Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

MBP2Go+: Erweiterung einer mobilen Applikation zur Überwachung und Steuerung von IoT-Umgebungen

Seda Ulusal

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr. -Ing. habil. Bernhard Mitschang
Betreuer/in:	Dr. rer. nat. Pascal Hirmer, Ana Cristina Franco da Silva, M. Sc.
Beginn am:	15. Juli 2018
Beendet am:	15. Januar 2019

Kurzfassung

Das Internet of Things, kurz IoT, ist heutzutage ein wichtiges Schlagwort. Es ermöglicht die Realisierung sogenannter intelligenter Umgebungen, die das Leben erleichtern sollen. Mit IoT verbindet man Begriffe wie Smart Home, Smart Factory oder Smart Cities, jedoch beschreibt IoT im Allgemeinen die Vernetzung von Geräten, die mit Sensoren und Aktuatoren ausgestattet sind. Insgesamt verfolgt IoT das Ziel eine adaptive und flexible Umgebung zu schaffen, die selbstständig auf Änderungen reagieren kann. Insbesondere haben sich mobile Applikationen in der IoT-Welt bewährt und werden zum Beispiel in Smart Homes für die Steuerung und Überwachung von Geräten verwendet. Die „Multi-purpose Binding and Provisioning Platform“ (MBP) ist eine an der Universität Stuttgart entwickelte Plattform, die es ermöglicht Sensoren und Aktuatoren von Geräten aus IoT-Umgebungen automatisch anzubinden, deren Daten zu erfassen und zu visualisieren. Die MBP wurde als Web-Applikation implementiert und bietet eine grafische Oberfläche. Außerdem wurde die MBP um die mobile Applikation „MBP2Go“ erweitert. Zusammen ermöglichen sie, ein automatisiertes Anbinden von Geräten und deren Sensoren und Aktuatoren an. Das Ziel dieser Bachelorarbeit ist es, die „MBP2Go“ um weitere Funktionalitäten zu erweitern. Dabei steht die Überwachung und die automatische Steuerung im Mittelpunkt dieser Arbeit. Mit Hilfe der Erweiterungen wird es möglich sein, Sensordaten zu visualisieren und Regeln zur aktiven automatischen Steuerung von IoT-Umgebungen zu definieren. Dadurch wird eine automatische Überwachung und Steuerung mit Hilfe der „MBP2Go“ ermöglicht.

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Listings

Abkürzungsverzeichnis

CEP Complex Event Processing 20

DNF Disjunktive Normalform 41

IoT Internet of Things 3

JSON JavaScript Object Notation 41

MBP Multi-purpose Binding and Provisioning Platform 3

1 Einleitung

Durch das Internet of Things - kurz IoT oder auf deutsch Internet der Dinge - wird die Welt um uns herum intelligenter und interaktiver. Erstmals wurde der Begriff „Internet of Things“ 1999 von Technologie-Pionier Kevin Ashton verwendet [6]. Heute werden immer mehr Alltagsgegenstände mit dem Internet verbunden und vernetzt. Dabei verfolgt man die Idee, physische Geräte mit Internettechnologien zu vernetzen, damit sie zu intelligenten Geräten werden, die eine intelligente Umgebung, zum Beispiel ein Smart Home, schaffen. Dementsprechend verfolgt IoT die Vision, dass diese intelligenten Geräte selbstständig handeln, ohne dass der Mensch interagieren muss. Ein einfaches Beispielszenario ist ein Heizkörper, der sich je nach Raum und Außentemperatur selbstständig anschaltet [39].

Damit solche Szenarien verwirklicht werden können, braucht man Sensoren und Aktuatoren. Durch das Sammeln und Aufzeichnen von Sensordaten aus der realen Welt kann ein Zustand erfasst werden. Daraufhin kann die Integration von Aktuatoren, das Steuern der intelligenten Umgebung ermöglichen. Jedoch braucht man noch eine weitere Komponenten, damit intelligente Umgebungen verwirklicht werden. IoT-Applikationen sorgen dafür, dass Sensoren und Aktuatoren miteinander verbunden, gesteuert und überwacht werden können. Dabei gibt es mehrere Herausforderungen, mit denen sich IoT-Applikationen befassen müssen. Intelligente Umgebungen sind sehr dynamisch in dem Sinne, dass Geräte ausfallen, verschoben, hinzugefügt oder entfernt werden können. Ein Beispiel hierfür ist, das Betreten oder Verlassen der Umgebung durch ein Smartphone. Des Weiteren könnten die zugrundeliegenden Netzwerke fehleranfällig sein oder eine beschränkte Bandbreite haben. Somit ist die Wahl des Netzwerkprotokolls zur Kommunikation wichtig [25]. Außerdem ist die automatisierte Bindung und Überwachung von Geräten in intelligenten Umgebungen nach wie vor ein großes Problem, insbesondere in mobilen Applikationen. Die Plattform MBP [26] und die dazugehörige App MBP2Go sind IoT-Applikationen, die sich den Herausforderungen der IoT stellen.

Das Ziel dieser Bachelorarbeit ist es, die IoT-App MBP2Go um weitere Funktionalitäten zu erweitern. Dabei steht die Überwachung und die automatische Steuerung im Mittelpunkt dieser Arbeit. Mit Hilfe der Erweiterungen wird es möglich sein, Sensordaten zu visualisieren und Regeln zur aktiven automatischen Steuerung von IoT-Umgebungen zu definieren. Dadurch wird eine automatische Überwachung und Steuerung mit Hilfe der MBP2Go App ermöglicht.

Bezüglich der Überwachung wurden drei grundlegende Erweiterungen implementiert. Die erste Erweiterung ermöglicht es, Echtzeit-Sensordaten zu beobachten. Durch die zweite Erweiterung bietet MBP2Go die Möglichkeit an, historische Daten zu visualisieren. Außerdem wurde die App um die Funktionalität zur Analyse historischer Daten ergänzt.

Im Bereich der Steuerung liegen zwei Erweiterungen im Fokus. Die erste Erweiterung ist die Definition von Schnittstellen, über die Regeln in der MBP hinzugefügt werden können. Es werden dabei Konzepte vorgestellt, wie Regeln für die Steuerung von IoT-Umgebungen aussehen könnten. Die Regeln definieren Bedingungen für Sensordaten und beschreiben, welche Aktuatoren beim Eintreffen dieser Bedingungen aktiviert werden sollen. Hierbei wird die Technologie Complex Event Processing betrachtet und Aspekte derer für die mobile Applikation integriert. Die letzte Erweiterung ist die Entwicklung einer prototypischen Demonstration einer Rule Engine innerhalb der MBP2Go. Dieser Prototyp wird zukünftig durch die Implementierung einer Rule Engine in der MBP ersetzt.

1.1 Struktur der Arbeit

Die nachfolgende Arbeit ist wie folgt gegliedert:

Chapter 2 – Grundlagen In diesem Kapitel werden die Grundlagen erklärt, welche für die Nachvollziehbarkeit der weiteren Kapitel notwendig sind. Dabei werden unter anderem die Begriffe MQTT, MBP, MBP2Go und Complex Event Processing erklärt.

Chapter 3 – Verwandte Arbeiten Dieses Kapitel beschäftigt sich mit Arbeiten, die im Zusammenhang dieser Bachelorarbeit stehen. Dabei werden insbesondere weitere IoT-Apps betrachtet, die ähnliche Funktionalitäten wie die MBP2Go aufweisen.

Chapter 4 –MBP2Go+: Erweiterung einer mobilen Applikation In diesem Kapitel wird auf die Erweiterungen der MBP2Go eingegangen. Dabei wird auf die Aufgabenstellung, die Lösung und Implementierung der mobilen Applikation eingegangen.

Chapter 5 – Zusammenfassung und Ausblick Abschließend werden in diesem Kapitel die Ergebnisse dieser Arbeit zusammengefasst. Dazu wird in einem Ausblick erörtert, welche zukünftigen Erweiterungen in der MBP2Go sinnvoll sind.

2 Grundlagen

In diesem Kapitel werden alle thematischen Grundlagen behandelt, die für die Nachvollziehbarkeit der Bachelorarbeit notwendig sind. Weiterführende Informationen sind den jeweiligen Quellen zu entnehmen.

2.1 MQTT

Damit die verschiedensten Geräten im IoT miteinander verbunden werden können, ist ein Übertragungsprotokoll unvermeidlich. Dabei sind folgende Eigenschaften sehr wichtig: In der IoT werden oft eingebettete Systeme verwendet, die wenig Speicherkapazität und Rechenkapazität aufweisen. Dies bedingt die Notwendigkeit eines leichtgewichtigen und schlanken Protokolls.

Des Weiteren, kann es in IoT-Umgebungen oft zu instablien Netzen mit einer geringen Bandbreite kommen, weshalb ein geringer Overhead sehr wichtig ist. Außerdem sollte das Protokoll für viele Programmiersprachen und jegliche Art von Geräten offen sein, da in der IoT verschiedenste Geräte miteinander verbunden werden. In MBP2Go und MBP wird das Protokoll MQTT eingesetzt.

MQTT ist ein leichtgewichtiges, ereignis- und nachrichtenorientiertes Anwendungsprotokoll, das für die Kommunikation zwischen Geräten entwickelt wurde. Das Protokoll wurde 1999 von IBM entwickelt und eignet sich besonders für IoT, da es einige Vorteile mit sich bringt. Das Nachrichtenprotokoll berücksichtigt, dass es in IoT-Umgebungen zu unstabilen Netzen kommen kann. Zudem erzeugt es nicht viel Overhead [38]. MQTT ist für große Netze geeignet, da es bis zu 10.000 Clients unterstützen kann. Die Funktionsweise von MQTT ist einfach, es basiert auf dem Server-Client-Protokoll und hat eine Publish und Subscribe-Architektur. Dabei dient der Server üblicherweise als Message Broker, der für die Nachrichtenverwaltung und -verteilung verantwortlich ist. Es kann eine Vielzahl von Clients geben, die auf einem Server publishen oder subscriben. Unter Publish versteht man, dass Geräte ihre Nachricht an den Broker senden, falls sie etwas zu melden haben. Der Broker sendet die Nachricht weiter an andere Geräte, falls sie sich auf diese Nachricht subscribed haben. Ein Vorteil dieses Prinzips ist, dass die Subscriber und Publisher entkoppelt sind. Das heißt, dass Sender und Empfänger sich weder kennen, noch zeitgleich ausgeführt werden müssen [24]. Außerdem ist dabei eine asynchrone Kommunikation möglich. Damit ein Client einen Subscribe oder Publish auf dem Server ausführen kann, muss ein Topic und eine QoS-Stufe angegeben werden. Dabei dient das Topic als Filter von Nachrichten

und ist eine Zeichenkette, die von der Schreibsynatx her dem von Dateisystemen und Ordnern ähnelt. So könnte zum Beispiel „sensor/temperatur/id“ als Topic geeignet sein. Ein Client, der zum Beispiel dieses Topic subscribed, würde vom Broker nur die Nachrichten empfangen die Temperaturdaten des Sensors beinhalten.

Die Quality of Service-Stufe (QoS-Stufe) gibt an, wie zuverlässig Nachrichten übermittelt werden sollen. Dabei gibt es 3 Kategorien. Bei einem QoS mit der Kategorie 0, auch als „Fire and forget“ bekannt, gibt es keine Maßnahme zur Sicherstellung, dass die Nachrichten bei ihrem Ziel ankommen. Das Nachrichtenpaket wird genau einmal versendet und bei instabilen Netzwerken kann es vorkommen, dass das Nachrichtenpaket nicht ankommt. Jedoch ist das Versenden sehr schnell und es werden wenige Pakete pro Nachricht geschickt. QoS 0 eignet sich somit für Situationen, in denen es nicht auf die einzelne Nachricht ankommt. Ein Beispiel dafür ist, wenn Nachrichten in einer hohen Frequenz zur Aktualisierung eines Wertes gesendet werden [24]. Bei QoS 1, dass auch „Acknowledgement“ genannt wird, bestätigt der Empfänger dem Sender, dass er das Nachrichtenpaket erhalten hat. Jedoch können Nachrichtenpakete mehrmals ankommen. Dies ist der entscheidende Unterschied zu QoS 2, dass auch unter dem Begriff „Synchronisiert“ bekannt ist. Bei QoS 2 wird sichergestellt, dass das Nachrichtenpaket genau einmal beim Empfänger ankommt und das Duplikate nicht beachtet werden. Dadurch bieten QoS1 und QoS 2 eine Zuverlässigkeit an, dass die Pakete ankommen, jedoch werden bei diesen Stufen gegebenenfalls viele Nachrichtenpakete verschickt, wodurch das Netz schneller ausgelastet werden kann. Bei einem Publish kommt im Vergleich zu einem Subscribe hinzu, dass der Publishvorgang des Clients neben dem Topic und QoS noch eine Nachricht beinhaltet [24].

2.2 Multi-purpose Binding and Provisioning Platform

Die „Multi-purpose Binding and Provisioning Platform“ [26], kurz MBP, ist eine Plattform mit der man automatisiert auf IoT-Geräte, mit ihren Sensoren und Aktuatoren, zugreifen kann. Abbildung 2.1 stellt die Architektur der MBP dar. Die Geräte können manuell registriert werden. Dies kann sowohl per REST-API erfolgen oder durch die vorhandene grafische Oberfläche der MBP, wie in Abbildung 2.2 zu sehen ist. Nach der Registrierung bekommt das Gerät eine eindeutige Id. Dies ist nötig, um das Gerät eindeutig Identifizieren zu können [29]. Danach können auch die zugehörigen Sensoren und Aktuatoren des Gerätes registriert werden. Die MBP bietet drei unterschiedliche Arten von Gerätebindungen an: Plug-and-Play-Gerätebindungen, konfigurierbare Gerätebindungen und eingeschränkt konfigurierbare Gerätebindungen [22]. Bei einem Plug-and-Play-Gerät sind die Sensoren und Aktuatoren eingebettet. Der Adapter solcher Geräte kann nicht auf dem Gerät selbst zur Verfügung gestellt werden, sondern muss an anderer Stelle zum Beispiel einem Gateway oder einer Virtuelle Maschine aufgesetzt werden. Dafür bieten diese Geräte Schnittstellen an, zum Beispiel REST oder MQTT, damit die Sensordaten extrahiert werden können [28]. Die MBP2Go stellt zum Beispiel das Smartphone als Plug and Play Gerät zur Verfügung.

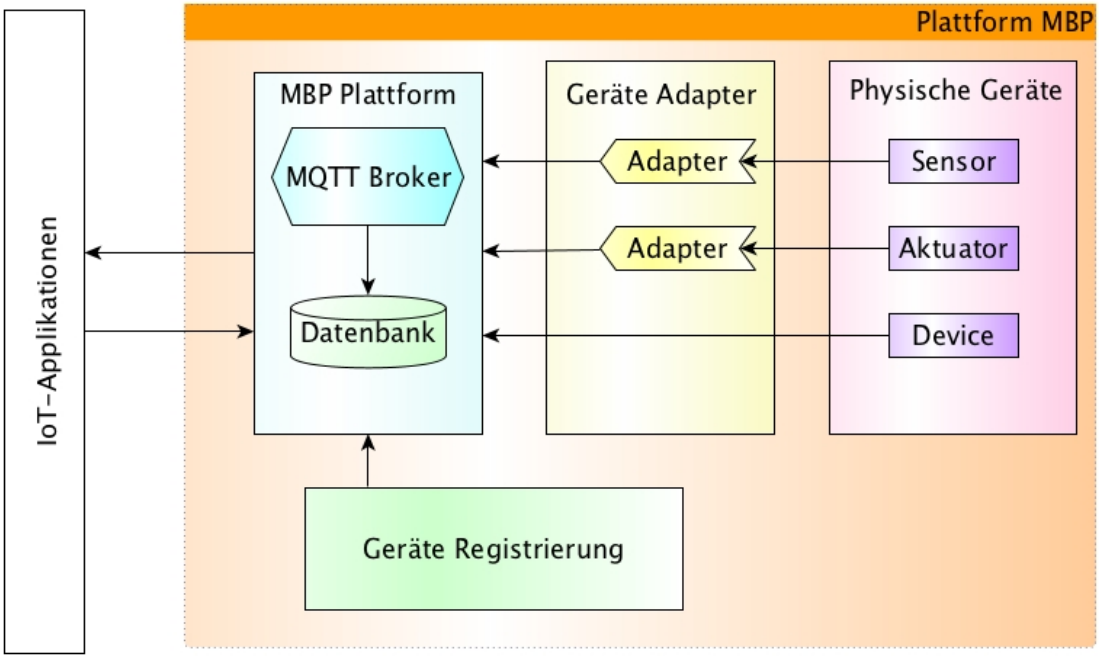

Abbildung 2.1: Die Architektur der MBP

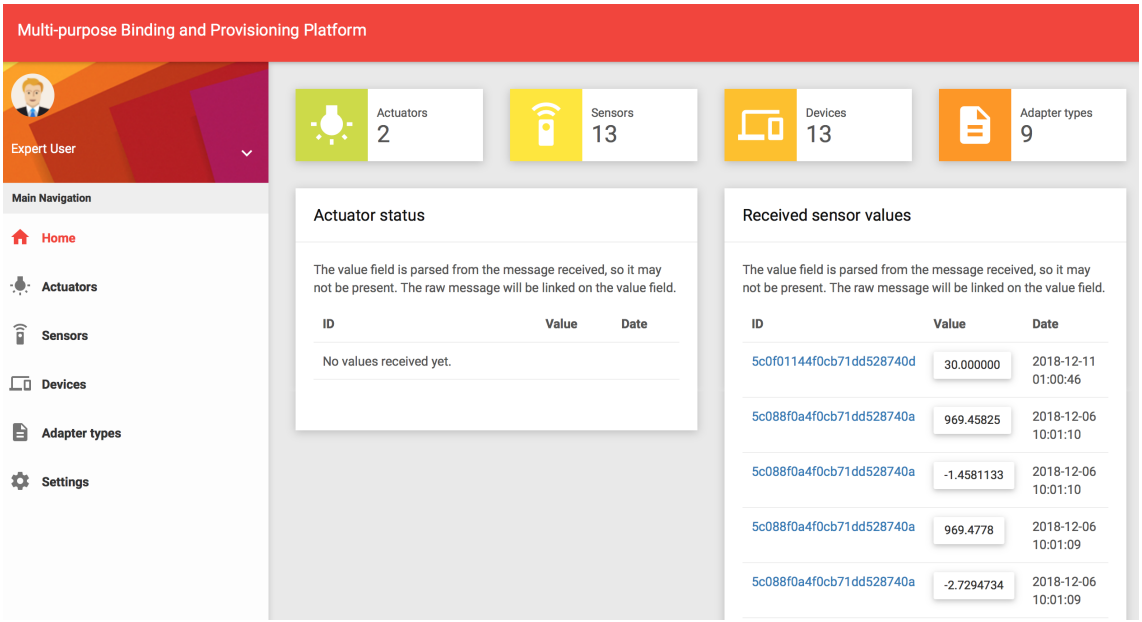

Abbildung 2.2: Die grafische Oberfläche der MBP

Unter konfigurierbarer Gerätbindung versteht man eine Bindung zwischen einem Gerät an dem Sensoren und Aktuatoren angeschlossen sind und der MBP. Ein mögliches Beispiel für ein konfigurierbares Gerät ist ein Raspberry Pi. Für diese Art von Geräteverbindung werden ebenfalls Adapter benötigt. Diese betreibt man hier auf dem Gerät selbst.

Das Konzept, Adapter zu verwenden um Geräte miteinander zu verbinden, ist in der IoT ein gängiges Konzept [22]. Der Adapter beinhaltet die Logik, Geräte zu verbinden, Aktuatoren zu aktivieren und die Sensordaten von der Schnittstelle zu extrahieren, damit sie der MBP zur Verfügung gestellt werden können. Dabei kann der Adapter parametrisiert werden, damit man dynamische Informationen angeben kann. Ein Beispiel für einen Parameter stellt die MAC-Adresse des Gerätes dar. Adapter können auf dem Gerät selbst oder auf einer externen Ressource bereitgestellt werden. Bei einem eingeschränkt konfigurierbaren Gerät sind Sensoren und Aktuatoren an Geräten angeschlossen, die eine beschränkte Verarbeitungs- und Speicherfunktion besitzen. Ein Beispiel-Gerät dafür ist ein Arduino2 [22].

Nach der Registrierung der Geräte und der Adapter können die zugehörigen Sensoren und Aktuatoren registriert werden. Diese können „deployed“ werden. Ab diesem Zeitpunkt erfasst der dazugehörige Adapter die Sensordaten [29]. Wenn das Gerät ausfällt oder ein Sensor nicht mehr benötigt wird, kann der Sensor „undeployed“ werden. Das bedeutet, dass der Adapter gestoppt wird. Ab diesem Zeitpunkt werden die Sensordaten nicht mehr erfasst. Die MBP bietet durch ihre grafische Oberfläche die Möglichkeit, die Sensordaten zu überwachen. Die grafische Oberfläche besitzt jeweils zwei Kategorien zur Überwachung der Sensordaten: „Live Sensor Values“ und „Historical Sensor Values“ . Bei der Kategorie „Live Sensor Values“ werden die aktuellen Sensordaten, sobald der Sensor ”„deployed“ wurde, grafisch angezeigt. Die Sensordaten werden innerhalb der MBP in einer Datenbank gespeichert. Dadurch können alle vergangene Sensordaten im Graph „Historical Sensor Values“ anzeigt werden. Somit wird ein Überblick über die Sensordaten geschaffen.

Mit den oben beschriebenen Schritten bietet die MBP einen vollständig automatisierten Prozess vom Binden von Geräten bis zur Bereitstellung ihrer Daten. Mit ihren Diagrammen ermöglicht die MBP das Überwachen der Geräte durch ihre grafische Oberfläche. Zurzeit bietet sie jedoch noch keine Möglichkeit, die Geräte zu steuern. Des Weiteren bietet die dazugehörige App MBP2Go noch nicht die Funktionen zur Überwachung der Sensoren. Eines der Ziele dieser Bachelorarbeit ist es, die MBP2Go um die Funktionen zur grafischen Überwachung der Sensoren und zur Steuerung der Geräte zu erweitern.

2.3 MBP2Go

Heutzutage gewinnen mobile Applikationen immer mehr an Bedeutung. Besonders Android ist heute auf dem Markt sehr erfolgreich [18]. Darüber hinaus bieten Smartphones durch ihre Sensoren und Dienste viele Möglichkeiten in der IoT. Ein Beispiel ist, dass durch ein Smartphone die Position einer Person bestimmt werden kann. Ein weiterer großer Vorteil ist, dass heutzutage nahezu jeder ein Smartphone besitzt. Etwa über 2 Milliarden

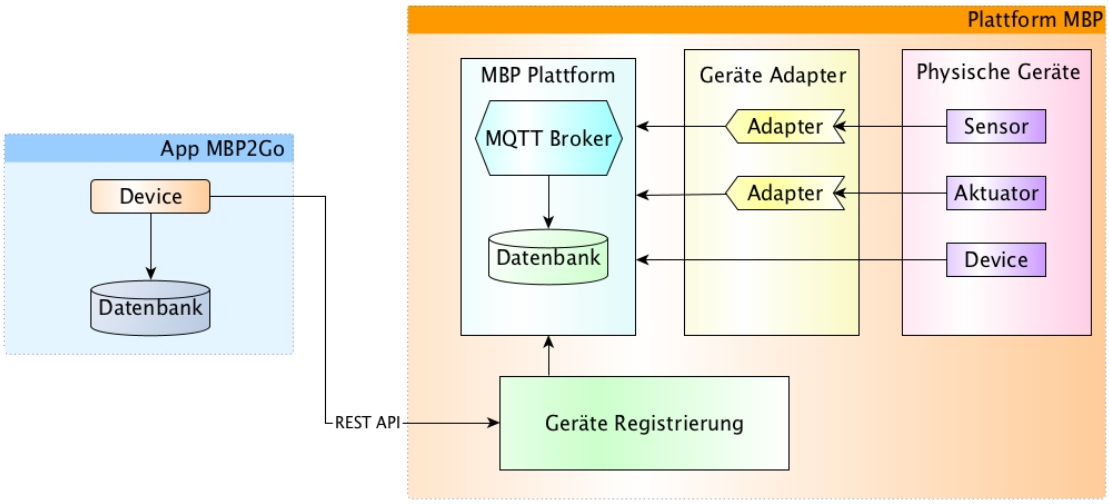

Abbildung 2.3: Die Architektur der MBP2Go

Menschen besitzen ein Smartphone mit dem Betriebssystem Android [7]. Zudem ist es Smartphones zu verdanken, dass eine kontinuierliche Verbindung zum Internet Einzug in den Lebensalltag heutiger Menschen gehalten hat.

Die MBP2Go ist eine IoT-App, die zu der Plattform MBP [26] gehört und sie in vielen Bereichen erweitert. Abbildung 2.4 beschreibt den Zustand der MBP2Go vor der Erweiterung der Überwachung und Steuerung. Die Komponente *Device* ist für die Registrierung der Geräte mit ihren jeweiligen Sensoren und Aktuatoren verantwortlich. Zum Beispiel können Geräte wie ein Raspberry Pi oder ein Arduino mit ihren jeweiligen Sensoren registriert werden. In Abbildung 2.4 ist die grafische Oberfläche der MBP2Go für die Registrierung von Geräten mit ihren Sensoren zu sehen. Dabei ist der Sensortype der jeweilige Adapter, der in der MBP vorhanden ist. Bei der Registrierung wird durch eine REST-API das Gerät sowohl in der MBP, als auch in der MBP2Go gespeichert. Für die Sicherung der Daten verwendet die MBP2Go eine lokale SQLite Datenbank. Die App ermöglicht es Angaben zu ergänzen und zu korrigieren. Dies ist eine Funktion, welche die Plattform MBP nicht anbietet. Die Geräte können auch in der MBP2Go entfernt werden. Die Synchronisation mit der Plattform MBP verläuft dabei automatisch. Außerdem bietet die MBP2Go zwei Möglichkeiten der Synchronisation mit der Plattform MBP. Die erste Möglichkeit ist, wenn die Activity mit der Übersicht der Geräte geöffnet wird. Dann prüft die MBP2Go automatisch, ob alle vorhandene Geräte in der MBP2Go mit der MBP übereinstimmen. Die zweite und einfache Möglichkeit ist ein Knopf, der dies ebenfalls prüft. Dabei verläuft die Synchronisation gleichermaßen ab. Falls es Geräte gibt die in der MBP2Go vorhanden sind, jedoch in der MBP gelöscht wurden, werden diese entfernt. Des Weiteren ermöglicht die MBP2Go Sensoren zu deployen und zu undeployen. Beide Begriffe bedeuten in dem Kontext dasselbe wie in der MBP. Sobald ein Sensor deployed wurde ist dieser aktiv und sendet Sensordaten an die MBP. Die MBP2Go bietet sogleich eine Übersicht, welche Sensoren eines Geräts deployed wurden. Durch den Knopf „deploy“ wird eine REST Anfrage

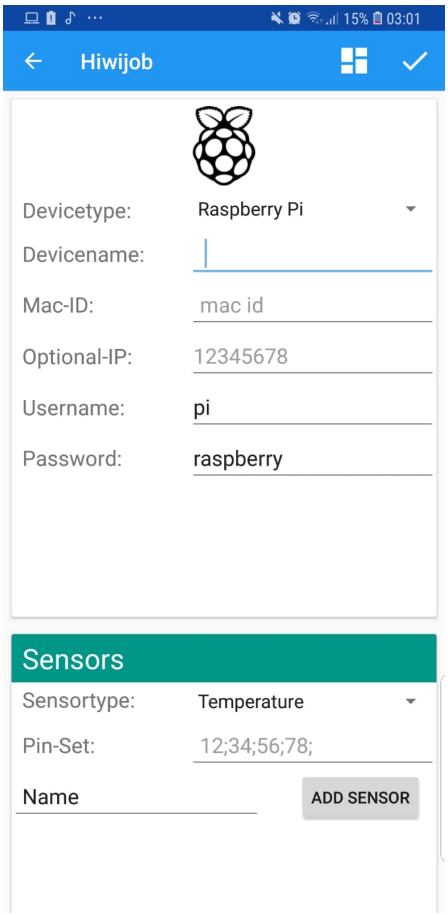

Abbildung 2.4: Registrierung eines Geräts in der MBP2Go

an die MBP geschickt. Wenn der Sensor vorhanden ist, wechselt der Button zu „deployed“ und die MBP empfängt die Sensordaten. Falls der Sensor nicht vorhanden oder ein Fehler aufgetreten ist, bleibt der Button auf „undeployed“. Weiterhin ist es mit der MBP2Go möglich die Sensoren vom Smartphone mit der Plattform MBP zu verbinden. Somit dient die App dazu, dass eigene Smartphone in ein IoT-Gerät umzuwandeln. Die Sensoren der App werden automatisch in einer Liste angezeigt. Die Sensoren können ausgewählt und mit dem Smartphone als Gerät registriert werden.

Die MBP2Go wird in dieser Bachelorarbeit erweitert. Die App wird danach zusätzlich Funktionen der Überwachung und Steuerung von IoT-Geräte besitzen.

2.4 Complex Event Processing

In der IoT gibt es oft große Datenmengen, zum Beispiel die erfassten Echtzeit-Sensordaten mehrerer Geräte. Oft ist das Ziel in der IoT aus den erfassten Daten Entscheidungen abzuleiten. Live Daten, also kontinuierliche Datenströme mit aktuellen Daten, sind in der

IoT ein wichtiger Bestandteil. Complex Event Processing (CEP) ist eine Softwaretechnologie, die sich mit der Analyse von riesigen Datenstören in Echtzeit beschäftigt [4]. Dabei beschreibt CEP eine dynamische Analyse eines Datenstroms in Echtzeit [4]. Dadurch können verschiedene Beziehungen zwischen Ereignissen, zum Beispiel temporale oder räumliche, beschrieben werden. Diese Beziehungen beschreiben schließlich Muster, die ausgewertet werden können. Complex Event Processing wurde von David Luckham entwickelt und rückt hauptsächlich Ereignisse, auf englisch Events genannt, in den Vordergrund von Softwarearchitekturen. Unter einem Ereignis versteht man etwas, das passieren kann oder dessen Eintreten erwartet wird. Es beschreibt die Veränderung eines Zustands, zum Beispiel wenn sich Sensordaten aktualisieren und ändern. Damit man Ereignisse verstehen und einordnen kann, braucht man Ereignisdaten. Ereignisdaten beinhalten Informationen, die für die Verarbeitungsvorgänge wichtig sind. Sie bestehen dabei aus Meta- und Kontextdaten. Metadaten beschreiben die allgemeinen Daten eines Ereignisses, zum Beispiel einen Zeitstempel oder die eindeutige ID des Ereignisses. Der Begriff Kontextdaten beschreibt, welchen Sachverhalt ein Ereignis hat. Allgemein ist der Begriff als Nutzdaten, auf englisch Payload, bekannt. Ein Beispiel für ein Kontextdatum ist der gemessene Temperaturwert eines Sensors. Falls mehrere Ereignisse in einer kontinuierlichen Sequenz eintreffen, ist das ein Ereignisstrom [4].

3 Verwandte Arbeiten

In diesem Kapitel werden zu dieser Bachelorarbeit verwandte Arbeiten beschrieben, die ähnliche Applikationen wie die MBP2Go umsetzen. Im Kontext dieser Arbeit wird sich dabei auf Applikationen bezogen, die eine Überwachung oder Steuerung von IoT-Umgebungen ermöglichen.

Die Fachstudie „Ultimativer Vergleich mobiler IoT-Applikationen“ [10] gibt eine Übersicht über mobile IoT-Apps. In dieser Fachstudie wurden Android Apps betrachtet. Der Google Play Store bietet eine große Menge an IoT-Applikationen. Im Folgenden werden nur die Apps betrachtet, die der MBP2Go ähneln.

Die App „Stringify - Smart Home and IoT“ bietet die Möglichkeit über 600 Geräte und Dienste zu steuern und zu automatisieren [35]. Sie wurde für die Gebiete Smart Health und Smart Home entwickelt und unterstützt Geräte wie Fitness Tracker zum Beispiel Fitbit und Smart Home Anwendungen wie Nest Thermostate. In der Öffentlichkeit erhielt die

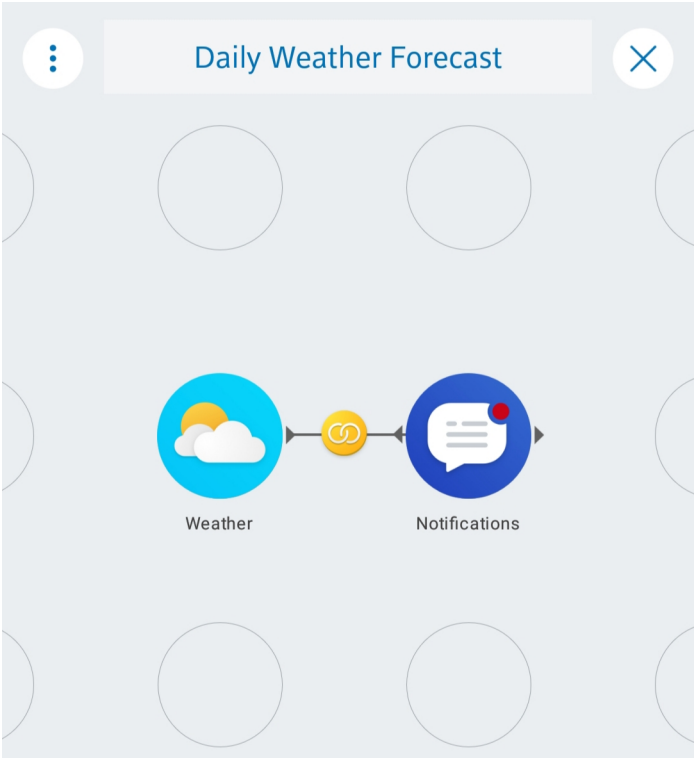

Abbildung 3.1: Ein Beispiel eines Flows in der „Stringify“-App

App Aufmerksamkeit durch Berichte von Technik-Blogs wie Lifehacker [37] und Digital Trends [36] und wurde für ihre Funktionalität ausgezeichnet [35]. Das automatisierte Binden der Geräte verläuft nach folgendem Prozess: Die App bietet eine Kategorie „Discover“. In dieser werden Geräte und Dienste vorgeschlagen. Für viele Geräte und Dienste muss man sich registrieren, zum Beispiel braucht man für Amazon Alexa neben dem Gerät, die App Amazon Alexa. Einige Dienste wie der Wetterdienst sind bereits vorinstalliert. Die Hauptfunktion der App ist, Geräte und Dienste durch genannte „Flows“ zu Steuern. Unter einem Flow kann man eine „Wenn-Dann-Regel“ verstehen, d.h. Regeln bestehen aus einer Bedingung und einer darauf folgenden Aktion. Eine Regel könnte beispielsweise folgendermaßen aussehen: „Wenn das Wetter schön ist, dann schicke mir eine Nachricht.“ . Ein Flow besteht aus einem oder mehreren Knoten. Ein Knoten stellt jeweils ein Gerät oder einen Dienst dar. Diese Knoten können dann miteinander verbunden und jeweils konfiguriert werden. In Abbildung 3.1 ist ein Flow, der dem vorigen Wetter-Beispiel entspricht, zu sehen. Die App verwendet als Architekturstil für ihre Regeln das Pipes and Filter Pattern [11]. Die Knoten stellen die Filter dar und die Verbindungen zwischen den Knoten die Pipes. Dadurch erhält der Anwender einen schnellen Überblick. In dieser Umsetzung sind jedoch lediglich „und“ -Verknüfungen möglich. Das heißt, Regeln können nur die folgende Struktur annehmen: „**Wenn** Zustand A *und* Zustand B *und* Zustand C... **dann** führe Aktion A *und* Aktion B *und* Aktion C ... aus.“ . Das schränkt die Menge der möglichen Regeln ein, da es keinen „oder“-Operator gibt.

Die App „Stringify“ Stringify enthält einige gute Konzepte, die für MBP2 Go als Vorbild dienen können. Die App gibt einen Überblick, wie die Geräte konfiguriert werden können.

Das Flow Modell für Regeln wie in Abbildung 3.2 zu sehen ist allerdings durch die fehlende Möglichkeit zur Oder-Verknüpfung stark eingeschränkt. Die MBP2Go soll komplexere Regeln erstellen und darstellen können. Die App „Stringify“ bietet zwar eine schöne Oberfläche, um viele Geräte einzubinden und zu steuern, jedoch fehlt der App die Möglichkeit eigene Geräte anzubinden und die Möglichkeit diese Geräte zu überwachen.

„IoTool Internet of Things (IoT) Dashboard/Gateway“ ist eine App, die mehr Möglichkeiten anbietet eigene Geräte anzubinden [**iotoolGate**]. Die App verspricht, sowohl die Geräte überwachen als auch steuern zu können. Um die Geräte zu überwachen, bietet die App Diagramme, mit denen man die Sensorwerte beobachten kann. Dabei sind diese Werte Echtzeit-Daten. Außerdem können auch historische Daten beobachtet werden. Durch Regeln, die in „Wenn-Dann-Regel“ umgesetzt wurden, wie bei der „Stringify“ App, kann man Aktionen durchführen lassen. In Abbildung 3.3 sieht man ein Beispiel, wie Regeln in der IoTool App erstellt werden. Besonders hervorzuheben sind die Einstellungsmöglichkeiten der Regeln. Dadurch ermöglicht die App die Steuerung von Geräten und erfüllt somit die Funktionen der Steuerung und Überwachung von IoT-Umgebungen.

Jedoch ist die Handhabung der App nicht einfach. Die Diagramme und das Layout der App sind schwer verständlich und führen zu einer geringen Usability. In Abbildung 3.4 ist das Diagramm der Echtzeit-Sensordaten zu sehen. Erkenntlich ist, dass die Diagramme dem Anwender keine schnelle Übersicht geben. Die App bietet keine automatische Anbindung der Geräte. Falls man eigene Geräte verwenden will, muss der Anwender eine E-mail an

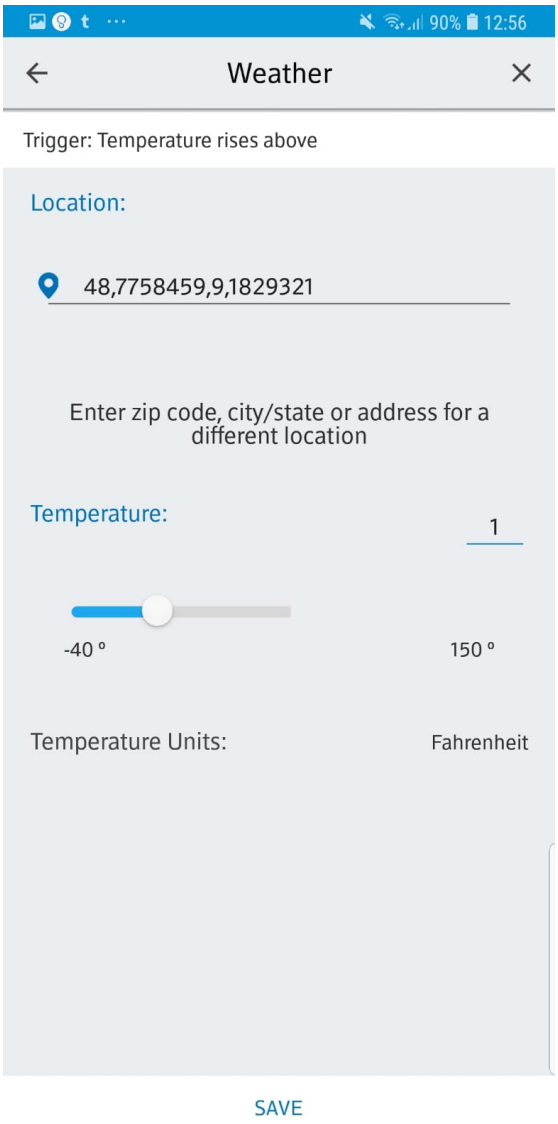

Abbildung 3.2: Konfiguration des Wetter-Knoten der „Stringify“-App

die Entwickler schreiben [34]. Es gibt viele Geräte, die mit der App funktionieren, wie zum Beispiel das Gerät Bosch XDK. Jedoch wird für die Anbindung der Geräte das Installieren von weiteren Apps verlangt, zum Beispiel die App „IoTool Device Sensors“ [**iotoolSensor**]. Auch für die Nutzung von weiteren Funktionen, zum Beispiel das Hinzufügen einer Aktion, muss die App „IoTool Device Actions“ [**iotoolDevice**] installiert werden. Das heißt, die App „IoTool Internet of Things (IoT) Dashboard/Gateway“ ist allein nicht vollständig. Das sorgt beim Anwender für eine schlechte User Experience.

Die MBP2Go soll ebenfalls Echtzeit-Sensordaten durch ein Diagramm anzeigen. Eine weitere wichtige Anforderung an die MBP2Go ist, dass die Handhabung leicht verständlich ist, weshalb ein schönes Layout und Diagramme wichtig sind.

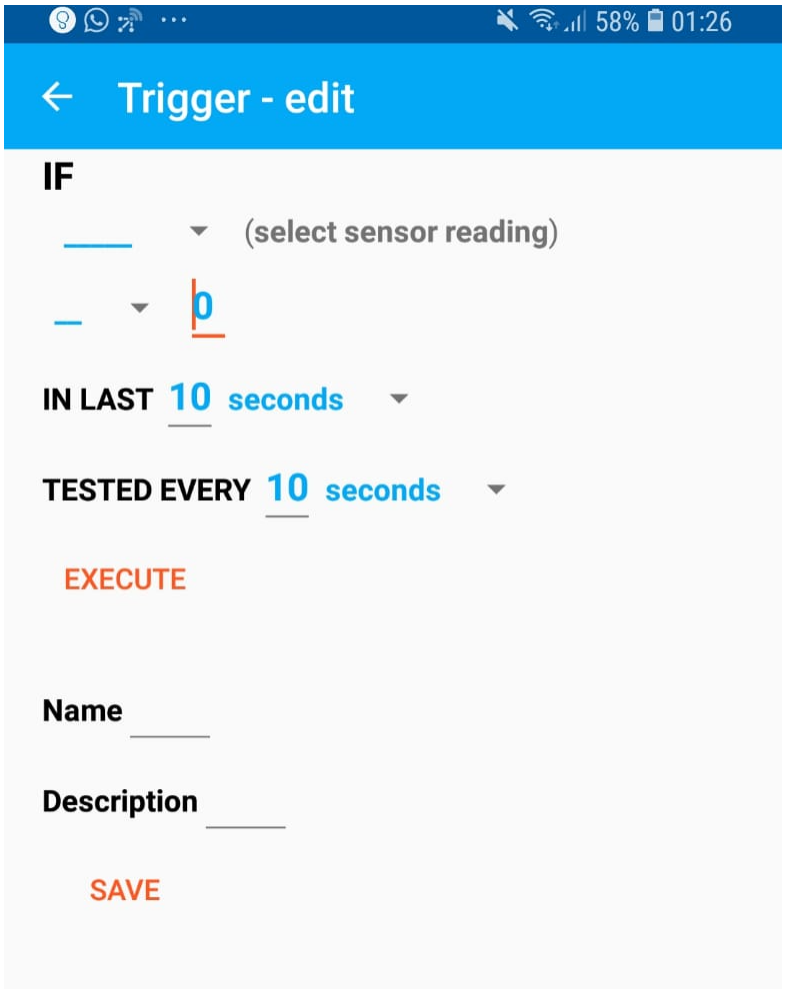

Abbildung 3.3: Erstellung einer Regel in der „IoTool Internet of Things (IoT) Dashboard/Gateway“-App

Die App „Yonomi Smart Home Automation“ wurde speziell für das Gebiet Smart Home entwickelt. Sie unterstützt viele Smart Home Geräte [40], jedoch bietet sie keine Möglichkeit an, eigene Geräte anzubinden. Die App betrachtet, anders als bei anderen IoT-Apps, auch das Smartphone als Gerät und bietet somit Dienste vom Smart Phone an. Ein Dienst kann zum Beispiel die Ortung vom Standort sein. Zudem können Regeln erstellt werden, um Geräte und Dienste zu steuern. Abbildung 3.5 zeigt wie Regeln in der App erstellt werden. Die Regeln bestehen aus einer „Wenn-Dann“-Struktur, also einer Bedingung und der darauffolgenden Aktion. In der Abbildung 3.5 ist ein Beispiel von einer Regel zu sehen. Die Regel lautet:

„Wenn ich zuhause ankomme oder es 15:14 Uhr ist, dann sende mir eine Nachricht mit dem Inhalt Welcome to Yonomi, aber nur am Wochenende.“ .

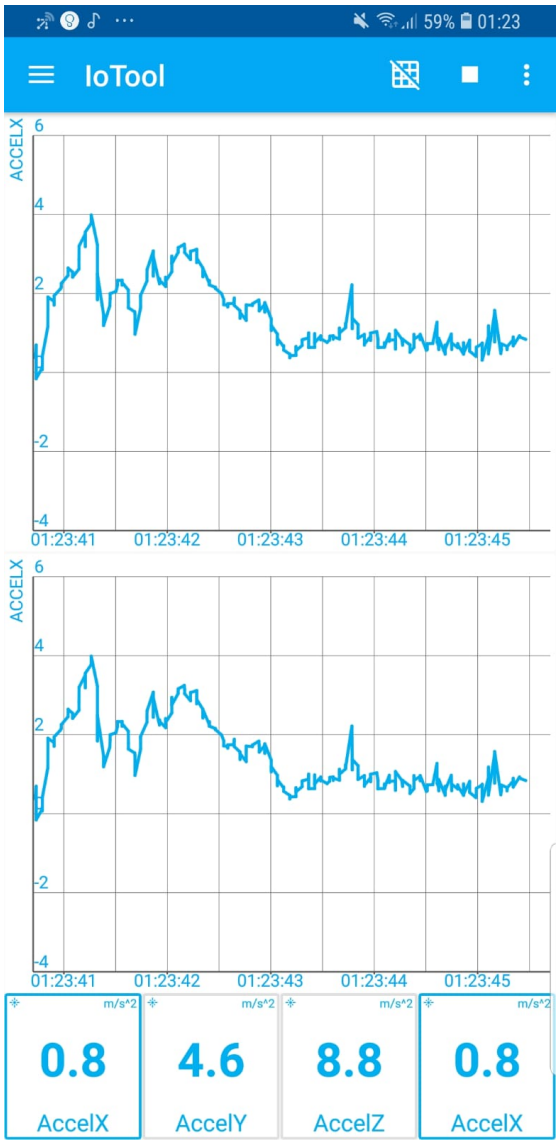

Abbildung 3.4: Echtzeit-Sensordaten Diagramme in der „IoTool Internet of Things (IoT) Dashboard/Gateway“-App

Besonders bei dieser Struktur ist die „But only if“-Bedingung. Jedoch bietet die App nur „oder“ -Operatoren an. Das schränkt die Erstellung von verschiedenen Regeln sehr ein. Der Aufbau und das Design der App sind jedoch einfach und weisen auf eine hohe Usability hin. Allerdings bietet die App keine Möglichkeit an, Geräte und ihre Werte zu überwachen.

Die MBP2Go soll ebenfalls Geräte durch die Erstellung von Regeln steuern. Ähnlich wie die App „Yonomi Smart Home Automation“ soll die Erstellung einfach und übersichtlich sein. Die Geräte werden einzeln konfigurierbar sein. Jedoch soll die MBP kompliziertere Regeln ermöglichen und zusätzliche Operatoren zur Verfügung haben. Zudem wird die MBP auch die Möglichkeit zur Überwachung der Geräte anbieten.

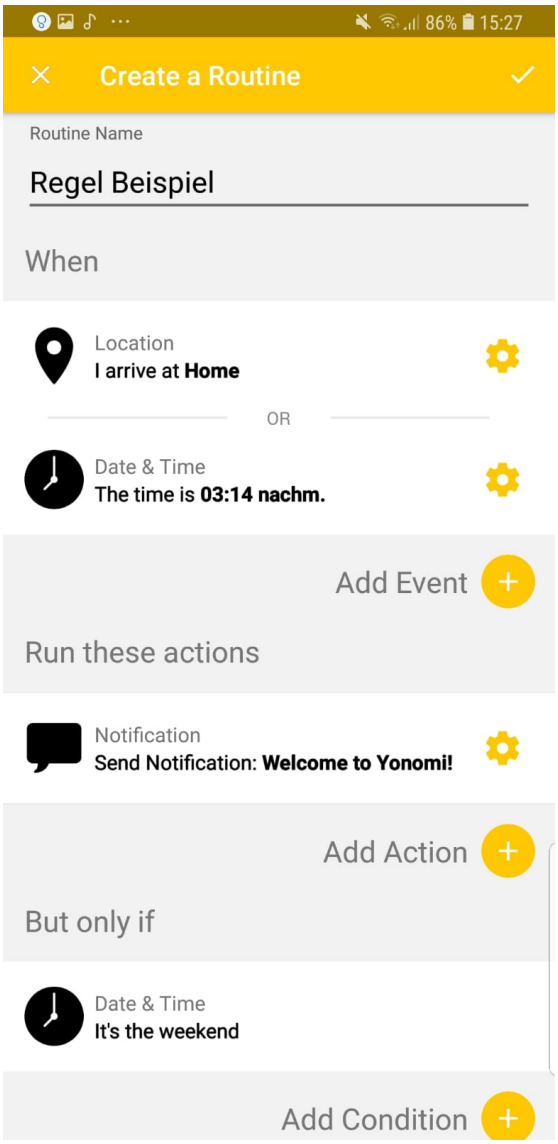

Abbildung 3.5: Erstellung von einer Regel in der App Yonomi Smart Home Automation

Im Playstore gibt es viele IoT-Apps und der Markt floriert. Jedoch zeigt die Fachstudie „Ultimativer Vergleich mobiler IoT-Applikationen“ [10], dass es in dem Bereich noch immer bisher ungelöste Probleme gibt. Viele Apps wurden für eine bestimmte Umgebung, zum Beispiel Smart Home, konstruiert und sind nicht kompatibel mit anderen Umgebungen oder Systemen. Das Registrieren von Geräten stellt für viele Apps ein Problem dar. Einige Apps unterstützen nur bestimmte oder ihre speziell zugehörigen Geräte. Es ist ebenfalls aufgefallen, dass Apps, die sowohl Steuerung und Überwachung von Geräten und das Registrieren von verschiedenen Geräten anbieten, keine schöne grafische Oberfläche besitzen. Außerdem konzentrieren sich zahlreiche Apps entweder auf das Überwachen oder das Steuern von Geräten. Dabei ist die Anzahl von Apps, die sich auf das Steuern von Geräten mittels Complex Event Processing konzentrieren, auffallend gering. Die Fachstudie [10]

zeigt auch, dass etliche Apps im Playstore noch in der Beta Phase sind. Die MBP2Go soll Lösungen für diese Probleme anbieten. Sie ermöglicht das Anbinden von verschiedenen Geräten. Außerdem wird es mit der App möglich sein IoT-Umgebungen zu Überwachen und zu Steuern [10].

4 MBP2Go+: Erweiterung einer mobilen Applikation

Dieses Kapitel beschreibt die Erweiterungen der MBP2Go. Zuerst wird ein Überblick über die Architektur der MBP2Go mit ihrer Erweiterung gegeben. Die MBP2 Go wird um Funktionen erweitert, die es ermöglichen IoT-Umgebungen zu überwachen und zu steuern. In Abschnitt 4.2 werden die Schritte erläutert, die gemacht wurden, damit IoT-Umgebungen überwacht werden können. In Abschnitt 4.3 wird erläutert, welche Möglichkeiten die MBP2Go für die Steuerung der IoT-Umgebungen anbietet.

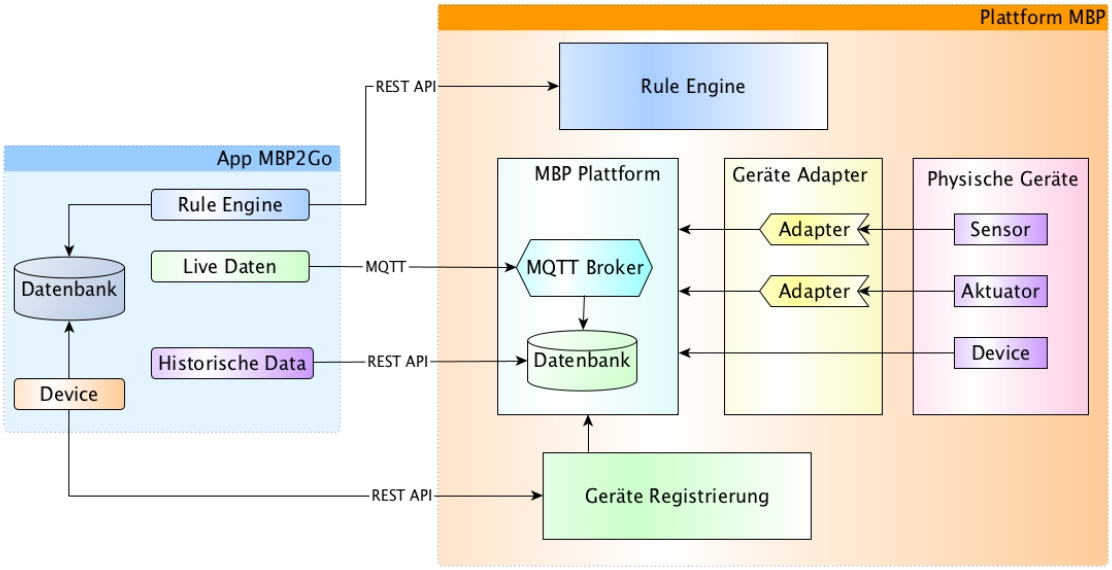

Abbildung 4.1: Übersicht der Architektur der MBP2Go und ihrer Kommunikation mit der Plattform MBP

4.1 Architektur der MBP2Go

Die MBP2Go wurde um folgenden drei Komponenten erweitert: den *Live Daten*, den *Historischen Daten* und der *Rule Engine*. Abbildung 4.1 gibt einen Überblick über die neuen Komponenten der MBP2Go und den Zusammenhang mit der MBP. Die Komponenten sind voneinander unabhängig.

Die Komponenten *Live Daten* und *Historische Daten* gehören zu dem Bereich Überwachung, da diese ein Monitoring der Sensordaten ermöglichen. Dabei wird in der Komponente *Live Daten* die Beobachtung der Echtzeit-Sensordaten behandelt. Die Komponente *Historische Daten* ermöglicht es eine Übersicht der historischen Daten zu erhalten und diese zu analysieren. Im Bereich der Steuerung wurde die Komponente *Rule Engine* entwickelt. Diese ist für die Erstellung von Regeln verantwortlich. Zudem ist diese Komponente für die Kommunikation mit der zukünftigen Rule Engine der MBP verantwortlich. Da die Rule Engine der MBP in der Entwicklung ist, wurde ein Prototyp der Rule Engine, der die Regeln auswertet, entwickelt. In Abbildung 4.1 ist ersichtlich, wie die Komponenten mit der Plattform MBP kommunizieren. Für die Echtzeit Sensordaten wird das Netzwerkprotokoll MQTT verwendet, da dieses für Echtzeit-Datenerfassung gut geeignet ist. Näheres wird in Unterabschnitt 4.2.1 behandelt. Die *Historische Daten* Komponente bekommt ihre Daten durch einen Aufruf der REST-API, die ihrerseits die Daten aus der Datenbank holt. Mit dessen Hilfe kommuniziert diese Komponente mit der Datenbank der MBP. In der Komponente *Rule Engine* werden Regeln erstellt und in der MBP2Go gespeichert. Durch eine REST-API übermittelt sie die Regeln an die Rule Engine der MBP, die diese Regeln auswertet. Diese Komponente beinhaltet zurzeit einen Prototyp einer Rule Engine, um die Regeln auswerten zu können.

4.2 Überwachung

Sensoren können eine riesige Menge an Daten erzeugen. Dabei können diese Daten in Echtzeit aufgezeichnet werden. Die Sensordaten können dann gesammelt und in einer Datenbank gespeichert werden. Diese Schritte erfüllt die Plattform MBP. Die angelegten Sensoren schicken, nachdem sie angebunden sind, ihre Sensordaten und werden in der Datenbank der MBP gespeichert.

Die MBP2Go soll, wie die MBP, die Möglichkeit anbieten, Geräte zu überwachen. Dabei versteht man unter dem Begriff „Überwachung“ die Übersicht der Sensordaten von Geräten. Zudem ist es wichtig, dass die Werte auf eine anschauliche Art dargestellt werden, damit der Nutzer schnell eine Übersicht bekommt. Die Überwachung wurde dabei in zwei Kategorien unterteilt. Die erste Kategorie ist Live Daten. Diese behandelt die Darstellung von Echtzeit-Sensordaten. Dabei wird auf das Layout der MBP2Go eingegangen, die eine gute Usability ermöglicht. Zudem werden verschiedene Bibliotheken, die das Visualisieren von Echtzeit-Daten in mobilen Applikationen ermöglichen miteinander verglichen. Weiterhin wird

Abbildung 4.2: Aufbau des Layouts der Komponente *Live Daten*

erläutert durch welches Netzwerkprotokoll die App MBP2Go die Echtzeit-Daten erhält und welche Vorteile das mit sich bringt. Zugleich wird die Implementierung der Komponente *Live Daten* erklärt.

Die zweite Kategorie Historische Daten beschreibt, wie vergangene Daten visualisiert werden können. Es wird auf die Unterschiede zu der Komponente *Live Daten* eingegangen. Dabei wird eine grundlegende Analyse durchgeführt, um eine schnelle Übersicht über die Datenmenge zu erhalten.

4.2.1 Live Daten

Die Komponente *Live Daten* ist für die Visualisierung der Echtzeit-Sensordaten verantwortlich. Dabei versteht man unter Echtzeit, dass die Informationen unmittelbar nach dem Erfassen bereitgestellt werden. Zudem erfolgt die Aktualisierung ohne Verzögerung [19].

Das heißt, diese Komponente ermöglicht es, ohne große Verzögerung, Sensordaten der registrierten Sensoren darzustellen. Dadurch ergeben sich viele Vorteile. Der erste Vorteil ist, dass der Nutzer von MBP2Go eine schnelle Übersicht bekommt, welche Sensoren aktiv sind. Der zweite Vorteil ist, dass ein Überblick der Sensorendaten vermittelt wird. Ein weiterer Vorteil ist, dass Fehlmessungen schnell erkannt werden können.

Im Fokus lag insbesondere die Usability und User Experience. Wie in Kapitel 3 erwähnt, gibt es IoT-Apps, die eine Überwachung von Sensordaten in Echtzeit anbieten. Jedoch wiesen diese Apps eine schlechte Usability auf. Der Nutzer der MBP2Go sollte bei der Verwendung eine schnelle Übersicht über die registrierten Geräte und Sensoren bekommen. Somit war die erste Aufgabe ein sinnvolles Layout zu entwerfen, damit die Handhabung der App einfach bleibt.

Der Aufbau der Komponente sollte es ermöglichen, innerhalb der Activity auf alle Geräte zugreifen zu können. Eine Activity ist eine Bildschirmseite der App [1]. Zudem musste das Layout sehr dynamisch sein, da die Anzahl der Sensoren eines Gerätes unterschiedlich sein kann. In Abbildung 4.2 ist der Aufbau des Layouts der Komponente *Live Daten* zu sehen. Oben können die Geräte ausgewählt werden. Damit man auf die Geräte zugreifen kann, wird die Datenbank abgefragt. Die MBP2Go benutzt eine SQLite Datenbank, in der die Geräte und ihre Sensoren abgefragt werden. Nach der Auswahl eines Gerätes werden bei den „Tabs“ die jeweiligen Sensoren dargestellt. Dabei werden die „Tabs“ dynamisch erstellt, da jedes Gerät eine unterschiedliche Anzahl an Sensoren besitzt. Die Generierung erfolgt ebenfalls durch Anfrage der Datenbank der MBP2Go. Für jeden einzelnen „Tab“ und damit jeden Sensor des ausgewählten Geräts, wird ein Fragment erstellt. Innerhalb dieses Fragments findet die Darstellung der Echtzeit-Sensordaten statt. Ein Fragment ist dabei nicht dasselbe wie eine Activity. Eine Activity beschreibt die dargestellte Bildschirmseite einer App. Das heißt, jede neue Bildschirmansicht einer App ist eine neue Activity. In der MBP2Go ist zum Beispiel die Liste der registrierten Geräte und die Ansicht zur Registrierung neuer Geräte jeweils eine eigene Activity. Ein Fragment dagegen ist nur ein modularer Teil einer Activity. Durch das Aufkommen von Tablets, stießen Activities schnell an ihre Grenzen [13]. Tablets haben im Vergleich zu Smartphones eine größere Bildschirmgröße. Fragments ermöglichen es die Benutzeroberfläche zu gestalten, ohne dass die Activity gewechselt werden muss. Zudem ist das Besondere an Fragments, dass diese unabhängig voneinander geladen werden können. Außerdem sind Fragments auch unabhängig vom Layout und können jeweils mit verschiedenen Funktionalitäten ausgestattet werden. Da Fragments Teil einer Activity sind, sind sie innerhalb einer Activity „aktiv“ und somit ideal für ein dynamisches Layout [3].

In jedem Fragment sollte die Visualisierung der Echtzeit Sensordaten stattfinden. Die Sensordaten sollten dabei innerhalb eines Diagramms angezeigt werden. Deshalb wurde für die Darstellung der Sensordaten eine Android-Bibliothek gesucht, die die folgenden Anforderungen erfüllt. Eine wichtige Anforderung war, dass die Bibliothek ein benutzerfreundliches und gut aussehendes Diagramm darstellen kann. Des Weiteren war eine weitere Anforderung, dass die Bibliothek Open Source ist. Die Bibliothek sollte zudem einfach in der Anwendung sein und die Möglichkeit anbieten schnell viele Anpassungen zu verwirklichen. Die letzte Anforderung war, dass die Bibliothek Echtzeit-Daten unterstützt.

Die Suche nach einer Bibliothek, die diese Anforderung erfüllte, ergab dabei drei folgende Ergebnisse: Graphview [23], MPAndroidChart[31] und Highcharts[20]. Alle drei Libraries wurden für die MBP2Go getestet. Dabei sollte das Diagramm wie folgt visualisiert werden.

Das Diagramm sollte eine Liniendiagramm-Darstellung besitzen. Die y-Achse zeigt den bekannten Wert eines Sensors an und die x-Achse gibt die Uhrzeit an. Falls keine aktuellen Werte gemessen werden, bleibt das Diagramm auf dem letzten gemessenen Wert.

Graphview ist eine Open Source Graphen Bibliothek für Android [23]. Die Bibliothek zeichnet sich durch die einfache Anwendung, schnelle Integration und Anpassung aus. Mit ihr kann man verschiedene Diagramme zum Beispiel Linien-Diagramme oder Balken-Diagramme visualisieren. Die Bibliothek wird von über 4000 Apps verwendet. Beispiele dafür sind die Apps Peak Flow und Sound Meter [23]. Des Weiteren gehört die Bibliothek zu einer der wenigen Android Libraries, die Echtzeit Daten darstellen kann. Dabei werden die Daten zur Laufzeit geändert. Jedoch erfüllt die Bibliothek nicht die Bedingung der Benutzerfreundlichkeit. Die App bietet keine flüssige Animation während die Daten in das Diagramm hinzugefügt werden. Zudem wirkt das Diagramm visuell nicht ansprechend für Anwender. Außerdem ergaben sich bei der App Probleme, als die x-Achse die aktuelle Uhrzeit aufzeigen sollte. Die x-Achse zeigte lediglich nur einen Zähler auf und zählte angefangen von 1 hoch. Die Dokumentation von Graphview zeigte keine Lösung für dieses grundlegende Problem auf. Zudem wären zu viele Anpassungen nötig gewesen, damit man ein grafisch ansprechendes Diagramm implementiert hätte können. In Abbildung 4.3 zeigt das linke Bild das Ergebnis mit der Bibliothek Graphview.

Die Bibliothek MPAndroidChart ist ebenfalls ein Open Source Projekt. Dabei verspricht die App eine leistungsstarke und benutzerfreundliche Bibliothek für die Erstellung von Diagrammen. Auch mit dieser Bibliothek können verschiedenste Diagramme erstellt werden. Jedoch wird in der Dokumentation darauf verwiesen, dass die Bibliothek nicht offiziell dynamische Daten und damit auch Echtzeit Daten unterstützt. Stattdessen verweist sie auf die Bibliothek SciChart Android [33]. Diese erfüllt jedoch die Bedingung nicht, kostenfrei und Open Source zu sein. Die MPAndroidChart Bibliothek bietet trotzdem Möglichkeiten an Echtzeit-Daten zu visualisieren. Auch hier ließ sich die x-Achse schwer anpassen. Die Zeitangaben wurden in Sekunden dargestellt und es bräuchte tiefgreifende Änderungen für die gewünschte Darstellung. Die Animation der Echtzeit-Daten war zu dem sehr stockig. Jedoch war ein Vorteil, dass die x-Achse eine Scrollfunktion beinhaltet, wodurch die vergangenen Sensordaten beobachtet werden können. Es wären jedoch viele Anpassungen nötig, damit die Bibliothek ansehnlich wird. In Abbildung 4.3 zeigt das mittlere Bild das Ergebnis mit MPAndroidChart.

Schließlich wurde die Bibliothek Highcharts, die schon in MBP verwendet wurde, verwendet. Highcharts ist ebenfalls eine Bibliothek für Diagramme. Die Libary wirbt dabei damit, dass sie von über 10 000 Entwicklern [20] verwendet wird und sehr einfach zu handhaben ist. Highcharts wurde mit JavaScript entwickelt und ist somit speziell auf Webseiten ausgerichtet. Um die Bibliothek in der MBP2Go einzubinden gab es zwei Möglichkeiten. Die erste Möglichkeit ist die Verwendung des Android Wrapper Highcharts [21]. Dieser ist ebenfalls

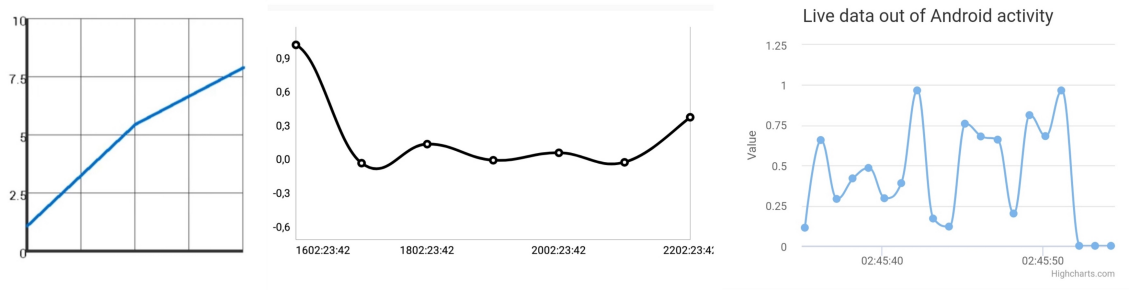

Abbildung 4.3: Überblick der Unterschiede zwischen den Diagrammen. Dabei sieht man von links nach rechts die Diagramme von Graphview, MPAndroidChart und Highcharts

Open Source und ermöglicht es Highcharts in mobilen Applikationen einzubauen. Jedoch ergaben sich einige Probleme bei der Verwendung des Wrappers. Der Wrapper stellte die Echtzeit-Funktionalitäten von Highchart nicht zur Verfügung. Dadurch konnte der Android Highcharts Wrapper nicht für die Komponente *Live Daten* verwendet werden. Die zweite Möglichkeit für die Einbindung der Highcharts in die MBP2Go ist das Implementieren eines eigenen Wrappers. Android bietet durch die assets [14] und der Webview [16] die Möglichkeit an, Javascript Code einzubinden. Durch den so selbst entwickelten Wrapper konnten somit die Echtzeitfunktionalitäten von Highcharts verwendet werden. Die Einbindung von Highcharts für die Echtzeitfunktionalitäten in die MBP2Go erwies sich als nicht einfach. Jedoch zeichnet sich Highcharts durch die visuelle Darstellung und einfache Anwendung aus. Die x-Achse konnte einfach die richtige aktuelle Uhrzeit anzeigen. Sie verfügt über eine schöne Animation. Die Bibliothek erfüllt somit die gestellten Anforderungen und wurde für die tatsächliche Implementierung genutzt. In Abbildung 4.3 zeigt das rechte Bild das Ergebnis mit Highcharts.

Um die Echtzeit-Sensordaten von der MBP zu erhalten gibt es jeweils zwei Möglichkeiten. Die erste Möglichkeit ist eine REST-API Anfrage an die MBP. Die MBP sichert die erfassten Sensordaten in einer Datenbank. Mit der Rest-API könnte man zum Beispiel jede Sekunde eine Anfrage stellen und so die Daten erhalten. Die Nachteile dieses Konzepts wären jedoch eine mögliche Überlastung des Systems mit einer Vielzahl von Anfragen und eine weit größere Latenzzeit. Zudem würden zu viele Nachrichten versendet werden, die auf eine Rücknachricht warten.

Die zweite Möglichkeit Sensordaten in Echtzeit zu erhalten ist durch das Protokoll MQTT. In Android existiert die Bibliothek Eclipse Paho Android Service [9] für die Verwendung von MQTT [9]. MQTT bietet im Vergleich zu Rest-Api Anfragen viele Vorteile, da es leichtgewichtig ist, wenig Overhead produziert und kein Polling notwendig ist. MBP2Go registriert sich auf den MQTT Broker. Sobald ein neuer Sensorwert gemessen wird, bekommt die MBP2Go den gemessenen Wert und zeigt diesen an. Sobald sich das Fragment ändert, subscribed sich die MBP2Go neu auf den MQTT Broker der MBP. Dabei wird das Diagramm neu erstellt. Die Verbindung mit dem MQTT Broker sollte hierbei nur innerhalb der Activity stattfinden. Activitys besitzen Lebenszyklen [1], weshalb es wichtig ist, sobald

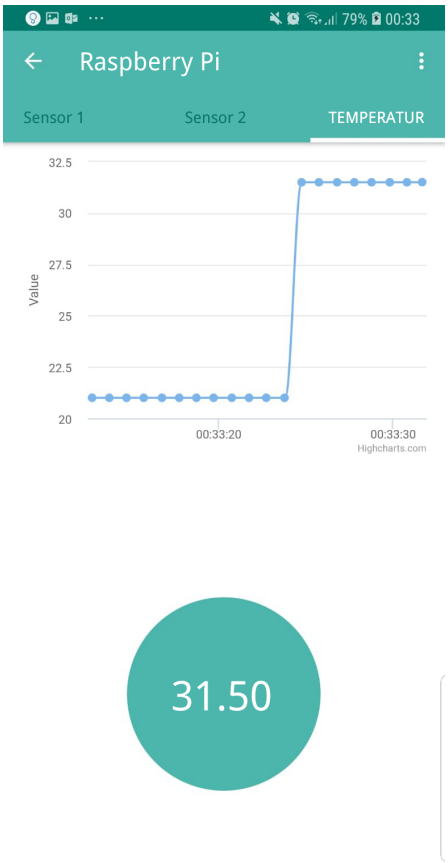

Abbildung 4.4: Endergebnis der Komponente *Live Daten*

sich die Activity ändert, dass sich die MBP2Go vom MQTT Broker unsubscribed. Das heißt, keine weiteren Echtzeit Sensordaten erhält. Abbildung 4.4 zeigt die Umsetzung der Komponente *Live Daten*. Damit der Anwender schnell einen Überblick des aktuellsten Sensors bekommt, wird der aktuellste gemessene Sensorwert groß dargestellt.

4.2.2 Historische Daten

Die Komponente *Historische Daten* ist für die Visualisierung und Darstellung von historischen Sensordaten verantwortlich. Dabei wurde bei der Komponente *Historische Daten* das selbe Layout wie in Unterabschnitt 4.2.1 verwendet, da die Bedingungen an das Layout die selben sind. Jedoch gibt es grundlegende Unterschiede im Inhalt der Fragments. In der Komponente sollte wie bei den *Live Daten* ein Diagramm dargestellt werden. Für die Implementierung des Diagramms wurde derselbe Wrapper wie in Live Daten verwendet. Das Diagramm visualisiert die historischen Daten der Sensoren. Dabei beschreibt die

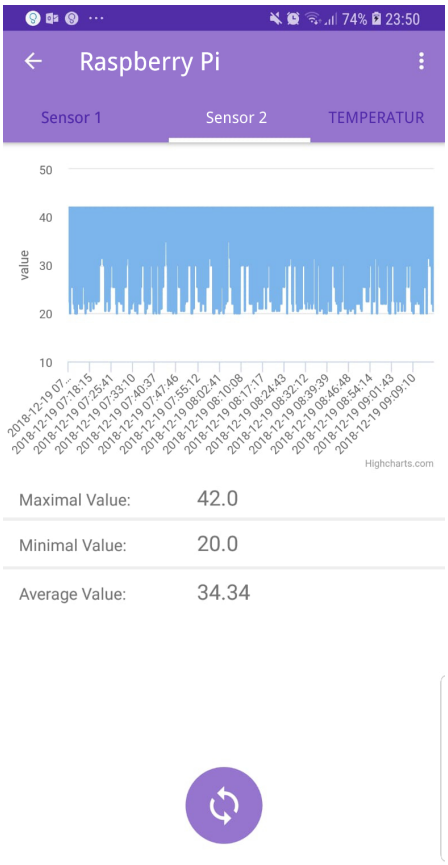

Abbildung 4.5: Überblick der Komponente *Historische Daten*

y-Achse die Sensorenwerte und die x-Achse die Uhrzeit mit dem jeweiligen Datum. Das heißt, es zeigt die Werte der Sensoren von einer bestimmten Zeitspanne an. Die Vorteile durch die Komponente sind dabei vielseitig.

Ein Vorteil dabei ist, dass eine schnelle Übersicht der historischen Sensordaten gegeben wird. Dadurch können Fehlmessungen über einen längeren Zeitraum beobachtet werden. Des Weiteren ist es durch die Datumsangabe möglich zu erkennen, wann ein Sensor zuletzt aktiv war. Zudem kann man durch das Diagramm die Messungen mit dem Zeitpunkt des Auftretens ablesen. In Abbildung 4.5 ist das Layout der Komponente *Historische Daten* zu sehen. Der Refreshknopf führt die Aktualisierung des jeweiligen Fragments aus.

Ein weiterer Unterschied zwischen der *Historischen Daten* Komponente und der *Live Daten* Komponente ist die Kommunikation mit der MBP. Wie in Unterabschnitt 4.2.1 erwähnt, wurde für die Echtzeitdatenerfassung MQTT verwendet. Die Vorteile von MQTT sind jedoch in der Komponente *Historische Daten* nicht notwendig. Die Komponente verwendet eine REST-API, die von der MBP [26] zur Verfügung gestellt wird, um die Daten der Sensoren zu ermitteln. Diese ermöglicht es historische Daten aus der Datenbank der MBP zu erhalten. Anders als bei der *Live Daten* Komponente muss bei der *Historischen Daten*

Komponente jeweils nur eine REST-API Anfrage pro Sensor gestellt werden. Dadurch bekommt die MBP für den jeweiligen Sensor die letzten 10 000 Sensorendaten. Die Zahl der Sensorendaten wurde auf 10 000 begrenzt, um die Übersichtlichkeit des Diagramms zu garantieren.

Schließlich wird auf den Daten eine Analyse vorgenommen, damit eine Übersicht der Daten erleichtert wird. Es werden dabei der Minimalwert, Maximalwert und der Mittelwert berechnet und angezeigt.

4.3 Steuerung

Für die Steuerung von IoT-Geräten wurde die MBP2Go in zwei Bereichen erweitert. Zuerst wurde die Schnittstelle definiert, über die Regeln in der MBP hinzugefügt werden können. Dafür wird das Konzept von Regeln und die Struktur von Regeln erläutert. Zudem wird das Layout der App für die Erstellung von Regeln beschrieben. Da die Rule Engine in der MBP noch in der Entwicklung ist, wird ein entwickelter Prototyp einer Rule Engine, der innerhalb der MBP2Go läuft, erläutert. Dieser wird zukünftig von der Rule Engine der MBP ersetzt. Unter dem Begriff Rule Engine versteht man ein ereignisgesteuertes System, das für die Erstellung von Ereignissen und das Reagieren auf Ereignisse genutzt werden kann. Die nachfolgenden Kapitel führen in diese Thematiken ein.

4.3.1 Konzepte von Regeln

Für die Steuerung von IoT-Geräten werden Regeln benötigt. Dabei können Regeln verschiedene Operatoren und Strukturen beinhalten. Deshalb wird zuerst ein Sprachkonstrukt vorgestellt, das alle für MBP2 Go gewünschten Regeln beschreiben kann. Für MBP2 Go ist eine *Regel* eine spezifische Aktion, die bei der Erkennung eines bestimmten Musters ausgelöst werden soll. Dabei besteht jede Regel aus zwei Teilen. Der erste Teil beschreibt die Bedingungen. Bedingungen können dabei aus einem oder mehreren *Mustern* bestehen. Diese Muster bestehen aus einer Sequenz von Ereignissen, die miteinander verknüpft sind. Wenn diese Muster von Ereignissen zutreffen, wird der zweite Teil einer Regel ausgeführt, der Aktionsteil [4]. Der Aktionsteil führt eine Aktion als Reaktion auf die vorgekommenen Ereignisse aus. Aktionen können zum Beispiel das Ausführen eines Dienstes sein. Eine Regel kann zum Beispiel folgende Struktur haben:

Wenn die Temperatur über 30 Grad ist, dann öffne die Fenster.

Dabei kann zum Beispiel die Ereignissinstanz „Temperatur über 30 Grad“ als A bezeichnet werden und die Ereignissinstanz „öffne die Fenster“ als B. Diese Instanzen können miteinander durch Operatoren verknüpft werden.

Die booleschen Operatoren $\wedge$ und $\vee$ verknüpfen, unabhängig von der Reihenfolge, zwei Ereignisse. Dabei bedeutet $\wedge$, dass alle Ereignisse, die damit verknüpft sind, auftreten müssen. Eine Beispiel-Regel mit einem $\wedge$ kann folgendermaßen aussehen:

Wenn die Temperatur 30 Grad ist $\wedge$ es regnet dann öffne das Fenster

Das Fenster wird nur geöffnet, falls beide Ereignisse zutreffen. Ein $\vee$ beschreibt, dass ein Ereignis in der Verknüpfung auftreten muss. Zum Beispiel kann die obige Beispiel-Regel folgendermaßen lauten:

Wenn die Temperatur 30 Grad ist $\vee$ es regnet dann öffne das Fenster

Diese Regel trifft bereits zu, falls nur eines der Ereignisse auftritt. Der Negationsoperator $\neg$ beschreibt, dass ein bestimmtes Ereignis nicht auftreten darf. Eine Beispiel-Regel für den Negationsoperator ist:

Wenn es $\neg$regnet dann öffne das Fenster

Das heißt, das Fenster wird geöffnet, wenn es nicht regnet [2]. Jedoch reicht es oft nicht Regeln allein durch Verknüpfung der Operatoren zu beschreiben. Oft muss der Kontext einer Regel interpretiert werden. Dafür gibt es Kontextbedingungen [4], die den Inhalt, also den Attributwert, einer Regel definieren. Für die Auswertung der Kontextbedingungen ist ein Datentyp eines Attributwerts notwendig. So können für numerische Datentypen folgende Kontextoperatoren verwendet werden: +,–, /, ·, $\neq$, =, >, <, $\leq$, $\geq$ [17]. Dadurch kann die Wetter-Regel folgendermaßen beschrieben werden:

Wenn Temperatur > 30 dann öffne das Fenster.

Um die Regeln lesbarer zu gestalten wird noch der „.“-Operator eingeführt. Dieser beschreibt den Zugriff auf das Attribut eines Ereignisses. Zum Beispiel kann die Regel folgendermaßen aussehen:

Wenn Temperatur.getCelsius > 30 dann Fenster.open

Der Sequenzoperator $\rightarrow$ [4] legt die zeitliche Reihenfolge von Ereignissen fest. Das heißt, eine Ereignisinstanz A muss vor der Ereignisinstanz B stattfinden. Zudem muss die Ereignisinstanz B gelten.

Damit man Ereignisinstanzen mehrfach verwenden kann, benötigt man Aliasnamen. Mit diesen ist es möglich, Ereignisinstanzen für verschiedene Instanzen zu verwenden. Dadurch kann man Regeln in der folgender Form beschreiben: „Wenn die Wohnzimmer-Temperatur über 30 Grad ist oder die Außen-Temperatur unter 20 Grad ist.“. Bei diesem Beispiel werden zwei verschiedene Instanzen verwendet. Sowohl die Wohnzimmertemperatur als auch die Außentemperatur sind verschiedene Ereignisinstanzen. Damit diese beschrieben werden können, verwendet man Aliasnamen mit dem Schlüsselwort „AS“Dadurch kann die Beispielregel neu beschrieben werden. Die Wohnzimmertemperatur wird als Ereignisinstanz A und die Außentemperatur als B beschrieben. Somit kommt man auf das Sprachkonzept A as a und B as b [4].

Durch diese Operatoren können einfach erkennbare Regeln definiert werden. Eine einfache Regel ist zum Beispiel eine Ereignisinstanz oder die Verknüpfung von Ereignisinstanzen mit einem booleschen Operator. In Kapitel 3 wurde erläutert, dass viele Apps Regeln ermöglichen, jedoch können diese nur einfach erkennbare Regeln darstellen [4]. Eine einfach erkennbare Regel kann beispielsweise folgendermaßen aussehen: „Wenn es regnet und es über 30 Grad ist, dann öffne das Fenster“.

Jedoch kann man die Regeln noch erweitern und somit komplexe erkennbare Regeln definieren. Eine komplexe erkennbare Regel beinhaltet die Reihenfolge oder das Zeitfenster einer Regel. Durch den Sequenzoperator können komplexe Regel definiert werden. Eine Beispielregel für eine komplexe erkennbare Regel kann folgendermaßen aussehen:

Wenn es über 30 Grad ist → es über 35 Grad ist dann öffne das Fenster.

Das bedeutet, wenn auf eine Ereignisinstanz über 30 Grad eine Ereignisinstanz über 35 Grad folgt, wird das Fenster geöffnet. Regeln können zudem mit einem Zeitfenster versehen werden [4].

Diese werden benötigt, da kontinuierlich neue Ereignisse entstehen können und die Menge der Ereignisse somit unbegrenzt wachsen würde. Damit die Echtzeitfähigkeit und die Effizienz der Regeln weiterhin bestehen bleibt, wird ein Zeitfenster benötigt, mit dem die Gesamtmenge aller Ereignisse auf die wenigen Ereignisse innerhalb des Zeitfensters eingeschränkt werden kann [17]. Diese Zeitfenster werden auch Sliding Windows genannt. Sliding Windows grenzen die Anzahl der betrachteten Menge von Ereignissen ein. Dafür werden zwei Arten von Sliding Windows definiert. Diese sind Längen- und Zeitfenster. Längenfenster beschreiben, welche Anzahl der letzten aufgetreten Ereignisse betrachtet werden sollen. Wenn zum Beispiel das Längenfenster die Länge 5 hat und ein Temperatursensor 10 Werte gemessen hat, werden nur die letzten 5 aufgezeichneten Ereignisse betrachtet. Ein Sliding Window bezeichnet man als Zeitfenster. Diese bezeichnen Ereignisse, die in einem bestimmten Zeitraum aufgetreten sind. Ein Beispiel für ein Zeitfenster ist die Zeitspanne von 5 Minuten vor der Auswertung der Regel bis zu dem Zeitpunkt der Auswertung. Dabei kann die Anzahl der tatsächlichen Ereignisse in einem Zeitfenster variieren [4].

Durch Sliding Windows wird nicht nur die Menge der Ereignisse eingeschränkt. Mit ihnen können Ereignisse zeitlich zusammengefasst werden. Damit Attributwerte einer bestimmten Menge von Ereignissen zusammengefasst werden können, existieren Aggregationsfunktionen. Bekannte Aggregationsfunktionen sind zum Beispiel die Berechnung der Summe oder des Durchschnittswerts aller Ereignisse [4]. Aggregationsfunktionen wurden zum Beispiel bei der Analyse in Unterabschnitt 4.2.2 verwendet. Dort wurde von 10 000 Werten der Durchschnitt, Maximalwert und Minimalwert berechnet.

Ist der Bedingungsteil erfüllt, wird der Aktionsteil ausgeführt. Dabei beschreibt der Aktionsteil die Behandlung von einer Sequenz erfüllter Ereignisse und die darauf folgende Reaktion. Dabei kann man den Aktionsteil auf zwei Arten unterscheiden. Die erste Art ist die eigenständige Generierung von neuen Ereignissen. Die neu generierten Ereignisse können dabei Daten des erfüllten Bedingungsteil aufbereiten oder neue Ereignisse mit den

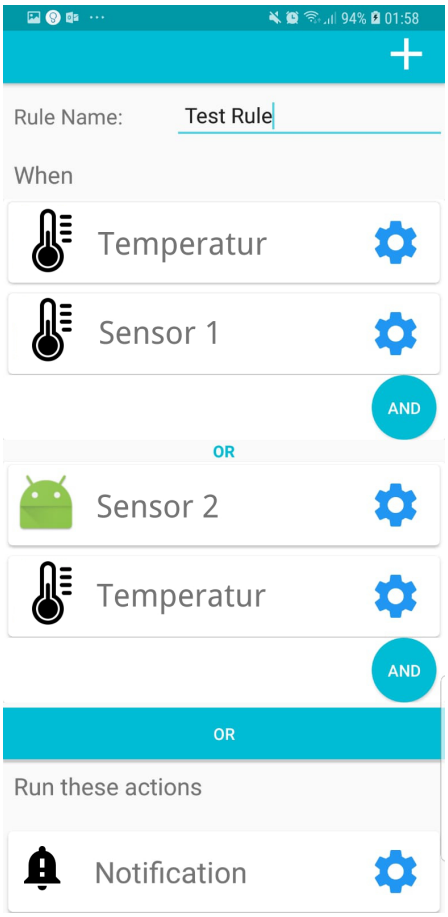

Abbildung 4.6: Layout von der Rule Engine der MBP2Go für die Erstellung von Regeln

erhaltenen Erkenntnissen generieren. Ein Beispiel für die Generierung eines neues Ereignis im Aktionsteil ist, wenn durch einen Temperatursensor und einen Helligkeitssensor die neue Erkenntnis, dass die Sonne scheint, gewonnen wird. Das neue erzeugte Ereignis wird dann wieder an die Rule Engine geleitet, damit es weiter verarbeitet werden kann. Die zweite Art eines Aktionsteils beschreibt das Aufrufen von eines Dienstes. Das heißt, sobald der Bedingungsteil erfüllt ist, wird ein Dienst ausgeführt. Ein Beispiel für das Aufrufen eines Dienstes ist, dass eine E-Mail verschickt wird, nachdem der Bedingungsteil erfüllt ist.

4.3.2 Definition einer Schnittstelle für die MBP

Die MBP2Go erstellt ebenfalls Regeln, die dann die zukünftige Rule Engine der MBP auswerten kann. Dabei ergaben sich drei Aufgaben für die Erstellung von Regeln. Als Erstes muss festgehalten werden, aus welchen Operatoren Regeln gebildet werden und welche generelle Struktur Regeln annehmen sollen. Die zweite Aufgabe bestand darin ein

Layout zu entwerfen, das die Erstellung von Regeln ermöglicht. Die dritte Aufgabe war ein Format für die Kommunikation der Rule Engine der MBP zu finden, mit dem Regeln beschrieben werden können.

In Unterabschnitt 4.3.1 werden die verschiedenen möglichen Operatoren und Strukturen von Regeln erläutert. Die MBP2Go ermöglicht das Konstruieren von einfach erkennbaren Regeln. Dabei soll die MBP2Go die gängigsten aussagenlogischen Operatoren oder alle aussagenlogische Regeln abbilden können. Hierfür wurde die disjunktive Normalform, bekannt als DNF, verwendet. Eine DNF beschreibt einen logischen Ausdruck, der aus einer Hierarchie von Oder-Verknüpfungen gefolgt von Und-Verknüpfungen besteht. Der logische Ausdruck *(A und B) oder (C und D)* ist ein Beispiel für eine DNF. Mit einer DNF ist es dabei möglich jede aussagenlogische Formel abzubilden. Denn jede aussagenlogische Formel besitzt einen Wahrheitswerteverlauf, der ihr Verhalten eindeutig in einer Wahrheitstafel beschreibt. Eine äquivalente DNF kann man aus einer Wahrheitstafel extrahieren, indem man die mit „wahr“ belegten Zeilen der Wahrheitstafel in den Konjuktionsgliedern codiert. Daher lässt sich mit DNFs die gesamte Aussagenlogik ausdrücken [32].

Damit die beschriebenen Regeln tatsächlich die Gesamtheit der DNFs darstellen kann, wird auch der Negationsoperator benötigt. In der MBP2Go wird dies durch die Existenz der kontextbezogenen Operatoren und ihrer jeweiligen Negationen realisiert. Will man zum Beispiel „Temperatur < 3“ negieren, kann dies durch „Temperatur >= 3“ ausgedrückt werden. Da eine Datentyp-Definition in der MBP fehlt, werden hauptsächlich numerische Datentypen behandelt. Für den Aktionsteil der MBP2Go wurde das Anstoßen von neuen Diensten definiert. Dabei besteht der Aktionsteil aus Und-Verknüpfungen, sodass mehrere Dienste gleichzeitig ausgeführt werden können.

Bei der Erstellung des Layouts wurde auf ein einfaches leicht verständliches Layout geachtet. Abbildung 4.6 zeigt das umgesetzte Layout in der MBP2Go. Dabei sind die Sensoren innerhalb der Cardview [12] mit einem „und“ verknüpft. Beim Hinzufügen eines Sensors werden alle Sensoren angezeigt, die in der SQLite Datenbank der MBP2Go zur Verfügung stehen. Durch das „oder“ wird eine neue Cardview erzeugt, in der wieder Sensoren miteinander mit einem „und“ verknüpft werden können. Sensoren können dabei einzeln konfiguriert werden. Abbildung 4.7 zeigt eine mögliche Konfiguration der Kontextoperatoren für Ereignisse von Sensoren. Bei der Konfiguration gibt man einen Vergleichswert an. Die MBP hat zurzeit keine Datentypen, weshalb die MBP2Go nur numerische Datentypen bearbeitet. Des Weiteren wird ein Kontextoperator angegeben. Die kontextbezogene Operatoren sind dabei für numerische Datentypen. Zudem können Aktionen wie zum Beispiel Aktuatoren oder Push-Nachrichten hinzugefügt werden. Diese werden innerhalb einer Cardview mit einem „und“ verknüpft. Auch Aktionen kann man konfigurieren. Bei einer Nachricht kann man zum Beispiel den Inhalt einer Nachricht definieren. Durch dieses Layout können Regeln, die zum Beispiel die Form *(A und B) oder (C und D)* haben, formuliert werden. In Abbildung 4.6 ist die Beispielregel „*Wenn* der Temperatur Sensorwert über 40 ist *und* der Sensor 1 einen Sensorwert unter 20 hat *oder* der Sensor 2 einen Sensorwert unter 34 *und* der Temperatur Sensorwert über 40 ist, *dann* schicke mir eine Push-Nachricht.“

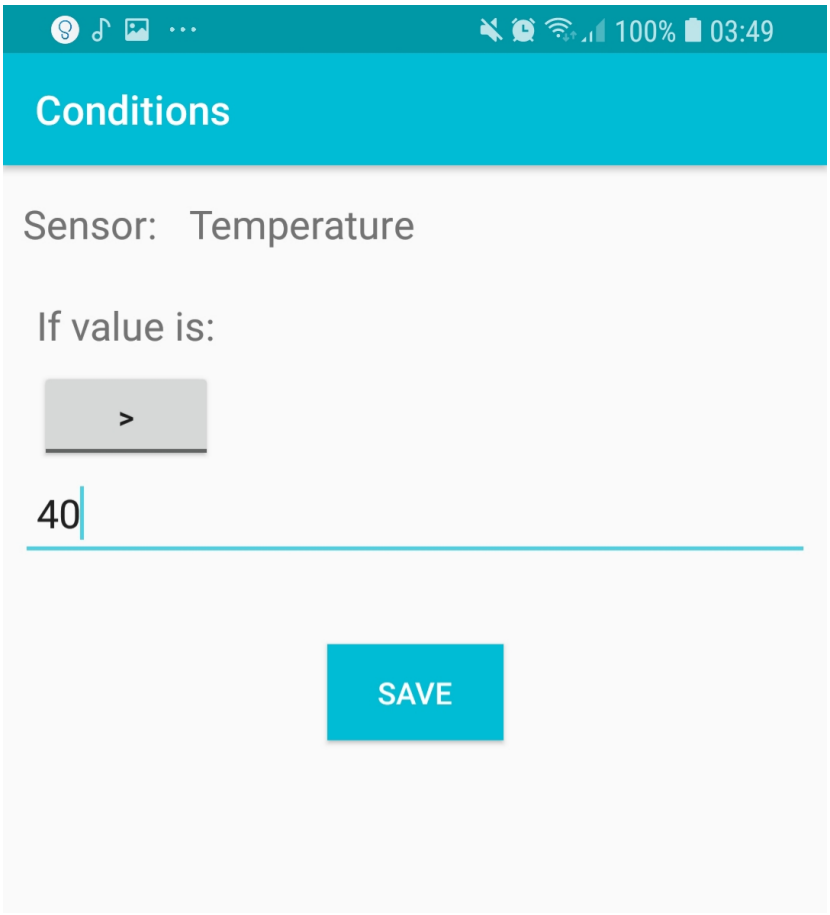

Abbildung 4.7: Layout von der Rule Engine der MBP2 Go für die Konfiguration einer Regel

Nachdem der Aufbau und die Erstellung von Regeln innerhalb der MBP2Go erläutert wurden, wurde ein Datenaustauschformat entworfen, das die Schnittstelle innerhalb der MBP und der MBP2Go definiert. Wie in Abbildung 4.1 erläutert, wird die Komponente *Rule Engine* der MBP2Go in Zukunft mit der Rule Engine der MBP per REST-API kommunizieren. Dabei zeichnet sich eine REST-API durch die folgenden Eigenschaften aus. REST-APIs sind unabhängig von Plattformen und einer Programmiersprache. Zudem zeichnet sich eine REST-API durch seine Zustandslosigkeit aus. Das heißt, dass jede Nachricht die notwendigen Informationen für eine erfolgreiche Verarbeitung beinhaltet [30]. Durch einen Payload können per REST-API die Informationen an die Rule Engine der MBP geschickt werden. Als Datenaustauschformat für den Payload wurde das JavaScript Object Notation [8], kurz JSON, gewählt. JSON ist ein einfach lesbares und schreibbares Datenaustauschformat, welches einfach zu parsen und zu generieren ist. JSON ist, wie die REST-API, unabhängig von der verwendeten Programmiersprache. Zudem bieten viele Programmiersprachen eine Bibliothek mit einem JSON-Parser an.

Listing 4.1 JSON-Format für die Kommunikation zwischen MBP und MBP2Go

```
{
   "association": "( s1 AND s2 ...) OR ( s1 AND s3 ...) ...",
   "conditions": {
      "sensors": [
         {
            "operator": "<",
            "sensorid": "sensorid1",
            "value": "10",
            "rulesensorid": "s1"
         },
         {
            "operator": "<",
            "sensorid": "sensorid2",
            "value": "20",
            "rulesensorid": "s2"
         },
         ...
      ]
   },
   "actions": {
      "actuators": [
         {
            "actuatorid": "actuatorid1",
            "value": "20",
            "ruleactuatorid": "a1"
         },
         {
            "actuatorid": "actuatorid2",
            "value": "100",
            "ruleactuatorid": "a2"
         },
         ...
      ]
   }
}
```

In Listing 4.1 ist das JSON-Format zu sehen, das für die Erstellung einer Regel in der MBP Rule Engine verantwortlich ist. Zudem sind die Objekte aufgeteilt in einen Bedingungsteil und einen Aktionsteil. Der Bedingungsteil besteht aus den Schlüssenwörtern „association“ und dem JSON-Objekt „conditions“. Dabei beschreibt das Schlüsselwort „association“die Verknüpfungen zwischen den Sensoren durch boolesche Operatoren. Innerhalb des JSON-Objekts „condition“ ist ein JSON-Array „sensors“ der die Sensoren als JSON-Objekt beinhaltet. Die jeweiligen JSON-Objekte enthalten dabei die kontextbezogenen Operatoren, die mit dem Schlüsselbegriff „operator“ bezeichnet wurden. Dieser kann jeweils die Operatoren =, >, <, ≤ oder ≥ annehmen. Der Schlüsselbegriff „sensorid“ bezeichnet dabei die eindeutige Id der Sensoren innerhalb der MBP. Der Begriff „value“ beschreibt den Vergleichswert, der vom Nutzer eingegeben wurde. Schließlich ist der Schlüsselbegriff „rulesensorid“ eine eindeutig identifizierbare Id, um Ereignisinstanzen innerhalb einer Regel auseinander halten zu können. Diese Id wird gebraucht, falls man denselben Sensor mehrfach in einer Regel verwenden möchte.

Der Aktionsteil wird beim JSON-Objet „actions“ beschrieben. Dieser beinhaltet das JSON-Array „actuators“ das die Aktuatoren oder Dienste beinhaltet, die beim Zutreffen der Regel ausgelöst werden.

4.3.3 Prototyp einer Rule Engine

Die Komponente *Rule Engine* ist dafür verantwortlich die oben eingeführten Regeln auszuwerten und die entsprechend definierten Aktionen durchzuführen. Im Rahmen dieser Arbeit entstand eine prototypische Implementierung dieser Rule Engine. Dabei ist die Rule Engine für das Verarbeiten, das Auswerten und für das Reagieren auf Ereignisse zuständig.

Bevor die Rule Engine der MBP2Go Ereignisse erkennen kann, müssen Regeln erstellt werden. Diese werden wie in Unterabschnitt 4.3.2 beschrieben erstellt. Nach der Erstellung werden die Regeln in der SQLite Datenbank der MBP2GO gesichert. Die in der Datenbank abgelegten Regeln werden in der MP2GO in einer Übersicht dargestellt. In Abbildung 4.8 ist das Layout mit der Übersicht der erstellten Regeln zu sehen.

Die Regeln werden in einer JSON-Struktur abgebildet. Ein Beispiel für eine solche Regel in ihrer entsprechenden Darstellung ist in Listing 4.2 ersichtlich. Der Schlüsselbegriff „association“ beschreibt dabei den Bedingungsteil. Dieser beinhaltet die Information, mit welchen Operatoren die Sensoren miteinander verknüpft sind. In dem Beispiel in Listing 4.2 sind die Sensoren „Wohnzimmertemperatur“ und „Badezimmertemperatur“ mit einem „und“-Operator verknüpft. Zudem sind sie durch den „oder“-Operator mit dem Sensor „Außentemperatur“ verknüpft. In einem ersten Verarbeitungsschritt werden die aktuellen Sensorinformationen in die entsprechende Regel eingefügt. Dadurch entsteht eine Regel, die dem aktuellen Zustand entspricht und deren Wahrheitswert bestimmt werden soll. Dabei fügt der Parser zusätzlich die Kontextbedingungen der jeweiligen Sensoren der Regel hinzu. Das heißt, die jeweiligen „operators“ und „values“ der Sensoren werden in der association

Listing 4.2 Beispiel JSON-Format das in der Rule Enginge verarbeitet wird

```
{
   "association": "( 1 AND 2 ) OR ( 3 ) ",
   "conditions": {
      "sensors": [
         {
            "operator": "<",
            "sensorid": "Wohnzimmertemperatur",
            "value": "25",
            "rulesensorid": "1"
         },
         {
            "operator": ">",
            "sensorid": "Badezimmertemperatur",
            "value": "28",
            "rulesensorid": "2"
         },
         {
           "operator": ">",
           "sensorid": "Außentemperatur",
           "value": "30",
           "rulesensorid": "3"
        }
      ]
   },
   "actions": {
      "actuators": [
         {
            "actuatorid": "Notification",
            "value": "Temperatur zu hoch",
            "ruleactuatorid": "1"
         }
      ]
   }
}
```

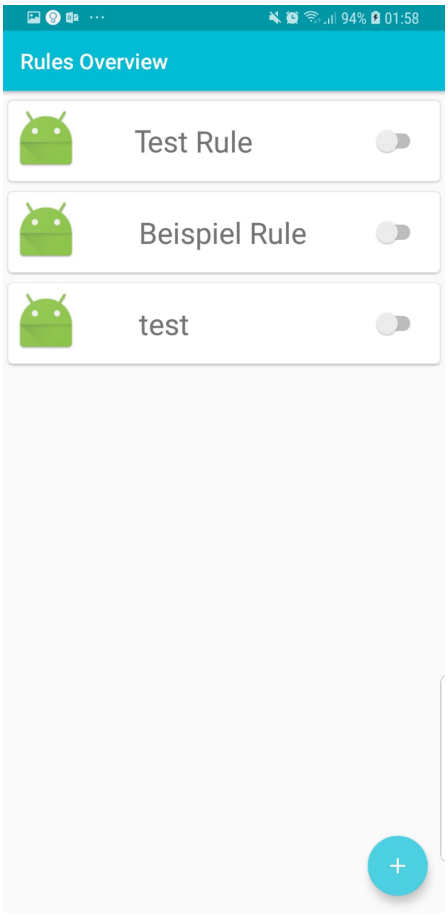

Abbildung 4.8: Layout das einen Überblick der Regeln verschafft

an den entsprechenden Stellen eingefügt. Die Beispiel-Regel von Listing 4.2 sieht nach diesem Schritt wie folgt aus:

$$(1 < 25 \text{ AND } 2 > 28) \text{ OR } (3 < 30)$$

Bei der jetzigen Umsetzung arbeitet die Rule Engine in drei Schritten: die Verarbeitung der Regel, die Auswertung der Regel und die Ausführung der Aktionen der Regel. In Abbildung 4.9 ist eine Übersicht der Grundschritte der Rule Engine zu sehen.

Der erste Schritt der Rule Engine ist das Integrieren der Sensordaten innerhalb der Regeln. Dieser Schritt wird hier auch als Verarbeiten bezeichnet. Die spätere tatsächliche Implementierung der Rule Engine in der MBP soll sich dabei auf die Verarbeitung der Datenströme des MQTT Brokers bedienen. Als Datenstrom bezeichnet man kontinuierlich eintreffende Daten [17]. Damit Datenströme verarbeitet werden können, muss man bestimmte Eigenschaften beachten. Datenströme sind aktuelle Daten, das heißt, die Reihenfolge und der Zeitpunkt der eintreffenden Daten sind relevant für die Verarbeitung. Zudem entsteht eine große Masse an Daten. Beim ersten Ansatz der Umsetzung der MBP2Go Rule Engine

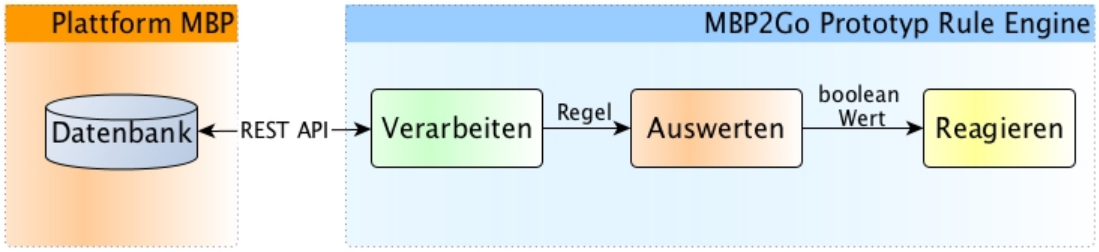

Abbildung 4.9: Grundschritte des Prototypen der Rule Engine

wurde MQTT verwendet. Die Implementierung zeigte jedoch Probleme. Die Performanz der Verarbeitung war unzureichend. Zudem war der Energieverbrauch der App sehr hoch und damit die Akkulaufzeit schlecht.

Da es bei der Umsetzung um einen Prototypen handelt, wurden diese Probleme umgangen, indem ein alternativer Lösungansatz verfolgt wurde. Für das Verarbeiten von Regeln, wurde beim Prototypen der Rule Engine ein Knopf integriert. Dabei ist der Knopf für das Erhalten der Sensordaten von der MBP zuständig. Für das Erhalten der Sensordaten wird dabei eine REST-API Anfrage an die Datenbank der MBP gestellt. Dadurch bekommt man den letzten aktuellen Wert eines Sensors. Die „sensorids“ in der Regel werden dann mit den Sensordaten ersetzt. Nehmen wir an die Wohnzimmertemperatur sei 22 Grad, die Badezimmertemperatur 23 Grad und die Außentemperatur 16 Grad. Dann nimmt die Beispiel-Regel folgende Form an:

$$(22 < 25 \text{ AND } 23 > 28) \text{ OR } (16 < 30)$$

Daraus folgt der zweite Schritt der Rule Engine. Durch diese Form kann die Rule Engine die Regel auswerten. Dafür wurde ebenfalls ein Parser entwickelt. Der Parser kann DNFs auswerten und gibt einen boolean Wert zurück. Der boolesche Wert gibt den Wert *true* zurück, falls die Regel zutrifft, ansonsten nimmt es den Wert *false* an.

Beim dritten Schritt wird schließlich der Aktionsteil ausgeführt. Falls der boolesche Wert *true* ist, werden alle Geräte und Dienste im Aktionsteil ausgeführt. Bei der Beispiel-Regel bekommt der Anwender eine Push-Nachricht auf dem Smartphone mit der Nachricht „Temperatur zu hoch“ [27].

5 Zusammenfassung und Ausblick

Mobile Applikationen bieten viele Möglichkeiten an, IoT-Applikationen zu verwirklichen. Der Markt bietet einige mobile IoT-Applikationen an, jedoch sind viele Apps noch nicht ausgereift. In Kapitel 3 werden vorhandene IoT-Apps vorgestellt. Viele IoT-Apps wurden für einen bestimmten Bereich entwickelt, zum Beispiel Smart Home. Dadurch wird die Nutzbarkeit der Apps eingeschränkt, da diese zu anderen Systemen oder IoT-Umgebungen nicht kompatibel sind. Zudem unterstützen viele IoT-Apps nur Geräte, die speziell für die Nutzung mit dieser App entwickelt wurden. Viele IoT-Apps wurden entweder auf das Gebiet der Überwachung oder auf das Gebiet der Steuerung von IoT-Umgebungen ausgerichtet. Es gibt jedoch nur wenige Apps, die beide Gebiete abdecken.

Die MBP2Go ist eine mobile IoT-Applikation, die speziell für das automatische Binden von IoT-Geräten und das Steuern und Überwachen von IoT-Umgebungen entwickelt wurde. Durch die Erweiterung der MBP2Go in dieser Arbeit deckt die App ebenfalls die Gebiete der Steuerung und Überwachung ab. Im Bereich der Überwachung wurde die App um zwei Komponenten erweitert.

Durch die Komponente *Live Daten* ermöglicht die MBP2Go die Überwachung von Echtzeit Sensordaten. Somit können Zustandsänderungen von IoT-Umgebungen in Echtzeit erkannt werden. Zudem bekommt der Nutzer der MBP2Go in Echtzeit eine Übersicht der aktiven Sensoren mit ihren jeweiligen Sensordaten. Dadurch können Fehlmessungen rechtzeitig erkannt werden.

Die zweite Komponente *Historische Daten* ermöglicht die Darstellung und Visualisierung vergangener Sensordaten. Dadurch wird die Möglichkeit angeboten eine schnelle Übersicht von vergangenen Sensordaten zu erhalten. Die Komponente ermöglicht die Aktivität eines Sensors über einen Zeitraum zu visualisieren. Durch das Diagramm können die Sensordaten abgelesen und Fehlmessungen erkannt werden. Zudem ermöglicht die Analyse des Datensatzes einen schnellen Überblick über den Maximal-, Minimal- und Durchschnittswert. Durch diese Erweiterungen der MBP2Go in dieser Arbeit deckt die App den Bereich der Überwachung in IoT-Umgebungen ab.

Im Gebiet der Steuerung bietet die MBP2Go nach der Erweiterung die Möglichkeit Regeln in DNF zu formulieren und zu speichern. Dabei wurden Aspekte der CEP-Technologie betrachtet, konzipiert und umgesetzt. In naher Zukunft wird die MBP2Go mit der Rule Engine der MBP kommunizieren. Für die geplante Schnittstelle wurde ein JSON-Format entworfen und umgesetzt. Da die Rule Engine der MBP in Entwicklung ist, beinhaltet die MBP2Go einen Prototypen der Rule Engine. Dieser kann die Regeln verarbeiten und

auswerten. Schließlich kann die Rule Engine bei Auslösen einer Regel, darauf reagieren und kann zum Beispiel eine Push-Nachricht senden. Der Prototyp wird in naher Zukunft mit der Rule Engine der MBP ersetzt.

Somit wurde in dieser Arbeit eine mobile Applikation um Funktionalitäten zur Überwachung und Steuerung von IoT-Umgebungen ergänzt.

Ausblick

Die Möglichkeiten die mobile Applikationen in IoT-Umgebungen erbringen sind vielseitig. Nach der Entwicklung der Rule Eninge in der MBP wird die MBP2Go mit dieser kommunizieren. Die Kommunikation soll dabei per MQTT funktionieren. Die App wird dann schließlich für die Erstellung von Regeln und das Reagieren von Ereignissen zuständig sein. Dafür bietet die MBP2Go viele Dienste an, zum Beispiel Push-Nachrichten.

Als mobile Applikation kann die MBP2Go nicht nur als IoT-Applikation, sondern auch als IoT-Gerät agieren. Ein Smartphone besitzt viele Sensoren, die in die MBP integriert werden können. Dadurch können die Sensordaten vom Smartphone in Echtzeit beobachtet werden. Jedoch muss die App MBP2Go die Sensordaten in Echtzeit an die MBP versenden. Somit können historische Sensordaten vom Smartphone beobachtet und analysiert werden. Die Analysen können dabei vertieft werden, beispielsweise durch den Gewinn von neuen Informationen per Data Mining [5].

Zurzeit werden viele Sensordaten in der MBP und MBP2Go 2-dimensional betrachtet. Jedoch gibt es viele Sensoren, die mehr als einen Wert messen, wodurch Sensordaten 3-dimensional betrachtet werden müssen. Zum Beispiel wird für die Ermittlung des Standortes der Längengrad, Breitengrad und der Zeitpunkt gemessen. Somit kann im Bereich der Überwachung die Visualisierung in der MBP2Go weiterhin erweitert werden.

Weitere Aspekte der CEP-Technologie können einige Vorteile bei der Entwicklung der MBP2Go erbringen. Dabei kann die Erstellung von Regeln weiter ausgebaut werden. Zurzeit werden Regeln in DNF unterstützt. Durch die Einführung des Sequenzoperators können neue Regeln definiert werden, die die Reihenfolge von Ereignissen festlegen.

Falls die MBP2Go als IoT-Gerät fungieren soll, kann sich die ständige Kommunikation mit der MBP negativ auf die Akkulaufzeit des Smartphones auswirken. Ein Lösungsansatz für das Problem kann das Einführen von Zeitabständen und Sliding Windows sein. Durch das Einführen von Zeitabständen in der MBP2Go kann definiert werden, in welchem zeitlichen Abstand Sensordaten an die MBP vermittelt werden. Zudem bringt das Einführen von Sliding Windows weitere Vorteile. Die Menge der Echtzeit-Sensordaten wächst kontinuierlich. Sliding Windows grenzen die Anzahl der betrachteten Ereignisse ein.

Außerdem müssen Lösungen gefunden werden, wie die MBP2Go im Ruhezustand die Sensordaten weiter vermitteln und erhalten kann. Ein Lösungsansatz ist die Implementierung eines Services. Ein Service ist eine Komponente, die für die Ausführung von langfristigen

Operationen im Hintergrund verantwortlich ist [15]. Dadurch kann die MBP2Go im Ruhezustand kontinuierlich Sensordaten an die MBP vermitteln und auf ausgelöste Regeln reagieren.

Literaturverzeichnis

[1] Activity. *Yonomi - Smart Home Automation*. URL: https://developer.android.com/reference/android/app/Activity (zitiert auf S. 32, 34).

[2] Ana Cristina Franco da Silva and Pascal Hirmer and Matthias Wieland and Bernhard Mitschang. „SitRS XT – Towards Near Real Time Situation Recognition“. Englisch. In: *Journal of Information and Data Management* 7.1 (Apr. 2016). URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=ART-2016-14&engl=0 (zitiert auf S. 38).

[3] brightsolutions. *Fragments in Android*. URL: https://www.brightsolutions.de/blog/fragments-android (zitiert auf S. 32).

[4] Bruns, Ralf and Dunkel, Jürgen. *Complex Event Processing: Komplexe Analyse von massiven Datenströmen mit CEP*. Springer-Verlag, 2015 (zitiert auf S. 21, 37–39).

[5] Chen, Feng and Deng, Pan and Wan, Jiafu and Zhang, Daqiang and Vasilakos, Athanasios V and Rong, Xiaohui. „Data mining for the internet of things: literature review and challenges“. In: *International Journal of Distributed Sensor Networks* 11.8 (2015), S. 431047 (zitiert auf S. 50).

[6] Christian Götz. „Ein Baukasten für das Internet der Dinge“. In: *Entwickler Magazin Spezial Internet of Things* (2014) (zitiert auf S. 13).

[7] Denny Fischer. *10 Fakten zu den 2,3 Milliarden aktiven Android-Smartphones weltweit*. URL: https://www.smartdroid.de/10-fakten-zu-den-23-milliarden-aktiven-android-smartphones-weltweit/ (zitiert auf S. 19).

[8] Douglas Crockford. *Einführung in JSON*. URL: http://www.json.org/json-de.html (zitiert auf S. 42).

[9] Eclipse Paho Android Service. *Eclipse Paho Android Service*. URL: https://www.eclipse.org/paho/clients/android/ (zitiert auf S. 34).

[10] Emil Czychon, Felix Scheerer, Seda Ulusal. *Ultimativer Vergleich mobiler IoT-Applikationen*. 2018 (zitiert auf S. 23, 27, 28).

[11] Gernot Starke. *Effektive Software-Architekturen*. Hannser, 2015 (zitiert auf S. 24).

[12] Google. *Create a Card-Based Layout*. URL: https://developer.android.com/guide/topics/ui/layout/cardview (zitiert auf S. 41).

[13] Google. *Fragments*. URL: https://developer.android.com/guide/components/fragments (zitiert auf S. 32).

[14] Google. *Resources*. URL: https://developer.android.com/reference/android/content/res/Resources (zitiert auf S. 34).

[15] Google. *Services overview*. URL: https://developer.android.com/guide/components/services (zitiert auf S. 51).

[16] Google. *Webview*. URL: https://developer.android.com/reference/android/webkit/WebView (zitiert auf S. 34).

[17] U. Hedtstück. *Complex Event Processing: Verarbeitung von Ereignismustern in Datenströmen*. eXamen.press. Springer Berlin Heidelberg, 2017. ISBN: 9783662534519. URL: https://books.google.de/books?id=UgAhDgAAQBAJ (zitiert auf S. 38, 39, 46).

[18] Heike Scholz. *Marktanteile: Android Is Eating The World*. URL: https://www.mobile-zeitgeist.com/marktanteile-android-eating-world/?cookie-state-change=1546137385477 (zitiert auf S. 18).

[19] Heinz Woern, Uwe Brinkschulte. *Echtzeitsysteme Grundlagen, Funktionsweisen, Anwendungen*. Springer. ISBN: 978-3-540-20588-3 (zitiert auf S. 31).

[20] Highcharts. *Highcharts*. URL: https://www.highcharts.com (zitiert auf S. 33).

[21] Highcharts. *Highcharts Android*. URL: https://github.com/highcharts/highcharts-android (zitiert auf S. 33).

[22] P. Hirmer, U. Breitenbücher, A. C. Franco da Silva, K. K. Képes, B. Mitschang, M. Wieland. „Automating the Provisioning and Configuration of Devices in the Internet of Things“. In: *Complex Systems Informatics and Modeling Quarterly* 9 (2016), S. 28–43 (zitiert auf S. 16, 18).

[23] Jonas Gehring. *MPAndroidChart*. URL: http://www.android-graphview.org (zitiert auf S. 33).

[24] Marc Mai. *MQTT für Dummies*. URL: https://blog.doubleslash.de/mqtt-fuer-dummies/ (zitiert auf S. 15, 16).

[25] Miorandi, Daniele and Sicari, Sabrina and De Pellegrini, Francesco and Chlamtac, Imrich. „Internet of things: Vision, applications and research challenges“. In: *Ad Hoc Networks* 10.7 (2012), S. 1497–1516 (zitiert auf S. 13).

[26] Pascal Hirmer, Ana Cristina Franco da Silva. *Multi-purpose Binding and Provisioning Platform (MBP)*. URL: https://github.com/ana-silva/MBP (zitiert auf S. 13, 16, 19, 36).

[27] Pascal Hirmer and Matthias Wieland and Holger Schwarz and Bernhard Mitschang and Uwe Breitenbücher and Santiago Gómez Sáez and Frank Leymann. „Situation recognition and handling based on executing situation templates and situation-aware workflows“. Englisch. In: *Computing* (Oktober 2016). DOI: 10.1007/s00607-016-0522-9. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=ART-2016-12&engl=0 (zitiert auf S. 47).

[28] Pascal Hirmer and Matthias Wieland and Uwe Breitenbücher and Bernhard Mitschang. „Automated Sensor Registration, Binding and Sensor Data Provisioning“. Englisch. In: *Proceedings of the CAiSE'16 Forum, at the 28th International Conference on Advanced Information Systems Engineering (CAiSE 2016)*. Bd. 1612. CEUR Workshop Proceedings. Ljubljana, Slovenia: CEUR-WS.org, Juni 2016, S. 81–88. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2016-22&engl=0 (zitiert auf S. 16).

[29] Pascal Hirmer and Matthias Wieland and Uwe Breitenbücher and Bernhard Mitschang. „Dynamic Ontology-based Sensor Binding“. Englisch. In: *Advances in Databases and Information Systems. 20th East European Conference, ADBIS 2016, Prague, Czech Republic, August 28-31, 2016, Proceedings*. Bd. 9809. Information Systems and Applications, incl. Internet/Web, and HCI. Prague, Czech Republic: Springer International Publishing, Aug. 2016, S. 323–337. ISBN: 978-3-319-44038-5. DOI: 10.1007/978-3-319-44039-2. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2016-25&engl=0 (zitiert auf S. 16, 18).

[30] Philipp Friberg. *RESTful APIs mit OData*. URL: https://entwickler.de/leseproben/odata-restful-api-579795152.html (zitiert auf S. 42).

[31] Phillip Jahoda. *MPAndroidChart*. URL: https://github.com/PhilJay/MPAndroidChart (zitiert auf S. 33).

[32] U. Schöning. *Logik für Informatiker*. Spektrum Akademischer Verlag, 2000. ISBN: 9783827410054. URL: https://books.google.de/books?id=IB9VAAAACAAJ (zitiert auf S. 41).

[33] SciChart. *SciChart WPF Chart*. URL: https://www.scichart.com (zitiert auf S. 33).

[34] SenLab. *Contact Us*. URL: https://iotool.io/contact-us?source=your-own-virtual-sensor (zitiert auf S. 25).

[35] Stringify, Inc. *Stringify - Smart Home and IoT*. URL: https://play.google.com/store/apps/details?id=com.stringify.stringify&hl=gsw (zitiert auf S. 23, 24).

[36] Ted Kritsonis. *Stringify automator app review*. URL: https://www.digitaltrends.com/home/stringify-review/ (zitiert auf S. 24).

[37] Thorin Klosowski. *How to Build a Smart, Connected Home with Stringify*. URL: https://lifehacker.com/how-to-build-a-smart-connected-home-with-stringify-1763921515 (zitiert auf S. 24).

[38] Trojan Walter. *Das MQTT-Praxisbuch*. Elektor Verlag, 2017 (zitiert auf S. 15).

[39] O. Vermesan, P. Friess. *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*. River Publishers, 2013 (zitiert auf S. 13).

[40] Yonomi. *Yonomi - Smart Home Automation*. URL: https://play.google.com/store/apps/details?id=com.yonomi&hl=de (zitiert auf S. 26).

Alle URLs wurden zuletzt am 15. 01. 2019 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift