-vertragliche Anforderungen sind Anforderungen, in den sich die Vertragspartner entsprechende Rechtssicherheit geben und z.B. Regelungen über die Zahlungsphasen oder Milestones treffen. (vgl. K.12.8 (Rupp and SOPHISTen 2014)). Abschätzung dieser Anforderungen wird nicht Fokus dieser Arbeit sein, da diese Arbeit in einem Forschungsprojekt eingebettet ist.

Anforderungen an durchzuführende Tätigkeiten sind Anforderungen, die die Art der Entwicklung, der Abnahme oder andere Tätigkeiten des Produkt-Life Cycle näher beschreiben (vgl.

K.12.7 (Rupp and SOPHISTen 2014)). In diesem konkreten Fall werden die Mitarbeiter in der Benutzung und zum Teil in der Weiterentwicklung der entstehenden Komponenten geschult, jedoch soll dieser Punkt nicht wissenschaftlich betrachtet werden, daher verzichte ich auf die Erfassung dieser NFA.

2.5.1 Qualitätsanforderungen

Die Qualitätsanforderungen wurden durch nicht formale Interviews mit dem Projektteam erhoben und in die Form der Anforderungen gebracht. Es folgt eine Liste von Qualitätsanforderungen nach Schablonen von (Rupp and SOPHISTen 2014), die durch das Team bestätigt wurde.

Das Gesamtsystem sollte so gestaltet sein,

- dass es anpassbar ist.
  - dass neue Geräte in den Modellen hinzugefügt werden können.
  - dass die Oberfläche für den Versuchsleiter (durch Entwickler) angepasst werden kann.
- dass die Beherrschbarkeit für weitere Entwickler in einem Monat erreichbar ist.
- dass die Beherrschbarkeit des Gesamtsystems trotz zunehmender Nutzung erhalten bleibt.
  - Der Hauptwachstumspunkt, der zur Nicht-Beherrschung führen kann, ist die Datenmenge, die bei Durchläufen erzeugt wird.
- dass der Nachbau durch andere Forschungsgruppen (ggf. ohne Budget für proprietäre Software) möglich ist.
  - Es wird nur offen verfügbare Software verwendet.
  - Es wird keine Sonderhardware durch die Mitteschicht gefordert.
  - Das Reposetory des Mittelschicht-Quellcodes wird öffentlich gemacht.
- dass eine Einarbeitung in die Nutzung des Gesamtsystems innerhalb von 1.5 Tagen erreicht werden kann.
- dass eine schnelle Einarbeitung in die Nutzung von Komponenten gewährleistet ist.

Die MQTT-Unity3D-Anbindung muss, so gestaltet sein, dass ein Entwickler nach einer ca. dreistündigen Schulung über die MQTT-Unity3D Anbindung

- auf dem MQTT-Protokoll Topics abonnieren und Inhalte empfangen kann.
- auf dem MQTT-Protokoll Topics absetzen und eigene Inhalte senden kann.

2.5.2 Technologische Anforderungen

Technologische Anforderungen beschreiben Anforderungen zur technologischen Einbettung des Systems in seine Umwelt und Vorgaben von Lösungen zur Realisierung. (vgl. Kap.12.3 (Rupp and SOPHISTen 2014)). Diese Anforderungen haben damit großen Einfluss auf die Art der Realisierung oder der Wahl weiterer Technologie für die Lösung der Anforderungen. Der größte Einflussfaktor für die Technologischen Anforderungen ist die IT-Ausstattung des Forschungsprojekts.

2.5.2.1 3D- und VR-Umsetzung

Es folgen Anforderungen nach den Anforderungsschablonen (siehe Kap. 2 Anforderungsanalyse):

Die 3D Anbindung (MQTT-Unity3D) muss so gestaltet sein, dass die Anwendung in einfacher Detaillierung ohne VR auf einem Laptop mit der Hardwareausstattung H1 betrieben werden kann.

Die VR Anbindung (MQTT-Unity3D) muss so gestaltet sein, dass die Anwendung in einfacher Detaillierung mit VR auf einem PC mit der Hardwareausstattung H2 betrieben werden kann.

Die VR Anbindung (MQTT-Unity3D) muss so gestaltet sein, dass die Anwendung mit der HTC-Vive-2016 darstellbar ist.

Die VR Anbindung (MQTT-Unity3D) soll so gestaltet sein, dass die Anwendung unter Verwendung von Steam-VR funktioniert.

Der MQTT-Unity3D Anbindung muss so gestaltet sein,

- dass ein Build erstellt werden kann, der auf folgenden Betriebssystemen OS1 betrieben werden kann.
- dass ein Build ohne die Mittelschicht, bei einem MQTT-Broker Topics abonnieren und dessen Inhalte empfangen kann.
- dass ein Build ohne die Mittelschicht, bei einem MQTT-Broker Topics absetzten und eigene Inhalte senden kann.

Die MQTT-Unity3D Anbindung sollte so gestaltet sein,

- dass ein Build erstellt werden kann, der auf folgenden Betriebssystemen OS2 betrieben werden kann.

Tabelle 10: Hardware H1 (Entwicklungsumgebung)

Hardware H1	
Typ:	Laptop
CPU:	Intel Core i7 7700HQ
RAM:	16GB DDR3
GPU:	Nvidia GTX 1050

Tabelle 11: Hardware H2 (VR Workstation)

Hardware H2	
Typ:	Workstation
CPU:	Intel Core i7 7700K
RAM:	32GB DDR3
GPU:	Nvidia GTX 1080 TI
VR-Device:	HTC Vive

Tabelle 12: Hardware H3 (Raspberry Pi)

Hardware H3	Raspberry Pi 3B+
Typ:	Einplatinencomputer
CPU:	Broadcom BCM2837B0, Cortex-A53
RAM:	1GB LPDDR2

Tabelle 13: Betriebssystem OS1 Windows

Betriebssystem OS1	
Microsoft Windows 10:	*Windows 10 Pro*
	Windows 10 Enterprise
	Windows 10 Education
	Windows 10 Pro Education

Tabelle 14: Betriebssystem OS2 Linux

Betriebssystem OS2	
Linux Distributionen:	Debian 9
	Ubuntu 18 LTS

2.5.2.2 Mittelschicht Umsetzung

Die Mittelschicht muss so umgesetzt sein,

- dass diese unter der Belastung von einem 1:18-Modell-Grundstück auf der Hardware H3 betrieben werden kann.
- dass diese unter der Belastung von drei 1:18-Modell Grundstücken und einer VR-Anwendung betrieben werden kann.
- dass diese unter der Belastung von drei 1:1-Modellen / Exponaten und einer VR-Anwendung betrieben werden kann.
- dass diese in einem Verbund von Rechnern zusammen mit OpenHAB2 koexistieren kann.
- dass diese mit MQTT-Brokern über MQTT/TCP kommunizieren kann.

2.5.2.3 Anforderungen an sonstige Lieferbestandteile

Von externen Anbietern oder Zulieferern kommende Komponenten des Systems müssen so gestaltet sein,

- dass diese für bis zu 100€ erwerbbar sind.
- dass der Code vor dem Buildvorgang einsehbar ist.
- dass die Komponenten noch eine Supportzeit von mindestens 3 Jahren haben
- dass die Herkunft des Codes bekannt ist.

2.6 Prüfen der Anforderungen

Für das Prüfen und Evaluieren des Systems werden alle in den Kapiteln erwähnten Anforderungen in eine Liste überführt, die bei der Evaluation der Ergebnisse eingesetzt werden soll. Diese Liste wird eingeteilt nach Testbarkeit und dem thematischen Bezug.

Zur nachträglichen Prüfung der Nicht-funktionalen Anforderungen, werden diese Anforderungen noch in den Bezug zu der ISO /IEC 25000 Norm für *Software Engineering – Software Product Quality Requirements and Evaluation* gestellt. Diese Norm unterscheidet sechs Merkmale der Dienstgüte/Qualitätsanforderungen: Änderbarkeit, Benutzbarkeit, Effizienz, Funktionalität, Zuverlässigkeit und Übertragbarkeit.

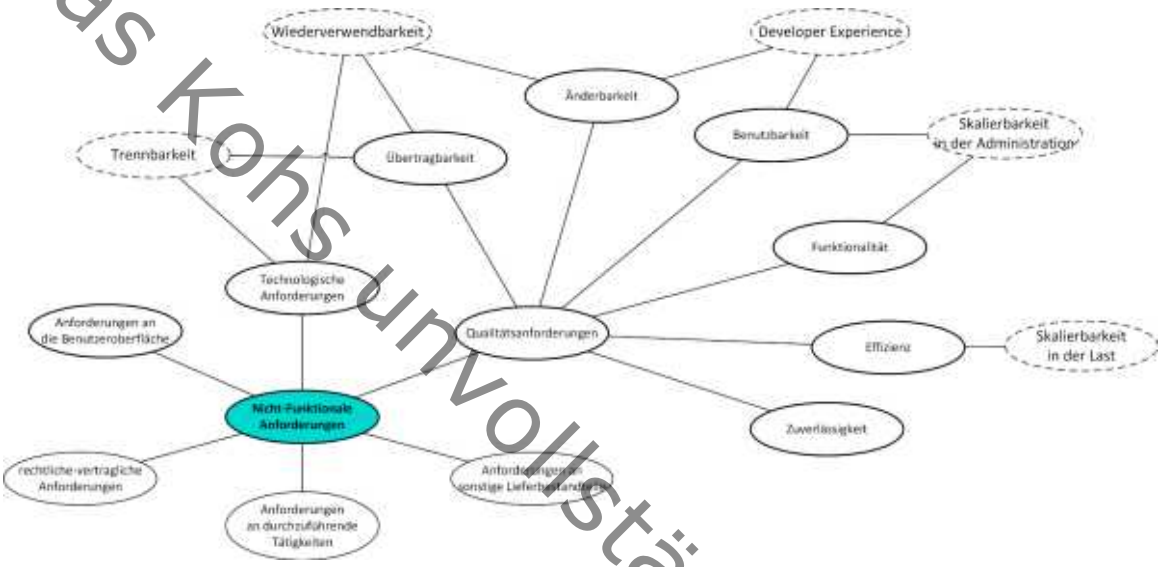


Abbildung 12: Mindmap mit Auswahl vernetzer Anforderungsmerkmale und Kategorien

Wie in Abbildung 12 zu sehen, sorgt die Verschneidung von NFAs mit den Merkmalen der Dienstgüte zusammen mit ausgewählten Begriffen dieser Arbeit zu einer Graphenstruktur. Die Graphenstruktur macht es nicht möglich, die Punkte in Kapitel-Unterkapitel (also Baumstruktur) zu erfassen, ohne schwer zu überschauende Doppelungen (und ggf. Mehrdeutigkeiten) zu erzeugen. Die Bewertung der Merkmale wurde im Kapitel 1.3 Herausforderungen mit Forschungsprototypen abgehandelt.

Es folgt die Zuordnung der Merkmale der Dienstgüte/Qualitätsanforderungen mit den entsprechenden konkreten Anforderungen, aus der Anforderungsliste (siehe Anhang: 7.2-Anforderungsliste).

Merkmal	**Anforderungs-Nr.**
Funktionalität	1-45, 50-53, 57, 58,61
Änderbarkeit	59, 60, 63-66
Benutzbarkeit	63, 64, 66, 67, 69, 70
Effizienz	46, 47, 54-56
Übertragbarkeit	48-57, 59-64.66-68
Zuverlässigkeit	47, 61, 62

2.7 Software Designprinzipien

Software Designprinzipien sind Regeln und Verhaltensmuster für den Entwickler bzw. den Architekten, die wie Design Pattern dabei helfen Herausforderungen in der Architektur der Software oder dem Schreiben von Code zu vereinfachen. Im Unterschied zu Design Pattern, die Probleme auf bekannte Strukturen zurückführen, geben die Designprinzipien vor, wie sich der Entwickler oder Architekt verhalten soll. In dieser Arbeit werden die SOLID-Prinzipien angewandt (vgl. Teil 3 (Martin 2017)).

Das direkte Ziel dieser Prinzipien ist es, mittelschichtige Softwarestrukturen zu erzeugen, die

- Modifikationen erlauben (SOLID-Z1)
- leicht nachzuvollziehen sind (SOLID-Z2)
- die Basis der Komponenten zu bilden, die in vielen Softwaresystemen eingesetzt werden können. (SOLID-Z3)

Tabelle 15: SOLID-Ziele zu Merkmalen der Qualitätsanforderungen (V: voll-verbunden, T: teil-verbunden)

	QAF: A	QAF: B	QAF: E	QAF: F	QAF: U	QAF: Z
SOLID-Z1	V			T	V	
SOLID-Z2	V	T		T	V	T
SOLID-Z3	V			T	V	

Die Ziele der SOLID-Prinzipien decken sich mit vielen der Nicht-Funktionalen Anforderungen und sind damit ideal einsetzbar. Tabelle 15 stellt SOLID-Ziele den Qualitätsanforderungen gegenüber, es wird ein V (voll-verbunden) vergeben, wenn die Verbindung direkt ersichtlich ist und ein T (teil-verbunden), wenn sie nicht ersichtlich ist oder nur zum Teil erfüllt wird.

SOLID-Z2 und QAF:B (Benutzbarkeit) sind nur teil-verbunden, da dieser Punkt die Benutzbarkeit für den Entwickler verbessert, aber nicht notwendigerweise für den Benutzer.

SOLID-Z1 bis Z3 sind mit QAF:F (Funktionalität) und SOLID-Z2 ist mit QAF:Z (Zuverlässigkeit) nur teil-verbunden, da sich durch die Prinzipien, bei Auftauchen von Abweichungen schneller Verbesserungen implementieren oder entwickeln lassen und Fehler reduziert werden, dies aber nicht direkt erfolgt.

Für diese Arbeit werden die angepassten SOLID Prinzipien von Robert C. Martin aus dem Buch Clean Architecture oder dessen Synthese angewendet (Teil3 (Martin 2017)). Diese sollen dabei helfen, Entwicklungsschritte zu verstehen, und bilden einen Teil der Grundlage für die Erfüllung der Qualitätsanforderungen, auch über die erste Inbetriebnahme hinaus.

2.7.1 SOLID-SRP: Single Responsibility-Prinzip

Ein Modul sollte für nur einen Akteur verantwortlich sein.

Dieses Prinzip sorgt dafür, dass es nie mehr als einen Grund gibt, eine Klasse zu modifizieren. Das wiederum sorgt dafür, dass Entwickler gut nebenläufig arbeiten können, ohne Probleme doppelt zu lösen oder häufige Merge[4]-Konflikte zu haben (vgl. Kapitel7 (Martin 2017)).

2.7.2 SOLID-OCP: Open Closed-Prinzip

Das Verhalten einer Software sollte erweiterbar sein, ohne dass die modifiziert werden muss.

Dieses Prinzip sorgt z.B. dafür, dass die Softwarearchitektur so gebaut wird, dass die Implementierung neuer Funktionalitäten z.B. eine neue Klasse mit einer neuen Funktion fordert, aber alter Code nicht oder fast nicht angepasst werden muss. Dadurch lassen sich Fehler vermeiden, die bei der Wartung oder Anpassung entstehen.

Das kann dadurch erreicht werden, dass man das System in Komponenten unterteilt und diese dann in einer Abhängigkeitshierarchie anordnet, die Komponenten höherer Ebenen vor Änderungen an untergeordneten Komponenten schützt (vgl. Kapitel 8 (Martin 2017)).

2.7.3 SOLID-LSP: Das Liskov'sche Substitutionsprinzip

Im Original von Barbara Liskov:

Was hier erreicht werden sollte, ist etwas wie die folgende Substitutionseigenschaft: Wenn für jedes Objekt o1 von Typ S ein Objekt o2 von Typ T existiert, sodass für alle Programme P, die in T definiert sind, das Verhalten von P unverändert bleibt, wenn o1 für o2 substituiert wird, dann ist S ein Subtyp von T.

Dieses Prinzip kommt aus einer Zeit (Liskov 1988), in der Software Architecture, in der Form wie es heute existiert, noch nicht angedacht wurde - dennoch gilt es weiterhin. Es kann um Interfaces und die Betrachtung auf Modulebene erweitert werden. Dies sorgt für die Entkoppelung von Systemen und damit zur verbesserten Anpassbar- und Erweiterbarkeit (vgl. Kapitel9 (Martin 2017)).

2.7.4 SOLID-ISP: Interface Segregation-Prinzip

Vermeide Abhängigkeiten von Klassen oder Modulen, die mehr Features enthalten als notwendig. Trenne Interfaces (Schnittstellen) oder Klassen in minimale Features auf, um geringe Abhängigkeit zu erzeugen.

Wenn eine Transitive Abhängigkeit von drei Systemen (System S, Framework F, Datenbank D) vorliegt und die Interfaces umfangreicher sind als für S notwendig, müssen diese trotzdem erfüllt werden. Modifikationen an Datenbank D könnten dann durchaus ein Anpassen und dadurch erneutes Deployen von Framework F erzwingen. Dadurch, dass System S das ganze Interface von F implementieren musste, ist ggf. auch ein neu Deployen von S notwendig. Dies gilt auch wenn D Funktionen anpasst, die durch S nicht genutzt werden. Die zu großen Interface-Definitionen sorgen für Abhängigkeit, auch im Fehlerfall (vgl. Kapitel10 (Martin 2017)).

[4] Der Merge im Sinn des Zusammenführens von Code und Quelldateien, z.B. mit einem Code-Verwaltungstools wie GIT oder SVN.

2.7.5 SOLID-DIP: Dependency Inversion-Prinzip

Referenzieren Sie keine flüchtigen konkreten Klassen.

Nutzen Sie keine Ableitung von flüchtigen konkreten Klassen.

Überschreiben Sie keine konkreten Funktionen.

Erwähnen Sie konkrete und flüchtige Elemente zu keinem Zeitpunkt namentlich.

Flüchtige Klassen sind Klassen, die sich schnell ändern können und quasi volatil sind. Eine Implementierung oder eine Architektur, die abhängig von flüchtigen Klassen ist, muss bei jeder Änderung reagieren, wie die intuitive Variante in Abbildung 13. Die Lösung ist, Interfaces zu implementieren und Abstract Factorys zu verwenden, die man definieren kann. Statt dass die Anwendung von der konkreten volatilen Klasse abhängig ist, ist die konkrete Factory von der dem Factory-Interface und die konkrete volatile Klasse von dem jeweiligen Interface abhängig.

In Abbildung 14 sieht man die Pfeilrichtung von unten nach oben abgebildet, da konkrete Klassen die Interfaces implementieren. Die Pfeile zeigen zudem auch die Abhängigkeit an, die nun andersherum als in Abbildung 13 ist; daher kommt der Name des Prinzips „Dependency Inversion“ (vgl. Kapitel 11 (Martin 2017)).

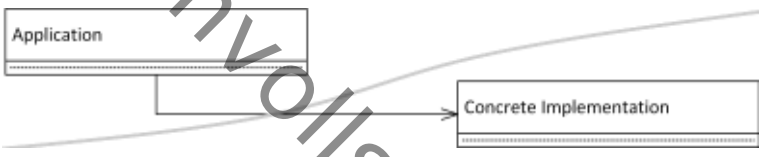


Abbildung 13: Klassendiagramm mit voller Abhängigkeit der Application von der konkreten Implementierung einer Klasse

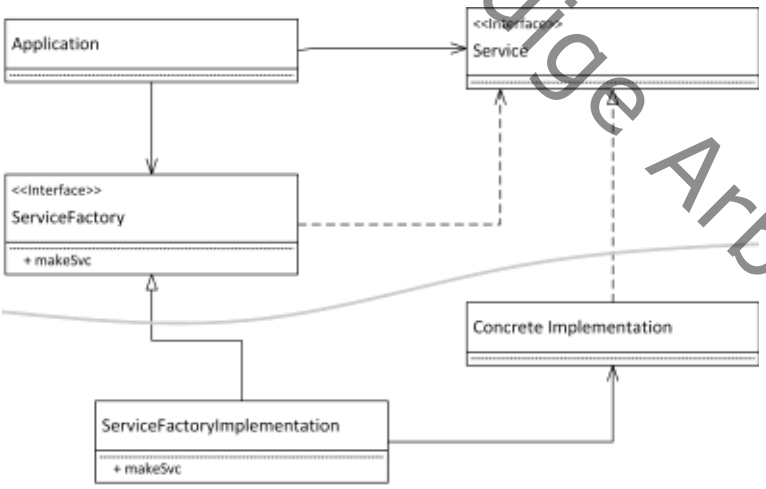


Abbildung 14: Klassendiagramm mit Abhängigkeitsumkehr durch Anwendung des DesignPatterns Abstract Factory (Dependency-Inversion)

3 Technologie Wahl

Die Technologie Wahl wird auf Basis der Anforderungen und Ziele des Projekts getroffen, dabei spielt Kommunikation zwischen den Komponenten eine wichtige Rolle.

Die Standard Kommunikation zwischen den Modellen und dem ZAS geschieht über MQTT/TCP. MQTT/TCP ist eine MQTT Variante, die direkt auf der TCP-Ebene aufsetzt und kein weiteres Protokoll als Zwischenschicht nutzt.

3.1 Modelllandschaft

Aus MQTT-Sicht sind die Systeme in der Modelllandschaft MQTT-Clients. Die MQTT-Clients halten die Kommunikation mit dem MQTT-Broker und agieren, wenn sich Werte der abonnierten Topics ändern. Ebenfalls senden MQTT-Clients Änderungen der eigenen Werte, die durch Sensorik ermittelt werden, an die entsprechenden eigenen Topics.

Da die Masterarbeit sich nicht auf den Bereich der Auswertung der Sensordaten bzw. das Reagieren auf die Topics mit entsprechender Aktorik fokussiert, werden die im SMILE-Projekt festgelegten Technologien verwendet.

Zu der Technologie gehören:

- Raspberry-PI-Boards über WLAN oder LAN an das Netzwerk angeschlossen
- Arduino-Boards über WLAN an das Netzwerk angeschlossen
- ESP32-Boards über WLAN an das Netzwerk angeschlossen
- Proprietäre Geräte wie Smarte-Lampen oder Steckdosen

3.2 3D- und VR-Welt

Für die digitale Welt werden zwei Varianten unterstützt. Die klassische 3D-Welt mit Ausgabe auf einem Bildschirm und die VR-Welt. Die 3D-Welt nutzt die Game Engine und die MQTT-Bindung. Die VR-Welt nutzt dieselbe Technologie wie die 3D-Welt mit Hinzunahme des VR-Toolkits.

3.2.1 Game Engine

Generell kommen diverse Game Engines in Frage, dazu gehören: Unity3D (“Unity3D WebPage Main” 2019), Unreal Engine (“UnrealEngine WebPage Main” 2019), CryEngine (“No CryEngine WebPage Main” 2019), Blender (“Unity3D WebPage Main” 2019), Godot Engine (“Godot Engine WebPage Main” 2019), Lumberyard Engine (“Godot Engine WebPage Main” 2019).

Im Forschungsteam wurde bereits Unity3D mehrfach verwendet und erprobt. Die Wahl von Unity3D liegt nah, da damit Ressourcen und Know-how aus dem Team verwendet werden können (vgl. ((ITOM/FH-Aachen) 2019; “MakeIt(True)” 2019)).

Es werden die folgenden Versionen untersucht:

Unity 2017.4 LTS
Diese Version wurde ausgewählt, da sie zum Beginn der Recherchearbeiten die letzte Version für den Long Time Support (LTS) ist. Der LTS geht über zwei Jahre ab dem Release-Datum der Version

2018.1 (ungefähr ende Q1 2018) (vgl. (Unity 2019a), (Unity 2018d)). Diese Version verarbeitet das die Programmierung in C# noch mit einem Subset von .Net2.0 (vgl. (Unity 2018c), (Unity 2019a)).

Unity 2018.1
Diese Version wurde ausgewählt, da sie Unterstützung für C#6 und .Net4.x bietet (vgl. (Unity 2018a)) und sich damit stark von Unity 2017.4 unterscheidet. Weitere Informationen zu den Änderungen in dieser Version sind in dem Roadmap-Archiv (Unity 2019g) zu finden.

Unity 2018.3
Diese Version wurde ausgewählt, da sie zum Zeitpunkt der Recherche für das Exposé die aktuelle Unity Release-Version darstellte. Im Unterschied zu Unity2018.1 bietet sie Unterstützung für C#7 und den neuen Compiler Roslyn (vgl.(Unity 2018b)).

Unity 2019.1
Diese Version wurde nach Beginn der Recherche dazu ausgewählt, da sie zum Recherche Zeitpunkt noch nicht released war, aber zum Zeitpunkt der Masterarbeit die aktuellste Unity Release-Version darstellt. Wie Unity 2018.3 werden die C# Compiler mcs für C#4 und Roslyn für c#7 genutzt (Unity 2019d). Neben diesem Punkt, der identisch ist, sind aber noch weitere Punkte hinzugekommen, weiterführende Informationen in den Release Notes (Unity 2019b).

3.2.2 VR-Toolkit

Die erste Wahl für die VR Anbindung fällt auf das VRKIT Virtual Reality Toolkit (VRTK)(Ltd 2016), da dieses Toolkit für verschiedene VR-Brillen gut funktioniert, das am besten bewertete Toolkit im Untiy3D-AssetStore(“VR KIT Assetstore Unity3D” 2019) ist und im ITOM bereits erfolgreich damit gearbeitet wurde (vgl. ((ITOM/FH-Aachen) 2019; “MakeIt(True)” 2019)).

Bei der genauen Betrachtung für die aktuelle Version VRTK v3.3 findet sich die Information, dass diese Version nicht mit Steam-VR v2 zusammen funktioniert (vgl. (Ltd 2016), ((VRTK-Forum) 2019), Abbildung 51, Abbildung 53). Steam-VR ist eine Schnittstelle der Steam Plattform zum Anbinden von VR-Brillen. Im ITOM wurde zum Anbinden von VR-Brillen bisher nur Steam-VR genutzt. Damit ist dies eine „Soll-Anforderung“. Ohne eine Schnittstelle wie Steam-VR ist die Anbindung an die VR-Brille erschwert.

Die *direkten Alternativen* sind,

a. VRTK v3.x mit Steam-VR v1.x zu nutzen
b. neues Steam-VR v2.x ohne VRTK zu nutzen
c. Beta VRTK v4.x mit Steam-VR v2.x zu nutzen

Bei weiterer Suche nach VR-Kits ergeben sich schnell weitere Kits und Lösungsansätze, wie das NewtonVR (Tomorrow Today 2016), das sehr ähnlich zu dem VRTK oder dem Steam-VR ist, aber mehr Wert auf Kollisionserkennung legt; oder wie das Bridge XR (Parkerhill Reality 2018a) (Parkerhill Reality 2018b), was mehr einen Manager zwischen den SDKs darstellt.

Viele Kits für Unity3D existieren und lösen ähnliche Herausforderungen. Einer der Herausforderungen ist der XPlattform-Gedanke, bei dem mit einer Implementierung vieler Geräte unterstützt werden. Denn aktuell gibt es mehr als nur einen Hersteller und mehr als nur eine API bzw. ein SDK.

Es folgt die Tabelle 16 mit einer Auswahl von VR-Brillen und SDKs (die nicht den Anspruch der Vollständigkeit hat):

Tabelle 16: Auswahl von VR-Brillen und SDKs

	Produkt	SDK	Referenz
Oculus VR	The Rift	Oculus SDK /Steam VR	(Oculus 2019)
HTC	Vive	Steam-VR	(Vive 2019)
Samsung	Gear VR	Gear VR SDK	(Samsung 2019b)
Google	Daydream View	Daydream SDK	(Google 2019b)
Samsung	Odyssey	Microsoft MR / Steam-VR	(Samsung 2019a), (Microsoft 2017)
Acer	AH100	Microsoft MR / Steam-VR	(Microsoft 2017)
Lenovo	Explorer	Microsoft MR / Steam-VR	(Microsoft 2017)
Dell	Visor	Microsoft MR / Steam-VR	(Microsoft 2017)
HP	VR1000	Microsoft MR / Steam-VR	(Microsoft 2017)
Asus	HC102	Microsoft MR / Steam-VR	(Microsoft 2017)

Aufgrund des Umfangs der zu testenden SDKs, VR-Kits und ggf. VR-Brillen, wird die Technologiewahl auf die *direkten Alternativen a bis c* reduziert, da die Betrachtung aller technischen Alternativen den zeitlichen Rahmen der Masterarbeit sprengen würde und nicht direkt in den Anforderungen liegt. Für die Auswahl wurden alle drei Alternativen auf Tauglichkeit getestet, indem die Beispielmechanik für die Bewegung in der Szene auf eine andere Szene übertragen und ausprobiert wurde. Alle Alternativen sind in diesem Punkt tauglich.

3.2.2.1 Auswahl

Die Auswahl kann durch Knockouts geschehen, die bedeuten, dass die Alternativen nicht alle Anforderungen erfüllen.

Alternative C ist im Beta-Status und damit ggf. noch dem Risiko ausgesetzt, dass sich Funktionen ändern und so Nachpflege notwendig wird. Diese Alternative fällt raus, da es zusätzlich die Long Term Version Unity 2017.4-LTS nicht unterstützt (vgl. „version 2018.3.10f1 (or above)“ (ExtendRealityLtd 2019)).

Alternative A würde das Nutzen einer alten (ggf. veralteten) Steam-VR-Version erfordern, da diese keinen Long Term Support genießt. Diese Handlung zieht das Risiko mit sich, dass ein Update der Treiber auf der Hardware, die nicht unter der Kontrolle der Projektmitarbeiter liegt, diese Implementierung ausschließt und damit unbrauchbar macht. Das kann nicht im Sinne der Wiederverwendbarkeit und Zuverlässigkeit sein.

Die Alternative B ist die sicherste Wahl, da aktuelle, unterstützte Software genutzt wird. Zusätzlich werden Risiken bzgl. Inkompatibilität reduziert, indem eine Softwarezwischensicht (VRTK) entfällt. Steam-VR hat viele Interaktionsmöglichkeiten direkt angeboten, aber bestimmte Spezialitäten nicht. Unter diese Spezialitäten fällt:

- dass Klettern nicht initial mit angeboten wird
- dass kein Kollisionskörper für den Spieler vorgesehen ist und so bewusster Objekte platziert werden müssen, mit denen man interagieren kann

Es wird angenommen, dass diese Einschränkungen in den meisten Fällen zu keinem schlechteren VR-Erlebnis führen sollten, da man sich zunächst nur laufend oder teleportierend durch die Szene bewegen wird und die Kulisse für die Interaktionsmöglichkeiten vorbereitet wird.

Alternative B als Steam-VR ohne weiter KITs und Wrapper ist somit eine gute Wahl, da es die voraussichtlich zuverlässigste Variante ist, sowie alle ausgewählte Unity3D-Version und viele VR-Headsets unterstützt.

3.2.3 MQTT-Binding an Unity3D (vorhandene Lösungen)

Die Recherche für die Libraries (Lib, Libs) bzw. APIs oder Toolkits geschieht in zwei voneinander abhängigen Schritten. Im ersten Schritt wird nach fertigen Lösungen (Libs, APIs, Kits) gesucht, die bereits MQTT erfolgreich und nutzbar an Unity3D angebunden haben. Diese Lösung wird getestet, indem ein Beispiel gebaut wird, welches den String „Hallo from Unity“ an das Topic „`Hallo/Welt`“ sendet.

Falls keine erfolgreiche Lösung zu finden ist wird nach C# Libraries gesucht, die aufgrund der Sprache und Kompatibilität mit .Net oder dem .Net-Subset von Unity3D kompatibel sind (Schritt zwei wird in Kap. 3.2.4 MQTT-Binding an Unity3D über betrachtet).

Die Recherche ist auf vier Kandidaten gestoßen:

- Web API Kit MQTT for IoT von Haptix Games auf UnityAssetStore (Kandidat 1)
- Unity3d_MQTT von vovacooper auf GITHUB (Kandidat 2)
- m2mqtt4unity von masatoshiitoh auf CodePlexArchive (Kandidat 3)
- Workaround von EdwinChua auf answers.unity3D.com (Kandidat 4)

3.2.3.1 Kandidat 1

Der Kandidat 1 **Web API Kit MQTT for IoT** aus dem **UnityAssetStore** ist direkt über den UnityAssetStore (“Web API Kit: MQTT for IoT” 2019) auffindbar und kostet 34,84€. Dieses API Kit funktioniert generell mit dem Beispiel, das beigefügt ist, jedoch ist unklar ob MQTT/TCP oder MQTT über WebSockets oder eine andere Implementierung vorliegt. Zusätzlich ist der Code nicht nach C# oder Unity3D Guides oder in einem ähnlichen Style geschrieben, was den Code schwer nachvollziehbar macht.

Nach einem halben Tag mit dem API Kit produktiv ein Beispiel zubauen, was „Hallo from Unity“ oder nur eine 1 sendet, wurde dieses API Kit als untauglich erklärt.

In den Bewertungen im UnityAssetStore finden sich 1/5 Sternen oder 5/5 Sternen, die Bewertung mit 5/5 Sternen sind Bewertungen von Usern, die den Entwickler direkt um Hilfe gefragt haben. Die Aufgabe einer Lib oder eines API Kit ist die Verwendung durch andere, diese Aufgabe wurde offensichtlich nicht erfüllt.

In Bewertung für diese Arbeit wurde das SOLID-Z2 und generelle Anforderungen an QAF:B und QAF:U nicht erfüllt. Damit ist dieser Kandidat nicht passend.

3.2.3.2 Kandidat 2

Der Kandidat 2 kommt von vovacooper und ist bei GITHUB unter dem Namen Unity3d_MQTT zu finden (Vovacooper 2014). Diese Variante funktioniert einfach im Unity3D-Editor und unter Windows und ist durch die Ähnlichkeit zu anderen Unity3D-Projekten intuitiv zu nutzen. Bei weiterer Recherche hat sich ergeben, dass die Basis des Codes von einer anderen Stelle kommt, die

Basis ist M2Mqtt for .Net: MQTT client for Internet of Things & M2M communication von Paolo Patierno (MVP) (Patierno 2015).

In Bewertung für diese Arbeit, ist dieser Kandidat generell passend, jedoch nicht optimal, da Lage der Weiterentwicklung fraglich ist. Der ursprüngliche Entwickler hat dieses Werk auf einer Community nahen Plattform „Windows Dev Center“ oder auch „code.msdn.microsoft.com“ geteilt und nicht für die Nutzung mit Unity3D geplant.

3.2.3.3 Kandidat 3

Der Kandidat 3 wurde nicht direkt entdeckt, sondern wurde über einen Reddit-Artikel geteilt (Armase and trecoolman 2017). Kandidat 3 wurde über codeplex.com bereitgestellt. Leider kann dieser Kandidat nicht verwendet werden, da die angegeben Quellen für den Source Code nicht zutreffen und auch keine weiteren Informationen zu finden sind. Der Inhalt, der geteilt wird, ist lediglich eine gebaute dll. Anforderungen nach Bekanntheit der Quellen und die Einsicht in den Code sind damit nicht gegeben. In der Bewertung für diese Arbeit wurden Qualitätsanforderungen aus Kapitel 2.5.1 nicht erfüllt, damit ist dieser Kandidat nicht tauglich.

3.2.3.4 Kandidat 4

Der Kandidat 4 (EdwinChua and FlutterShift 2017) ist ein Workaround aus den Unity-Answers[5] und keine Lib oder Kit. Zusätzlich werden weitere Empfehlungen für Libs auf Basis von C# vorgeschlagen. In der Bewertung für diese Arbeit ist dieser Kandidat keine Lösung für eine Lib oder ein Kit was die Arbeit mit MQTT vereinfacht.

3.2.4 MQTT-Binding an Unity3D über angepasste Libraries

Aus der vorangegangenen Recherche zum Anbinden von Unity3D über fertige Lösungen (siehe 3.2.3) ist keine Lösung als umfassend tauglich befunden worden. Es wurde die Erkenntnis gewonnen, dass auch andere Entwickler bereits C#-Libs, die nicht für Unity3D entwickelt wurden, tauglich gemacht haben.

Auf Basis der Aussage der offiziellen Librarys (Libs), die über mqtt/mqtt.github.io referenziert werden (vgl. (Mqtt.org 2014), (Mqtt.org 2018)) kommen folgende Libs, die für die .NET Umgebung erstellt wurden infrage:

- xljiulang/Paho.MqttDotnet (Xljiulang 2017)
- chkr1011/MQTTnet (chkr1011 2018)
- stevenlovegrove/MqttDotNet (stevenlovegrove 2014)
- markallanson/nmqtt (markallanson 2013)
- eclipse/paho.mqtt.m2mqtt (Eclipse 2014)
- mFourLabs/KittyHawkMQ (mFourLabs 2016)
- ericvoid/StriderMqtt (Ericvoid 2016)
- xamarin/mqtt (xamarin 2014)

Zusätzlich wird der Kandidat 2, des ersten Durchlaufs (Patierno 2015), mit verglichen und unter dem Namen: „Patierno /msdn.M2Mqtt-MQTT-client“ geführt. Die Tabelle der Kandidaten befindet sich im Anhang Tabelle 22: Technologiewahl MQTT-Lib für Unity3D Stufe 1-2.

[5] Unity-Ansers ist ein StakeExchange ähnliches Forum für Fragen rund um die Unity3D Technologie. https://answers.unity.com/index.html

3.2.4.1 Vorgehen

Da die Arbeit mit den Libs, die nun ausgewählt werden, enger sein muss als die mit Fertigen Lösungen, fällt diese Auswahl aufwändiger aus. Die Auswahl der Lib geschieht in drei Stufen. Die erste Stufe sortiert schnell ohne viel Aufwand die Libs aus, die nicht näher betrachtet werden sollten. Die zweite Stufe sortiert über einen minimalen Prototyp aus, der prüft ob die Lib mit der Unity-Umgebung zusammenarbeitet. In der dritten Stufe wird die Gesamtauswertung durchgeführt. Durch die zweite Stufe kann die Gesamtauswertung viel genauer einschätzen ob die Doku gut ist und ob es notwendig ist, dass der Code kommentiert ist (vgl. „Richtig kommentieren“ (Ralf and Stefan 2018)).

3.2.4.1.1 Erste Stufe

In der ersten Stufe werden folgende Attribute erfasst:

- Code Verwaltungsplattform (z.B. GitHub, GitLab, …)
- Doku vorhanden / Beispiel vorhanden
- Code-Update-Datum
- Lizenz
- Datum letzter Aktivität über ein Kommunikationskanal (z.B. Bug Tracker, FAQ, …)
- Sprache

Für den Übergang in die zweite Stufe werden folgende K.O.-Kriterien ausgewertet. Sobald ein Kriterium zutrifft scheidet die Lib aus.

- Doku fehlt und Beispiel fehlt
- Sprache ist weder Deutsch noch Englisch
- Lizenz ist nicht bekannt oder lässt eine Weiterverwendung nicht zu
- Der Code ist über 5 Jahre alt
- Die Aktivität auf einem Kommunikationskanal ist mehr als 3 Jahr her

3.2.4.1.2 Zweite Stufe

In der zweiten Stufe wird geprüft ob die Libs in einer der drei letzten Unity-Versionen funktionieren. Die Funktionsprüfung wurde durch das Absetzen einer Debug-Meldung für das Lesen von Nachrichten und durch das absetzten einer Nachricht für das Schreiben von Nachrichten geprüft. Dieses Vorgehen wurde gewählt, um mit möglichst wenig Aufwand die generelle Funktion zu bestätigen.

Die Funktion wurde unter den folgenden Unity3D-Versionen geprüft.

- *Unity 2017.4 LTS*
- *Unity 2018.1*
- *Unity 2018.3*
- *Unity 2019.1*

Für den Übergang in die dritte Stufe werden die Libs entfernt, die nicht unter mindestens einer der drei gewählten Versionen mit verträglichem Aufwand lauffähig gemacht werden können.

Die Lauffähigkeit wird mit einem minimalen Prototyp überprüft, der die Lib verwendet und „Hallo Welt“ plus den eigenen Namen an das Topic „hello/world“ sendet und das Topic „Modellhaus/Fenster/SetGpio“ abonniert und den Wert auf der Unity-Konsole ausgibt.

3.2.4.1.3 Dritte Stufe

In der dritten Stufe werden weitere Attribute erfasst:

- Doku Qualität
- Code Kommentiert (Ja / Nein)
- Code Nachvollziehbar

In der dritten Stufe werden alle Attribute gesamt betrachtet, um die Lib auszuwählen, die in allen Bereichen am besten passt.

3.2.4.2 Auswahl

Die Auswahl der Libs geschieht nach dem oben beschriebenen Vorgehen. Aus den zunächst neun Libs haben 3 Libs den Kriterien aus Stufe 1-2 standgehalten. Genauere Informationen sind im Anhang in den Tabellen *Tabelle* 21 und *Tabelle* 22 zu finden.

- `code.msdn.M2Mqtt-MQTT-client` (Patierno 2015) (Kandidat B1)
- `chkr1011/MQTTnet` (chkr1011 2018) (Kandidat B2)
- `eclipse/paho.mqtt.m2mqtt` (Eclipse 2014) (Kandidat B3)

Diese drei Libs sind alle für die Verwendung in Unity 2018.3 mit dem C#7-Standard geeignet. Für die weitere Verwendung in diesem Projekt wird Kandidat B3 ausgewählt.

Der Vorteil, der sich aus dieser Lib ergibt, ist, dass diese im Gegensatz zu Kandidat B1 auf der offenen Plattform GitHub gehostet wird, dadurch einen gewissen Community-Support erfreuen kann.

Der Vorteil von Kandidat B3 gegenüber Kandidat B2 ist, dass Kandidat B3 auch auf den Unity Versionen 2017.4 und 2018.1 funktioniert, was das nachträgliches Integrieren von anderen Unity-Projekten erleichtert.

Zusammengefasst fällt die Wahl auf `eclipse/paho.mqtt.m2mqtt` (Eclipse 2014) (Kandidat Bs3), da diese Lib alle Kriterien erfüllt und die Anforderung der Nachhaltigkeit, im Sinne der Wiederverwendbarkeit, am besten erfüllt, da sowohl mit zukünftigen Support und Weiterentwicklung zu rechnen ist, als auch die Abwärtskompatibilität gewährleistet ist.

3.3 Mittelschicht

Die Mittelschicht ist das System, was die Zwillinge miteinander verbindet und die Simulation durchführt. Die Mittelschicht wird durch den Sim-Leiter eingestellt und diese regelt den Kommunikationsverlauf zwischen den unterschiedlichen Endgeräten, wie z.B. der VR-Welt, den 1:1-Modellen oder 1:18-Modellen.

3.3.1 Programmiersprache

Für die Wahl der Programmiersprache wird ein 2-Stufiges System genutzt. Die erste Stufe ist die Vorauswahl der möglichen Programmiersprachen. Die zweite Stufe wählt anhand einer Entscheidungsmatrix die Programmiersprache aus. Die Wahl der Programmiersprache stand in Wechselwirkung mit der Wahl der MQTT-Mittelschicht-Varianten (siehe Kapitel 4.3.1). Damit die beiden wechselwirkenden Entscheidungsprozesse keine Blockade erleiden, wurde der Grob-Entwurf der Mittelschicht zeitlich vor der Wahl der Programmiersprache betrachtet.

3.3.1.1 Stufe 1 Vorauswahl

Für die Vorauswahl werden drei Quellen von Programmiersprachen-Ratings kombiniert, jedes Rating arbeitet auf eine andere Weise. Das erste Rating ist der PYPL-Index aus April 2019 (PYPL 2019) (vgl. Abbildung 52), basierend auf der Suchhäufigkeit von Programmiersprachen auf Google. Das zweite Rating ist der TIOBE-Index aus April 2019 (Tiobe 2019) (vgl. Abbildung 55), basierend auf mehreren Faktoren, darunter mehr als eine Suchmaschine. Das dritte Rating ist The State of Octoverse aus dem November 2018 ((Octoverse) 2018) (vgl. Abbildung 54), basieren auf genutzten Programmiersprachen auf GitHub. Details wie sich die Ratings zusammensetzen können eingesehen werden.

Für die Vorauswahl in dieser Arbeit werden diese Ratings durch Punkte in Zusammenhang gebracht. Die Punkte pro Programmiersprache ergeben sich aus 11 minus Rang im Originalrating. Die Summe der Punkte ergeben die Reihenfolge für die Vorauswahl, aus der die besten 5 Sprachen weiterverwendet werden.

Tabelle 17: Verrechnungstabelle der Programmiersprachen Indices

	PYPL-Index		**TIOBE-Index**		**Octoverse**		**Punkte**	
	Rang	Punkte	Rang	Punkte	Rang	Punkte	Summe	Pos
Java	2	9	1	10	2	9	28	1
Python	1	10	3	8	3	8	26	2
JS / TS	3	8	6	5	1	10	23	3
C/C++	6	5	2	9	5	6	20	4
C#	4	7	5	6	6	5	18	5
PHP	5	6	8	3	4	7	16	6
VB	11	0	4	7	11	0	7	7
Obj.-C	8	3	10	1	9	2	6	8
R	7	4	11	0	11	0	4	9
SQL	11	0	7	4	11	0	4	10
Shell	11	0	11	0	7	4	4	11
Ruby	11	0	11	0	8	3	3	12
Swift	9	2	11	0	11	0	2	13
Assembly	11	0	9	2	11	0	2	14
Matlab	10	1	11	0	11	0	1	15

3.3.1.2 Stufe 2 Auswahl

Für die Auswahl der Programmiersprachen wird nun ein Fragenkatalog herangezogen, der zur Auswahl führt. Dabei bekommt jede Frage je Programmiersprache Punkte, einen Punkt für trifft zu, keinen Punkt für trifft nicht zu und einen halben Punkt für trifft teilweise zu.

Aus den Anforderungen geht hervor, dass die Mittelschicht über Webtechnologien ein Frontend zur Verfügung stellt. Dazu muss mit der gewählten Technologie Webentwicklung direkt möglich sein. Ebenfalls aus den Anforderungen ablesbar ist, dass die Mittelschicht über MQTT kommuniziert, daraus ergibt sich für den Fragenkatalog die Frage des Vorhandenseins mindestens einer MQTT Library in der jeweiligen Sprache.

Da die Mittelschicht voraussichtlich mit mehreren Clients oder Sessions arbeiten muss, muss die Mittelschicht Nebenläufigkeiten abbilden. Die gewählte Programmiersprache sollte ein Konzept für Nebenläufigkeiten besitzen, ohne dass die eine weitere Technologie bedarf, damit kein extra Know-how erworben werden muss.

Aus den Nichtfunktionalen Anforderungen ergeben sich die Forderungen nach Übertragbarkeit und Änderbarkeit, diese Qualitätsanforderungen müssen auf technologischer Seite berücksichtigt werden und wird in dem Katalog durch den Punkt Plattformwechsel abgedeckt. Bei direkt für die Plattform kompilierten Programmiersprachen muss bei einem Plattformwechsel die komplette Software neu gebuildet werden, wogegen das bei Interpretierten Sprachen oder Sprachen mit Zwischenschicht nicht der Fall ist.

Die Forderungen nach Übertragbarkeit und Änderbarkeit sind nicht nur Qualitätsanforderungen auf technologischer Seite, sondern auch auf der Organisatorischen. Es ist somit wünschenswert, dass die nachfolgenden Entwickler die Programmiersprache kennen. In dem Fachbereich 5 werden bestimmte Programmiersprachen gelehrt. Da die Entwickler, wie in dem Kapitel 2.1 Stakeholder erwähnt, Studenten aus diesem Fachbereich sind, ergibt es Sinn, eine Programmiersprache zu wählen, die bekannt ist. Ebenfalls sinnvoll ist, eine Programmiersprache zu wählen, die bereits in der Projektumgebung genutzt wurde, um nicht das Repertoire der Entwickler zu breit zu machen.

Tabelle 18: Bewertungsmatrix Programmiersprachen

	C++	C# / .Net	Java	JavaScript	Python
Webentwicklung vorgesehen	0	1	0,5	1	1
MQTT Libs vorhanden	1	1	1	1	1
Nebenläufigkeiten abbildbar	1	1	1	1	1
Plattformwechsel vorgesehen	0	1	1	1	1
Entwicklern bekannt	1	1	0,5	0,5	0
Zukünftigen Entw. bekannt	1	1	0,5	0,5	0,5
Bereits im Projekt eingesetzt	1	0,5	1	0,5	1
Dynamisches Speicher Management	0,5	1	1	1	1
SUMME	**5,5**	**7,5**	**6,5**	**6**	**6,5**

Die Wahl der Programmiersprachen geht zugunsten von C# aus, wobei jede der betrachteten Sprachen bei einem Neustart des Projekts SMILE wohl eine gute Wahl wäre. C# wird indirekt in dem Projekt bereits eigesetzt, da die Wahl der Engine Unity3D zur Wahl von C# führt.

3.3.2 Software Plattform / Framework

Basierend auf der Auswahl der Programmiersprache gibt es drei mögliche Softwareplattformen oder Frameworks, die sich selbst als Framework oder als Softwareplattform bezeichnen. Im Folgenden wird nur von Plattform gesprochen. Die drei zur Auswahl stehenden Plattformen sind Mono, .Net[6] (im Sinn von des .Net Frameworks) und .Net Core[7](Wenzel et al. 2017).

.Net

.Net als das ursprüngliche .Net Framework / Plattform, was erstmals im Jahr 2000 herausgekommen ist, ist ein eher monolithisch aufgebautes Framework und war zunächst nur unter Windows lauffähig (wikipedia 2019b)(It-visions.de 2017).

Mono

Mono ist eine alternative und quelloffene Implementierung von .Net, die erstmals im Jahr 2001 erschien. Ziel von Mono war es, sowohl die generellen Vorteile von Quelloffenheit zu nutzen als auch explizit auf verschiedenen Betriebssystemen crossplattform zu funktionieren (Project 2019).

.Net Core

.Net Core ist eine freie und quelloffene Software-Plattform, die deutlich modularer und leichtgewichtiger ist als .Net und auf mehreren Betriebssystemen funktioniert (Lander 2019, 2018).

.Net 5.0 (2020)

Im November 2020 soll eine neue .Net Version erscheinen, die als eine Zusammenführung von .Net Framework, .Net Core und Mono angedacht ist. Viel wird in .Net Core integriert und ausgebaut, um die Feature von .Net und Mono zu nutzen, ohne die Architektur von .Net Core zu verlieren (Schwichtenberg 2019; Lander 2019).

Auswahl

Für die Anforderungen, dass das System nach Möglichkeit kostenneutral und betriebssystemunabhängig (siehe Qualitätsanforderungen) ist macht die Wahl von .Net Core und Mono Sinn, da es quelloffene Implementierungen sind. Im Punkt der Weiterentwicklung, die mit .Net 5.0 stattfinden wird und der modernen leichtgewichtigen Architektur von .Net Core fällt die Wahl auf .Net Core als Plattform für die Mitteschicht.

3.3.3 MQTT-Client

Durch die Wahl von C# als Programmiersprache kann die Library, die als C#-Lib für die Anbindung von Unity3D an MQTT (siehe Kap. 3.2.4.2) gewählt wurde, auch für die Mittelschicht genutzt werden.

Die Library `Eclipse/paho.mqtt.m2mqtt` (Eclipse 2014) ist nicht exotisch und funktioniert auch außerhalb der Unity3D-Umgebung, sowohl auf Linux als auch auf Windows-Systemen. Mit der

[6] „.Net“ wird „dotNet“ gesprochen und einigen Quellen auch so geschrieben, die beiden Worte sind synonym.
[7] Wie „.Net“ wird „.Net Core“ „dotNet Core“ gesprochen und in einigen Quellen synonym verwendet.

Eclipse Foundation als User des GitHub-Profils und der Information das sowohl der MQTT-Broker Mosquito als auch der HIVEMQ-Broker Libraries der Eclipse Foundation nutzt, kann man von einer vertrauenswürdigen Basis ausgehen, die weiter gepflegt wird.

3.3.4 Trennung der Oberfläche vom Backend

Die Trennung der Oberfläche von dem Backend, ist aktueller Stand der Technik und unterstützt die Kapselung der Businesslogik von der Darstellung (vgl. „Clean architecture“ S.28ff (Smith 2019a), vgl. Kap. 6.2.4.2 (Starke 2014), Kap. 7.1.3 (Starke 2011)). Speziell für den Fall der Auslieferung der Oberfläche als Webanwendung hilft diese Trennung höhere Benutzeroberflächen Anforderungen abzubilden (vgl. S.12f (Smith 2019a)).

Die Trennung der Oberfläche von dem Backend erschlägt zudem die Schnittstellenoffenheit, die für weitere Entwicklungen gewünscht ist (siehe Kap. 2.4.3 3rd-Party System, Kap. 2.5.1 Qualitätsanforderungen). Innerhalb des SMILE-Teams wurde sich auf das Data-Interchange-Format JSON verständigt (Crockford 2006; (IETF) 2014, 2017). Dieses Format, ist leichtgewichtig, menschenlesbar und wird durch alle betrachteten Sprachen und Frameworks unterstützt (Li, Olprod (Microsoft), and Cai 2019; Group 2019; Python Software Foundation 2019; Kotamraju 2013) und bietet damit eine Wahl, die sowohl für die Schnittstelle zum GUI-Client (Mittelschicht Frontend) als auch für externe Systeme gut geeignet ist. Um die Leichtgewichtigkeit von JSON und der einfachen Abbildbarkeit für jedes System zu behalten, wird auf die Zusatz Spezifikation {json:api} verzichtet, die weitere Metadaten und weitere Spezifizierungsdetails fordert und mit überträgt (Klabnik et al. 2019).

3.4 MQTT 5.0 oder MQTT 3.1.1

MQTT5 ist aktuell auf dem Stand „Candidate OASIS Standard“ (vgl. (A. (IBM) Banks et al. 2019)) was heißt, dass die Implementierung der Clients durchaus jetzt stattfinden kann, da die Definition quasi fertig ist, aber die Implementierung ggf. geändert werden müssen falls sich trotzdem was ändern sollte.

Für ein Team, das mit der Technik komplett vertraut ist, ergibt es Sinn jetzt MQTT5 zu verwenden oder zu erproben. Da in dem Projekt SMILE jedoch die Entwickler und Administratoren (*siehe Stakeholder*) häufiger wechseln, ist das nicht ratsam, da auf Änderungen nicht schnell genug reagiert werden kann.

OpenHAB, als ein wichtiges System im Projekt SMILE, hat noch keine genaue Aussage zu MQTT5 getroffen. Viele Client-Libs sind noch nicht auf MQTT5 umgestellt (siehe *Anhang Abbildung* 50).

Damit wird für dieses Projekt die MQTT Version 3.1.1 verwendet, was der Version 4 im Header des MQTT-Protokolls entspricht, in den Standards jedoch Version 3.1.1. genannt wird: „The value of the Protocol Level field for the version 3.1.1 of the protocol is 4 (0x04).“ (((HiveMQ) 2017), vgl. (“MQTT Version 3.1.1 OASIS Standard” 2015)). Der Vorteil, der sich durch diese Version ergibt, die weitreichende Kompatibilität und die Stabilität der bereits vorhanden Implantierungen in Tools, Librarys und Systemen.

3.5 Bezug zur Forschungsfrage

In der Entwicklung eines Systems ist die Wahl der Technologie ein wichtiger Schritt, da man sich damit an diese bindet. Die Bindung an die Technologie ermöglicht es bestimmte Anforderungen zu erfüllen oder einfacher zu erfüllen und andere nicht. Daher ist die Technologiewahl stark mit den Anforderungen verbunden.

Im Kapitel *Modelllandschaft* wurde grob die im Projekt gewählte Technologie aufgezeigt, die die Endpunkte der realen Smart-Living-Modellandschaft bilden. Diese Technologie macht im Modellhaus das Licht an oder steuert die Aktoren, öffnet automatisch ein Fester oder ist in Geräten und Modellen verbaut, die zu digitalen Zwillingen werden.

Im *Kapitel 3D- und VR-Welt* wurde die Technologie gewählt, um den virtuellen Gegenständen, neben der virtuellen Repräsentanz, im zentralen System, auch virtuell darstellen zu können. Diese Darstellung muss zunächst mit dem MQTT-Protokoll arbeiten können und in 3D und VR Interaktion erlauben. Um das abzubilden, wurden eine Engine, ein VR-Toolkit und eine MQTT-Library ausgewählt.

Im Kapitel *Mittelschicht* wurde basierend auf den Anforderungen die Programmiersprache ausgewählt, dessen Wahl durch die Software-Plattform verstärkt wird. Es wurde eine Programmiersprache und eine Plattform gewählt, mit der viele Studenten im Studium am FB5 an der FH Aachen konfrontiert werden. Passend für die Programmiersprache wurde ein MQTT-Client ausgewählt. Aufgrund der Synergie der Programmiersprache in Unity3D und der Mittelschicht konnte dieselbe MQTT-Library gewählt werden. Um die Anforderungen der Modularität besser zu erfüllen und die Schnittstellen für die 3rd-Party Anwendungen einfach zu ermöglichen, wurde entschieden, die Oberfläche von dem Backend der Mittelschicht zu trennen. Aufgrund dieser Modularität ermöglicht dieser Schritt auch in Zukunft eine einfache Anpassung, Erweiterung oder Entwicklung der Oberfläche, da keine direkte Abhängigkeit zum Mittelschicht-Backend existiert.

Im Kapitel *MQTT 5.0 oder MQTT 3.1.1* wurde noch einmal auf die aktuelle Technologie MQTT eingegangen und begründet, warum in dieser Arbeit und dem Forschungsprojekt die neue MQTT 5.0 Version noch keine Vorteile bringt. Der Hauptgrund für MQTT 3.1.1. ist, dass diese Version bereits viele unterstützte Clients und Anwendungen hat und dadurch kompatibel ist und bleibt. Die neue Version wird noch nicht umfangreich unterstützt und würde eine höhere Entwicklungszeit bedeuten inklusive der Anpassung bei Änderungen, die sich bei neuen Anwendungen ergeben.

4 Systementwicklung

In diesem Kapitel werden die Grundlagen geschaffen und aus unterschiedlichen Richtungen die Bestandteile des Systems beleuchtet und die Entwicklung aufgezeigt. Durch die hohe Abhängigkeit von MQTT werden Richtlinien für die Verwendung der Technologie ausgewählt, die MQTT-Anbindung an die 3D-Engine Unity3D geklärt, sowie die Mittelschicht entwickelt. Für die Mittelschicht wird eine Variante gewählt die Entwickelt wird und diese aus Verfahrens- und Struktursicht beleuchtet.

4.1 MQTT Topic Richtlinien

Unterschiedliche Quellen, die MQTT benutzen, empfehlen Konventionen oder nutzen Konventionen für den Aufbau der Topics (vgl. (C. 2018), (West and Roger 2018), ((HiveMQ) 2015)). Die Konventionen von Homie-IoT gehen sogar darüber hinaus. Generell schränken Konventionen den Raum der Möglichkeiten für die Lösung eines Problems ein (vgl. (West and Roger 2018)). Die Einschränkung erfolgt in der Form, dass wenn es mehre Lösungen gäbe, die Lösung verwendet wird, die der Konvention entspricht. Konventionen sorgen damit dafür, dass die Lösungen durch einen weiteren Betrachter, der die Konventionen kennt, schneller ersichtlich wird.

Das Muster, dass Homie-IoT verwendet ist folgendes:

```
homie/device/node/property
```

Das Muster, das man verwendet, muss den Gegebenheiten entsprechen, wie auch alle Einschränkungen, die vorgenommen werden. Homie baut Topics auf, damit diese Geräte innerhalb des Homie-Universums funktionieren, dazu melden sich die Gräte an und versenden Metadaten. In dem SMILE-Projekt arbeiten wir mit festen Topics und ohne die Möglichkeit Metadaten zu versenden. Die Topics selbst müssen somit so viele Informationen geben wie möglich und einer ähnlichen Struktur entsprechen. Für die Arbeit in SMILE, wird mit einer eigenen Konvention gearbeitet:

```
{Modellbereich}/{Gebäude}/{Etage}/{Raum}/{Gerät oder Funktion}
```

Beispiel Topics:

```
modell1zu18/haus5/eg/anmeldung/lichtled
modell1zu18/haus2/eg/kueche/herd1
modellvr/haus2/eg/kueche/herd1
```

Die beispielhaft gezeigten Topics zeigen Topics in der Form, wie diese in SMILE eingesetzt werden. Man erkennt ein ähnliches System wie bei Homie-IoT, jedes Topic-Level ist weitestgehend atomar und wandernd von links nach rechts verjüngt jedes Topic-Level das Topic. Ebenfalls ist bei ähnlichen Gegenständen die Verwandtschaft ersichtlich.

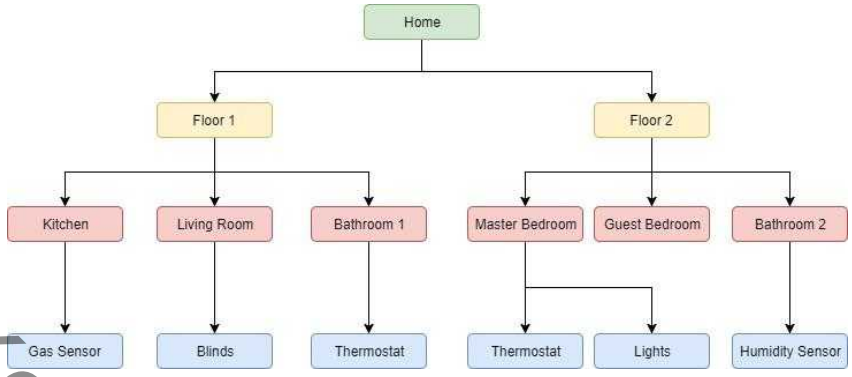


Abbildung 15: Beispiel eines MQTT-Topic-Baum von Starthomeblog.net

Die beispielhaft gezeigten Topics entsprechen ungefähr der Idee, die in Abbildung 15 gezeigt wird.

Neben der Lesbarkeit für Menschen ergeben sich aus dieser Struktur auch die Möglichkeiten, von Wildcards Gebrauch zu machen, wie diese unter ((HiveMQ) 2015) detailliert beschrieben werden. Aus den Beispiel Topics ließen sich über `Modell1zu18/#` alle Topics, die mit Modell1zu18 beginnen, abonnieren, wogegen `+/Haus2/EG/Kueche/Herd1` beide Topics mit `/Haus2/EG/Kueche/Herd1` an ende abonniert, auch wenn diese auf unterschiedlichen „Welten“ bzw. Modellbereichen laufen.

4.2 MQTT Unity3D Anbindung

Anders als in der Mittelschicht werden für die Anbindung der 3D-Engine nicht grundsätzliche Architekturansätze verglichen, da bereits viel Struktur durch die Engine und die Technologiewahl gegeben ist. Hier müssen speziell die Details der Kombination dieser Technologien untersucht werden.

4.2.1 Versuche mit der Bindung

Nach dem in *Kapitel 3.2* die Technologie gewählt wurde, die nun die Grundlage für die Bindung von Untiy3D und MQTT wird, ist die Bindung damit noch nicht fertig. In der Technologiewahl wurde nur geprüft, ob die Lib generell mit der Umgebung von Unity3D und damit verbundenen Einschränkungen in den .Net-Versionen funktioniert.

Bevor die Entwicklung tiefer geht, muss geprüft werden, ob noch weitere Anpassungen notwendig sind, um `GameObjects` in der Scene zu manipulieren, auszuschalten oder zu erstellen.

Der Test wurde mit allen Unity3D-Versionen wie in Technologiewahl beschrieben durchgeführt.

4.2.1.1 Versuch 1 dry Metall

Im ersten Versuch wurde auf das MQTT-Topic „`hello/world`“ reagiert und eine Abstraktion des Aufrufes über die Klasse `UnityEvent` ((Unity 2019h, 2019f)) durchgeführt, um einfach wechseln zu können.

Unity3D lieferte über die Konsole folgende Ausgabe:

```
get_isPlaying can only be called from the main thread.
Constructors and field initializers will be executed from the
loading thread when loading a scene.
Don't use this function in the constructor or field initializers,
instead move initialization code to the Awake or Start function.
UnityEngine.Events.UnityEvent:Invoke()
```

Aus dieser Ausgabe lässt sich einiges herauslesen, da die MQTT-Lib, auf der wir aufbauen, nicht für Unity3D gedacht war und damit den Game Loop (Details zum Konzept Game Loop Kap (Nystrom 2015)), der die Logic und das Ausführen von Code (Details zum Ausführen von Methode in der Unity-Scene (Unity 2019c)) in der Engine steuert, nicht berücksichtigt.

Die erste Zeile der Ausgabe gibt die Hauptfehler an, die man weiter untersuchen kann: `get_isPlaying can only be called from the main thread`. Es kann nur aus dem Hauptthread (und über `Coroutinen` des Hauptthread) auf die Methode `get_isPlaying` und damit auf die `GameObjects` zugegriffen werden.

Generell ist Threading ein Konzept aus der Verteilen Systeme Welt bzw. der Welt der Nebenläufigkeit, da mit Threads das Ausführen von Code neben dem Hauptthread ermöglichen. Das Konzept der Threads ist aus der Verschlankung des Prozess-Konzepts entstanden (vgl. (Tanenbaum and Steen 2008)). Threads laufen ähnlich wie Prozesse nebenläufig (quasi paarallel), aber teilen einen gemeinsamen Bereich an Variablen (die Variablen, des Prozesses, in dem sich diese Threads befinden). Wichtig bei dem Konzept ist, dass jedem Thread ein gewisses Budget an CPU-Zeit zur Verfügung gestellt wird und die anderen Threads dann warten. Der Scheduler (vgl. Kap. 2.4 (Tanenbaum 2009)) ist die Zentrale Einheit, die rundgehend jedem Thread wiederhold CPU-Zeit zuteilt, damit diese dann weiterlaufen.

Die Game Loop in Game Engines ist ein Konzept, das anders ist, als das der üblichen Programme, die auf einem Betriebssystem und damit auf einem Scheduler laufen. Programme in der normalen Benutzerinteraktion warten auf den Benutzereingabe, die Game Loop läuft immer weiter, um die Spielwelt weiterlaufen zu lassen (vgl. Kap. 9.2 (Nystrom 2015)). Im Beispiel eines fallenden Objekts wäre es merkwürdig, wenn das Objekt plötzlich aufhört zu fallen, bis der Benutzer eine weitere Eingabe tätigt. Damit diese Fälle nicht auftreten, verfährt die Game Loop immer, wartet aber nicht darauf, dass ein Ergebnis vorliegt.

`Coroutines` sind in der Unity3D Umgebung ((Unity 2019e, 2018f, 2018e)) ebenfalls ein Konzept, das Nebenläufigkeit realisiert, jedoch ist dieses Konzept in den Game Loop der Game Engine eingebaut und berücksichtigt die Execution Order ((Unity 2019c)), was durch das festgelegte Verhalten die Wahrscheinlichkeit für Deadlocks (Kap. 6 (Tanenbaum 2009)) und Race-Conditions (Kap. 2.3.1 (Tanenbaum 2009)) verringert.

Die Lib nutzt in der internen Abarbeitung der Netzwerknachrichten, die alle nebenläufig eintreffen, Threads. In der Implementierung der Lib rufen die Threads direkt die Methoden der Subscriber (vgl. Observer Pattern (Gamma et al. 2001), (Freeman Eric, Freeman Elisabeth, Sierra Kathy 2008)) auf, die damit in dem laufenden Thread ausgeführt werden und nicht im Mainthread. Unity3D schützt sich vor dem Vermischen der beiden Konzepte und unterstützt nur das Verändern von Objekten

aus dem Mainthread oder der `Coroutinen` des Mainthread. Dadurch wird der Fehler geworfen: `get_isPlaying can only be called from the main thread.`

4.2.1.2 Versuch 2 Blocking Queue

Um das Vermischen der Konzepte zu vermeiden, werden die Konzepte mit einer Nachrichtenwarteschlange (Kap. 4.3.2f (Tanenbaum and Steen 2008), Kap. 15 (Nystrom 2015)) (oder Blocking-Queue, Ereigniswarteschlange) getrennt. In dieser Umsetzung sollen die Konzepte Threading, auf der einen und `Coroutinen` in dem Game Loop auf der anderen Seite voneinander entkoppelt werden, wie zwei mit einander Kommunizierende (Verteilte-) Systeme.

In dieser Implementierung sendet der MQTT- Client aus der Lib die Messages in die Queue und eine `Coroutine` auf Seiten der Game Loop-Welt holt diese ab und verarbeitet diese.

```
UnityEditor blockiert und der Prozess der Engine auf
Betriebssystemebene wird abgebrochen
```

In dieser Implementierung wurde eine C# `BlockingCollection` (vgl. .Net API (Microsoft 2019a)) als `BlockingQueue` eingesetzt, um keine unnötigen Loops zu fahren. Konkret hat genau das Blocken der `BlockingCollection` dafür gesorgt, dass die Game Loop ebenfalls blockierte und damit der Prozess auf Betriebssystemebene.

4.2.1.3 Versuch 3 Active Waiting Queue

Um das Blockieren aufzuheben, wird eine andere Implementierung gewählt, die einer Active - Waiting-Queue Implementierung entspricht und damit besser zu einer Game Loop passt. Dazu wird die Collection in eine Queue (vgl. .Net API (Microsoft 2019b)) geändert und ein `yield return` eingefügt, welches in jeder Iteration um den Playload der `Coroutine` nach Durchlauf die Verbindung in die Game Loop schafft.

Der Test dieser Umsetzung funktioniert gut, und alle Versuche manipulieren, auszuschalten und erstellen von Objekten funktioniert.

```
Funktioniert korrekt.
```

Eine Active-Waiting Implementierung ist häufig eine Variante, die aktiv Rechenzeit verbrennt. Statt durch ein Blocking oder andere Methoden diesen Thread zu pausieren und damit die Rechenleistung einem anderen Thread zukommen zu lassen läuft dieser Thread im Kreis und beansprucht unnötig Rechenzeit.

Da diese Implementierung aber nicht in der Threading-Welt passiert (in der es ein Scheduler gibt der Rechenzeit vergibt), sondern in der Welt der `Coroutines`, die sowieso in einer Loop (der Game Loop) läuft, ist diese Implementierung passend. Das `yield return` sorgt dafür, dass in die Game Loop zurückgeführt wird und keine unnötige Zeit beansprucht wird.

4.2.2 Details der Umsetzung

Um den Einstieg für Entwickler in die Entwicklung zu ermöglichen, werden Abstraktionen und Design Pattern verwendet, die einen Teil der Komplexität ausblenden, um eine initiale Barriere für den Einstieg zu umgehen. Dieses Vorgehen ist sinnvoll, da die Entwickler (siehe 2.1 Stakeholder) standartmäßig kein genaues Know-how bzgl. der Unterschiede zwischen Entwicklung für Game Engines und klassischer Anwendungsentwicklung haben. Da jedes Design Pattern Vor- und Nachteile mitbringt, werden diese Design Pattern einzeln beleuchtet.

4.2.2.1 Zentraler-MQTT-Connector (ZMC)

Für die Verbindung der MQTT-Unity Lib mit dem MQTT-Broker wird ein Zentraler-Connector verwendet. Dieser ZMC managet die Verbindung zum MQTT-Broker und die Weiterverteilung der Nachrichten.

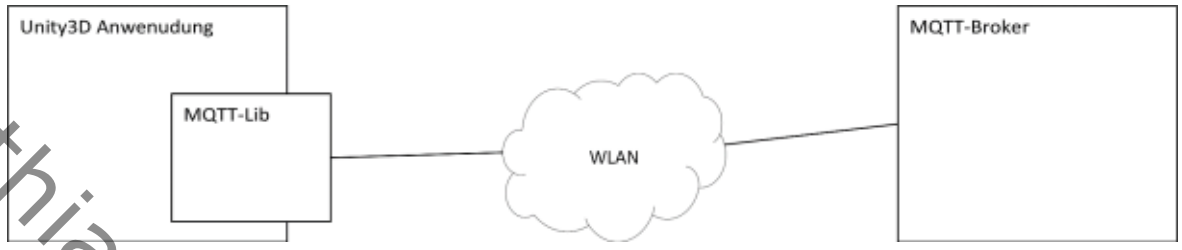


Abbildung 16: Anbindung von Unity3D an einen MQTT-Broker

Das hat mehrere Vorteile:

- Es gibt nur einen Ort, an dem die Konfiguration angegeben oder referenziert werden muss.
- Alle Bausteine (Verwendungen), die den Broker nutzen, können denselben Connector nutzen und brauchen nicht jeweils eine Verbindung zu erstellen.
- Aus Sicht des MQTT-Brokers können alle Anfragen einem Connector, und damit einem definierten Identifier, zugeordnet werden.
- Topics des MQTT Brokers müssen nicht doppelt abonniert werden, sondern der Connector händelt Doppelungen über ein Dictionary und verteilt ankommende Nachrichten.

Dem ZMC Ansatz steht gegenüber, dass es nicht einen ZMC pro Broker gibt, sondern jede Verwendung einen eigenen Kanal zum Broker öffnet und diesen ggf. hält. Diese Lösung ist nicht ratsam, da es mehrere Hundert Verwendungen geben kann in der ein MQTT-Topic gesendet oder empfangen wird. In dem Fall, dass jede Verwendung einen Kanal öffnet und schließt, hat man N-Mal den Overhead für das Öffnen und Schließen der jeweiligen Verbindung. Für den Fall, dass jede Verwendung einen Kanal öffnet und diesen hält, was speziell für das Empfangen von Nachrichten sinnvoll ist, wären mehrere Kanäle gleichzeitig offen, die jeweils eine Netzwerkverbindung halten. Neben der Belastung dieser Anwendung, würde der Broker zusätzlich belastet, da jede Verbindung wie eine eigene Anwendung gehandhabt würde und die jeweiligen Daten zugesendet bekommt. Der Broker würde an dieselbe Anwendung, wenn N Verwendungen in dieser Anwendung dasselbe Topic abonniert haben, N Nachrichten senden. Damit würde der Broker N mal statt einmal belastet, ebenfalls das Netzwerk und die Anwendung selbst.

Zusammengefasst ist die Anwendung eines zentralen Connectors sinnvoll, da des Gesamtsystem entlastet wird und Doppelungen vermieden werden können. Dieser Ansatz ermöglicht es auch mehrere Connector pro Unity-Szene und damit pro Anwendung zu haben, dies wäre sinnvoll, wenn mehrere unterschiedliche MQTT-Broker angebunden werden müssten.

4.2.2.2 Singleton-Like Ansatz

Das Singleton-Pattern ist ein Designpattern der Gang of Four (vgl. Singleton (Gamma et al. 2001)), mit der Ziel sicherzustellen, dass nur eine Instance dieser Klasse existiert und diese Instance über einen globalen Zugangspunkt erreichbar ist. Der Standardanwendungsfall für dieses Pattern ist, der dass mit einer API interagiert werden muss, die nur einen Zugangspunkt haben darf (vgl. Singleton

(Gamma et al. 2001; Nystrom 2015)). Dieser Standardanwendungsfall existiert mit dem ZMC (siehe Zentraler-MQTT-Connector (ZMC)).

Wie im Kapitel Singleton von (Nystrom 2015) diskutiert, löst das Singleton-Designpattern zwei Probleme. Zum einen das Problem, dass nur eine Instance existieren darf und zum anderen die Erreichbarkeit dieser Instance über einen globalen Zugangspunkt. Das erste Problem ist durch die Anbindung des ZMC (siehe Zentraler-MQTT-Connector (ZMC)) gegeben. Das zweite Problem ist durch den Zugriff der Bricks bzw. `UnityEvent`-Bausteine gegeben, denn entweder muss jeder `UnityEvent`-Baustein, der MQTT benutzt, händisch verbunden werden, oder diese Bausteine finden Ihren MQTT-Connector selbständig. Für die Anwendung der Bausteine ist es natürlich einfacher, wenn die Verbindung automatische erstellt wird, daher wird hier das Singleton genutzt, damit die MQTT-Bricks sich automatisch verbinden. Das sorgt dafür, dass der Entwickler oder der Unity3D-User direkt mit den Bricks arbeiten kann, ohne sich um die restliche Technik zu kümmern (Developer Experience).

Moderne Literatur warnt davor das Singleton-Pattern zu häufig zu verwenden. Einige Gründe dafür sind:

a) Die Verbindung zu Singletons ist schwer zu erkennen, da diese dynamisch gebunden werden können
b) Singletons begünstigen die Kopplung
c) Singletons behindern Multithreading

Um die potentiellen Probleme, die mit einem Singleton auftreten können, zu mildern, werden nun die Gründe dafür erläutert und Lösungen gefunden.

4.2.2.2.1 a: Sichtbarkeit von Singletons

Die Verbindung zu Singletons ist schwer zu erkennen, wenn diese dynamisch gebunden werden und es wird erschwert, wenn die Verwendung über den Namespace hinaus geschieht.

Das Singleton wird in dem MQTT-Unity-Anbindung eingeführt, damit die Bricks / `UnityEvent`-Bausteine auf das Singleton zugreifen können. Ein Zugreifen auf diesen Namespace ist von außen nicht vorgesehen und kann mit entsprechenden Regeln den Entwicklerwerkzeugen hinterlegt werden und durch das Team in einem Pull-Request hinterfragt werden.

Der Fall der Erkennbarkeit in den Bricks wird dadurch erreicht, dass eine Referenz auf das Singleton bzw. ZMC gehalten wird. Diese Referenz wird unteranderem auch durch Tools wie Doxygen[8] erkannt und zählt damit als sichtbar/erkennbar. Die Verbindung der abstrakten Klasse (grau) und dem `MqttConnectorSingleton` ist in dem Klassendiagram Abbildung 17 direkt abzulesen.

[8] Doxygen ist ein „de fakto“ Standard-Werkzeug, um Dokumentation aus Code, in unterschiedlichen Programmiersprachen, zu erstellen. Mit Zusatzwerkzeugen generiert Doxygen auch ausgewählte Diagramme des UML-Standards. http://www.doxygen.nl/

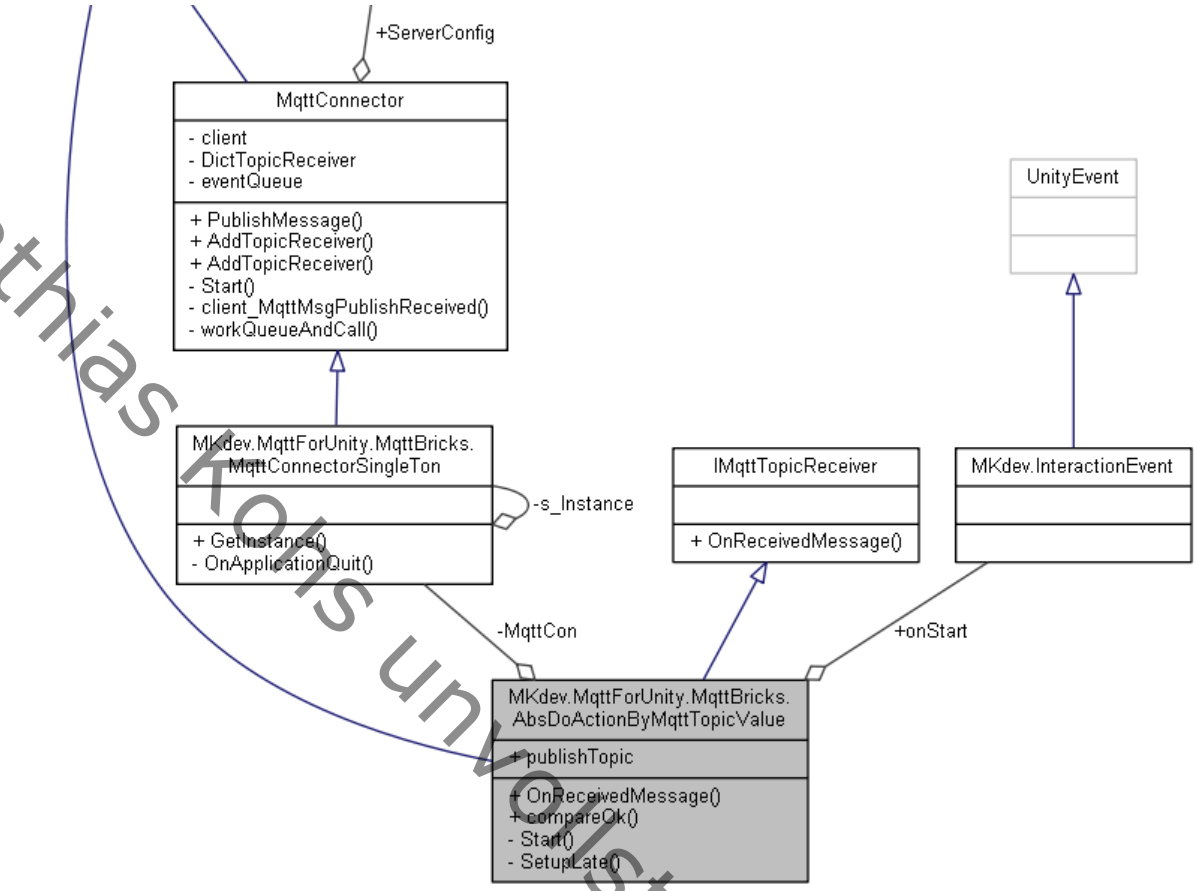


Abbildung 17: UML-Klassendiagramm generiert mit Doxygen Sichtbarkeit des Singleton des MqttConnector als Reverenz in der abstrakten Klasse AbsDoActionByMqttTopicValue
(Commit: 5b81ec90fdb78ebd2c1c657219b2457a5f469489 GitHub: MathiasKoAc/MKdev.M2Mqtt.Unity3D.DoxyDocu)

Es ist vorgesehen, dass neue Bricks, die den `MqttConnector` verwenden, von `AbsDoActionByMqttTopicValue` (siehe Abbildung 17) erben, um die Erkennbarkeit zu behalten. Die Erkennbarkeit ist ein Grundelement der formulierten Qualitätskriterien und erhöht die Änderbarkeit, Wiederverwendbarkeit und in der Weiterentwicklung die Zuverlässigkeit, da die Kapselung die für den Entwickler betrachtete Komplexität reduziert wird. Wir stark die Komplexität reduziert wird, lässt sich am folgen Beispiel erkennen.

4.2.2.2.2 b: Singletons begünstigen die Kopplung

Unter Kopplung versteht man das Maß für die zwischen zwei Bausteinen bestehende Abhängigkeit. Eine geringe Kopplung erfüllt die Forderung nach Einfachheit und ist eine wichtige Voraussetzung für Flexibilität und Verständlichkeit (vgl. Kapitel 6.4 (Starke 2011)). Eine geringe oder lose Kopplung ist eine wichtige Voraussetzung für das Open Close Prinzip (siehe SOLID-OCP: Open Closed-Prinzip).

Dass Singletons die Kopplung begünstigen, ist eine wahre Aussage, der jedoch organisatorisch begegnet werden kann. Wie bereits unter a erwähnt, wurde das Singleton in der MQTT-Unity-Anbindung eingeführt, damit die Bricks / `UnityEvent`-Bausteine auf das Singleton zugreifen können. Ein Zugreifen auf diesen Namespace ist von außen nicht vorgesehen. Das Verwenden von Namespaces kann automatisch detektiert werden, so dass das Team den Zugriff auf den Namespace entsprechend hinterfragen kann.

4.2.2.2.3 c: Singletons behindern Multithreading

Robert Nystrom warnt vor der Benutzung des Singleton-Patterns bei nebenläufigen Anwendungen wie modernen Spielen. Singletons können wie globale Variablen gesehen werden und haben damit ähnliche Probleme wie globale / geteilte Zustände (vgl. Kapitel 6.3f (Nystrom 2015)) (weiter Quelle: 9:10ff (Hipple 2017)). Wenn man geteilte Zustände hat, kann es zu Problemen mit Nebenläufigkeit (Deadlocks und Race-Conditions) kommen (vgl. Kapitel 2.3 (Tanenbaum 2009)).

Für ein Singleton kann es an drei Punkten zu Problemen mit Nebenläufigkeit kommen:

- Bei der Erstellung
- Bei der Benutzung
- Bei dem Löschen

Bei der Erstellung kann es zu Race-Conditions kommen, wenn zwei Threads gleichzeigt (bzw. ineinander verschränkt) ein noch nicht erzeugtes Singleton aufrufen. Wenn das geschieht, kann es zwei (oder n) Instanzen geben, die jeweils eine andere Instance des Singletons halten. Eine programmatische Lösung ist ein Mutex Lock (oder ein ähnliches Konstrukt je nach Sprache) zu verwenden, dass diese Threads geregelt in den kritischen Bereich einzutreten lässt. Eine solche Lösung mit Mutex Locks, kann je nach Implementierung zu Deadlocks führen. Die Lösung, die in dieser Implementierung genutzt wird, ist eine Architektonische Lösung. Die Architektonische Lösung vermeidet diese Probleme komplett, indem es das dynamische Erzeugen herausstreicht. Damit ist diese Lösung nicht vollständig nach dem Designpattern der „Gang of Four“ (vgl. Kapitel Singleton (Gamma et al. 2001)) jedoch löst sie eines der größten Probleme. In der Unity-Szene wird dieser Singleton-Like Zentraler-MQTT-Connector (ZMC) im Editor platziert und existiert zu einem sehr frühen Zeitpunkt des LifeCycle des Spiels und wir gesucht, aber nicht instanziiert. Das Nicht-Instanziieren sorgt dafür, dass immer die korrekte Instance verwendet wird.

Bei der Benutzung kann es zu Problemen mit Nebenläufigkeit kommen, wenn Zustände in dem Singleton gespeichert werden, denn dann ist das Singleton inklusive dieser Zustände wie ein oder mehrere verteile Zustände zu sehen. In der Implementierung, wie sie hier vorliegt ,ist das jedoch nicht der Fall, denn das Singleton ist nur der Zugangspunkt zu dem Zentraler-MQTT-Connector (ZMC). Damit korrekt mit Nebenläufigkeit umgegangen werden kann, hält der ZMC zwei Queues, eine Queue für die MQTT-Nachrichten für den Ausgang, die andere Queue für die MQTT-Nachrichten für den Eingang. Der Nachrichten-Eingang wurde in den Versuchen unter Kapitel 4.2.1 Versuche mit der Bindung thematisiert. Die Entkopplung der Unity-Welt mit dem Game Loop-Ansatz (für das Realisieren von Nebenläufigkeit), von dem Threading-Ansatz sorgt gleichzeitig für ein korrektes Verarbeiten von Nebenläufigkeit in sich. Der Nachrichten Ausgang wurde ähnlich gestaltet, um den Herausforderungen der Nebenläufigkeit zuvorzukommen. Wenn nun unterschiedliche `Coroutinen` in der Unity-Szene MQTT-Nachrichten absetzen, werden diese in die Ausgangs-Queue geschrieben, dieser Vorgang verlangt kaum Rechenzeit. Für die Verarbeitung der Nachrichten kümmerte sich eine `Coroutine`, die Nebenläufig arbeitet und die Nachrichten sequenziell aus der Queue entnimmt und diese über das Netzwerk versendet. Der Vorgang des Versendens kann, je nach Situation, mehr Zeit beanspruchen. Da das Verarbeiten und das Absetzten von Nachrichten über diesen Active-Waiting-Queue-Ansatz realisiert ist, kann es nicht zu Race-Conditions kommen, da nicht auf denselben Zustand zugegriffen wird.

Bei dem Löschen von Singletons kann es zu Problemen kommen, wenn Prozesse in unterschiedlichen LiveCycle-Zuständen sind und gerade im Bereich des Schließens der Anwendung auf das Singleton zugreifen. Speziell dann kommt es zu Problemen, wenn das Singleton sich

klassisch selbst instanziiert. Dieses Problem ist jedoch dadurch, dass die Instance des Singleton-Like ZMC technisch an die Szene angebunden ist und sich nicht selbst instanziiert, gelöst. Die Instance des Singleton-Like ZMC wird durch den Standartprozess abgebaut, der die Objekte in der Szene abgebaut. Um noch potenzielle Bindungen zu löschen, wird vor Beendigung des Programm by `OnApplicationQuit` die instance-haltende Variable im Singleton auf null gesetzt und alle `Coroutines` gestoppt.

4.2.2.3 Observer Pattern / Publish-subscribe Pattern

Im Sinn der Entkopplung der Verantwortlichkeit der Klassen, wird die Reaktion auf die MQTT-Nachrichten separiert von dem Empfangen der Nachrichten zu betrachten. Das Oberserver-Pattern definiert eine 1-zu-n Verbindung, zwischen Objekten, die dafür sorgt, dass Änderungen des Zustands des Providers dazu führt, dass alle Beobachter benachrichtigt und automatisch aktualisiert werden (vgl. Beobachter/Observer (Gamma et al. 2001; Nystrom 2015)).

In Weiterentwicklung des Observer Pattern und Anlehnung an das Publish-Subscribe-Pattern arbeitet der ZMC als erweiterter Provider also Broker. Dieser Broker nimmt Receiver in ein Dictionary als eines von vielen Values auf und nutzt das Topic, welches der Receiver abonniert, als Key. So kann der Broker, in diesem Fall der ZMC, direkt alle Subscriber benachrichtigen sobald, ein neuer Wert auf das Topic empfangen wird. Anders als Broker im klassischen Modell, empfängt der ZMC nur die Nachrichten, die vom MQTT-Broker kommen. In der Implementierung müssen die Receiver das minimale Interface `IMqttTopicReceiver` implementieren.

4.2.2.4 UnityEvent Baustein

Um die Usability für Unity-Editor-Benutzer zu verbessern, wurde aufbauend auf den Erfahrungen der LogicBricks im Forschungsprojekt GHOST und der Entkopplung durch das Observer Pattern `UnityEvent` Bausteine gebaut, die die ersten Basis Funktionen bereitstellen.

Diese `UnityEvent` Bausteine abonnieren im ZMC die Topics, auf die der Unity-Editor-Benutzer eine Reaktion haben möchte und kann ab diesem Punkt über die `UnityEvent` Klasse, die entsprechende Funktion hinterlegen.

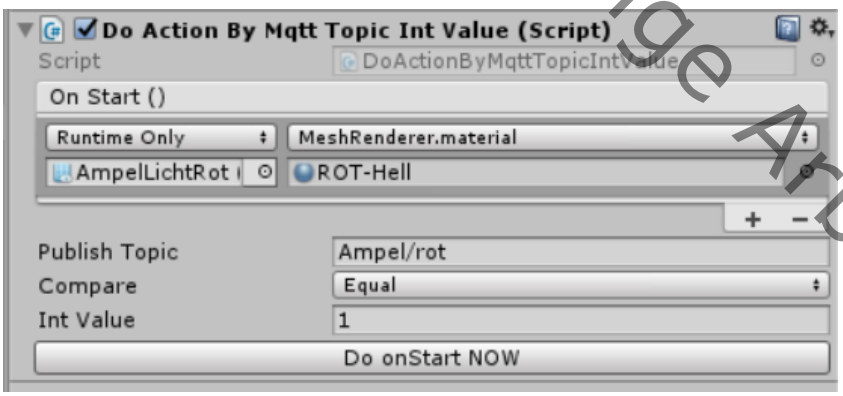


Abbildung 18: Screenshot eines UnityEvent Bausteins (Brick), welches das Topic Ampel/rot aboniert und am Modelteil AmpleLichtRot das MeshRenderer-Material auf ROT-Hell setzt

Wie die Benutzung eines solchen MQTT-Bricks oder `UnityEvent` Baustein aussieht, sieht man in Abbildung 18. Diese Abbildung zeigt ein Brick, das sobald das Topic `Ampel/rot` auf 1 gesetzt ist, für das Modelteil `AmpleLichtRot` das `MeshRenderer`-Material auf das Material ROT-Hell setzt. In der Benutzung muss sich der Unity-Editor-Benutzer um, nichts kümmern als das Topic und Reaktion

selbst. Der Unity-Editor-Benutzer ist entbunden von der Konfiguration der Zugangsdaten für die MQTT-Verbindung und durch das Singleton entbunden von der Verbindung zum ZMC.

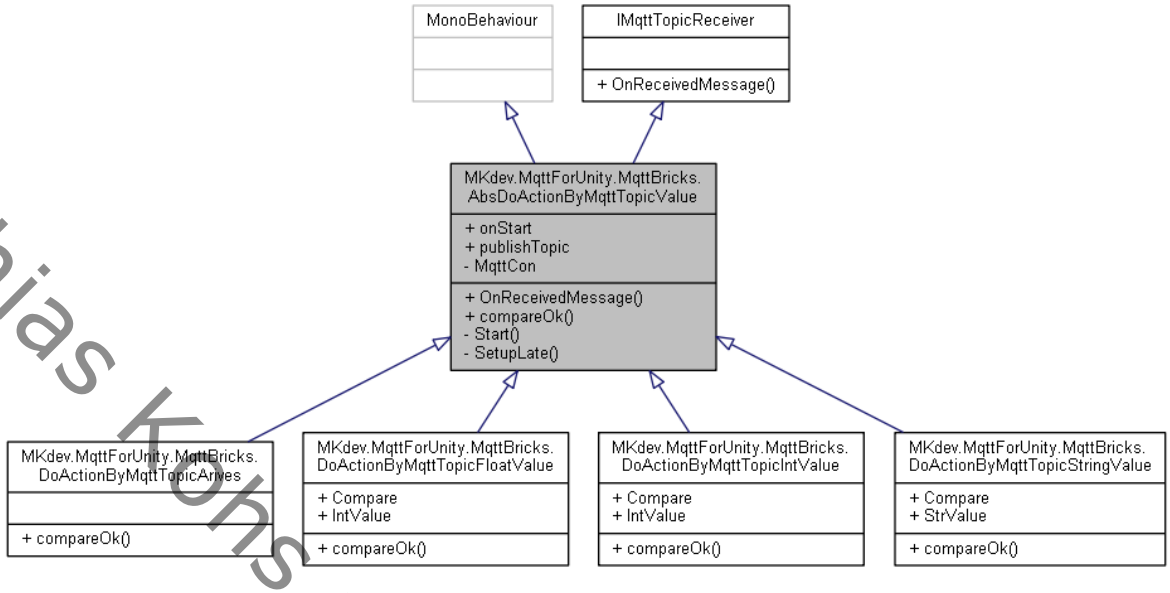


Abbildung 19: UML Klassendiagramm abstrakte Klasse AbsDoActionByMqttTopicValue

Um die Erweiterbarkeit dieses `UnityEvent` Baustein- bzw. MQTT-Brick-Ansatzes zu gewährleisten, ist die Grundfunktionalität in die abstrakte Klasse `AbsDoActionByMqttTopicValue` generalisiert. Wenn nun ein neues MQTT-Brick geschrieben wird um z.B. einen ganz besonderen Datentypen zu unterstützen oder eine andere Art von Vergleich durchzuführen, reicht es, die Methode `compareOk()` zu überschreiben.

4.2.2.5 MQTT-Konfiguration

Um die lose Kopplung der Komponenten über den Build hinaus zu behalten macht es Sinn, die Konfiguration für die Verbindung an den MQTT-Broker auslagerbar zu machen. Genauso sinnvoll kann es sein, einen Build zu erzeugen, der an nicht erfahrene Benutzer heraus geht und mit einem vorkonfigurierten Broker im Internet arbeitet. Ebenfalls denkbar sind Konfigurationsänderungen über ein GUI.

Um so viele Möglichkeiten der Konfigurationshaltung zu unterstützen wie möglich, abstrahiert eine abstrakte Klasse die drei notwendigen Methoden, die durch die jeweilige Implementation erfüllt werden müssen. Diese Definition der „Schnittstelle“ an den ZMC wird über eine abstrakte Klasse, statt durch ein Interface, durchgeführt damit die Implementierung in jedem Fall von `ScriptableObject` ableitet. `ScriptableObject` können als gespeicherte Instanzen außerhalb von Unity Szene innerhalb der Projekt-Assets verwaltet werden, was dem Unity-Editor-Benutzer die Möglichkeit bietet per Drag-and-drop zwischen unterschiedlichen Konfigurationen oder Konfigurationsarten zu wechseln.

Zum Zeitpunkt der Arbeit werden zwei Konfigurationsarten unterstützt. Die erste Art ist die Konfiguration über den Editor und das Speichern der Werte in den `ScriptableObjects`, was nach den Buildvorgang nicht mehr editierbar ist. Die zweite Art ist die Konfiguration über JSON-Files, die über den Editor erzeugt und nach den Build immer noch editiert werden können.

4.2.3 Nachträglicher Vergleich

Während des Veröffentlichens dieser Unity-MQTT-Bindung, für die Verwendung durch einen Projektpartner, ist eine weitere alternative Implementierung aufgefallen. Diese Implementierung ist konkret dabei aufgefallen, als nach Forks der paho.mqtt.m2mqtt-Lib (Eclipse 2014) gesehen wurde, um einen passenden Namen für den eigenen Fork zu finden.

Die Veröffentlichung der eigenen Unity-MQTT-Bindung, während diese Arbeit noch in Bearbeitung war, ist nötig geworden, da ein weiterer Entwickler für die Modelllandschaft nun eine Unity3D-Application baut. Das Teilen dieser Unity-MQTT-Bindung hat zwei mögliche Vorteile, es gibt echtes Feedback von einem Entwickler und der Entwickler muss den Aufwand der Recherche nicht nochmal aufbringen.

Die in dieser Arbeit entstandene Unity-MQTT-Bindung ist unter `MathiasKoAc/MKdev.M2Mqtt` (Kohs 2019) zu finden. Die aufgefallene alternative Implementierung ist unter `gpvigano/M2MqttUnity` (Viganò 2018) zu finden.

4.2.3.1 Vergleich Grobdesign

Im Grobdesign der beiden Ansätze sieht man unterschiedliche Designmodelle, wie im Vergleich der Abbildungen Abbildung 20 Abbildung 21 direkt sichtbar wird. Der Ansatz, dieser Arbeit (`MKdev.M2Mqtt.Unity3D`) ist schlanker und erschlägt nicht alle Anforderungen in einer Klasse. Der Ansatz von Gpvigano ist sehr monolithisch und sorgt für diverse Methoden und Attribute in einer Klasse.

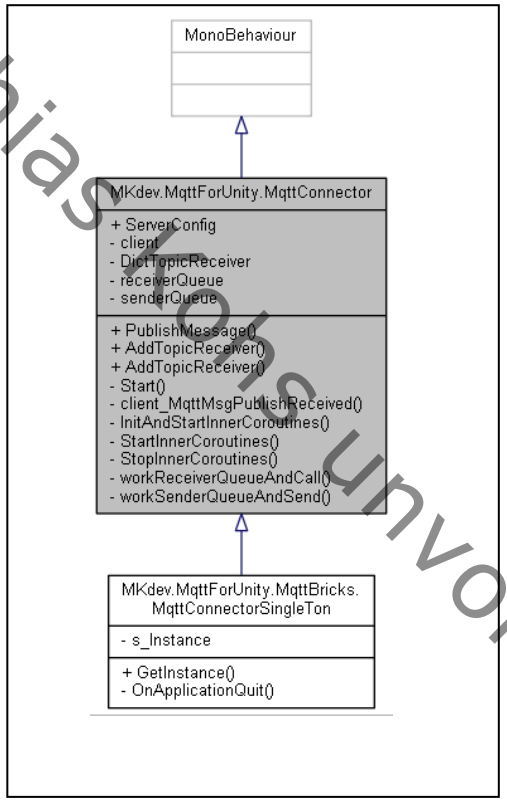


Abbildung 20: UML-Vererbungsdiagramm generiert durch Doxygen aus dem MqttConnector des MQTT-Unity-Binding MKdev.M2Mqtt.Unity3D

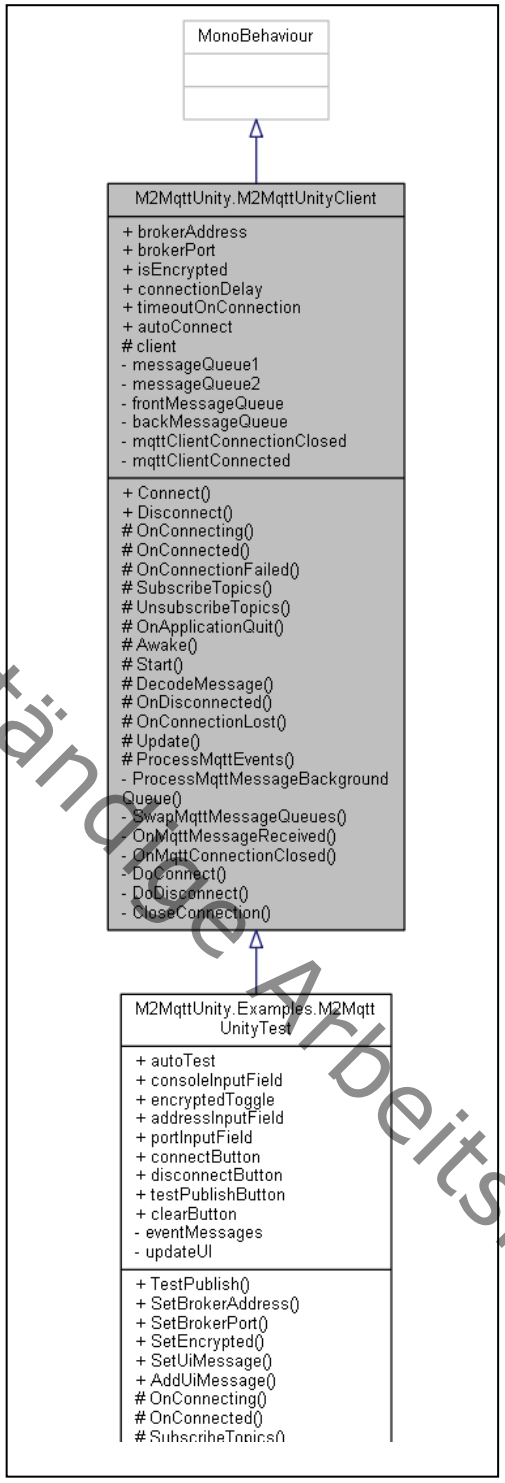


Abbildung 21: UML-Vererbungsdiagramm generiert durch Doxygen aus dem M2MqttUnityClient aus gpvigano/M2MqttUnity

In der Betrachtung, der Attribute der Klassen lassen sich die Auswirkungen bereits erläuterte Design-Entscheidungen wiederfinden, wie z.B. die Attribute der MQTT-Server-Konfiguration. Desweiterten kann man erkennen, dass die Variante von Gpvigano mehr Queues hält. Diese Queues werden im folgenden Kapitel erläutert.

In der Betrachtung, der ableitenden Klasse sind weitere Auswirkungen von bereits wirkenden Entscheidungen zu sehen, denn in der Variante von Gpvigano leitet die Beispiel-Applikation direkt von dieser Klasse ab, was die monolithischen Züge unterstreicht. In der Variante dieser Arbeit, leitet ein schlank gehaltenes Singleton von dem `MqttConnector` ab, mit den Vor- und Nachteilen wie in Kapitel 4.2.2 beschrieben.

4.2.3.2 Vergleich Lösung der Nebenläufigkeit

Das Problem der Nebenläufigkeit für die Anbindung von MQTT-Libs an Unity wurde bereits im Kapitel 4.2.1 untersucht und hat zu einer Lösung geführt. Für den Vergleich der beiden Lösungen ist dieser Punkt interessant, denn die Lösung von Gpvigano verarbeitet das Problem der Nebenläufigkeit anders. Während in der Lösung dieser Arbeit das Prinzip der Waiting-Queue (in Variante der Game Loop) verwendet wird, verwendet Gpvigano das Pattern des Double Buffers (Nystrom 2015).

Das Double Buffer Pattern verwendet man dann, wenn (vgl. Kap 8.3 (Nystrom 2015)):

- es einen Zustand gibt, der schrittweise modifiziert wird.
- es möglich ist, auf diesen Zustand während der Modifizierung zuzugreifen.
- es verhindert werden soll, dass der auf diesen Zustand zugreifende Code die noch in Arbeit befindliche Änderungen bemerkt.
- der Zustand jederzeit lesbar sein soll, ohne darauf warten zu müssen, dass die Modifikation abgeschlossen ist.

Kürzer ausgedrückt: *„Cause a series of sequential operations to appear instantaneous or simultaneous.”* (Nystrom 2014)

Gpvigano nutzt zwei Listen als Queues und swappt diese bei jedem `Update`(), um die Events durchzuführen. Der Grund für das swappen der Queues ist voraussichtlich, dass damit Race-Conditions vermieden werden sollen. Dieses Durchgehen wird jedes für `Update` durchgeführt und damit jedes Bild. Dieses Vorgehen ist effektiv, da nie beide Seiten der Trennung zwischen Unity-Welt und MQTT-Welt dieselbe Liste bearbeiten. Dieses Vorgehen ist jedoch höchst ineffizient, da durch den Aufruf über das `Update` auch das Rendern der Bilder in dem Loop auf das Wechseln Listen warten muss (vgl. `Update()` in (Unity 2019c)). Da der Austausch Zeit kosten kann, kann hiermit auch ein Risiko eingebaut werden, da zu dem Zeitpunkt des Austauschd beide Queues nicht zur Verfügung stehen (vgl. Kapitel 8.4 (Nystrom 2015)).

In meiner Implementierung wird das nicht gemacht, statt zwei Listen wird eine Queue verwendet, die als zirkuläres Array umgesetzt ist (vgl. Hinweise (Microsoft 2019b)), durch diese Datenstruktur ist ein Swappen nicht notwendig und ist damit die performantere (effizientere) Umsetzung. Das Swappen ist in der Queue nicht notwendig, da aufgrund der Ringform und der klaren Arbeitsstruktur nie auf dasselbe Element zugegriffen wird und so potenzielle Race-Conditions und Deadlocks vermieden werden.

4.2.3.3 Beispielbenutzung

Die Beispielimplementierung `M2MqttUnityTest` erbt von der eigentlichen MQTT-Client Implementierung und ist damit ein unglückliches Beispiel für eine starke Bindung von GUI und Controllerelementen. Eine solche starke Bindung zwischen Grundfunktionalität und GUI sollte grundsätzlich vermieden werden, um eine lose Kopplung zu ermöglichen.

Dem Beispiel nach, was in dem Repository dieser Arbeit (Kohs 2019) hinterlegt ist, werden Bricks nach dem Verfahren, wie in Kapitel 4.2.2.4 beschrieben, verwendet. Das Aufzeigen in dem Beispielsorgt ggf. dafür, dass die Nutzer dieses Repository das Verfahren erkennen und weiterverwenden.

4.2.3.4 Wahl der Lösung für diese Anwendung

Aus zwei Gründen ist die Wahl der Anbindung von Unity3D an MQTT der Lösungsweg, der in dieser Arbeit gewählt wurde. Der erste Grund ist, dass diese Lösung einen flexiblen Aufbau hat, der zu dem entkoppelten Aufbau dieser Arbeit und den Anforderungen passt. Der Zweite ist, dass diese Lösung besser den Herausforderungen der Nebenläufigkeit begegnet, in dem das passendere Designpattern bzw. Vorgehen gewählt wurde.

4.3 MQTT-Mittelschicht

Die MQTT-Mittelschicht bildet im übertragenen Sinn den Klebstoff für die unterschiedlichen Welten, denn dieser Klebstoff verbindet die Geräte oder ganze Welten zu Zwillingen. Neben der Verbindung der Welten soll die MQTT-Mittelschicht auch die Anforderungen der Fehlerfall-Simulation erfüllen.

4.3.1 MQTT-Mittelschicht-Varianten

Die herausforderndsten Anforderungen sind die, die den Aufbau der Zwillinge und das Abkoppeln ganzer Geräte fordern. Da die Technologie, die diese Welten anbindet, MQTT ist, ergeben sich unterschiedliche Varianten entlang der MQTT-Kommunikation.

Es ergeben sich logisch „mehrere Orte“, an denen die Kommunikation verbunden werden kann:

- Am Endgerät
- Im Netzwerk
- An einer zentralen Stelle

Entsprechend dieser Orte ergeben sich Lösungsvarianten, die auf diese Orte aufbauen.

4.3.1.1 Regelung je Endgerät / Micro-Controller

Die Regelung je Endgerät ist eine Variante, bei der die Logic, wie mit den Anweisungen (Welt verbinden, Welt trennen, Topic Fehler, usw.) umgegangen wird, auf dem Endgerät ausgeführt wird. Dies ist in nächster Zeit nur mit im Projekt programmierbaren Endgeräten wie Microcontrollern, Einplatinencomputer oder Virtuellen Endgeräten machbar. Nachträglich könnte man geschlossene Geräte erweitern, indem dem entsprechenden Gerät ein MQTT-Client-Broker vorgesetzt wird, der die vorgeschaltete Logic ausführt und die Daten an die eigentlichen Endgeräte entsprechend der Manipulation weitergibt.

Für den Fehlerfall bekommt das Endgerät die Information, die jeweiligen MQTT-Nachrichten zu ignorieren oder Nachrichten nicht zu senden (siehe Anforderungen). Diese Information selbst kann z.B. über spezielle MQTT-Topics versendet werden.

Für das Verbinden der Geräte und Welten zu Zwillingen könnte man Verfahren und dem Micro-Controller die Information geben, welche Topics abonniert werden.

Vorteile

- Mit unterschiedlichen Brokern anwendbar
- Skaliert in der Last durch Anzahl der Geräte, da die Last auf dem Gerät selber getragen wird

Nachteile

- Skaliert nicht in der Entwicklung, da für jede Plattform und Programmiersprache eine Lösung vorhanden sein muss
- In jedem Endgerät / Micro-Controller muss Assembly-Platz eingeräumt werden
- Belastet die Endgeräte / Micro-Controller und damit ggf. die langsamste Komponente im Verbund
- Nicht alle Geräte in der Betrachtung abbildbar → Hardwarelösungen notwendig

4.3.1.2 Topic Trenner

Der Topic Trenner ist eine Variante, die eine echte Mittelschicht baut und neben den anderen Komponenten existieren kann. In dieser Variante wird die Logik, wie mit den Anweisungen (Welt verbinden, Welt trennen, Topic Fehler, usw.) umgegangen wird, zentral ausgeführt.

Das zentrale Ausführen funktioniert in der Art, dass die Topics in Eingang und Ausgang aufgetrennt werden. Alle Endgeräte (auch OpenHAB) abonnieren den Eingang und publishen auf den Ausgang. Die Mittelschicht leitet bei funktionierenden Verbindungen die Ausgänge auf die Eingänge, damit kann die Mittelschicht auch Verbindungen trennen, in dem diese Weiterleitung unterbrochen wird. Zusätzlich kann die Mittelschicht Zwillingen diese Ausgänge (out) und Eingänge (in) wiederrum weiterleiten damit diese auf gemeinsame Aktionen reagieren. Da die Ausgänge und Eingänge getrennt sind, ist es ebenfalls möglich, unidirektionale Verbindungen aufzubauen, so dass die VR Welt den Einfluss der 1u18-Welt hat, aber die 1:18-Welt den Einfluss der VR-Welt nicht.

Vorteile

- Mit unterschiedlichen Brokern anwendbar
- Keine Bindung dieser Komponente an eine andere Komponente oder Technologie
- Die Geräte erkennen den Grund für das Ausbleiben von Nachrichten nicht
- Diese Lösung unterstützt einen Z-Achsen-Split entlang der MQTT Topic-Bereiche, was diese Variante, durch Lastverteilung, in große MQTT-Netzwerken anwendbar macht (vgl. Z-Achsen Split SKU oder data-partitioning (McGlothin 2018; Richardson 2018))

Nachteile

- Diese Komponente wird durch die steigende Anzahl von Geräten stark belastet
- Der Broker wird durch Last durch Anzahl stark belastet
- Jede Nachricht wird potenziell zwei Mal versendet

4.3.1.3 MQTT-Broker neu schreiben

Die zentralste Stelle in dem MQTT-Netzwerk ist der Broker. Eine Variante, um entsprechende Geräte und damit auch Welten zu trennen, kann über den Broker geregelt werden. Der Broker besitzt die Information, welche Geräte verbunden sind und welches Topic diese abonnieren. Über

das Erweitern des Brokers um die Information, welche Geräte verbunden werden, kann der Broker so die Informationen an jene Zwillinge mit senden. Ähnlich kann verfahren werden, wenn Fehler in der Kommunikation erzeugt werden sollen, der Broker kann die entsprechenden Nachrichten verwerfen oder zurückhalten. In der Außenbetrachtung haben die Geräte einen Fehler oder Ausfall.

Vorteil

- Hohe Freiheiten in der Umsetzung
- Die Geräte erkennen den Grund für das Ausbleiben von Nachrichten nicht
- Keine weitere Anwendung notwendig

Nachteil

- Skalierung durch Anzahl sorgt für eine hohe Belastung auf diese Komponente
- Einen Broker neu zu schreiben oder von einem bestehen zu Branchen ist sehr aufwendig und erfordert „gute Entwickler“
- Jeder Anpassung am Broker kann eine Anpassung an der Änderung des Brokers erfordern (Starke Kopplung)

4.3.1.4 MQTT-Broker-Plugin schreiben

Ein ähnlicher Ansatz, wie den Broker komplett neu zu schreiben, ist ein Login-Plugin zu schreiben. Einige Broker erlauben Plugins für die Art der Authentifizierung zu schreiben (vgl. Beispiele ((HiveMQ) 2019d), (Jpmens 2019)). Über diesen Weg wäre es möglich, die Verbindung zu bestimmten Geräten zu trennen und diese damit von der Kommunikation auszuschließen. Wie dadurch Welten verbunden werden, ist jedoch nicht eindeutig geregelt, da dafür weitere Plugins notwendig werden.

Vorteil

- Keine weitere Anwendung notwendig
- Weniger Aufwand als bei der Neuimplementierung des Brokers

Nachteil

- Skalierung durch Anzahl sorgt für eine hohe Belastung auf diese Komponente
- Abhängigkeit der Komponente zum Broker ist groß
- Endgeräte können anzeigen, dass die vom Login ausgeschossen sind

4.3.1.5 Wahl der Entwicklungsvariante

Für die Wahl der Entwicklungsvariante wurden Vor- und Nachteile aufgezählt, diese werden nun in ein Verhältnis gesetzt.

Die Endgerätebasierte Lösung wird nicht gewählt, da sie in drei Punkten schlecht abschneidet. Diese Lösung ist nicht unabhängig von der Plattform, sondern stark abhängig. Sie ist abhängig von der Endgeräteplattform und der Programmiersprache, die auf dem Endgerät läuft. Im Zusammenhang damit skaliert, diese Lösung auch nicht in der Entwicklung, da für jede Plattform und Sprache eine entsprechende Lösung vorliegen muss. Hinzu kommt, dass nicht alle Geräte abgebildet werden können.

Die Lösung, den MQTT-Broker komplett zu schreiben oder von einer vorhandenen Lösung zu forken[9], ist am aufwendigsten in der Erst-Implementation und durch die hohe Abhängigkeit auch aufwendig in der Wartung. Ebenfalls bedingt durch die direkte Kopplung, ist das wechseln eines Brokers mit großem Mehraufwand verbunden. Aus den bisher genannten Gründen fällt, diese Lösung ebenfalls weg.

Die Lösung ein Login-Plugin zu einem MQTT-Broker zu schreiben, löst bzw. mildert die Kritikpunkte, da keine Code-Abhängigkeit mehr existiert, da diese Abhängigkeit durch eine vorgesehene Plugin-Schnittstelle abstrahiert wird und ein Plugin deutlich weniger Entwicklungsaufwand fordert als eine Neuentwicklung. Dennoch existiert eine nicht erhebliche Bindung an den Broker, der gewählt wurde, die man nach Möglichkeit umgeht. Diese Lösung hat im Gegensatz zu den anderen Lösungen den Nachteil, dass der Fehlergrund bzw. der Grund für das Ausbleiben der Nachrichten nicht verborgen bleibt und sich damit schlechter für die Simulation eignet.

Die Topic Trenner Lösung ist die präferierte Variante, da diese Lösung eine hohe Unabhängigkeit aufweist und in den meisten Punkten volle Bewertung erhält. Diese Lösung skaliert in der Entwicklung, da dieses System geschlossen ist und keine Bindung oder starke Kopplung an die anderen Technologien hat. Diese Lösung skaliert in Arbeitsaufwand, da kein Mehraufwand ersteht, da lediglich Topics per Regeln getrennt oder verbunden werden.

Tabelle 19: Bewertungsmatrix der Entwicklungsvarianten der Mittelschicht

	Endgeräte Basiert	MQTT Broker komplett	MQTT Broker Plugin	Topic Trenner
Unabhängig	1	0	1	3
Skaliert in der Last	3	3	3	2
Skaliert in der Entwicklung	0	0	2	3
Skaliert in der Administration	2	2	3	3
Fehlergrund verborgen	1	1	0	1
Alle Geräte abdeckbar	0	1	1	1
	7	**7**	**11**	**13**

[9] Forken im Sinne eines Repository-Forks auf GIT; eine Kopie des originalen Projekts, mit Verweis auf das Original, ohne durch Änderungen das Original Projekt zu beeinflussen.

4.3.2 Bausteinsicht der Mittelschicht

Die Mittelschicht besteht aus drei großen Teilen: WebFrontend, Backend und Datenhaltung.

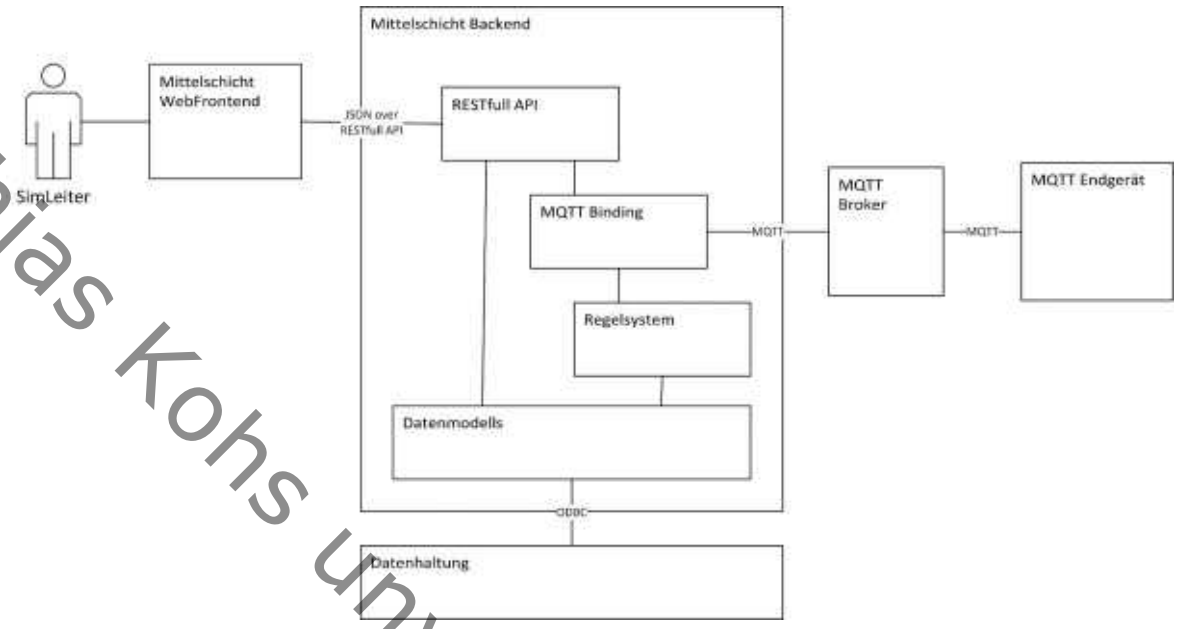


Abbildung 22: Baustein und Abgrenzungssicht der Mittelschicht

Das Web Frontend ermöglicht dem Versuchsleiter (Simulations-Leiter) zum einen Regeln für den MQTT-Topic-Trenner festzulegen, um damit Zwillinge zu verbinden oder Verbindungen zu Geräten zu trennen. Zum anderen ermöglicht das Web Frontend, direkt von MQTT-Verbindungen zu lesen und zu schreiben, um damit Testwerte zu setzen oder nicht funktionierende Geräte zu simulieren. Damit das Frontend, ihre Funktion erfüllt arbeitet dieser mit der RESTfull API des Mittelschicht Backends.

Das Backend der Mittelschicht besteht aus mindestens vier Bausteinen: RESTfull API, MQTT Binding, Regelsystem und Datenmodell. Zusätzlich kann noch zu dem Regelsystem ein Simulationssystem festgelegt werden, was wiederkehrende Events behandelt.

Die RESTfull API und das MQTT Binding in Kombination bilden das System, welches die Schnittstelle anbietet, MQTT Nachrichten zu versenden. Diese Schnittstelle, kann durch andere Systeme genutzt werden, die das MQTT Protokoll nicht selbst abbilden.

Das MQTT Binding mit den Datenmodells sorgen für die Regeln, die die Zwillinge in Verbindung halten und die Regeln durchsetzen, die Geräte ausschalten und Störungen simulieren.

Die RESTfull API und das Datenmodell bilden die Möglichkeit ab, Regeln zu lesen, zu setzen, zu verändern und zu löschen.

4.3.2.1 Grundstruktur

Die Grundstruktur des Systems wird in der Ausgangslage durch die Standard Struktur von ASP.NET Core für die Web-API vorgegeben (Anderson and Wasson 2019). Diese Struktur eignet sich gut, um erweitert zu werden, und ist in einer ähnlichen Ausprägung den potenziellen, nachfolgenden

Entwicklern durch das Erstellen von WebApp mit dem MVC-Pattern (Smith 2019b) aus Vorlesungen bekannt.

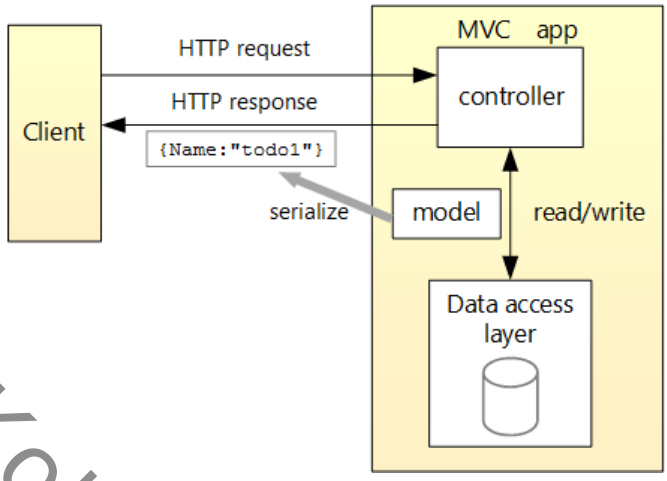


Abbildung 23: Vereinfachter Aufbau einer ASP.NET Core API-App auf Basis von MVC https://docs.microsoft.com/de-de/aspnet/core/tutorials/first-web-api

Abbildung 23 zeigt den Aufbau einer ASP.NET Core API-App auf Basis von MVC. Die API-App Struktur entspricht weitestgehend der MVC Struktur mit dem Unterschied, dass die View nicht vorkommt. Der Controller nimmt HTTP-Requests entgegen und lädt das Model auf der Datenhaltung und sendet es serialisiert, normalerweise als JSON-String, zurück. In der MVC Struktur würde eine View mit den Daten zurückgeliefert werden.

4.3.2.2 Services der Bausteine

Ausgehend von den Bausteinen und den Anforderungen lassen sich mehrere Services definieren, die selbständig oder durch einen Anstoß durch den User Datenliefern oder weiter Prozesse ausführen.

Die Bausteine der RESTfull API und der Datenhaltung sind durch die Controller, die Model und das Data access layer erfüllt (vgl. Abbildung 22 und Abbildung 23).

Damit die MQTT-Kommunikation abgebildet werden kann benötigt das System einen MQTT-Service, der selbständig auf die hineinkommende Nachrichten hört und diese weitergibt. Über denselben Service sollen Nachrichten abgesetzt werden.

Um Anforderung 13 gerecht zu werden ist ein Log-Service notwendig, denn in dem System arbeiten aufgrund der unterschiedlichen Kommunikationswege viele Informationen nebenläufig, um die Logs zu persistieren wird ein entsprechender Service benötigt.

Um die Anforderungen an die Regeln und an den Topic Trenner zu ermöglichen, ist ein Rule-Service notwendig, der die Regelauswertung und Verarbeitung durchführt. Dieser Rule-Service wird durch den MQTT-Service getriggert, sobald Nachrichten ankommen.

Um die Anforderungen an das konstant Halten von Werten und geplante Ausführen von Wertmanipulation und weitern Anforderungen 12, 25-30 gerecht zu werden muss, ein Service vorliegen, zeitabhängig oder in Abhängigkeit von Eingängen werden dann Events ausgelöst.

Um den Anforderungen zum Versenden und Vermitteln der Zeit gerecht zu werden, muss ein Time-Service erstellt werden.

4.4 Verfahrensansicht Mittelschicht

In dem folgenden Kapitel werden Hauptverfahren und Prozesse der Mittelschicht und deren direkte Folgen, wie das Abbilden der Verfahren und Prozesse, betrachtet.

4.4.1 Verfahren des Topic Trenners

In dem Verfahren des Topic Trenners wird zwischen Eingängen und Ausgängen unterschieden, damit die Geräte funktionieren, werden diese verbunden. Um zwischen Ein- und Ausgängen zu unterscheiden, sind alle Topics mit dem Präfix IN oder OUT versehen. Die Verwendung des Präfixes für die Topix ist Konvention und muss bei jedem Gerät eingestellt werden. (Für den Übergang kann das Präfix OUT gelassen werden und man nutzt die Ursprünglichen MQTT-Topics.)

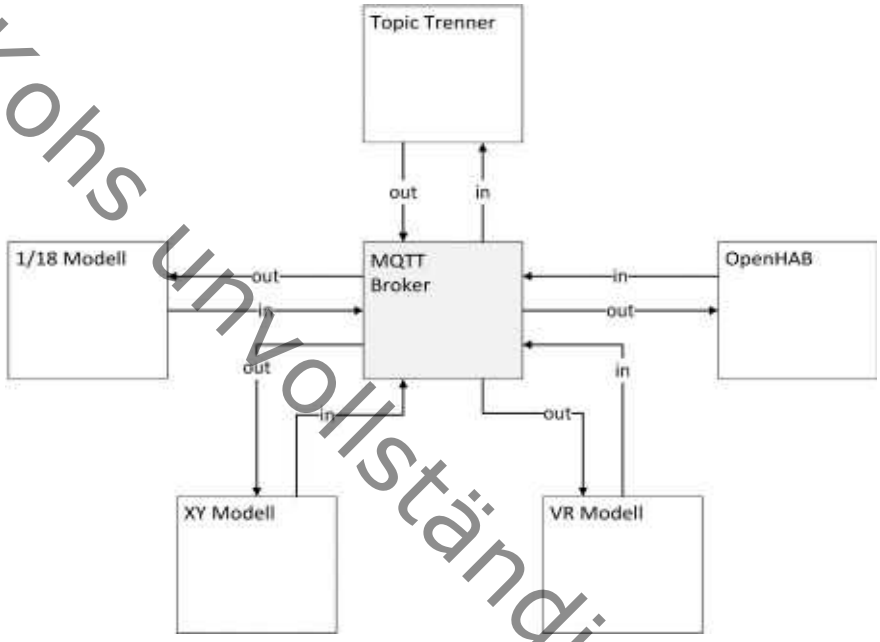


Abbildung 24: Geräte in dem Verfahren des Topic Trenner

Für die Geräte, die feste Topics haben, werden Sonderregelungen auf deren Topic geschaffen. Alle Topics werden zentral auf den Topic Trenner ausgerichtet. Alle Geräte schreiben auf die jeweiligen Topics mit dem Präfix *in* (da es in den Topic Trenner hinein geht) und subscriben (also lesen) auf den Topics mit dem Präfix *out*. Der Topic Trenner arbeitet entsprechend andersherum, da dieser die Topics verbindet bzw. trennt. Wenn man aus der Abbildung 24 den Broker ausblendet, verändert sich das Bild hin zum Topic Trenner als zentrales Element, was es in der logischen Sicht der Mittelschicht auch darstellt (Abbildung 25).

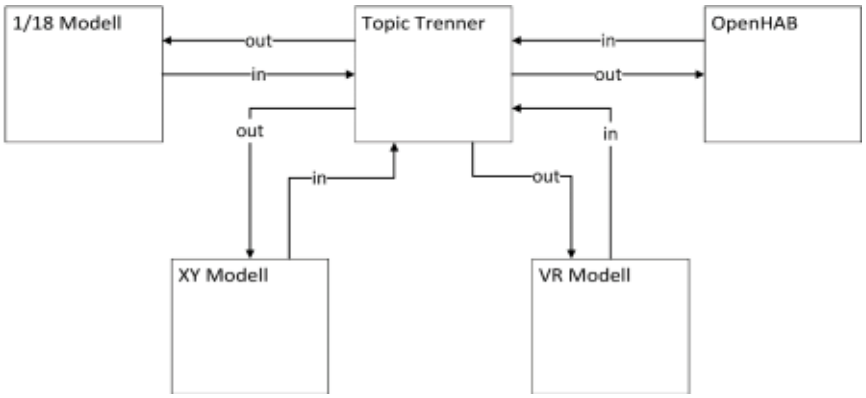


Abbildung 25: Logische Ansicht des Topic Trenners ohne Broker

Aus Sicht des Topic Trenners:

1. kommen die Messages über *in* rein
2. die Regeln werden ausgeführt
3. und die Messages gehen über *out* der generierten Topics wieder raus.

Aus der Sicht der Module:

1. kommen die Messages über *out* rein
2. und die Module senden selbst über *in* raus

Aus der Sicht der Module wirkt die Betrachtung kontraintuitiv und aus Sicht des Topic Trenners wirkt diese Betrachtung intuitiv. Wenn man die Zukunft der Topic Trenner und das Einsatzgebiet betrachtet ergibt es Sinn. Die Module werden selten und wenn von Entwicklern oder Systemarchitekten verändert oder erstellt, die das Gesamtsystem durchdrungen haben (siehe Kap. 2.1). Die Regeln, die nun die Topics matchen, werden durch Versuchsleiter geschrieben und angepasst, dieser Fall findet deutlich häufiger statt, als das Entwickeln von Modulen oder Ergänzen von Modellen in der Landschaft. Aus Sicht der Regeln im Topic Trenner kommen die Nachrichten über IN-Topics mit den Präfix *in* an und werden an OUT-Topics mit dem Präfix *out* versendet, was das Verständnis des Systems leichter macht.

4.4.2 Mittelschicht Topic Regeln

Die Regeln in der Mittelschicht, werden verwendet, um die eingehenden Messages anhand ihrer Topics an die passenden weiteren Topics zu senden. Diese Regeln verändern den Inhalt der Messages nicht, anders als die Regeln, die durch OpenHAB ausgeführt werden. Es gibt zwei Arten von Regeln, die erlaubenden (access) Regeln und verbietende (deny) Regeln.

4.4.2.1 Access Regelauswertung

Dadurch, dass die Regeln der Topic Trenners ausschließlich mit den Topics der Messages arbeiten, kann man die Regeln in Regelbestandteile der linken Regelseite und rechten Regelseite aufteilen. Die linke Regelseite beschreibt das Matching und die rechte Regelseite beschreibt das Topic, auf das gesendet werden soll. Die linke und rechte Regelseite beeinflussen sich in der Ausführung nicht

gegenseitig. Wenn die linke Regelseite matcht, wird mit der rechte Regelseite das neue Topic gesetzt und ausgeführt.

Angelehnt an das Subscriben der Topics im Broker ((HiveMQ) 2015) können die Regelseiten zwei Arten von Wildcards haben. Das „Plus“ + als Singlelevel-Wildcard, was ausdrückt, dass genau an dieser Stelle ein beliebiges Topic Part stehen kann und das „Sharp“ # als Multilevel-Wildcard, was ausdrückt, dass ab hier beliebige Topic Parts folgen können. Das „Sharp“ kann ähnlich, wie beim Subscriben nur am Ende stehen.

Wildcards auf der linken Regelseite beschreiben nur das Matching. Wildcards auf der rechten Regelseite lassen Raum für Variationen der Topics, an die gesendet werden können.

BEISPIEL 1:

```
Linke Regelseite: VR/+/EG/Anmeldung/LichtLed
Rechte Regelseite: Modell1zu18/+/EG/Anmeldung/LichtLed
```

Wenn nun eine Message mit folgendem Topic ankommt:

```
Topic-IN: VR/Haus1/EG/Anmeldung/LichtLed
```

Wird durch die linke Regelseite auf ein Match geprüft. Die linken Regelseite matcht, da alle Topic Parts an den entsprechenden Leveln gleich sind, bis auf das Topic Part *Haus1*, das die Wildcard *+* matcht. Damit wird die rechte Regelseite ausgeführt. In der rechten Regelseite wird das + durch das Element in Topic-IN an diesem Topic-Level (an dieser Stelle) ersetzt. Durch die rechte Regelseite wird auf folgendes Topic weitergeleitet.

```
Topic-OUT: Modell1zu18/Haus1/EG/Anmeldung/LichtLed
```

BEISPIEL 2:

Beispiel für nicht erfolgreiches Matchen.

```
Linke Regelseite: VR/+/EG/Anmeldung/LichtLed
Rechte Regelseite: Modell1zu18/+/OG1/Anmeldung/LichtLed
```

Wenn nun eine Message mit folgendem Topic ankommt:

```
Topic-IN: VR/Haus1/OG1/Anmeldung/LichtLed
```

Es wird ein Match mit der linken Regelseite versucht, dieser aber nicht erfolgreich durchgeführt, da in der linken Regelseite an Topic-Level 3[10] das Topic-Part **EG** gefordert wird, die Message aber **OG1** liefert. Dadurch, dass die linke Regelseite keinen Match hat, kann die rechte Regelseite nicht ausgeführt werden.

[10] Hier wird mit Topic-Level 1 angefangen zu zählen

BEISPIEL 3:

Die Regeln erlauben Abweichungen in ihrem Aufbau:

```
Linke Regelseite: VR/+/EG/Wohnzimmer/LichtLed
Rechte Regelseite: Modell1zu18/+/EG/Anmeldung/LichtLed
```

Wenn nun eine Message mit folgendem Topic ankommt:

```
Topic-IN: VR/Haus1/EG/Wohnzimmer/LichtLed
```

Matcht die linke Regelseite und es wir durch die rechte Regelseite auf folgendes Topic weitergeleitet:

```
Topic-OUT: Modell1zu18/Haus1/EG/Anmeldung/TannenbaumLed
```

Das heißt, die Regeln müssen keinen Identischen Aufbau haben.

BEISPIEL 4:

```
Linke Regelseite: VR/+/EG/Wohnzimmer/LichtLed
Rechte Regelseite: Modell1zu18/Haus6/#
```

Wenn nun eine Message mit folgendem Topic ankommt:

```
Topic-IN: VR/Haus1/EG/Wohnzimmer/LichtLed
```

Wird durch die linke Regelseite gematcht und die rechte Regelseite baut das neue Topic. Das Sharp in der rechten Regelseite wird durch die Topic Parts, aus der Message ab dem Topic-Level 3. So wird auf folgendes Topic weitergeleitet:

```
Topic-OUT: Modell1zu18/Haus6/EG/Wohnzimmer/LichtLed
```

BEISPIEL 5:

Für das Verbinden von Welten, bei dem eine identische Struktur angenommen wird, können natürlich auch freiere Regeln formuliert werden, die eine größere Menge matchen:

```
Linke Regelseite: VR/#
Rechte Regelseite: Modell1zu18/#
```

Wenn nun eine Message mit folgendem Topic ankommt:

```
Topic-IN: VR/Haus1/EG/Wohnzimmer/LichtLed
```

Wird durch die rechte Regelseite auf folgendes Topic weitergeleitet:

```
Topic-OUT: Modell1zu18/Haus1/EG/Wohnzimmer/LichtLed
```

Wie im Beispiel zuvor wird das Sharp in der rechten Regelseite durch die Topic Parts ab dem Topic-Level, auf dem die Raute steht, ersetzt.

4.4.2.2 Deny Regelauswertung

Nach einem reduzierten Kognitiven Walkthrough hat sich ergeben, dass neben den Access-Regeln auch Deny-Regeln existieren müssen um spezielle Fälle abzubilden wie: Das Erlauben von einem

ganzen Topic Bereich *A/#* außer einem Teilbereich *A/B/#*, der ausgelassen wird. Die Deny-Regeln ermöglichen bei größeren Topic-Bäumen einfacher denselben Sachverhalt abzubilden, da weniger Regeln geschrieben werden müssen.

Um die Komplexität niedrig zu halten werden die Deny- und Access-Regeln in einer festen Reihenfolge am Block je Typ ausgeführt. Es wird also keine Ausführungsreihenfolge geben, die zum Beispiel folgende Ausführungsreihenfolge erlaubt:

```
Deny1, Deny2, Access1, Access2, Deny3, Access3
```

Die Deny-Regeln können nun theoretisch Topic-IN Topics verbieten oder Topic-OUT Topics verbieten. Daraus ergeben sich folgende Möglichkeiten der Ausführungsreihenfolge (jeweils am Block):

```
A: Access, Deny-OUT
B: Deny-IN, Access
C: Deny-IN, Access, Deny-OUT
```

Da die Möglichkeit C die größte Freiheit in der Wahl der Deny-Regeln darstellt wird und die Komplexität nicht viel höher wird, wird diese gewählt. Diese Möglichkeit kann jedoch auch den größten Rechenaufwand bedeuten. Da hier jedoch von dem Faktor 2 ausgegangen werden kann, wird diese Lösung genutzt und nach der Implementation getestet.

4.4.2.3 Anlehnung an das Subscriben der Topics

Für die Access- und Deny- Regeln wurde ein Format, speziell mit Fokus auf die Wildcards, gewählt, was in der Welt von MQTT bereits vorhanden war, anders als z.B. das Format von Regulären Ausdrücken. Diese Entscheidung basiert auf der ersten Regel der „8 Golden Rules of Interface Design“ (Shneiderman 1998), dem Streben nach Konsistenz. Das Streben nach Konsistenz sorgt für ein besseres Annehmen und Anlernen von Usern, ohne dass die User neues Wissen für ähnliche Vorgänge erwerben müssen.

4.4.3 Verarbeitungsansatz der Topics

Die Verarbeitung der Messages und Topics in der Mittelschicht stellt den am häufigsten durchlaufenden Teil in der Mittelschicht dar, da bei jeder Message die Verarbeitung durchlaufen werden muss. Neben dem Punkt der Häufigkeit, mit der dieser Prozess durchlaufen wird, ist diese Verarbeitung ein wichtiger Teil, der ausschlaggebend für die Reaktionszeit ist. Entsprechend muss bei diesem Teil der Anwendung ein besonderes Augenmerk auf die Effizienz und Skalierbarkeit dieses Prozesses gelegt werden.

Zunächst muss bei einer festen Anzahl von Regeln und Anfragen eine gute Reaktionszeit gewährleistet werden. Für die Skalierbarkeit gibt es ein bis drei Dimensionen:

- Mehr Regeln
- Mehr Messages
- (länge des Topic im Request)

Diese drei Dimensionen können unabhängig voneinander Wachsen.

4.4.3.1 Analyse einer Bestehenden Lösung aus einem Broker

Die Mittelschicht hat in der Überprüfung der Topics eine ähnliche Aufgabe wie Broker diese haben. Daher ist es von Vorteil sich bestehende Lösungen anzusehen.

Der MQTT-Broker HiveMQ hat eine Community-Edition, die unter GitHub zu finden ist ((HiveMQ) 2019b). Für den Match zwischen den Topics ist die Klasse *TokenizedTopicMatcher* verantwortlich ((HiveMQ) 2019a).

Die public-Methode der Klasse `TokenizedTopicMatcher` bekommt zwei Strings übergeben, den String, der die aktuelle Subscription enthält und den String, der das aktuell zu verarbeitendem Topic enthält. Als Rückgabe gibt es ein nativ Boolean (`boolean`) mit dem Wert `true`, bei einem erfolgreichen Match, und dem Wert `false`, wenn nicht.

Im Kern bereitet diese Methode, die Strings vor und vergleicht diese über einen klassischen String-Vergleich, wenn keine Wildcard (# oder +) enthalten ist. Wenn eine Wildcard enthalten ist, wird eine private Methode aufgerufen, die wiederum Sonderfälle betrachtet und Vorbereitungen trifft, um den eigentlichen Vergleich durch zu führen. Dieser Vergleich zerteilt die Strings jeweils in Arrays von Strings, mit einem Topic Part je Feld im Array. Diese Parts werden verglichen und bei Ungleichheit wird nochmal geprüft, ob Wildcards vorliegen. Wenn innerhalb der Iteration die Methode nicht bereits ein Match oder ein Nicht-Match ausgegeben hat, wird final untersucht, ob die Arrays gleich lang sind oder das subscriptionPart an der letzten stelle nach dem Index der Iteration eine Wildcard hat.

4.4.3.1.1 Topic Trenner Besonderheiten (Andere Anforderungen)

In der Mittelschicht wird der Matcher eingesetzt, um die Topics der Messages mit den Topics der Regeln zu matchen. In dem Broker wird der Matcher eingesetzt um die Topics der Messages mit den Topics der Subscriptions zu Matchen.

In dem Einsatzfeld des Brokers sollten Anwendung kurzfristig Topics abonnieren oder deabonnieren können und der Broker muss schnell darauf reagieren. Die Mittelschicht hat diese Anforderungen nicht, da die Regeln primär durch den Versuchsleiter (siehe Kapitel 2.4) gesetzt werden. Diese andere Anforderung ermöglicht es, bei Änderungen der Regeln etwas mehr Zeit zu beanspruchen. Diese Mehr Zeit kann dazu verwendet werden, eine Datenstruktur aufzubauen oder zu befüllen, die bei der Abfrage Zeit spart. Anders Formuliert ermöglichen die etwas geänderten Anforderungen, den Zeitaufwand der Abfrage auf den Zeitaufwand der Erstellung in Teilen umzulegen.

4.4.3.1.2 Schluss der Lösung

Teile der bestehen Lösungen können konzeptionell übernommen werden, wie das Abfragen der unterschiedlichen Sonderfälle. Die Grundidee direkt mit den vollständigen Topic-Strings zu arbeiten ist aufgrund dessen, dass die Mittelschicht die Hauptperformance im Durchführen der Topic-Matches hat, nicht sinnvoll. Eine Datenstruktur, die dafür sorgt, dass nicht alle Strings einzeln verglichen werden müssen ist sinnvoller.

4.4.4 Abbilden der Regel über Knoten

Um nicht alle Ausprägungen der Topics in Strings vorzuhalten und einzeln vergleichen zu müssen, macht eine Abbildung durch Knoten in Graphen oder Baumstrukturen Sinn, da damit nur Teilstrings und Pfade verglichen werden müssen, was in der Auswertung schneller funktioniert.

Wichtig für das Abbilden der Regel in erster Linie ist das Matchen und die damit verbundene Datenstruktur und der Algorithmus. Bevor man in die Umsetzung der Strukturen geht, macht es Sinn, diese Strukturen formal, ohne den direkten Einfluss der Technik zu modellieren.

Generell lassen sich Topic Parts der Topics als Knoten in einem Graphen oder Baum modellieren. Die Topic-Struktur ist generell eine baumartige. Die Frage ist nun die, ob sich eher die baumartige oder eher die graphenartige Struktur zum Suchen der Topics in der Struktur eignet.

```
VR/Haus1/EG/Anmeldung/LichtLed
VR/Haus1/EG/Anmeldung/Tisch
1zu18/Haus1/EG/Anmeldung/LichtLed
1zu18/Haus1/EG/Wohnzimmer/Tisch
1zu18/Haus1/Alarmanlage
```

Die obenstehende Struktur ist eine baumartige, wenn man diese nun in eine generelle Graphen-Struktur überführt, lassen sich Doppelungen reduzieren. Wichtig bei der Überführung ist, dass das Graph gerichtet sein, muss da die Notation in Textform von links nach rechts gelesen wird. Würde man einen ungerichteten Graphen nutzen, gäbe es dadurch Informationsverlust, der z.B. auch die Interpretation von `Alarmanlage/Haus1/1zu18` ermöglicht. Der Graph, der aus den vorhergegangen Überlegungen entsteht, ist folgender Abbildung 26.

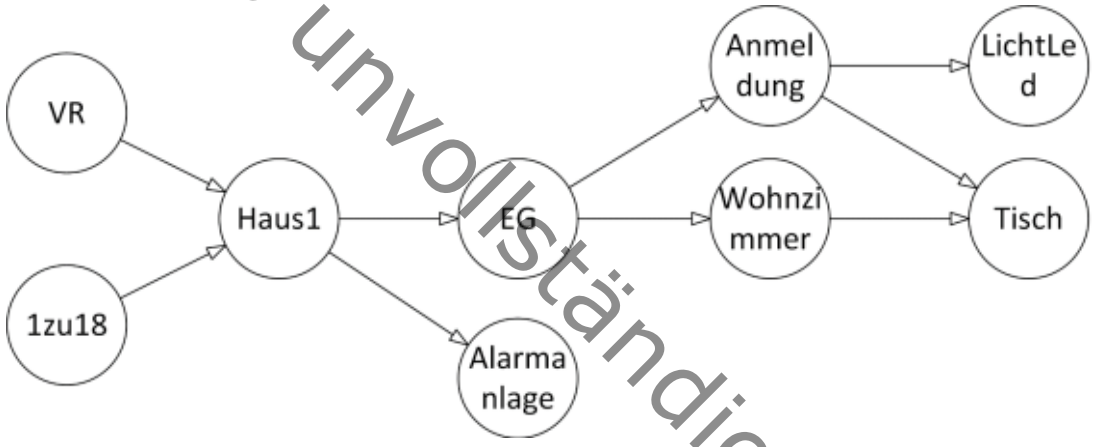


Abbildung 26: Topics naiv in einen Graphen überführt

Die Abbildung der Regel-Topics in den Graphen soll dafür sorgen, dass einfacher und schneller ein Match gefunden werden kann. Dafür muss untersucht werden, ob die Abbildung in dem Graphen, dieselbe Aussage hat, wie die der Topics.

An dem Beispiel kann man erkennen, dass die Knoten mit demselben Namen nicht dieselben Knoten sind. Wie man in Abbildung 26 zieht weisen sowohl VR und 1:18 auf Haus1, aber 1:18 hat in den Topics keine weitere Verbindung zu Alarmanlage, obwohl diese durch aus durch den Graphen ablesbar wären. Damit macht ein Graph, in der Form wie dieser erzeugt wurde, keinen Sinn. Das Haus1 in VR hat wohl eine andere Ausprägung als Haus1 in 1:18.

In der Sprache der ER-Modellierung oder der Modellierung von Datenbank-Modellen, ist das Label nicht der Schlüssel für den Knoten, da dieser den Knoten nicht eindeutig beschreibt. Der Knoten ist in diesem Fall von seinem Vorgänger Knoten abhängig, denn die „Topic-Parts“ unterscheiden sich dadurch welcher „Topic-Part“ vorangegangen ist. Auf das Beispiel oben bezogen, ist das Haus1 aus der VR(-Welt) nicht dasselbe Haus1, wie das aus dem 1:18-Modell. In ER-Modelliert sähe das wie in Abbildung 27 beschrieben aus.

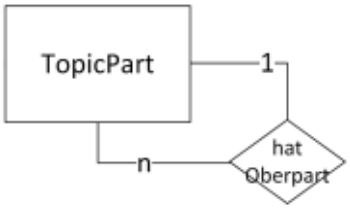


Abbildung 27: ER-Modell zum Abbilden von TopicParts, die Oberparts (bzw. Vorgänger-Knoten) haben

Wenn man nun eine Struktur wie diese in Abbildung 27 wieder in einen Graphen überführt, hat man automatisch eine Baumstruktur. Dasselbe erhält man, wenn man die Topic Beschreibung direkt in eine Baumstruktur überführt (siehe Abbildung 28).

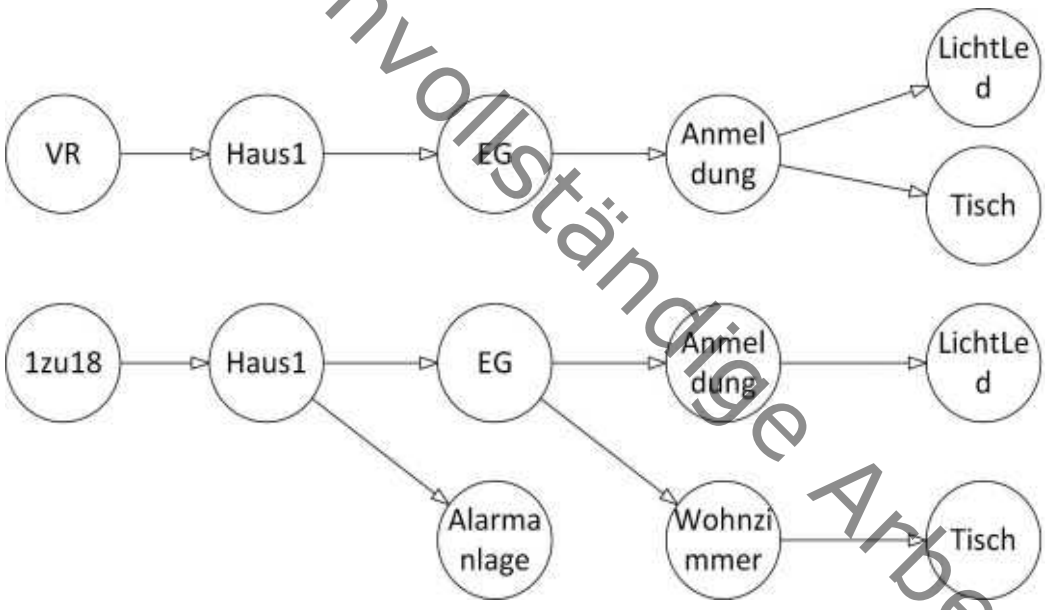


Abbildung 28: Topics in eine Baumstruktur überführt

Mit den Bedenken der Datenmodellierung, kann man die Frage stellen: „Sind die Knoten mit demselben Namen (Label) in dem Modell redundant?“ Diese Frage sorgt dafür, wie ein Unit-Test, nochmal eine Überprüfung zu haben. Die Antwort darauf ist: „Nein, die Knoten sind nicht redundant, denn ein Knoten wird nur über die Kombination von Vorgängerknoten und Label eindeutig identifiziert.“

Aus der Modellierung des Sachverhalts in Abbildung 26, Abbildung 27 und Abbildung 28 ist nun eine Struktur entstanden, die sowohl in einer Relationalen Datenbank als auch in er Objektorientierten Datenstruktur abbildbar ist. Die konkrete Ausgestaltung wird in den folgenden Kapiteln behandelt.

4.4.5 Abbilden der Daten in Datenbank

Ähnlich wie das Abbilden der Topics in Knoten-Modelle lassen sich die Daten in Modelle abbilden, ohne zu stark auf die Implementierung einzugehen, um unabhängig von der konkreten Implementierung den Sachverhalt dazustellen. Die Darstellung geschieht in der Chen-Datenbanknotation (Peter Pin-Shan Chen 1976) in Abwandlung durch das genutzte Tool[11]. Diese Darstellung wurde gewählt, obwohl sie von der in dieser Arbeit stark nutzen UML-Sprache abweicht, da diese Notation Teil der Vorlesungen ist, die ein großer Teil der Stakeholder besuchen oder kennen (siehe Kap. 2.1 Stakeholder). Im Folgenden wird das Modell ER-Modell genannt (Entity Relation Modell).

Folgendes ER-Modell ergab sich aus mehreren Iterationen aus den Funktionalen Anforderungen (siehe Kap. 2.4). Um die Übersichtlichkeit zu bewahren wurde auf das Darstellen der Attribute verzichtet.

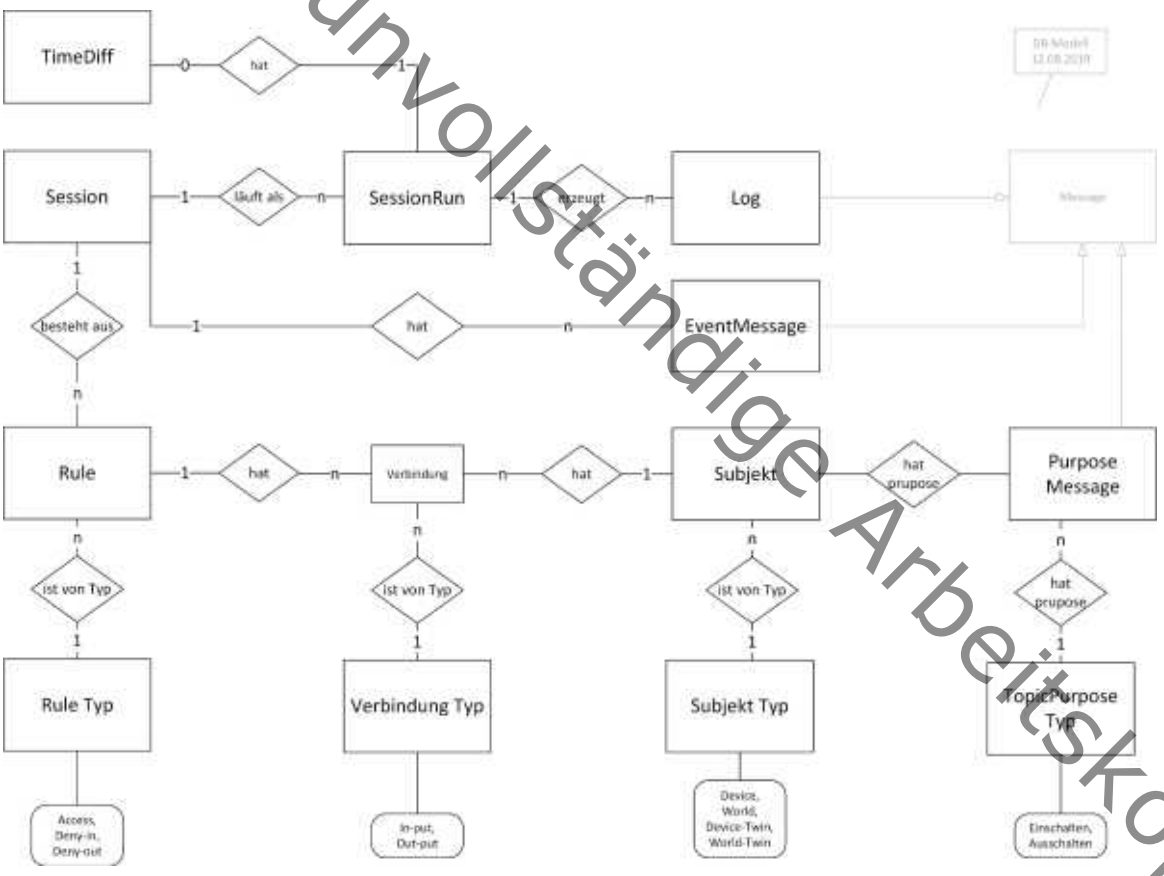


Abbildung 29: ER-Modell der Mittelschicht

[11] Mircosoft Viso Professional 2016 / Shapes: Chen-Datenbanknotation

Die Session ist die Entität, die die Konfiguration eines Durchlaufs hält. Die Session wird durch den Versuchsleiter eingestellt und genutzt. Das Starten der Session geschieht über die SessionRun. Die SessionRun sind Durchlaufe der Session. Die Relation von Session und SessionRun ermöglicht es, dieselbe Konfiguration mehrfach zu durchlaufen und für jeden Durchlauf über TimeDiff eine eigene Zeit und einen eigenen Log zu nutzen.

Über EventMessage werden Events, wie das Festlegen von Werten für einen bestimmten Zeitraum, abgebildet Anforderung 12 (vgl. Abbildung 78, Abbildung 79).

Über die Rule werden die Regeln abgebildet, die durch den RuleService umgesetzt werden. Wie angegeben, ist eine Rule von dem Typ Access, Deny-In oder Deny-Out. Regeln können allein existieren oder sind Subjects zugeordnet, die jeweils den Typ Device, World, Device-Twin oder World-Twin haben. Über diese Subjects kann man dann Rule einschalten, die dafür sorgen, dass Welten oder Devices zu Zwillingen verbunden werden. Ebenfalls lässt sich durch das Ausschalten auch das Trennen von Devices vornehmen, indem die entsprechenden Regeln für das Anbinden des Devices ausgeschaltet werden. Damit das Device in keinem undefinierten Zustand stehen bleibt, gibt es die PurposeMessages. Diese können mit dem Einschalten oder Ausschalten abgesetzt werden, damit das Device korrekt ein oder ausgeschaltet ist.

In der Abbildung 29 leiten die Entitäten von Log, EventMessage und PurposeMessage von der hell dargestellten Entität Message ab. Diese Ableitung stellt dar, dass diese Entitäten alle Attribute von Message besitzen. Die Ableitung wurde im umgesetzten Physischem Modell nicht als Ableitung also als „Is-A“ oder als 0-zu-1 Ableitung umgesetzt, sondern die Attribute sind direkt in den Tabellen enthalten.

4.4.6 Service Start über Controller

Durch die Wahl der Grundstruktur der API-App, der Entscheidung, der Trennung der Oberfläche vom Backend, sowie der Festlegung der Services ergibt sich, dass über die REST-API, die Services gesteuert werden müssen. Die REST-API ist die Schnittstelle zur Oberfläche, die der Versuchsleiter nutzt, bzw. die Schnittstelle, die durch autorisierte 3rd-Party-Systeme genutzt werden soll.

Aus der Datenmodellierung ergeben sich die Controller zu den Models. Jede Entität hat ein Model und damit einen Controller. Die Typen im Datenmodell haben keinen Controller, da die als Enum umgesetzt sind. Die Services erlauben den Ablauf des Szenarios im SMILE-Kontext, damit muss man nun den Controller die Services zuordnen.

Der SessionRunController übernimmt die CRUD-Aufgaben für das Model SessionRun und die SessionRun startet die Session. Solange die SessionRun aktiv ist, läuft eine Session. Dadurch startet der SessionRunController bei einem ankommenden SessionRun-Model mit Attribut aktiv=true die Services und stoppt diese mit Attribut aktiv=false. Die Services, die durch den SessionRunController angebunden werden, sind: EventService, LogService, RuleService und der TimeService.

Die Controller RuleController, TimeDiffController und EventMessageController verhalten sich alle sehr ähnlich, sie übernehmen die CRUD-Aufgaben für das Model Rule, TimeDiff oder EventMessage und sorgen für ein Update der Daten im eigenen Service, wenn das Model hinzukommt, sich ändert, oder gelöscht wird.

Der TimeDiffController übernimmt die CRUD-Aufgaben für das Model TimeDiff und sorgt für ein Update der Daten im TimeService, wenn die Zeit geändert wird.

Eine Ausnehme unter den Controller bildet der `MqttController`, der keine CRUD Aufgaben übernimmt, sondern direkt die Daten des Models `Mqtt` an den `MqttService` weitergibt, der wiederum die `MqttMessage` direkt absendet.

4.5 Strukturansicht der Mittelschicht

Die Architektur eines Softwaresystems besteht aus Strukturen (Bass, Clements, and Kazman 2012). Diese Architektur und die Strukturen sorgen für Beherrschbarkeit und Verständlichkeit (Starke 2011). Um die Beherrschbarkeit und Verständlichkeit zu erreichen, basieren die Strukturen in der Mittelschicht auf Strukturen von bereits bekannten Softwaresystemen (siehe Kapitel 4.3.2.1). Software-Architektur ist nach Tom DeMarco ein „framework for change“ und damit ein Rahmen der Flexibilität und Erweiterbarkeit von Software sicherstellt.

4.5.1 Dependency Injection

Die Dependency Injection (DI) ist eine Technik, die Kopplung von Objekten verringert indem es einem Objekt die Abhängigkeiten eines anderen Objekts liefert. Eine "Abhängigkeit" ist ein Objekt, das z.B. als Service verwendet werden kann. Anstatt dass der „Client“ angibt welchen Service er verwendet, gibt der Injektor dem „Client“, einen Service, der den Anforderungen entspricht (vgl. (wikipedia 2019a)). DI ist ein Muster des SOLID-DIP: Dependency Inversion-Prinzip (siehe Kapitel 2.7.5) und sorgt daher für eine höhere Flexibilität, da nun Services einfach ausgetauscht werden können.

ASP NET Core unterstützt die DI und gibt eine leicht geänderte, aber funktionale Lösung vor (Latham et al. 2019). In der Lösung fragt der „Client“ das Interface der Abhängigkeit über den eigenen Konstruktor an und bekommt die konfigurierte Abhängigkeit für dieses Interface. Die Konfiguration welches Interface welche konkrete Implementierung nutzt, wird an einem zentralen Ort vorgenommen: in der Klasse `Startup` in der Methode `ConfigureServices`. Dadurch, dass die Abhängigkeit abstrahiert ist, kann man nun einen Service austauschen und braucht nur die Änderungen einer Zeile in der Klasse `Startup` durchzuführen.

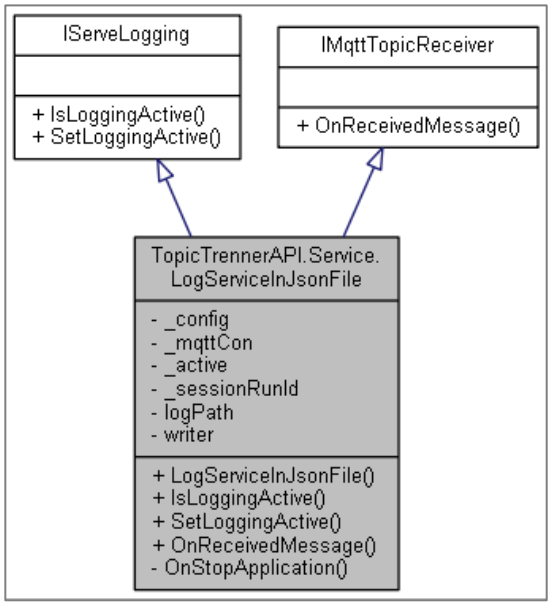


Abbildung 31: LogServiceInJsonFile eine Implementierung von IServeLogging

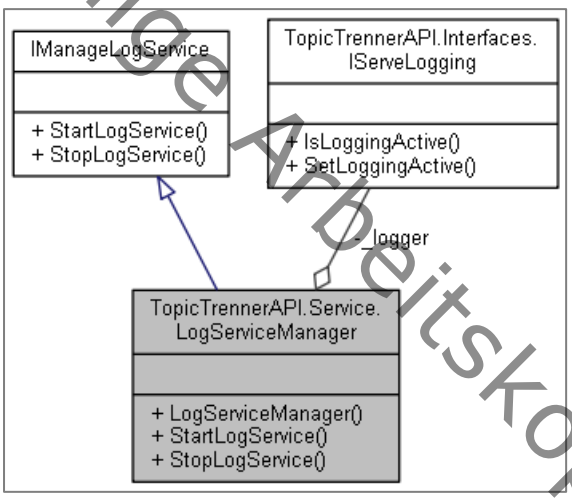


Abbildung 30: LogServiceManager eine Implementierung von IManageLogService, die IServeLogging nutzt

Die Abstraktion auf Interfaces sorgt für eine gute Austauschbarkeit und die Erfüllung des SOLID-OCP: Open Closed-Prinzip und des SOLID-LSP: Das Liskov'sche Substitutionsprinzip (siehe Kapitel 2.7.3 und 2.7.2) aber auch dafür, dass die Graphen nicht auf die direkten Implementierungen verweisen, sondern auf ihre Interfaces, wie in Abbildung 31 und Abbildung 30 zu sehen. In Abbildung 30 sieht man, dass `IServeLogging` verwendet wird, jedoch nicht welche Implementierung. Dass die Graphen nicht direkt auf die Implementierung verweisen, kann unerfahrene Entwickler verunsichern. Letztlich sollte aber die konkrete Implementierung des Interfaces während der Implementierung einer anderen Klasse soweit egal sein, dass man sich auf das Verhalten stützen kann, das über das Interface vorgegeben ist. Wenn es trotzdem wichtig sein sollte, zu wissen welche konkrete Implementierung aktuell verwendet wird, kann man diese Information zentral in der Klasse `Startup` finden.

4.5.2 Services als Singleton

Wie bereits in vorangegangenen Kapiteln dargestellt, wird die API-App Struktur um Services erweitert. Diese Erweiterung wird durch Dependency Injection (DI) durchgeführt. Einige Aufgaben dürfen aber nicht mehr als einmal erfüllt werden, da es sonst zu Doppelungen oder anderen Problemen, bei der Arbeit mit Nebenläufigkeit, führen kann.

Das Singleton Pattern der Gang of Four (vgl. Singleton (Gamma et al. 2001)) löst genau diese Probleme, hat aber jeweils Herausforderungen und Grundbedingungen, denen man begegnen muss. Diese Punkte wurden in dem Zusammengang mit Unity bereits diskutiert, sind aber auch hier anwendbar (siehe 4.2.2.2 Singleton-Like Ansatz).

Im Zusammenhang mit DI kann man auch einen Service bereits durch die Konfiguration zum Singleton machen. Dabei muss dennoch darauf geachtet werden, dass nicht mehr Services als nötig Singleton werden. Es wurden bereits 5 Services erkannt:

- MQTT-Service
- Event-Service
- Log-Service
- Rule-Service[12]
- Time-Service

Alle 5 Services sind notwendigerweise Singleton:

Der MQTT-Service muss Singleton sein, damit es nur einen MQTT-Connector gibt, der MQTT-Nachrichten entgegennimmt und verteilt. Argumente für einen MQTT-Connector sind im Kapitel 4.2.2.1 Zentraler-MQTT-Connector (ZMC) zu finden. Die Argumente der Doppelung und des Single Point of Truths lassen sich auf alle Services gleichermaßen anwenden.

4.5.3 Dependency Inversion

Aus der Lösung, die Services als Singleton zu instanziieren, ergibt sich die Herausforderung im Design, dass diese Services keinen Zugrifft auf andere Klassen haben dürfen, die nicht Singleton

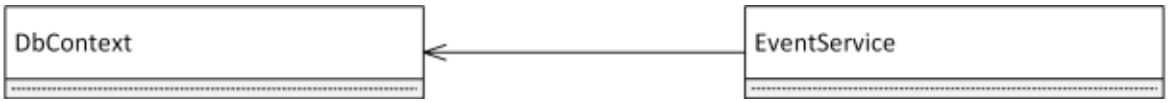


Abbildung 32: Klassendiagramm EventService hält eine Referenz auf DbContext

[12] Später auch RuleEvaluatioService genannt

sind. Singleton dürfen nur Zugriff auf Klassen haben, die selber bereits Singleton sind. Wenn eine Singleton Instanzen von anderen Klassen hält, existieren diese Instanzen solange wie das Singleton und können damit nicht vom Garbage Collector aufgeräumt werden, das unterminiert das Speichermanagement. Damit das nicht unsichtbar geschieht, dürfen Singletons nur Instanzen von anderen Singletons halten.

Aus dem o.g. Punkt ergibt sich das Problem, dass die Services Event-Service, Rule-Service und Time-Service Daten von dem `DbContext` brauchen, um zu arbeiten. Intuitiv würden, die Services den `DbContext` über DI anfordern und als Instanz halten, um die Daten zu laden, wenn es notwendig ist. Die Abhängigkeit der Klassen zueinander sähe dann wie in Abbildung 32 dargestellt aus. Der `DbContext` darf aufgrund der internen Struktur von ASP.NET nicht Singleton werden, damit können die Services die Instance des `DbContextes` nicht anfordern und haben damit auch keinen Zugriff auf die Daten der Datenbank.

Wie aus der Beschreibung der Verfahren bekannt ist werden die Daten der Services nur über die REST-Schnittstelle geändert[13]. Da die Daten aus einer Definierten Richtung geändert werden, sind die Umstände bekannt, unter den die Daten geändert werden. Aktuell werden die Daten immer dann geändert, wenn der Controller der API-App Struktur angesprochen wird und dieser die Daten an den `DbContext` weiterleitet.

Damit die Services an die Daten gelangen muss eine Dependency Inversion (siehe Kapitel 2.7.5) durchgeführt werden. Die Abhängigkeitsrichtung wird invertiert, indem der Controller eine Instanz von `DbContext` und eine Instanz des Service hält. Der Controller liefert die Daten dann an den Service (siehe Abbildung 33).

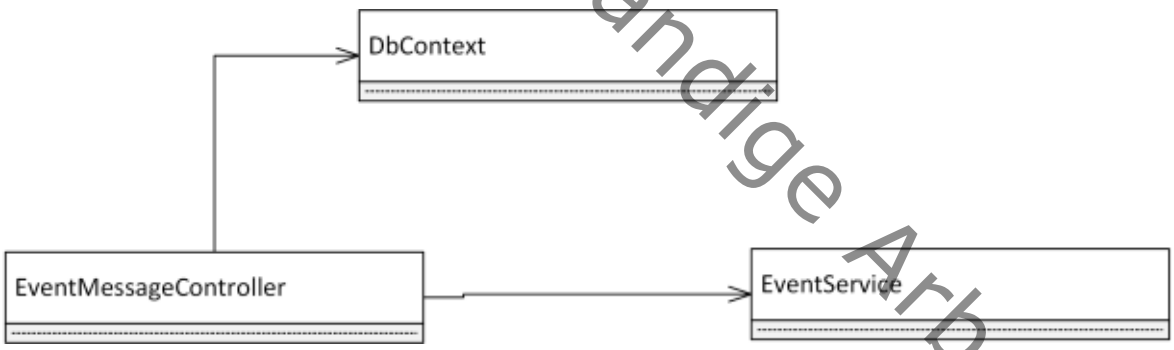


Abbildung 33: Klassendiagramm Dependency Inversion EventMessageController, EventService, DbContext

[13] In Zukunft kann auch eine andere Schnittstelle genutzt werden, für diese Betrachtung ist es einfacher von der vorhandenen Struktur auszugehen.

4.5.4 Controller Manager Service

Durch die Dependency Inversion des vorhergegangenen Kapitels sind die Services angebunden und können als Singleton laufen. Die Koppelung der Services an die Controller ist jedoch ungünstig, da die Controller dann den Services zuarbeiten und nicht nach dem Single-Responsibility-Prinzip nur eigene Aufgaben erfüllen. Speziell in dem Fall eines `SessionRunControllers`, der bei aktiv schalten einer `SessionRun` mehrere Services einschalten und mit Daten füllen muss, sorgt das für komplizierte Strukturen. Um diese Strukturen zu vereinfachen, wird eine Zwischenschicht eingeführt: die Manager. Die Manager liegen zwischen Controller und Service und laden die Daten aus dem `DbContext` und geben die Daten an die Services weiter. Damit ist der Controller entlastet und erfüllt nur eine Aufgabe und „delegiert" das Laden und aufbereiten der Daten an den Manager. Da jeder Service ein eigenes Datenset braucht, bekommt jeder Service einen eigenen Manager, der wiederrum durch mehrere Controller genutzt werden kann.

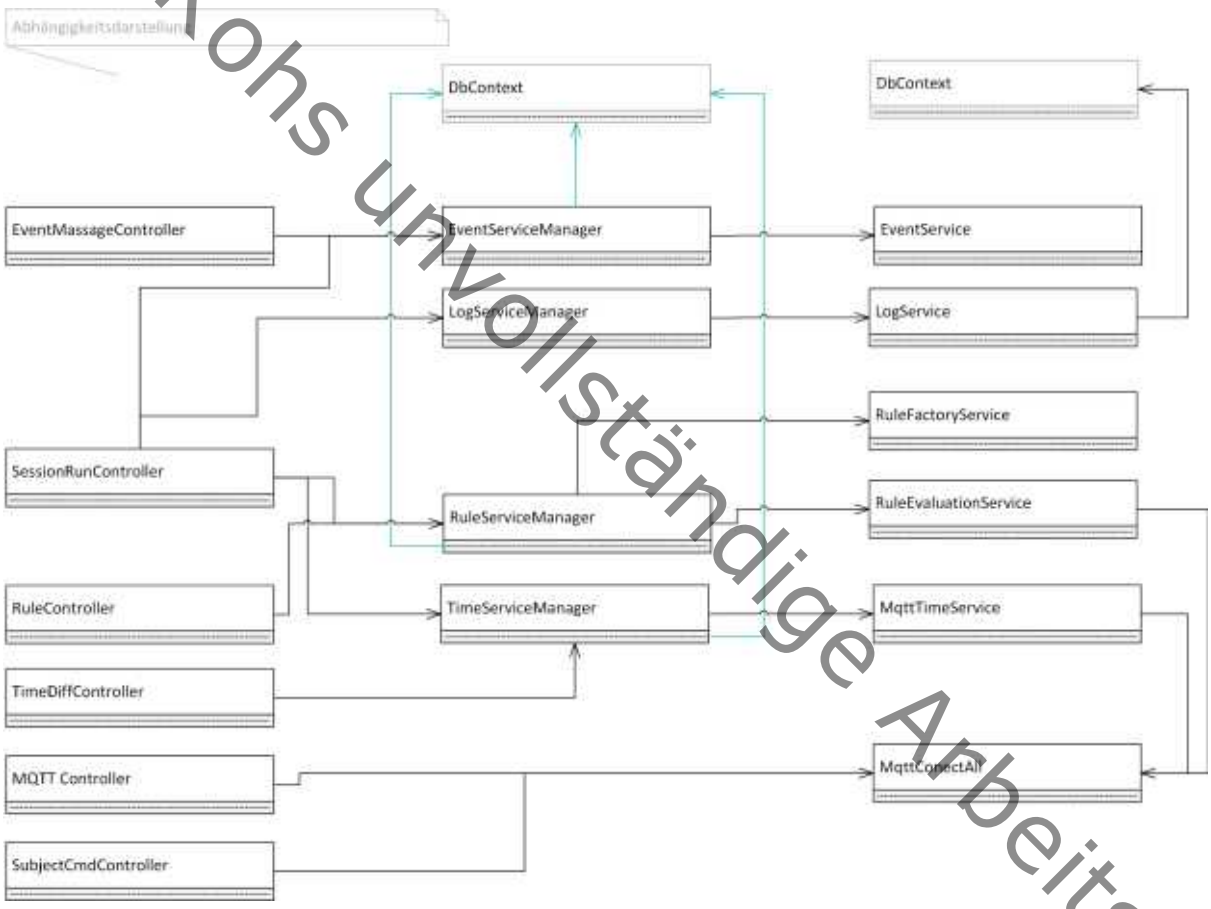


Abbildung 34: Klassendiagramm Controller Manager Service Abhängigkeitsdarstellend

Die bei der Einführung der Manager-Schicht entstehende Struktur ist in Abbildung 34 dargestellt. In dieser Abbildung wurden die Verbindungen der Controller zum `DbContext` aus Gründen der Übersicht entfernt. Aus Gründen der Konsistenz wurden die Verbindungen zwischen den Klassen als abhängigkeitsangebende Pfeile dargestellt. Das Original ist im Anhang als Abbildung 49 zu finden.

4.5.4.1 Controller Manager Service Muster

Aus der entstehenden Struktur lässt sich ein Muster ableiten, was angewandt wird, wenn ein Singleton-Service in einer API-App zugrifft auf die Daten in DbContext benötigt (Abbildung 35).

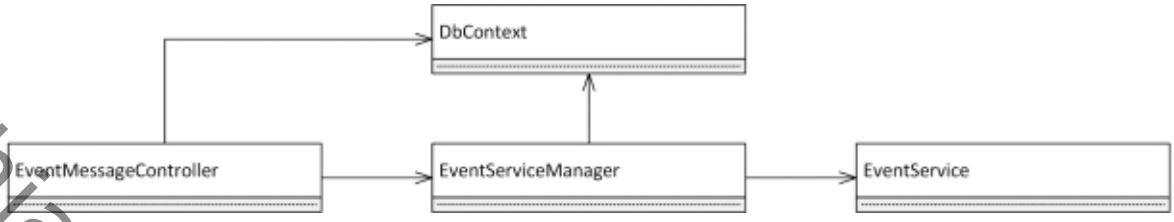


Abbildung 35: Klassendiagramm Controller Manager Service Muster am Beispiel des EventService

Vier Aktoren mit jeweils einer Aufgabe:

`DbContext`	stellt die Verbindung zu Datenbank.
`Controller`	nimmt die Anfragen der API-Schnittstelle an, verarbeitet CRUD-Aufgaben mit dem DbContext und delegiert die weiteren Aufgaben den Manager.
`Manager`	nimmt die Anfragen der Controller entgegen, lädt notwendige Daten aus dem DbContext und liefert aufbereitete Daten an den zu managenden Service.
`Service`	nimmt die Daten und Anweisungen des Managers entgegen und bietet selbständig einen Service an.

Die Bedingungen dieses Musters lassen sich weiter verallgemeinern: Dieses Muster lässt sich nutzen, wenn ein Singleton-Service in einer API-App zugrifft auf die Daten eines Services benötigt, der nicht selber Singleton werden darf (Abbildung 35).

Wie in Abbildung 34 zusehen, passen vier Services auf dieses Muster: `EventService`, `LogService`, `RuleEvaluationService`, `MqttTimeService`.

`MqttConectAll` (zuvor `MqttService` genannt) ist ein Service, der als Singleton arbeitet, jedoch braucht dieser keinen Zugriff auf weitere Daten, und entspricht damit nicht dem Muster. Ebenso verhält es sich mit dem `RuleFactoryService`, dieser Service erfüllt auch nicht alle Anforderungen für das Muster, denn dieser Service ist nicht Singleton. Denn der Dienst, Daten aus der Datenbank in Rules zu verwandeln, darf mehrfach in dem System vorhanden sein. Dieser Service wird durch den `RuleServiceManager` genutzt, der die Daten für den `RuleEvaluationService` aufbereiten lässt.

4.6 Herausforderungen in der Anwendung

Während der Entwicklung der Mittelschicht haben sich Herausforderungen ergeben, die noch weiter betrachtet werden müssen. Einige Herausforderungen können technisch gelöst werden, andere müssen organisatorisch gelöst werden. Dieses Kapitel gibt einen Überblick über bekannte Herausforderungen und potenzielle Lösungen, die aber im Rahmen der Masterarbeit nicht mehr gelöst werden können.

4.6.1 Kreisschlüsse im MQTT-Netzwerk

Es sind Kreisschlüsse durch Rules möglich, ähnlich wie in vielen Netzen ist es möglich Kreise zu binden. In diesem Fall kann man Kreise bauen, die mit MQTT-Nachrichten weitere MQTT-Nachrichten triggern und damit einen Kreis bilden.

Diese Kreise sind sowohl jeweils über OpenHAB und über den Topic Trenner umsetzbar und über die Kombination beider Systeme. In der Anwendung können diese Kreise zu hoher Last oder sogar zum Absturz von den Beteiligten Systemen führen oder das Gesamtsystem lahmlegen.

4.6.1.1 Lösungsvorschlag Kreiserkennungskomponente

Eine Lösung für die Kreiserkennung innerhalb der Mittelschicht könnte eine Kreiserkennungskomponente sein. Durch den Modularen Aufbau würde ich das problemlos nachrüsten lassen. Da der Topic Trenner alle Topic Rules als Knoten darstellt, kann man diese Knoten verbinden indem man vom `InTopic` zum `OutTopic` eine gerichtete Kante einfügt. Auf den dadurch entstandenen Graphen lassen sich Graphen-Algorithmen anwenden, die Kreise erkennen. Würde man diese Kreiserkennung bei jeder Änderung einer Regel in einer aktiven `SessionRun` oder vor dem Starten eines `SessionRuns` durchführen, könnte man Kreise, die durch die Mittelschicht alleine erzeugt werden, erkennen und verhindern.

4.6.1.2 Lösungsvorschlag Kreiserkennung durch Heuristiken

Man kann Heuristiken aufstellen, die einen aktiven Kreis detektieren, da Kreise sehr schnell ein und dieselbe Nachricht wiederholt im Kreis senden. Diese Heuristik könnte man über einen Service in der Mittelschicht Struktur umsetzen. Dieser Dienst könnte Alarm schlagen oder direkt die betroffene Regel unterbrechen. Heuristiken in System sind mit „Vorsicht zu genießen“, da man Erfahrungswerte benötigt, um Schwellwerte einzustellen. Da die Nutzung der Mittelschicht noch weitegenug vorgeschritten ist, wird das Ansatz der Heuristiken in dieser Arbeit nicht weiterverfolgt.

4.6.2 Regeln mit Einfluss auf mehrere Devices und Welten

Regeln können mehrere Devices und Welten beeinflussen, daher kann man die Regeln im Datenmodel auch mehreren Subjekten (Devices / Welten) zuordnen. In der Benutzung durch den Versuchsleiter kann das Abschalten von Regeln in der Mittelschicht über das Ausschalten von Subjekten auch andere Subjekte beeinflussen. Diese Nebeneffekte könnte man darstellen, wenn der entsprechende Fall modelliert wird. Dieser Punkt wird aus zeitlichen Gründen als weiterer Forschungsbedarf deklariert, da für die Lösung dieses Problems sowohl Versuche mit Probanden, die Versuchsleiter darstellen notwendig werden als auch die Entwicklung von unterschiedlichen Lösungsmöglichkeiten, die voraussichtlich nicht trivial sind.

4.6.3 Sonderfall Geräte ohne MQTT-Anbindung

Aktuell kann die Mittelschicht nur Geräte mit MQTT-Anbindung verbinden, die den Großteil der Geräte in der SMILE-Landschaft darstellen. Es existieren Geräte wie smarte Lampen, die das MQTT-Protokoll nicht nutzen.

4.6.3.1 Versuchsvorschlag Implementieren der Protokolle

Die Mittelschicht ist durch den DI-Ansatz und den Controller Manager Service Ansatz flexibel im Umgang mit neuen Diensten. Durch diese Flexibilität wäre es möglich Protokoll abzubilden, die von der Logischen Arbeitsweise mit MQTT kompatibel sind. Dafür muss extra Zeit in die Untersuchung der Kompatibilität investiert werden und der Ansatz prototypisch umgesetzt werden damit dieser untersucht werden kann. Diese Lösung aus zeitlichen Gründen in dieser Arbeit nicht mehr möglich.

4.6.3.2 Lösungsvorschlag Abbilden der Protokolle durch OpenHAB Regeln

OpenHAB unterstütz viele Protokolle und bietet an, unabhängig der Protokolle Regeln zu definieren, wie sie durch die Arbeit von Steffen Kaiser bereits genutzt wurden (Kaiser 2018). Man kann über diese Regeln ein „virtuelles Device“ definieren, was die Übertragung aus dem echten Gerät auf das „virtuelle Device“ durchführt und umgekehrt. Wenn die Topics des „virtuellen Devices“ den Richtlinien für den Topic Trenner genügen, lässt sich das echte Gerät über die Mittelschicht steuern, verbinden oder abtrennen.

4.7 Bezug zur Forschungsfrage

In dem vorher gegangenen Kapitel wurde die Systementwicklung betrachtet, inklusive der der Richtlinien und Herausforderungen, die durch die Designentscheidungen entstanden sind.

Im Kapitel *MQTT Topic Richtlinien* wurden die Richtlinien ausgewählt, wie mit den MQTT Topic umzugehen ist, da das MQTT-Protokoll und damit die MQTT Topics ein sehr fundamentaler Bestandteil des Gesamtsystems sind. Auf der Grundlage, dass die MQTT Topics in dieser Arbeit eine berechenbare Form haben, werden Designentscheidungen getroffen.

Die *MQTT Unity3D Anbindung,* im gleichnamigen Kapitel, wurde entwickelt, um in Unity3D entstehende 3D-Modelle oder 3D-Welten in der Modelllandschaft zu betreiben und als Digitale Zwillinge betreiben zu können. Die Anbindung wurde auf Basis einer vorhandenen C#-Library umgesetzt. Die Herausforderung bestand zum einen in der Verbindung der beiden Arten mit Nebenläufigkeit umzugehen. Zum anderen Bestand die Herausforderung darin einen Angenehmen Weg zu finden, diese Funktionalität bereit zu stellen.

Im Kapitel *MQTT-Mittelschicht* wurde basierend auf den Anforderungen und der Technologiewahl Varianten miteinander verglichen, wie man die Modelllandschaft verbindet oder trennt. Die Wahl ist auf den Topic Trenner gefallen, der eine zentrale, unabhängige Lösung ohne zu große Entwicklungsaufwände bedeutet.

Im Kapitel *Verfahrensansicht Mittelschicht* wurden die Hauptverfahren und deren Abbildungen betrachtet, darunter die Detailbetrachtung des Topic Trenner Verfahrens und das Abbilden der Topic Trenner Funktionalität in Regeln, die wiederum in Algorithmen und Datenmodelle übersetzt wurden.

Im Kapitel *Strukturansicht der Mittelschicht* wurden die Strukturgebenden Elemente untersucht. Die Struktur der Mittelschicht unterstützt die Flexibilität des Systems durch die Dependency Injection (DI) und Dependency Inversion (DIv), die zusammen erlauben Services und andere logiktragende Klassen mit wenig Aufwand zu tauschen. Im Rahmen dieses Kapitels ist das Controller Manager Service Muster mit Hilfe von DI und DIv entstanden.

Abschließend wurde im Kapitel *Herausforderung in der Anwendung* auf die Herausforderungen Aufmerksam gemacht, die mit den getroffenen Entscheidungen auftreten.

5 Evaluation des Systems

Die Evaluation erfolgt in drei Schritten, die die unterschiedlichen Anforderungen für die Evaluation des Systems heranziehen. In einem Vorabschritt wurde die Anforderungsabdeckung betrachtet, dieser Schritt ist im Anhang zu finden: 7.3 Anforderungsabdeckung. Im ersten Schritt wird eine Empirische Studie mit Probanden durchgeführt, um das System anhand von Aussagen zu untersuchen. Das Ergebnis dieser Studie ist qualitativer Art und wird durch die Interviews mit den Probanden unterstützt. Im zweiten Schritt wird ein Experten-Interview durchgeführt, um nochmal die Softwarearchitektur und die Arbeit der Mittelschicht als Software zu betrachten. Das Feedback soll dabei helfen die Mittelschicht darauf vorzubereiten, durch andere weiterentwickelt zu werden. Im dritten und letzten Schritt wird die Kombination aus Implementierung und Hardwarewahl durch einen Empirischen Belastungstest durchgeführt. Dieser Test hilft dabei, die Leistungsgrenzen des Systems herauszufinden.

5.1 Empirische Studie mit Probanden

In diesem Unterkapitel wird das Ziel verfolgt, Experimente vorzubereiten, um Aussagen über das Gesamtsystem zu evaluieren. Die Aussagen lassen sich auf die Anforderungen zurückführen. Nachfolgenden wird zunächst die Aussage vorgestellt, um danach die Experimente zu erläutern.

Die Datenerhebung zu den Tests geschieht über die Beobachtung der Probanden während des Durchlaufs und nachgelagertem Interview. Da alle Aussagen durch ja oder nein beantwortbar sind, reicht für diese Betrachtung die Beobachtung. Diese Beobachtung wird durch ein semistrukturiertes Interview[14] ergänzt, was im Kapitel 5.1.7 Feedback-Interview weiter erläutert wird.

Für alle Beobachtungen kann man folgende Beobachtungsfragen stellen:

1. Wird durch den Probanden Hilfestellung erfragt?
    - Was wird gefragt?
    - Wie oft wird gefragt?
    - Ist die Frage Missionskritisch?
2. Macht der Proband Fehler?
    - Welcher Fehler wird gemacht?
    - Wie viele Fehler werden gemacht?
    - Ist der Fehler Missionskritisch?

5.1.1 Aussage: Darstellen von Virtuellen Gegenständen

Aussage: Mit der erarbeiteten Technologie ist das Steuern eines Virtuellen Gegenstandes mit hohem nachsteuerbedarf möglich.

Die Aussage stützt sich auf Anforderungen 39-44, 46 und den Qualitätsanforderungen Benutzbarkeit, Funktionalität, Effizienz (siehe Kapitel: 2. Anforderungsanalyse)

Stakeholder: Proband

[14] In dieser Arbeit wird das semistrukturierte Interview so genutzt, das die Befragten einerseits thematisch gelenkt, aber nicht eingeschränkt werden. So kann dynamisch auf Anregungen eingegangen werden und die Möglichkeit einer Skop-Erweiterung in Feedback bleibt erhalten bleiben.

Erläuterung zur Stakeholderwahl: Mit der Aussage wird der Stakeholder Proband adressiert, da der Proband das durchläuft, was ein Proband in der Nutzung des Exponats der SLE auch durchlaufen würde (siehe Kapitel: 2.1. Stakeholder).

5.1.1.1 Konzept zum empirischen Evaluieren der Aussage

Versuchsgegenstand: Fliegen einer Modell-Drohne in der Perspektive eines Modellflugzeug-Piloten mit direkt angebundenen und über MQTT angebundenen Kontrollern.

Evaluierungsfragen:

1. Ist die Testimplementierung gut genug, damit ein Proband störungsfrei durchkommt?
2. Ist die Implementierung schnell genug, damit eine Drohne, auf der Hardware, im virtuellen Raum über MQTT-gesteuert werden kann?
3. Ist die Implementierung schnell genug, dass der Proband den Unterschied nicht wahrnimmt?

Probanden: Als Probanden eignen sich Personen, die sich als technikaffin bezeichnen.

Aufgabenstellung an die Probanden: Sie sind nun ein Testpilot, der drei Steuersysteme gegeneinander untersuchen muss, Sie bekommen dazu drei Steuersysteme in die Hand. Sie fliegen drei Tracks je mit einem Controller. Sie haben drei Runden Zeit, sich mit der Steuerung der Controller vertraut zu machen, nach den drei Runden fliegen Sie drei Runden auf Zeit.

5.1.1.2 Einordnung in die Gesamtarchitektur

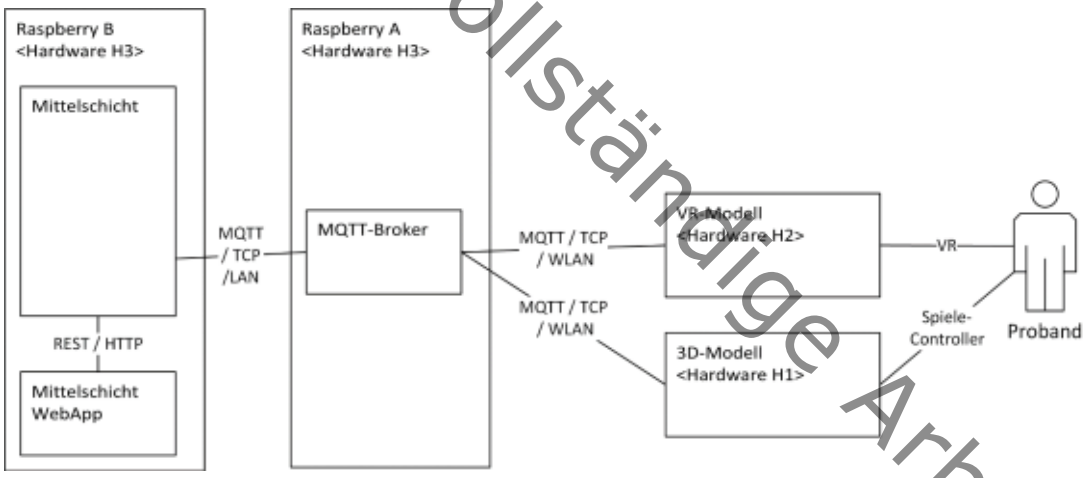


Abbildung 36: Teil des Meta-Modells des Gesammtsystems zum aktuellen Stand, mit Fokus auf eine Konfiguration, die das Eingeben über einen Computer und das Darstellen mit einem anderen über das MQTT-System erlaubt.

Der Proband sieht über das VR-Modell auf der Hardware H2 eine Szene und gibt Befehle über den Spiel-Controller an das Gesamtsystem. Der Spiel-Controller ist an Hardware H1 angeschlossen, die mit er Mittelschicht über MQTT-Nachrichten kommuniziert. Der Broker verteilt die Nachrichten und die Mittelschicht sendet die Nachrichten über den Broker zurück oder blockiert die Nachrichten. Das VR-Modell interpretiert die ankommenden MQTT-Nachrichten und gibt die Steuerbefehle an die geflogene Drohne weiter. Als alternativ Controller wird der Spiel-Controller direkt an H2 angeschlossen, was ein direkteres Feedback erlaubt, und als weitere Alternative steuert der Proband über die HTC-Vive Handcontroller.

5.1.2 Aussage: SLE-Simulation auf Laborhardware

Aussage: Die Mittelschicht ermöglicht dem Versuchsleiter, die Konfiguration und Durchführung einer Simulation eines SLE auf laborüblicher Hardware.

Die Aussage stützt sich auf Anforderungen 1,2,3,8,9,12-30, 47-51 und den Qualitätsanforderungen Benutzbarkeit, Funktionalität, Effizienz (siehe Kapitel: 2. Anforderungsanalyse)

Stakeholder: Versuchsleiter

Erläuterung zur Stakeholderwahl: Mit der Aussage wird ausschließlich der 'Versuchsleiter' adressiert, da es um die konkrete Durchführung der Simulation geht, für die kein Eingriff in die Software notwendig ist (siehe. Kapitel: 2.1. Stakeholder).

5.1.2.1 Konzept zum empirischen Evaluieren der Aussage

Versuchsgegenstand: Konfiguration und Durchführung einer SLE-Simulation durch einen Versuchsleiter auf laborüblicher Hardware.

Evaluierungsfragen:

1. Kann ein Versuchsleiter eine Simulation konfigurieren und durchführen?
2. Ist diese Simulation auf laborüblicher Hardware durchführbar?
3. Wie hoch ist die Arbeitszeit für die Konfiguration?

Probanden: Als Probanden eignen sich Personen, die sich als technikaffin bezeichnen und die Mathematische Formeln mit zwei Variablen und einfache Wenn-Dann-Beziehungen auflösen können.

Aufgabenstellung an die Probanden: Sie sind ein Versuchsleiter, der jetzt eine neue Session aufbaut. Die Session ist für das Thema Smart-Living gedacht. In diesem Versuch werden Sie zwei Häuser modellhaft als Zwilling zusammenschalten und den Versuch Stück für Stück prüfen. Sie bekommen die Datenblätter der Häuser.

5.1.2.2 Einordnung in die Gesamtarchitektur

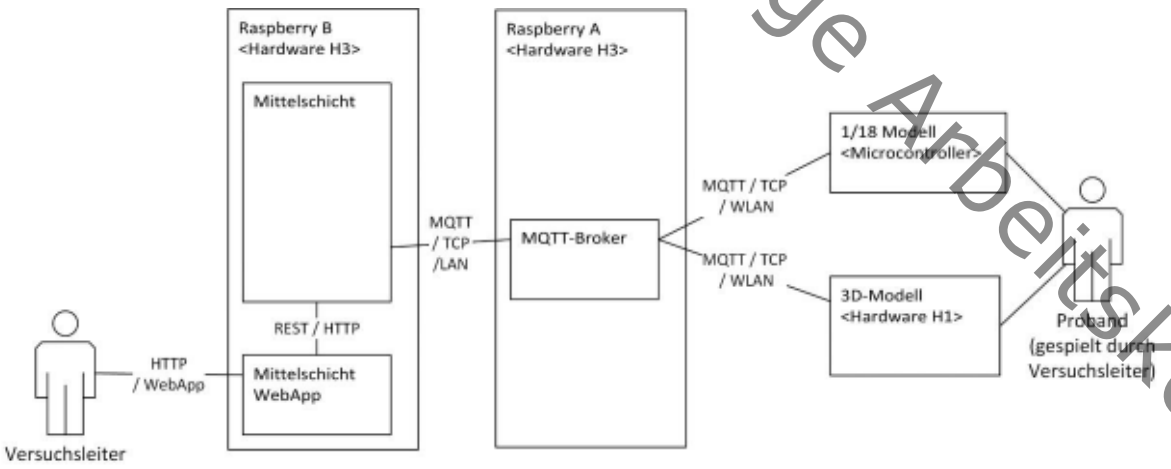


Abbildung 37: Teil des Meta-Modells des Gesamtsystems zum aktuellen Stand, mit allen Teilen, die für diesen Test von Digitialen Zwillingen gebraucht werden

Der Versuchsleiter stellt mit der WebApp der Mittelschicht die Mittelschicht ein und kontrolliert die Einstellungen, indem der Versuchsleiter den Probanden spielt. Eingaben auf von dem 1:18 Modell sollen in dem 3D-Modell sichtbar werden und anders herum. Die MQTT-Nachrichten werden über den MQTT-Broker verteil und durch die Mittelschicht weitergeleitet oder gestoppt. Ob Nachrichten weitergeleitet werden und an welches Topic, wurde durch den Versuchsleiter definiert, in dem dieser über die WebApp Regeln definiert hat.

5.1.3 Aussage: Entwickeln neuer Anwendungen

Aussage: Das ZAS ermöglicht es Entwicklern neue Anwendungen an das Gesamtsystem anzubinden.

Die Aussage stützt sich auf Anforderungen 31, 32, 58 und die Qualitätsanforderungen: Funktionalität, Übertragbarkeit, Änderbarkeit (siehe Kapitel: 2. Anforderungsanalyse).

Stakeholder: Entwickler, Projektpartner, Systemarchitekt

Erläuterung zur Stakeholderwahl: Mit der Aussage werden 'Entwickler' deren Ableitungen adressiert, da es um die Benutzung und Adressierung der REST-Schnittstellen der Mittelschicht geht. Diese Schnittstelle dient der Anbindung der aktuellen Versuchsleiter-Oberfläche, neuer Benutzeroberflächen und 3rd-Party Systeme (siehe Kapitel: 2. Anforderungsanalyse, 2.4.3. 3rd-Party-System - Zentrales-System, 2.5.1. Qualitätsanforderungen).

5.1.3.1 Konzept zum empirischen Evaluieren der Aussage

Versuchsgegenstand: Ansprechen der REST-Schnittstelle der Mittelschicht durch einen Entwickler mit dem Tool Postman.

Evaluierungsfragen:

1. Kann ein definierter Entwickler ein 3rd-Party System oder Benutzerschnittstelle an die Mittelsicht anschließen.

Probanden: Die Probanden sollten Studenten oder Bachelor der Informatik und Wirtschaftsinformatik sein, die mindestens das Modul „Datenbanken und Webtechnologien“ bestanden haben oder ähnliches theoretisches und praktisches Wissen besitzen.

Konkret wird folgendes vorausgesetzt:

- das Verwenden einer REST-Schnittstelle
- das Lesen und Verstehen eines ER-Modells

Aufgabenstellung an die Probanden A:

Sie werden eine Weile nicht zu Hause sein, dennoch wollen Sie eine Regelmäßige Anwesenheit vortäuschen. Nutzen Sie für das Durchführen dieses Szenarios die Mittelschicht-REST Schnittstellen. Bauen Sie eine Session mit Regeln und Events auf, die eine Anwesenheit durch Regeln oder Events ausführt und prüfen Sie das Ergebnis durch ausführen der SessionRun. Ein Beispiel für das Vortäuschen von Anwesenheit kann das Einschalten von Licht um eine gewisse Uhrzeit und das Ausschalten von diesem Licht zu einer anderen Uhrzeit sein. Sie dürfen auch die GUI verwenden, um sich die Arbeit der REST-Schnittstelle herzuleiten.

5.1.3.2 Einordnung in die Gesamtarchitektur

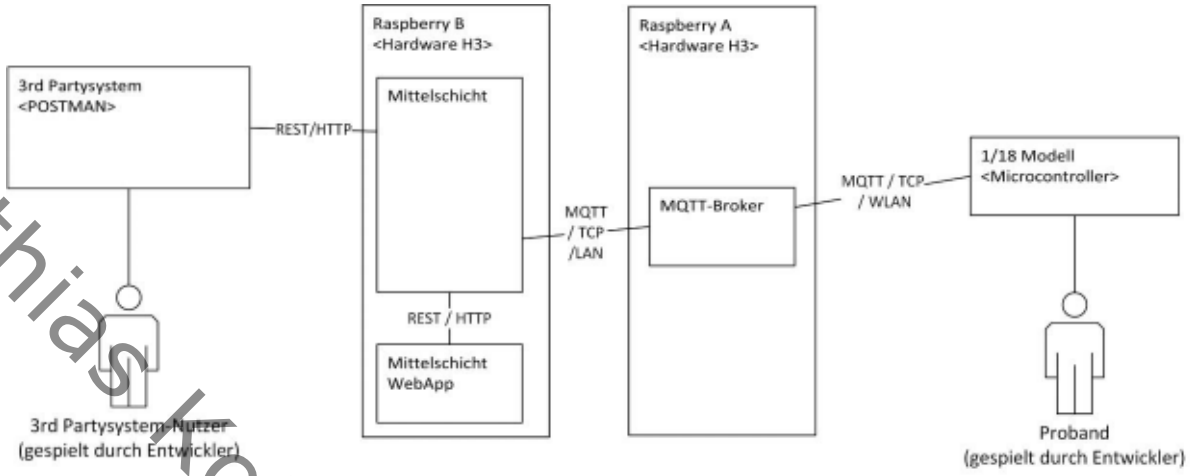


Abbildung 38: Teil des Meta-Modells des Gesamtsystems zum aktuellen Stand, mit allen Teilen, für den Versuch zum Anbinden von neuen Systemen

Der Proband konfiguriert eine Session inklusive Regeln über die REST-Schnittstelle. Er imitiert aus der Perspektive der Mittelschicht ein 3rd-Partysystem oder die Mittelschicht Webapp, die über die REST-over-HTTP-Schnittstelle eine Konfiguration festlegt. Wenn die Schnittstelle korrekt benutzt wird, merkt die Mittelschicht nicht, ob es sich im die Webapp ein 3rd-Partysystem oder ein anderes System handelt. Über diesen Test kann die Erweiterbarkeit des Mittelschicht-Kerns geprüft werden, denn wenn Probanden das System über diese Schnittstelle steuern und konfigurieren können, können sie auch Systeme schreiben, die diese Schnittstelle verwenden (insofern sie der Entwicklung von Clients mächtig sind).

5.1.4 Aussage: Mittelschicht erweiterbar durch API-Controller

Aussage: Die APIs der Mittelschicht sind durch Entwickler erweiterbar.

Die Aussage stützt sich auf die Anforderungen: 56-59,62,63 und die Qualitätsanforderungen: Übertragbarkeit, Änderbarkeit (siehe Kapitel: 2. Anforderungsanalyse).

Stakeholder: Entwickler und Systemarchitekt

Erläuterung zur Stakeholderwahl: Mit der Aussage werden ausschließlich der Entwickler und der Systemarchitekt adressiert, da es ein direkter Eingriff in die Mittelschicht ist, bei der Code der Mittelschicht verändert wird und die Mittelschicht neu kompiliert und deployed werden muss (siehe Kapitel: 2.1. Stakeholder ff).

5.1.4.1 Konzept zum empirischen Evaluieren der Aussage

Versuchsgegenstand: Erweitern der Schnittstelle der Mittelschicht durch neu erstellen und Implementierung eines API-Controllers der Logdaten aus der Datenbank abruft und bereitstellt.

Evaluierungsfragen:

1. Sind Entwickler unter den Studierenden und Bacheloranten nach einer Schulung in der Lage das API-Controller Modellkonzept zu verstehen?

2. Sind Entwickler unter den Studierenden und Bacheloranten nach einer Schulung in der Lage die Schnittstellen der Mittelschicht zu erweitern in dem sie einen API-Controller hinzufügen?

Probanden: Die Probanden sollten Studenten oder Bachelor der Informatik und Wirtschaftsinformatik sein, die mindestens das Modul „Datenbanken und Webtechnologien“ bestanden haben oder ähnliche Theoretische und Praktisches Wissen besitzen. Es ist von Vorteil, wenn die Probanden erste Erfahrungen mit Design Pattern gemacht haben.

Konkret wird folgendes vorausgesetzt:

- das Verwenden und Beschreiben einer REST-Schnittstelle
- das Lesen und Verstehens eines ER-Modells
- das Verstehen der MVC oder API-Controller Pattern
- das Programmieren mit einer OO-Sprache wie Java oder C#

Aufgabenstellung an die Probanden:

Das Team möchte, dass eine Schnittstelle existiert, die Logdaten einer `SessionRun` ausgibt. Dazu erstellen Sie einen API-Controller, der die Logdaten der entsprechenden `SessionRun` ausgibt.

Extra Material Vorgabe:

Durchlesen der Anleitung von ASP.NET Core: https://docs.microsoft.com/de-de/aspnet/core/tutorials/first-web-api?view=aspnetcore-2.2&tabs=visual-studio

5.1.4.2 Einordnung in die Gesamtarchitektur

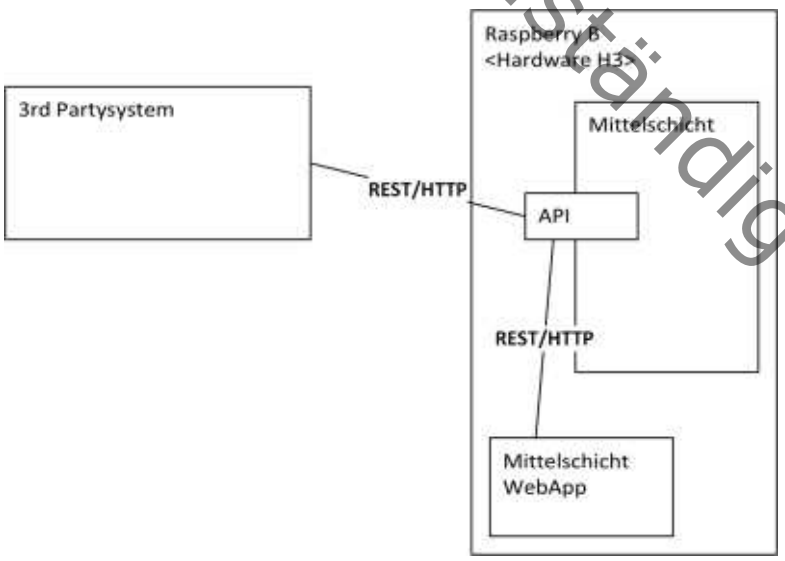


Abbildung 39: Teil des Meta-Modells des Gesamtsystems zum aktuellen Stand mit Fokus auf die API

Der Entwickler implementiert eine neue Schnittstelle zu den Systemen, die an die REST-Schnittstelle angehängt werden. Dies ist ein Beispiel für die Erweiterbarkeit der Mittelschicht.

5.1.5 Aussage: Mittelschicht erweiterbar durch Logger

Aussage: Das Gesamtsystem ist mit Entwicklern anpassbar und erweiterbar.

Entwickler verstehen das Muster, nach dem die Manager und Services in der Mittelschicht erstellt werden müssen und können einen einfachen Service für die Mittelschicht hinzufügen.

Die Aussage stützt sich auf die Anforderungen: 56-59,62,63 und die Qualitätsanforderungen: Übertragbarkeit, Änderbarkeit (siehe Kapitel: 2. Anforderungsanalyse).

Stakeholder: Entwickler und Systemarchitekt

Erläuterung zur Stakeholderwahl: Mit der Aussage werden ausschließlich der Entwickler und der Systemarchitekt adressiert, da es ein direkter Eingriff in die Mittelschicht ist, bei der Code der Mittedelschicht verändert wird und die Mittelschicht neu kompiliert und deployed werden muss (siehe Kapitel: 2.1. Stakeholder ff).

5.1.5.1 Konzept zum empirischen Evaluieren der Aussage

Versuchsgegenstand: Ändern der Mittelschicht durch neuimplementieren und austauschen des `LogService` und `LogManager` als Implementierung der Interfaces `IServeLogging` und `IManageLogService`.

Evaluierungsfragen:

1. Sind Entwickler unter den Probanden nach einer Schulung in der Lage das Manager Service Konzept zu verstehen?
2. Sind Entwickler unter den Probanden nach einer Schulung in der Lage den `LogService` inklusive `LogManager` der Mittelschicht zu ersetzen?

Probanden: Die Probanden sollten Studenten oder Bachelor der Informatik und Wirtschaftsinformatik sein, die mindestens das Modul „Datenbanken und Webtechnologien“ bestanden haben oder ähnliche Theoretische und Praktisches Wissen besitzen. Es ist von Vorteil wenn die Probanden erste Erfahrungen mit Design Pattern gemacht haben.

Konkret wird folgendes vorausgesetzt:

- das Verwenden und Beschreiben einer REST-Schnittstelle
- das Lesen und Verstehens eines ER-Modells
- das Verstehen der MVC oder API-Controller Pattern
- das Programmieren mit einer OO-Sprache wie Java oder C#

Aufgabenstellung an die Probanden:

Das Team möchte, dass die Daten nicht mehr in die Datenbank geschrieben werden, sondern direkt auf die Festplatte des Systems. Dazu muss nach der Manager Service Struktur das Backend, der Mittelschicht, erweitert werden. Speichern Sie alle Log-Einträge einer `SessionRun` in einer eigenen Datei, die durch die `SessionRunId` identifiziert wird. Die Datei soll die Benamung `{{SessionRunId}}.log` tragen.

5.1.5.2 Einordnung in die Gesamtarchitektur

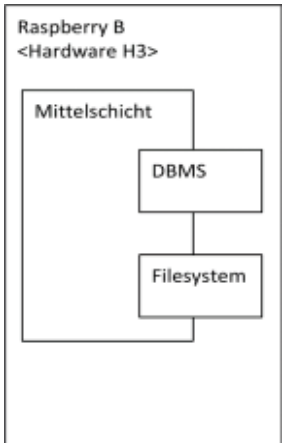


Abbildung 40: Teil des Meta-Modells des Gesamtsystems zum aktuellen Stand, mit Fokus auf die Mittelschicht

Der Entwickler ersetzt die alte Implementierung der internen Schnittstelle in der Mittelschicht, die den Log in die Datenbank geschrieben hat durch eine neue Implementierung, die den Log direkt als Dateien auf das Dateisystem des Betriebssystems schreibt.

5.1.6 Aussage: Unity3D Endpunkt erweiterbar

Aussage: Entwickler können Objekte bauen, die in Unity3D auf MQTT-Nachrichten reagieren und MQTT-Nachrichten absetzen.

Die Aussage stützt sich auf die Anforderungen: 43-45, 56, 57, 58a, (59) und die Qualitätsanforderungen: Übertragbarkeit, Änderbarkeit, Benutzbarkeit (siehe Kapitel: 2. Anforderungsanalyse).

Stakeholder: Entwickler, Projektpartner und Systemarchitekt

Erläuterung zur Stakeholderwahl: Mit der Aussage werden der Entwickler, Projektpartner und der Systemarchitekt adressiert, da Software entwickelt werden muss, diese aber von der Mittelschicht wie ein Endgerät behandelt wird (siehe Kapitel: 2.1. Stakeholder, 2.4.3. 3rd-Party-System - Zentrales-System).

5.1.6.1 Konzept zum empirischen Evaluieren der Aussage

Versuchsgegenstand: Mit dem entwickelten Tool der MQTT-Unity3D-Anbindung bauen Entwickler in Unity3D 3D-Objekte, die einfache MQTT-Nachrichten entgegennehmen und in Handlungen übersetzten, sowie auf Click MQTT-Nachrichten versenden.

Evaluierungsfragen:

1. Sind Entwickler in der Lage, Unity3D Objekte zu bauen, die auf MQTT Nachrichten reagieren?
2. Sind Entwickler in der Lage, Unity3D Objekte zu bauen, die MQTT Nachrichten versenden?

Probanden: Die Probanden sollten Studenten oder Bachelor der Informatik und Wirtschaftsinformatik sein, die mindestens das Modul „OOS“ bestanden haben oder ähnliches theoretische und praktisches Wissen besitzen.

Konkret wird folgendes vorausgesetzt:

- das Programmieren mit einer OO-Sprache wie Java oder C#
- einfaches Verständnis von Nebenläufigkeit

Aufgabenstellung an die Probanden: Bauen Sie ein Debug-Tool zum Drohnentest, den Sie zu Beginn durchgeführt haben (siehe 5.1.1 Aussage: Darstellen von Virtuellen Gegenständen). Bauen Sie eine Anzeige je Steuerachse und färben Sie die Anzeige weiß für den Nullzustand der Achse, grün für den Vorwärtswert oder Linkswert der Achse und rot für den Rückwärtswert oder Rechtswert der Achse. Erweitern Sie die Tore in der Szene so, dass das Ziel-Tor beim Durchflug die aktuelle Anzahl der Durchflüge über MQTT an das Topic: `vr/drohnendemo/zieltor` sendet.

5.1.6.2 Einordnung in die Gesamtarchitektur

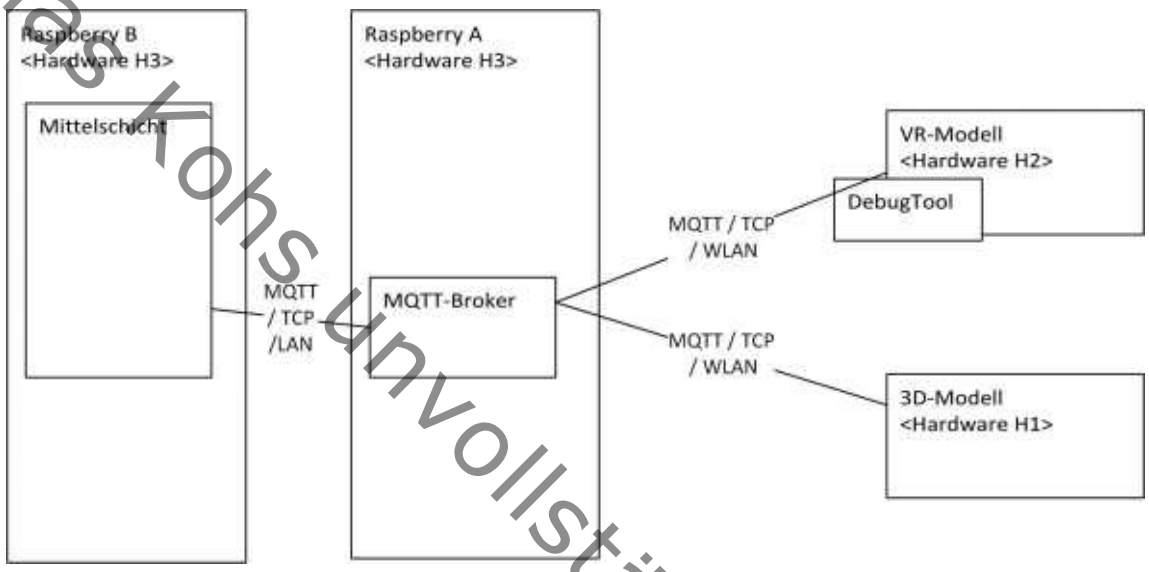


Abbildung 41: Teil des Meta-Modells des Gesamtsystems zum aktuellen Stand, mit Fokus auf eine Konfiguration, die das Eingeben über einen Computer und das Darstellen mit einem anderen über das MQTT-System erlaubt, sowie das Debugen auf VR-Modell-Seite.

Der Entwickler baut ein Debug-Tool, welches in der Szene des VR-Modells eingebaut wird und auf MQTT-Nachrichten zur Steuerung hört sowie eine MQTT-Nachricht absendet, wenn durch das Ziel-Tor geflogen wurde. Die MQTT-Nachrichten zur Steuerung werden von der Unity3D-Szene auf Hardware H1 (3D-Modell) abgesendet, durch den Broker verteilt, über die Mittelschicht gefiltert und weitergesendet. Die Nachrichten werden in dem VR-Modell genutzt, um die Drohne zu steuern bzw. die Debug Anzeigen anzusteuern.

5.1.7 Feedback-Interview

Um Ungenauigkeiten der Beobachtung auszugleichen und weitere Erkenntnisse zur Verbesserung des Gesamtsystems zu gewinnen, wird ein semistrukturiertes Interview geführt. Diese Erkenntnisse können durch eine weitere Iteration der Software umgesetzt werden oder zu einer genaueren Untersuchung speziell dieser Punkte führen.

Da die Anforderungen zum großen Teil als erfüllt oder nicht erfüllt bestätigt werden können, hilft das Interview weitere Attribute zu finden. Das semistrukturierte Interview hilft speziell mit dem Punkt, dass wenn Anmerkungen gemacht werden, direkt darauf eingegangen werden kann, auch wenn es den Pfad der Struktur verlässt. Vor Beginn der Evaluation mit jedem Probanden wird, die Vorgehensweise geklärt, um die Probanden nicht zu überraschen. Während des Interviews wird bei

den Probanden, die zustimmen, eine Audioaufnahme des Interview erstellt. Die Aufnahme wird in eine Zusammenfassung umgesetzt und alle Teile der Aufnahme werden gelöscht.

5.1.7.1 Fragen des Interviewleitfadens

1. Hat das Implementieren (der Durchlauf) wie gewünscht funktioniert?
2. Gab es Sachverhalte, die nicht verständlich waren?
3. Was würdest du direkt verbessern?
4. Was gefällt dir am Design am besten? (Software und GUI)
5. Wo siehst du die kritischen Punkte des Designs? (Software und GUI)
6. Hast du weitere Anmerkungen? Oder möchtest du etwas über das System wissen?

Die Frage1 funktioniert als lockerer Einstieg in das Interview und nimmt direkt Bezug auf den Versuch selbst. Diese Frage soll dabei helfen die Beobachtung zu bestätigen oder Lücken in der Beobachtung aufzudecken.

Die Frage2 fragt nach nicht verständlichen Sachverhalten. Sowohl ein Softwareentwurf als auch eine GUI sollten im Idealfall selbsterklärend oder unmissverständlich sein. Diese Frage steht an der zweiten Position, um direkt von den frischen Erinnerungen zu profitieren.

Die Frage3 ermutigt den Probanden nochmals, Kritik zu äußern, und ermöglicht auch, explizit persönliches Empfinden mit in die Frage einzulegen.

Die Frage4 ermöglicht einen Punkt im Design anzusprechen der gut gelungen ist. Das Wort Design ist offen und meint sowohl GUI-Design als auch Software-Design.

Die Frage5 folgt inhaltlich direkt auf Frage4 und ermöglicht in der Erinnerung an das Design direkt, die Störfaktoren mit zu benennen.

Die Frage6 schließt das Interview mit der Möglichkeit ab, weitere Anmerkungen anzubringen oder Rückfragen zu stellen.

5.2 Auswertung der empirischen Studie mit Probanden

In der Durchführung wurden die Probanden nach der Selbsteinschätzung ihrer Fähigkeiten gefragt (siehe 7.8.1, 7.8.2, 7.8.3, 7.8.4, 7.8.5). Abhängig davon wurde je Proband eine voraussichtliche Tauglichkeit aufgestellt (siehe 7.8 Studie mit Probanden und Interviews). Mit den Probanden wurden, dann die Aufgaben in der Reihenfolge ihrer voraussichtlichen Tauglichkeit durchgeführt. Die Probanden haben nur begrenze Zeit zur Verfügung gestellt, daher ist die Teilnahme an den Aufgaben unterschiedlich ausgefallen. Je Aufgabe war die Abfolge immer dieselbe: Einführung in das Thema, Fragen klären, Aufgabe durchführen, Feedback Interview.

5.2.1 Auswertung zur Aussage 5.2.2

Die Aussage 5.2.2 ist: Die Mittelschicht ermöglicht dem Versuchsleiter die Konfiguration und Durchführung einer Simulation eines SLE auf laborüblicher Hardware.

Für diese Aussage wurden Tests durchgeführt, die durch ein qualitatives Interview unterstützt wurden. Für die Tests wurden fünf Probanden aus dem direkten Umfeld des ITOM gewählt, die die Anforderungen erfüllten. Die Tests sind Aufgrund der geringen Menge der Probanden und der eher

qualitativen und offenen Befragung nicht repräsentativ. Zusätzlich wurde eine Schulung nach dem Durchlauf des ersten Probanden eingebaut, die die Ergebnisse untereinander nicht vergleichbar macht.

Die Aussage 5.2.2 besteht aus zwei Aussageteilen:

1. Die Konfiguration ist möglich auf laborüblicher Hardware.
2. Die Mittelschicht ermöglicht den Versuchsleitern die Konfiguration und Durchführung einer Simulation eines SLE.

Der erste Aussageteil wurde mängellos bestätigt, den in keiner Situation wurde die Hardware an ihre Grenzen gebracht.

Der zweite Aussageteil wurde bedingt bestätigt, denn es gab Ausreißer, die einzeln betrachtet werden müssen. Von fünf Probanden haben vier Probanden die Aufgabe erfolgreich abgeschlossen und von den vier Probanden hat ein Proband nochmal die Erklärung zu der Regelauswertung angefragt. Bei beiden Probanden gab es ein Problem mit dem Verständnis der Regelauswertung, das nachträglich geklärt werden konnte. Der erste Durchlauf wurde durch externe Umstände abgebrochen.

Nach dem ersten Durchlauf wurde eine erweiterte Erklärung eingebaut. Wenn man nun von vier Probanden mit derselben Erklärung ausgeht, hat nur ein Proband eine weitere Erklärung gebraucht, um das System komplett zu durchdringen. Dieser Proband hat im Interview bestätigt, dass er sich in der Lage fühlt, das System selbständig zu benutzen. Man kann damit darauf schließen, dass mit leichtem Anpassen der Anleitung die Probanden die Konfiguration und Durchführung einer Simulation eines SLE vollziehen können.

5.2.1.1 Verbesserungen der Anleitung

Da der Knackpunkt der Aufgaben die Regelauswertung ist muss diesem Punkt mehr Zeit in der Einführung dieser Software gewährt werden:

- Die Regeln sind keine Regulären-Ausdrücke
- Der Unterschied zwischen # und + muss deutlicher werden
- Die Level in # und + werden mit dem Level der Topics in der Eingangsmessage gefüllt

In den Interviews mit den Probanden und in der Betrachtung haben sich Beispiele als Sinnvolles Tool zum Aufzeigen der Unterschiede herausgestellt.

Als mögliches Medium für die Anleitung wurden sowohl Videos in kurzer Länge von 60 Sekunden als auch klassische Anleitung mit erklärendem Text und Bildern genannt. Wichtig ist in allen Formen der Anleitung, mehrere Beispiele für die Verwendung der Regeln zu geben.

5.2.1.2 Verbesserungen der GUI

Obwohl die GUI nicht Hauptpunkt dieser Arbeit ist, wurde die Vorhandene Oberfläche zum Anlass genommen, Feedback dazu zu geben. Die Probanden haben folgende Wünsche geäußert, die alle machbar und sinnvoll sind. Im Folgenden werden die Wünsche in Anforderungen, im Stiel nach Rupp, umgewandelt dargestellt:

- Auf dem Dashboard,
    - soll das System eine Info anzeigen, wenn die Rules gespeichert werden konnten.

    - soll das System eine Warnung anzeigen, wenn die Rules nicht gespeichert werden konnten.
    - soll das System nur Rules anzeigen, die zu der aktiven `SessionRun` gehören.
    - soll es möglichsein sich vorherige MQTT-Message-Topic anzeigen zu lassen
    - soll das Hovern der Maus über dem Feld für Q.o.S die Info anzeigen, dass es sich um das Feld für Quality of Service handelt.
- Das System soll einen Menüpunkt für FAQ erhalten.
- Das System soll in dem Menü-Band unterschiedliche Symbole pro Eintrag anzeigen, um die Verständlichkeit zu unterstützen.
- In der View der Rules, soll das System bei Auswahl des Rule-Typs nur die notwendigen Felder anzeigen.

5.2.1.3 Verbesserung des Systems

Aus dem Feedback und den Tests lassen sich weitere Punkte als Anforderungen finden:

- Das System muss in Bezug auf die Sicherheit gehärtet werden, damit die ungewollte Injektion von Code nicht funktioniert.
- Das System muss Regeln, die zu Loops führen,
    - erkennen
    - vor der Ausführung stoppen
    - und melden

Für die weitere Verbesserung des Systems ist die Erweiterung der automatisierten Tests sinnvoll. Die Tests helfen die Qualität des Systems zu verbessern und die Qualität bei der Erweiterung des Systems zu behalten.

5.2.2 Auswertung zur Aussage 5.2.3

Die Aussage 5.2.3 ist: Das ZAS ermöglicht es Entwicklern, neue Anwendungen an das Gesamtsystem anzubinden.

Für diese Aussage wurden Tests durchgeführt, die durch ein qualitatives Interview unterstützt wurden. Für die Tests wurden drei Probanden aus dem direkten ITOM-Umfeld gewählt, die die Anforderungen erfüllten. Die Tests sind Aufgrund der geringen Menge der Probanden und der qualitativen und offenen Befragung nicht repräsentativ.

Die Aussage 5.2.3 besteht aus zwei implizierten Aussageteilen:

1. Die Technologiewahl ist so getroffen, dass die Entwickler wie unter Kapitel 2.1 beschrieben neue Systeme anbinden können.
2. Die Modellierung der Schnittstelle ist klar und klar genug erklärt.

Der erste Aussagenteil konnte nicht widerlegt werden und wird daher als korrekt angenommen. Alle Probanden haben die Schnittstelle verwendet und konnten das System über das Tool Postman bzw. `Curl` einstellen und haben keine Probleme mit der Systematik der Schnittstelle gehabt.

Die zweite Aussage wurde ebenfalls nicht widerlegt, jedoch wurde das Feedback gegeben, dass die Modellierung für die Event Message nicht optimal ist. Für die weitere Verwendung sollte dort die Vereinfachung der Schnittstellen betrachtet werden.

Generell kann angenommen werden, dass das ZAS es Entwicklern ermöglicht neue Anwendungen an das Gesamtsystem anzubinden. Für die weitere Verwendung sollten die Endpunkte und das durch die Endpunkte getriggert Verhalten in der Endpunkt-Dokumentation besser erklärt werden.

5.2.2.1 Verbesserungen der Dokumentation

Die Dokumentation wurde speziell in Form des Postman-Documenters zur Verfügung gestellt, dieses Dokumentationstool ist speziell für REST-APIs gedacht. Das Tool wurde generell als positiv wahrgenommen, muss aber detaillierter gefüllt werden.

Die Dokumentation soll mehr Informationen über das Verhalten des Systems angeben, damit der Benutzer der Dokumentation keinen Wechseln zwischen Dokumentationen und Anleitungen haben. Speziell für die Punkte `EventMessage`, `MessagePurpose` und `Rule` wurde angemerkt, dass dort mehr Details nötig sind.

Die Herausforderung für die `EventMessage` ist, dass sich das Verhalten ändert, wenn sich Attribute ändern. Dies gehört besser dokumentiert, damit es in dem Zusammenhang keine Missverständnisse gibt. Ebenfalls notwendig zu erwähnen ist, dass bei Änderungen der `Rule` wären einer `SessionRun`, die Änderung direkt angewandt wird und nicht erst nach einem Neustart.

5.2.3 Auswertung zur Aussage 5.2.4

Die Aussage 5.2.4 ist: Die APIs der Mittelschicht sind durch Entwickler erweiterbar.

Für diese Aussage wurden Tests durchgeführt, die durch ein qualitatives Interview unterstützt wurden. Für die Tests wurden fünf Probanden aus dem direkten ITOM-Umfeld gewählt, die die Anforderungen erfüllten. Die Tests sind aufgrund der geringen Menge der Probanden und der qualitativen und offenen Befragung nicht repräsentativ.

Alle Probanden sind durch diese Aufgabe durchgekommen, es brauchten jedoch vier von fünf Probanden Hilfestellungen bei der Abfrage mit LINQ. Von den vier Probanden haben alle bis auf einen angegeben, nicht mit LINQ gearbeitet zu haben.

Letztlich kann man sagen, dass die Aussage nicht bestätigt werden konnte, denn Entwickler wie in unter 2.1 Stakeholder definiert sind nicht ohne weiteres in der Lage, die APIs der Mittelschicht zu erweitern. Für das Projekt sollte das aber kein Hindernis sein, denn die Entwickler wie unter 2.1 Stakeholder definiert sollten nach kurzer Zeit in der Lage sein die Lücken zu schließen. Alternativ könnte man die Beschreibung des Entwicklers anpassen, um Personen zu finden, die den Anforderungen entsprechen, die sich während der Entwicklung ergeben haben. Mit den geänderten Bedingungen könnte man diesen Versuch wiederholen und die Aussage bestätigen oder wiederlegen.

5.2.3.1 Verbesserungen der Anleitung

Durch die Probanden wurde die allgemeine Anleitung als gut wahrgenommen und die Struktur als simpel und hilfreich. Damit wird das Tutorial als Anleitung weiterverwendet.

Ähnliche Tutorials wie für das „Erstellen einer Web-API mit ASP.NET Core“ (Anderson and Wasson 2019) existieren auch für die Sprache LINQ (Olprod (Microsoft) and OpenLocalizationService (Microsoft) 2017; Olprod (Microsoft) and (Microsoft) 2017). Für die weitere Betrachtung könnte man eine Untersuchung durchführen, ob diese Tutorials bereits ausreichen, um das Know-How soweit zu erweitern, dass die Lücken in Bezug auf LINQ geschlossen sind.

5.3 Auswertung der Experteninterviews

In den Experteninterviews wurde sowohl auf den Code als auch die Architektur eingegangen.

Die Architektur wurde im Ganzen als gut aber und nicht außergewöhnlich wahrgenommen. Für den Anwendungsfall, dass diese Software durch andere Entwickler weiterentwickelt werden soll, ist das der beste Case, da diesbezüglich keine Hürde existiert. Jedoch gab es einen Punkt, der zu einer Diskussion der Architektur geführt hat: die Dependency Injection (DI). Die DI wird in ASP.Net Core anders realisiert als in anderen Programmiersprachen und sorgt damit für eine andere Struktur der Interfaces. Für Entwickler, die C# oder ASP.Net Core gewohnt sind, stellt es kein Problem dar und die Struktur sieht aus wie gewohnt. Für andere Entwickler kann diese Art mit DI umzugehen nicht so eingängig sein. Damit die Struktur verständlich ist, ergibt es Sinn den nachfolgenden Entwicklern die notwendigen Unterlagen direkt mitzugeben:

- Tutorial: Erstellen einer Web-API mit ASP.NET Core (Anderson and Wasson 2019)
- Dependency Injection in ASP.NET Core (Latham et al. 2019)
- Arbeiten mit LINQ (Olprod (Microsoft) and OpenLocalizationService (Microsoft) 2017)
- Sprachintegrierte Abfrage (Language-Integrated Query, LINQ) (Olprod (Microsoft) and (Microsoft) 2017)
- Architecting Modern Web Applications with ASP.NET Core and Microsoft Azure (Smith 2019a)

Neben der Architektur wurde auch der Code geprüft. Der Code ist generell gut und passt zu der Architektur, doch bevor das System produktiv genutzt werden soll, sollte die Sprache auf Englisch vereinheitlicht und die Guidelines der Sprache besser eingehalten werden. Beides sind Punkte, die noch in der nächsten Zeit umgesetzt werden, moderne IDEs unterstützen den Prozess. Zusätzlich sollten mehr automatisierte Tests umgesetzt werden, damit der Code Zuverlässigkeit gewinnt und generelle Qualität behält. Um die Qualität weiter zu verbessern und zu sichern, muss die Code Dokumentation noch weiter ausgebaut werden und die API Dokumentation, die mit POSTMAN-`Documenter` erweitert werden.

5.4 Empirischer Belastungstest

Der Belastungstest soll die Grenzen des Systems unter der Annahme der Technischen Anforderungen untersuchen. Dazu werden Aussagen mit Kennzahlen aufgestellt, die geprüft werden. Die Kennzahlen werden bei einem erfolgreichen Durchlauf erhöht und nochmal durchgeführt. Die Kennzahlen werden bei einem nicht erfolgreichen Durchlauf erhöht und nochmal durchgeführt. Die Durchläufe werden gestoppt, wenn das System nicht mehr funktioniert oder eine Verschlechterung zu messen oder „bemerken“ ist. Durch diese Tests soll die Grenze der Belastbarkeit des Gesamtsystems und speziell der Mittelschicht herausgefunden werden. So kann wenn notwendig noch ein Umzug auf stärkere Hardware oder eine Beschränkung der Parameter vorgenommen werden um die Nutzung durch die Forschungsgruppen nicht zu gefährden.

1. Die Mittelschicht funktioniert nicht, wenn man diese A Mal in B Minuten ein und ausschaltet und zum Schluss eingeschaltet lässt.
2. Die Mittelschicht (ausgeführt auf der Hardware H3) stürzt ab, wenn A Regeln mit Wildcards aktiv sind und eine Anwendung B Messages pro Sekunde versendet. Alle A Regeln werden getriggert und 1000 Nachrichten werden versendet.
3. Die Mittelschicht funktioniert nicht, ohne Verzögerungen, auf der Hardware H3, wenn A Regeln mit Wildcards aktiv sind und eine Anwendung B Messages pro Sekunde versendet und eine weiter Anwendung Test-Messages sendet.
4. Die Mittelschicht ermöglicht es nicht, ohne Verzögerungen, auf der Hardware H3, die Eingaben für ein Spiel abzubilden und weiterzugeben, um eine Drohne mit drei Translations-Achsen und einer Rotations-Achse zu fliegen.
    a. Durch eine Regel mit Wildcard
    b. Durch 4 Regeln / eine Regel je Achse
    c. Durch 8 Regeln / eine Regel je Button

Tabelle 20: Belastungstest

Annahme	Variable A	Variable B	Ergebnis
1	10	2	System funktioniert, aber die Variablen wurden nicht erfüllt
1	7	2	System funktioniert; Variablen erfüllt
2	10	2	funktioniert
2	10	8 (480 APM)	funktioniert (verzögerte Regelumsetzung, Log verzögert)
2	0	8	funktioniert (Log verzögert)
2	0	48 (6 * 480 APM)	funktioniert (Log verzögert)
2	10	48	funktioniert (verzögerte Regelumsetzung,

			Log verzögert über Minuten)
2	10	200 (12000)	funktioniert (verzögerte Regelumsetzung über Minuten, Log verzögert über mehr als 10 Minuten)
3	5	8	
4a	/	/	Nicht durchgeführt
4b	/	/	Nicht durchgeführt
4c	/	/	Nicht durchgeführt

5.4.1 Benchmarks zur Optimierung des Systems

Aufgrund der hohen Stabilität des Systems unter Last aber der schlechten Performance im Bezug auf die Reaktionszeit, wurden Benchmarks durchgeführt mit zwei Zielen. Das erste Ziel ist, herauszufinden, welche Teile des Systems optimiert werden können, um die Reaktionszeit zu verbessern. Das zweite Ziel ist, herauszufinden, welche Faktoren die Reaktionszeit am stärksten beeinflussen, um abzuschätzen ob für den Anwendungsfall im SMILE Projekt eine stärkere Hardware verbaut werden muss.

Die Benchmarks wurden auf zwei Systemen durchgeführt, da eines der Entwicklungsumgebung (siehe Hardware H1 im Kapitel 2.5.2) entspricht und das andere der Ausführungsumgebung im SMILE Projekt (siehe Hardware H3 im Kapitel 2.5.2).

5.4.1.1 Versuchsaufbau

Der Versuchsaufbau besteht aus folgenden Komponenten, die zum normalen Betrieb der SMILE Landschaft gehören: MQTT-Broker, Datenbank, Mittelschicht-Frontend, Mittelschicht-Backend und den ausschließlich für den Test relevanten Komponenten: Sender, Empfänger.

Die Aufgabe des Senders ist es in regelmäßigen eingestellten Abständen MQTT-Messages mit Timestamps an die eingestellten Topics zu versenden. Die Aufgabe des Empfängers ist es auf Topics zuhören, den Timestamp der Ankunft zu bestimmen und die Daten auszugeben. Anhand der beiden Timestamps kann man dann die Traveltime bestimmen.

Um die Systeme an die Grenze der Belastbarkeit zu fahren, wurden drei Größen betrachtet:

- Anzahl der MQTT-Regeln (in Bereich von 1-10 Regeln)
- Die Rate mit der der Sender sendet (1-1000 Messages pro Sekunde)
- Die Gesamtanzahl der Messages, die durch den Sender in einem Durchlauf abgesetzt werden (meistens 500 Messages)

5.4.1.2 Versuch auf H1

Die ersten Versuche wurden auf der Hardware H1 durchgeführt, da es der Entwicklungsumgebung entspricht und so eine schnellere Durchführung möglich war. Zusätzlich zeigen die Versuche auf der dieser Hardware die Leistungsfähigkeit des Systems auf „handelsüblicher Laptophardware“.

Wie in Abbildung 42 zu sehen, wurden um dem Originalsystem nahe zu kommen, der MQTT-Broker und die OpenHAB-Instanz in einer Virtuellen Maschine mit Linux gekapselt. Die Testtools sowie die Mittelschicht wurden auf dem System direkt durchgeführt. Der MQTT-Sender und MQTT-Empfänger laufen getrennt, damit diese Anwendung so wenig wechselseitige Beeinflussung haben wie möglich.

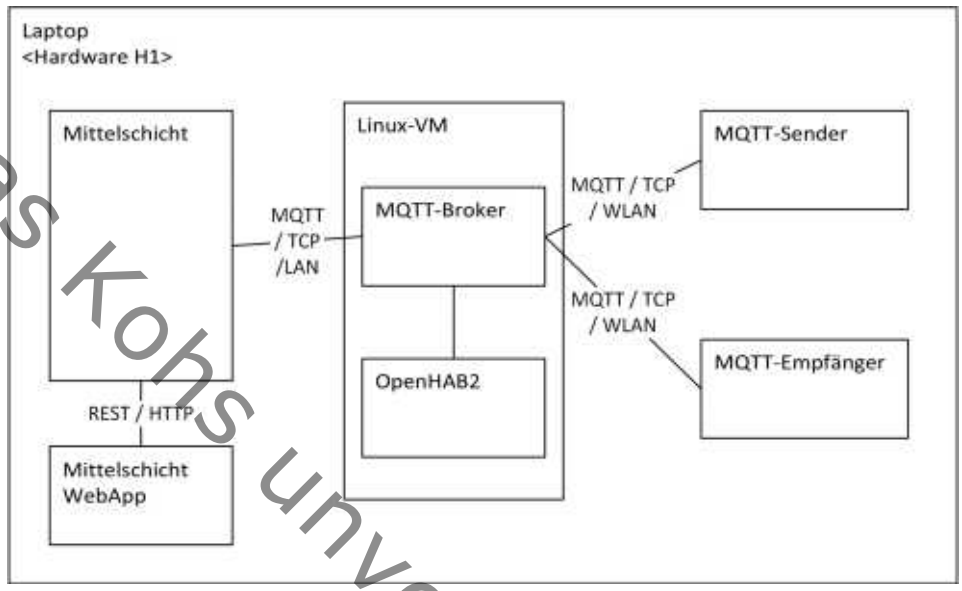


Abbildung 42: Benchmark auf Entwicklungshardware H1

In den Durchläufen wurden initial zwei Algorithmen untersucht, SP für Single-Prozess und TP für Task-Prozess. Der Unterschied liegt in dem Punkt, dass SP für jede ankommende Nachricht der Check auf Rules einzeln seriell in einem Prozess abarbeitet. TP führt im Gegensatz zu SP für jede Nachricht einen Task ein, der über einen Thread aus dem Threadpool abgearbeitet wird.

In der Abbildung 44 werden mehrere Durchläufe mit unterschiedlichen Algorithmen und unterschiedlicher Anzahl an Regeln dargestellt. Die Abkürzungen der Graphen sind wie folgt aufgebaut:

```
{Algorithmus}-{Anzahl Rules}R-{Rate in Message pro Sekunde}pS-{Gesamtanzahl der Messages}msg
```

Der Vorteil von SP ist die Simplizität des Systems im Ganzen und die Nachvollziehbarkeit der Reihenfolge, in der die Messages abgearbeitet und weitergesendet werden. Die Simplizität des Systems sorgt für Einfachheit in der Nachvollziehbarkeit, Änderbarkeit und anderen Teilen weitere Arbeit, die darauf angewiesen sind, den Prozess zu verstehen.

TP wurde gebaut, um den Verlauf der Reaktionszeiten von SP zu mildern. Wenn man sich die Graphen ansieht (siehe Abbildung 44), sieht man einen klaren unterschied bei einer größeren Menge Nachrichten. Bei kleinen Mengen (siehe Abbildung 43) kann die Verwaltung der Task sogar zu Mehraufwand führen.

Durchlauf TP-10R-200pS-500msg geht nach ungefähr nach 300 Messages in eine „Sättigung“ und verliert den Abstand in der Reaktionszeit zum Vergleichslauf SP-10R-200pS-500msg. Aus der Sättigung kann man schlussfolgern, dass ggf. ein anderer Teil der Anwendung die Threads hält oder eine Serialisierung forciert. Um diesem Problem zu begegnen, wurde TP2 gebaut.

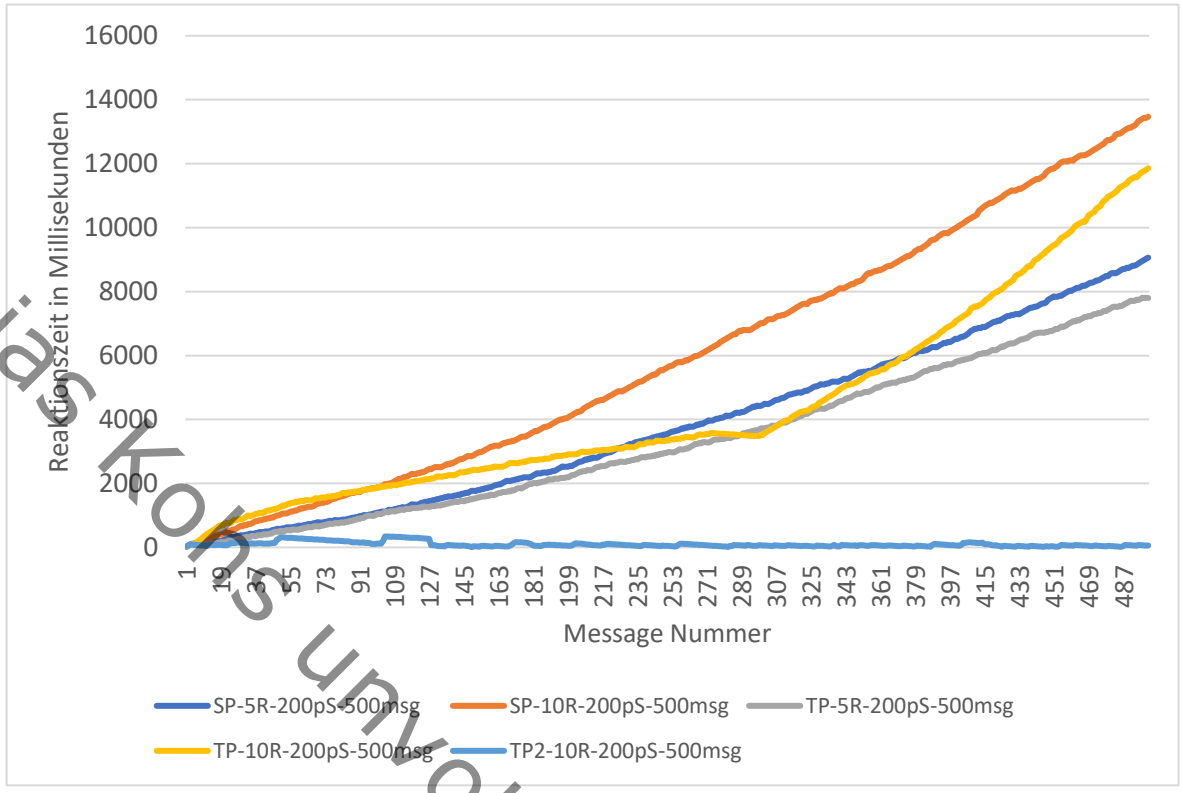


Abbildung 44: Diagramm der Reaktionszeitverläufe in Millisekunden pro Message

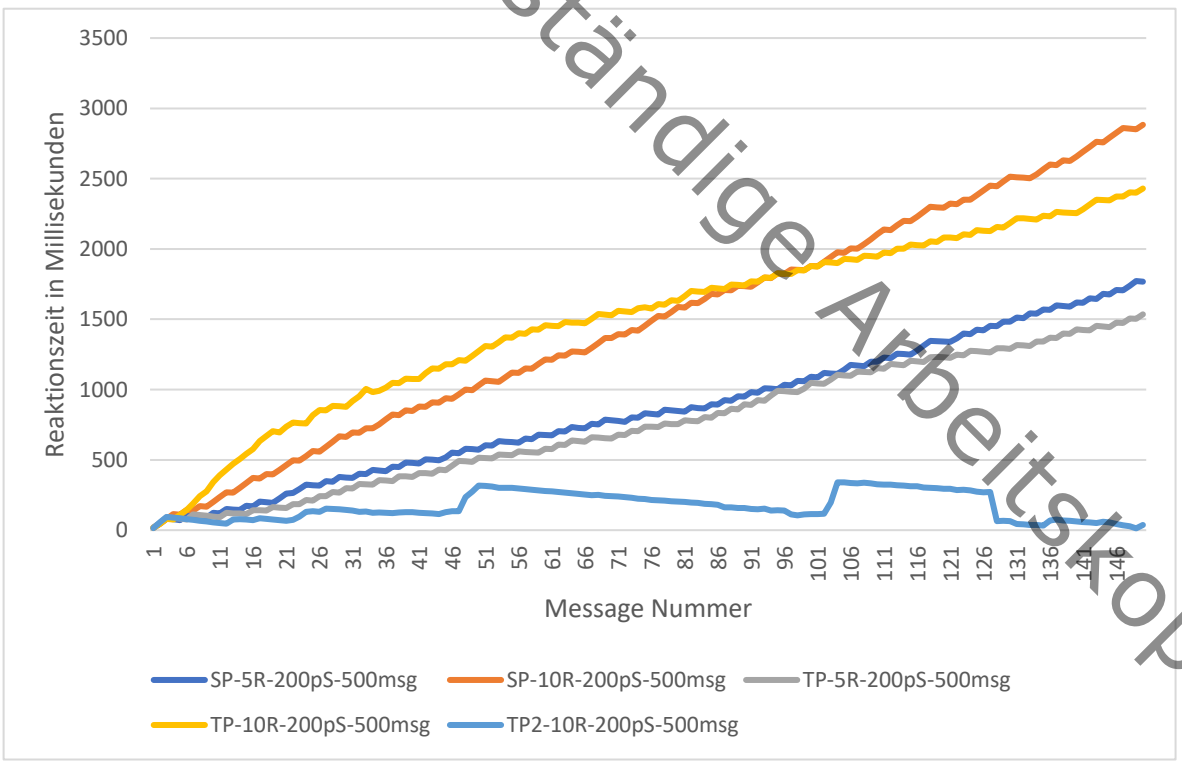


Abbildung 43: Diagramm der Reaktionszeitzeitverläufe in Millisekunden pro Message (bis 150 Messages)

TP2 wurde gebaut unter dem Wissen, dass TP durch die nebenläufige Abarbeitung der Regeln die Nachvollziehbarkeit der Reihenfolge bereits verändert hat. TP2 basiert auf der Hypothese, dass ggf. Teile der Anwendung Rechenzeit brauchen und die Weiterarbeit durch Serialisierung aufhalten.

TP2 delegiert jeden durch eine MQTT-Message getriggerten „Service-Aufruf“ an einen eigenen Task. Dieses Vorgehen entkoppelt jede Durchführung voneinander und die Ausführung muss nicht aufeinander warten.

In Abbildung 44 und Abbildung 43 ist der Effekt zu sehen, TP2 hat bei der selben Belastung eine Reaktionszeit von Maximal 340Millisekunden bei einem Durchschnitt von 95Millisekunden und einem Median von 66Millisekunden gegenüber TP mit einer Reaktionszeit von Maximal 11857Millisekunden (~11Sekunden) bei einem Durchschnitt von 4315Millisekunden und einem Median von 3347Millisekunden.

Mit dem Erfolg, dass die Hauptanforderung an das System, das Ausführen von Regeln und damit das Stellen des Topic Trenners unterdauerlast auf eine Durchschnittliche Zeit von 95Millisekunden geschrumpft ist wird der Algorithmus TP2 auf im Versuch auf H3 untersucht.

5.4.1.3 Versuch auf H3

Der abschließende Versuch wurde auf der Hardware des Projekts durchgeführt, um die Frage zu klären, ob die Hardware die Leistungen erreichen kann oder ein Upgrade notwendig wird. Abbildung 45 zeigt die Konstellation des Versuchs. Wie auch in früheren Anwendungen läuft die Mittelschicht auf einem Raspberry und der Broker auf einem anderen. Der Laptop nimmt in der Konstellation, als der Sender und Empfänger, die Position ein, die sonst durch die Endgeräte eingenommen wird.

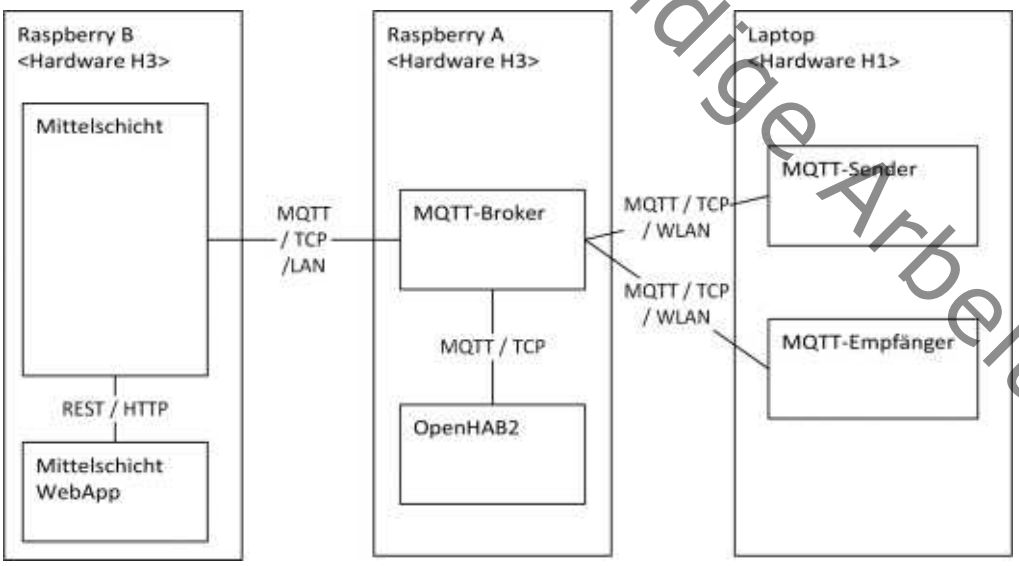


Abbildung 45: Benchmark auf Projekthardware H3

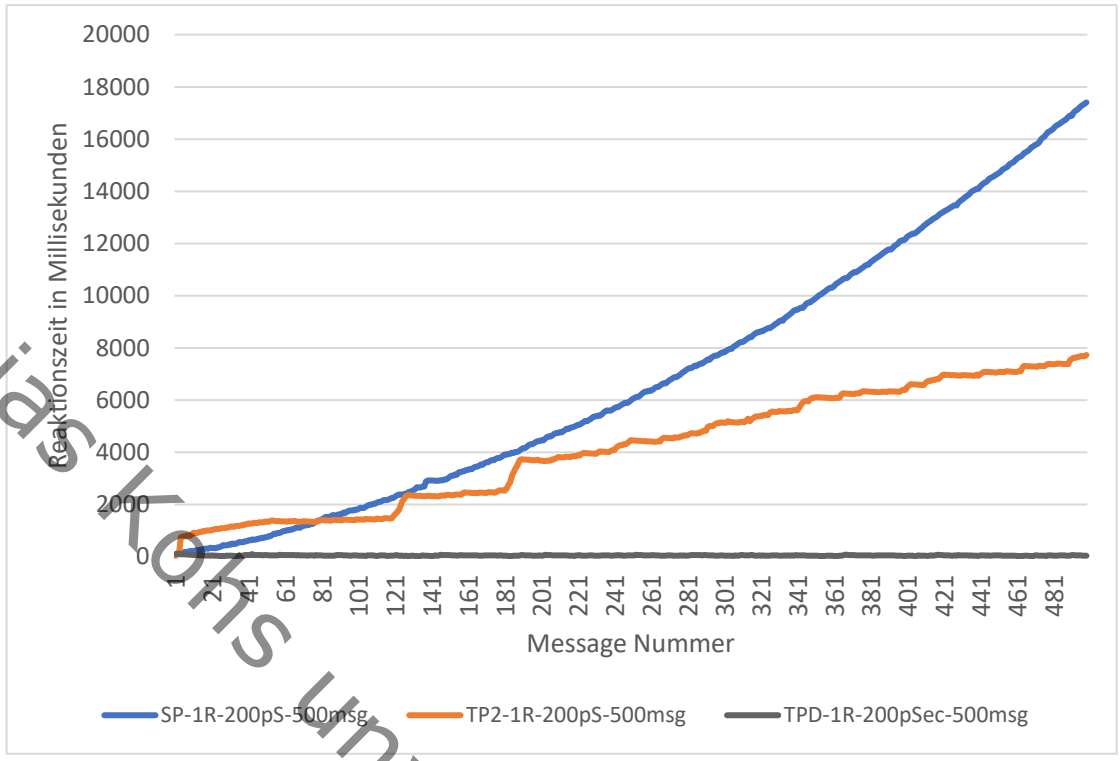


Abbildung 46: Diagramm der Reaktionszeitzeitverläufe in Millisekunden pro Message auf Hardware H3 für SP, TP2, TPD mit einer Regel

In den Durchläufen wurden primär zwei Algorithmen untersucht TP2 und SP. TP2 hat wie erwartet eine bessere Reaktionszeit als SP, jedoch in der absoluten Betrachtung deutlich schlechter als erwartet (siehe Abbildung 46).

Bei genauerer Untersuchung ist aufgefallen, dass der Log-Service nach Abschluss der Durchläufe weiter Daten geschrieben hat, da der Service nicht alle Daten in der Zeit abarbeiten konnte. Wenn sich in TP2 die Performance über die Zeit verschlechtert, wie in Abbildung 46 zu sehen, ist damit zu rechnen, dass Threads, die Tasks abarbeiten, für lange Zeit gebunden bleiben. Wenn mehr Tasks gebunden bleiben, als Threads im Task-Pool vorhanden sind, können die Tasks nicht mehr alle nebenläufig abgearbeitet werden und es kommt zu einer künstlichen Serialisierung. Der flache Anstieg, gefolgt von der Treppenstufe in der Abbildung 46, spricht dafür, dass die Tasks gut laufen, bis der Pool aufgebraucht wird und dann folgt ein erhöhter Anstieg des Zeitbedarfs. Das Verhalten der Reaktionszeit zusammen damit, dass der Log nach dem Durchlauf weiter aufgebaut wird, spricht dafür, dass das Schreiben in den Log der Hauptzeitfaktor ist.

Unter der Hypothese, dass der Log-Service durch das Binden von Threads der Hauptzeitfaktor ist, wurde der Log-Service für den Durchlauf TPD durch einen Dummy ersetzt. Dieser Dummy nimmt die Anfragen an, tut nichts und gibt den Thread zurück. Mit diesem Verhalten des Dummy soll der Service schneller die Threads freigeben und zu langhaltig besseren Ergebnissen in der Reaktionszeit führen.

In Abbildung 46 ist zu sehen, dass sich der Durchlauf TPD deutlich besser verhält als die Alternativen. Der Durchlauf TPD hat einen Maximalwert von 117Millisekeunden, bei einem Mittelwert von 53Millisekunden und einem Median von 51Millisekunden. In Abbildung 47 sieht man, dass man in bestimmten Umständen immer noch Peaks erzeugen kann, diese sind aber generell niedriger als eine Sekunde und damit zu vernachlässigen. Da das Herausnehmen des Log-Service einen so großen Unterschied macht, ist die Hypothese bestätigt, dass der Hauptzeitfaktor der Log-Service ist.

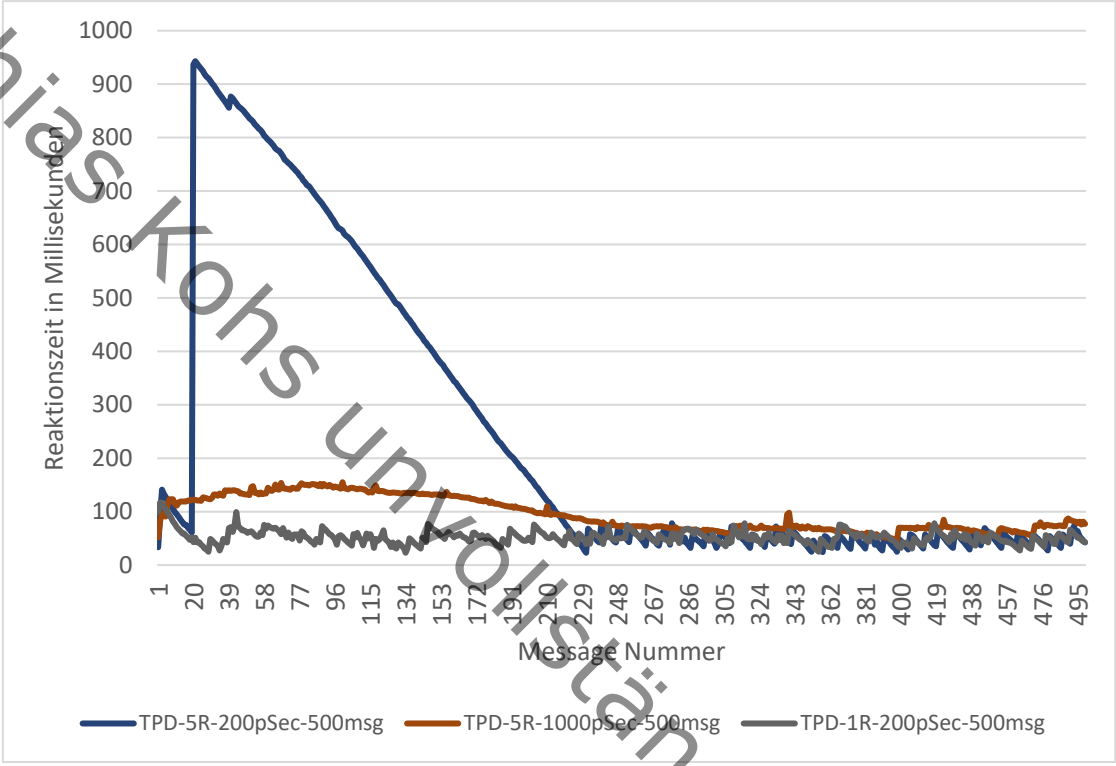


Abbildung 47: Diagramm der Reaktionszeitzeitverläufe in Millisekunden pro Message auf Hardware H3 für TPD mit unterschiedlichen Regeln und Raten

Die Hardware-Wahl spricht dafür, dass der Log-Service aus Hardwaregründen nicht in der Lage ist, die nötige Schreibrate zu erreichen. Die notwendige Schreibrate liegt bei 200Messages pro Sekunde, abgeschätzt[15] entspricht das bis zu 100Mbyte/s. Die Untersuchung, welche Hardwareupgrades die Hardware H3 befähigen würde, die Schreibrate zu erreichen, wird in dieser Arbeit aus Gründen der Zeit nicht durchgeführt.

5.4.1.4 Abschluss des Benchmarks

Abschließend kann man festhalten, dass auf der Hardware H3 die Mittelschicht generell funktioniert, wenn man auf das Loggen der MQTT-Messages verzichtet.

Wenn man die Mittelschicht weiter erweitern will und die Logdaten erfassen möchte, ergibt das Ausführen der Mittelschicht auf einer klassischen PC-Hardware (auch mit Desktop Komponenten) Sinn. Den Leistungstest, wie er hier durchgeführt wurde, kann auch auf anderer Hardware problemlos durchgeführt werden, um weiter abzuschätzen ob die betrachtete Hardware den Anforderungen genügt.

[15] Angenommen eine Message ist bis zu 512Byte groß, ungefähr 255Byte Topic und 255Byte Value, 1Byte QoS und 1Byte Retain.

6 Fazit und Ausblick

In dieser Arbeit wurde ein System entwickelt, das zur Erzeugung, Anbindung und Simulation von virtuellen Gegenständen geeignet ist. Konkret wurde dazu auf Basis der Stakeholder der SMILE-Landschaft und deren Anforderungen Technologie gewählt, ein Gesamtkonzept entwickelt und umgesetzt, das durch die Stakeholder betreibbar und weiterentwickelbar ist.

6.1 Zusammenfassung der Ergebnisse

Zum Beginn dieser Arbeit wurden Stakeholder, Anforderungen und Designprinzipien erschlossen. Aus den Stakeholdern haben sich im Verlauf der Arbeit der Versuchsleiter und der Entwickler als Hauptstakeholder herauskristallisiert. Der Versuchsleiter ist der Stakeholder, der die meiste direkte Interaktion mit dem System hat, und der Entwickler ist der Stakeholder, dessen Randbedingungen Bedingungen für die Entwicklung des Systems bilden. Dadurch, dass der Entwickler im weiteren Verlauf das System erweitern und pflegen soll, mussten Technologien und Konzepte gewählt werden, die zu diesem Stakeholder passen.

Auf Basis der Stakeholder wurden Technologien ausgewählt, die die weitere Entwicklung der Mittelschicht, der Virtuellen Gegenstände und der Anbindung der virtuellen Welt ermöglicht haben. Für die Verbindung der 3D-Engine Unity3D mit MQTT, die aus der Vergangenheit des Forschungsteams gewählt wurde, musste dann festgestellt werden, dass keine direkte MQTT-Verbindung mit Unity3D zu finden war, die den Anforderungen entsprach. Es wurde auf C#-Libraries ausgewichen, die dann an die Benutzung mit Unity3D angepasst wurden.

Anschließend an die Technologiewahl und teilweise in Wechselwirkung wurden die Systeme geformt. Zunächst wurde die Unity3D-MQTT-Bindung ermöglicht. Dabei musste aufgrund der unterschiedlichen Art, wie Nebenläufigkeit erreicht wird, unityseitig über das Game-Loop-Konzept und in C#-seitig über das Task-Threadpool-Konzept eine Verbindung über ein Queue gebaut werden. Nachdem die Verbindung zu Unity3D geschehen ist, wurden Varianten für das logische Verbinden und Trennen der Gegenstände verglichen. Aufgrund unterschiedlicher Vorteile aber primär der geringen Entwicklungskosten an einem zentralen Ort (in der Modelllandschaft) und der geringen Bindung an unterschiedliche Technologien, wurde sich für die Variante des Topic Trenner entschieden. Der Topic Trenner trennt und verbindet Gegenstände, indem er die Eingänge und Ausgänge der Geräte voneinander trennt und nur die Topics miteinander verbindet, dessen Geräte angebunden und verbunden sein sollen. Durch diese Variante kann MQTT ohne weitere Veränderung genutzt werden und alles funktioniert zentral. Für die Entwicklung des Topic Trenners in der Mittelschicht wurde sich für eine Abbildung in Regeln entschieden, da diese auch andere Anwendungsfälle des Systems erfüllten und eine hohe Flexibilität ermöglichen. Diese Regeln werden aufgrund der guten Darstellbarkeit der MQTT-Topic-Struktur in einer Baumstruktur im System gehalten und ausgeführt. Um eine geringe Koppelung zwischen den Services, die sich aus den Anforderungen ergeben haben, zu erreichen, wurde Dependency Injection verwendet. Dadurch, dass viele dieser Services Singleton waren, auf Daten aus der Datenbank zugreifen mussten und über die Controller umstellbar sein mussten, wurde ein Muster entwickelt, was die Abhängigkeitsrichtung korrekt ausrichtet und die Verantwortlichkeit korrekt kapselt. Diese Ausrichtung und Kapselung sorgt wiederum für eine einfachere Erweiterbarkeit des Systems, da sich an dem Muster orientiert werden kann und dieses Muster für eine gute Austauschbarkeit der Elemente sorgt.

Die Entwicklung wurde qualitativ evaluiert, indem Probanden Szenarien durchgespielt haben. Diese Szenarien wurden auf die Hauptstakeholder ausgerichtet: den Versuchsleiter und den Entwickler. In der Auswertung hat sich ergeben, dass noch viel Potential existiert, die Anleitungen zu verbessern, die sowohl für die Entwickler als auch die Versuchsleiter notwendig sind. Ebenfalls hat sich ergeben, dass der Entwickler mehr Schulung in bestimmten Bereichen benötigt, oder der die Anforderungen an den Entwickler verschärft werden müssen, damit die beschriebenen Entwickler das System selbständig erweitern können. Angeschlossen an die Szenarien wurden noch zwei Experteninterviews durchgeführt in der auf den Code und die Architektur eingegangen wurde, um die Qualität zu erhöhen. Die Architektur wurde als gut befunden, was für die Weiternutzung und Weiterentwickelbarkeit spricht. Neben einer Auswahl an Informationsmaterial, was in der Nachbetrachtung entstanden ist, wurden noch mehr automatisierte Tests empfohlen, um technisch die Qualität des Systems zu gewährleisten und weiter zu sichern. Abschießend wurde noch ein Belastungstest, gefolgt von Verbesserungen des Systems durchgeführt. Aus den Belastungstest ergibt sich, dass das System mit der Mittelschicht auf der gewünschten Raspberry-Hardware auch unter großer Last funktioniert, wenn man auf das Loggen der MQTT-Nachrichten verzichtet. Auf PC-Hardware funktioniert das System so schnell, dass selbst Spieleingaben möglich wären.

6.2 Fazit

Das in der Arbeit entwickelte Gesamtsystem eignet sich gut, um die formulierten Ziele zu erreichen.

Für das Ziel der Entwicklung (siehe Kapitel 1.4) gilt, dass es erfüllt wurde. Mit dem System ist es möglich, in der Benutzung zwischen den Modellen zu wechseln und in dem 1:1-Modell dieselben Zustände wiederzufinden wie in dem 1:18-Modell oder in der Virtuellen Welt. Ebenso ist es möglich, durch logisches Trennen von Modellen, diese Modelle parallel zu nutzen und die Zustände nicht modellübergreifend zu teilen, damit die Modelle autark arbeiten. Darüber hinaus gibt die Mittelschicht dem Versuchsleiter die Möglichkeit, Störfälle auszulösen. Die Anbindung an die VR-Welt ist über die Engine Unity3D möglich und die MQTT-Unity3D Bindung wurde erstellt.

Das Ziel, die Forschungsfrage (siehe Kapitel 1.6) zu beantworten, ist ebenfalls erfüllt und erfüllt implizit den Arbeitstitel. Der technische Ansatz des Topic Trenners als ein regelzentriertes System in der Mittelschicht ermöglicht es Gegenstände und digitale Zwillinge in einer mit OpenHAB und MQTT betriebenen SLE-Modellandschaft einzubinden und zu simulieren.

- Das System ist generisch verwendbar, da es sich nicht auf bestimmte Arten von Geräten beschränkt und arbeitet auch über den Einsatz von SLE hinaus.
- Das System ist skalierbar, unter Berücksichtigung der Hardwaregrenzen.
- Das System ist modular, in der Betrachtung, dass viele der Teilsysteme einzeln benutzbar und auch außerhalb dieses Projekts einsetzbar sind.
- Die Mittelschicht ist modular, da jeder Service, der implementiert wurde, austauschbar ist und über Dependency Injection (DI) injiziert werden kann. Zusätzlich lassen sich weitere Dienste entwickeln und einbinden, die durch DI und das Controller Manager Service Muster einfacher eingebunden werden können.
- Die Mittelschicht ist kompatibel zu OpenHAB, das regelzentriert arbeitet und ist selbst ebenfalls regelzentriert umgesetzt.

Das Ziel, das aus der Stakeholdergruppe der SMILE-Landschaft kommt, kann ebenfalls als erfüllt angesehen werden, da alle wichtigen Anforderungen erfüllt wurden und ein System geschaffen wurde, das erweiterbar ist. Für die Benutzung sollten aber noch Iterationen durchgeführt werden, da die reale Erprobung noch aussteht. Zusätzlich gibt es noch Herausforderungen, die auftreten können, wie in Kapitel 4.6 angesprochen. Für den Einsatz in einer realen Umgebung sollte das Feedback der Experten weitergegeben werden, dass das System zusätzlich noch automatisierte Tests benötigt bzw. mehr automatisierte Tests benötigt, damit die Qualität auch bei Weiterentwicklung und Änderung erhalten bleibt.

Meinem persönlichen Ziel ein System aus Teilsystemen zu entwickeln, das vollständig oder in Teilen in der Lage ist den initialen Anwendungsfall zu überstehen, sehe ich als erfüllt und sehe der Zukunft positiv entgegen.

6.3 Weiterer Forschungsbedarf

In folgenden Punkten existiert noch weiterer Forschungsbedarf:

Weitere Untersuchung: Das Gesamtsystem wurde in Hinblick auf spezielle Szenarien qualitativ untersucht, jedoch wurden nicht alle Szenarien umgesetzt und die Anzahl der Probanden war nicht hoch. Daher existiert hier weiter Forschungsbedarf, der weitere qualitative Hinweise geben kann.

GUI: Wie in der Arbeit erwähnt ist, hat die Arbeit keinen Fokus auf die GUI gelegt. Die weitere Verwendung des Systems würde stark von einer benutzerfokussiert entwickelten GUI profitieren. Im weiteren Verlauf können Daten für die Weiterentwicklung sogar in der Nutzungsphase im echten Betrieb stattfinden.

IT-Security: Es gibt in dieser Arbeit Anforderungen, die nicht erfüllt wurden. Diese Anforderungen können noch untersucht werden, ob diese noch gefordert werden und wie diese zu erfüllen sind. Unter den Anforderungen sind Anforderungen, die das Feld der IT-Security anschneiden. Zu dem Feld, wie man MQTT-Systeme sicher macht, gibt es Unterlagen, jedoch wie man dieses System konkret absichert wurde noch nicht betrachtet.

Regel Kreisschlüsse: Die Art, wie die Regeln formuliert werden, ermöglicht es, Kreisschlüsse zu bilden. Wenn diese Kreisschlüsse mit Regeln ausschließlich in der Mittelschicht gebildet werden, könnte man diese Kreisschlüsse mit Algorithmen für die Kreiserkennung auf der Datenstruktur der Mittelschicht erfassen. Wenn sich Kreisschlüsse durch die Kombination von Regelsystemen bilden, könnte man diese ggf. über Heuristiken erkennen. Beide Wege wurden in Zusammenhang mit dieser Software nicht ausreichend betrachtet und sind weiterer Forschungsdarf mit einer Entwicklungs- und Evaluationskomponente.

Regel Seiteneffekte: Man kann durch die hohe Flexibilität des Systems Regeln definieren, die mehrere Geräte betreffen. Durch das Ein- und Ausschalten von Subjekten können Regeln durch ihr Verhalten andere Geräte mit beeinflussen. Durch das Abgleichen der Regeln könnte man ein Auswertungssystem bauen, das diese Beeinflussung vorhersagt. Dieser Punkt ist Forschungsbedarf, der sowohl Entwicklungs- als auch Organisations- sowie Evaluationskomponenten aufweist.

Unity3D-MQTT-Kompatibilität: In dieser Arbeit ist die Unity3D-MQTT-Anbindung entstanden (siehe Kapitel 4.2), diese Bindung wurde nur im Editormodus und im kompilierten Zustand auf klassischen Rechner auf ihre Funktion untersucht. Um diese Bindung über diese Arbeit hinaus zu nutzen, fehlt die Untersuchung ob sie auch auf Mobilgeräten, Unterhaltungskonsolen, Mobile-VR-Devices oder im Web funktioniert.

7 Anhang

7.1 Tools

- Word 356 ProPlus
- Excel 356 ProPlus
- Visio Pro. 2016
- Umlet 14.x
- OpenHAB2.2
- OpenHAB2.4
- MQTT-Box
- Mosquito
- OneNote 356 ProPlus (Freihandzeichnungen)
- Visual Studio Community 2017 (15.x)
- JetBrains Rider 2019.2
- Unity 2017.4.x – 2019.1.x
- GitLab
- GitHub

7.2 Anforderungsliste

Simleiter - Zentrales-System

Die folgenden Anforderungen beschreiben, welche Möglichkeiten das Zentrale System dem Simleiter anbieten muss.

Das ZAS muss dem Versuchsleiter die Möglichkeit geben,

1. eine Modell-Welt einzuschalten.
2. eine Modell-Welt auszuschalten.
3. auf Welt X umzuschalten.
4. die Welt X und die Welt Y zu verbinden.
5. die Welt X und die Welt Y zu trennen.
6. das Gerät X und das Gerät Y zu verbinden.
7. das Gerät X und das Gerät Y zu trennen.
8. das Ausschalten von Geräten zu simulieren.
9. das Einschalten von Geräten zu simulieren.
10. Geräte mit dem ZAS zu verbinden.
11. Geräte mit dem ZAS zu trennen.
12. Werte zu manipulieren.
    a) einen manuellen Wert zu forcieren.
    b) manuell Werte zu senden.
    c) einen manuellen Wert mit zeitlicher Beschränkung Beginn und Ende zu planen.
    d) einen geplanten Wert zu stornieren.
    e) (Einen manuellen Wert X Mal in einem Zeitraum senden zu lassen).
    f) einen manuellen Wert zu deaktivieren.
13. ein Log aller Werte auszugeben.
14. Topics zu abonnieren.
15. bei abonnierten Topics Grenzwerte einzugeben.
16. die Systemuhrzeit einzustellen.
17. die Systemuhrzeit auszugeben.

Modelle - Zentrales System

Die folgenden Anforderungen beschreiben, was das Zentrale System im Zusammenhang mit den Modellen, wie dem 1:18- oder dem 1:1 Modell, können muss.

Das Zentrale Auswertungssystem muss fähig sein,

18. die oben genannten Modelle (über ihre Welten) als Zwillinge zu verbinden.
19. die verbundenen Modelle wieder zu trennen.
20. die sich in den Modellen befindenden Geräte zu verbinden.
21. die verbundenen Geräte wieder zu trennen.
22. die Systemuhrzeit auszugeben.
23. die Systemuhrzeit zu manipulieren.
    a) die Systemuhrzeit zu setzten.
24. die sich in den Modellen befindenden Geräte einzuschalten (sofern die Geräte es unterstützen).
25. die sich in den Modellen befindenden Geräte auszuschalten (sofern die Geräte es unterstützen).
26. manuelle Werte zu senden.
27. manuelle Werte zu forcieren.
28. ein geplantes Konstanthalten von Werten durchsetzen.
29. ein geplantes Konstanthalten von Werten wieder aufzuheben.
30. einen geplanten, manuellen Werteverlauf durchzusetzen.

3rd-Party-System - Zentrales-System

Die folgenden Anforderungen beschreiben, welche Möglichkeiten das Zentrale System den 3rd-Party-Systemen anbieten muss.

Das Zentrale-System sollte 3rd-Party -Systemen die Möglichkeit bieten

31. alle Wert-Manipulationen durchzuführen.
32. alle Geräte-Manipulationen durchzuführen.

Indirekt ergibt sich die Anforderung für den weiteren Betrieb:

Das Zentrale-System wird 3rd-Party -Systemen die Möglichkeit bieten

33. sich zu authentifizieren.
34. autorisierte Handlungen durchzuführen.

Administrator - Zentrales-System

Die folgenden Anforderungen beschreiben, welche Möglichkeiten das Zentrale System dem Administrator anbieten wird.

Das Zentrale-System wird dem Administrator die Möglichkeit bieten

35. andere Systeme für den Zugang zu autorisieren.
36. Zugänge mit Authentifizierungen zu verbinden.

37. Geräte in die Modelllandschaft hinzuzufügen.
38. Geräte aus der Modelllandschaft zu entfernen.

Technische Anforderungen

Die folgenden Technischen Anforderungen beschreiben wie die MQTT-Anbindung an 3D bzw. VR Systeme geartet sein muss.

39. Die 3D-Anbindung (MQTT-Unity3D) muss so gestaltet sein, dass die Anwendung in einfacher Detaillierung ohne VR auf einem Laptop mit der Hardwareausstattung H1 betrieben werden kann.
40. Die VR-Anbindung (MQTT-Unity3D) muss so gestaltet sein, dass die Anwendung in einfacher Detaillierung mit VR auf einem PC mit der Hardwareausstattung H2 betrieben werden kann.
41. Die VR-Anbindung (MQTT-Unity3D) muss so gestaltet sein, dass die Anwendung mit der HTC-Vive-2016 darstellbar ist.
42. Die VR-Anbindung (MQTT-Unity3D) soll so gestaltet sein, dass die Anwendung unter Verwendung von Steam-VR funktioniert.

Die MQTT-Unity3D Anbindung muss so gestaltet sein,

43. dass ein Build erstellt werden kann, der auf folgenden Betriebssystemen OS1 betrieben werden kann.
44. dass ein Build ohne die Mittelschicht bei einem MQTT-Broker Topics abonnieren und dessen Inhalte empfangen kann.
45. dass ein Build ohne die Mittelschicht bei einem MQTT-Broker Topics absetzen und eigene Inhalte senden kann.

46. Die MQTT-Unity3D Anbindung sollte so gestaltet sein, dass ein Build erstellt werden kann, der auf folgenden Betriebssystemen OS2 betrieben werden kann.

Die folgenden Technischen Anforderungen beschreiben, wie die Mittelschicht geartet sein muss, damit diese auf der IT des Projekts ausgeführt werden kann.

Die Mittelschicht muss so umgesetzt sein,

47. dass diese unter der Belastung von einem 1:18-Modell Grundstück auf Hardware H3 betrieben werden kann.
48. dass diese unter der Belastung von drei 1:18-Modell Grundstücken und einer VR-Anwendung betrieben werden kann.
49. dass diese unter der Belastung von drei 1:1-Modellen / Exponaten und einer VR-Anwendung betrieben werden kann.
50. dass diese in einem Verbund von Rechnern zusammen mit OpenHAB2 koexistieren kann.
51. dass diese mit MQTT-Brokern über MQTT/TCP kommunizieren kann.

Hardware H1	
Typ:	Laptop
CPU:	Intel Core i7 7700HQ
RAM:	16GB DDR3
GPU:	Nvidia GTX 1050

Hardware H2	
Typ:	Workstation
CPU:	Intel Core i7 7700K
RAM:	32GB DDR3
GPU:	Nvidia GTX 1080 TI
VR-Device:	HTC Vive

Hardware H3	Raspberry Pi 3B+
Typ:	Einplatinencomputer
CPU:	Broadcom BCM2837B0, Cortex-A53
RAM:	1GB LPDDR2

Betriebssystem OS1	
Microsoft Windows 10:	*Windows 10 Pro*
	Windows 10 Enterprise
	Windows 10 Education
	Windows 10 Pro Education

Betriebssystem OS2	
Linux Distributionen:	Debian 9
	Ubuntu 18 LTS

Anforderungen an sonstige Lieferbestandteile

Es Anforderungen an sonstige Lieferbestandteile, die beschreiben wie die Komponenten gestaltet werden sein müssen.

Von externen Anbietern oder Zulieferern kommende Komponenten des Systems müssen so gestaltet sein,

52. dass diese für bis zu 100€ erwerbbar sind.
53. dass der Code vor dem Buildvorgang einsehbar ist.
54. dass die Komponenten noch eine Supportzeit von mindestens 3 Jahren haben
55. dass die Herkunft des Codes bekannt ist.

7.2.1 Studien Anforderungen

Es folgen Anforderungen, die nicht nur durch den Proof of Concept und einen Selbsttest auf Erfüllung überprüfbar sind. Diese Anforderungen erfordern eine empirische Studie.

Die MQTT-Unity3D-Anbindung muss so gestaltet sein, dass ein Entwickler nach einer Schulung von ca. 3Std über die MQTT-Unity3D Anbindung

56. auf dem MQTT-Protokoll Topics abonnieren und Inhalte empfangen kann.

Simleiter - Zentrales-System

Die Anforderungen von 1-11 wurden durch das Ein- und Ausschalten `Subjects` realisiert. `Subjects` sind von dem Typ `Device`, `World`, `Device-Twin` oder `World-Twin`. In der Oberfläche geschieht dies über das Dashboard (Abbildung 56: SMILE MQTT Mittelschicht Webapp Dashboard).

Die Anforderung 12 wird durch die `Eventmessage` realisiert, diese kann Werte per Zeit oder per MQTT-Nachricht forcieren und zeitlich mit Beginn und Ende beschränkt sein. In der Oberfläche geschieht dies die Views Add Eventmessage, Edit Eventmessage, und Eventmessage List (Abbildung 78, Abbildung 79, Abbildung 77).

Die Anforderungen 13-15 werden über den Log realisiert. In der Oberfläche geschieht dies die Log View (Abbildung 74, Abbildung 75, Abbildung 76).

Die Anforderungen 16 und 17 werden über `TimeDiff` dargestellt, die an der `SessionRun` angebunden ist. Durch die Anbindung an `SessionRun` kann jeder Durchlauf seine eigene Anpassung der Zeit haben.

7.3 Anforderungsabdeckung

Modelle - Zentrales System

Die Anforderungen 18-21 können händisch über Regeln dargestellt werden oder in Zusammenhang mit den `Subjects`, die die entsprechenden Regeln zusammenfasst.

Die Anforderungen 22-23 werden erreicht in dem die Mittelschicht in einer Loop die Zeit über MQTT allen Devices bereitstellt. Die Zeit kann über die REST-Schnittstelle `TimeDiff` gesetzt werden.

Die Anforderungen 24-25 können über die `PurposeMessages` der `Subjects` festgelegt werden. Wir ein `Subject` auf der Mittelschicht eingeschaltet, wird die entsprechende `PurposeMessages` für den Start des `Subjects` getriggert und die Verbundenen MQTT-Regeln ausgeführt. Entsprechend werden die Verbundenen Geräte eingeschaltet und die Regeln verbinden das Gerät, die Welt oder die Zwillinge.

Die Anforderung 26-30 wird durch die Eventmessage realisiert, diese kann Werte per Zeit oder per MQTT-Nachricht forcieren und zeitlich mit Beginn und Ende beschränkt sein. In der Oberfläche geschieht dies die Views Add Eventmessage, Edit Eventmessage, und Eventmessage List (Abbildung 78, Abbildung 79, Abbildung 77).

3rd-Party-System - Zentrales-System

Die Anforderungen 31-32 werden durch das offenlegen der REST-API für die Mittelschicht erreicht. Das 3rd-Party-System kann die REST-API verwenden und alle Funktionen Triggern, die auch über die Mittelschicht-GUI für den Versuchsleiter möglich sind.

Die Anforderungen 33-34 sind aktuell nicht implementiert, können aber durch das Middleware Designpattern oder den Middleware Ansatz von ASP.Net Core implementiert werden (Anderson and Smith 2019).

Administrator - Zentrales-System

Die Anforderungen 35-36 sind aktuell nicht implementiert müssen aber mit den Anforderungen 33 und 34 umgesetzt werden.

Es existiert gerade kein White- oder Black-Listing, deshalb ist das Hinzufügen von Geräten in die Modelllandschaft direkt möglich. Indirekt sind damit Anforderung 37 und 38 direkt erfüllt.

Technische Anforderungen

Eine Auswahl der Technischen Anforderungen werden in Kapitel 5.4 Empirischer Belastungstest detaillierter behandelt.

Anforderung 39	Voll erfüllt
Anforderung 40	Aus zeitlichen Gründen, nicht im vollen Umfang durchgeführt.
Anforderung 41	Voll erfüllt
Anforderung 42	Voll erfüllt
Anforderung 43	Voll erfüllt
Anforderung 44	Voll erfüllt
Anforderung 45	Voll erfüllt
Anforderung 46	Voll erfüllt
Anforderung 47	Voll erfüllt
Anforderung 48	Voll erfüllt
Anforderung 49	Voll erfüllt
Anforderung 50	Voll erfüllt
Anforderung 51	Voll erfüllt

Anforderungen an sonstige Lieferbestandteile

Die Anforderung 52 ist erfüllt, wenn man die VR-Hardware nicht betrachtet.

- Die Software, die verwendet wird, ist open Source.
- Die Libraries, auf die aufgebaut wird, sind open Source.
- Die Hardwareanforderungen an Rechner, auf den das System lauft, liegen im Bereich eines Raspberry Pi3.

Die Anforderungen 53-55 sind ebenfalls erfüllt, alleine die Wahl der Sprache für die Oberfläche, die nicht näher Teil dieser Arbeit ist, hat keine Supportzeit von 3 Jahren. Die Oberfläche die nur dem ersten Entwurf und kann ohne die Veränderung der anderen Bestandteile ausgetauscht werden.

7.4 UML

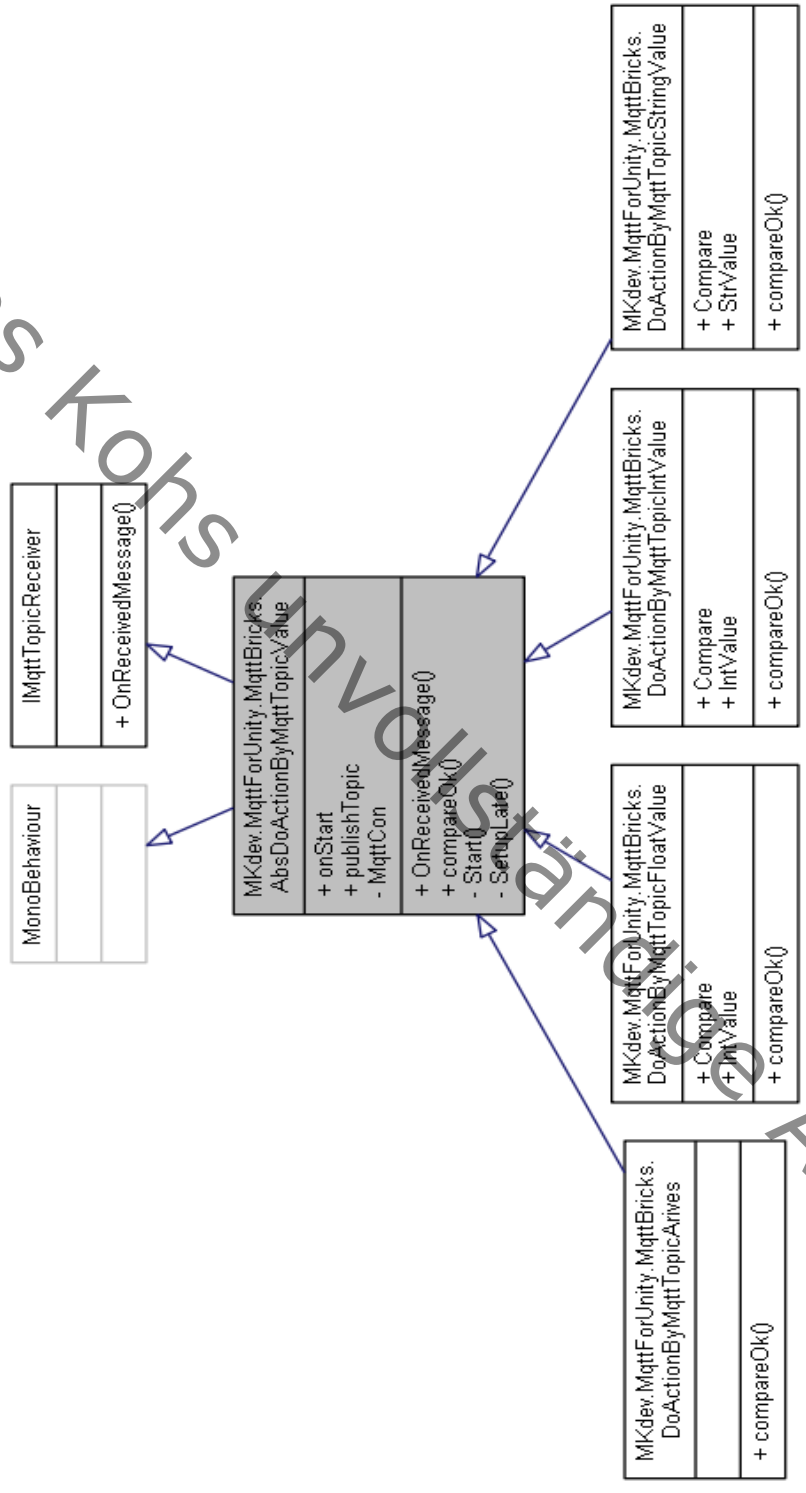


Abbildung 48: UML-Klassendiagramm generiert mit Doxygen Ableitung diverser MQTT-Bricks von der abstrakten Klasse AbsDoActionByMqttTopicValue (Commit: 5b81ec90fdb78e6d2c1c657219b2457a5f469489 GitHub: MathiasKoAc/MKdev.M2Mqtt.Unity3D.DoxyDocu)

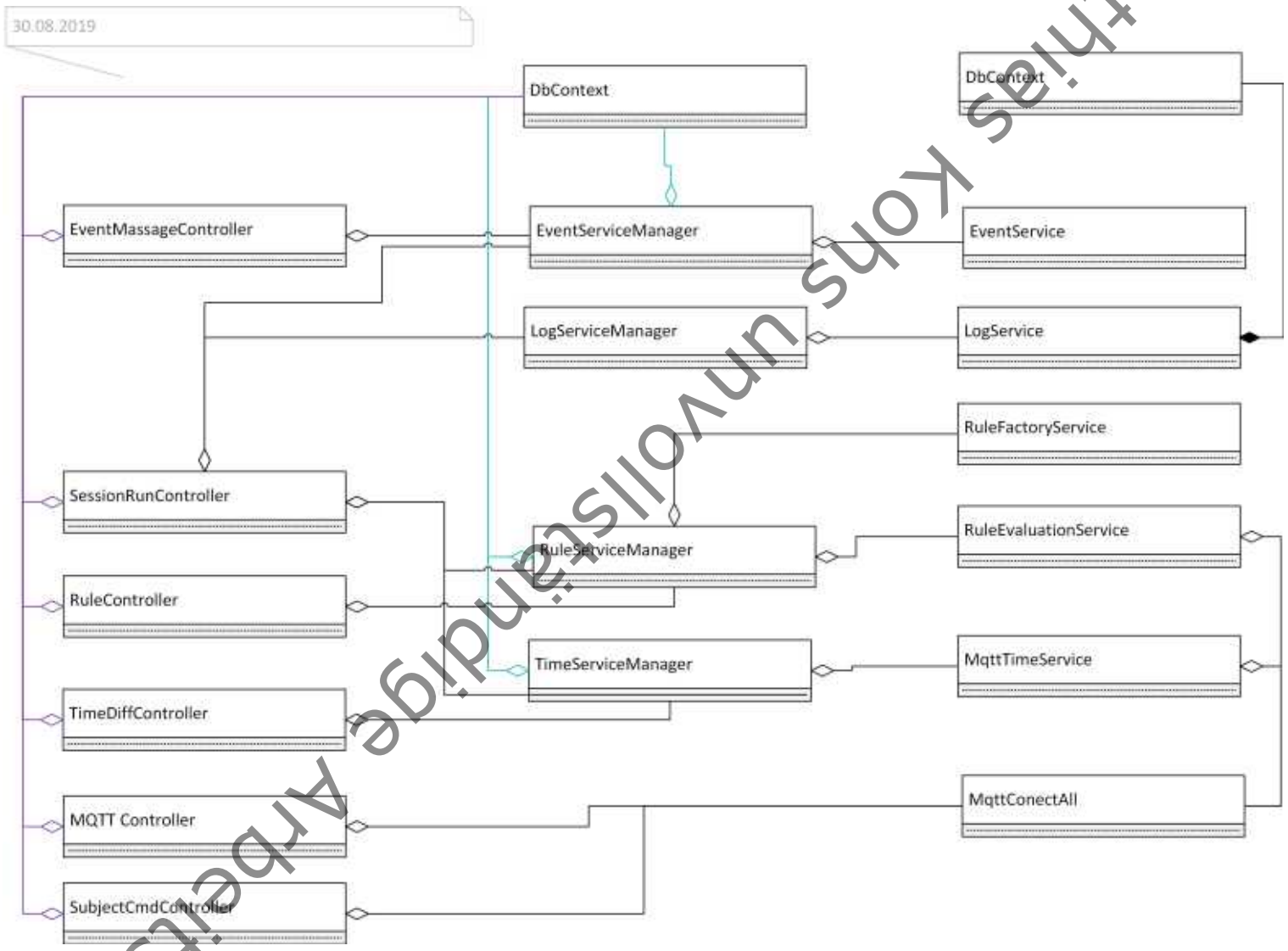
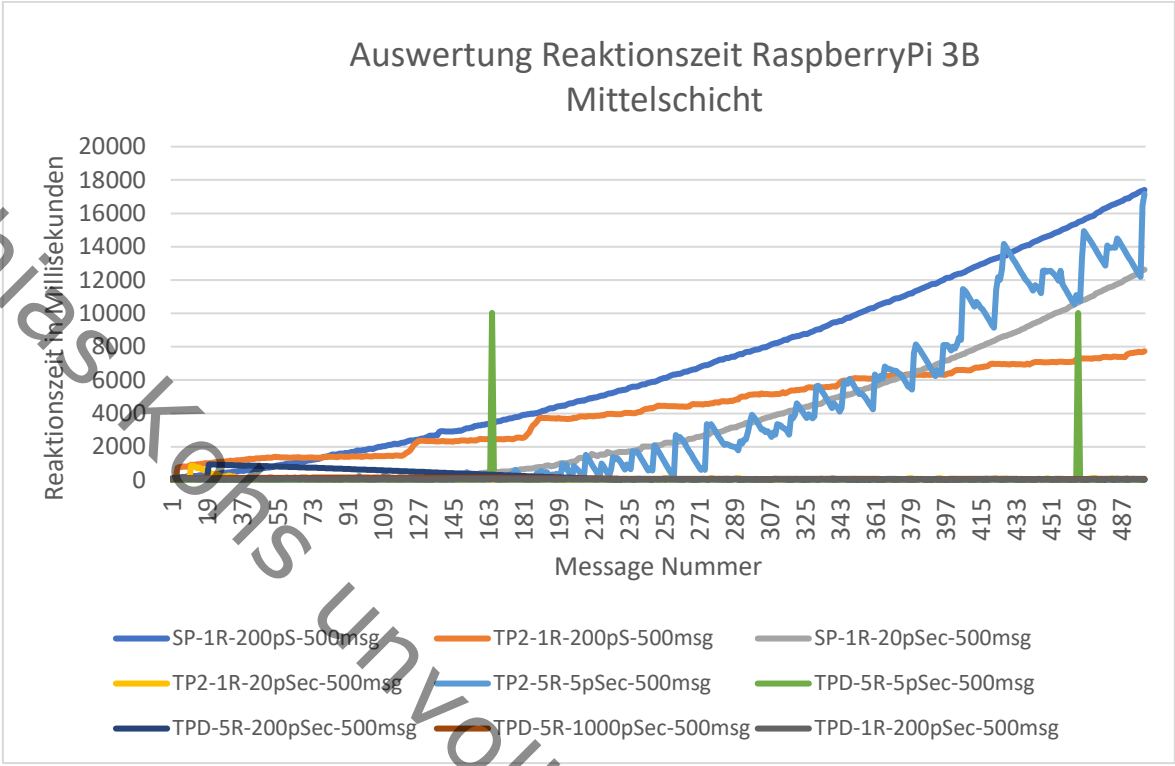


Abbildung 49: Klassendiagramm Controller Manager Service mit Diamantsymbolik für Aggregationen

7.5 Diagramme



7.6 Screenshots

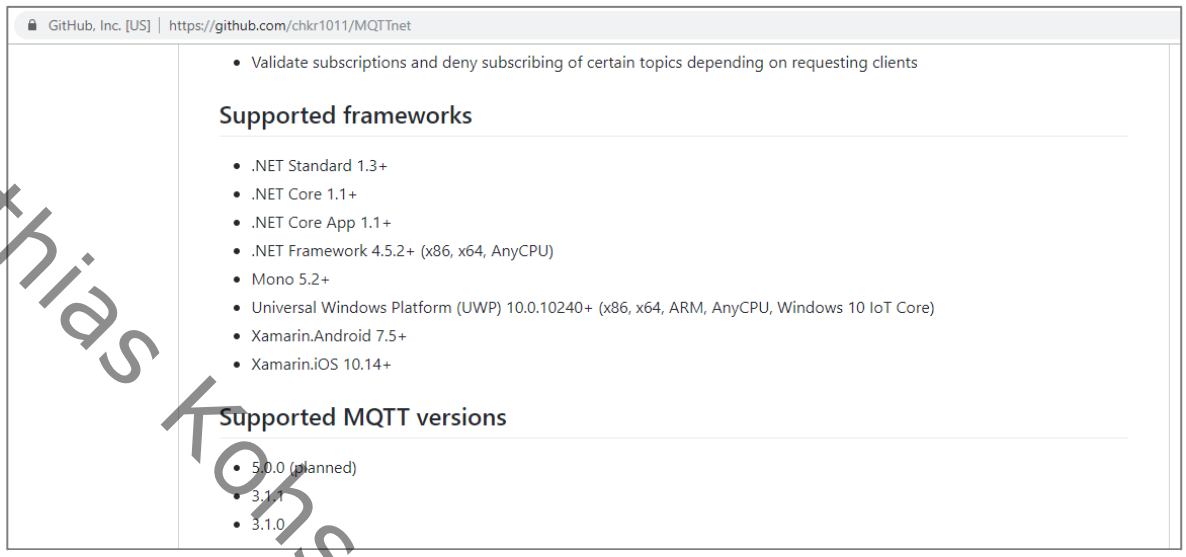


Abbildung 50 Beispiel für Supported frameworks und MQTT Versionen

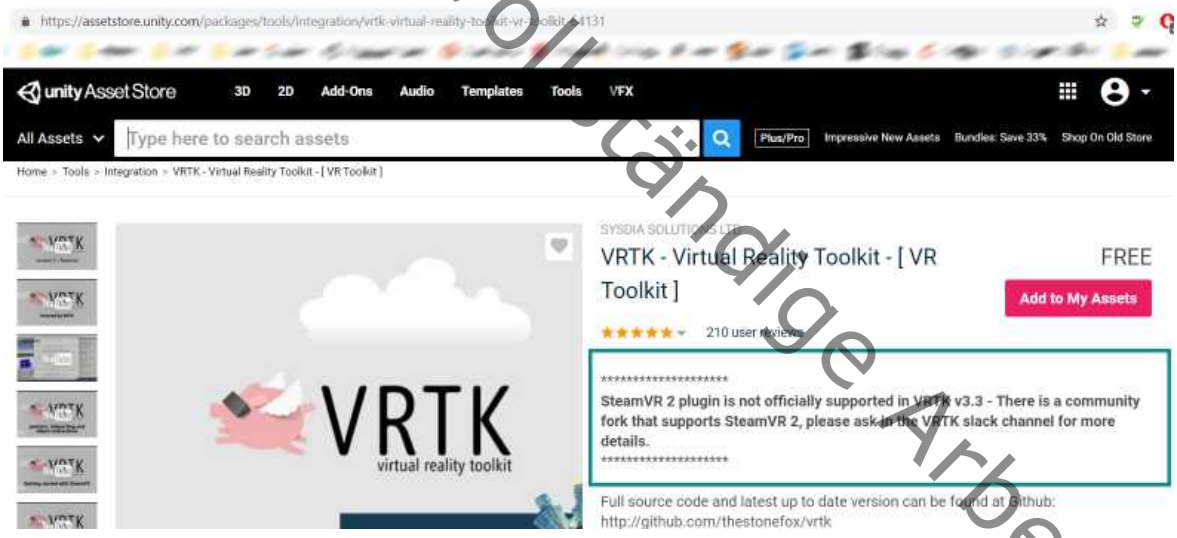


Abbildung 51: Screenshot des VRTK - Virtual Reality Toolkit im Unity Asset Store, mit der Information, dass SteamVR 2 in dieser VRTK-Version nicht unterstützt wird.

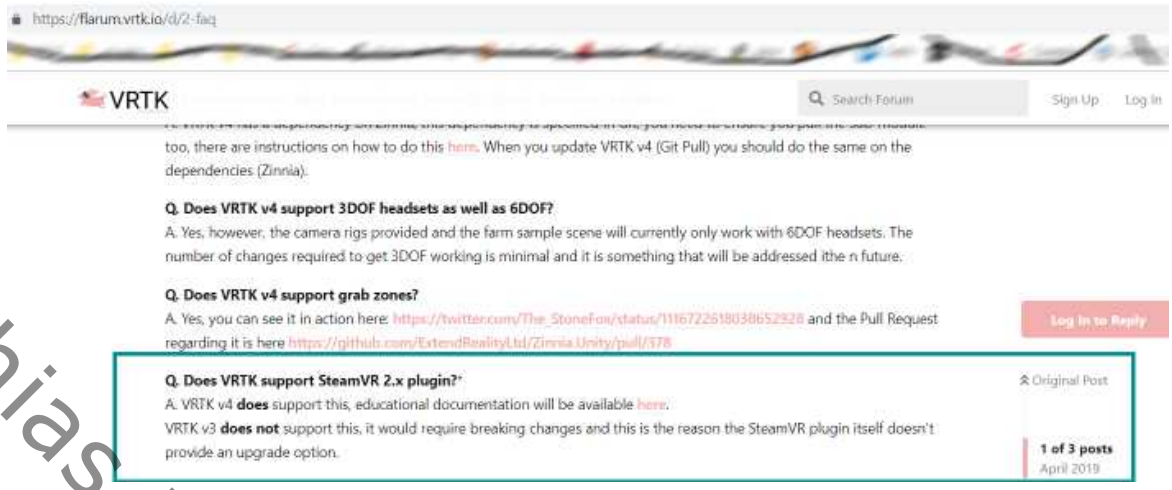


Abbildung 53: Screenshot des VRTK Forums mit der Information, dass Steam VR v2 in VRTK v3.3 nicht aber in VRTK v4.x unterstützt ist

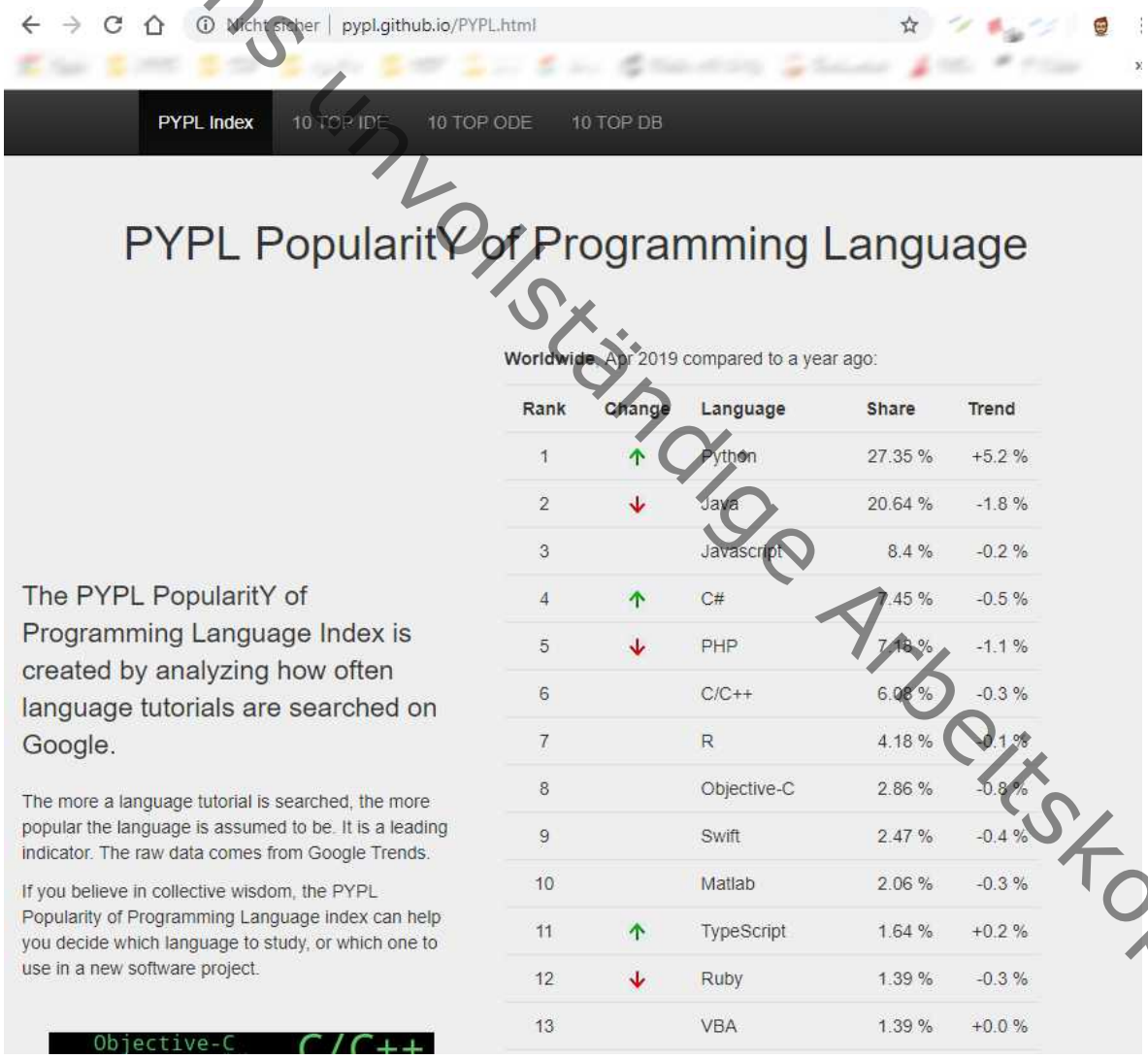
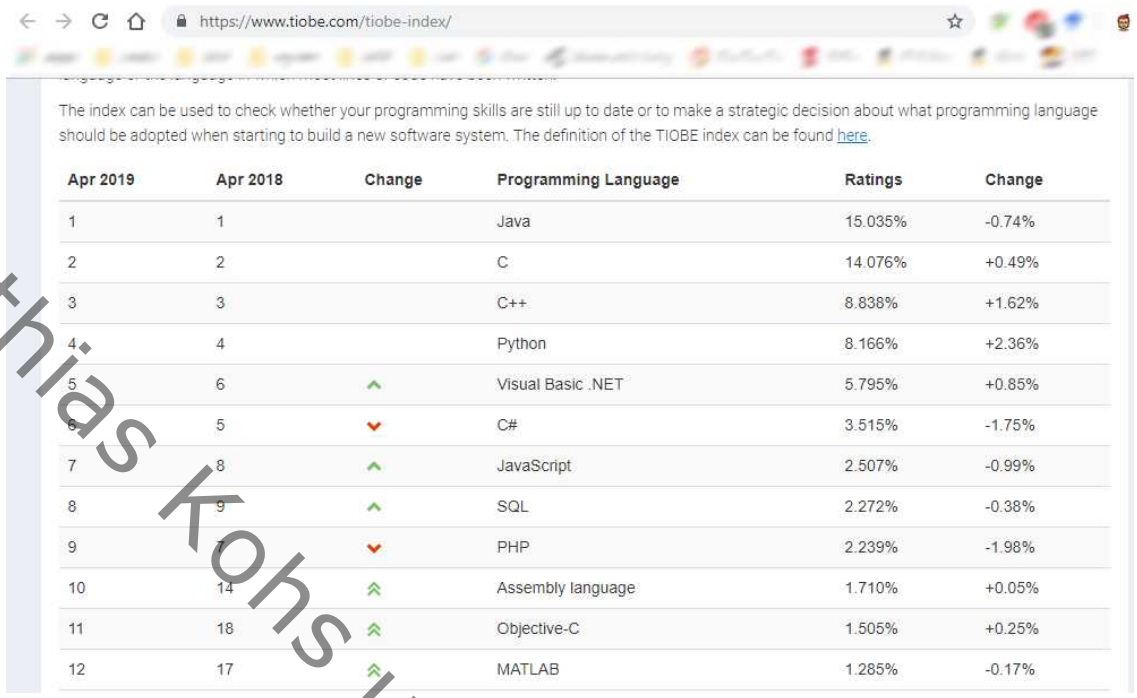


Abbildung 52: Screenshot der Rangliste nach dem PYPL-Index vom 30.04.2019

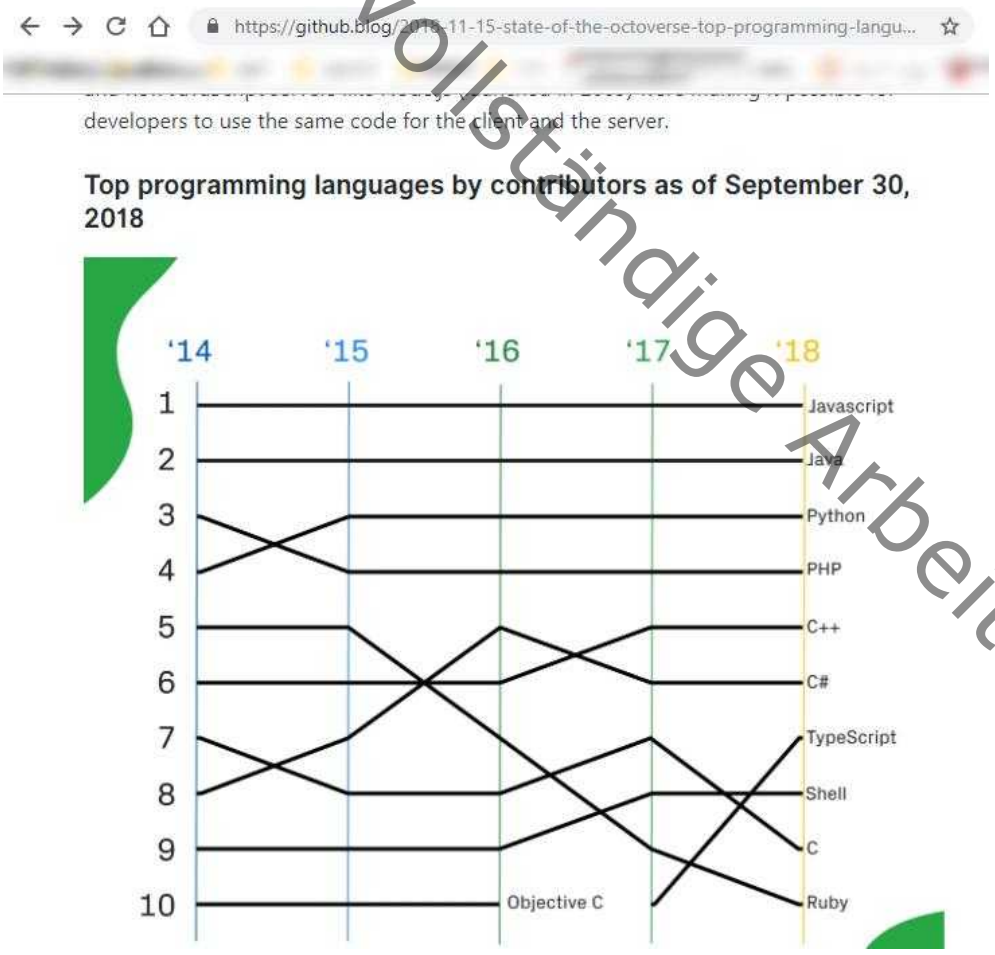


https://www.tiobe.com/tiobe-index/

The index can be used to check whether your programming skills are still up to date or to make a strategic decision about what programming language should be adopted when starting to build a new software system. The definition of the TIOBE index can be found here.

Apr 2019	Apr 2018	Change	Programming Language	Ratings	Change
1	1		Java	15.035%	-0.74%
2	2		C	14.076%	+0.49%
3	3		C++	8.838%	+1.62%
4	4		Python	8.166%	+2.36%
5	6	↑	Visual Basic .NET	5.795%	+0.85%
6	5	↓	C#	3.515%	-1.75%
7	8	↑	JavaScript	2.507%	-0.99%
8	9	↑	SQL	2.272%	-0.38%
9	7	↓	PHP	2.239%	-1.98%
10	14	⇈	Assembly language	1.710%	+0.05%
11	18	⇈	Objective-C	1.505%	+0.25%
12	17	⇈	MATLAB	1.285%	-0.17%

Abbildung 55: Screenshot des TIOBE-Index am 30.04.2019



https://github.blog/2018-11-15-state-of-the-octoverse-top-programming-langu...

developers to use the same code for the client and the server.

Top programming languages by contributors as of September 30, 2018

Abbildung 54: Screenshot des State of the Octoverse der Programmiersprachen auf GitHub

Abbildung 56: SMILE MQTT Mittelschicht Webapp Dashboard

Abbildung 57: SMILE MQTT Mittelschicht Webapp Dashboard SessionRun ist aktiv

Abbildung 58: SMILE MQTT Mittelschicht Webapp Sessions List

Abbildung 59: SMILE MQTT Mittelschicht Webapp Add Session

Abbildung 60: SMILE MQTT Mittelschicht Webapp Edit Session

Abbildung 61: SMILE MQTT Mittelschicht Webapp SessionRuns List

Abbildung 62: SMILE MQTT Mittelschicht Webapp Add SessionRun

Abbildung 63: SMILE MQTT Mittelschicht Webapp Edit SessionRun

Abbildung 64: SMILE MQTT Mittelschicht Webapp Rules List

Abbildung 65: SMILE MQTT Mittelschicht Webapp Add Rule

Abbildung 66: SMILE MQTT Mittelschicht Webapp Edit Rule

Abbildung 67: SMILE MQTT Mittelschicht Webapp Remove Rules

Abbildung 68: SMILE MQTT Mittelschicht Webapp Subjects List

Abbildung 69: SMILE MQTT Mittelschicht Webapp Add Subject

Abbildung 70: SMILE MQTT Mittelschicht Webapp Edit Subject

Abbildung 71: SMILE MQTT Mittelschicht Webapp PurposeMessages List

Abbildung 72: SMILE MQTT Mittelschicht Webapp Add Purposemessage

Abbildung 73: SMILE MQTT Mittelschicht Webapp Edit PurposeMessage

Abbildung 74: : SMILE MQTT Mittelschicht Webapp Log

Abbildung 75: SMILE MQTT Mittelschicht Webapp Log Topic

Abbildung 76: SMILE MQTT Mittelschicht Webapp Wert

Abbildung 77: SMILE MQTT Mittelschicht Webapp Eventmessages List

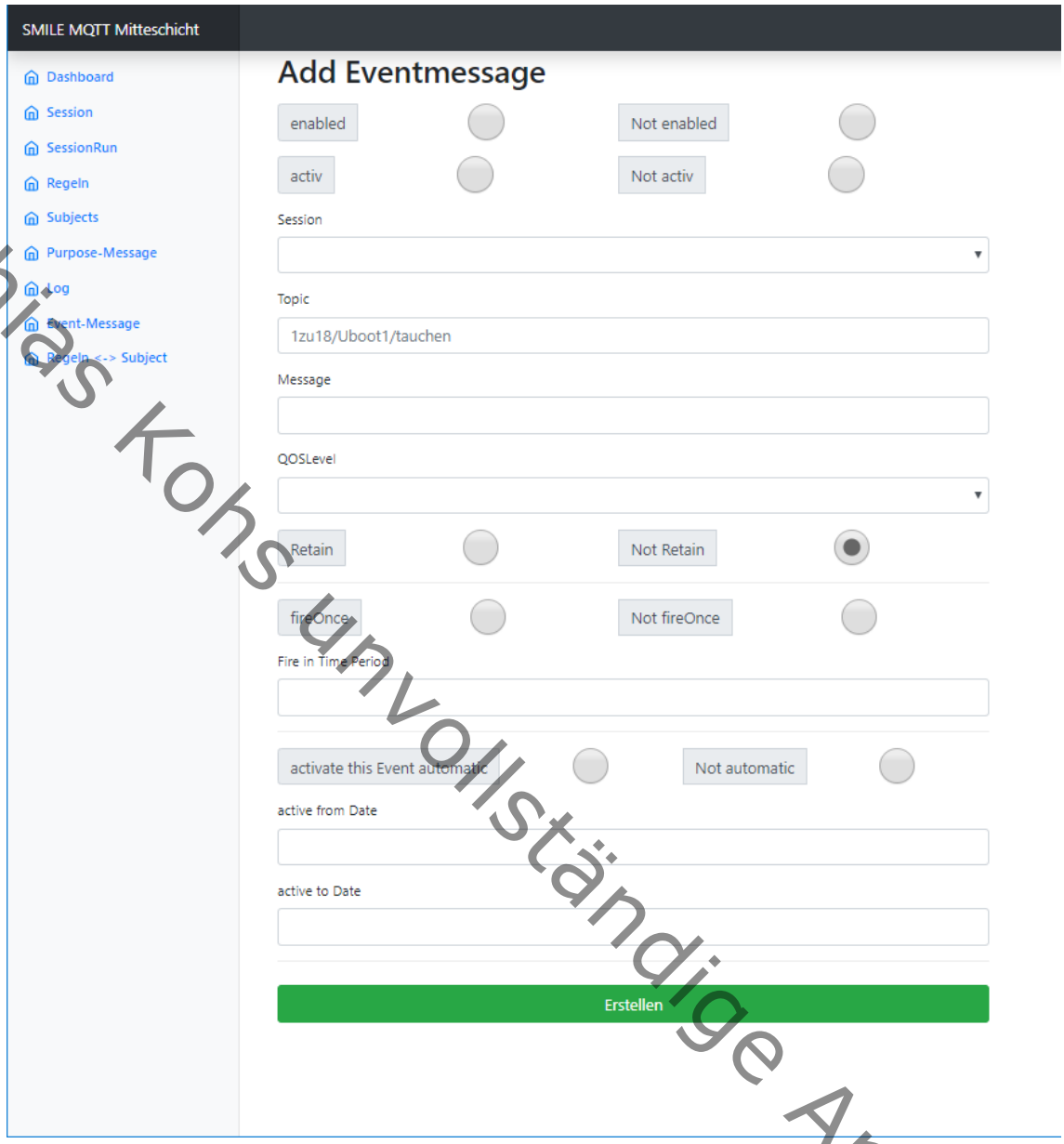


Abbildung 78: SMILE MQTT Mittelschicht Webapp Add Eventmessage

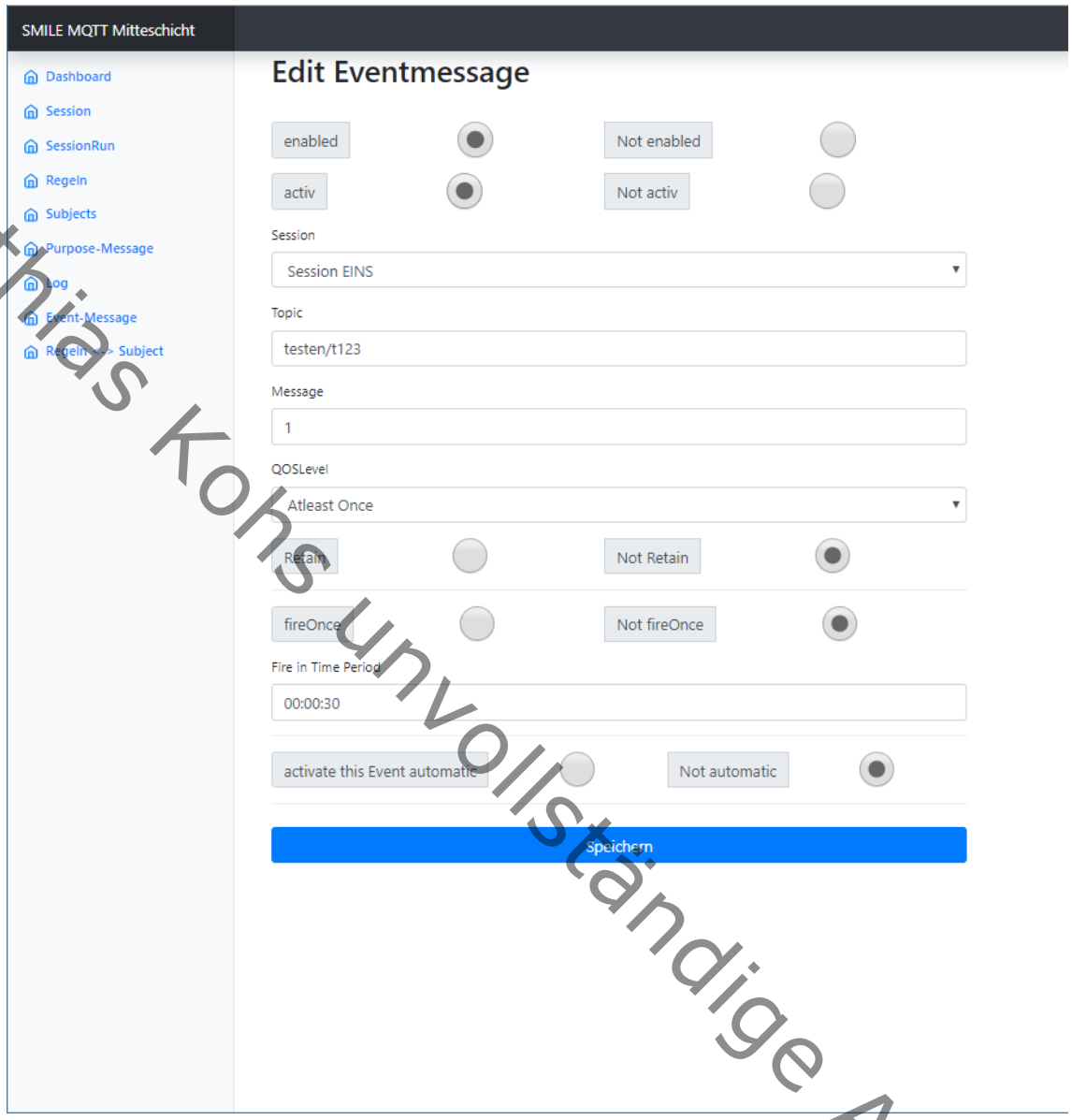


Abbildung 79: SMILE MQTT Mittelschicht Webapp Edit Eventmessage

Abbildung 80: SMILE MQTT Mittelschicht Webapp Add Subject-Rules

7.7 Technologie Auswahl

	code.msdn.M2Mqtt-MQTT-client	chkr1011/MQTTnet [21]	eclipse/paho.mqtt.m2mqtt [24]
	Kandidat B1	**Kandidat B2**	**Kandidat B3**
Codeverwaltungs-Plattform	code.msdn	GITHUB	GITHUB
Doku Vorhanden	JA	JA	JA
Updates im Code	28.03.2015	20.12.2018	15.08.2018
Lizenz	Apache-Lizenz V2.0	MIT	Eclipse Public License 1.0
Aktivität Bug	11.05.2018	20.02.2019	17.01.2019
Sprache	Englisch	Englisch	Englisch
	JA	JA	JA
Unity 2017.4	JA	NEIN	JA
Bemerkung	ohne SercuritySettings SSL	braucht net4.6+ ; braucht c# 7	Verwand mit code.msdn; Probleme mit fehlendem usings behoben
Unity 2018.1	JA	NEIN	JA
Bemerkung		braucht net4.6+ ; braucht c# 7	
Unity 2018.3		JA	JA
Bemerkung			
	JA	JA	JA
Doku Qualität	Beispiel auf der Seite	Wiki Vorhanden	GITHUB Readme mit vielen Informationen; wenig Wiki aber ein gutes Beispiel
Code Commentiert	Sehr gut	Nicht kommentiert	Sehr gut
Code Nachfolziehbar	JA	JA	JA
Projekt erwähnungen			https://www.hivemq.com/mqtt-client-library-encyclopedia/
MQTT Version	3-1-1	3-1-1	3-1-1
			Ausgewählt

Tabelle 21: Finiale Technologiewahl MQTT-Lib für Unity3D

		Auswahl Stufe 1					WEITER
	Codeverwaltungs-Plattform	Doku Vorhanden	Updates im Code	Lizenz	Aktivität Bug	Sprache	
xljiulang/Paho.MqttDotnet [20]	GITHUB	Eher Nein	15.01.2018	?	31.12.2017	Chinesisch / Englisch	NEIN
stevenlovegrove/MqttDotNet [22]	GITHUB	NEIN	23.07.2014	MIT	19.01.2018	Engisch	NEIN
markallanson/nmqtt [23]	GITHUB	Eher Nein	23.11.2013	MIT	18.08.2016	Englisch	NEIN
mFourLabs/KittyHawkMQ [25]	GITHUB	JA	10.03.2016	MIT	17.08.2018	Englisch	JA
ericvoid/StriderMqtt [26]	GITHUB	JA	28.06.2018	MIT	19.12.2017	Englisch	JA
xamarin/mqtt [27]	GITHUB	JA	23.07.2018	MIT	19.02.2019	Englisch	JA
Patierno/msdn.M2Mqtt-MQTT-client [35]	code.msdn	JA	28.03.2015	Apache-Lizenz V2.0	11.05.2018	Englisch	JA
chkr1011/MQTTnet [21]	GITHUB	JA	20.12.2018	MIT	20.02.2019	Englisch	JA
eclipse/paho.mqtt.m2mqtt [24]	GITHUB	JA	15.08.2018	Eclipse Public License 1.0	17.01.2019	Englisch	JA

		Auswahl Stufe 2					
Unity 2017.4	**Bemerkung**	**Unity 2018.1**	**Bemerkung**	**Unity 2018.3**	**Bemerkung**	**Unity 2019.1**	**Bemerkung**
unklar	NichtGenutzteMQTT Clients löschen, MQTT Worker und andere parallel Entwicklungen; Authentifizierung mit dem Client unklar gelöst						
unklar	Nicht klar wie diese Lib in Unity genutzt werden soll						
NEIN	braucht net4.6+ ; braucht c# 7 speziell weil out-parameter; Schlecht umzuschreiben weil Generics und Out verwendert wurden	Schlecht umzuschreiben weil Generics und Out verwendert wurden	braucht net4.6+ ; braucht c# 7 speziell weil out-parameter	Problem mit C#7 gelöst, aber Abhänigkeiten zu andernen Teilen nicht lösbar	Speziell Abhängigkeit zu System.Diagnostics.Tracer; System.Net.Sockets; System.Reactive; System.Runtime.Serialization.Primitives	#--	#--
JA	ohne SercuritySettings SSL	JA				JA	
NEIN	braucht net4.6+ ; braucht c# 7	NEIN	braucht net4.6+ ; braucht c# 7	JA		JA	
JA	Verwand mit code.msdn; Probleme mit fehlendem usings behoben	JA		JA		JA	

WEITER	Auswahl Stufe 3		
	Doku Qualität	Code Commentiert	Code Nachfolziehbar
NEIN			
NEIN			
NEIN			
JA	Beispiel auf der Seite	Sehr gut	GUT
JA	Wiki Vorhanden	NEIN	GUT
JA	GITHUB Readme mit vielen Informationen; Note 2-3; kein Wiki; ein gutes Beispiel	Sehr gut	GUT

7.8 Studie mit Probanden und Interviews

Tauglichkeit der Probanden

Prozentuale Tauglichkeit: Anzahl erfüllter Anforderungen / Anzahl aller Anforderungen

	Aufgabe	**B12**	**T13**	**L14**	**I16**	**T19**
Versuchsleitung	5-2-2	100%	100%	100%	100%	100%
REST Anbinden	5-2-3	100%	50%	50%	50%	100%
API Controller	5-2-4	100%	75%	50%	50%	75%

7.8.1 Proband B12

Proband	
Datum	12.08.2019 / 13.08.2019
Studiengang	Inf.
Laut Selbsteinschätzung technikaffin	Ja
Laut Selbsteinschätzung programmieraffin	Ja
Fachliches Semester	6
Programmiert OO-Sprachen	PHP (seit 2010), C++ (seit 2004), GO(seit 2016), Python (seit 2010), Java (seit 2010)
Bereits mit REST-APIs gearbeitet	Ja
Kann ER-Modelle lesen	Ja
Kennt das MVC-Pattern	Ja
Kennt das ASP-API-Controller Pattern	Nein
Kennt MQTT	Ja
Kennt LINQ	Ja
Kennt Dependency Injection	Ja

7.8.1.1 Beobachtung 5.2.2

Durchlauf abgebrochen

Der Proband hat Regex geschrieben statt Rules für SMILE-MQTT. Diese nicht böswillige Injektion einer Ausführbaren Sprache hat das Programm zum Absturz gebracht. Wenigen Minuten nach Start der Aufgabe wurde der Versuch durch äußere Einflüsse[16] abgebrochen.

7.8.1.2 Feedback Interview 5.2.2

Der Durchlauf hat sich aufgrund von nicht Abfangen von Fehlverhalten verlängert und wurde abgebrochen. Der Proband würde sich eine genauere Einführung in die Regeln wünschen, in den unterschiedliche Beispiele zum Nachlesen vorhanden sind. Des Weiteren ist es notwendig Fehler durch den User wie: Loops, Skripte oder Regex zu unterbinden, auch wenn die Software nur in einer Laborumgebung genutzt wird.

7.8.1.3 Beobachtung 5.2.4

Durchlaufzeit: 8min

[16] Rauswurf durch den Hausmeister aus dem Versuchslabor.

Der Proband hat sich andere Controller angesehen, den RuleController kopiert und alles entfernt, was nicht passt. Danach wurden die GET und GET/id Methoden angepasst.

7.8.1.4 Feedback Interview 5.2.4

Der Proband merkt an, dass diese Aufgabe eine Teilaufgabe eines jeden DBWT-Praktikums ist und man deshalb davon ausgehen sollte, dass jeder diese Aufgabe ohne Probleme lösen können sollte. Der Proband gibt an, dass er nichts direkt verbessern wollte außer, vielleicht auf golang umzusteigen. Das umsteigen auf golang würde laut dem Proband viel „by design vereinfachen“. Zu der Frage, welche Dokumentation notwendig sei, antwortete der Proband, dass man einfach auf die Dokumentation von DOT NET Core weisen kann. Kritische Punkte sieht er in der Wahl der Programmiersprache.

7.8.1.5 Beobachtung 5.2.5

Durchlaufzeit: 8min

Der Proband hat die Klasse erstellt, nachdem ich das Interface genannt hatte. Er hat in der Suchmaschine Google nach der Serialisierung von Objekten von in JSON gesucht und nach paar Minuten eine Lösung mit Newtonsoft.Json gefunden und genutzt.

7.8.1.6 Feedback Interview 5.2.5

Der Proband merkt an, dass der Hauptschwerpunkt im „googlen“ nach einer Serialisierung lag und „alles easy sei“. Er merkt an, dass die Software nicht viel anders aussieht, als eine Standard C# ASP Anwendung und daher alles wie gewohnt ist. Er persönlich mag ASP nicht, da es seiner Meinung nach „zu viel Overhead mitbringt“. (In dem Punkt sei angemerkt, dass der Vergleich mit der Programmiersprache GO gezogen wird. GO wurde explizit als simple Programmiersprache ohne notwendige Ähnlichkeit zu einer älteren Programmiersprachen entwickelt, anders als C# oder die Entwicklung vom Dot-Net-Framework zu Dot-Net-Core).

7.8.2 Proband T13

Proband	as
Datum	13.08.2019
Studiengang	Wirt. Inf.
Laut Selbsteinschätzung technikaffin	Ja
Laut Selbsteinschätzung programmieraffin	Nein
Fachliches Semester	6
Programmiert OO-Sprachen	C++ (seit 2012), PHP (seit 2019)
Bereits mit REST-APIs gearbeitet	Nein
Kann ER-Modelle lesen	Ja
Kennt das MVC-Pattern	Ja
Kennt das ASP-API-Controller Pattern	Nein
Kennt MQTT	Nein
Kennt LINQ	Nein
Kennt Dependency Injection	Nein

7.8.2.1 Beobachtung 5.2.2

Durchlaufzeit: 42min

Der Proband hat das Prinzip von MQTT für erkannt und hat die Unterschiede in den Modellen erkannt und angesprochen. Hat früh mit dem Verwenden von Variablen angefangen. Der Proband brauchte die Erklärung bzgl. + und # erneut, da initial das System nicht deutlich genug wurde.

Beispiel:

```
MSG-Topic:      auto/hupe/1 : 1
IN-Topic: auto/#
OUT-Topic:      boot/#
Ausgabe:  boot/hupe/1 : 1
```

Hat nach 30 min das Wildcard # korrekt eingesetzt und damit Schritt für Schritt die Level des MQTT-Topic-Baums vereinfacht. Nach 42min waren die Gebäude als Zwilling verbunden und der Proband war in der Lage, die Randfälle der Regeln zu demonstrieren und im LOG auf die Nachrichten zu verweisen.

7.8.2.2 Feedback Interview 5.2.2

Der Proband gibt an, dass der Durchlauf funktioniert hat, aber initiale Logikschwierigkeiten vorlagen, die durch ein Beispiel geklärt werden konnten. Der Proband nennt die GUI simple, aber gut. Er wünscht sich folgende Verbesserungen:

1. Es sollte ein Info (success alert) beim Speichern der Rules in der Dashboard-Ansicht eingefügt werden. Es sollte sich in dem Design verhalten wie in den anderen Views.
2. Im Dashboard werden aktuell alle Rules angezeigt, er wünscht sich nur Rules der SessionRun anzuzeigen, damit nur anwählbar Rules zu sehen sind.

Auf die Frage, was nicht klar war, nannte der Proband, dass initial der Unterschied zwischen Wildcard + und Wildcard # nicht klar war, sich aber mit dem Beispiel geklärt hat. Für die Dokumentation wünscht sich der Proband eine Datei oder ein Wiki-Eintrag, in dem die Allgemeine Einführung steht zusammen mit FAQs. Die FAQs könnte man ggf. sogar im System einbetten. Ein Beispiel für eine Frage-Antwort wäre hilfreich im Umgang mit Rule-Loops. Der Proband gibt an, dass das Einheitliches Farbschema der Buttons und Infoanzeigen im gut gefallen hat. Der Proband gibt an, dass das System sehr angenehm ist und hat bereits auf Vermarktungsideen hingewiesen. Ebenso sieht sich der Proband ab dem aktuellen Punkt in der Lage das System selbständig zu benutzten.

7.8.2.3 Beobachtung 5.2.4

Durchlaufzeit 27min

Der Proband hat das Prinzip verstanden und über die IDE einen API-Controller mit Lese -/Schreibaktionen angelegt und diesen mit anderen Controllern verglichen, um die Aufgabe zu lösen. Auf die Funktion des Controllers muss etwas hingewiesen werden doch der Controller wurde korrekt umgebaut. Hilfestellungen wurden im Dialog gegeben und waren daher nicht zählbar. Fragen bezogen sich auf Probleme mit der Dependency Injection und LINQ, was bereits als nicht bekannt angegeben wurde.

7.8.2.4 Feedback Interview 5.2.4

Der Proband gab an, dass die Aufgabe schwierig aber nicht unmöglich war. Nach der Selbsteinschätzung ist ein Anpassen der Software Controllerseitig, nach dem einarbeiten in das Thema kein Problem, aber nicht im Interessensbereich. Die ähnliche Struktur in den Controllern wurde als hilfreich angegeben.

7.8.3 Proband L14

Proband	
Datum	14.08.2019
Studiengang	Wirt. Inf.
Laut Selbsteinschätzung technikaffin	Ja
Laut Selbsteinschätzung programmieraffin	Ja
Fachliches Semester	3
Programmiert OO-Sprachen	C++ (seit 2015), Java (seit 2015), PHP (seit 2013)
Bereits mit REST-APIs gearbeitet	Nein
Kann ER-Modelle lesen	Ja
Kennt das MVC-Pattern	Nein
Kennt das ASP-API-Controller Pattern	Nein
Kennt MQTT	Ja
Kennt LINQ	Nein
Kennt Dependency Injection	Nein

7.8.3.1 Beobachtung 5.2.2

Durchlaufzeit: 14min

Der Proband hat das Prinzip von MQTT für erkannt und direkt angefangen Regeln zu definieren. Nach ca. 10min war die Richtige Rule gefunden. Nach weiteren 4 Minuten wurden Edge Cases der Regeln dargestellt und demonstriert. Nachfragen bzgl. Sonderfälle wurden schnell und klar kommuniziert.

Brauchte die erklärung bzgl. + und # erneut, da inital das System nicht deutlich genug wurde.

7.8.3.2 Feedback Interview 5.2.2

Der Durchlauf hat wie geplant und schnell funktioniert. Weitere Fragen wurden bzgl. Sonderfälle aber nicht der generischen Verwendung des Systems gestellt. Eine Beispielfrage war: „Wenn man nun die genaue Menge der Topics überdecken will, ist es dann sinnvoller alles zu erlauben und Deny-Rules zu erstellen oder initial nur die Rules minimal zu erlauben“. Verbesserungsvorschläge wurde angemerkt, die die GUI im Bereich der Deny-In und Deny-Out Rules anzupassen und nur die nötigen Felder darzustellen. Die Frage, ob Sachverhalte unklar waren wurde verneint. Als Doku hätte sich der Proband die Einleitung die mündlich passiert ist, zusammen mit einer allgemeinen Einleitung gewünscht. Die gegeben Einleitung war gut, aber nicht nachlesbar. Der Allgemeine Aufbau der Oberfläche wurde für gut befunden.

7.8.3.3 Beobachtung 5.2.3

Durchlaufzeit: 30min

Der Proband hat angemerkt, dass er mit WebAPI egal ob REST oder anderer API nicht vertraut ist und schließt aber von der MQTT-API auf die REST-API. Zusätzlich zu der API-Dokumentation, nutzt der Proband den WebClient als Beispiel für Requests und Datenmodelle.

Nach einer Zeit von ca. 30min stellt der Proband eine gültige Lösung vor, die auf Grund eines internen Fehlers im System nicht korrekt funktioniert hat.

7.8.3.4 Feedback Interview 5.2.3

Bis auf den Bug hat die Implementierung wie gewünscht funktioniert. Der Proband merkt auf die Frage 3 an, dass der den Bug fixen würde und in die GUI einen Timepicker und einen Datepicker integrieren würde. Sachverhalte die schwierig zu verstehen waren, waren die Felder enabled und active. Die beiden Felder sind schwer zu unterscheiden und brauchen mehr Informationen. Zusätzlich wurde angemerkt, dass die Lösung für den Sachverhalt nicht elegant gelöst ist und zu erkennen ist, dass die GUI und die Schnittstellt nicht für das Betreiben eines Smarthome gedacht ist sondern für den Simulationsleiterbetrieb. Dem Probanden hat die zur Hilfenahme der GUI geholfen und würde sich eine Anleitung mit Verweis auf die GUI wünschen, ebenfalls fehlen zusätzliche Informationen zu den Attributen dem Modells. Dem Proband hat die Analogie zwischen der Schnittstellt und dem GUI-Client gut gefallen und sieht das als einen positiven Punkt des Systems.

7.8.3.5 Beobachtung 5.2.4

Durchlaufzeit: 25min

Der Proband ist nicht für die Aufgabe geeignet will jedoch die Aufgabe durchführen.

Der Proband hat viel Zeit investiert um den Zusammenhang des Systems und des API-Controllerpatterns zu verstehen und hat nach wenigen Minuten nach Lesen des gegeben Tutorials den Rumpf des Controller stehen gehabt. An den kritischen Stellen in der Benutzung von LINQ gab es größer Probleme bei denen Hilfe notwendig war.

7.8.3.6 Feedback Interview 5.2.4

Der Proband hat sich mit der Technik nicht wohl gefühlt, obwohl er die Aufgabe gelöst hatte. Der Proband gibt an, dass der ohne den Hinweis bei der Where-Clause in LINQ nicht fertig geworden wäre. Zu weiterem Informationsbedarf gibt der Proband an, mehr über C# und den Gesamtzusammenhang des Systems auf Technischer Ebene wissen zu wollen. Der Proband würde sich für seinen Wissenstand lieber Live-Beispiele wünschen statt Tutorials oder Dokumentation.

7.8.4 Proband I16

Proband	
Datum	16.08.2019
Studiengang	Inf.
Laut Selbsteinschätzung technikaffin	Ja
Laut Selbsteinschätzung programmieraffin	Ja
Fachliches Semester	3
Programmiert mit OO-Sprachen	Java (seit 2019), C++ (2016)
Bereits mit REST-APIs gearbeitet	Nein
Kann ER-Modelle lesen	Ja
Kennt das MVC-Pattern	Ja
Kennt das ASP-API-Controller Pattern	Nein
Kennt MQTT	Nein
Kennt LINQ	Nein
Kennt Dependency Injection	Nein

7.8.4.1 Beobachtung 5.2.2

Durchlaufzeit: 20min

Der Proband hat das Prinzip von MQTT früh erkannt und begann zügig mit dem Bau der Regeln. Der Proband hat zunächst Konkrete Regeln komplett ohne Wildcards umgesetzt und hat dann Level pro Level verallgemeinert, bis die Regel korrekt und komplett verallgemeinert war.

7.8.4.2 Feedback Interview 5.2.2

Der Proband hat bestätigt, dass der Durchlauf wie gewünscht funktioniert hat. Er kann sich vorstellen, dass eine Technologie wie MQTT Anwendung im Notfallmanagement wie Feuer in Häusern oder im Schalten von Klimaanlagen gut Anwendung findet. Der Proband wurde sich bei dem Absenden der MQTT-Nachrichten ein Dropdown oder eine Autovervollständigung der Topics wünschen. Er empfand alle Sachverhalte als verständlich und wurde sich als Anleitung für das System eine Reihe an 60-Sekunden-Videos wünschen, die einzelne Bereiche beschreiben. Er würde sich wünschen, dass die Radio-Button in dem System mehr als Radio-Button erkennbar werden, die aktuellen Radio-Button wirken auf Grund der Größe eher wie Lampensymbole.

7.8.4.3 Beobachtung 5.2.4

Durchlaufzeit: 25min

Der Proband hat zum Start des Durchlaufs angemerkt, dass er selten entwickelt. Der Proband hat stark mit anderen Controllern im Vergleich gearbeitet und viele Inhalte aus anderen Controllern übernommen. Bei der Verwendung von LINQ hat er um Hilfe gebeten und eine SQL Lösung angeboten. Die SQL Lösung wurde durch die Aufsichtsperson in LINQ umgewandelt.

7.8.4.4 Feedback Interview 5.2.4

Der Proband hat bestätigt, dass die Implementierung wie gewünscht funktioniert hat, jedoch ohne die Hilfe beim der LINQ-Teil keine Möglichkeit bestand weiter zu kommen ohne viel Zeit in Recherche zu investieren. Der Proband würde an der Software soweit nichts verbessern, wurde aber jedem Entwickler ausgewählte Quellen für LINQ zur Verfügung stellen. Die Dokumentation und das Tutorial aus der ASP-NET Core Dokumentation würde er beibehalten.

7.8.5 Proband T19

Prob	
Datum	19.08.2019
Studiengang	Wirt. Inf.
Laut Selbsteinschätzung technikaffin	Ja
Laut Selbsteinschätzung programmieraffin	Ja
Fachliches Semester	5
Programmiert mit OO-Sprachen	C++ (seit 2017)
Bereits mit REST-APIs gearbeitet	Ja
Kann ER-Modelle lesen	Ja
Kennt das MVC-Pattern	Nein
Kennt das ASP-API-Controller Pattern	Nein
Kennt MQTT	Nein
Kennt LINQ	Ja
Kennt Dependency Injection	Nein

7.8.5.1 Beobachtung 5.2.2

Durchlaufzeit: 12min

Die Probandin hat das Prinzip von MQTT für erkannt und in dem Modell zu nächst MQTT-Nachrichten versendet, um die Elemente im Modell wieder zu finden. Danach hat sie angefangen Regeln zu definieren; erst zwei Regeln ohne Variable, dann direkt die Gültige Lösung. Paar Minuten wurden eingesetzt, um auszuprobieren, ob es eine bessere (noch allgemeinere) Lösung gibt. Nach ca. 8min war die Richtige Rule gefunden. Nach weiteren 4 Minuten waren alternative Regeln durchgesprochen. Dabei wurde eine Regel erstellt, die einen Loop auslöst. Der Loop wurde erst in der Nachbesprechung bekannt.

7.8.5.2 Feedback Interview 5.2.2

Die Probandin hat bestätigt, dass der Durchlauf wie gewünscht funktioniert hat, bis auf den Fall, dass Versuchsleiter in der Lage sind Regeln anzugeben, die einen Loop erzeugen können. Die Probandin wünscht sich aktuelle keine Änderungen oder Verbesserungen des Systems. Als Einführung für die Benutzung des Systems wünscht sie sich eine Anleitung mit Stichpunkten und bebilderten Beispielen.

7.8.5.3 Beobachtung 5.2.4

Durchlaufzeit: 32min

Die Probandin hat zum Start des Durchlaufs selbst angemerkt, dass die Struktur wie MVC ist. Damit die Aussage nicht ohne Antwort bleibt, wurde die Aussage bestätigt und ergänzt. Die Ergänzung durch folgende Anmerkung statt: „Wie MVC nur ohne V, da keine View in dem System mitausgeliefert wird“. Die Probandin hat länger gebraucht um die Datenbank-Verbindung als DB-Context hinzufügen. Nach einigen Minuten wurde der Hinweis gegeben inklusive der Information, dass der Context über den Konstruktor geladen wird.

Nach 15min wurde ein Controller gefunden, der die Lösung zu 90% liefert und das Ergebnis beschrieben. Darauf hin wurde die Aufgabe soweit, angepasst, dass nun die Logs der Session, statt der Logs der SessionRun geladen werden sollen. Nach 20min war die Methode des allgemeinen Controllers fertig. Nach 32min war die Methode fertig, die die Logs des Session lädt.

7.8.5.4 Feedback Interview 5.2.4

Die Probandin hat bestätigt, dass der Durchlauf wie gewünscht funktioniert hat, aber hinweise bzgl. DB-Context nötig waren. Sie merkte an, dass direkt nach der Vorlesung DBWT wohl weniger Hilfestellung notwendig gewesen wäre. Die Probandin hat sich nach der Wahl der Entwicklung Umgebung Visual Studio und der Wahl der Programmiersprache erkundigt. Die Probandin fand, die Aufgabe und den allgemeinen Aufbau der Software weniger komplex als in Implementierungen im DBWT-Praktikum. Sie würde in der Software zunächst nichts ändern, merkt aber an das die Begriffe wie die Rückgabe Typen der Controller-Methoden deutlicher beschrieben werden sollten, wenn diese Technik für die Entwickler neu ist. Als Anleitung findet, die Probandin das Tutorial aus der ASP.NET Core Dokumentation gut. Ggf. würde ein Video noch besser funktionieren. Am besten hat der Probandin die klare Struktur der Software gefallen. Als kritisch sieht sie, dass ohne eine Auffrischung dieses System wohl zu neu für einen Studenten ist, der nicht regelmäßig damit Kontakt hat.

7.8.5.5 Beobachtung 5.2.3

Durchlaufzeit: 14min

Die Probandin hat das Prinzip von REST erkannt, braucht jedoch Erklärung zu den http-Verben. Nach der Erklärung hat sie selbständig Beispiele gemacht vor Sie an die Eigentliche Aufgabe ging. Diese Aufgabe hat sie dann sehr schnell umgesetzt und über die REST-Schnittstelle getestet, die MQTT-Nachrichten versendet.

7.8.5.6 Feedback Interview 5.2.3

Die Probandin hat bestätigt, dass der Durchlauf wie gewünscht funktioniert hat und würde diese Aufgabe gerne nochmal machen: „Können wir das nochmal machen“. Sie merkt an, dass es cool ist, dass alles mit unterschiedlichen Tools funktioniert. Sie würde aktuell nichts verbessern. In einer Anleitung würde sie sich wünschen, das erklärt wird, wie man mit Postman arbeitet und eine Erklärung, wie man die API liest. Am besten hat ihr gefallen, dass die Struktur klar ist und „alles mit allem funktioniert“. Details waren nicht immer klar, dazu muss die Doku genauer werden.

7.8.6 Proband / Expertin S15

Proband	
Datum	19.08.2019
Studiengang	Mitarbeiterin im SMILE Projekt
Laut Selbsteinschätzung technikaffin	Ja
Laut Selbsteinschätzung programmieraffin	Ja
Fachliches Semester	abgeschlossen M. Sc.
Programmiert mit OO-Sprachen	C++ (seit 2013)
Bereits mit REST-APIs gearbeitet	Nein
Kann ER-Modelle lesen	Ja
Kennt das MVC-Pattern	Nein
Kennt das ASP-API-Controller Pattern	Nein
Kennt MQTT	Ja
Kennt LINQ	Nein
Kennt Dependency Injection	Nein

Die der Probandin / Expertin ist eine Mitarbeiterin des SMILE-Projekts und kenn die Software auch seid des Erstellungsprozesses. Das Interview wurde paarallel zu der Nutzung der Software durchgeführt.

Während des Interviews wurden diverse Vorschläge für das Anpassen der GUI angebracht:

- Aus dem MQTT Kontext existiert die Abkürzung Q.o.S, die Probandin merkt an, dass es Sinn macht diese Abkürzung mit einem, Hovertext auszustatten oder einer Anderen Information, da die Begrifflichkeit nicht im Allgemeinem Verständnis genutzt wird. Zusätzlich sollte ein Default-Wert ausgewählt sein, der den Default-Fall abdeckt.
- In der Aktuellen Version sind Symbole in den Menü-Bands links vorhanden aber nicht individualisiert. Das individualisieren der Symbole kann dabei helfen die Begriffen intuitiver zu verstehen.
- In der Regelanwahl für die Subjects werden die Topics voller Länge der Topics aufgelistet, dadurch wird viel Platz für die Radioboxen verbraucht und diese sehr klein angezeigt. Das zu kleine Darstellen der Radioboxen kann zu Missverständnissen führen.
- In der GUI werden unterschiedliche Dropdowns verwendet. Die Dropdowns, die sich auf Session oder SessionRun beziehen, sollten die ID sichtbar anzeigen, um den Bezug zu den Session einfacher herstellen zu können.
- Im Dashboard werden aktuelle alle Rules angezeigt, die in der Datenbank vorhanden sind. Im Dashboard sollten nur die Rules angezeigt werden, die zu der aktuellen SessionRun gehören damit der Versuchsleiter das Verhalten anhand der Rule ableiten kann. Zusätzlich sollte aber die Erklärung geben werden, dass nur Rules angezeigt werden, die zur aktuellen SessionRun gehören.
- Um das Verständnis der Software zu erhöhen sollte eine Grafik wie ein ER-Modell angezeigt werden können.

Bei Testen der Software auf Fehleingaben konnten Fehler forciert werden, der persistiert wurde.

7.9 Experteninterview

K (K12) und B (B12) haben angeboten ein Code-Review der Mittelschicht durchzuführen. Dieser Code-Review soll dabei helfen, die Mittelschicht besser darauf vorzubereiten durch Dritte weiter entwickelt zu werden.

7.9.1 Experte B12

Das Interview wird nicht als Transkription, sondern als Zusammenfassung angeführt, die durch den Experten bestätigt wurde.

B (B12) ist Student der FH Aachen und Mitarbeiter bei GridX. GridX ist ein junges Softwareunternehmen in Aachen mit starkem Technischen Know-how im Bereich Development, DevOps und Energiemanagement. B12 arbeitet privat an diversen Community-Projekten im Softwarebereich und bringen praktisches Wissen genau in diesen Bereichen (vgl. GitHub (Benedikt 2015)).

Indem Interview wurde der generelle Aufbau der Software als positiv wahrgenommen, da es wir ein Standard ASP.net core Projekt aussieht. Das Design zur Dependency-Inversion ist ebenfalls einfach und gut nachvollziehbar.

Die Art wie Dependency Injection (DI) verwendet wurde, hat zu Diskussionen geführt, da die DI in C# anders umgesetzt wird, als in anderen Sprachen wie GO. In dem Zusammenhang wurde auch die Art der Verwendung von Interfaces kritisch betrachtet. Interfaces sind in der Betrachtung von B12 verhaltens-definierend und damit sind die Interfaces der unterschiedlichen `ServiceManager` vereinheitlichbar. B12 nannte zwei Aussagen, die den Grundgedanken genau treffen: „Ein interface beschreibt das grundlegende verhalten des darunter liegenden Objektes… Umso kleiner das Interface umso besser die Abstraktion“. Als Beispiel haben sowohl das Interface `IManageRuleService` und das Interfaces `IManageLogService` eine Definition, die den jeweiligen Service startet. Diese beiden Interfaces lassen sich abstrakt zusammenfassen. Gestützt wird die These von B12 durch einen Satz, der sich im C#-Programmierhandbuch finden lässt: „Eine Schnittstelle enthält Definitionen für eine Gruppe von zugehörigen Funktionalitäten, die von einer Klasse oder einer Struktur implementiert werden können.“ (vgl. docs.ms (Olprod (Microsoft), OpenLocalizationService, et al. 2018)).

Unter der Schlüsselwortdefinition in der C#-Referenz für interface lässt sich dagegen ein generischer Ansatz finden „Eine Schnittstelle enthält nur die Signaturen von Methoden, Eigenschaften, Ereignissen oder Indexern. Eine Klasse oder eine Struktur, die die Schnittstelle implementiert, muss die Member der Schnittstelle implementieren, die in der Schnittstellendefinition angegeben werden.“ (vgl. docs.ms (Olprod (Microsoft), OpenLocalizationService (Microsoft), et al. 2018)).

Während des Interviews ließ sich letztlich herausstellen, dass beide Sichtweisen korrekt sind jedoch unterschiedliche Anwendungsfälle für Interfaces beschreiben. Die Interfaces wie `IManageRuleService` beschreiben die Funktionalität des `MangerRuleService` als den Manager, der den Rule Service managet. Zu diesem Interface dazu gehören die Methoden dessen Signaturen in den Interfaces definiert und in der implementierenden Klasse umgesetzt sind. Durch die Dependency Injection werden diese Interfaces genutzt, um genau eine spezielle Klasse zuzuordnen. Diese Zuordnung findet an einem zentralen Punkt im Projekt statt. Überall, wo eine Klasse geladen werden soll, wird das Interfaces dieser Klasse angefragt. Dadurch, dass an einer Stelle im System diese Klasse fest diesem Interface zugeordnet wurde, lässt sich eindeutig diese Klasse laden. Wenn

es im Laufe der Entwicklung des Projektes nötig sein sollte eine andere Klasse für die Funktionalität dieses Interfaces zu nutzen, verbindet man eine neue Klasse. Über die Dependency Injection wird die neue Klasse geladen, ohne das der andere Code angepasst werden muss.

Damit das Laden über Dependency Injection funktioniert, ist es also notwendig, dass in dem Programm, den Interfaces, die für Dependency Injection genutzt werden, nur eine Klasse zugeordnet ist. Anders verhält es sich mit Interfaces, die Standardverhalten wie die Methode ToString von IFormattable definieren oder der Kombination von IObservable und IObserver. Diese Interfaces werden durch viele Klassen implementiert. Dabei ist dann der Sinn, dass diese Klassen ein gemeinsames Verhalten haben. Offensichtlich dabei ist, dass wenn viele Klassen ein gemeinsames Verhalten haben sollen, diesem Interface keine eindeutige Klasse zugeordnet werden kann. Damit verhalten sich die Interfaces für das Laden über Dependency Injection anders als die Interfaces, die für das Definieren von Standartschnittstellen genutzt werden.

Da in den aktuellen Stand der Software noch keine Authentifizierung implementiert ist, hat B12 auf das Middleware Pattern verwiesen und angesprochen, dass sich über eine Middleware ideal die Authentifizierung und Autorisierung abdecken lässt. (Nach dem Interview gefunden: (Anderson and Smith 2019)).

Als letztes und wurde noch auf Automatisierte Tests hingewiesen, da in dieser Arbeit nur wenig Gebrauch davon gemacht wurde. Durch automatisierte Tests wird die Code- und Software-Qualität stark gesteigert und auf lange Sicht preiswert hochgehalten. Laut B12 sind Automatisierte Tests ein Schlüssel für professionelle kleine und große Teams.

7.9.2 Experte K12

Das Interview wird nicht als Transkription, sondern als Zusammenfassung angeführt, die durch den Experten bestätigt wurde.

K (K12) ist Student der FH Aachen und Mitarbeiter bei GridX, einem jungen Softwareunternehmen in Aachen mit starkem Technischen Know-how im Bereich Development, DevOps und Energiemanagement. Wie auch B12 arbeitet K12 privat an diversen Community-Projekten im Softwarebereich und bringt praktisches Wissen genau in diesen Bereichen (vgl. GitHub (Knut 2017)). Knut Zuidema hat mehr Zeit mit C# und Dot.Net core verbracht als B12 und ist daher eher an die Strukturen gewöhnt.

Indem Interview hat K12 die Architektur als gut und nicht außergewöhnlich wahrgenommen, was generell für eine einfach zu verstehende Struktur spricht. Im Bezug auf die Interfaces ist nichts aufgefallen, was C# untypisch wäre. Die Dependency Injection, die zum Teil durch B12 bemängelt wurde, sieht für K12 aus wie gewohnt.

Auf die Frage was er als Ehesten ändern würde nannte K12, dass die Architektur so bleiben soll und kann. Er würde aber im Code am ehesten die Inkonsistenz der Sprache und damit den Wechsel zwischen Deutsch und Englisch auflösen. Als zweiten Punkt würde er im Code den Style bzgl. der Art der Variablen Benennung vereinheitlichen, da an ein paar Stellen der Style gebrochen ist.

Im Zusammenhang mit dem Style würde er anderen Einstellungen in der IDE oder eine andere IDE empfehlen, die einen bei dem Prozess unterstützt. Im Punkt der Unterstützung wünscht sich K12 deutlich mehr automatisierte Tests, damit die Qualität sichergestellt ist und auch nach einer Übergabe erhalten bleibt. Im Refactoring von Code können sich Fehler einschleichen, die durch diese Tests geringgehalten werden oder gar nicht erst entstehen.

Auf die Frage welche Dokumentation sich K12 wünschen würde, nannte er vollständige Klassen und Methodenbeschreibung in Code über JavaDoc oder C#Doc angefangen mit den Interfaces. Als jemand der die API benutzen soll würde er sich eine Endpunktbeschreibung wünschen.

7.10 Fotos SMILE Exponat

Abbildung 81: SMILE Haus5, eine Arzt-Praxis aus der 1zu18-Modelllandschaft des SMILE Projekts

Abbildung 82: SMILE Haus3, ein Haus aus der 1zu18-Modelllandschaft des SMILE Projekts

Abbildung 83SMILE Haus2, das am Besten ausgestattete Haus der 1zu18-Modelllandschaft des SMILE Projekts

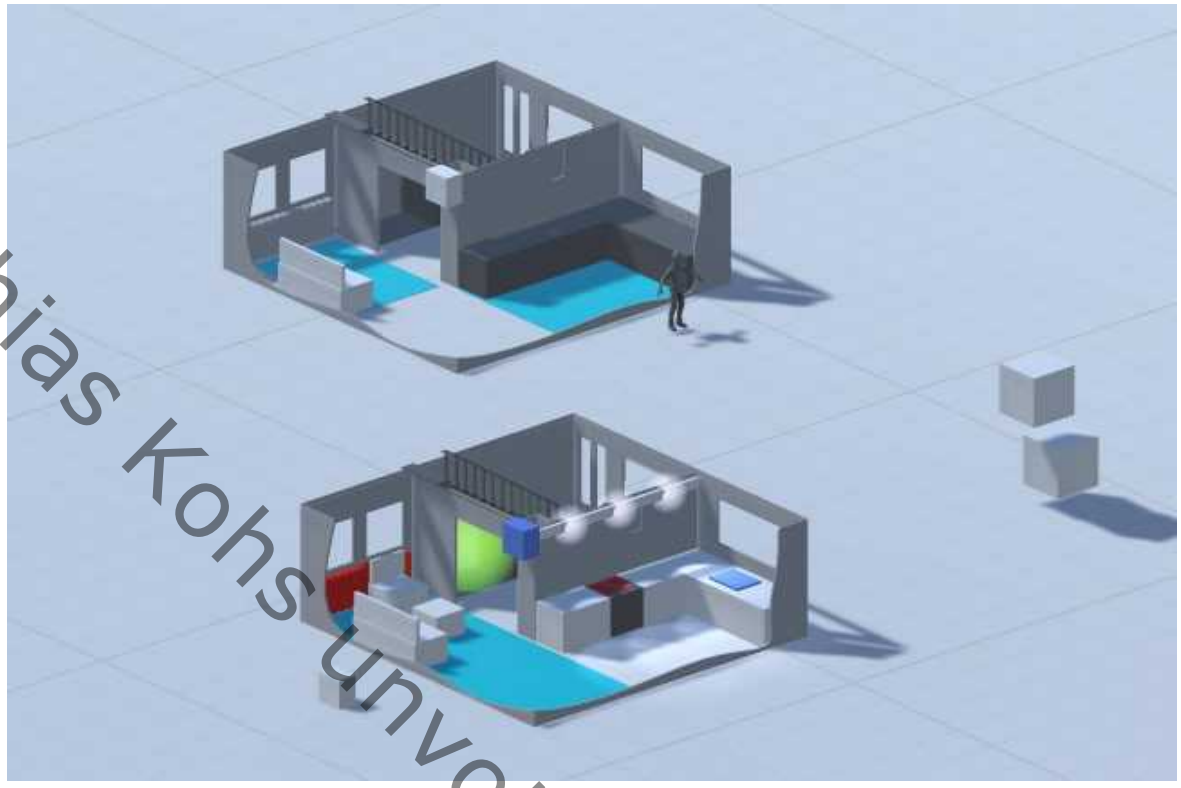


Abbildung 84: sehr einfache 3D Zwillinge der Häuser 2 und 3 der SMILE Modelllandschaft

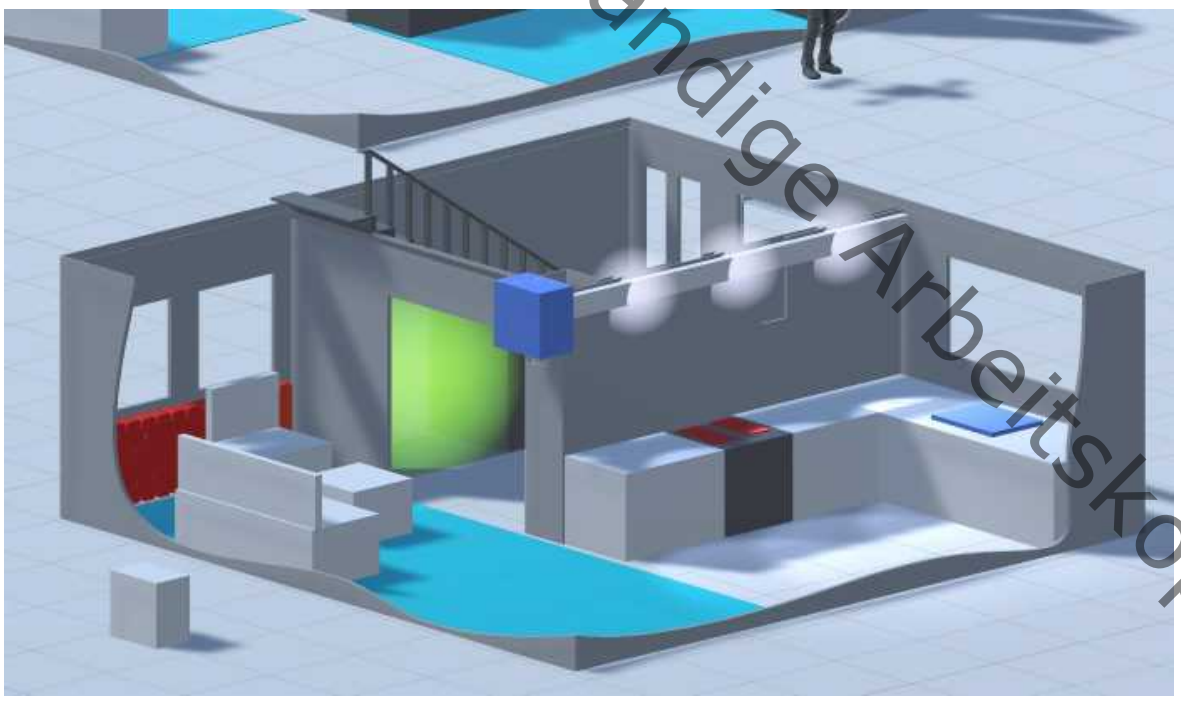


Abbildung 85: Ein sehr einfacher 3D Zwillinge des Häus 2 der SMILE Modelllandschaft

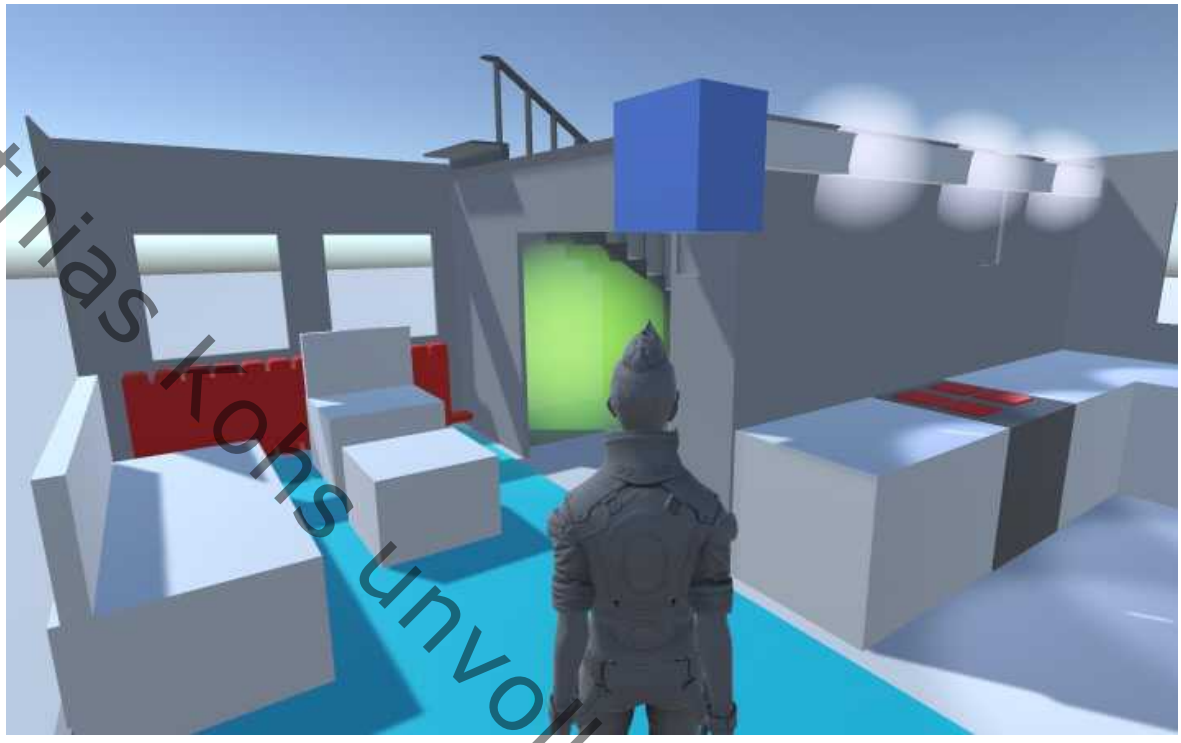


Abbildung 86: 3rd-Person-Sicht auf den einfach gehaltenen 3D-Zwilling des Haus 2 der SMILE Modelllandschaft

8 Abbildungsverzeichnis

Abbildung 1: Google-Trends Mqtt coap xmpp von informatik-aktuell.de 2
Abbildung 2: Google-Trends Mqtt coap xmpp Q1 2019 3
Abbildung 3: Google Trends Mqtt ZigBee Z-Wave Q1 2019 3
Abbildung 4: Netzdiagramm mit 6 Merkmalen der Dienstgüte für den Prototypen (Richtwert) 4
Abbildung 5: Ableitung (Generalisierung / Spezialisierung) von Stakeholdern 10
Abbildung 6: Meta-Modell des zu erstellenden Gesamtsystems (SOLL-Situation) 14
Abbildung 7: Meta-Modell der IST-Situation 15
Abbildung 8: Use Case-Diagramm SimLeiter nutzt ZAS 18
Abbildung 9: Use-Case-Diagramm SimLeiter GerätManipulieren 20
Abbildung 10: Use-Case-Diagramm SimLeiter WerteManipulieren 20
Abbildung 11: Kategorien für nicht-funktionale Anforderungen nach den 'SOPHISTen' 22
Abbildung 12: Mindmap mit Auswahl vernetzer Anforderungsmerkmale und Kategorien 26
Abbildung 13: Klassendiagramm mit voller Abhängigkeit der Application von der konkreten Klasse 29
Abbildung 14: Klassendiagramm mit Abhängigkeitsumkehr durch Anwendung Abstract Factory 29
Abbildung 15: Beispiel eines MQTT-Topic-Baum von Starthomeblog.net 43
Abbildung 16: Anbindung von Unity3D an einen MQTT-Broker 46
Abbildung 17: UML generiert mit Doxygen Sichtbarkeit des Singleton des MqttConnector 48
Abbildung 18: Screenshot eines UnityEvent Bausteins (Brick) 50
Abbildung 19: UML Klassendiagramm abstrakte Klasse AbsDoActionByMqttTopicValue 51
Abbildung 20: UML-Vererbungsdiagramm generiert durch Doxygen aus dem MqttConnector 53
Abbildung 21: UML-Vererbungsdiagramm generiert durch Doxygen aus dem M2MqttUnityClient 53
Abbildung 22: Baustein und Abgrenzungssicht der Mittelschicht 59
Abbildung 23: Vereinfachter Aufbau einer ASP.NET Core API-App auf Basis von MVC 60
Abbildung 24: Geräte in dem Verfahren des Topic Trenner 61
Abbildung 25: Logische Ansicht des Topic Trenners ohne Broker 62
Abbildung 26: Topics naiv in einen Graphen überführt 67
Abbildung 27: ER-Modell zum Abbilden von TopicParts, die Oberparts (bzw. Vorgänger-Knoten) haben 68
Abbildung 28: Topics in eine Baumstruktur überführt 68
Abbildung 29: ER-Modell der Mittelschicht 69
Abbildung 30: LogServiceManager eine Implementierung von IManageLogService, die IServeLogging 71
Abbildung 31: LogServiceInJsonFile eine Implementierung von IServeLogging 71
Abbildung 32: Klassendiagramm EventService hält eine Referenz auf DbContext 72
Abbildung 33: Klassendia. Dependency Inversion EventMessageController, EventService, DbContext 73
Abbildung 34: Klassendiagramm Controller Manager Service Abhängigkeitsdarstellend 74
Abbildung 35: Klassendiagramm Controller Manager Service Muster am Beispiel des EventService 75
Abbildung 36: Teil des Meta-Modells des Gesammtsystems zum aktuellen Stand 79
Abbildung 37: Teil des Meta-Modells des Gesamtsystems zum aktuellen Stand; Digitalen Zwillingen 80
Abbildung 38: Teil des Meta-Modells des Gesamtsystems; Versuch zum Anbinden von neuen Systemen 82
Abbildung 39: Teil des Meta-Modells des Gesamtsystems zum aktuellen Stand mit Fokus auf die API 83
Abbildung 40: Teil des Meta-Modells des GS zum aktuellen Stand, mit Fokus auf die Mittelschicht 85
Abbildung 41: Teil des Meta-Modells des GS zum aktuellen Stand, mit Fokus auf eine Konfiguration 86
Abbildung 42: Benchmark auf Entwicklungshardware H1 94
Abbildung 43: Diagramm der Reaktionszeitzeitverläufe in Millisekunden pro Message 95
Abbildung 44: Diagramm der Reaktionszeitverläufe in Millisekunden pro Message 95
Abbildung 45: Benchmark auf Projekthardware H3 96
Abbildung 46: Dia. der Reaktionszeitzeitver. in Millisek. / Message auf Hardware H3 für SP, TP2, TPD 97
Abbildung 47: Diagramm der Reaktionszeitzeitver. in Millisek / Message auf Hardware H3 für TPD 98
Abbildung 48: UML-Klassendiagramm generiert mit Doxygen Ableitung diverser MQTT-Bricks von der 109
Abbildung 49: Klassendiagramm Controller Manager Service mit Diamantsymbolik für Aggregationen 110

Abbildung 50 Beispiel für Supported frameworks und MQTT Versionen 112
Abbildung 51: Screenshot des VRTK - Virtual Reality Toolkit im Unity Asset Store, mit der Information . 112
Abbildung 52: Screenshot der Rangliste nach dem PYPL-Index vom 30.04.2019 113
Abbildung 53: Screenshot des VRTK Forums mit der Information, dass Steam VR v2 in VRTK v3.3 113
Abbildung 54: Screenshot des State of the Octoverse der Programmiersprachen auf GitHub 114
Abbildung 55: Screenshot des TIOBE-Index am 30.04.2019 114
Abbildung 56: SMILE MQTT Mittelschicht Webapp Dashboard 115
Abbildung 57: SMILE MQTT Mittelschicht Webapp Dashboard SessionRun ist aktiv 116
Abbildung 58: SMILE MQTT Mittelschicht Webapp Sessions List 117
Abbildung 59: SMILE MQTT Mittelschicht Webapp Add Session 118
Abbildung 60: SMILE MQTT Mittelschicht Webapp Edit Session 119
Abbildung 61: SMILE MQTT Mittelschicht Webapp SessionRuns List 120
Abbildung 62: SMILE MQTT Mittelschicht Webapp Add SessionRun 121
Abbildung 63: SMILE MQTT Mittelschicht Webapp Edit SessionRun 122
Abbildung 64: SMILE MQTT Mittelschicht Webapp Rules List 123
Abbildung 65: SMILE MQTT Mittelschicht Webapp Add Rule 124
Abbildung 66: SMILE MQTT Mittelschicht Webapp Edit Rule 125
Abbildung 67: SMILE MQTT Mittelschicht Webapp Remove Rules 126
Abbildung 68: SMILE MQTT Mittelschicht Webapp Subjects List 127
Abbildung 69: SMILE MQTT Mittelschicht Webapp Add Subject 128
Abbildung 70: SMILE MQTT Mittelschicht Webapp Edit Subject 129
Abbildung 71: SMILE MQTT Mittelschicht Webapp PurposeMessages List 130
Abbildung 72: SMILE MQTT Mittelschicht Webapp Add Purposemessage 131
Abbildung 73: SMILE MQTT Mittelschicht Webapp Edit PurposeMessage 132
Abbildung 74: : SMILE MQTT Mittelschicht Webapp Log 133
Abbildung 75: SMILE MQTT Mittelschicht Webapp Log Topic 134
Abbildung 76: SMILE MQTT Mittelschicht Webapp Wert 135
Abbildung 77: SMILE MQTT Mittelschicht Webapp Eventmessages List 136
Abbildung 78: SMILE MQTT Mittelschicht Webapp Add Eventmessage 137
Abbildung 79: SMILE MQTT Mittelschicht Webapp Edit Eventmessage 138
Abbildung 80: SMILE MQTT Mittelschicht Webapp Add Subject-Rules 139
Abbildung 81: SMILE Haus5, eine Arzt-Praxis aus der 1zu18-Modelllandschaft des SMILE Projekts 156
Abbildung 82: SMILE Haus3, ein Haus aus der 1zu18-Modelllandschaft des SMILE Projekts 157
Abbildung 83SMILE Haus2, das am Besten ausgestattete Haus der 1zu18-Modelllandschaft 158
Abbildung 84: sehr einfache 3D Zwillinge der Häuser 2 und 3 der SMILE Modelllandschaft 159
Abbildung 85: Ein sehr einfacher 3D Zwillinge des Häus 2 der SMILE Modelllandschaft 159
Abbildung 86: 3rd-Person-Sicht auf den einfach gehaltenen 3D Zwill des Haus 2 der Modelllandschaft. 160

9 Tabellenverzeichnis

Tabelle 1: Stakeholder Projektleiter 10
Tabelle 2: Stakeholder Endanwender 11
Tabelle 3: Stakeholder Probanden 11
Tabelle 4: Stakeholder Versuchsleiter 11
Tabelle 5: Stakeholder Entwickler 12
Tabelle 6: Stakeholder Softwarearchitekt 12
Tabelle 7: Stakeholder Administrator 13
Tabelle 8: Stakeholder Projektpartner 13
Tabelle 9: Programmiersprachen im Projekt 15
Tabelle 10: Hardware H1 (Entwicklungsumgebung) 24
Tabelle 11: Hardware H2 (VR Workstation) 24
Tabelle 12: Hardware H3 (Raspberry Pi) 25
Tabelle 13: Betriebssystem OS1 Windows 25
Tabelle 14: Betriebssystem OS2 Linux 25
Tabelle 15: SOLID-Ziele zu Merkmalen der Qualitätsanforderungen 27
Tabelle 16: Auswahl von VR-Brillen und SDKs 32
Tabelle 17: Verrechnungstabelle der Programmiersprachen Indices 37
Tabelle 18: Bewertungsmatrix Programmiersprachen 38
Tabelle 19: Bewertungsmatrix der Entwicklungsvarianten der Mittelschicht 58
Tabelle 20: Belastungstest 92
Tabelle 21: Finiale Technologiewahl MQTT-Lib für Unity3D 140
Tabelle 22: Technologiewahl MQTT-Lib für Unity3D Stufe 1-2 141

10 Literaturverzeichnis

(HiveMQ), The HiveMQ Team. 2015. “MQTT Essentials Part 5: MQTT Topics & Best Practices.” 2015. https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/.

———. 2017. “MQTT 5 Features & Hidden Gems: Introduction to MQTT 5.” Hivemq. 2017. https://www.hivemq.com/blog/mqtt-5-introduction-to-mqtt-5/.

———. 2019a. “HiveMQ Communit Edition TokenizedTopicMatcher on GitHub.” Github. 2019. https://github.com/hivemq/hivemq-community-edition/blob/master/src/main/java/com/hivemq/mqtt/topic/TokenizedTopicMatcher.java.

———. 2019b. “HiveMQ Community Edition on GitHub.” Github. 2019. https://github.com/hivemq/hivemq-community-edition.

———. 2019c. “HiveMQ MQTT.” Hivemq. 2019. https://www.hivemq.com/mqtt/.

———. 2019d. “HiveMQ Plugins.” Hivemq.Com/Docs. 2019. https://www.hivemq.com/docs/3.4/hivemq/plugins.html.

(IETF), Internet Engineering Task Force. 2014. “Rfc7159 JSON.” https://tools.ietf.org/html/rfc7159.

———. 2017. “Rfc8259 JSON.” https://tools.ietf.org/html/rfc8259.

(ITOM/FH-Aachen). 2019. “GHOST Forschungsprojekt Webseite.” 2019. http://www.itom.fh-aachen.de/index.php/forschung/ghost.

(mqtt.org). 2019. “Mqtt.Org FAQ.” 2019. http://mqtt.org/faq.

(Octoverse), Telliott27. 2018. “The State of the Octoverse: Top Programming Languages of 2018.” Github. 2018. https://github.blog/2018-11-15-state-of-the-octoverse-top-programming-languages/.

(VRTK-Forum), Scraft. 2019. “VRTK Forum FAQ.” VRTK Forum. 2019. https://flarum.vrtk.io/d/2-faq.

Anderson, Rick (Microsoft), and Steve (Microsoft) Smith. 2019. “ASP.NET Core Middleware.” Docs.Microsoft.Com. 2019. https://docs.microsoft.com/de-de/aspnet/core/fundamentals/middleware/?view=aspnetcore-2.2.

Anderson, Rick (Microsoft), and Mike (Microsoft) Wasson. 2019. “Tutorial: Erstellen Einer Web-API Mit ASP.NET Core.” Docs.Microsoft.Com. 2019. https://docs.microsoft.com/de-de/aspnet/core/tutorials/first-web-api?view=aspnetcore-2.2&tabs=visual-studio.

Armase, and trecoolman. 2017. “MQTT in Unity3D for Leap Motion on Android.” Reddit. 2017. https://www.reddit.com/r/Unity3D/comments/4gdy3a/mqtt_in_unity3d_for_leap_motion_on_android/.

Banks, Andrew (IBM), Ed (Microsoft) Briggs, Ken (IBM) Borgendale, and Rahul (IBM) Gupta. 2019. “MQTT Version 5.0 Candidate OASIS Standard 02.” *OASIS Standard*. http://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf.

Banks, Andrew, Ed Briggs, Ken Borgendale, and Rahul Gupta. 2018. “MQTT Version 5.0 Candidate OASIS Standard 01.” OASIS Standard. 2018. http://docs.oasis-

open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf.

Basili, VR, and AJ Turner. 1975. *Iterative Enhancement: A Practical Technique for Software Development. IEEE Transactions on Software Engineering 1*. Vol. 4.

Bass, Len, Paul Clements, and Rick Kazman. 2012. *Software Architecture in Practice*. Pearson Education.

Benedikt, Bongartz. 2015. "Benedikt Bongartz GitHub Profil." Github. 2015. https://github.com/frzifus.

Bracht, Uwe, Dieter Geckler, and Sigrid Wenzel. 2018. "Anwendungsfelder Der Digitalen Fabrik Im Überblick." Digitale Fabrik. Berlin, Heidelberg: Springer Berlin Heidelberg. 2018. https://doi.org/10.1007/978-3-662-55783-9_2.

C., David. 2018. "What Is MQTT and How to Use It with OpenHab? - The Smart Home Blog." Smarthomeblog.Net. 2018. https://www.smarthomeblog.net/mqtt-openhab/.

chkr1011. 2018. "MQTT-Lib MQTTnet C#." Github. GITHUB. 2018. https://github.com/chkr1011/MQTTnet.

Crockford, D. 2006. "Rfc4627 JSON." JSON.org. https://tools.ietf.org/html/rfc4627.

Cunningham, Ward, Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, and Martin Fowler. 2001. "Manifesto for Agile Software Development." Agilemanifesto.Org. 2001. http://agilemanifesto.org.

Eclipse. 2014. "MQTT-Lib M2Mqtt C#." Github. eclipse / GITHUB. 2014. https://github.com/eclipse/paho.mqtt.m2mqtt.

EdwinChua, and FlutterShift. 2017. "Receiving MQTT Messages on Unity-Answers." Answers.Unity.Com. 2017. https://answers.unity.com/questions/1330095/receiving-mqtt-messages.html.

Ericvoid. 2016. "MQTT-Lib StriderMqtt C#." Github. GITHUB. 2016. https://github.com/ericvoid/StriderMqtt.

ExtendRealityLtd. 2019. "ExtendRealityLtd/VRTK on GitHub." Github. 2019. https://github.com/ExtendRealityLtd/VRTK.

Freeman Eric, Freeman Elisabeth, Sierra Kathy, Bates Bert. 2008. *Entwurfsmuster von Kopf Bis Fuß*. O'REILLY.

Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. 2001. *Design Patterns, Elements of Reusable*. *Applied Java Patterns*.

"Godot Engine WebPage Main." 2019. 2019. https://godotengine.org/.

Google. 2019a. "Google Trends Mqtt, Coap, Xmpp." Trends.Google.De. 2019. https://trends.google.de/trends/explore?date=all&q=mqtt,coap,xmpp.

———. 2019b. "VR Daydream." Vr.Google.Com. 2019. https://vr.google.com/daydream/.

Grösser Stefan (Berner Fachhochschule). 2019. "Digitaler Zwilling Definition." Gabler Wirtschaftslexikon. 2019. https://wirtschaftslexikon.gabler.de/definition/digitaler-zwilling-54371.

Group, The PHP. 2019. "PHP Json_encode." Https://Www.Php.Net/Manual. 2019.

https://www.php.net/manual/de/function.json-encode.php.

Hipple, Ryan (Schell Games). 2017. “Unite Austin 2017 - Game Architecture with Scriptable Objects.” Texas: Unity3D on Youtube.com. https://youtu.be/raQ3iHhE_Kk.

ISO. 2016. “ISO/IEC 20922:2016.” ISO. 2016. https://www.iso.org/standard/69466.html.

It-visions.de. 2017. “Erklärung Des Begriffs: .NET (DOTNET).” Www.It-Visions.De. 2017. https://www.it-visions.de/glossar/alle/306/NET.aspx.

Jacobas, Stephan, Matthias Meinecke, Thomas Ritz, Marko Schuba, Martin Wolf, and Johannes König. 2016. “Antrag Vorhabenbeschreibung SMILE – Smart Infrastructure for Living Environments.” Aachen.

Jpmens. 2019. “Jpmens/Mosquitto-Auth-Plug on GitHub.” Github. 2019. https://github.com/jpmens/mosquitto-auth-plug.

Kaiser, Steffen. 2018. “Bachelorarbeit Entwicklung Einer Serviceorientierten Modellierung von Regeln in Smarten Umgebungen.”

Kaufmann, Timothy, Armin Pühringer, and Benedikt Rauscher. 2016. “Der Digitale Zwilling.” Www.Computer-Automation.De. 2016. https://www.computer-automation.de/unternehmensebene/produktionssoftware/artikel/132264/2/.

Klabnik, Steve, Yehuda Katz, Dan Gebhardt, Tyler Kellen, and Ethan Resnick. 2019. “JSON API.” Jsonapi.Org. 2019. https://jsonapi.org.

Kleuker, Stephan. 2009. *Grundkurs Software-Engineering Mit UML*. Wiesbaden: Vieweg+Teubner. https://doi.org/10.1007/978-3-8348-9235-5.

———. 2011. *Grundkurs Software-Engineering Mit UML: Der Pragmatische Weg Zu Erfolgreichen Softwareprojekten*. Vieweg+Teubner.

Knut, Zuidema. 2017. “KnutZuidema GitHub Profil.” Github. 2017. https://github.com/KnutZuidema.

Kohs, Mathias (MathiasKoAc). 2019. “MathiasKoAc/MKdev.M2Mqtt.Unity3D on GITHUB.” Github. 2019. https://github.com/MathiasKoAc/MKdev.M2Mqtt.Unity3D.

Kotamraju, Jitendra (Oracle). 2013. “JSON Java Oracle.” Https://Www.Oracle.Com. 2013. https://www.oracle.com/technetwork/articles/java/json-1973242.html.

Kuhn, Thomas (Fraunhofer IESE). 2017. “Digitaler Zwilling.” Gi.de/Informatiklexikon. 2017. https://gi.de/informatiklexikon/digitaler-zwilling/.

Kunze, Sariana. 2016. “Neues Aus Der Digital Enterprise-Produktkiste Auf Dem Weg Zu Industrie 4.0.” Www.Elektrotechnik.Vogel.De. 2016. https://www.elektrotechnik.vogel.de/neues-aus-der-digital-enterprise-produktkiste-auf-dem-weg-zu-industrie-40-a-556534/.

Lander, Richard (Microsoft). 2018. “Leitfaden Für .NET Core.” Docs.Microsoft.Com. 2018. https://docs.microsoft.com/de-de/dotnet/core/.

———. 2019. “MS Devblogs Introducing .NET 5.” Devblogs.Microsoft.Com. 2019. https://devblogs.microsoft.com/dotnet/introducing-net-5/.

Latham, Luke, Leif Mensing, Steve Smith, Scott Addie, and olprod. 2019. “Dependency Injection in ASP.NET Core | Microsoft Docs.” Docs.Microsoft.Com. 2019. https://docs.microsoft.com/de-de/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-2.2.

Li, Hong (Microsoft), Olprod (Microsoft), and Saisang (Microsoft) Cai. 2019. .“.Net JSON Data.” Docs.Microsoft.Com. 2019. https://docs.microsoft.com/de-de/dotnet/framework/wcf/feature-details/how-to-serialize-and-deserialize-json-data.

Liskov, Barbara. 1988. “Data Abstraction and Hierarchy.” *ACM SIGPLAN Notices* 23 (5): 17–34. https://doi.org/http://dx.doi.org/10.1145/62138.62141.

Ltd, Sysdia Solutions. 2016. “VRTK - Virtual Reality Toolkit - [VR Toolkit].” Redditch B976HA: Sysdia Solutions Ltd. https://assetstore.unity.com/packages/tools/vrtk-virtual-reality-toolkit-vr-toolkit-64131.

“MakeIt(True).” 2019. 2019. https://www.fh-aachen.de/studium/studentische-projekte/laufende-projekte/makeittrue/.

markallanson. 2013. “MQTT-Lib Nmqtt C#.” Github. GITHUB. 2013. https://github.com/markallanson/nmqtt.

Martin, Robert C. 2017. *Clean Architecture: A Craftsman’s Guide to Software Structure and Design*. Deutsche A. mitp-Verlag.

McGlothin, Robin. 2018. “AKF THE SCALE CUBE.” Akfpartners.Com. 2018. https://akfpartners.com/growth-blog/scale-cube.

mFourLabs. 2016. “MQTT-Lib KittyHawkMQ C#.” Github. GITHUB. 2016. https://github.com/mFourLabs/KittyHawkMQ.

Microsoft. 2017. “Microsoft Mixed Reality Headsets.” Blogs.Windows.Com. 2017. https://blogs.windows.com/windowsexperience/2017/10/03/how-to-pre-order-your-windows-mixed-reality-headset/#yKaDYKoX3HVPAJCU.97.

———. 2019a. .“.Net API Blocking Collection.” Docs.Microsoft.Com. 2019. https://docs.microsoft.com/de-de/dotnet/api/system.collections.concurrent.blockingcollection-1?view=netframework-4.7.2.

———. 2019b. .“.Net API Queue.” Docs.Microsoft.Com. 2019. https://docs.microsoft.com/de-de/dotnet/api/system.collections.queue?view=netframework-4.7.2.

Monostori, L., B. Kádár, T. Bauernhansl, S. Kondoh, S. Kumara, G. Reinhart, O. Sauer, G. Schuh, W. Sihn, and K. Ueda. 2016. “Cyber-Physical Systems in Manufacturing.” CIRP Annals. 2016. https://doi.org/10.1016/j.cirp.2016.06.005.

Mqtt.org, community members. 2014. “Mqtt/Mqtt.Github.Io.” Github. 2014. https://github.com/mqtt/mqtt.github.io/wiki/software?id=software.

———. 2018. “Mqtt/Mqtt.Github.Io/Wiki/Libraries.” https://github.com/mqtt/mqtt.github.io/wiki/libraries.

“MQTT Version 3.1.1 OASIS Standard.” 2015. OASIS Standard. 2015. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.pdf.

“No CryEngine WebPage Main.” 2019. 2019. https://www.cryengine.com/.

Nystrom, Robert. 2014. “GameprogrammingPatternsDouble Buffer.” Gameprogrammingpatterns.Com. 2014. https://gameprogrammingpatterns.com/double-buffer.html.

———. 2015. *Design Patterns Für Die Spieleprogrammierung*. mitp-Verlag.

OASIS. 2014. “MQTT Version 3.1.1 OASIS Standard.” OASIS Open. 2014. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718022.

Obermaier, Dominik (HiveMQ). 2015. “IoT-Protokolldschungel – Ein Wegweiser.” Www.Informatik-Aktuell.De. 2015. https://www.informatik-aktuell.de/betrieb/netzwerke/iot-protokolldschungel-ein-wegweiser.html.

Object Management Group (OMG). 2017. “Unified Modeling Language (UML) Specification V2.5.1.” *Omg.Org*, 796. https://www.omg.org/spec/UML/.

Oculus. 2019. “VR Oculus Rift.” 2019. https://www.oculus.com/rift/#.

Olprod (Microsoft), and OpenLocalizationService (Microsoft). 2017. “Sprachintegrierte Abfrage (Language-Integrated Query, LINQ).” Docs.Microsoft.Com. 2017. https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide/concepts/linq/index.

Olprod (Microsoft), and OpenLocalizationService (Microsoft). 2017. “Arbeiten Mit LINQ.” Docs.Microsoft.Com. 2017. https://docs.microsoft.com/de-de/dotnet/csharp/tutorials/working-with-linq.

Olprod (Microsoft), OpenLocalizationService (Microsoft), Saisang (Microsoft) Cai, and Yishengjin1413 (Microsoft). 2018. “Interface (C# Reference).” Docs.Microsoft.Com. 2018. https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/interface.

Olprod (Microsoft), OpenLocalizationService, Hung Truong, Saisang Cai, and Yishengjin1413. 2018. “Interfaces (C# Programming Guide).” Docs.Microsoft.Com. 2018. https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide/interfaces/index.

Parkerhill Reality, Labs. 2018a. “Bridge XR Unity AssetStore.” Assetstore.Unity.Com. 2018. https://assetstore.unity.com/packages/tools/utilities/bridgexr-116811.

———. 2018b. “Parkerhill Bridge XR.” Http://Www.Parkerhill.Com. 2018. http://www.parkerhill.com/bridgexr.

Patierno, Paolo (Microsoft). 2015. “M2Mqtt for .Net : MQTT Client for Internet of Things & M2M Communication.” Code.Msdn. 2015. https://code.msdn.microsoft.com/windowsdesktop/M2Mqtt-MQTT-client-library-ac6d3858.

Peter Pin-Shan Chen. 1976. “The Entity-Relationship Model--Toward a Unified View of Data.” *ACM Transactions on Database Systems*. https://dspace.mit.edu/bitstream/handle/1721.1/47432/entityrelationshx00chen.pdf.

Project, Mono. 2019. “Https://Www.Mono-Project.Com/.” Mono-Project. 2019. https://www.mono-project.com/.

PYPL. 2019. “PYPL-Index.” Github. 2019. http://pypl.github.io/PYPL.html.

Python Software Foundation. 2019. “Python JSON Encoder and Decoder.” Https://Docs.Python.Org. 2019. https://docs.python.org/3/library/json.html.

Ralf, Westphal, and Lieser Stefan. 2018. “Clean Code Developer.” 2018. http://clean-code-developer.de.

Richardson, Chris. 2018. “Microservices.Io The Scale Cube (XYZ-Split).” Microservices.Io. 2018. https://microservices.io/articles/scalecube.html.

Ries, Eric. 2014. *LEAN Startup*. 5. Auflage. Redline Verlag.

Rowe, Walker. 2019. “Nicht in Echtzeit: IoT-Protokolle Für Den Datenaustausch.” *Walker Rowe*, 2019. https://www.computerweekly.com/de/feature/Nicht-in-Echtzeit-IoT-Protokolle-fuer-den-Datenaustausch.

Rupp, Chris, Stefan Queins, and die SOPHISTen. 2012. *UML 2 Glasklar*. München: Carl Hanser Verlag GmbH & Co. KG. https://doi.org/10.3139/9783446431973.

Rupp, Chris, and Die SOPHISTen. 2014. *Requirements-Engineering Und -Management: Aus Der Praxis von Klassisch Bis Agil*. 6., aktual. München: Hanser. http://digitale-objekte.hbz-nrw.de/storage2/2014/11/09/file_37/5855311.pdf.

Samsung. 2019a. “MR Headset Samsung HMD Odyssey.” Samsung.Com. 2019. https://www.samsung.com/us/computing/hmd/windows-mixed-reality/xe800zaa-hc1us-xe800zaa-hc1us/.

———. 2019b. “VR Samsung Gear VR.” Samsung.Com. 2019. https://www.samsung.com/de/wearables/gear-vr-r323/.

Sand, Ron, and Jörg Reiff-Stephan. 2018. “Anwendung von Universell Einsetzbaren M2M-Protokollen Zur Ganzheitlichen Prozesskommunikation.” In .

Sand, Ron Van de, Sebastian Schulz, Kay Ritzmann, and Jörg Reiff-Stephan. 2019. “Vernetzung von Physischen Und Virtuellen Entitäten Im CPPS.” *Atp Magazin* 60 (09): 36. https://doi.org/10.17560/atp.v60i09.2351.

Schwichtenberg, Holger (Heise). 2019. “Build-2019-Microsoft-Fuehrt-Mono-Und-NET-Core-Zusammen-Zu-NET-5.” Heise.De. 2019. https://www.heise.de/developer/meldung/Build-2019-Microsoft-fuehrt-Mono-und-NET-Core-zusammen-zu-NET-5-0-4414166.html.

Scrum.org. 2019. “Scrum.Org WHAT IS SCRUM?” Scrum.Org. 2019. https://www.scrum.org/resources/what-is-scrum.

Shneiderman, Ben. 1998. “Designing the User Interface.” *Wesley*.

Smith, Steve. 2019a. *Architecting Modern Web Applications with ASP.NET Core and Microsoft Azure*. 2.2. Redmond: NET and Visual Studio product teams. https://aka.ms/webappebook PUBLISHED.

———. 2019b. “Übersicht Über ASP.NET Core MVC.” Docs.Microsoft.Com. 2019. https://docs.microsoft.com/de-de/aspnet/core/mvc/overview?view=aspnetcore-2.2.

Stansberry, James. 2015. “Zwei Protokolle Zur Auswahl.” *Elektroniknet.De*, 2015. https://www.elektroniknet.de/elektronik/kommunikation/zwei-protokolle-zur-auswahl-126289.html.

Starke, Gernot. 2011. “Effektive Software-Architekturen.” In *Effektive Software-Architekturen*, I–XII. München: Carl Hanser Verlag GmbH & Co. KG. https://doi.org/10.3139/9783446428515.fm.

———. 2014. *Effektive Software-Architekturen*. 6th ed. Hanser.

stevenlovegrove. 2014. “MQTT-Lib MqttDotNet C#.” Github. GITHUB. 2014. https://github.com/stevenlovegrove/MqttDotNet.

Tanenbaum, Andrew S. 2009. *Moderne Betriebssysteme*. 3rd ed. Pearson Studium.

Tanenbaum, Andrew S, and Maarten ˜vanœ Steen. 2008. *Verteilte Systeme: Prinzipien Und Paradigmen; Distributed Systems Dt.* 2., aktual. IT - Informatik. München u.a.: Pearson Studium. http://digitale-objekte.hbz-nrw.de/storage/2008/06/04/file_101/2426486.pdf.

Tiobe. 2019. “Tiobe Index.” 2019. https://www.tiobe.com/tiobe-index/.

Tomorrow Today, Labs. 2016. “Newton VR Unity AssetStore.” Assetstore.Unity.Com. 2016. https://assetstore.unity.com/packages/tools/newtonvr-75712.

Unity, Technologies. 2018a. “Unity3D 2018.1 DotNet Support.” Docs.Unity3d.Com. 2018. https://docs.unity3d.com/2018.1/Documentation/Manual/dotnetProfileSupport.html.

———. 2018b. “Unity3D 2018.3 CSharp Compiler.” Docs.Unity3d.Com. 2018. https://docs.unity3d.com/2018.3/Documentation/Manual/CSharpCompiler.html.

———. 2018c. “Unity3D PlayerSettings .Net.” Docs.Unity3d.Com. 2018. https://docs.unity3d.com/2017.4/Documentation/Manual/class-PlayerSettingsStandalone.html.

———. 2018d. “Unity3D QA LTS.” Unity3d. 2018. https://unity3d.com/unity/qa/lts-releases.

———. 2018e. “Unity3D ScriptReference Coroutine.” Docs.Unity3d.Com. 2018. https://docs.unity3d.com/ScriptReference/Coroutine.html.

———. 2018f. “Unity3D Understanding Coroutines.” Docs.Unity3d.Com. 2018. https://docs.unity3d.com/Manual/BestPracticeUnderstandingPerformanceInUnity3.html.

———. 2019a. “New in Unity 2017.4.” Docs.Unity3d.Com. 2019. https://docs.unity3d.com/2017.4/Documentation/Manual/WhatsNew2017.html.

———. 2019b. “New in Unity 2019.1.” Unity3d. 2019. https://unity3d.com/unity/whats-new/2019.1.0.

———. 2019c. “Unity3d.Com Manual Execution Order.” Docs.Unity3d.Com. 2019. https://docs.unity3d.com/Manual/ExecutionOrder.html.

———. 2019d. “Unity3D 2019.1 C# Comilper.” Docs.Unity3d.Com. 2019. https://docs.unity3d.com/2019.1/Documentation/Manual/CSharpCompiler.html.

———. 2019e. “Unity3D Manual Coroutines.” Docs.Unity3d.Com. 2019. https://docs.unity3d.com/Manual/Coroutines.html.

———. 2019f. “Unity3D Manual UnityEvent.” Docs.Unity3d.Com. 2019. https://docs.unity3d.com/Manual/UnityEvents.html.

———. 2019g. “Unity3D Roadmap Archive.” Unity3d. 2019. https://unity3d.com/unity/roadmap/archive.

———. 2019h. “Unity3D ScriptReference UnityEvent.” Docs.Unity3d.Com. 2019. https://docs.unity3d.com/ScriptReference/Events.UnityEvent.html.

“Unity3D WebPage Main.” 2019. 2019. https://unity3d.com/de.

“UnrealEngine WebPage Main.” 2019. 2019. https://www.unrealengine.com/en-US/what-is-unreal-engine-4.

Viganò, Giovanni Paolo Viganò. 2018. “Gpvigano/M2MqttUnity on GITHUB.” Github. 2018. https://github.com/gpvigano/M2MqttUnity.

vive.com. 2019. “Vive Tracker.” Www.Vive.Com. 2019. https://www.vive.com/de/vive-tracker/.

Vive. 2019. “VR HTC Vive.” 2019. https://www.vive.com/de/product/.

Vovacooper. 2014. “Vovacooper Unity3d_MQTT on GITHUB.” Github. 2014. https://github.com/vovacooper/Unity3d_MQTT.

“VR KIT Assetstore Unity3D.” 2019. Assetstore.Unity.Com. 2019. https://assetstore.unity.com/search/?k=VR+KIT&order_by=rating&q=VR&q=KIT&rows=42.

“Web API Kit: MQTT for IoT.” 2019. Haptix Games. 2019. https://assetstore.unity.com/packages/tools/network/web-api-kit-mqtt-for-iot-70367.

Wenzel, Maira, Luke Latham, Yishengjin1413, Aymeric A, Next Turn, Tom Dykstra, Bishnu Rawal, et al. 2017. .“.NET Architectural Components | Microsoft Docs.” Docs.Microsoft.Com. 2017. https://docs.microsoft.com/en-us/dotnet/standard/components.

West, Loren, and Marvin Roger. 2018. “Homie-IoT on GitHub.” Github. 2018. https://homieiot.github.io/.

Wiener, Norbert. 1961. *Cybernetics, or Control and Communication in the Animal and the Machine (2nd Ed.).* Cambridge: MIT Press. https://doi.org/10.1037/13140-000.

wikipedia. 2019a. “Dependency Injection Wikipedia.” Wikipedia.Org. 2019. https://en.wikipedia.org/wiki/Dependency_injection.

———. 2019b. “Wikipedia .NET Framework.” Wikipedia.De. 2019. https://de.wikipedia.org/wiki/.NET_Framework.

xamarin. 2014. “MQTT-Lib Xamarin/Mqtt C#.” Github. GITHUB. 2014. https://github.com/xamarin/mqtt.

Xljiulang. 2017. “MQTT-LIb Paho.MqttDotnet C#.” Github. GITHUB. 2017. https://github.com/xljiulang/Paho.MqttDotnet/.