FHDW Hannover

Platform-App for IoT-Devices

– AI-based Configuration and Data Representation –

Spezifikation 0v1

Jan Beckmann **Volkhard Klinger**

Contents

1 Introduction

The objective is to develop an app based on a platform concept that makes it possible to control and monitor several sensors, collect and evaluate the data. A wide variety of sensors and scenarios are conceivable, which can be configured and used with the help of an AI-based system. The type and number of sensors is not important; a relationship between the various sensors can be established with the help of model identification. The app is the link and gateway in a three-component architecture between sensors and server systems.

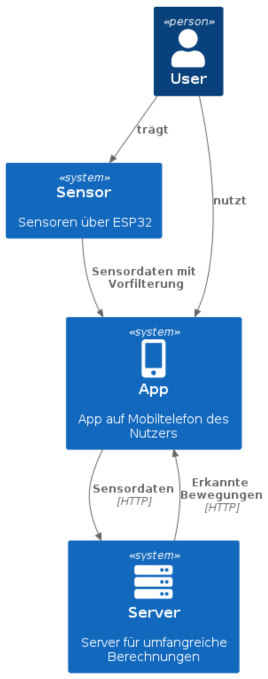

Figure 1.1: System-Overview

2 System Components

2.1 Functionality

2.2 IoT-System

The Internet of Things (IoT)-systems, hereafter called SmartBox, consists of an ESP32 [1] and application-specific sensors; for example a micro-electro-mechanical systems (MEMS)-sensor with 9 axes (acceleration (x, y, z), gyroscope (x , y, z) and magnetometer (x, y, z)). This ESP32 platform is the workhorse for all applications and can be adapted to the specific application using a variety of sensors.

2.3 SmartDevice

The SmartDevice is any Tablet or Mobile phone providing a number of essential feaures:

- Graphical User Interface (GUI)
  Screen for visualization and I/O-functions, like keyboard or speaker, are integrated. Based on the screen data isualization and event triggering i possible. Furthermore the system configuration can be realized using the GUI.

- Connectivity
  The SmartDevice provides proivdes an integration of th IoT-devices into communication services via Bluetooth Low Energy (BLE), wireless local area network according IEEE 802.11 (WiFi) or Global System for Mobile Communication (GSM), i.e. cellular-secured Internet.

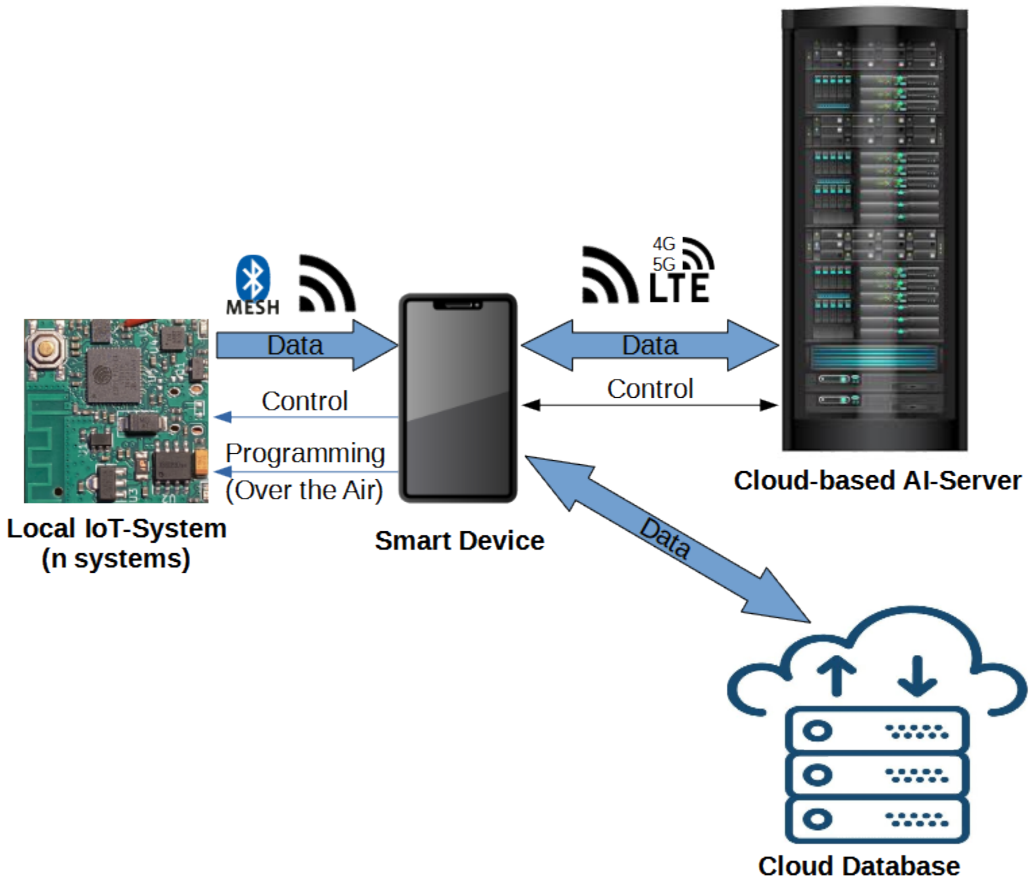

Figure 2.1: System-Architecture

- Messaging Functions
  tbd !!!

Thanks to the platform architecture, the various modules span their own network (via BLE) or wireless local network in accordance with IEEE 802.11 (WiFi)) and connect to a gateway over a greater range. This gateway, i.e. a SmartDevice, enables the system to be integrated via WiFi or GSM, i.e. cellular-secured Internet. To realize a platform-based system architecture, which enables mobile operation that is only which is only person-bound and not location-bound, the following features are essential:

This means that the SmartDevice is only required for certain displays and events. A more advanced system mode, which allows the raw measurement data to be transferred at certain times from the local system to a cloud architecture and thus also to a server for evaluation, has already been presented in [2]. In gait monitoring and posture assessment in particular, the main task is to compare trajectories or curves in order to identify specific movements. In the following subsections, we present this relationship in more detail.

The SmartDevice is connected for configuration and data modification, e.g. to download and display all the raw data stored on the ESP32 or to configure the platform. The other levels are not the focus here, a brief description will suffice: The cloud database is the sink for all data from all local IoT systems and the source for all data mining and data-based fusion and identification operations. This architecture integrates all IoT systems and supports various different modes and the corresponding scenarios [2]. In all applications, we try to improve the Plug-and-Play (PnP) character by achieving as little system complexity as possible for the measurement and at the same time as much measurement and operating convenience as possible and a low cost approach. This means that the SmartDevice is only required for certain displays and events. A more advanced system mode, which allows the raw measurement data to be transf

2.4 Database

InfluxDB is an open source database specially developed for time series data, which is particularly suitable for storing and analyzing sensor data. Time series data consists of continuously recorded measured values in chronological order (e.g. temperature or humidity in a room). InfluxDB 2.x offers a scalable platform for collecting, storing and analyzing this data. In InfluxDB, the basic organization of the data is based on four central elements: Measurements, Tags, Fields and Timestamp. A measurement represents a logical grouping of time series data, comparable to a table in relational databases. Within a measurement, all data points contain the same tags, but different values in the fields. [13] Tags are key-value pairs that contain metadata such as sensor or station IDs. These rarely change and are primarily used to identify and categorize data points, for example by location or device. As tags are indexed, they enable fast filtering for queries. In contrast, the fields contain the actual measured values, such as temperature or humidity. These values change over time and are not indexed, which means that although they are optimized for storage and write optimization, they should not be used for high-performance filtering.

2.5 AI-Server

The Artificial Intelligence (AI) solution enables information to be extracted from many documents in the shortest possible time without the need for them to be read by humans. It realizes the PnP-function and helps by using and configuration of the system. The AI learns which data is usually requested by users and simplifies the presentation by supporting the selection of display options. Furthermore, the AI helps with calibration by independently prompting the user to perform certain movements, which enables simple configuration of the overall system based on multiple sensors.

3 Introduction

Figure 1: System level architecture

3.1 Functionality

The functionality of the app again breaks down into three parts:

- Management of the sensors
- Transmission and visualization of the data
- Evaluation of the data on the ESP itself and within the cell phone
- Provide an AI-based PnP.

Table 3.1: Requirements

ID	Task	Definition
R1	Initialization	The app makes it possible to establish an initial connection to the sensor and to provision it. In particular, this can also be used for WLAN configuration (e.g. via SoftAP or BLE, SmartConfig or other provisioning mechanisms supported by the ESP32). Once a connection has been established, the app enables a basic teach-in mechanism in addition to the mechanisms outlined below. In addition, the app allows a basic calibration of the sensor.

Continued on next page ⟶

Table 3.1 – continued from previous page

ID	Task	Definition
R2	Data Transmission	The app allows to transfer and visualize the data of the sensors. In doing so, it is possible to transmit and visualize the data in real time, but also to collect the data and transmit it at a later time. In this way, the app also can transfer the data collected in this way to a server, where further processing or storage of the data can take place. On the visualization side, it is possible to visualize the data in different representations. For example, it may be possible to display the data in a diagram, but also in a 3D view. The visualization can also be extended to several sensors, so that the data of the sensors can be combined in a diagram or a 3D view.
R3	Data Evaluation	The goal should be that a basic evaluation of the data, in particular a motion detection directly on the sensor or ESP, or at least on the cell phone can take place. To evaluate the data, it may be necessary for the app to learn the user's motion geometry. learn. For this purpose, it is necessary that the user requests the app to learn the motion geometry. learn. The app then performs a series of motions for the user to track. In the process the app visualizes the movements in a 3D view so that the user can better understand the movements. can better understand the movements. The app can also divide the movements into several steps, so that the user can follow the movements in several passes. can follow the movements. From this motion data, the app can then derive a motion geometry in the form of a model, which can then be used for evaluation. which can then be used to analyze the data. If individual data is still missing data is still missing, the process can be triggered again by teaching in new data (cycle). Evaluation The evaluation of the data should be trajectory-oriented. This means that the sensor and app trajectory from the acquired data and compare it with the stored trajectories of the model. trajectories of the model

Continued on next page ⟶

Table 3.1 – continued from previous page

ID	Task	Definition
R4	PnP by AI	The app allows to transfer and visualize the data of the sensors. In doing so, it is possible to transmit and visualize the data in real time, but also to collect the data and transmit it at a later time. In this way, the app also can transfer the data collected in this way to a server, where further processing or storage of the data can take place. On the visualization side, it is possible to visualize the data in different representations. For example, it may be possible to display the data in a diagram, but also in a 3D view. The visualization can also be extended to several sensors, so that the data of the sensors can be combined in a diagram or a 3D view.

3.2 Technical implementation

In a first phase, the app shall be developed for Android only. The development is to be done as a native app (Kotlin / Java), so that the evaluation on the cell phone and the visualization the visualization of the data on the mobile phone. In the following, the functionality of the individual modules will first be outlined with reference to the above before the technical implementation is described.

3.2.1 Architecture

Based on the three-part functionality outlined above, the technical level results in various modules that form the technical framework of the app. They are shown in the Figure 3.1.

3.2.1.1 Module 1: Communication with sensor(s)

This module includes all necessary functionality for provisioning and subsequent Data exchange with one or more sensors. In particular, this includes:

- Basic data exchange (via WiFi + MQTT/HTTP or BL(E))
- Provisioning (via SoftAP or BLE, cf. above)
- Receiving data from the sensor

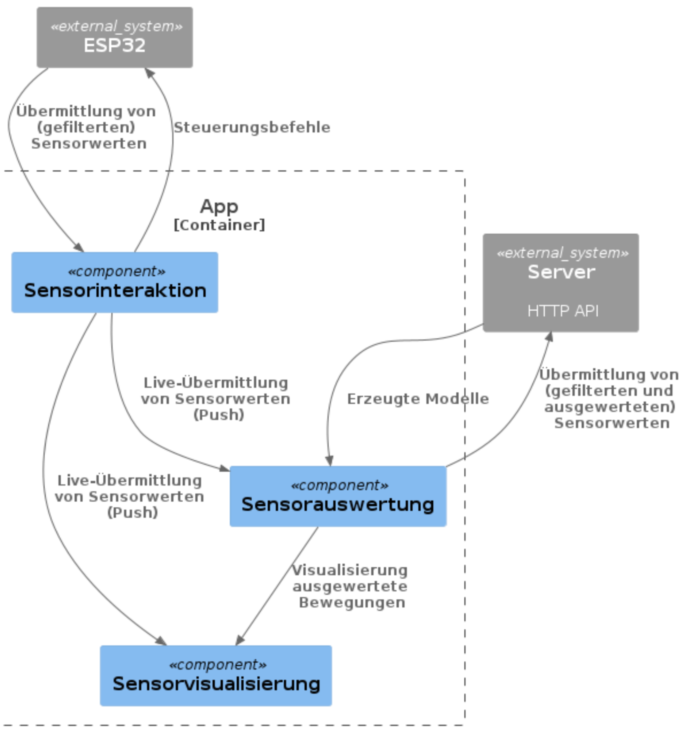
«external_system»
ESP32
Übermittlung von (gefilterten) Sensorwerten
Steuerungsbefehle
App
[Container]
«component»
Sensorinteraktion
«external_system»
Server
HTTP API
Live-Übermittlung von Sensorwerten (Push)
Erzeugte Modelle
Übermittlung von (gefilterten und ausgewerteten) Sensorwerten
Live-Übermittlung von Sensorwerten (Push)
«component»
Sensorauswertung
Visualisierung ausgewertete Bewegungen
«component»
Sensorvisualisierung

Figure 3.1: Component-Architecture

- Transmitting control commands (e.g. configuration motion parameters, calibration, etc.)

With regard to the trials made so far regarding the use of BL(E)-Mesh, it offers data exchange, it would be a good idea to continue to rely on a WLAN-based solution.1 could also be evaluated whether a BLE-based solution makes sense, at least for provisioning.

The exact network topology is then questionable, in particular 1. which device functions as the WiFi access point (AP) and 2. whether, how and when the sensors communicate directly with the app (pull/push via HTTP) or whether a third party acts as an MQTT broker (although this function can also be performed by the app).

Details of the initial calibration will also be discussed again in the following.

In any case, existing components/code components can be used for both the app and the ESP32.

3.2.1.2 Network connection

The use of an additional stationary device as a mandatory requirement (as was sometimes the case in the previous the Raspberry PI) is difficult in that it makes any mobility outside the WLAN range. It also does not seem to make much sense to restrict the task of the WiFi AP to the ESP32 (SoftAP), because it can be assumed that a considerable reduction in battery life can be expected, especially in the case of permanent operation. In addition to these considerations, there are still several possibilities: Based on the power considerations in permanent operation, it would seem logical to only use the WLAN network of the ESP32 only temporarily3. This could be triggered by an interval-based mechanism (every x minutes / hours) (Pro: very easy to implement, Con: event identification) Nevertheless, it should be obvious from the abundance of the following explanations that a purely BLE-based solution should be preferred. However, this is of course the basic problem with any kind of WLAN use. Such a temporary coupling naturally makes it impossible to perform a permanent live evaluation on the smartphone.

Instead, the data on the ESP must at least be pre-processed and temporarily stored (SD card). This cannot be solved in any other way, since permanent background operation of the app is not possible for technical reasons. Here, it is interesting to introduce modes, describing different levels of communication

with the smartphone) on the other hand also a trigger through the app (for example via BLE, if possible) or via a hardware switch. Furthermore, this solution leads to the fact that the ESP - and in the smartphone / the app as long as it is connected to the AP - do not have access to the internet. access to the Internet. This may be a good thing for security reasons, but it is problematic in that the app cannot problematic in that the app itself does not have the ability to transmit data at that moment. transmit data. It is also not possible to simply synchronize the ESP with an Internet time server via NTP. with an internet time server. However, alternatives could be found for both.4 Conversely, the smartphone can also map the access point. Here too, for reasons of battery capacity, it will make to activate the AP only temporarily, but this is much easier to implement there. easier to implement. On the other hand, this does not solve the problem that here too the ESP would either have to permanently search for this AP or it has to be triggered again in one of the ways described above. ways described above to establish a connection. Compared to the just described solution described above, this solution would have the advantage that the smartphone could still offer Internet access at the same time.

For the moment, it is best to ignore this question as far as possible. For the time of basic development it will not be necessary to consider limited battery capacities. What is important is that an AP is available; this can be realized by using a smartphone, a separate AP (e.g. Raspberry PI) or even a regular WLAN router, because none of these three variants requires a larger implementation effort, but can be handled in exactly the same way from the point of view of the app <-> ESP32.

Nevertheless, it makes sense to consider in the context of the provisioning mechanism whether here components can also be used later for repeated connection establishment.

3.2.1.3 Network protocol

Similar questions arise in connection with the question of how exactly the app and the ESP32, given the WLAN network outlined above, communicate with each other. Here, too, several options are conceivable:

- via MQTT
  The ESP regularly sends data to an MQTT broker if a network connection exists. an MQTT broker. This MQTT broker could be either the app itself (useful) or a third party server (not useful). third server (not useful). This way has often been used so far.

- via TCP (or corresponding layer-7 protocol, e.g. HTTP) in push mode
  The app starts a corresponding TCP server (web server) on the smartphone and the ESP32 tries to connect to it, tries to connect to it. If this is successful, the ESP32 sends data to the app. For example, the app could transmit the data only after it has disconnected from the ESP. It is also conceivable to use a connection that exists in parallel, if supported by the operating system. Time synchronization could instead take place in direct exchange with the app (see below).

- via TCP (or corresponding layer-7 protocol, e.g. HTTP) in pull mode
  Here the ESP starts a corresponding TCP server and the app tries to connect to it. If successful, the app requests data from the ESP32. The first option is the most flexible in many respects: Not only does it enable true bidirectionality without additional effort, it also offers easy

expandability with regard to multi-sensor operation. to multi-sensor operation.

3.2.1.4 Provisioning and configuration

Provisioning is understood here to mean the following:

- Discovery of the ESP32 in the network / environment
- Configuration of the ESP32 with the necessary information to connect to the WLAN network.

The provisioning can then be followed by further configuration, e.g.

- Initial calibration (by performing certain action, e.g. level position of the sensor, no movement, etc.)
- Transmission of relevant configuration parameters for filters performed on the ESP, sampling rate, etc.

For the configuration steps, the communication approaches discussed above can be used. For the provisioning it was already referred at the beginning to the multiplicity of the methods already natively offered by ESP. However, some of them are already ruled out:

- Wi-Fi Easy-Connect (Device Provisioning Protocol, DDP)
  Requires a display on which a QR code can be displayed by the ESP. can be displayed by the ESP. Such a display is not available here.
- SmartConfig (also ESP-Touch)
  It is intended precisely for headless devices and would work by the app sends the access data for the WLAN network to the still unprovisioned ESP.

In any case, this is much more elegant than the also available option of performing provisioning via the SoftAP method, where the ESP itself provides a SoftAP together with an HTTP server. Because that requires changing the network on the smartphone to this new network, in order to be able to perform the actual provisioning. Most interesting, however, as already mentioned above, is to perform the provisioning via BLE (Gatt). For this, ESP provides examples both for a corresponding app integration and for the

ESP itself. (both ESP-IDF and Arduino). This could be a harbinger for a complete handling of the communication via Bluetooth. communication via Bluetooth.

3.2.1.5 Module 2: Visualization

This module builds on the first module and enables the visualization of the data. Thereby it is necessary to visualize the data in real time, but also to collect the data and visualize it offline. The visualization can be extended to several sensors, so that the data of the sensors can be displayed in a diagram or a 3D view.

In addition to a series of sensor evaluations, detected movements (see below) can be displayed and and reconstructed. In this way, it is possible to understand in particular where any possible weak points in the recognition. For a first "puncture" it will be sufficient, if the data would be displayed for example in the Matlab-box or using the game engine Unity.

In a next step the application of the data to at least an abstract-humanoid model would be desirable. Technically, it would be desirable if the existing Unity artifacts could be used for this purpose. (Unity also offers the possibility to develop for mobile platforms). Alternatively the native OpenGL-ES-APIs of Android also represent a fallback possibility.

3.2.1.6 Module 3: Data analysis

This module also builds on the data of the first module. It represents the core of the application represents: On the basis of the measured values supplied by the ESP32 and its sensors, a sensor fusion takes place with the goal of recognizing and evaluating executed movements. Here, for the beginning, one can refer to the results already achieved in various places (Unity, Matlab, etc.). (e.g. Peucker algorithm) can be used. To clarify is thereby quite substantially the interfaces between the three involved systems (what happens on the ESP, what in the App, what on a possible server?). The consideration takes place along two clearly defined axes: The more downstream the data is processed, the more computing power is available for it. The earlier the data is processed, the less data has to be transferred (and stored). be transferred (and stored). A precise breakdown will require further consideration and will therefore be omitted here. Nevertheless, a few basic principles should be suggested at this time:

- It makes sense to map most of the logic on the smartphone (= in the app). On the one hand, this avoids having to rely on a third component (and thus on the necessary connectivity), but on the other hand, it also offers an alternative to 8On the problematic nature of the term "real-time", cf. the explanations above on only intermittent coupling. What is meant here obviously means a live evaluation with an existing, active connection between the app and ESP. In all other cases, only a replay function based on the data stored on the SD card will remain.

  the ESP significantly higher computing power. In addition, this also allows the data from several ESPs to be combined.

- Preprocessing should also take place on the ESP itself in order to avoid having to transfer every sensor value. not have to transmit every sensor value.

- On a server component, processing should take place in particular of those arithmetical operations which have to fall back on a comprehensive historical data stock (time series database). have to fall back. Real-time processing is suitable if a network connection exists and the computing power of the smartphone is not (or no longer) sufficient. of the smartphone is not (or no longer) sufficient.

3.2.1.7 Module 4: Model generation

This module is complementary to the previous one: It should make it possible to improve the accuracy of recognition by collecting additional data based on various types of calibrations and teach-in processes by the user. by the user. In view of the considerable effort required to implement the three aforementioned modules, it is a good idea modules, it is advisable to postpone the implementation here for the moment and to fall back on manually to fall back on manually generated models.

3.2.1.8 Module 5: AI-based Plug-and-Play

This module enables the integration of AI to improve PnP. There are different tasks:

- Carrying out the system initialization, identification of the IoT-devices and their localization in dialog.

- Assists the user with the various events and explains what to do. Depending on the critical level, this is done in an appropriate manner (information, reminder, independent contact with medical staff).

3.2.2 Development

As mentioned above, it is proposed to develop the app as a native Android app in a first step. This offers a number of advantages over a hybrid app:

- It is easier to access system-APIs directly, without having to resort to corresponding (often only conditionally (often poorly maintained) plugins. This concerns in particular the communication with the ESP outside of TCP (e.g. BLE).

- As far as sample code from Espressif can be accessed in some places, this is implemented as a native Android app. This facilitates the adoption into the own app. Compared to an iOS app as a first step, the lower entry hurdle for developers is particularly worth mentioning. (no Mac required for development, no iPhone required for development). Nevertheless, it is also clear that the Android emulator can be used if no real smartphone is available for development, especially in terms of Bluetooth. development, especially in terms of Bluetooth communication with the ESP. will come to its limits. The main programming languages that can be used are Kotlin and Java. Java enjoys meanwhile increasing popularity and could be therefore to be preferred.

Ultimately, the Google's own statement: "used by over 60% of professional Android developers."' depends on the prior knowledge of the respective developers.

3.2.2.1 Development planning

Milestone planning, etc.

List of Figures

List of Tables

Bibliography

[1] Espressif. Esp32 series, datasheet. Technical report, Espressif, 2019.

[2] Volkhard Klinger and Sebastian Bohlmann. Application-Based IoT-System for Pandemic Prevention Based on Platform-Approach. In Agostino Bruzzone, Marco Frascio, Vera Novak, and Francesco Longo Eds., editors, *9th International Workshop on Innovative Simulation for Health Care (IWISH 2020)*, 9 2020.