hochschule mannheim

Fakultät Informationstechnik
Studiengang Technische Informatik

Bachelorarbeit

Smartphones als Sensor- und Aktor

Marius Cerwenetz

Abschlussarbeit

zur Erlangung des akademischen Grades

Bachelor of Science

Vorgelegt von	Marius Cerwenetz
am	08. Juli 2022
Referent	Prof. Dr. Peter Barth
Korreferent	Prof. Dr. Jens-Matthias Bohli

Schriftliche Versicherung laut Studien- und Prüfungsordnung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Mannheim, 08. Juli 2022

Marius Cerwenetz

Zusammenfassung

Um Programmieraufgaben interaktiv zu gestalten, eignen sich Projekte mit Microcontrollern besonders gut. Smartphones bieten einen vergleichbaren Funktionsumfang und müssen meist nicht zusätzlich beschafft werden. In dieser Arbeit wurde eine Softwarelösung erstellt, um Smartphonesensoren über eine Programmierumgebung auszulesen und Ausgaben auf dem Smartphone auszuführen. Hierfür wurde eine Android-Anwendung, eine Kontrollanwendung und eine programiersprachenunabhängige Softwarebibliothek erstellt.

Für die Nutzung der Lösung werden Beispiel-Programmieraufgaben dazugereicht. Programmierer schreiben Programme auf dem PC, welche auf Änderungen von Smartphonesensorwerten wie beispielsweise Beschleunigungssensoren reagieren und die Ausgabemöglichkeiten des Smartphones nutzen.

Inhaltsverzeichnis

Ist das absichtlich blau?

Kapitel 1

Einführung

Viele Programmierer oder Programmierwillige Anfänger mühen sich beim Programmierenlernen mit der Semantik von Programmiersprachen und grundlegenden algorithmischen Konzepten. Akademische Übungsaufgaben senken die Lernmotivation durch rein virtuelle Aufgabenstellungen ohne Interaktionsmöglichkeiten. Projekte mit Microcontrollern dagegen bieten eine praktische, fordernde und spielerische Einstiegsmöglichkeit. Es werden kleine Projekte realisiert, die durch die Verwendung von Sensoren Programmierer einladen, sich an Programmieraufgaben auszuprobieren. Diese Eigenschaften sind sinnvoll, insbesondere bei Projekten für Programmierer mit wenig Vorwissen wie Schüler oder Erstemester-Studierende. Gelerntes kann direkt angewandt werden. Praktische Programmieraufgaben bieten für Programmieranfänger den besten Lerneffekt bei höchster Motivation [?]. Die in Microcontroller integrierten Sensoren sind Voraussetzung, um physikalische Eigenschaften in der realen Welt zu messen. Programme auf dem Microcontroller können die ~~die~~ Sensoren auslesen und auf Änderungen der gemessenen Werte reagieren. Sensoren, Microcontroller und das entwickelte Programm ermöglichen zusammen eine Bedienung durch Nutzer. Selten verhält sich das Programm beim ersten Versuch korrekt. Eine Anpassung des Codes ist nötig, bis das Fehlverhalten beseitigt ist. Diese kontinierliche Weiterentwicklung mindert Ängste vor Änderungen des Codes, schafft Routine in der Entwicklung und damit ein tieferes Verständnis und Hintergrundwissen für die Problemstellung.

Microcontroller-Projekte benötigen allerdings kostspielige Einstiegs-Kits. Ein Arduino-Development-Board kostet im Arduino-Shop über 80,00 € [?]. Ein Großteil der Kosten entfällt zwar auf den eigentlichen Microcontroller, ein nicht unmittelbarer Teil jedoch auf Peripherie wie wie Breadboards, Verbindungskabel und Erweiterungsboards. Die Peripherie-Anbindung setzt daneben Hintergrundwissen in elektrotechnischen Bereichen voraus, wie zum Beispiel den Verschaltungskonventionen bei Breadboards. Dies stellt ebenfalls eine Einstiegshürde dar, die die eigentliche interaktive Lernerfahrung

herauszögert und die Motivation senkt.

Smartphones dienen als Alternative für Microcontroller-Projekte für Einstiegs-Programmierer. Integriert sind zahlreiche Sensoren wie Lagesensoren, Gyroskop oder Näherungssensoren. Der Sensoren-Umfang ist vergleichbar mit dem von Microcontrollern. Elektrische Bauteile konventioneller Microcontroller-Sets erlauben einen Fehlgebrauch, der im schlimmsten Fall in der Zerstörung von Komponenten enden kann. Projekte mit Smartphones reduzieren dieses Risiko dadurch, dass Schaltkreise bereits intern verknüpft und somit von äußerlicher Fehlverwendung geschützt sind. Ein weiterer Vorteil Smartphones gegebüber Microcontrollern liegt in der Verfügbarkeit. Weltweit besaßen 2022 5,2 Mrd. Menschen ein Smartphone [?]. Sie sind gerade unter Kindern, Jugendlichen und jungen Erwachsenen weit verbreitet. Kinder besitzen häufig bereits mit 10 Jahren ein Smartphone [?]. Im Alltag wird es für Chats, Social-Media oder Spiele verwendet. Sie sind also häufig bereits in Gebrauch und müssten für die Nutzung von Programmierprojekten nicht zusätzlich beschafft werden. Durch Ihre eingesetzten Sensoren können Sie zuverlässig physikalische Umgebungseigenschaften messen. Neben kabelgebundenen Übertragungsschnittstellen wie USB besteht auch die Möglichkeit, sich mit drahtlosen Verbindungsmöglichkeiten wie WLAN zu verbinden. Die Geräte sind zudem batteriebetrieben, was Lösungen ermöglicht, die von einer Spannungsversorgung unabhängig sind. Eine Einbindung von Smartphones ist in den meisten Entwicklungsumgebungen jedoch nicht möglich. Visuelle und haptische Ausgaben auf dem Smartphone erfordern zudem eine mobile Anwendung, da Smartphone-Betriebssysteme keine nativen Ausgabemethoden außerhalb von Apps bieten.

Ziel der Arbeit ist es, eine Smartphone-Anwendung für visuelle Ausgaben sowie der Messung und Übertragung von Sensorwerten zu entwickeln. Außerdem werden Schnittstellen in einer Programmierumgebung geschaffen, die die Interaktionsmöglichkeiten von Smartphones nutzbar macht. Hierfür stellen Softwarebibliotheken Funktionsaufrufe zur Verfügung, um Ausgaben auf dem Smartphone zu tätigen oder Sensorwerte auszulesen. Das Smartphone reagiert auf die empfangen Anfragen und führt die entsprechenden Kommandos aus. Zusammengesetzt besteht die Lösung aus einer programmiersprachenunabhängigen Programmierumgebung, einer Kontroll-Anwendung und einer mobilen Anwendung für Android Smartphones. Für die Verwendung werden angehenden Programmiereren Beispielaufgaben gereicht. Um die Beispielaufgaben zu bewältigen, muss die Lösung auch Benutzungsmöglichkeiten bereitstellen. Hierdurch werden Anforderungen an die in dieser Arbeit implementierte Lösung gestellt. Die Aufgaben, Anforderungen und Rahmenbedingungen sind in Kapitel 2 zu finden. Die drei Komponenten Smartphone-App, Kontrollanwendung und Programmierumgebung und ihr Zusammenspiel werden in Kapitel 3 vorgestellt. Die dafür benötigten Nachrichtenformate werden in Kapitel 4 gezeigt. Ihr Zweck

wird erklärt und und der Nachrichtenaustausch dort exemplarisch veranschaulicht. Kapitel 5.1 behandelt die Funktionsweise und den Aufbau der Android-App im Detail. Diese tauscht Nachrichten mit der Programmierumgebung aus. Als Zwischenvermittlung fungiert das zentrale Kontrollprogramm, was in Kapitel 5.2 erklärt wird. Einbindung, Nutzung, und externe Schnittstellen der Bibliotheken zur Android-App und dem Kontrollprogramm werden in Kapitel 5.3 erklärt. In Kapitel wird 7 untersucht, ob die vorgegebenen Anforderungen erfüllt wurden. Dort wird zudem die Verwendung der Lösung anhand einer Beispielaufgabe vorgestellt. Schwierigkeiten, die bei der Entwicklung auftraten, werden in Kapitel 8 diskutiert. Erweiterungsmöglichkeiten und Verbesserungen werden diskutiert.

diskutiert
diskutiert

Was ist mit Kapitel 6?

Kapitel 2

Smartphones als Microcontroller-Ersatz

Smartphones sind in sich geschlossene technische Geräte, die neben vordefinierten Verbindungsschnittstellen wie einem USB-Port, WLAN und Bluetooth keine weiteren Schnittstellen bieten, um externe Hardware und Schaltungen anzuschließen und fernzusteuern. Microcontroller-Schaltungen zum Programmieren lernen bieten meistens mehrere Ausgabemöglichkeiten wie LEDs, Lautsprecher oder Piepser. Smartphones können diese Bausteine nicht anschließen, aber virtuell darstellen. Gewohnte Ausgabeelemente können virtualisiert werden. Die Funktionen sind außerdem nicht nur von Mehrzweck-Ausgaben, wie LED-Grids begrenzt, die zum Beispiel für die Text- oder Bildanzeige verwendet werden können. Zweckgebundene Elemente wie Textfelder, Textausgaben oder Bildausgaben in der App können beliebig kombiniert werden. Darüber hinaus ist die Anordnung der jeweiligen Elemente frei wählbar, so dass das Layout anders als bei Microcontrollern auch im Nachhinein noch geändert werden kann. Die Ausgabemöglichkeiten können in Kombination mit Sensor-Daten verwendet werden, um kleine Programmieraufgaben zu lösen. Einige Beispiele werden in diesem Kapitel vorgestellt. Da diese nicht auf einem Microcontroller, sondern einem Smartphone ausgeführt werden, müssen gewisse Rahmenbedingungen erfüllt sein, wie beispielsweise geringe Latenzen in der Sensordatenübertragung.

2.1 Beispielprogrammieraufgaben

Praxisnahe Programmieraufgaben mit interessanten Aufgabenstellungen motivieren Softwareentwickler. Die in diesem Abschnitt vorgestellten Beispielaufgaben definieren das von Sensormesswerten abhängige Verhalten interaktiver Programme. Die Aufgabenstellungen sind in Tabelle 2.1 aufgeführt. Für jede Aufgabe werden die

benötigen Sensortypen und Ausgabeschnittstellen beschrieben. Leserinnen und Leser werden in der Entscheidungsfindung durch die Angabe eines dreistufigen Schwierigkeitsgrades unterstützt. Dies soll verhindern, dass sich unerfahrene Programmierer mit komplizierten Aufgaben zu Anfang überfordern.

Name der Aufgabe	Benötigte Sensoren	Verwendete Ausgaben	Schwierigkeitsgrad
Disco	-	Led	+
Würfeln	Lagesensor	Textfeld	+
Diebstahl-Alarm	Näherungssensor	Textfeld, Led, Vibration	++
Klatsch-Zähler	Mikrofon	Textfeld	++
Dreh-Zähler	Lagesensor	Textfeld	+++

Tabelle 2.1: Beispielprogrammieraufgaben

In der Aufgabe *Disco* soll eine virtuelle LED für eine Zeitdauer von 500 ms grün und anschließen 500ms rot leuchten. Sensoren sind in dieser Aufgabe überflüssig, da die LED-Ausgabe unabhängig von Sensormesswertänderungen ausgeführt wird. Aus diesem Grund ist die Aufgabe als Einfach eingestuft.

In der Aufgabe *Würfeln* soll ein Schütteln des Geräts erkannt werden. Zur Verwendung kommt dabei ein Lagesensor zum Einsatz, welcher Gerätebeschleunigungen messen kann. Wird ein Schütteln erkannt, soll auf dem PC eine Zufallszahl generiert werden. Anschließend wird diese auf dem Gerät in einem Textfeld ausgegeben. Der Schwierigkeitsgrad wird ebenfalls auf Einfach eingeschätzt, da die Aufgabe unter Verwendung eines Sensors und einer Ausgabe lösbar ist.

In der Aufgabe *Diebstahl-Alarm* wird unter Verwendung des Näherungssensors das örtliche Umfeld des Geräts auf eine Annäherung kontrolliert. Wird eine Annäherung festgestellt, wird ein alarmierender Text im Textfeld ausgegeben. Zusätzlich soll die LED die wie in Aufgabe *Disco* die Farbe wechseln. Neben den visuellen ausgaben wird die Vibrationsfunktion als haptisches Feedback benötigt. Im Falle einer Annäherung, soll das Gerät fünf mal vibrieren. Der Schwierigkeitsgrad der Aufgabe ist als Mittel eingestuft, da hier zwischen Alarm- und Normalzustand unterschieden werden muss. Entfernt sich eine Person, muss der Alarm-Zustand verlassen werden können. Alle Ausgaben werden auf ihren initialwert zurückgesetzt.

Bei der Aufgabenstellung *Klatsch-Zähler* muss für einen definierten Zeitraum die Anzahl der Händeklatscher gemessen werden. Diese Anzahl wird anschließend im Textfeld des Geräts ausgegeben. Zur Verwendung kommen hierfür das Mikrophon als Ein- und das Textfeld als Ausgabe. Die Schwierigkeit ist auf *Mittel* angesetzt, da die Messwerte von Händeklatschern unterschiedliche Intensitäten aufweisen. Um einen

Händeklatscher zu identifizieren, ist es nötig, Grenzwerte zu ermitteln, um Sie von normalen Hintergrundgeräuschen abzugrenzen. Eine Visualisierung der Messdaten kann bei dieser Festlegung helfen.

Bei der letzen Aufgabe *Drehzähler* soll der Programmierer das Device innerhalb eines definierten Zeitraums drehen und anhand der Messdaten die Anzahl der Umdrehungen ermitteln. Diese Anzahl soll anschließend im Textfeld auf dem Gerät ausgegeben werden. Zur Verwendung kommen hier ebenfalls der Lagesensor als Sensoreingabe und das Textfeld als Ausgabe. Die Aufgabe ist als Schwer bewertet, da hier Wiederholungen in einer Werteabfolge erkannt werden müssen, welche jedoch wie bei der Aufgabe *Klatsch-Zähler* in ihrer Größe variieren können. Abhängigkeiten zwischen den Messwertergebnissen erschweren eine Umdrehungserkennung zusätzlich.

2.2 Anforderungen der Implementierung

Aus den Beispielaufgaben leiten sich Anforderungen an die implementierte Lösung ab. Geringe Latenzen sind bei der Sensordatenübermittlung für ein responsives Verhalten von Programmen nötig. Sie tragen zur Lernerfahrung bei, da sich physikalische Änderungen unmittelbar auf das Verhalten des entwickelten Progamms auswirken. Eine Verkürzung der Latenzen ist von mehreren Faktoren abhängig.

Für die Übermittlung von Sensorwerten müssen diese initial vorliegen. Sensormessprozesse müssen gestartet und Sensoren ausgelesen werden, was die Latenzzeiten erhöht. Sensormessungen sollten daher nicht erst nach einer Anfrage oder einem Ereignis, sondern direkt zu Anfang gestartet werden und kontinuierlich messen. Der Versand der Sensorwerte wird durch den Transportweg zwischen Smartphone und Programmierumgebung und deren lokale Gegebenheiten verzögert. Smartphones bieten keine kabelgebundenen Netzwerkschnittstellen. Latenzen können je nach Mobilfunk- bzw WLAN Standard, Umwelteinflüssen und der Geräteanzahl in Funkzellen variieren. Für Latenzen dient die Round-Trip-Time (RTT) als Messgröße. Sie beschreibt die Zeitdauer der Übermittlung einer Nachricht über Hin- und Rückweg eines Hosts zu einem Anderen. Präventionsprinzipen wie CSMA/CA verhindern bei WLAN-Verbindungen Interferenzen, erhöhen allerdings die Latenzen. Während sie bei Standards wie WLAN 802.11b ca. 10 ms beträgt, kann Sie bei UMTS auf 300 ms bis 400 ms ansteigen [**?**] . Zudem ist Sie variabel, was bei einer synchronen Übertragung zu ungewollten Verzögerungen führt. Um die Latenz des Transportweges geringer darzustellen, müssen Sensorwerte zwischengespeichert werden, um einen Puffer aufzubauen, auf den in Fällen erhöhter Latenz zugegriffen werden kann. Eine weitere Latenz-Optimierung ist durch den Einsatz effizienter Protokolle möglich. Der Verlust weniger Sensorwerte ist für die Funktionsweise des Programms unerheblich.

Verbindungsorientierte Protokolle erhöhen die Menge der zu sendenen Nachrichten pro Sensorwert. Verbindungslose Protokolle sind deshalb zu bevorzugen.

Die implementierte Lösung muss einen benutzerfreundlichen Nutzungszugang für Programmierer und die Anbindung ihrer Programme bieten. Boilerplate-Code soll reduziert werden, um die Verwendung zu erleichtern. Die angebotene Funktionalität soll mit möglichst wenig Vorwissen nutzbar sein. Auf serialisierte Ausgabeformate muss verzichtet werden. Funktionen müssen Rückgabewerte in Form von primitiven Datentypen zurückgeben. Für die Bedienung aller Ausgabemöglichkeiten des Smartphones wird muss die Lösung eine Funktion zur programmierung bereitstellen. Das Einbinden in bestehende Entwicklungsumgebungen ist neben der benutzerfreundlichen Schnittstellengestaltung Voraussetzung für die Nutzung der Lösung. Die Entwicklungsumgebung Eclipse ist im Labor-Kontext weit verbreitet. Da es sich um eine plattformunabhängige IDE handelt, muss auch die Lösung plattformübergreifend in diese Entwicklungsumgebung integrierbar sein. Nachrichtenabläufe sollen nachvollziehbar sein um fehlerhafte Konfigurationen rasch zu identifizieren und zu korrigieren. Hierfür muss es eine Logging-Funktion geben, die den Nachrichtenverlauf aufzeichnet und ausgibt. Programmierer können anhand der Logeinträge den Versand nachvollziehen, wodurch Fehler ersichtlich werden.

Neben den Netzwerktechnischen- und Nutzungsbezogenen Anforderungen muss die Lösung auch Ausgabemöglichkeiten über die grafische Benutzeroberfläche einer Smartphone-App bereitstellen. Teil dieser Oberfläche ist eine farbwechselbare Signal-LED. Zur Ausgabe von Texten und Zeichen muss ein Textfeld implementiert werden. Durch zwei Tasten muss der Nutzer außerdem mit der Anwendung interargieren können. Neben den optischen Ausgaben bzw. Bedienelementen muss die Smartphone-App auch haptische Ausgaben wie Vibrationen umsetzen können. Für die Einstellung von Vibrationsmustern muss eine Vibrationszeitdauer konfigurierbar sein.

Kapitel 3

Architektur

Insgesamt besteht die Lösung aus einer Programmierumgebung, einem Kontrollprogramm zur Nachrichtenkoordinierung und einer mobilen Android-Anwendung für die Sensordatenerhebung und für die Anzeige der Ausgaben. Der Aufbau ist in Abbildung 3.1 dargestellt. Die Programmierumgebung besteht aus einer pro-

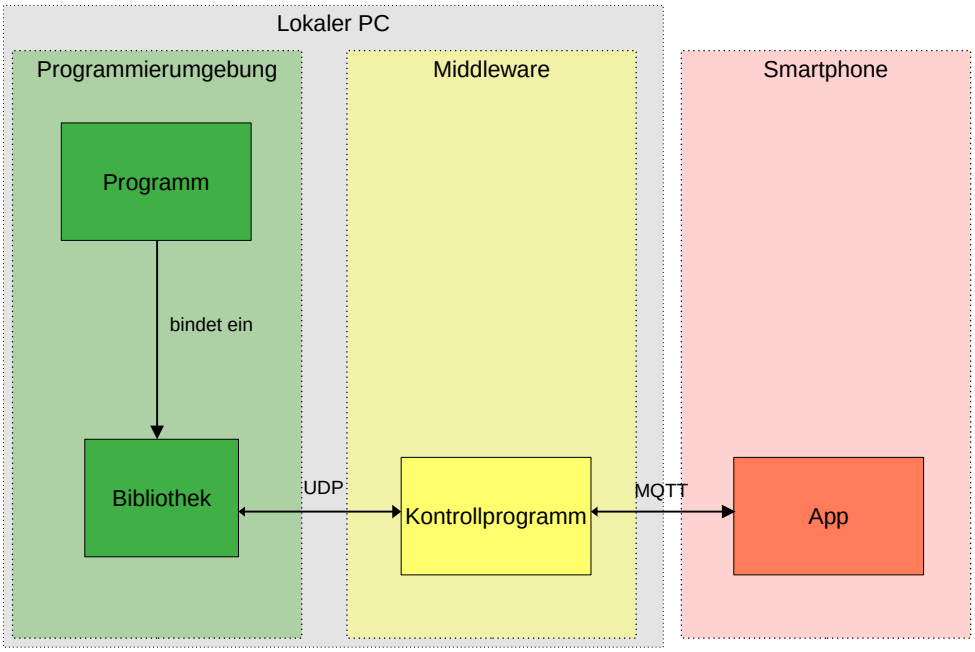

Abbildung 3.1: System-Aufbau

grammiersprachenunabhänigen Bibliothek und dem vom Entwickler geschriebenen Programm. Der Entwickler kann die Funktionalität der Lösung nach Einbindung

der Bibliothek nutzen. Neben der Programmierumgebung läuft auf dem lokalen PC auch das Kontrollprogramm. Dieses ist für die Koordinierung des Nachrichtenaustauschs zwischen der Android-App und der Programmierumgebung zuständig. Startet ein Entwickler einen Aufruf der Bibliothek, kommuniziert diese ihn per UDP dem Kontrollprogramm. Das Kontrollprogramm kommunziert dann weiter über MQTT mit dem Smartphone. MQTT ist ein auf TCP basierendes Client-Server-Protokoll. Durch einen 2 Byte großen Header und maximalen Payload-Größe von 260 MB ist es leichtgewichtig und gleichzeitig flexibel. Nachrichten werden an einen MQTT Broker gesendet, der die Nachrichten dann an alle Clients weiterreicht, die das Topic, auf dem die Nachricht gesendet wurde, abboniert haben. Die UDP-Kommunikation zwischen Bibliothek und Kontrollprogramm verläuft auf dem lokalen PC über ein Loopback-Interface. Das Kontrollprogramm erfüllt drei Aufgaben: Die Beantwortung von Sensoranfragen der Bibliothek, dem Zwischenspeichern von Sensorwerten in einem Pufferspeicher und dem Weiterleiten von Ausgabe-Kommandos auf das Smartphone. Bei Sensoranfragen werden von der Bibliothek aus an die Kontrollanwendung gesendet. Dieses beantwortet Sie mit dem aktuell vorliegenden Sensorwert. Die Sensorwerte werden fortlaufend durch das Smartphone aktualisiert. Es sendet fortwährend aktuelle Sensordaten an das Kontrollprogramm. Dieses nimmt Sie entgegen und sichert Sie in einem internen Puffer. Ausgabe-Anfragen für das Smartphone werden ebenfalls über das Kontrollprogramm an das Smartphone per MQTT gesendet. Die Smartphone-Anwendung beginnt, sobald Sie startet mit der Erhebung der Sensordaten, welche sie anschließend an die Kontrollanwendung sendet. Daneben reagiert Sie auf Ausgabeanfragen von der Bibliothek, welche bei Eingang ausführt werden.

Kapitel 4

Nachrichtenformate

Ein einheitliches Kommunikationsformat ist für den Nachrichtenaustausch unabdingbar. Der zur Übermittlung ausgearbeitete Standard definiert die Nachrichten in einem Klartextformat. Als Darstellungsform wird JSON verwendet. Die in einer Datei gespeicherten Nachrichten-Vorlagen werden in der Bibliothek, dem Kontrollprogramm und in der Smartphone-App eingelesen.So sind die Nachrichten in jeder Komponente gleich. Es gibt unterschiedliche Nachrichtentypen für verschiedene Zwecke. Diese sind in Tabelle 4.1 aufgeführt. Um besser nachvollziehen zu können, welche Kommunikationspartner welche Nachrichten austauschen, sind außerdem Quelle und Ziel und das verwendete Netzwerkprotokoll angegeben.

Nachrichtentyp	**Quelle**	**Ziel**	**Netzwerkprotokoll**
sensor_request	Bibliothek	Kontrollprogramm	UDP
sensor_response	Kontrollprogramm	Bibliothek	UDP
update_request	Smartphone	Kontrollprogramm	MQTT
rpc_request	Bibliothek, Kontrollprogramm	Smartphone	UDP/MQTT
rpc_response	Smartphone, Kontrollprogramm	Bibliothek	UDP/MQTT

Tabelle 4.1: Nachrichten-Typen

*Sensor_request*s kommen zum Einsatz, wenn Programmierer einen Sensorwert abfragen. Dann wird die Nachricht von der Bibliothek an das Kontrollprogramm gesendet. Dieses hat vom Smartphone übermittelte Sensorwerte zwischengespeichert. Nach Eingang ermittelt das Kontrollprogramm den gewünschten Sensorwert. Die Angabe des gewünschten Sensors wird im Feld `sensor_type` beschrieben. Alle Sensortypen besitzen ein festgelegtes Kürzel. Dadurch können ihre Werte in der Datenbank in der Kontrollanwendung addressiert werden und Messwerte eingespeichert oder

zurückgegeben werden. Die Sensortyp Kürzel sind in Tabelle 4.2 zu finden. Ist er

TYPE-Kürzel	Beschreibung
accel_xyz	Lagesensor für die X, Y oder Z-Richtung
gyro_xyz	Gyroskopsensor für die X, Y oder Z-Richtung
prox	Näherungssensor

Tabelle 4.2: Sensor-Kürzel mit Beschreibung

in der gespeicherten Datenstruktur zu finden, wird er in einer neuen Nachricht im Nachrichtenformat *sensor_response* zurückgesendet. Der gespeicherte Wert ist im Feld `sensor_value` zu finden. Die Antwort wird von der Bibliothek angenommen und anschließend an das Programm des Entwicklers zurückgegeben, das den Sensorwert ursprünglich forderte. Sensor-Abfragen laufen blockierend ab. Das bedeutet, dass jede Sensor-Wert-Anfrage immer erst eine Rückgabe erhalten muss, bevor die nächste Sensor-Wert-Anfrage gestartet werden kann. Es ist unmöglich, dass die Kontrollanwendung zwei Sensor-Wert-Anfragen gleichzeitig erhält und die Antworten in umgekehrter Reihenfolge zurücksendet, was eine Vertauschung der Sensorwerte bedeutete. Durch Angabe des ursprünglich geforderten Sensor-Typs in der Antwort könnte dies verhindert werden, ist allerdings nicht nötig. Sensorwerte müssen kontinuierlich vom Smartphone aktualisiert werden, damit das Kontrollprogramm immer den aktuellsten Wert zurückliefern kann. Die Android-App sendet daher in periodischen Abständen Nachrichten des typs *update_request* an das Kontrollprogramm. Sie kann für jede Sensor-Art verwendet werden. Übermittelt wird sowohl der Sensor-Typ, als auch der gemessene Sensor-Wert. Die Kontrollanwendung kann den Wert über den Typ zuordnen und in den internen Datenpuffer eintragen. Der gesamte Ablauf für Sensoranfragen ist in Abbildung 4.1 dargestellt. Zu sehen sind die drei Komponenten Bibliothek, Kontrollprogramm und App auf dem Smartphone. Die Bibliothek sendet sensor_requests per UDP an das Kontrollprogramm. Dieses antwortet über UDP mit einer sensor_response. Währenddessen sendet die App auf dem Smartphone kontinuierlich update_requests per MQTT um der Kontrollanwendung neue Messergebnisse mitzuteilen.

Neben den Sensordaten betreffenden Nachrichten existieren auch Ausgabe-Kommandos, um Ausgaben in der Smartphone-App umzusetzen. Es wird unterschieden zwischen Ausgaben mit und ohne Rückgabewert. Für erstere gibt es den Nachrichtentyp *rpc_request*. RPC steht für Remote Procedure Call und bezeichnet Clientseitige Funktionsaufrufe, die auf einem Server ausgeführt werden. Die Bezeichnung entspricht nicht dem Konzept, da die Smartphone-App in diesem Fall die Rolle des Servers einnehmen würde. Die Voraussetzung einer Client-Server-Anwendung ist dadurch nicht gegeben. Kommunikationspartner tauschen auf gleicher Ebene Daten aus. Die Bezeichnung wurde unter dem Fokus auf der entfernten Ausführung einer Funktion

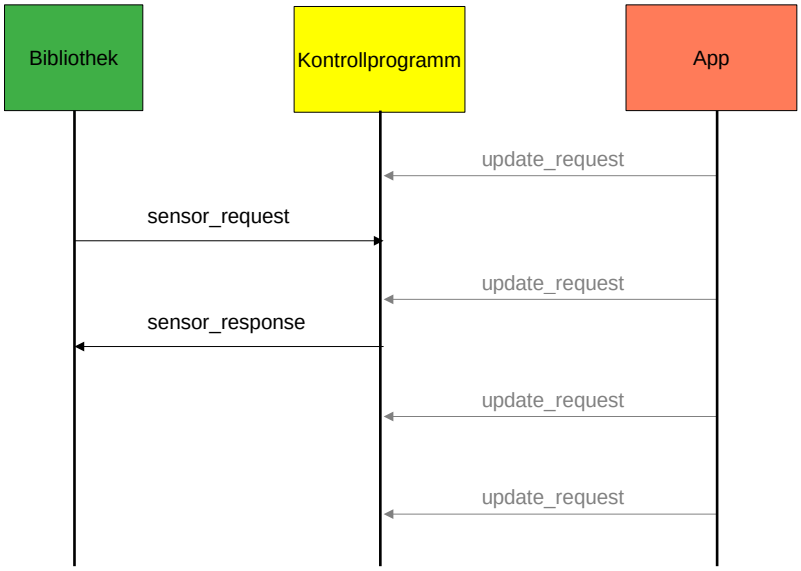

Abbildung 4.1: Nachrichtenablauf der Sensordatenübermittlung

gewählt. Der Nachrichtentyp enthält die Felder `command` und `value`. Ersters spezifiziert das Ausgabe-Kommando, das Zweite die Größe des Parameters für das Ausgabekommando. Es ist fast immer befüllt. Vereinzelt gibt es jedoch auch Kommandos die keinen Parameter benötigen. Dann bleibt dieses Feld leer. Die Nachricht wird von der Bibliothek per UDP an die Kontrollanwendung und von dort aus per MQTT an das Smartphone gesendet. Die Smartphone-App nimmt die Anfrage an und führt die Ausgabe aus. Manche Kommandos erheben zusätzlich einen Rückgabewert. Damit dieser vom Smartphone zurück an die Bibliothek gesendet werden kann, gibt es das Nachrichtenformat *rpc_response*. Dieses wird erst per MQTT an das Kontrollprogramm und von dort aus per UDP an die Bibliothek gesendet. Wie bei sensor_responses können sich die Antworten nicht gegenseitig überholen, was die Übertragung des zugrundeliegenden Ausgabe-Kommandos überflüssig macht. Nur der ermittelte Wert des Kommandos ist relevant und wird in der Nachricht übermittelt. Der Nachrichtenablauf wird in Abbildung 4.2 zusammengefasst. Zu sehen sind die drei Komponenten Bibliothek, Kontrollprogramm und Android-App. Die Bibliothek sendet rpc_requests per UDP an das Kontrollprogramm. Dieses leitet die Nachricht per MQTT direkt weiter an die App. Dort wird das gewünschte Kommando ausgeführt. Fällt ein Rückgabewert an, wird eine rpc_response generiert und vom Smartphone per MQTT zurück an die Kontrollanwendung gesendet. Diese leitet die Nachricht dann per UDP weiter an die Bibliothek.

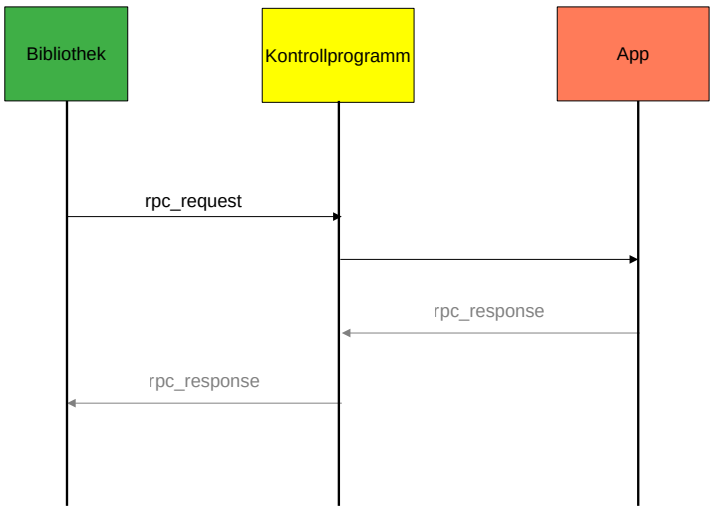

Abbildung 4.2: Nachrichtenablauf der RPC-Anfragen

Kapitel 5

Implementierung der Komponenten

In diesem Kapitel werden die Designentscheidungen in der Implementierung der drei Komponenten Android-Anwendung, Kontrollanwendung und Programmierumgebung erörtert. Der Fokus liegt auf der internen Struktur der jeweiligen Komponenten ihrer Verknüpfung. Startabläufe werden dargelegt und die Funktionsweisen der einzelnen Komponenten vermittelt.

5.1 Android-Anwendung

Die Android-Anwendung ist eine der drei zentralen Bestandteile des Frameworks. Sie dient dazu, Sensormessprozesse zu starten, Sensordaten zu übermitteln und Ausgabe-Kommandos auszuführen. Für diese bietet Sie unterschiedliche UI-Elemente in einer Activity an, der RootActivity. Diese sind: eine Signal-Led, ein Textfeld und zwei Buttons. Neben UI Elementen gibt es zusätzlich noch eine Vibrationsausgabe. Jedes Messergebnis wird anschließend per MQTT an die Kontrollanwendung gesendet. Versandt werden die Sensorwerte über einen Service, der im Hintergrund ausgeführt wird. Dieser reagiert ebenfalls auf eingehende Anfragen, um Ausgaben auf dem Smartphone auszulösen. Gibt die Ausgabe einen Rückgabewert, diese an die Kontrollanwendung weitergereicht.

Die Anwendung misst in periodischen Zeitabständen Sensorwerte und sendet Sie an die Kontrollanwendung. Messungen werden über SensorEventListener realisiert. Diese starten Messvorgänge und ermitteln in periodischen Zeitabständen den aktuell vorliegenden Messwert. Verwendete Sensoren sind beispielsweise der Lagesensor oder das Gyroskop.

Die Erhebung der Messwerte erfolgt zum Start der Anwendung. Ein Ablaufplan ist in Abbildung 5.1 zu sehen. In der Root-Activity werden zuerst alle UI-Elemente

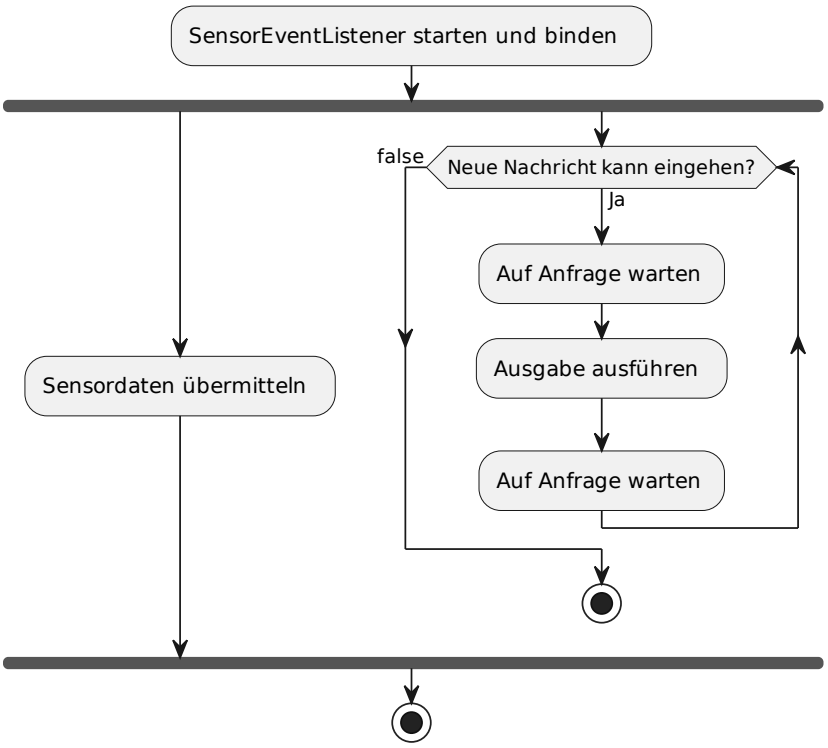

Abbildung 5.1: Ablaufdiagramm Android Anwendung

eingebunden, um Sie über Kommandos zu manipulieren. Anschließend werden Konfigurationsdaten eingelesen. Insgesamt gibt es zwei Konfigurationsdateien: config.json und protocol.json. In Ersterem ist zum Beispiel der Hostname des MQTT-Brokers, der Port oder das Topic definiert. Diese Daten sind für die Übermittlung der Nachrichten per MQTT wichtig. In protocol.json wird die Form der für das Smartphone relevanten Nachrichtenformate update_request und rpc_response definiert. Außerden sind dort die unterstützten Sensoren und Ausgabekommandos beschrieben. Relevant werden diese bei der Entscheidungsfindung bei Eingang eines rpc_requests, wenn determiniert werden muss, welche Aktion die Nachricht beeinhaltet.
Nach dem Einlesen der Konfigurationen wird der zur Kommunikation verwendete MQTT Service eingebunden. Dies geschieht asynchron. Über eine ServiceConnection wird beim erfolgreichen einbinden über eine Callback-Methode der weitere Verlauf definiert. Die Root-Activity speichert dann die Referenz auf den Service und der Service die Referenz auf die Root-Activity. Grund für dieses gegenseitige Einbinden ist, dass Nachrichten im MQTT Service in einem seperaten Thread behandelt werden. Bei RPC-Requests müssen jedoch UI-Elemente verändert werden können. Dies ist

ohne weiteres nicht aus dem Service heraus möglich. Mit einer Referenz auf die Activity kann der Service UI-Andernde Funtionen auf der Activity ausrufen. Android unterbindet jedoch UI-Manipulationen durch Threads, die nicht der UI-Thread sind. Dieses Problem wird durch die Methode *runOnUiThread* umgangen, welche die Änderung in der Ausführungswarteschlange des UIThreads einreiht. Der Service baut eine Verbindung zu einem MQTT Server auf. Beim Einbinden des MQTT Servers stellt dieser eine Verbindung zu einem in config.json definierten MQTT-Broker und Topic her. Ist der Service final eingebunden, können die Sensormessprozesse gestartet werden, da Messdaten nun zuverlässig gesendet werden können. Verschiedene SensorEventListener werden nun gestartet und zentral in einem SensorEventListener-Container gesammelt gespeichert und die Messprozesse jeweils angestoßen. Somit ist die Startroutine der Mobilen Anwendung abgeschlossen. Auf Nachrichten wird nun nur noch im MQTT-Service in einem MessageListener mit entsprechendem Callback reagiert.

Die Funktionsweise der Sensordatenübertragung wird in Abbildung 5.2 nocheinmal zusammenfassend dargestellt. Die Klasse *SmartBitEventListenerContainer* beeinhaltet

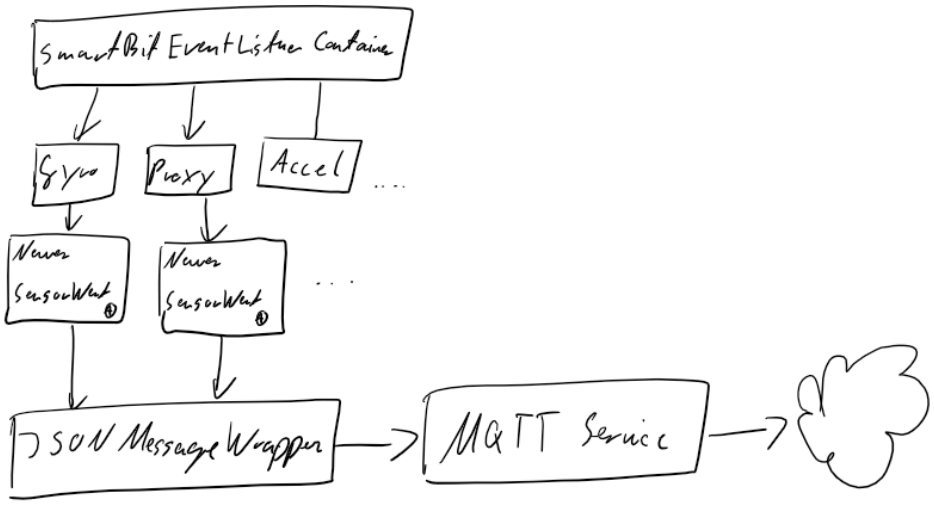

Abbildung 5.2: Ablaufdiagramm SensorEventListener

SensorEventListener für alle Arten von unterstützten Sensoren. Der Container dient lediglich der Datenhaltung. Aufgabe der SensorEventListener ist es, auf Sensorwert-Änderungen zu reagieren und eine entsprechende Callback-Funktion aufzurufen. In dieser werden dann über statische Methoden der Klasse *JSONMessageWrapper* update_requests generiert und der gemessene Wert eingesetzt. Die so generierte Nachricht wird anschließend über den gebundenen MQTT-Service an das vorher definierte Topic versendet.

Die Anwendung ist nun betriebsbereit und beginnt bereits erste Nachrichten an die Kontrollanwendung zu senden. Übermittelt werden die Sensordaten an den Broker mit einer QOS-Stufe von 0. Verluste von update_requests sind unproblematisch, da es je nach Taktung sehr schnell neue Sensorwerte gibt die übertragen werden können. Eine exakte Zustellung ist hier nicht notwendig und verlangsamt eher den Übertragungsprozess.

Smartphones beeinhalten Sensoren die Daten über die Umgebungseigenschaften erfassen. Dazu zählen Beispielsweise Bewegung, Näherung, aber auch Temperatur oder Luftdruck. In der Lösung besteht ihr Zweck darin auf Änderungen der Werte zu reagieren. Für unterschiedliche Aufgaben werden unterschiedliche Sensoren benötigt. Beispielsweise word für *Diebstahl-Alarm* nur der Näherungssensor verwendet, für *Dreh-Zähler* der Lagesensor. Insgesamt werden in der Android-Anwendung werden folgende Sensortypen verwendet: Lineare Beschleunigungssensoren, Gyroskop und Annäherungssensor.

Beschleunigungs- bzw. Lagesensoren messen die Beschleunigung in m/s^2 für die drei Bewegungsrichtungen: X-, Y- und Z-Achse in einem festgelegten Zeitraum. Die Erdbeschleunigung ist auch in diesen Messwerten enthalten. Diese muss für die bereinigten, realen Werte von den aufgenommenen Werten subtrahiert werden[?]. Messeinheiten unterscheiden sich je nach Sensor. Das Gyroskop misst keine Beschleunigung sondern die aktuelle Geschwindigkeit in rad/s der gleichen Achsen. Zur Übersicht sind diese in Abbildung 5.3 zu dargestellt. Die Frequenz mit der Messwerte erfasst werden kann manuell angegeben werden. Hierfür stehen vier Stufen zur Auswahl. In der

Bezeichnung	Verzögerung
SENSOR_DELAY_FASTEST	Keine. Verwendet die Frequenz des Sensors.
SENSOR_DELAY_GAME	20 ms
SENSOR_DELAY_UI	60 ms
SENSOR_DELAY_NORMAL	200 ms

Tabelle 5.1: Sensor-Taktgeschwindigkeiten[?]

Android-APP erfolgen alle Messungen mit SENSOR_DELAY_NORMAL. Die Stufe gilt jedoch nicht als festes Limit, sondern eher als Richt-Frequenz. Android kann die reale Frequenz auch erhöhen. Nicht alle Smartphones besitzen alle Sensoren. Daher wird beim Start überprüft ob der Sensor auch wirklich vorhanden ist. Ist er es nicht, wird auch keine Messung gestartet.

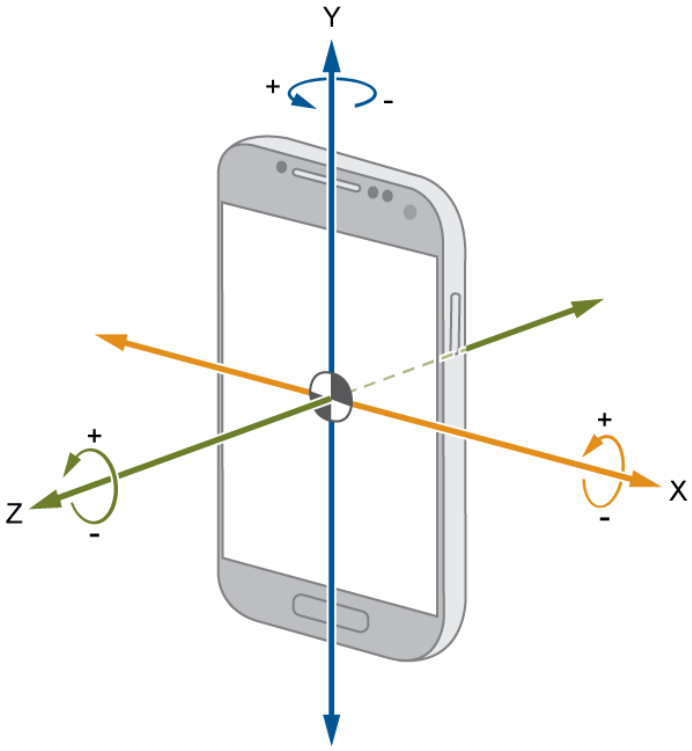

Abbildung 5.3: Android-Koordinatensystem

5.2 Kontrollanwendung

Für Vermittlung zwischen den Komponenten fungiert eine Server-Anwendung. Sie ist in Python geschrieben und vermittelt zwischen UDP-Anfragen auf der einen Seite vom Client aus und MQTT-Anfragen vom Smartphone auf der anderen Seite. User können über die Library RPC-Anfragen oder Sensor-Anfragen an den Server stellen. Dies geschieht in form von JSON-Anfragen, die per UDP übermittelt werden. Der Server ist unter der localhost-Adresse 127.0.0.1 auf dem Port 5006 erreichbar. Für die MQTT-Verbindung kommt dabei die unter OpenSource-Lizenz stehende MQTT-Library Paho der Eclipse-Foundation zum Einsatz. [?]

Die Serveranwendung mit dem Namen server.py ist aufgeteilt in einen Datenverwaltungsteil, DataHandler, und eine MQTT Anbindung, MQTTHandlerThread. Da die Sensorwerte des Smartphoens vorrätig gehalten werden, wird außerdem eine Datenklasse für diese, SensorDB, intern gehalten. Eine Übersicht über die Komponenten ist Abbildung 5.4 zu entnehmen. DataHandler wiederrum teilt sich nochmal auf in vier seperate Funktionen, die als Threads nebenläufig laufen: MqttRequestHandler, UDPRequestHandler, UDPRequestQueueWorker und AnswerQueueWorker.

Die Funktionsweise und Zwecke dieser Threads wird im Folgenden an zwei Beispie-

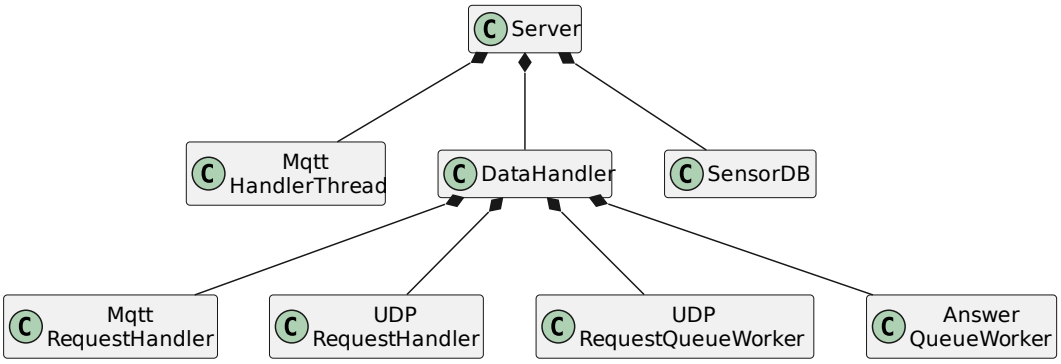

Abbildung 5.4: UML Digaramm Server

len erläutert. Für das erste Beispiel wird Abbildung 5.5 betrachtet. Zu sehen ist ein MQTT-Request, also eine Anfragen des Smartphones, dass über MQTT an den Server gesendet wird. Schnittstellen zu MQTT sind in der Abbildung grün, Schnittstellen zur Library per UDP, ~~sind~~ blau markiert. Erreicht ein MQTT Request den Server, wird

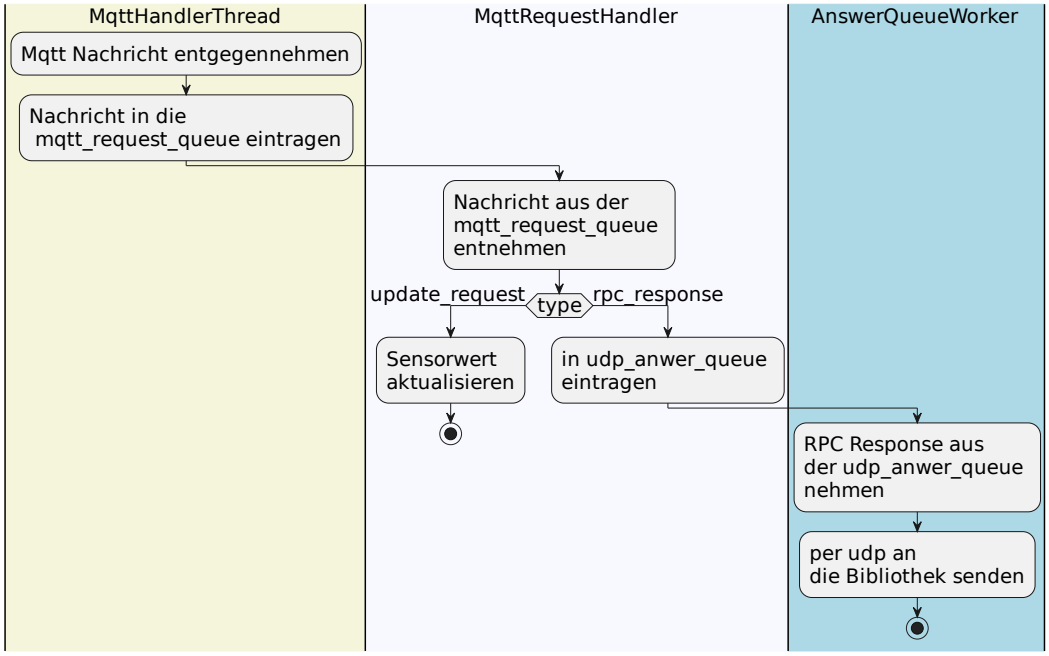

Abbildung 5.5: Ablaufdiagramm MQTT Request

es im MQTTHandlerThread entgegengenommen. Dieser setzt die Nachricht in eine MQTTRequestQueue ein. Der MQTTRequestHandler des DataHandlers wartet, bis ein Eintrag in der Queue vorhanden ist und nimmt gegebenenfalls eine Nachricht. Daraufhin wird der Typ des Requests bestimmt. Handelt es sich um ein Sensorupdate, muss nur der Sensorwert in der Datenbank aktualisiert werden. Handelt es sich um

eine rpc_response, also um eine Antwort auf eine vorausgegangenes rpc_request, dass einen Rückgabewert fordert, wird das request in eine udp_answer_queue eingefügt. Der AnswerQueue Worker wartet, ähnlich wie der MQTT Request Handler, bis eine neue Nachricht vorhanden ist, die per UDP an den Client gesendet werden soll und sendet diese dann gegebenfalls ab.

Das zweite Beispiel befasst sich mit dem Ablauf eines UDP-Requests, also einer Anfrage, die mithilfe der Library gesendet wurde. Der Ablauf ist in Abbildung 5.6 dargestellt. Wie auch schon in der letzten Abbildung sind alle Schnittstellen zum Smartphone grün und alle Schnittstellen zur Library blau markiert. Erreicht ein UDP

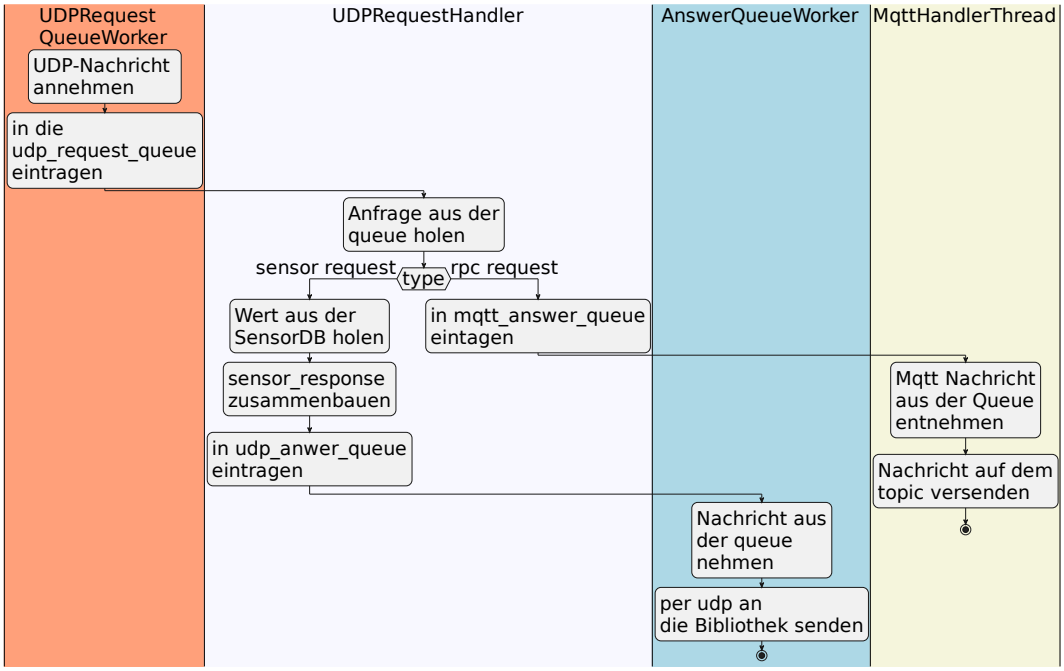

Abbildung 5.6: Ablaufdiagramm UDP Request

Request den Server, wird es vom UDPRequestQueue-Worker in eine UDP Request Queue gelegt. Der UDPRequestHandler-Thread entnimmt die Nachricht und bestimmt den Anfragentyp. Handelt es sich um eine Anfrage des Types RPC_Request, soll sie Aktionen auf dem Smartphone auslösen. Sie muss an das Smartphone gesendet werden, was über MQTT möglich ist. Dafür wird Sie in eine MqttAnswerQueue eingesetzt. Der MQTTHandlerThread entnimmt Sie und sendet Sie per MQTT ab. Ist Request hingegen ein SensorRequest, also eine Anfrage auf die ein Sensorwert geantwortet werden soll, wird der nachgefragte Sensorwert über die Klasse SensorDB entnommen und in die Udp_answer_queue eingetragen. Der AnswerQueueWorker entnimmt die Anfrage und sendet Sie per udp an die Library zurück.

Zusammenfassend erfüllen die Komponenten folgende Aufgaben. Der MQTTHand-

lerThread nimmt Nachrichten direkt per MQTT an und gibt die Anfrage weiter. Außerdem sendet er Nachrichten per MQTT, falls welche anfallen. Der MQTTRequestHandler kümmert sich um das Verfahren von MQTT Requests. Der UDPRequestQueue Worker nimmt wie der MQTTHandlerThread Anfragen, die per UDP übermittelt wurden an und gibt Sie zur Behandlung entsprechend weiter. Er sendet jedoch im Gegensatz keine Responses zurück. Hierfür gibt es den AnswerQueueWorker, dessen einzige Aufgabe es ist, Antworten per UDP zurück zu übermitteln.

Die Kommunikation zwischen den Threads funktioniert über synchronisierte Queues des queue-Moduls[?] der cpython Implementierung. Es handelt sich um eine threadsichere Monitorklasse, die einen gleichzeitigen Zugriff zweier unterschiedlicher Threads durch Locks verhindert.

5.3 Programmierumgebung

Die Programmierumgebung ist die Schnittstelle, die Programmierer für die Interaktion mit dem Smartphone verwenden. Sie besteht aus einer Bibliothek, die Funktionen anbietet, mit denen Programmierer Sensorwerte einlesen, oder Ausgaben auf dem Smartphone tätigen können. Sie können die Funktionen in ihren bestehenden Quellcode einbinden und die Funktionen dort verwenden. Die Bibliothek ist in den Programmiersprachen C, Java und Python vorhanden, damit Sie mit verschiedenen Programmiersprachen genutzt werden kann. In Java und Python ist Sie zudem Plattformunabhängig. Für C gibt es zwei Bibliotheken: Eine für Unix- und eine für Windows-Systeme. In C ist die Bibliothek prozedural mit statischen Methoden, in Java und Python objektorientiert implementiert.

Werden die bereitgestellten Funktionen aufgerufen, werden standardisierte Anfragen im JSON-Format generiert und an die Kontrollanwendung gesendet. Diese sendet die Daten gegebenenfalls an das Smartphone weiter oder antwortet direkt. Eine Übersicht ist in Abbildung 5.7 dargestellt.

Alle Anfragen werden über das UDP-Protokoll unter IPv4 versendet. Das Kontrollprogramm ist auf dem Port 5006 erreichbar, Bibliotheken auf dem Port 5005. Beide kommunizieren über die localhost-Adresse 127.0.0.1. Dadurch werden Datagramme über das Loopback-Interface gesendet. Das Loopback-Interface ist eine virtuelle Netzwerk-Schnittstelle des Betriebssystems eines PCs. Pakete werden nicht über externe Netzwerk-Schnittstellen wie Netzwerkkarten versendet, sondern verbleiben im Netzwerk-Stack des Betriebssystems. Die Latenzen sind dadurch mit weniger als 1 ms äußerst gering. Zum Senden und Empfangen von Anfragen werden Sockets verwendet. Für das Empfangen von Paketen müssen diese gebunden werden, für Sendevorgänge nicht.

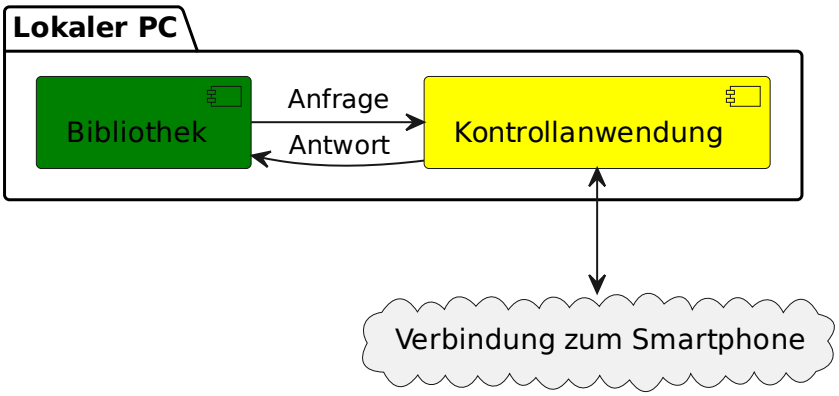

Abbildung 5.7: Schnittstellen der Bibliothek

Bei der Erstellung eines Phone-Objekts in Python und Java werden die Nachrichtenvorlagen für Anfragen und Antworten aus der Datei `protocol.json` geladen. Die Datei muss sich im Dateisystem im gleichen Ordner befinden wie die Bibliothek. Für die C-Bibliothek sind alle Methoden statisch definiert. Es gibt somit keinen Start-Punkt zu dem die Datei `protocol.json` eingelesen werden kann. Damit die Datei nicht für jeden Funktionsaufruf kontinuierlich eingelesen werden muss, muss Sie vom Programmierer einmal zum Start des Programms als cstring eingelesen werden. Anschließend muss dieser cstring für jeden Aufruf einer Funktion der Bibliothek als Parameter angegeben werden. Die Methode `get_file_content` kann, unter der Angabe des Dateipfades der `protocol.json`-Datei, aufgerufen werden, um den Dateiinhalt einzulesen. Zurückgegeben wird der Inhalt als cstring. Diese Lösung verringert die Anzahl der Lesevorgänge und die damit verbundenen durch IO verursachten Latenzen. Der Inhalt ist ab dem Einlesezeitpunkt auf dem Heap des Arbeitsspeichers gespeichert. Der Programmierer muss diesen am Ende seines Programms durch den Aufruf der `free`-Funktion wieder freigeben.

Kapitel 6

Verwendung der Lösung

In disem Kapitel wird die Verwendung des Frameworks anhand dem Beispiel *Alarmanlage* vorgestellt. Gezeigt wird, wie die Aufgabe in der Programmierumgebung gelöst wurde und wie sich die Aufgabe auf dem Smartphone äußert. In der Programmierumgebung wird dabei die Python-Bibliothek verwendet.

Damit Nachrichten ausgetauscht werden können,muss zuerst das Kontrollprogramm gestartet werden. Es wird mit dem Befehl `python ./server.py` in einer lokalen Shell gestartet. Der Vorgang wird in Abbildung 6.1 dargestellt. Es handelt sich um das Python-Skript *server.py*. Die Anwendung meldet, dass Sie sich erfolgreich

```
swt@pb22:~/thesis01/software$ python3 ./server.py
INFO:MqttHandlerThread:trying to connect to mqtt server
INFO:MqttHandlerThread:connected to server pma.inftech.hs-mannheim.de
INFO:MqttHandlerThread:mqtt subscribed to topic: 22thesis01/test
```

Abbildung 6.1: Start der Kontrollanwendung

mit dem MQTT-Broker verbunden und das korrekte Topic abboniert hat. Die Kontrollanwendung loggt neben diesen Informationen auch rpc-requests und responses. Sensor-requests und responses werden wegen ihrer hohen Anzahl nicht geloggt.

Die Implementierung der Lösung der Aufgabe ist in Listing 6.1 vorgestellt.

```
1 from time import sleep
2 import smartbit
3
4 p = smartbit.Phone()
5
6 while True:
7    prox_val = float(p.get_proxy())
8    if prox_val == 0.0:
```

```
9          p.write_text("ALARM")
10         for _ in range(5):
11            p.vibrate(1000)
12            p.toggle_led()
13            sleep(0.2)
14     sleep(0.5)
```

Listing 6.1: Alarmanlage-Beispiel

Die Bibliothek wird in Zeile 2 importiert. Dafür muss die Datei `smartbit.py` im gleichen Ordner wie das Programm gespeichert sein. In Zeile 4 wird ein Phone-Objekt erstellt, über das Sensor-Auslesemethoden wie `get_x_accel()` oder Smartphone-Ausgaben wie `vibrate()` augerufen werden können. Da das Programm nie abbrechen soll, außer wenn es in der shell gestoppt wird, wird in einer While-True Schleife der Näherungssensor immer wieder abgefragt. Anschließend wird eine halbe Sekunde gewartet um den Server nicht mit Anfragen zu überlasten. Ist der abgerfragte Näherungssensorwert wie in der achten Zeile überprüft wird 0.0, dann liegt eine Näherung vor. Auf dem Smartphone wird dann mit der Methode `write_text()` der Text ALARM usgegeben. Das Smartphone vibriert 5 mal hintereinander im Abstand von 200 ms für eine Sekunde und lässt die Signal-LED blinken. Insgesamt werden die RPC-Funktionen vibrate, toggle_button und write_text aufgerufen. Keine dieser Funktionen liefert einen Rückgabewert.

Die Bibliothek sendet die Nachrichten an das Smartphone. Nähert sich eine Person dem Smartphone, wird der Code in der If-Bedingung ausgeführt. Die jeweiligen Nachrichten werden als rpc_request per UDP an die Kontrollanwendung gesendet. Dort werden Sie angenommen, geloggt und per MQTT an das Smartphone weitergereicht. Eine Übersicht der gesendeten Nachrichten ist in Abbildung 6.2 zu sehen. Die Nachrichten werden im JSON Format übertragen. rpc_requests haben ein Typ-

```
swt@pb22:~/thesis01/software$ python3 ./server.py
INFO:MqttHandlerThread:trying to connect to mqtt server
INFO:MqttHandlerThread:connected to server pma.inftech.hs-mannheim.de
INFO:MqttHandlerThread:mqtt subscribed to topic: 22thesis01/test
INFO:DataHandler:rpc request: {'type': 'rpc_request', 'command': 'write_text', 'value': 'ALARM'}
INFO:DataHandler:rpc request: {'type': 'rpc_request', 'command': 'vibrate', 'value': '1000'}
INFO:DataHandler:rpc request: {'type': 'rpc_request', 'command': 'led_toggle', 'value': ''}
INFO:DataHandler:rpc request: {'type': 'rpc_request', 'command': 'vibrate', 'value': '1000'}
INFO:DataHandler:rpc request: {'type': 'rpc_request', 'command': 'led_toggle', 'value': ''}
INFO:DataHandler:rpc request: {'type': 'rpc_request', 'command': 'vibrate', 'value': '1000'}
INFO:DataHandler:rpc request: {'type': 'rpc_request', 'command': 'led_toggle', 'value': ''}
INFO:DataHandler:rpc request: {'type': 'rpc_request', 'command': 'vibrate', 'value': '1000'}
INFO:DataHandler:rpc request: {'type': 'rpc_request', 'command': 'led_toggle', 'value': ''}
INFO:DataHandler:rpc request: {'type': 'rpc_request', 'command': 'vibrate', 'value': '1000'}
INFO:DataHandler:rpc request: {'type': 'rpc_request', 'command': 'led_toggle', 'value': ''}
```

Abbildung 6.2: Nachrichtenversand der Kontrollanwendung

Feld, ein Kommando und einen Paramterwert für das Kommando. Write_text ist für Textausgaben auf dem Smartphone, vibrate um das Smartphone vibrieren zu lassen und led_toggle um die LED anzusprechen. Für write_text ~~wird der~~ gibt der Parameter-Wert den anzuzeigenden Text an. Vibrate erwartet für value die Zeitdauer der vibrierung in ms. led_toggle erwartet keinen Parameterwert, da der gesamte Aufruf lediglich die Farbe der LED wechselt. Es ist nicht vorgesehen, dass die Farbe manuell gesetzt werden kann.

Auf dem Smartphone ist die App zu Anfang im Initialmodus. Dieser ist in Abbildung 6.3 dargestellt. Zu sehen sind zwei Buttons, beschriftet mit A und B. Das textfeld

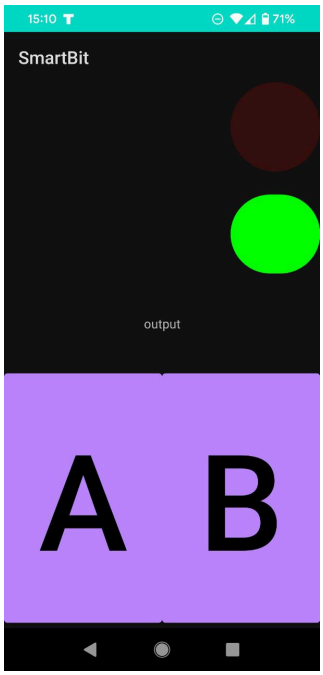

Abbildung 6.3: Initialzustand der Anwendung

liegt in der Mitte und zeigt zum Start den Text *output* an. Daneben gibt es noch eine Vorgangs-LED, die ausgegraut oben rechts über der Signal-LED liegt. Sie leuchtet, wenn gerade ein Vorgang wie das Ausführen einer Vibration auf dem Smartphone ausgeführt wird.

Nähert man sich dem Smartphone, geht es über in den Alarmzustand, der in Abbildung 6.4 dargestellt ist. Erkennbar ist, dass das Textfeld den Wert *ALARM* darstellt. Die Farbe der LED hat sich von grün auf rot und wieder auf grün geändert. Das ist erkennbar an der kleinen Screenshot-Vorschau links unten, die unmittelbar vor dem in der Abbildung zu sehenden Screenshot zu sehen ist. Die Vorgangs-LED leuchtet tief rot, um zu signalisieren, dass gerade eine Ausgabe ausgeführt wird. Nicht sichtbar ist das haptische vibrationsfeedback. Hierfür dient jedoch die Vorgangs-LED. Sie ist

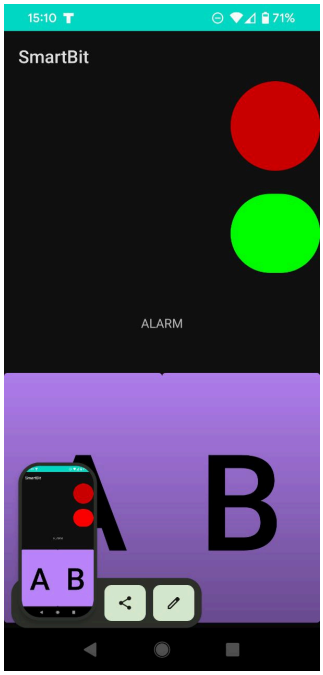

Abbildung 6.4: Initialzustand der Anwendung

auch hilfreich um bei Smartphones mit sanften Vibrationsausgaben eine Ausgabe zu erkennen.

Barth mag solche kurzen Absätze nicht, soweit ich weiß

Die Reaktionszeit liegt unter einer Sekunde. Die Ausgabe erscheint als direkter Grund der Annäherung.

Kapitel 7

Evaluation

In diesem Kapitel wird überprüft, ob die implementierte Lösung die Anforderungen erfüllt. Überprüft wird die qualitive Nutzbarkeit der Lösung anhand der spezifizierten Anforderungen. Für verschiedene Nutzungszenarien wie der Labor- oder Heimarbeit werden Latenzverhalten gemessen.

Transportwege müssen berücksichtigt werden. Die Zeiten werden in drei verschiedenen Umgebungen überprüft: In nächster Nähe zum MQTT-Server, in mittlerer Nähe und in einer virtuellen Umgebung über ein simuliertes NAT-Netzwerk.

Damit die Benutzbarkeit sichergestellt ist, müssen Latenzzeiten zwischen Smartphone und Programmierumgebung in einem akzeptablen Rahmen liegen. Um dies herauszufinden, werden die Latenzzeiten gemessen. In diesem Kapitel wird die Latenzzeit von drei Standorten aus gemessen.

Der erste Einsatzort ist im gleichen Netzbereich wie der MQTT-Broker, wodurch die Transferstrecke reduziert wird. Neben der Latenzzeit zum Broker ist jedoch auch die Anzahl der angemeldeten Geräte am gleichen AccessPoint interessant. Da die Geräte in direktem Wettbewerb um Sendezeit stehen, bedeuten mehr Geräte an einem AccessPoint größere Latenzen. In den Messungen werden nur WLAN-Verbindungen nach dem Standard WLAN 802.11n berücksichtigt. Es sind dabei stets beide Geräte, Lokaler PC und Smartphone per WLAN mit dem Internet verbunden. Untersucht wird lediglich die Zeitspanne zwischen rpc_request und rpc_response. Sensor_requests kommunizieren stets per loopback-interface mit der Kontrollanwendung. Daher sind die Latenzzeiten gering. Im Durchschnitt belaufen sie sich auf unter 1ms. Die Messungen sind für den Fall interessant, für den Studierende an der Hochschule die Lösung nutzen möchten, Zum Beispiel in Laboren.

Der zweite Einsatzort ist ein Heimnetz, das etwa 8 Hops entfernt vom MQTT-Broker liegt. Etwa 8 Geräte sind dabei auf dem Access Point angemeldet. Die Messergeb-

nisse sind relevant für die Heimarbeit oder das Erledigen von Hausaufgaben. Die Ergebnisse sind in Abbildung 7.1 dargestellt. Insgesamt wurden 100 Messungen

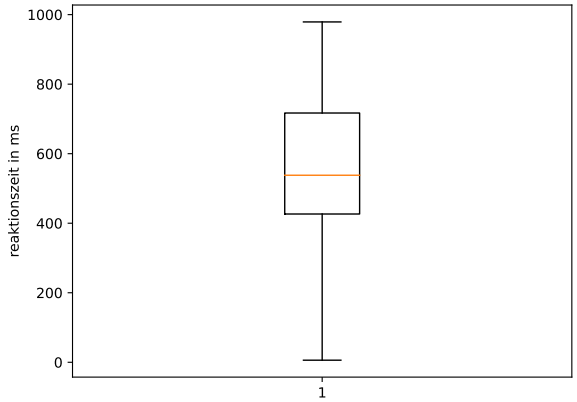

Abbildung 7.1: Zeitmessungen im Heimnetz

durchgeführt. Im Mittel beträgt die Reaktionszeit 537 ms. Die Streuung ist bei einer Standardabweichung von 238ms allerdings auch sehr breit gestreut. Alle Messungen blieben jedoch unter 1s Reaktionszeit.

Der letzte Einsatzort ist in einer virtuellen Maschine auf dem lokalen PC, das über den virtuellen Netzwerkkontroller aus einem NAT-Netzbereich auf das Internet zugreift. Das Smartphone befindet sich direkt im Netzbereich des lokalen PCs. Die gemessenen Daten sind in Abbildung 7.2 dargestellt. Wieder wurden 100 Messungen

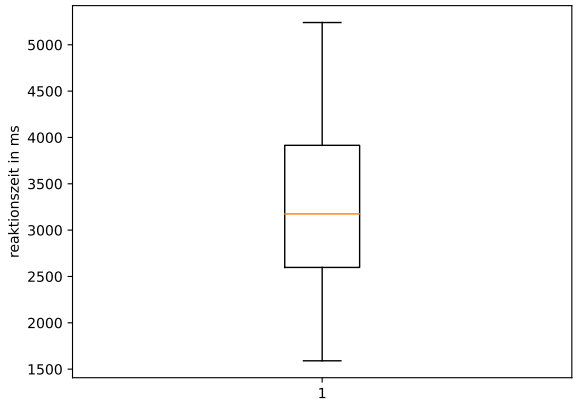

Abbildung 7.2: Zeitmessungen in der virtuellen Maschine

vorgenommen. Im Mittel beträgt die Reaktionszeit nun 3255 ms. Die Streuung ist mit einer Standardabweichung von 845 ms relativ gleichgroß. Die Reaktionszeit ist

also im Durchschnitt sehr hoch, etwa sechs mal langsamer als bei einer Verwendung der Programmierumgebung auf dem Hostsystem. Eine direkte Assoziation zwischen Eingabe und Ausgabe kann nicht garantiert werden.

Latenzprobleme per MQTT treten nicht auf. Trotz einem QOS-Level 0 werden auch Ausgabeanfragen sicher übertragen und ausgeführt. Da die Nachrichten über TLS übertragen werden, ist der Austausch sicher. Logging-Möglichkeiten in der Kontrollanwendung und der Android-App erleichterten die Entwicklung sehr und halfen bei der Fehlersuche.

Kapitel 8

Fazit

Die implementierte Lösung erfüllt die Anforderungen zufriedenstellend. Die im Vorfeld durchgeführte Konzeption der Lösung ermöglichte eine problemlose Implementierung. Während der Entwicklung wurden jedoch einige Designentscheidungen getroffen, welche die Lösung beschränken. Sensordaten werden von der Kontrollanwendung nicht für einzelne Geräte eingespeichert. Es ist nicht möglich, Programmcode simultan auf mehreren Smartphones gleichzeitig auszuführen. Gruppen-Sessions können nicht erstellt werden. Für Programme die die Java und Python-Schnittstelle könnte dies unintuitiv erscheinen. Die Bedeutung eines Phone-Objektes lässt auf eine zwischen Einzelgeräten unterscheidende Verwendung schließen. Ein weiterer Nachteil der Benutzerfreundlichkeit besteht in der zwingenden Verwendung von Hilfs-Bibliotheken für den Nachrichtenaustausch über JSON. Sowohl für Java, als auch für C sind zusätzliche JSON-Parser für die Kommunikation zwischen Bibliothek und Kontrollanwendung nötig. Für eine erfolgreiche Kompilierung in C muss der Dateipfad für diesen dem Linker bekannt gemacht werden. Für die interne Struktur ist die Datenverwaltung innerhalb des Programms ausreichend. Durch den asynchronen Ansatz des Multithreading können beide Kommunikationspartner gleichzeitig über das Kontrollprogramm kommunizieren und zwischengespeicherte Daten auslesen oder bearbeiten. Die Möglichkeit, mehrere Werte zum internen Datenspeicher hinzuzufügen, wird jedoch nicht unterstützt. Häufige Sensor-Anfragen werden mit dem gleichen, einzeln zwischengespeicherten Sensorwert beantwortet, welcher zusätzlich kein Ablaufdatum besitzt. Abbrüche der Übertragung resultieren in der kontinuierlichen Übermittlung des zuletzt eingespeicherten Sensorwerts.

Zur Ablösung von Hilfsbibliotheken eignet sich die Verwendung eines Binärprotokolls. In Enumerationen konvertierte Nachrichtenformate und Parameter könnten statt der implementierten Darstellung mit JSON eingesetzt werden. Dadurch entfiehle nicht nur der Nachteil der unintuitiven Einbindung. Die Nachrichtengröße

würde ebenfalls verringert werden. Energieeffizienz der Android-App wurde zum Vorteil der Latenzreduzierung während der Implementierung nicht wesentlich berücksichtigt. Zum Start der Anwendung werden die Sensormessprozesse unmittelbar gestartet und Sensorwerte an die Kontrollanwendung gesendet, was einen erhöhten Energieverbrauch bedeutet. Sollten sich die Startzeiten der Sensormessprozesse nicht wesentlich auf die Latenzen auswirken, wäre eine auf Anfragen basierende Sensmessdatenübertragung erwägenswert,um den Energieverbrauch zu reduzieren.

Literaturverzeichnis

Online-Quellen