hochschule mannheim

Fakultät Informationstechnik
Studiengang Technische Informatik

Bachelorarbeit

Smartphones als Sensorbox

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

In dieser Arbeit wurde ein Framework erstellt um Smartphonesensoren über eine Programmierumgebung auszulesen und Ausgaben auf dem Smartphone auszuführen. Hierfür wurde eine Android-Anwendung, eine Kontrollanwendung und drei Softwarebibliotheken in dem Sprachen C, Java und Python implementiert. Als Verbindungstechnologien kommen UDP und MQTT zum Einsatz.

Inhaltsverzeichnis

Kapitel 1

Einführung

MINT-Berufe leiden hierzulande unter einem akuten Fachkräftemangel. Das Institut der deutschen Wirtschaft ermittelte für April 2021 ein Unterangebot von 145.100 Personen [**?**] in 36 MINT-Berufskategorien. Digitalisierungs-Projekte geraten dadurch ins Stocken.
Nicht zuletzt er auch ein Kräftemangel in der Softwareentwicklung. Es fehlen Programmiererinnen und Programmierer. Softwareentwicklung ist gerade in der Lernphase nicht trivial und abstrakt. Unlebendige Übungsaufgaben die beispielsweise Konsolenein- und ausgaben realisieren schrecken zukünftige Programmierinnen und Programmierer eher ab als sie zu ermutigen.
Microcontroller sind bereits eine große Hilfe, da hier spielerisch kleine Projekte realisiert werden können. So können schon früh in Schulen Kinder an die Programmierung herangerführt werden. Sie lernen spielerisch kleine Programme zu entwickleln und verstehen die ihnen beigebrachten Abläufe durch schnelle Anwendung. Microcontroller sind jedoch auch mit Anschaffungskosten verbunden und für kleine Anwendungen, welche Sensoren verwenden, wird viel zusätzliches Material wie zum Beispiel Breadboards, Verbindungskabel und Erweiterungsboards benötigt. Moderne Smartphones bieten hier Abhilfe da Sie meistens mit verschiedenen Sensoren bespickt sind, wie zum Beispiel: Kompass, GPS, Microphon oder Kamera. Viele Kinder besitzen bereits mit 10 Jahren [**?**] ein Smartphone.

Im Rahmen dieser Arbeit wurde ein Framework entwickelt, dass das Smartphone von Anwendern einbindet um Sie beim Programmierenlernen zu unterstützen. Es heißt SmartBit und orientiert sich an bereits bestehenden Schulungsprojekten wie dem Arduino oder BBC Microbit.
Moderne Smartphones sind mit einer Vielzahl an Sensoren ausgestattet. Diese umfassen zum Beispiel Lage- und Gyrosokopsensoren, jedoch auch Magnetfeld-, Näherungs oder Luftdrucksensoren. Diese können von Schülern oder Studenten ausgele-

sen werden um so zum Beispiel auf Smartphonerotationen, Lageänderungen, usw. reagieren zu können. So können abhängige Ausgaben angezeigt werden. Die direkte Nutzung eines haptischen, physischen Geräts in kombination mit statischem Code fördert die Einbindung in die Aufgabe.
Schüler und Studenten lösen vorgefertigte Aufgaben in denen Sie Sensordaten auslesen, oder Messungen bzw. Aktionen auf dem Smartphone starten müssen. Dieses Interaktive Element soll die Lernerfahrung optimieren. Eine Studie von Martinha Piteria und Carlos Costa analysierte 2013 am Polytechnischen Institut Setubal in Portugal die Lernerfahrung von 64 Studierenden.[**?**] Praktische Lernsituationen wie Labore oder praktischen Lehrstunden während der Vorlesung.
Dieser positive Lerneffekt soll durch die Aufgabenstellungen ausgenutzt werden.

Das Framework besteht aus einer programmiersprachenunabhängigen Bibliothek, einer Server-Anwendung und einer mobilen Anwendung für Android Smartphones. Die Library stellt Funktionsaufrufe bereit. Diese können aufgerufen werden und geben Sensordaten zurück oder starten Aktionen in der Smartphone-Anwendung.
Die Serveranwendung dient als Vermittlungseinheit zwischen Smartphone und Bibliothek. Aufbau und Funktionsweise werden in Kapitel 3.7 erläutert.
Das Smartphone dient als Interaktionsobjekt. Hierfür wurde eine Android-Anwendung entwickelt. Sie stellt Ausgabemöglichkeiten beispielsweise über ein Textfeld oder Vibration zur Verfügung. Zudem misst Sie kontinuierlich Sensordaten und übermittelt diese. Der konkrete Aufbau der Anwendung wird in 2.3 erläutert.

Kapitel 2

Aufgabenstellungen

2.1 Aufgaben

Dieses Kapitel enthält verschiedene Beispielaufgaben die mit dem Framework gelöst werden sollen. Die Benutzung der API dazu wird später geschildert.

Disco
Die LED muss ganz schnell blinken.

Diebstahl-Alarm
Wenn nach dem Telefon gegriffen wird soll die LED blinken. Wenn man sich davon entfernt soll sie wieder aus sein.

Würfeln
Das Smartphone wird geschüttelt. Ein random Zahlenwert wird zurückgegeben. Je nachdem welches Ergebnis zurückkommt soll ein Wert auf dem Display angezeigt werden.

Klatsch-Zähler
Der Anwender möchte wissen wie oft in einem bestimmten Zeitraum geklatscht wurde. Ein Methodenaufruf startet in der Androidanwendung eine Activity, die innerhalb des im Argument genannten Zeitraums die Anzahl der maximalen Lautstärkeamplituden des Mikrophons misst und die Anzahl zurückgibt. Diese soll im Textfeld angezeigt werden.

Dreh-Zähler
Ein Nutzer möchte in einem bestimmten Zeitraum zählen wie oft das Smartphone

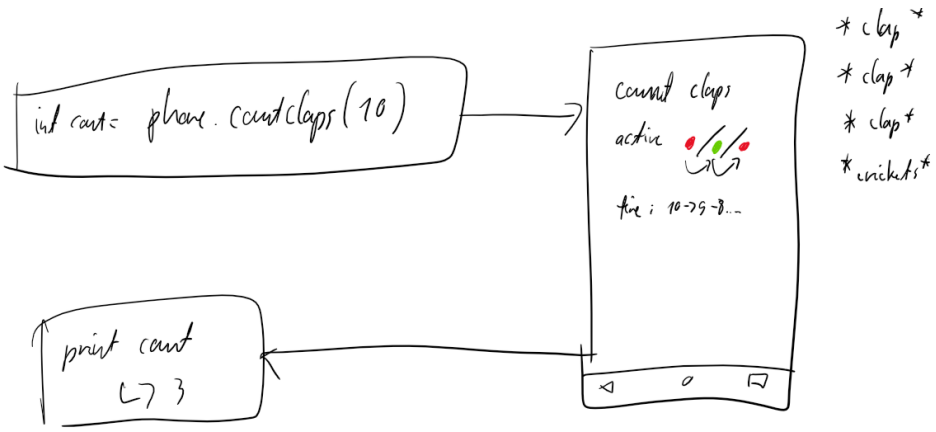

Abbildung 2.1: Klatsch-Zähler

gedreht wurde.

2.2 API

Gelöst werden sollen die Aufgaben durch das Aufrufen der API. Diese bietet die benötigten Funktionen an. Die Aufrufe sind frei miteinander kombinierbar, so dass Aufgaben erweitert werden können.

Eingaben

Auslesen der accelerometer-Daten
Ein User will den Wert der X, Y und Z Koordinaten des Smartphones wissen. So kann er bspw. feststellen, ob das Smartphone gerade nach unten, oben oder horizontal bewegt wurde. Kippbewegungen werden nicht detektiert.

```
1 Phone p;
2 float x_val = p.get_x_accell();
```

Lautstärkepegelmessung
Ein User möchte den aktuellen Lautstärkepegel messen.

```
1 Phone p;
2 float volume = p.getVolume();
```

Annäherungssensor-Messung
Ein User möchte wissen, ob ein Objekt unmittelbar vor den Annäherungssensor steht. Der Aufruf erfolgt folgendermaßen.

```
1 Phone p;
2 bool triggered = p.primity_triggerd();
```

Amplituden-Spike-Messung mit Zeitraum
Ein User möchte wissen, wie oft die Lautstärke innerhalb eines angegebenen Zeitraums t ein gewisses Limit überstiegen hat. Der Aufruf könnte dabei folgendermaßen laufen.

```
1 Phone p;
2 int timeframe = 1000; //1000 ms = 1s
3 int num = p.getNumOfSpikes(timeframe);
```

Umdrehungsmessung mit Zeitraum
Ein User möchte wissen, wie oft das Smartphone innerhalb eines angegebenen Zeitraums t gedreht wurde. Der Aufruf sieht folgendermaßen aus.

```
1 Phone p;
2 int timeframe = 1000; //1000 ms = 1s
3 int num = p.getNumOfSpikes(timeframe);
```

Ausgaben

Text-Ausgabe
Um auf dem Smartphone einen beliebigen Text anzuzeigen. Dies kann er zum Beispiel wie im folgenden Beispielcode machen.

```
1 Phone p;
2 p.displayText("Hallo Welt");
```

LED leuchten lassen
Ein User möchte eine LEDäuf dem Display ansteuern. Dies kann er so machen.

```
1 Phone p;
2 p.led_on();
3 p.led_off();
```

LED toggeln
Ein User möchte den Zustand einer LED auf dem Display auf äusßtellen wenn Sie an war und auf än"wenn Sie aus war. Dies kann er so machen.

```
1 Phone p;
2 p.led_on();
3 p.led_off();
```

Vibrationsauslöser
Ein Nutzer möchte das Smartphone für eine gewisse Zeitspanne vibrieren lassen.

```
1 Phone p;
2 p.vibrate(500);
```

2.3 Android Anwendung

Für Ausgaben auf dem Smartphone existiert die Android Anwendung Smartbit, die verschiedene Interaktionsmöglichkeiten bereitstellt. Das Userinterface wird in Abbildung 2.2 gezeigt.
Die Anwendung besteht aus zwei Leds, den beiden Punkten im oberen rechten Eck. Ein Textfeld in der Mitte kann zur Ausgabe von Text verwendet werden. Die beiden Buttons können gedrückt werden. Neben den sichtbaren Elementen kann die Anwendung das Smartphone außerdem vibrieren lassen.

2.4 Softwareanforderungen

Aus den unter 2.2 genannten Funktionen ergeben sich die Anforderungen an das Framework.
Es wird unterschieden in Funktionen die Sensordaten auslesen und Funktionen die Aktionen auf dem auslösen.

2.4.1 Sensordaten-Funktionen

Um das optimale Lernergebnis zu erhalten müssen Sensordaten responsiv vorliegen. Je schneller Richtungs-, Helligkeits oder Lautstärkedaten vorliegen desto schneller kann im Programm drauf reagiert werden. So können Experimente eher begriffen werden, da Auswirkungen in der Realität instantan Auswirkungen auf den vom Schüler geschriebenen Programmablauf haben. Eine maximale Responszeit von ca. 20 ms ist hierfür zielführend.
Beim Aufruf der Funktion wird kein Aufruf gestartet der das Smartphone anweist eine einzelne Sensormessung zu starten und den aktuell vorliegenden Sensorwert zurückzusenden. Die Latenzen zwischen Smartphone und Bibliothek könnten je nach Verbindungsart, Bandbreite und Geräteanzahl im WLAN stark varrieren. Das Smartphone muss in periodischen Abständen Sensordaten messen und senden. Die Sensorwerte werden dann zentral auf dem gleichen Rechner, auf dem auch die Bibliothek läuft zwischengespeichert. So wird die mögliche Distanz zwischen Sender

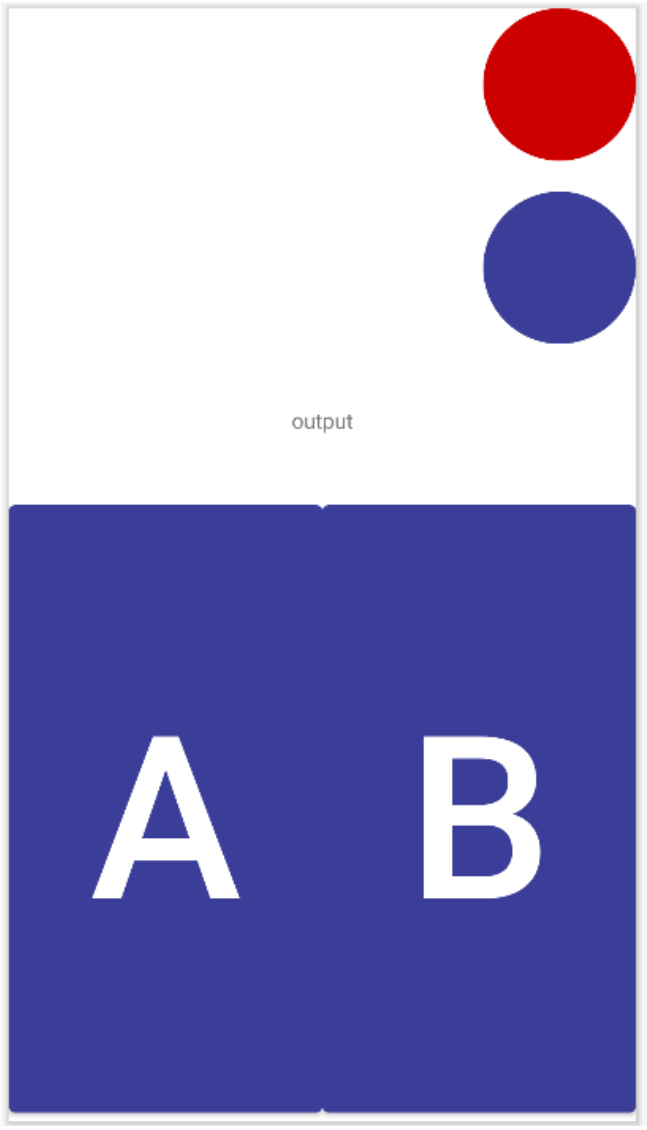

Abbildung 2.2: Userinterface Android Anwendung

und Empfänger beschränkt auf den gleichen Rechner. Sensordaten liegen immer zwischengespeichert vor. Neue Daten werden beim Eintreffen der periodischen Aktualisierungen eingetragen. Ein Funktionsaufruf der Sensordaten abfragt muss dann auf die zentral zwischengespeicherten Sensordaten zurückgreifen. So wird der Responszeit erhöht und es liegt immer das aktuellste Sensor-Ergebnis im Zwischenspeicher vor. Der Tätigkeitsbereich vom Smartphone umfasst jedoch nicht nur die Messung und Übermittlung der gemessenen Daten.

2.4.2 Auslösende Funktionen

Die Anwendung auf dem Smartphone bietet ebenfalls verschiedene UI-Elemente welche aus der Ferne per Funktionsaufruf bedient werden. Außerdem werden zeitlich beschränkte Messungen umgesetzt die auf Anfrage gestartet werden sollen. Diese Art der Aufrufe benötigt keinen kontunierlichen Strom aus Werten, sondern wird einzeln aufgerufen und geben gegebenenfalls einzeln Werte zurück. Der Fokus liegt bei diesen Anfragen nicht auf Responsivität, sondern auf Sicherheit des Aufrufs. Bei dieser Art Aufruf werden Daten erst auf Anfrage generiert und dann, sofern vorgesehen, zurückgesendet.

Kapitel 3

Architektur

Der Aufbau des Frameworks besteht aus einer mobilen Anwendung für Android-Smartphones, Bibliothek in der Programmiersprache C und einer Middleware die den Austausch Koordiniert. Der Aufbau ist in Abbildung 3.1 dargestellt. Clientseitig

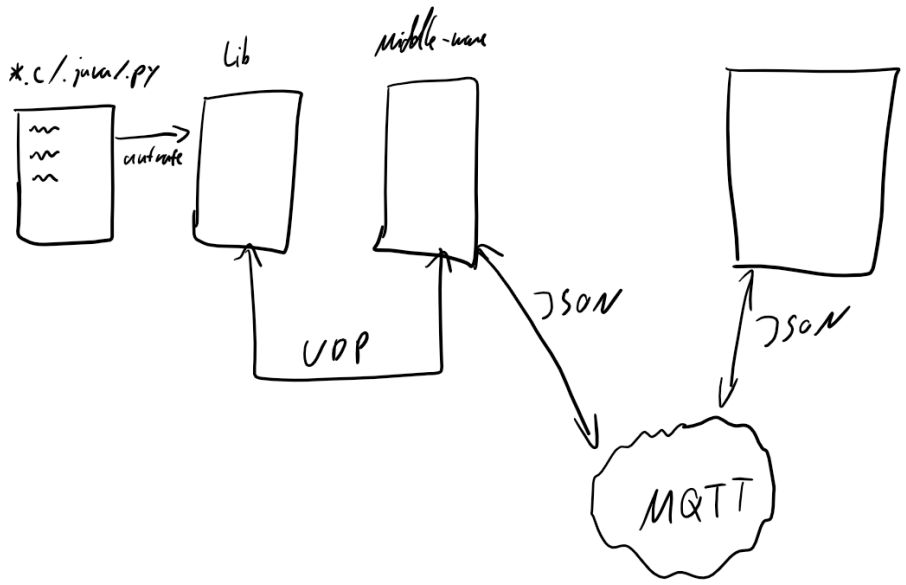

Abbildung 3.1: System-Aufbau

erfolgen sämtliche Aufrufe die Sensordaten abgreifen oder Steueranfragen senden immer erst per UDP über die Middleware.
Die Middleware sammelt, sobald sich eine Android-Anwendung bei ihr anmeldet, gewisse Sensordaten und speichert Sie intern zwischen, damit Sie schnell vorrätig vorliegen. Die Speicherung erfolgt in einem seperaten Thread, der die letzten Zustände der Sensordaten hält. Werden Sensordaten abgerufen werden Sie aus der Liste entfernt. Sind noch keine Daten vorhanden, oder wird ein Sensor angefragt

der im Smartphone nicht existent ist, wird ein Fehlercode zwischengespeichert. Die Verfügbarkeit der jeweiligen Sensoren wird beim Start der Anwendung ermittelt. Steueraufrufe werden beim Aufruf über ein seperates Topic versandt. Der Austausch erfolgt auf mehreren Topics, da manche Nachrichten, wie Steuerbefehle wie in 2.2 erwähnt, mit einem höheren QOS-Level versendet werden müssen als im Moment existente Sensordaten die nur eine kurzzeitige Relevanz besitzen und deren Zustellung nicht obligatorisch ist. Auf diesem Topic werden Nachrichten mit der QOS von 2 versandt auf dem für reguläre Sensordaten mit einer QOS von 0.

3.1 Nachrichtenformate

Damit der gesamte Funktionsablauf des Frameworks korrekt funktioniert müssen Nachrichten zwischen den einzelenen Komponenten ausgetauscht werden. Hierfür wird ein Format definiert dass die in Kapitel 2.4 erwähnten Anforderungen sowohl auf Library-, Middleware-, und Smartphone-Seite umsetzt.

Die Nachrichten werden im JSON-Format übertragen. Jede Nachricht beeinhaltet mindestens die Angabe eines Nachrichtenformat-Typs. Für die Aufrufe wurden folgende Nachrichtenformate definiert:

1. sensor_request
2. sensor_update_request
3. rpc_request
4. rpc_response

Die Nachrichtenformate werden als Vorlagen in einer zentralen Datei abgelegt. Diese werden in der Library, Middleware und auf dem Smartphone eingelesen, damit Sie nur einer Stelle definiert werden müssen, wie in 3.2 dargestellt. Neben den Nachrichtenformaten werden hier auch weitere Konfigurations-Konstanten wie Commands für rpc_requests, Namen der Sensordaten und Namen der MQTT Topics spezifiziert spezifiziert.

sensor_requests werden vom Endanwender-PC per UDP an die Middleware gesendet. Sie beeinhalten neben dem Typ das Feld `sensor_type`. Dieses definiert den Sensortyp für den eine Anfrage generiert wurde. Eine Übersicht ist in Listing zu finden.
Im Listing wurde der Platzhalter `TYPE` für alle möglichen Sensortypen angegeben. Eine vollständige Aufschlüsselung ist in Tabelle 3.1 zu finden.

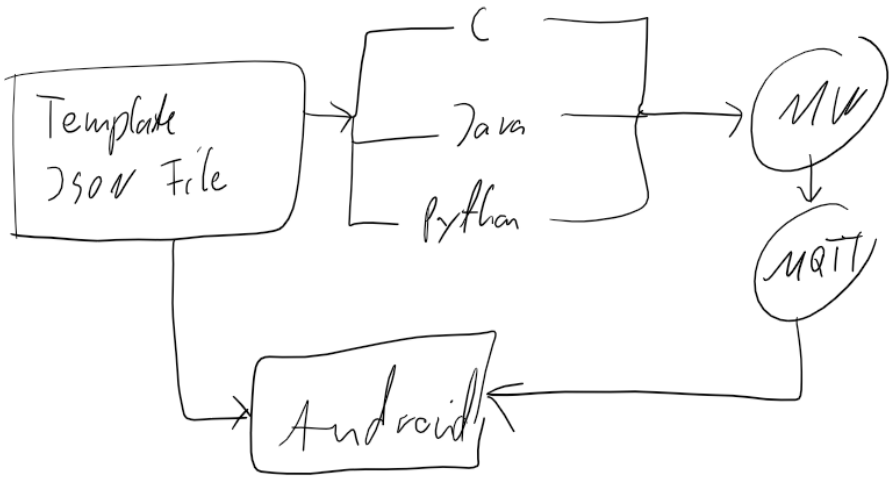

Abbildung 3.2: Zentrale Konfigurationsdatei

```
1 {
2     "type":"sensor_request",
3     "sensor_type": "TYPE"
4 }
```

Listing 3.1: sensor_request

sensor_update_requests werden von Smartphone über MQTT an die Middleware versandt. Sie umfassen ebenfalls wie `sensor_requests` das Feld `sensor_type`, jedoch zusätzlich auch das Feld `value` in dem der gemessene Sensorwert gespeichert wird. Dieser wird von der Middleware in eine interne Datenstruktur eingetragen.

rpc_requests werden vom Endanwender-PC per UDP an die Middleware gesendet. Sie lösen Aktionen auf dem Smartphone aus wie zum Beispiel das anschalten der LED, oder das Starten von Messungen. Sie werden von der Middleware per MQTT an das Smartphone weitergereicht. Die Übermittlung erfolgt jedoch über ein seperates Topic mit einem Quality of Service Wert von 2 um eine garantierte Übertragung zu gewährleisten.

rpc_requests und response beeinhalten außerdem mindestens ein `command`-Feld und ein `value`-Feld.

Nachfolgendes Listing zeigt beispielsweise den Aufruf die LED einzusschalten.

```
1 {
2     "type" : "rpc_request",
3     "command" : "set_button",
4     "value" : "true"
```

```
}
```

Folgende Nachricht beschreibt beispielsweise den übermittelten Sensorwert des accelerometers.

```
{
    "type" : "sensor_response",
    "sensor_type" : "accell_x",
    "value" : "-0.151342"
}
```

3.2 Nachrichtenablauf

Der Nachrichtenablauf ist in Abbildung 3.3 dargestellt.

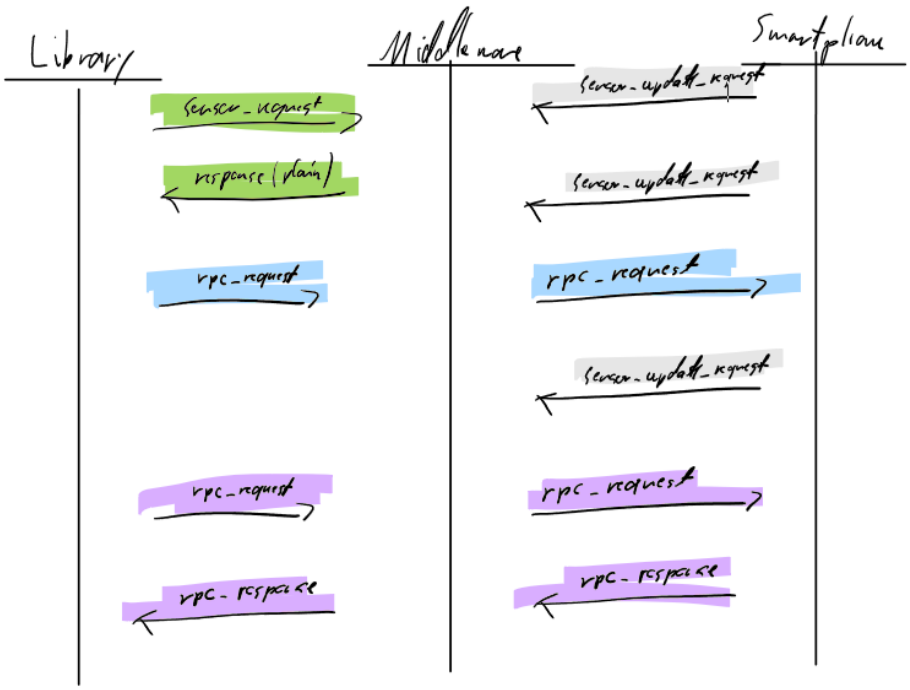

Abbildung 3.3: Nachrichtenablauf

Grau dargestellt ist hier die periodische aktualisierung der Sensordaten durch sensor_update_requests. Diese werden mit einer auf dem Smartphone definierten Taktung an die Middleware übertragen Die Middleware trägt die neuen Daten dann in der internen Datenbank ein.

Grün dargestellt ist der Ablauf eines sensor_requests. Aus der Library heraus werden sensor_requests gesendet. Die Middleware, die die aktuellen Sensorwerte vorhält sendet dann als Antwort den Sensorwert als einzelnen Wert, also nicht im JSON-Format zurück.
Blau dargestellt ist ein rpc_request, dass ein Command auf dem Smartphone ausführt, dass keinen Rückgabewert zurückliefert. Änlich wie bei sensor_update_requests handelt es sich hier um fire-and-forget-requests.
Lila dargestellt sind rpc_requests die ein Command ausführen dass einen Rückgabewert erwartet, wie es beispielsweise bei zeitlich getakteten Messungen der Fall ist. In diesem Fall wird eine rpc_response-Nachricht mit versendet.

3.3 Verwendete Technologien

In diesem Kapitel werden die Technologien und Protokolle beschrieben die beim Nachrichtenaustausch zwischen Library, Middleware und Smartphone zum Einsatz kommen.
Der Fokus liegt auf Serialisierung und Transport. Da im gesamten Prozess zwischen den drei Einheiten unterschiedliche Programmiersprachen auf verschiedenen Plattformen zum Einsatz kommen müssen sämtliche Protokolle und Notationen von möglichst vielen Programmiersprachen unterstztützt werden.
Beim Transport liegt die Anforderung vor allem auf einer schnellen Ausführung, da sich Sensordaten schnell ändern und ein Lerneffekt am Besten eintritt wenn keine hohen Latenzen bei der Übertragung auftreten.

3.3.1 JSON

JSON ist eine Abkürzung für JavaScript Object Notation. Es handelt sich um ein Nachrichten-Austausch-Format [**?**]

Der Aufbau ist minimalistisch. Objekte können mit Schlüssel-Wert-Paaren serialisiert werden. Auch Arrays können eingesetzt werden. Durch den einfachen Aufbau werden für eine Nachricht auch nicht viele Ressourcen benötigt. Gerade bei der Übertragung von Sensordaten ist ein schneller Datenaustausch sehr positiv.
JSON wird außerdem von vielen Programmiersprachen wie Python und Java nativ unterstützt. Da die Bibliotheken programmiersprachenunabhäng bereitgestellt werden sollen bietet sich JSON als gemeinsame Notation hervorrangend an. Da es in C keinen eingebauten JSON Parser gibt wird hier auf den cJSON [**?**] zurückgegriffen.

3.3.2 MQTT

MQTT steht für Message Queuing Telemetry Transport und ist ein Client-Server Protokoll das auf TPC basiert. Es kommt häufig mei Machine-to-Machine Interaktionen zum Einsatz. Dies umfasst Lösungen im IoT-, Smarthome und Automatisierungstechnik-Bereich. Durch den platzsparenden Header von 2 Bytes und einer maximalen Nachrichtengröße von ca. 260 MB ist es gleichzeitig ein leichtgewichtiges jedoch auch flexibles Protokoll zum Nachrichtenaustausch.
Für den Nachrichtenaustausch registrieren sich Clients an einem Broker, dem MQTT-Server, an ein oder mehrere Topics. Auf diesen Topics können dann Nachrichten gesendet und empfangen werden. Topics können hierarchisch geschichtet werden. Subtopics können dann pro ebene oder rekursiv für alle Subtopics abboniert werden. Dieses Feature ist jedoch für die Entwicklung nicht weiter von Belang. Nachrichten werden von Clients an den Broker gesendet. Dieser hält den Nachrichteninhalt intern für das jeweilige Topic.
Alle Clients besitzen einen Identifier, so dass empfangene Nachrichten zurückgeführt werden können. Somit sind auch Topic-Interne Filter möglich nur Nachrichten von bestimmten Clients zu empfangen.
Nachrichten können mit einer Quality of Service (QoS), einer Übertragungsgüte, übertragen werden. Insgesamt gibt es drei Stufen: 0, 1 und 2. Je höher die QoS-Stufe desto höher die Latenz der Übertragung.
Bei einer QoS von 0 werden Daten einfach übertragen ohne sicherzugehen, dass die Nachricht auch wirklich beim Broker ankommt. Bei QoS 1 wird eine Nachricht so lange auf einem Topic gesendet bis der Broker den Empfang einmal bestätigt. Es kann jedoch auch passieren, dass der Broker die Nachricht mehrmals empfängt und nur einmal bestätigt. Bei QoS 2 wird die Nachricht einmal an den Broker gesendet. Dieser muss dann den Empfang der Nachricht einmal bestätigen. Erst dann gilt die Nachricht als Empfangen.
Damit die verlangte Güte sichergestellt wird müssen Sender und Empfänger mit der gleichen Güte senden bzw. ein Topic mit der gleichen Güte-Stufe abbonieren. Wird ein Topic mit einer geringeren Güte abboniert als die Güte der publizierten Nachricht wird die Nachricht mit der Güte des Abbonenten übertragen. Wenn eine Nachricht mit geringerer Güte auf einem Topic publiziert wurde als dieses Topic abboniert wurde wird die Übertragung mit der Güte-Stufe des publizierenden Clients übertragen.

3.3.3 UDP

UDP ist ein verbindungsloses Netzwerkprotokoll, dass auf der Transportschicht des TCP/IP-Stack zum Einsatz kommt.

3.4 Bibliotheken

Die Bibliotheken bilden die Einstiegsstelle für Nutzer um die Anwendung fernzusteuern oder Sensorwerte abzufragen. Funktionsaufrufe liegen jeweils in den Sprachen C, Java und Python vor. Daten werden wie in Kapitel 3 in Abbildung 3.1 gezeigt per UDP an eine Serveranwendung gesendet. Diese sendet die Daten dann nach gegebenenfalls per MQTT an das Smartphone weiter, oder wieder per UDP zurück. Die Bibliotheken senden und empfangen alle jeweils Daten per UDP. Hierfür werden Sockets benötigt. Für's Empfangen muss der Socket binded sein. Die Serveranwendung ist auf dem Port 5006 erreichbar. Antworten erwarten die Bibliotheken auf dem Port 5005.

3.5 Android Anwendung

Die Android-Anwendung läuft auf dem Smartphone des Anwenders. Sie besteht aus einer Haupt-Activity deren außerlicher Aufbau in Kapitel 2.3 beschrieben ist.
In der Haupt-Activity wird ein Service gestartet und gebunden. Der Service baut eine Verbindung zu einem MQTT Server auf. Dort verbindet er sich den beiden Topics. Die Activity bindet beim Start den MQTT-Service ein. Stellt ein Client über die Bibliothek auf einem Computer eine Anfrage wird dieser Aufruf, wie z.B. eine Ausgabe auf einem Textfeld an das Smartphones, zuerst zur Middleware geleitet. Diese leitet die Anfrage weiter an das Smartphone, dass dann den jeweiligen Befehl ausführt. Dies beeinhaltet vor allem Ausgaben, sowie Eingaben die eine Nutzerinteraktion mit reduziertem Ergebnis. Zum Beispiel alle Eingaben die etwas in einem angegebenen Zeitraum messen. Falls ein Sensor angefragt wird der auf dem Smartphone nicht existiert wird eine Nachricht mit einer Fehlermeldung zurückgegeben.

3.5.1 Kommandos und Ausgaben

Für die Ausgabe auf dem Smartphone sind verschiedene Kommandos definiert. Diese sind der Tabelle **??** zu entnehmen. `CMD-Kürzel` beschreibt die Notation des Kürzels mit dem eine Aktion ausgeführt werden kann. Diese wird unter `Beschreibung` kurz zusammengefasst. `return` gibt an, ob der Aufruf des Requests eine Antwort rücksendet und somit auch, ob ein Aufruf der Funktion in der Library blockiert oder nur sendet.

3.6 Sensoren

Smartphones beeinhalten verschiedene Sensoren die Daten über die Umgebung erfassen können. In der Android Anwendung werden folgende Sensortypen verwendet:

- Lineare Beschleunigungssensoren
- Mikrofon
- Annäherungssensor
- Gyroskop

3.6.1 Lineare Beschleunigungssensoren

Beschleunigungs- bzw. Lagesensoren messen die Beschleunigung in m/s^2 für die drei Bewegungsrichtungen: X-, Y- und Z-Achse in einem festgelegten Zeitraum. Eine Übersicht über die Anordungen der drei Axen ist in Grafik 3.4 zu sehen. Die X-Achse

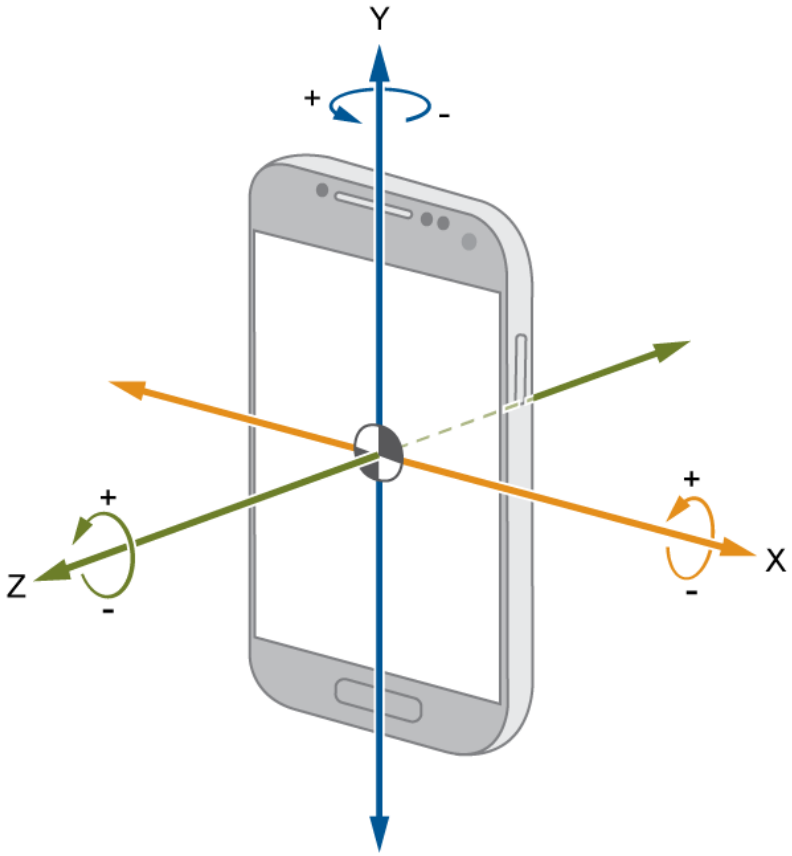

Abbildung 3.4: Android-Koordinatensystem

verläuft horizontal durch das Display des Smartphones hindruch, die Y-Achse vertikal und die Z-Achse durchschneidet das Smartphone in die Tiefe.

Die Frequenz mit der Messwerte erstellt werden kann manuell angegeben werden. Hierfür stehen vier Schnelligkeitsstufen bereit.:

- SENSOR_DELAY_FASTEST : Kein Verzögerung. Verwendet die Frequenz des Sensors.
- SENSOR_DELAY_GAME : Verzögerung um 1ms
- SENSOR_DELAY_UI : Verzögerung um 2ms
- SENSOR_DELAY_NORMAL : Verzögerung um 3ms

Die mit der jeweiligen Frequenz aufgenommenen Beschleunigungssensordaten beeinhalten jedoch auch die Ergbeschleunigung. Diese muss für die bereinigten, realen Werte zuerst noch von den aufgenommenen Werten subtrahiert werden[**?**].

3.6.2 Angaben für die Nachrichtenformate

Alle Sensordaten besitzen ein festgelegtes Kürzel zur Standardisierung des Nachrichtenverkehrs. Sie dienen vor allem der Adressierung der jeweiligen Daten in der Middleware und in der Library.
Die Sensortyp Kürzel sind in Tabelle 3.1 zu finden.

TYPE-Kürzel	Beschreibung
accell_x	Lagesensor für die X-Richtung
accell_y	Lagesensor für die Y-Richtung
accell_z	Lagesensor für die Z-Richtung
gyro_x	Gyroskopsensor für die X-Richtung
gyro_y	Gyroskopsensor für die Y-Richtung
gyro_z	Gyroskopsensor für die Z-Richtung
prox	Näherungssensor

Tabelle 3.1: Sensor-Kürzel mit Beschreibung

3.7 Server-Anwendung

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-Liznenz stehende MQTT-Library Paho der Eclipse-Foundation zum Einsatz. [**?**]

3.7.1 Anforderungen

Aus den Anforderungen ergeben sich folgende Aufgaben die der Server umsetzen muss:

- Sensoranfragen schnell beantworten
- Anfragen die Funktionen auf dem Smartphone starte, müssen schnell an das Smartphone weitergereicht und gegebenfalls wieder beantwortet werden.

Damit Sensoranfragen schnell beantwortet werden können, werden die aktuellen Sensorworte fortlaufend vom Smartphone an den Server übermittelt. Dieser speichert Sie dann, sobald ein Sensorwert gemessen und übermittelt wurde in eine interne, threadsichere Datenstruktur. Hierdurch wird insbesondere der Latenzunterschied zwischen Smartphone und Localhost berücksichtigt. UDP Anfragen per Localhost haben eine wesentlich geringe Latenz als MQTT-Anfragen, auf die das Smartphone reagieren muss. Durch das Cachen auf dem Server liegen bei Anfragen per UDP, also von der Library aus, immer Sensordaten vor.
Um Funktionsanfragen zeitnah auszuführen müssen unterschiedliche Netzwerklatenzen berücksichtigt werden. Die Übermittlung erfolgt asynchron zwischen Library und Smartphone, wobei zu erwarten ist dass Übermittlungen an das Smartphone mit einer höheren Latenz übertragen werden, als Anfragen zwischen Library und Server, da sich beide auf dem gleichen Host befinden. Die Verbindung zwischen Library und Server erfolgt darüber hinaus über localhost und somit über ein Loopback-Device. Loopback-Devices reichen Netzwerkpakete nicht herkömmlich über ein physisches Netzwerkinterface weiter sondern stellen ein virtuelles Gerät dar, dessen Übertragungsrate an die CPU gekoppelt ist. Die Bandbreiten sind dadurch sehr hoch und die Latenzen gering.
Um die Latenzen und die damit verbundenen unterschiedlichen Auftrittszeitpunkte von Anfragen zu berücksichten, sowie eine möglichst effiziente Abarbeitung der selbiger zu ermöglichen wird eien paralelle Algorithmusstruktur verwendet.

3.7.2 Interner Aufbau

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 3.5 zu entnehmen.

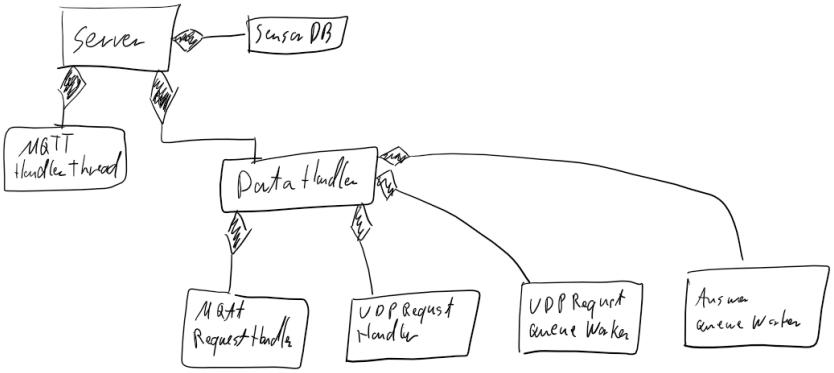

Abbildung 3.5: UML Digaramm Server

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 Beispielen erläutert. Für das erste Beispiel wird Abbildung 3.6 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 Grafik grün, Schnittstellen zur Library per UDP, sind blau markiert.
Erreicht ein MQTT Request den Server wird 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 derTyp des Requests bestimmt. Handelt 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.

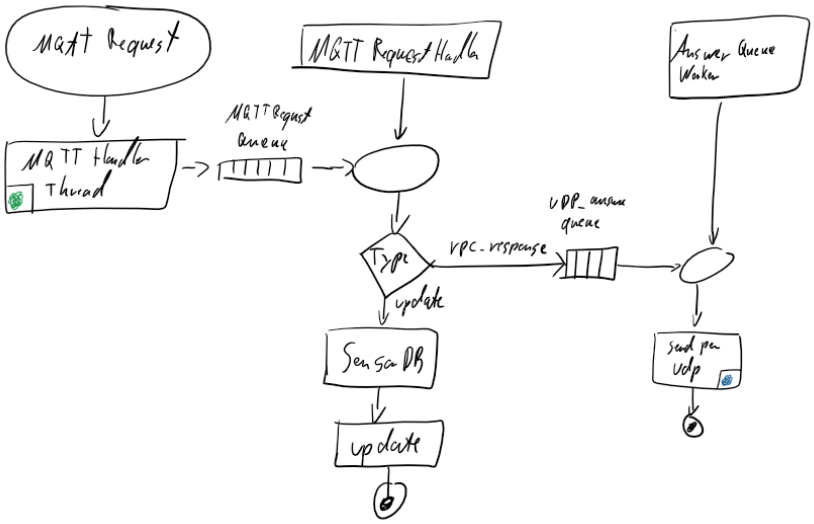

Abbildung 3.6: Ablaufdiagramm MQTT Request

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 3.7 dargestellt. Wie auch schon in der letzten Grafik sind alle Schnittstellen zum Smartphone grün und alle Schnittstellen zur Library blau markiert. Erreicht ein UDP 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 ineine 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 MQTTHandlerThread 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.

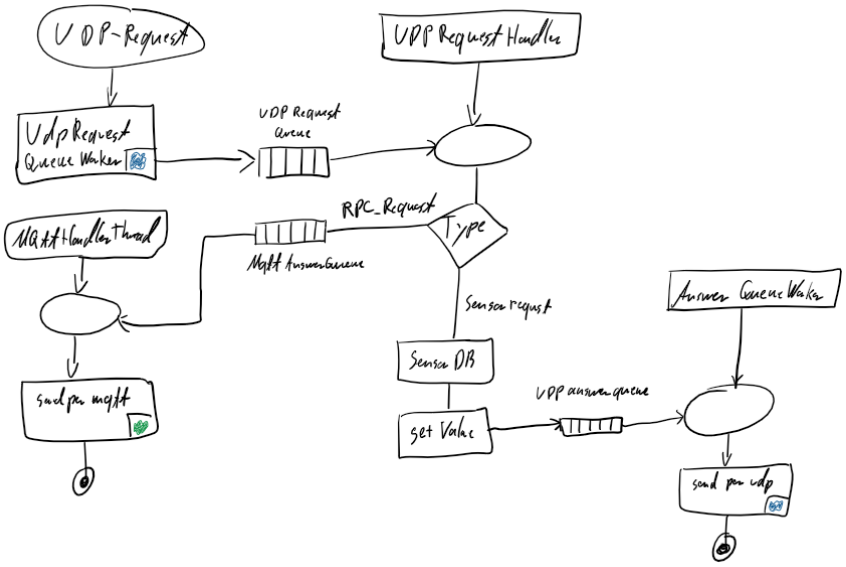

Abbildung 3.7: Ablaufdiagramm UDP Request

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.

Kapitel 4

Fazit

Kapitel 5

Quellen

Literaturverzeichnis

Online-Quellen