Smartbit

Marius Cerwenetz

Betreuung: Prof. Dr. Peter Barth, Prof. Dr. Jens-Matthias Bohli

Institut für Softwaretechnik und Datenkommunikation

2022–07–14

Agenda

Übersicht

Schwierigkeiten beim Programmierenlernen

- Semantik
- Grundkonzepte
- Theoretische Übungsaufgaben

Microcontroller als Alternative

- Interaktivität durch Sensoren
- Ausgabemöglichkeiten (LEDs, Piepser, Aktoren)
- Ausprobieren

Nachteile

- Beschaffung
- Peripherie

Smartphones für interaktive Lernaufgaben

- Hohe Verfügbarkeit
- Keine Verdrahtungsfehler
- Unabhängige Spannungsversorgung
- Zahlreiche Sensoren bereits integriert
- Drahtlose Verbindungstechnologien (WLAN, Bluetooth, UMTS/LTE)

Technische Hürden

Voraussetzungen

- Anbindung in Programmierumgebungen
- Smartphone-App

Lösungsansatz

Software-Lösung zur Kommunikation mit dem Smartphone

Übersicht

Vorgehen

- Beispielaufgaben für µC und Smartphones
- Anforderungen ableiten

Beispielaufgaben

- Disco
- Würfeln
- Diebstahl-Alarm
- Dreh-Zähler

Anforderungen

- UI-Elemente der Beispielaufgaben
- Sensordatenübermittlung
- Geringe Latenzen / Hohe Interaktivität

Übersicht

Aufbau der Smartbit-Lösung

Bibliothek

Kontrollanwendung

Android-App

Bibliothek

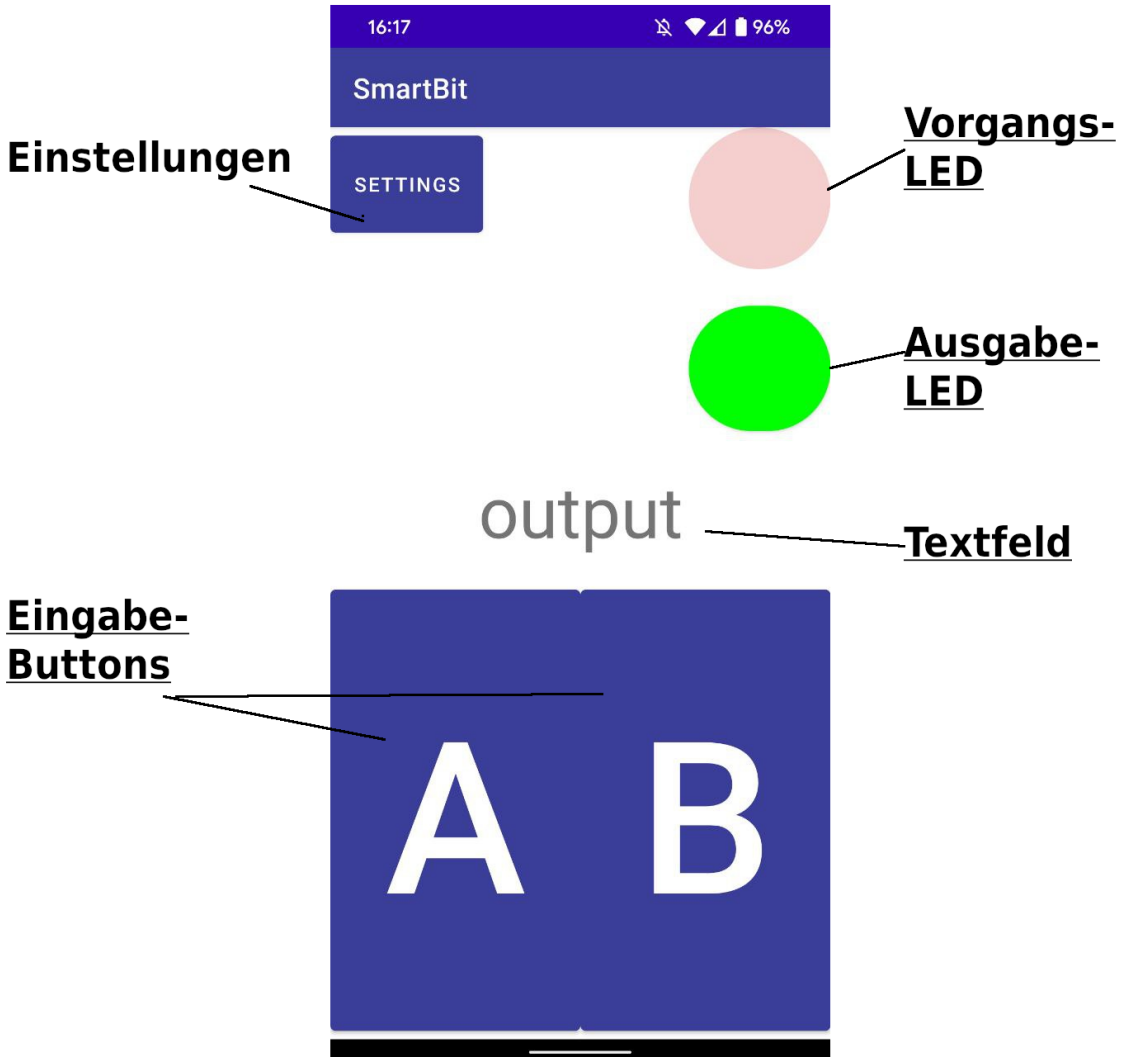
Funktionsaufrufe für Sensorwerte und Ausgaben.

Einbindbar in:

- C
- Java
- Python

```
1 from  smartbit import Phone
2 p = Phone()
3 accel = p.get_x_accelo()
4 p.write_text("hallo")
5
```

Smartphone-App



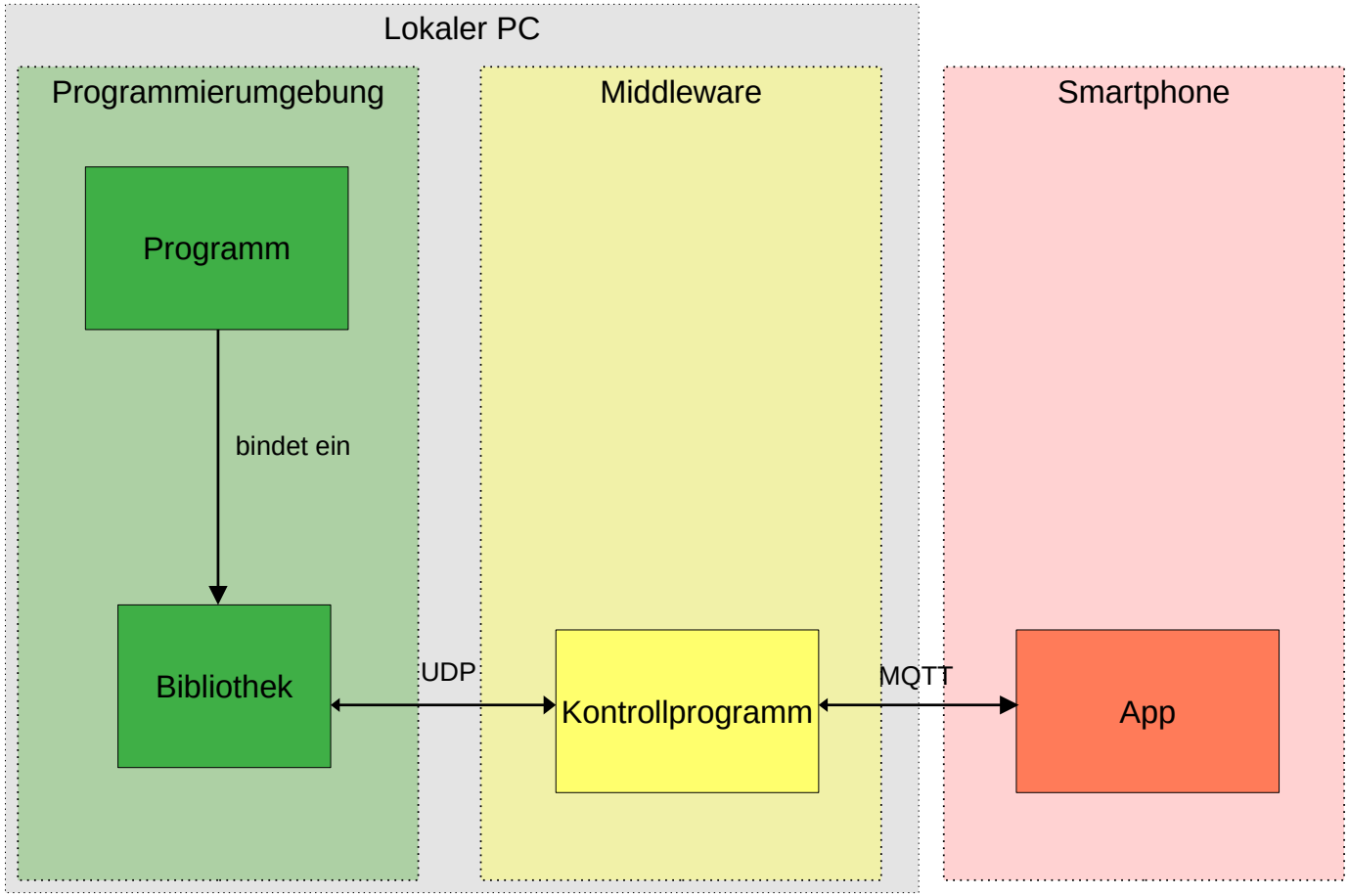
Kontrollanwendun

```
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
```

Übersicht

Übersicht der Gesamtarchitektur

Architektur



Nachrichtenformate

Requests

- update_request
- sensor_request
- rpc_request

Responses

- sensor_response
- rpc_response

Beispielnachricht - Neuer Sensorwert

```
{
    "type": "update_request",
    "sensor_type": "",
    "sensor_value": ""
}
```

Konfigurationsdateien

- protocol.json
- config.json
- Dezentral auf alle drei Komponenten verteilt

Bibliothek

Aufgaben

Funktionen für:

- Ausgabemöglichkeiten
- Sensorwertabfragen

Funktionsweise

- Stub-Funktionen
- Local-Loopback
- UDP, Port 5006
- CJSON und JSON for Java

Beispiel-Funktion: Vibration

```
1 def vibrate(self, time : int):
2 "vibrates phone for time miliseconds"
3 def _vibrate(self):
4     rpc_message = JsonMessagesWrapper.get_rpc_request(
    command="vibrate", value=str(time))
5     self._sendMessage(rpc_message)
6 threading.Thread(target=_vibrate, args=(self, )).start
    ()
7
```

Smartphone-App

Aufgaben

- Sensorwert-Übermittlung
- Ausgeben von Ausgabeanfragen

Funktionsweise

SensorEventListener

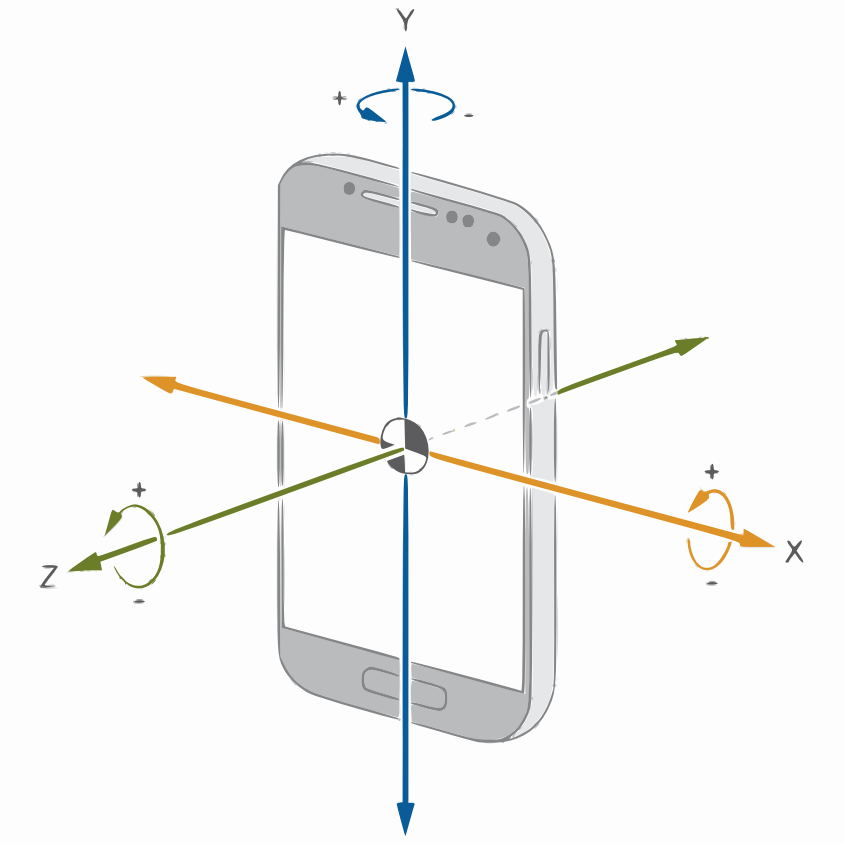
- Neue Sensormessdaten
- Übermittlung an Kontrollanwendung

MessageListener

- Warten auf Nachrichten
- Reaktion Ausgabe
- Ggf. RPC-Antwort

Sensoren

- Beschleunigungssensor
- Gyroskop
- Annäherungssensor



Kontrollanwendung

Aufgaben

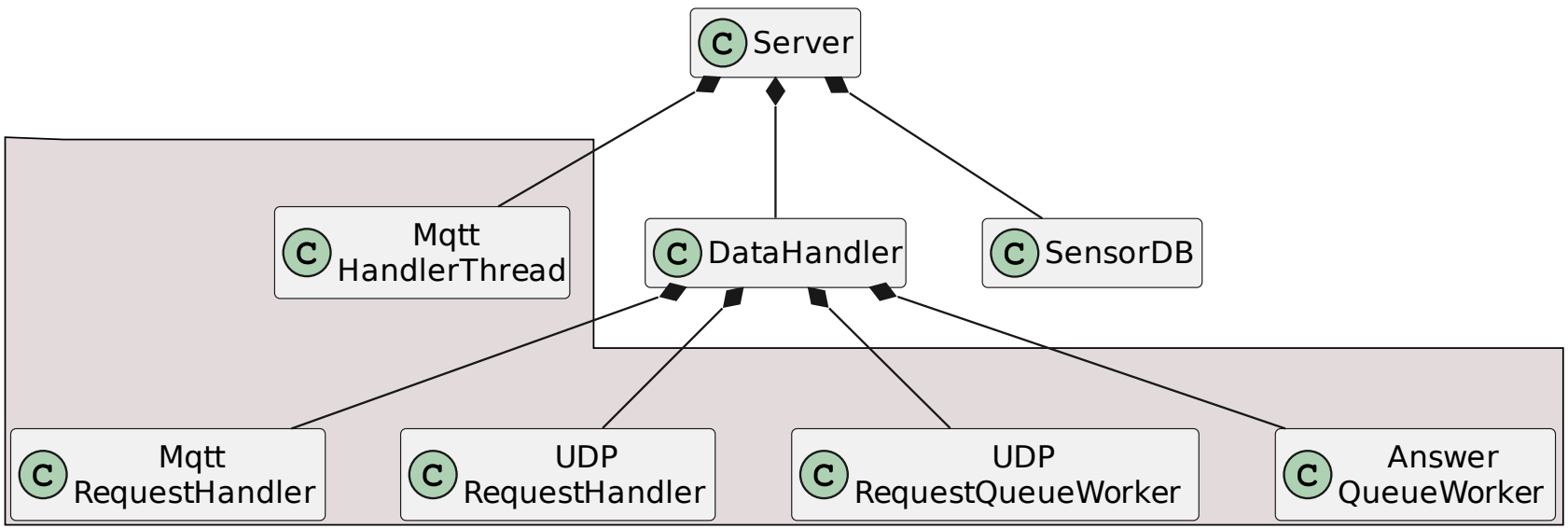
Aufgaben:

- Sensormesswerte annehmen und speichern
- Sensor-Anfragen beantworten
- RPC-Anfragen weiterleiten
- RPC-Antworten weiterleiten

Funktionsweise

- Mehrere Threads
- Kommunikation per MQTT und UDP
- UDP Port 5005
- Thread-Kommunikation durch Python-Queues

Aufbau der Kontrollanwendung



Übersicht

Latenzen

Kontext

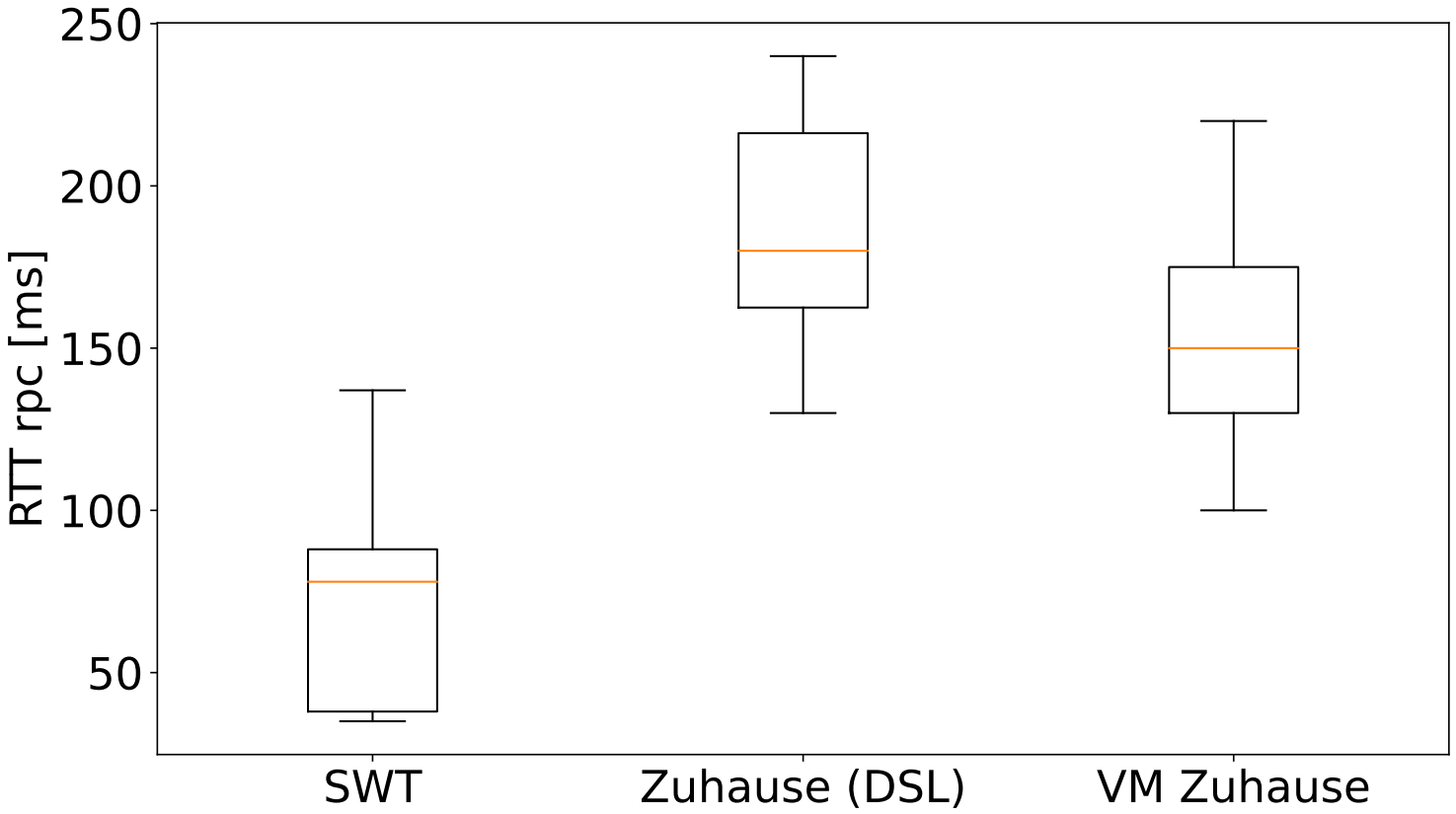
Szenarios

1. SWT-Labor
2. Zuhause
3. Per VM

Nebenbedingungen

- Umlaufzeit
- Unter Last
- Python

Latenzen



Sonstiges

- Portabilität
- Latenzverschleierung durch Caching
- Transparenter Ablauf durch Logging

Übersicht

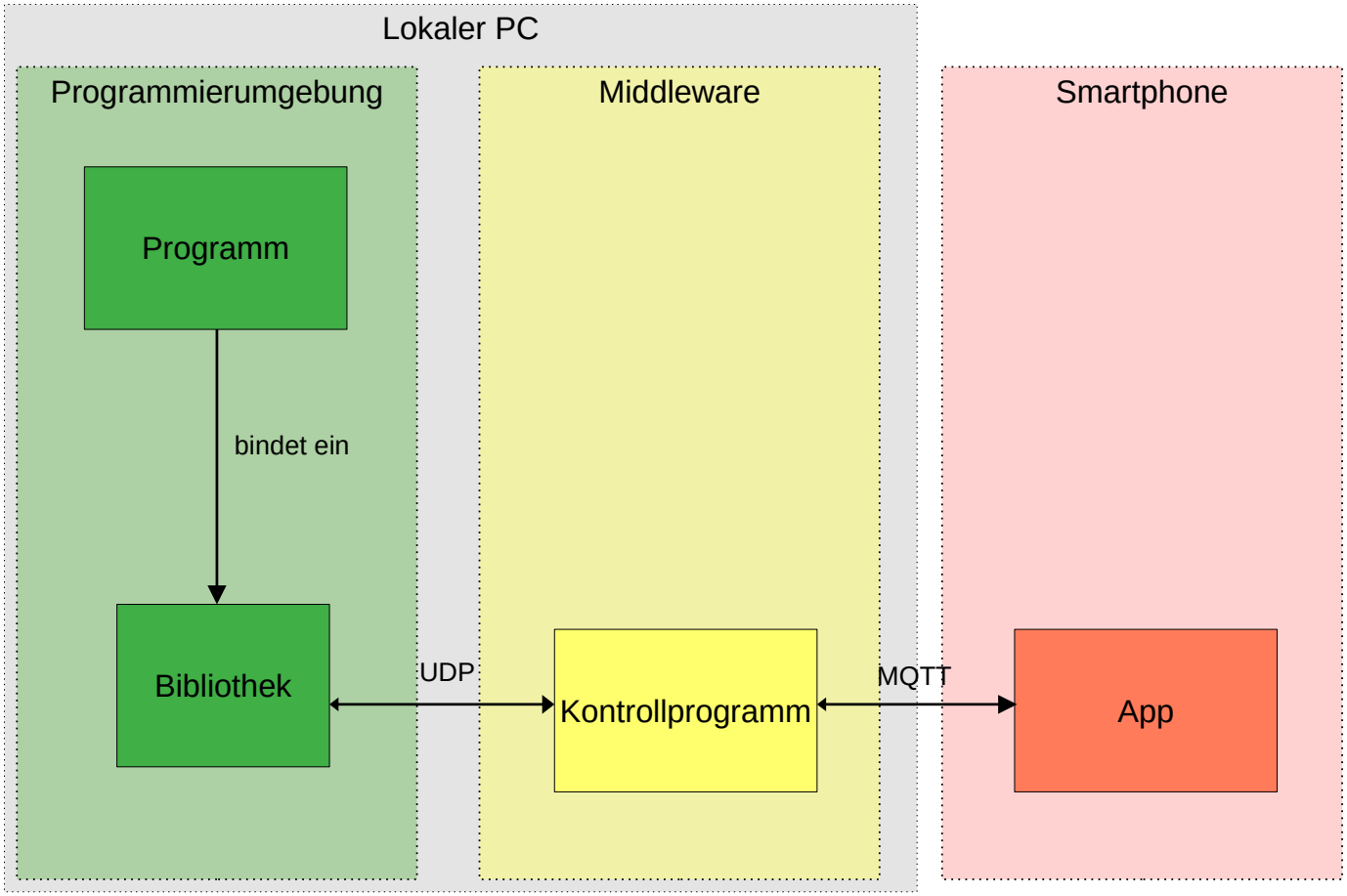
Fazit

- Alle Anforderungen implementiert
- JSON-Parser
- Latenzzeiten tolerabel
- Objekt-Orientierung
- Verfügbarkeit

Ausblick

- QR-Code für Konfigurationen
- Mehrere Sensorwerte zwischenspeichern
- Burst-Mode / Selektive Sensormessung
- Group-Sessions

Ende


Lokaler PC
Programmierumgebung
Middleware
Smartphone
Programm
bindet ein
Bibliothek
UDP
Kontrollprogramm
MQTT
App

Demo