Hochschule fuer Telekommunikation
Leipzig

Hochschule für Telekommunikation Leipzig
University of Applied Sciences

Technische Dokumentation

Entwicklung eines Smart Home Systems mit Steuerung durch Android Apps

Autor:
Alexander Hatzold & Patrick Sudhaus

Betreuer:
Prof. Dr. Ulf Schemmert

11. Mai 2018

Diese Arbeit wurde vorgelegt von
Alexander Hatzold, Patrick Sudhaus

Hochschule:	Hochschule für Telekommunikation Leipzig
Matrikelnummer:	158105, 158116
Studienbereich:	Angewandte Informatik
Betreuer:	Prof. Dr. Ulf Schemmert

Inhaltsverzeichnis

1 Einleitung

Im Zuge der Digitalisierung werden alle Bereiche des täglichen Lebens auf neue, digitale Systeme umgestellt. So wird beispielsweise im Bereich des Wohnens auf Technologien gesetzt, die über das Internet steuerbar sind. Damit einhergehend wird oft der Begriff des Internet of Things (IoT) genannt, zu dem auch der Bereich Smart Home gezählt werden kann.
Diese Dokumentation beschäftigt sich mit dem Entwurf und der Umsetzung einer Smart Home- Architektur und der Implementierung zweier Apps, die für die Steuerung der Komponenten in diesem Netzwerk genutzt werden sollen. Die erste App soll eine LED-Lichterkette in verschiedenen Modi ansteuern können. Die zweite Anwendung soll eine Multitimer-App darstellen, die als wichtiger Helfer im Haushalt und insbesondere beim Kochen agiert. Dabei ist diese Timer-App ebensfalls an das Smart Home-Netzwerk angebunden und kann einzelne Komponenten des Smart Homes steuern.
Das Ergebnis dieser Arbeit ist ein prototypisch implementiertes Netzwerk, in das die beiden Apps eingepflegt werden und die IoT-Geräte angesteuert werden können.

2 Grundlagen

Um die weiteren Betrachtungen nachvollziehen zu können, werden an dieser Stelle die nötigen Grundlagen dargestellt.

2.1 Message-Broker und MQTT

Message-Broker gehören zu den nachrichtenorientieren Middlewares und sind Server-Anwendungen, die Nachrichten empfangen und weiterleiten. Dabei kann eine Umwandlung des Protokolls des Senders in ein für den Empfänger verständliches Protokoll geschehen.
Message Queue Telemetry Transport (MQTT) ist ein Publish/Subscribe Messaging Transport Protokoll, das von einem Message Broker verwendet werden kann. Es ist leichtgewichtig, einfach aufgebaut, einfach zu implementieren und steht zur freien Verfügung. Damit soll gewährleistet werden, dass es vor allem für IoT-Anwendungen und für Machine-to-Machine (M2M)-Anwendungen geeignet ist, da hier häufig eine geringe Netzwerkbandbreite zur Verfügung steht. Als zugrundeliegendes Protokoll wird TCP/IP verwendet.
Das Publish/Subscribe-Verfahren funktioniert folgendermaßen:

Publisher Dieser Teilnehmer ist die sendende Instanz im Netzwerk. Jede Nachricht wird einer bestimmten Kategorie, auch Topic genannt, zugeordnet. Der Sender weiß dabei nicht, wer die Nachricht empfangen wird.

Subscriber Der Subscriber oder Empfänger empfängt alle Nachrichten, zu deren Topic er sich beim Message-Broker registriert hat. Dieser Teilnehmer hat keine Kenntnis über die Herkunft der Nachricht.

Broker Diese Instanz ist der Vermittler zwischen Publisher und Subscriber. Er empfängt alle Nachrichten und stellt sie allen relevanten Empfänger zu.

Bei der Verwendung von MQTT gibt es drei Stufen des Quality of Service (QoS):

1. QoS Level 0: Dieses Level agiert nach dem Ansatz „At most once“, d.h. jede Nachricht wird maximal einmal zugestellt und sollte dies fehlschlagen, erfolgt keine erneute Zustellung. Dementsprechend können Nachrichten verloren gehen.

2. QoS Level 1: Die unter dem Ansatz „At least once“ bekannte Stufe stellt Nachrichten mindestens einmal zu, wobei mehrmalige Zustellung nicht ausgeschlossen ist.

3. QoS Level 2: Der „Exactly once“-Ansatz stellt jede Nachricht genau einmal zu.

Die Spezifikation von MQTT Version 3.1.1 ist unter [1] verfügbar.

2.2 Digitale LEDs

Digitale LEDs erlauben es mit nur einem Datenkanal mehrere LEDs unterschiedlich anzusteuern. Dies wird durch den Einsatz eines Chips für jede LED möglich, der nur die für ihn relevanten Daten extrahiert und anzeigt. Hierdurch ergibt sich nicht nur die Möglichkeit, manuell jede LED mit einer bestimmten Farbe anzusteuern, sondern auch Effekte wie Lauflicht oder Regenbogen sind umsetzbar.
Für dieses Projekt wurde sich für WS2812B 5050 SMD LEDs entschieden, da diese einfach über Libraries mit dem Mikrocontroller ansteuerbar und verhältnismäßig kostengünstig sind. Zum Ansteuern dieser LEDs kann die populäre Arduino Library „Adafruit Neopixels“ verwendet werden.

3 Anforderungen & Planung

3.1 Aufbau der Infrastruktur & Technische Details

Die Applikationen werden für das Betriebssystem Android in der Programmiersprache Java entwickelt, wobei auf API-Level 21 (Android 5.0) zurückgegriffen wird. Android Studio stellt dabei die Entwicklungsumgebung der Wahl dar.
Als Protokoll zum Austausch der Nachrichten wird das MQTT-Protokoll verwendet. Die zugrundeliegende MQTT-Software heißt *Mosquitto*.
Die zentrale Serverkomponente „Home Manager“, die für bestimmte Funktionen (siehe Abschnitt **??**) genutzt wird, wird ebenfalls in der Programmiersprache Java geschrieben. Dabei wird zum persistenten Speichern von Informationen die auf MySQL basierende Datenbank MariaDB verwendet.
Die Mikrocontroller, die für die Steuerung der LED-Lichterketten verantwortlich sind, basieren auf dem ESP8266 CPU. Der verwendete NodeMCU V3 koppelt diese Architektur mit einem WLAN-Chip und 13 GPIO Pins, wodurch der Mikrocontroller Nachrichten empfangen kann und Informationen für die Lichtsteuerung über GPIO Pins ausgeben kann. Das Programm auf diesen wird mithilfe der Arduino IDE in der Programmiersprache C geschrieben.
Die Infrastruktur ist in Abbildung 3.1 dargestellt. Die Android-Geräte mit den entsprechenden Apps werden im Folgenden als *Sender* bezeichnet, da sie überwiegend Befehle zum Message-Broker senden. Dieser wertet die Nachrichten aus und stellt die den entsprechenden Empfängern zu. Die Nachrichten werden über das jeweilige Topic identifiziert, d.h. jede LED besitzt ein eigenes Topic, sodass jede LED einzeln

angesteuert werden kann. Dabei folgen die Topics dem Schema *Kanal/Zimmer/-Gerät/Funktion*, sodass auch mehrere LED Streifen in einem Raum angesprochen werden können. Die Nachricht selbst enthält die Information, wie die LEDs leuchten sollen. Die Nachrichten werden von dem im WLAN-Netzwerk angemeldeten Mikrocontroller empfangen und verarbeitet. Diese im Folgenden als *Empfänger* bezeichneten Geräte führen die gewünschte Aktion aus, bspw. das Leuchten der LED Steifen in den gewünschten Farben.
Für die Notation der Nachrichten wurde sich für JSON entschieden.

Server-Komponente *Home Manager*

Die Nachricht wird außerdem noch vom sog. *Home Manager* empfangen, der für sich relevante Nachrichten herausfiltert und verarbeitet. Zu seinen Funktionalitäten gehört die Umsetzung einer Timer-Funktion. In einer der beiden Apps soll eine Funktion eingebaut werden, um Timer einzustellen, d.h. dass zu bestimmten Uhrzeiten bestimmte LEDs automatisch angesteuert werden. So kann erreicht werden, dass man bspw. durch Hellwerden am Morgen geweckt wird oder abends das Licht bereits angeschaltet ist, wenn man nach Hause kommt. Dies soll durch Eintragungen der Zeitpunkte in die Datenbank MariaDB geschehen, welche in bestimmten Intervallen gelesen wird und die LEDs durch die HomeManager-Komponente aktiviert werden. Aufgrund der Komplexität und den Rahmenbedingungen dieser Arbeit im Rahmen des Moduls „Mobile Applikationen“ ist diese Funktion allerdings optional.

3.2 Funktionale Anforderungen

Die Infrastruktur der SmartHome-Anwendung soll dazu in der Lage sein, die folgenden Anwendungsfälle zu unterstützen:

- Es ist möglich, sich mit jedem möglichen Gerät im Netzwerk (im Speziellen dem WLAN-Netzwerk) anzumelden und bestimmten Topics zuzuhören und Nachrichten zu diesen zu senden. So kann später auch eine Anwendung für Windows-PCs oder eine Webanwendung entwickelt werden. Diese Arbeit beschäftigt sich nur mit der Implementierung zweier Android-Apps als Frontend.

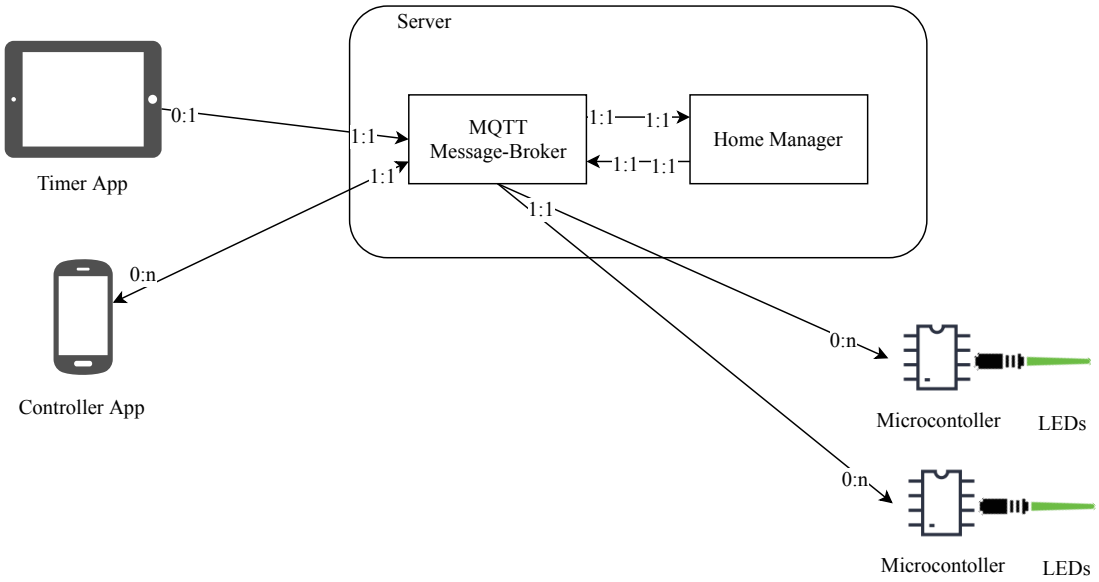

Abbildung 3.1: Architektur der Smart-Home Infrastruktur

- Durch die Anmeldung der empfangenden Geräte, wie in dieser Arbeit die verwendeten LED-Lichterkette, kann prinzipiell jedes angemeldete Gerät angesteuert werden. Ein anderer denkbarer Ansatz wäre beispielsweise die Ansteuerung von Jalousinen über die beschriebene Infrastruktur.
- Die Server-Komponente auf dem Raspberry Pi übernimmt dabei besondere Aufgaben, die entweder zentral verwaltet werden müssen, wie bpsw. Timer-Funktionalität (siehe **??**), oder die nicht auf den Mikrocontrollern realisiert werden können (siehe 4.3).

3.3 Nicht-funktionale Anforderungen

Neben den funktionalen Anforderungen müssen einige nicht-funktionale Anforderungen erfüllt werden, um die Erstellung und Weiterentwicklung der Lösung zu gewährleisten. Dazu zählen insbesondere

- die Erstellung eines sauberen Programmcodes, sodass auch spätere Entwickler weiterhin am Projekt arbeiten können.
- die Performanz der Lösung, damit diese frustfrei verwendet werden kann.
- die Absturzsicherheit der verschiedenen Programme, sodass keine manuellen Neustarts oder dergleichen getätigt werden müssen.

- die Übersichtlichkeit der Apps, damit der Nutzer diese möglichst intuitiv bedienen kann.
- die Skalierbarkeit, sodass spätere Komponenten möglichst einfach hinzugefügt werden können.

4 Implementierung

4.1 App zur LED-Steuerung

4.1.1 Aufbau der Anwendung

Die Anwendung soll auf der Startseite die Ansicht eines Colorpickers darstellen, sodass sofort nach Start der App eine Farbe ausgewählt werden kann. Diese Farbe wird sofort ohne Bestätigung an die ausgewählte LED Leiste gesendet.
Über einen Navigation-Drawer vom linken Bildschrim sind neben dem bereits ausgewählten Reiter *Colorpicker* noch die Reiter *Effects*, *State* und *Settings* auswählbar. Diese haben folgende Funktionalitäten:

Effects Zeigt alle möglichen Effekte, die der Home Manager momentan umsetzen kann. Durch einen Klick auf die entsprechende Schaltfläche wird der Effekt auf der ausgewählten LED Leiste wiedergegeben.

State Diese Ansicht zeigt die Zustände (Farbe, Effekt) der LEDs, die sich momentan im Netzwerk befinden. Diese Liste wird vom Home Manager geliefert.

Settings Hier werden die nötigen Einstellungen getroffen. Momentan zählen hierzu die Adresse und der Port, unter der der Message-Broker erreichbar ist. Zusätzlich werden hier die Topics der einzelnen LEDs festgelegt, damit diese in anderen Teilen einfach ausgewählt werden können.

4.1.2 Funktionale Anforderungen

Die Funktionen wurden teilweise schon im vorherigen Abschnitt erläutert. Im Allgemeinen kann die App als eine grafische Oberfläche zur Erstellung & dem Versenden der verschiedenen JSON-Nachrichten gesehen werden. Die App stellt einen grafischen Colorpicker dar, wovon sich der Benutzer eine Farbe aussuchen kann. Sobald diese ausgewählt wurde, wird ein Listener aufgerufen, der den Farbcode des Colorpickers in RGB umwandelt und diese Informationen über entsprechende Objekte in ein JSON-Objekt umwandelt. Dieses wird mit dem vorher ausgewählten Topic, das über einen Button repräsentiert wird, auf das Netzwerk geschrieben, wodurch der Rest der Infrastruktur die LED zum Leuchten bringt.
Die JSON-Dokumente werden dabei von einer selbst erstellten Klasse *JsonBuilder* erstellt. Diese ist nach dem Builder-Pattern aufgebaut, sodass möglich einfach ein neues JSON-Dokument erstellt und in neue Komponenten eingepflegt werden kann.
Beim Reiter *Effects* werden entsprechend andere Teile des JSON-Dokuments eingefügt, sodass der Mikrocontroller diese versteht und die Effekte wie Regenbogen darstellt.
Im Reiter *State* befindet sich eine Liste aller in der App hinterlegten LEDs. Hier wird immer ein Paar aus dem Namen der LED, welcher in den anderen Komponenten angezeigt wird, und dem Topic der LED hinterlegt. Das Hinzufügen wird über einen Floating Action Button vorgenommen, der einen Dialog öffnet, in dem die Informationen eingetragen werden. Einzelne LEDs können mit einem langen Klicken des Elements einfach gelöscht werden.
Die letzte Seite *Settings* beinhaltet die Verbindungsparameter, des Message Brokers im Netzwerk. Hier wird die IP und der Port des Servers eingetragen, welche in den Shared Preferences abgelegt und im Rest der Anwendung verwendet werden.

4.2 Implementierung der Timer-App

Neben der Steuerungs-App für die LED-Leisten wurde eine weitere Android Applikation entwickelt, welche abhängig vom Fortschritt eines Timers eine LED-Leiste als Fortschrittsanzeige ansteuert.
Für eine Verwendung auf einem Tablet in einer Küche ist es wichtig, mehrere parallele Timer zu erstellen und diese auf einer Seite gleichzeitig laufen zu lassen.

Bei der Entwicklung sollte berücksichtigt werden, dass auch Benutzer ohne Smart Home Infrastrukutur die App verwenden können. Hierfür wird zwischen den beiden Flavors „default“ und „mqtt“ unterschieden. Die MQTT Flavor enthält alle Funktionalitäten der normalen Applikation und zusätzlich Funktionalitäten zur Integration in das Smart Home System.

4.2.1 Funktionale Anforderungen

Die Applikation soll folgende Anforderungen zuverlässig erfüllen:

- Mehrere Timer erstellen und zeitgleich ablaufen zu lassen. Dabei sollen mehrere Timer gleichzeitig sichtbar sein.
- Beim Ablauf eines Timers soll visuell und akustisch auf den Ablauf hingewiesen werden.
- Auch beim Schließen der Applikation sollen die Timer im Hintergrund weiterlaufen.

Zusätzlich sollen bei Verbindung mit einem MQTT-Broker folgende Funktionalitäten umgesetzt werden (nur in der Flavor „mqtt“):

- Konfigurieren eines MQTT Servers und eine automatische Verbindung mit dem Server bei App-Start
- Die Auswahl eines Timers in der App überträgt automatisch den aktuellen Fortschritt (Prozentual die abgelaufene Zeit zu der restlichen Zeit) an eine LED Kette. Hierbei wird die LED-Leiste in zwei Teile unterteilt, welche in verschiedenen Farben angezeigt werden
- Beim Ablauf eines Timers signalisiert die LED Leiste, dass ein Timer abgelaufen ist. Dies wird durch eine blinkende LED-Leiste repräsentiert.

4.2.2 Aufbau der Anwendung

Wie in Diagramm 4.1 gezeigt, besteht die App aus zwei Teilen. Der auch im Hintergrund laufende Service beinhaltet die Logik des sekündlichen Herunterzählens und verwaltet alle Timer. Die Benutzeroberfläche zeigt diese Werte in einem GridView

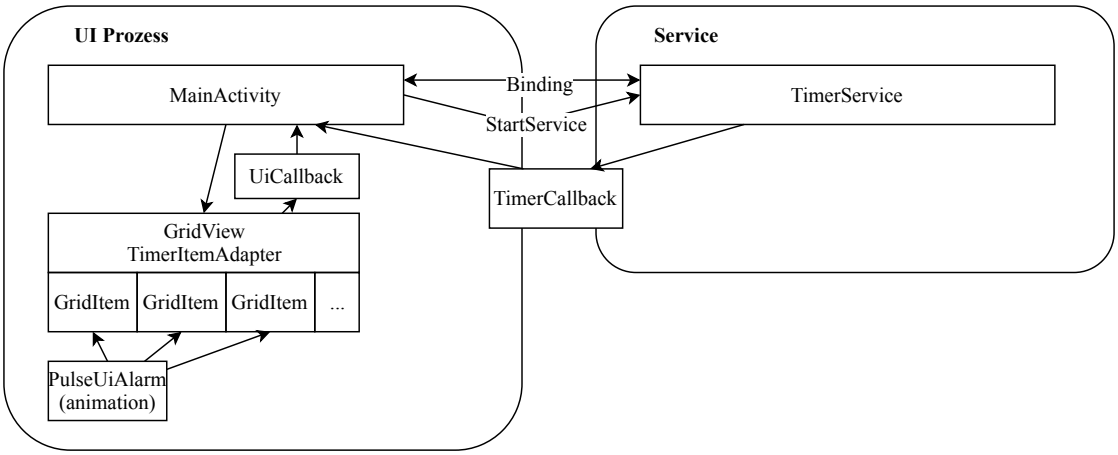

Abbildung 4.1: Timer App Überblicksarchitektur

an, wobei der Benutzer auch über die UI die Timer ändern kann. Das eigentliche Ändern der Objekte wird allerdings wieder von dem Service gemacht.

4.3 Mikrocontroller

Das in C geschriebene Programm für die einzelnen LED Steuereinheiten wurde als „thin client“ umgesetzt. Das heißt, dass nur möglichst einfache Funktionen auf dem Mikrocontroller ausgeführt werden und die komplette Logik auf die steuernden Clienten (siehe 4.1 und 4.2) und dem Manager (siehe **??**) verschoben werden. Mit der Arduino PubSubClient MQTT Library hört der Mikrocontroller ständig, ob eine neue Nachricht auf ein relevantes Topic angekommen ist. Wenn eine Nachricht ankommt, wird diese durch die „ArduinoJson“ Library geparsed und die Werte ausgelesen. Anschließend werden die verarbeiteten Werte mittels der Adafruit Neopixel Library auf die digitalen LEDs geschrieben. Sobald alle LEDs neu beschrieben wurden, hört der Mikrocontroller wieder auf neue Nachrichten vom MQTT Broker.

5 Ablauf des Projektes

5.1 Projektorganisation

Das Projekt wurde von Patrick Sudhaus und Alexander Hatzold verwirklicht. Aufgrund der geringen Dienstwege wurde keine besondere Form der Projektorganisation gewählt. Die Anforderungen wurden initial festgelegt und dann gemeinsam umgesetzt.

5.2 Technische Schwierigkeiten & deren Lösung

5.2.1 Verwendung einer Microcontroller MQTT Library

Ein Problem, welches bei der Implementierung des Mikrocontroller Programms aufgetreten ist, war die Entscheidung die Adafuit_MQTT_Library zu verwenden. Diese Library beschränkt MQTT Nachrichten auf eine Länge von 128 Bytes pro Nachricht. Die JSON-Nachrichten überschreiten manchmal diese Größe, sodass nicht die gesamte Nachricht übermittelt wird und somit nicht verarbeitbar ist. Da auch Anpassung der Library nicht erfolgreich war musste eine andere Library verwendet werden. Die Library *pubsubclient*[1] hat eine viel höhere Längenbeschränkung, wodurch sie für den use-case geeignet ist. Ein Problem beim PubSub Client sind die fehlenden QOS Level eins und zwei bei ausgehenden Nachrichten. Das heißt, dass es möglich ist, dass Nachrichten unbemerkt nicht ankommen können

[1] https://pubsubclient.knolleary.net/

(siehe 2.1). Da die Mikrocontroller nur Nachrichten empfangen ist dies aktuell nicht relevant.

5.2.2 Fehlende Nebenläufigkeit auf den Microcontrollern

Da die Microcontroller basierend auf ESP8266 keine Nebenläufigkeit unterstützen, musste sich für bestimmte Funktionalitäten eine andere Lösung überlegt werden, um gleichzeitig einen Befehl auszuführen (z.B. LED Lauflicht Effekte) und gleichzeitig andere Befehle zu erhalten (z.B. neue Nachrichten vom MQTT Broker empfangen) und zu starten. Die Parallelität wurde deswegen teilweise auf die Serverkomponente *Home Manager* (siehte Abschnitt **??**) ausgelagert, sodass alle gewünschten Funktionen realisierbar sind.

6 Fazit & Auswertung?

ALEX?

Literaturverzeichnis

[1] Banks, Andrew ; Gupta, Rahul: *MQTT Version 3.1.1 - Specification.* https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html. Version: Oktober 2014