Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Adaptives Testwerkzeug für IoT-Applikationen

Katja Seber

Studiengang:	Softwaretechnik
Prüfer/in:	PD Dr. rer. nat. habil. Holger Schwarz
Betreuer/in:	Dr. rer. nat. Pascal Hirmer
Beginn am:	21. Oktober 2019
Beendet am:	21. April 2020

Kurzfassung

Durch das Internet der Dinge (IoT) wird die Möglichkeit geboten, mit Hilfe von IoT-Applikationen Verbesserungen und Vorteile in diversen Anwendungsbereichen zu schaffen. Dabei werden mit Sensoren und Aktuatoren ausgestattete IoT-Geräte miteinander verknüpft, um die sich ständig ändernde Umgebung wahrzunehmen und auf diese möglichst ohne menschliches Eingreifen reagieren zu können. Die Stabilität und die korrekte Funktionsweise einer solchen Anwendung stellen dabei wichtige Anforderungen dar, welche besonders durch Ausnahmefälle (z.B. Ausreißer bei Sensorwerten) gefährdet werden. Aus diesem Grund sind Software-Tests bei der Entwicklung von IoT-Applikationen unabdinglich. Ziel dieser Bachelorarbeit ist es daher, ein weitgehend automatisiertes Testwerkzeug bereitzustellen, um stabile und korrekt arbeitende Anwendungen zu schaffen. Um effektive Tests von verschiedenen Testfällen zu ermöglichen, sollte die Simulation von Sensoren und Aktuatoren ermöglicht werden. Zur Integration der Simulatoren und des Testwerkzeugs lag die IoT-Plattform Multi-purpose Binding and Provisioning Platform (MBP) zugrunde. Innerhalb der Literatur konnten keine Quellen gefunden werden, welche konkrete Lösungen, beziehungsweise bereits bestehende Tools zur Erfüllung der Ziele zur Verfügung stellen. Daher wurden das Testwerkzeug sowie die Simulatoren eigenständig innerhalb dieser Arbeit entwickelt. Dadurch können mit Hilfe dieser verschiedene Testfälle einer IoT-Applikation effektiv geprüft werden. Anhand der Ergebnisse eines Tests können bei detektiertem Fehlverhalten erforderliche Maßnahmen getroffen werden. Dadurch sind die Anforderungen einer stabilen und korrekten Funktionsweise einer IoT-Applikation erfüllbar.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

CEP Complex Event Processing. 16

DPWS Devices Profile for Web Services. 22

IoT Internet der Dinge. 3

MBP Multi-purpose Binding and Provisioning Platform. 3

MQTT Message Queuing Telemetry Transport Protocol. 15

Operatoren Extraction/Control Operators. 27

TaaS Test-As-A-Service. 23

VM Virtuelle Maschine. 29

1. Einleitung

Das Internet mitsamt dessen Anwendungen ist ein fester Bestandteil im alltäglichen Leben der Menschheit geworden [2]. Aus diesem Grund wurden über die Jahre weitere Forschungen in Hinsicht auf die Vernetzung und Kommunikation internetfähiger Geräte unternommen, um deren Funktionen miteinander zu verbinden und gemeinsame Ziele zu erreichen [3][2]. Das Resultat dessen wird heute als Internet der Dinge (IoT) bezeichnet [2][1].

Das IoT ermöglicht es Netzwerke verschiedener internetfähiger physischer Objekte zu schaffen, welche dazu in der Lage sind, Daten zu verarbeiten, sich untereinander auszutauschen und miteinander zu interagieren. Durch die häufige Ausstattung dieser Objekte mit Sensoren und Aktuatoren, können sich ändernde Situationen der externen Umwelt beobachtet, analysiert und auf diese ohne menschliche Eingriffe reagiert werden [3]. In Folge der Automatisierung von menschlichen Eingriffen durch das IoT können Prozesse unter anderem effizienter und schneller durchgeführt werden [3]. Mit Hilfe dieser Fähigkeiten bietet sich das Potential kreative Anwendungen zu schaffen, welche das Leben in einer breiten Anzahl an Bereichen erleichtern [3]. Beispielsweise kann durch die Verknüpfung eines Temperatursensors mit einer Klimaanlage und Heizung die Raumtemperatur automatisch, möglichst ohne menschliches Eingreifen, gesteuert werden. Derartige Anwendungen, in welchen Objekte des IoT miteinander verknüpft werden, werden als IoT-Applikationen bezeichnet.

Durch schnelle Fortschritte und eine Vielzahl an Technologien im Bereich des IoT können durch IoT-Applikationen Vorteile und Verbesserungen in vielen verschiedenen Anwendungsbereichen geschaffen werden [1]. Dazu gehört die Erleichterung des alltäglichen Lebens, beispielsweise durch *Smart-Homes* und *Smart-Cities*. Durch die Verknüpfung von Alltagsgegenständen innerhalb eines Smart-Homes werden dem Nutzer automatisierte Dienste wie beispielsweise intelligente Beleuchtungs-, Sicherheits- und Unterhaltungssysteme geboten [4]. Smart-Cities schaffen im Gegensatz dazu komplexere IoT-Applikationen, wie intelligente Verkehrs- und Parksysteme [3]. Des Weiteren werden Verbesserungen durch IoT-Applikationen in Bereichen wie der Industrie, dem Gesundheits-, Bildungs- und Energiewesen geschaffen [3]. Um die Realisierung solcher IoT-Applikationen zu vereinfachen und somit auch Nichtexperten zugänglich zu machen, wurden IoT-Plattformen geschaffen. Durch diese soll die Handhabung der IoT-Applikationen und -Objekte vereinfacht und für den Alltag gebräuchlich gemacht werden [5].

1.1. Problemdefinition

IoT-Applikationen finden, wie beschrieben, in diversen Bereichen des Lebens Einzug, worunter beispielsweise durch das Gesundheitswesen und intelligente Verkehrssysteme sicherheitskritische Anwendungen fallen. Daher und aufgrund der zunehmenden Zahl an verschiedenen Technologien und Anwendungsbereichen des IoT stellen stabile und korrekt arbeitende IoT-Applikationen eine wichtige Anforderung dar. Die Schwierigkeit dabei liegt darin, dass IoT-Applikationen aus einer

großen Menge an verschiedenen Objekten bestehen, welche die Anforderung durch jeweiliges Fehlverhalten gefährden können. Hauptursache für ein falsches Verhalten einer IoT-Applikation stellen Sensoren dar. Ihre aus der externen Umwelt extrahierten Daten werden analysiert und auf diese durch Aktuatoren reagiert. Als eine Folge eines solchen Fehlverhaltens, aufgrund einer Anomalie innerhalb der Sensorwerte, kann es zu Kommunikationsproblemen der IoT-Geräte untereinander kommen, die eine Beeinträchtigung der Interaktion innerhalb der IoT-Applikationen bedingen können [6]. Tritt ein Fehlverhalten eines Sensors auf, löst die Analyse im schlechtesten Fall eine nicht zum Kontext passende Aktion durch Aktuatoren aus. Dies kann insbesondere in sicherheitskritischen Anwendungsbereichen (z.B. autonome Fahrzeuge) fatale Auswirkungen mit sich bringen.

Das Fehlverhalten von Sensoren stellt keine Seltenheit dar. Dabei können diverse Anomalien innerhalb der Sensorwerte als ein solches definiert werden:

- Ausreißer-Werte,
- Fehlmessungen,
- falsche Wertetypen,
- fehlende Werte,
- Ausfälle,
- ...

Für die Anzahl an Verknüpfungen innerhalb von IoT-Applikationen existiert keine Begrenzung nach oben. Eine Konsequenz daraus sind große drahtlose Netzwerke, deren Volumen an Austausch von IoT-Daten sehr groß werden kann [6]. In Anbetracht dieser Tatsache und vor dem Hintergrund, dass das Auftreten solcher Anomalien keine Seltenheit ist, steigt die Wahrscheinlichkeit eines Fehlverhaltens der IoT-Applikation mit jedem weiteren Gerät beziehungsweise Sensor an. Eine Nichtbehandlung des Problems könnte somit zu vermehrtem Fehlverhalten und sicherheitskritischen Situationen führen.

Da die zentrale Komponente von IoT-Applikationen typischerweise die IoT-Plattformen darstellen, ist es sinnvoll ein Testwerkzeug zur Prüfung zu integrieren. Durch die resultierenden Ergebnissen können Maßnahmen, wie die Definition weiterer Regelungen, welche das Fehlverhalten von IoT-Applikationen durch Anomalien eindämmen, definiert werden.

Prüft man die oben beschriebene Anforderung durch Tests, so wird eine zusätzliche Problematik deutlich. Es besteht eine große Schwierigkeit darin, Sensoren äußerlich so zu manipulieren und anzupassen, dass diese für bestimmte Testfälle einsetzbar sind [7]. Des Weiteren können Werte, mit denen bereits getestet wurde, nicht wiederverwendet werden, um Tests wiederholt unter den gleichen Bedingungen ausführen zu können [8]. Soll eine große Zahl an IoT-Geräten innerhalb von IoT-Applikationen zu einem großen Netzwerk verbunden werden, können zusätzlich hohe Anschaffungskosten anfallen [8].

1.2. Zielsetzung

Diese Bachelorarbeit zielt auf das Testen der Anforderung einer stabilen und korrekt arbeitenden IoT-Applikation ab. Dadurch sollen bei Fehlverhalten Maßnahmen ergriffen werden können, um diese und daraus resultierende sicherheitskritische Situationen zu verhindern. Das Hauptziel besteht somit darin, ein weitgehend automatisiertes Werkzeug für das Testen von IoT-Applikationen zu entwickeln. Dabei liegt der Fokus auf der Gewährleistung einer korrekten und stabilen Interaktion, auch in Ausnahmefällen, welche durch Anomalien innerhalb von übertragenen Sensorwerten ausgelöst werden können. Bei der Plattform, in welche das zu entwickelnde Tool integriert werden soll, handelt es sich um die bereits existierende IoT-Plattform MBP[1], welche in Abschnitt 2.3 näher beschrieben wird. Innerhalb der Benutzeroberfläche des Testwerkzeugs soll es möglich sein die Tests zu verwalten, was das Anlegen, Starten, Stoppen und Löschen beinhaltet. Zudem soll mit der Ausgabe eines Testberichts die Grundlage für die Ergreifung von Maßnahmen geliefert werden.

Anhand der Problemdefinition lässt sich ein Teilziel ableiten, welches innerhalb dieser Arbeit verfolgt wird. Aufgrund der Hindernisse bei der Bereitstellung von IoT-Geräten und zu Testfällen passenden Sensorwerten ist es sinnvoll, einen Sensor- und Aktuator-Simulator für das Testen von IoT-Applikationen zu entwickeln. Dabei sollen die Sensor-Simulatoren kommunikationsfähig und dazu in der Lage sein, auf verschiedene Anwendungsfälle einzugehen und realitätsnahe Werte zu generieren. Hierzu gehören die Simulation von Änderungen der externen Umgebung sowie die Kombination von Anomalien mit diesen. Im Gegensatz dazu soll der Simulator des Aktuators ebenfalls kommunikationsfähig sein und mit der IoT-Applikation verbunden werden können, jedoch keine Aktionen ausführen.

1.3. Aufbau der Arbeit

Zu Beginn dieser Arbeit werden grundlegende Begriffe beleuchtet, um ein Verständnis für die Thematik der darauffolgenden Kapitel zu schaffen. Dabei spielen sowohl verschiedene Begriffe der IoT, die IoT-Plattform MBP aber auch Testwerkzeuge eine Rolle. Im Anschluss wird in Kapitel 3 auf die verwandten Arbeiten eingegangen, welche sich mit der genannten Problemstellung befassen und in der Literatur gefunden werden konnten.

In Kapitel 4 wird die methodische Vorgehensweise zur Erreichung der Ziele dieser Arbeit beschrieben. Dabei wird zunächst auf die bereits bestehende IoT-Plattform MBP eingegangen, in welche das Testwerkzeug integriert werden soll, um nötige Anpassungen detektieren zu können. Die Details zur Umsetzung dieser und die Ausführung eines Tests anhand des entwickelten Testwerkzeugs werden im Anschluss mit Hilfe einer Testmethodik und eines fortlaufenden Beispiels beleuchtet.

Der Abschluss der Arbeit bildet das Fazit, in welchem beurteilt wird, ob die definierten Ziele der Arbeit erreicht wurden. Dabei werden eine kurze Zusammenfassung der Arbeit sowie ein Ausblick über zukünftige Arbeiten in diesem Themenbereich gegeben.

[1] Open-Source-Projekt: https://github.com/IPVS-AS/MBP

2. Grundlagen

Dieses Kapitel umfasst die Grundlagen, welche für das Verständnis der vorliegenden Bachelorarbeit notwendig sind. Dazu zählen IoT-Applikationen sowie -Plattformen, das Kommunikationsprotokoll Message Queuing Telemetry Transport Protocol (MQTT), die Beschreibung der IoT-Plattform MBP, sowie eine Übersicht über Testwerkzeuge.

2.1. Internet der Dinge

Durch das IoT wird die Möglichkeit geboten internetfähige Geräte zu verteilten Netzwerken zusammenzuführen, welche durch verschiedene Kommunikationsnetzwerke dazu in der Lage sind, sowohl mit dem Menschen als auch mit anderen internetfähigen Geräten zu kommunizieren [1]. Solche Geräte werden als IoT-Geräte bezeichnet und steigen in der Zahl immer weiter an. Dieser Umstand eröffnet ein breites Feld an nützlichen Anwendungen (IoT-Applikationen), welche Lösungen in den verschiedensten Anwendungsbereichen wie beispielsweise in der Industrie, dem Gesundheitswesen und dem alltäglichen Leben bieten [3][9].

2.1.1. IoT-Applikationen

Allgemein basieren IoT-Applikationen auf einer Vielzahl an individuellen Komponenten und ihrer Beziehung zueinander [10]. Diese beschreiben damit heterogene IoT-Geräte, sowie ihre Kommunikation und Interaktion untereinander. Durch die Verknüpfungen können die verschiedenen Funktionen, welche die einzelnen Geräte mit sich bringen, gebündelt werden [11][12]. Um ein IoT-Gerät als ein solches bezeichnen zu können, sollte es zumindest die folgenden Eigenschaften erfüllen [13]:

- Es handelt sich um physisches Objekt
- Es existieren Funktionen für die Kommunikation innerhalb von Netzwerken
- Es ist eindeutig identifizierbar
- Es besitzt einen Namen und eine Adresse, um mit anderen IoT-Geräten kommunizieren zu können
- Es besitzt zumindest grundlegende Rechenkapazitäten für das Empfangen und Verarbeiten von Daten
- Es kann über sensorische und aktuatorische Fähigkeiten verfügen

Sensoren von IoT-Geräten wandeln physikalische Größen in digitale Signale um, die vom Gerät weiter verarbeitet werden können [12][14]. Diese Daten können durch die Rechenkapazitäten analysiert und durch die Kommunikationsfähigkeit an Aktuatoren weitergeleitet werden. Die Aktuatoren führen dann eine zum Kontext passende physische Aktion aus [14]. Dabei wird eine zeitnahe Entscheidungsfindung auf die Änderungen der externen Umgebung vorausgesetzt [9]. Eine durch dieses Prinzip entstehende IoT-Applikation kann die automatisierte Steuerung des optimalen Raumklimas innerhalb eines Wohnhauses bereitstellen. Hier werden beispielsweise Temperatursensoren und Feuchtigkeitssensoren eingesetzt, um den Zustand der Umgebung zu erfassen. Die extrahierten Werte werden zeitnah verarbeitet und analysiert. Wenn notwendig, werden Aktuatoren wie eine Heizung, Klimaanlage, Luftbefeuchter oder Luftentfeuchter eingesetzt, um Missstände automatisch anzupassen und das perfekte Raumklima zu gewährleisten.

2.1.2. IoT-Plattformen

Um als ungeübter Nutzer des IoT Applikationen einfach realisieren zu können, wurden IoT-Plattformen als Middleware und Infrastruktur geschaffen [6][5]. Solche Plattformen ermöglichen die Verwaltung und Koordination von IoT-Geräten mit Sensoren und Aktuatoren, wobei oftmals Regeln definiert werden, um auf bestimmte Ereignisse reagieren zu können. Dabei wird häufig Complex Event Processing (CEP) genutzt [9]. Durch CEP können in komplexen Ereignissen, hier die Datenströme der Sensoren, Muster erkannt und auf sie reagiert werden. Dieser Prozess geschieht kontinuierlich und gegenwartsnah [15].

Beispielsweise können Applikationen eines Smart-Homes durch den Nutzer über IoT-Plattformen definiert werden. Durch Smart-Homes werden Alltagsgegenstände mit den Eigenschaften von IoT-Geräten von Gebäuden untereinander verknüpft, um ein intelligentes Miteinander und Wohnen zu schaffen. Im Beispiel der Steuerung des optimalen Raumklimas können die verschiedenen IoT-Geräte mit dazugehörigen Sensoren und Aktuatoren an die Plattform angebunden und deren Umgebung modelliert werden. Um auf unerwünschte Temperaturänderungen reagieren zu können, werden beispielsweise Regeln definiert, welche besagen, dass ab einer Raumtemperatur von 28.5°C die Klimaanlage eingeschalten werden soll.

2.2. Message Queuing Telemetry Transport Protocol

Innerhalb von IoT-Applikationen können durch eine Vielzahl an miteinander verbundenen Komponenten große drahtlose Netzwerke entstehen. Diese stellen eine Herausforderung in Bezug auf die Echtzeit-Kommunikation untereinander und im Bereich der Speicherkapazität und Rechenkapazität dar [16]. Eines der etablierten Netzwerkprotokolle zur Lösung dieses Problems ist MQTT [17]. Bei MQTT handelt es sich um ein Ereignis- und Nachrichtenprotokoll, welches auf einem Server-Client-Protokoll basiert und mit dem Prinzip des *Publish and Subscribe Messaging* ausgestattet ist [16][17].

In Abbildung 2.1 werden die Funktionsweise und die Hauptkomponenten des MQTT visualisiert. Zu den Hauptkomponenten zählen sowohl die *MQTT-Clients*, unter die die sogenannten *Publisher* und *Subscriber* fallen, als auch der *MQTT-Broker*, welcher für den Nachrichtenaustausch zwischen den Clients verantwortlich ist [18]. Ein weiterer wichtiger Bestandteil sind die sogenannten *Topics*,

Abbildung 2.1.: Funktionsweise des MQTT

welche es ermöglichen Nachrichten zu kategorisieren. Während Subscriber den Broker darüber informieren, welche Topics für sie von Relevanz sind, senden die Publisher ihre Nachrichten in Bezug auf ein bestimmtes Topic an den Broker [16]. Stimmen die Topics einer vom Publisher eingehenden Nachricht und das Interesse des Subscribers überein, sendet der Broker diese Nachricht an die Subscriber weiter [18][16].

2.3. Multi-purpose Binding and Provisioning Platform

Die MBP stellt eine wie im Abschnitt 2.1.2 beschriebene Open-Source-IoT-Plattform für die einfache Realisierung von IoT-Applikationen dar. Sie wurde durch das Institut IPVS/AS der Universität Stuttgart entwickelt, um eine automatische Anbindung, Bereitstellung und Verwaltung der zu einer IoT-Applikation gehörenden IoT-Geräte zu ermöglichen [19][5]. Dadurch soll der Umfang der manuellen Aufgaben, welche sonst durch die Nutzer getätigt werden müssen, auf einem Minimum gehalten werden [5].

Mit der MBP wird sowohl eine graphische Benutzeroberfläche (Abbildung 2.2) als auch eine REST-API bereitgestellt, welche die selben Funktionalitäten bieten [5]. Es besteht die Möglichkeit IoT-Applikationen zu realisieren, indem einzelne IoT-Geräte registriert, IoT-Umgebungen modelliert, grundlegende Informationen und Daten über Dashboards visualisiert und benutzerdefinierte Regeln zur Ausführung der Aktuatoren auf bestimmte Ereignisse definiert werden. Diese Regeln arbeiten nach dem *event-condition-action*-Prinzip [5]. Während ein Event (*event*), als ein gesendeter Wert vom Sensor an die Plattform definiert wird, stellen die Komponenten Bedingung (*condition*) und Aktion (*action*) vom Nutzer definierte Parameter dar. Verletzt ein Event eine Bedingung beziehungsweise Regel, erfolgt die definierte Aktion durch einen Aktuator [5]. Dieser Prozess der Entscheidungsfindung auf bestimmte Events wird durch CEP gewährleistet.

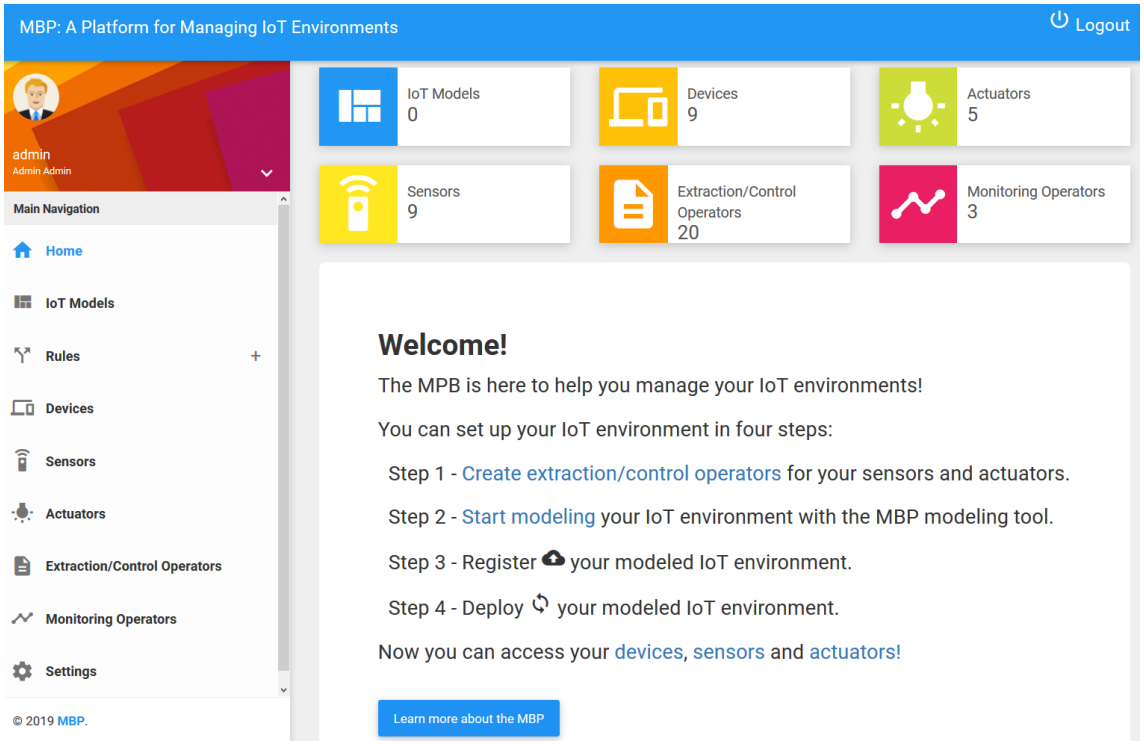


Abbildung 2.2.: Benutzeroberfläche der MBP

2.4. Testwerkzeuge

Durch den hohen Wettbewerb und die dadurch entstehende Vielzahl an Angeboten steht die Kundenzufriedenheit an höchster Stelle, um die Abnehmer an sich zu binden. Da Software, auch im IoT-Bereich, meist eine erhebliche Schlüsselrolle besitzt, besteht eine starke Wechselbeziehung zwischen der Kundenzufriedenheit und Softwarequalität [20]. Je besser die Qualität der Software, desto zufriedener ist der Kunde. Dazu gehört ebenfalls die Sicherheit der Nutzer beim Verwenden, da in einigen Branchen das menschliche Leben durch Softwaredefekte gefährdet werden kann [20]. Folgendes Zitat von Myers 1979 zeigt damit eine der Absichten, welche hinter dem Testen von Software steckt:

> ”Testen ist die Ausführung eines Programms mit dem Ziel, Fehler zu entdecken.” [21].

Es existieren viele verschiedene Arten von Softwaretests, jedoch wird sich hier auf die für diese Bachelorarbeit relevanten Methoden beschränkt. Systemtests beziehen sich auf das Testen eines ausführbaren Systems [21]. Sie haben die Aufgabe, die Kommunikation zwischen einzelnen Subsystemen, sowie die geforderte Funktionalität zu prüfen [21].

Nach Ludewig und Lichter kann der prinzipielle Verlauf eines Tests mit Hilfe von Abbildung 2.3 beschrieben werden.

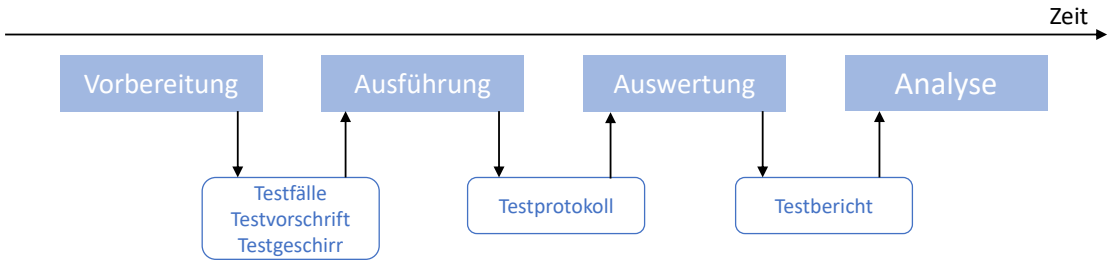


Abbildung 2.3.: Prinzipieller Testverlauf [21]

Für jeden Test sollten, wie in Abbildung 2.3 dargestellt, Vorbereitungen getroffen werden, um systematisch und geplant vorgehen zu können. Zu den Vorbereitungen zählt unter anderem die Definition von Testfällen. Durch sie wird festgelegt, welche Voraussetzungen für den Test erfüllt sein müssen, welche Systemeingaben benötigt werden und sie definieren das gewünschten Ergebnis (Soll-Resultat) [21]. Nach Abschluss der Vorbereitungen, wird das System mit den definierten Eingaben ausgeführt, woraufhin schließlich eine Auswertung erfolgt [21]. Im Zuge der Auswertung werden die zuvor definierten Soll-, sowie die im Ausführungsschritt entstandenen Ist-Werte miteinander verglichen und ein Testbericht über das Erreichen des Testkriteriums erstellt [21]. Der abschließende Schritt des vorgestellten Verlaufs bildet die Analyse des Testberichts durch den Nutzer. Auf deren Grundlage können bei einem Misserfolg Entscheidungen über nötige Maßnahmen getroffen werden.

Dieses Vorgehen kann entweder manuell oder automatisiert durch spezielle Testwerkzeuge durchgeführt werden. Testwerkzeuge haben gegenüber der manuellen Ausführung Vorteile und decken zudem Probleme, wie etwa einen hohen Zeitaufwand ab [22]. Zu den Vorteilen zählen unter anderem die Unterstützung der Verwaltung, die automatisierte Ausführung und Auswertung und die Wiederverwendbarkeit bereits angelegter Tests [22].

3. Verwandte Arbeiten

Dieses Kapitel beschreibt thematisch verwandte Arbeiten zu dieser Bachelorarbeit. Der erste Abschnitt beleuchtet Arbeiten, welche sich auf die Simulationen von verschiedenen Sensoren konzentrieren, wohingegen sich der zweite Abschnitt mit Arbeiten zur Thematik eines Testwerkzeuges für IoT-Applikationen befasst. Schließlich wird eine Auswahl an Tools getroffen, welche innerhalb der Arbeit zur Erreichung der Ziele verwendet werden.

3.1. Sensor-Simulation für IoT-Applikationen

Während die Simulation von Sensornetzwerken in der Forschung umfassend behandelt wurde und hierfür viele Tools bereitstehen, können nur wenige Quellen zur Erstellung von Simulationen einzelner Sensoren gefunden werden. Da das entstehende Testwerkzeug zusammen mit der Sensor-Simulation in die bestehende Plattform MBP integriert werden soll, und an diese lediglich einzelne Sensoren angebunden werden können, sind die Simulatoren für Sensornetzwerke von keinem Nutzen für diese Arbeit. Diese Arbeiten werden jedoch zusätzlich herangezogen, um Methoden für den Aufbau der Simulatoren zu extrahieren.

Giménez et al. [23] gehen mit der Intention die Umsetzbarkeit von großen Systemen in simulierten Risikoszenarien zu testen, an diese Thematik heran. Dabei sollen kostengünstige Simulationen von Sensornetzwerken zum Einsatz kommen. Im Zuge der Sensor-Simulationen innerhalb des Netzwerkes sollen diese je nach Testfall konfiguriert werden können. Dabei können Parameter wie die Ober- und Untergrenze der Sensorwerte, die Frequenz, mit der die Daten versendet werden, die Art der Simulation (Sinus, Exponential, usw.) sowie Simulation von Anomalien und der Standort des Sensors definiert werden. So besteht die Möglichkeit der Simulation eines oder mehrerer Sensornetzwerke. Diese Sensornetzwerke stehen in diesem Fall in einer drahtgebundenen oder drahtlosen Verbindung mit einem *Sensor Observation Service* und dem *Control Center*, welche der Auffassung und Weiterverarbeitung der simulierten Sensornetzwerke dienen. Da es sich hier um die Simulation von ganzen Sensornetzwerken handelt, kann lediglich die Beschreibung, welche Parameter zusammen mit einem Sensor des Netzwerkes definiert werden, für diese Arbeit heran gezogen werden.

Durch Pflanzner et al. [24] wurde mit *MobIoTSim* ein Echtzeit-Simulator für Sensoren einzelner IoT-Geräte entwickelt. Dieser ist Android-basiert und wird über eine mobile App gesteuert. Der Simulator definiert Eigenschaften, welche IoT-Geräte ausmachen. Dabei besitzen die Simulatoren eine eindeutige ID, mit der sie direkt angesprochen werden können, sind dazu in der Lage Sensordaten zu simulieren und unterstützen die Kommunikation mittels MQTT. Um die Simulation der Sensorwerte zu konfigurieren, besteht die Möglichkeit Ober- und Untergrenzen der Sensorwerte sowie die Frequenz, in welcher diese gesendet werden, zu definieren. Sind alle Einstellungen getätigt, kann die Simulation gestartet, gestoppt, bearbeitet oder gelöscht werden. *MobIoTSim* beschreibt

somit beinahe den Simulator, welcher für diese Arbeit benötigt wird. Es fehlt jedoch, die für diese Arbeit notwendige Möglichkeit, die Art der Simulation sowie mögliche Anomalien definieren zu können. Dies und die Tatsache, dass es sich bei *MobIoTSim* lediglich um eine mobile Anwendung handelt, macht die Anwendung für diese Arbeit unbrauchbar.

Mit dem *Simple Sensor Simulator* [25] ist ebenfalls ein Sensor-Simulator gegeben. Um die Art der Sensor-Simulation zu definieren, bestehen drei Möglichkeiten. Werte können entweder zufällig zwischen einer Ober- und Untergrenze, durch einen gegebenen Algorithmus oder ein *record and play*-Prinzip simuliert werden. Zudem können Konfigurationen wie die Frequenz, das Datenformat und die Kommunikationsart definiert werden. Die Schwierigkeit, diesen Simulator für den hier definierten Zweck zu nutzen, besteht in der freien Zugänglichkeit und der stark beschränkten Dokumentation dieses Tools, was mögliche Änderungen, welche zur Anpassung getätigt werden müssten, zusätzlich erschwert.

DPWSim [26] ist ein von Han et al. entwickelter Werkzeugkasten, welcher die Möglichkeit bietet Sensornetzwerke zu simulieren, welche über Devices Profile for Web Services (DPWS)-Protokolle miteinander kommunizieren [27]. Für die Verwaltung der Simulatoren steht eine intuitive graphische Benutzeroberfläche zur Verfügung [26]. Ein Nachteil dieses Tools besteht darin, dass keine Beschreibung darüber gefunden werden kann, auf welche Art und Weise die Sensoren simuliert und welche Anwendungsfälle dabei abgedeckt werden können. Weitere Nachteile finden sich in der eingeschränkten Anpassbarkeit durch mangelnde Code-Kommentare sowie dem sich zur MBP unterscheidenden Kommunikationsprotokoll DPWS. Auf Grund dieser Nachteile wird das vorgestellte Tool nicht für diese Arbeit herangezogen.

Einen weiteren Ansatz zur Generierung von Sensor-Daten wird durch das Projekt *IoT-Lab* von Fernandes et al. [28] geliefert. In diesem Fall wird ein Co-design vorgeschlagen, welches die Einbeziehung vieler Nutzer zum Implementieren und Testen von Lösungen im Bereich der IoT fordert. Sensordaten, welche unter anderem für Tests benötigt werden, werden durch diesen Ansatz kostengünstig von der Öffentlichkeit über die integrierten Sensoren des Smartphones oder Tablets bereitgestellt. Dadurch kann ein breites Spektrum an Sensordaten für verschiedene Anwendungsfälle gewonnen werden. Eine Schwierigkeit dieses Ansatzes besteht darin, dass ein Anreiz für eine breite Masse von Menschen geliefert werden muss, um so viele verschiedene Sensordaten für verschiedene Anwendungsfälle wie möglich zu erhalten. Des Weiteren existieren beträchtliche Unterschiede in Hinblick auf Zuverlässigkeit und Verfügbarkeit der Daten sowie die Gefahr, dass keine für einen speziellen Anwendungsfall passenden Daten durch die miteinbezogenen Nutzer geliefert werden. Aus diesen Gründen scheidet die Integration dieses Ansatzes für die Erstellung der Sensor-Simulatoren in dieser Bachelorarbeit aus.

Durch Bosmans et al. [8] wird ein Testansatz für IoT-Systeme beschrieben, welcher auf hybriden Simulationen von lokalen Komponenten basiert. Dabei wird eine lokale Komponente als eine externe unvorhersehbare Komponente definiert, welche durch einen Sensor, Aktuator oder menschlichen Akteur repräsentiert wird. Eine Hybride Simulation hat die Bedeutung, dass ein Teil der Testdaten aus simulierten und der andere Teil aus realen Daten besteht, wobei die Größe der Anteile je nach Phase der Entwicklung variieren. Für die Simulation der lokalen Komponenten werden in diesem Ansatz zwei Methoden beschrieben. Zum einen kann das Verhalten explizit, meist durch eine agentbasierte Modellierung beschrieben werden, zum anderen können Daten von bereits existierenden Datensätzen wiedergegeben, welche beispielsweise durch die Öffentlichkeit bereitgestellt werden. Die simulierten und realen Daten sollten keine großen Inkonsistenzen aufweisen, weshalb ein Synchronisationsschritt eingeführt wird. In diesem Schritt werden reale Daten als Vorhersagen

verwendet, um simulierte Daten gegebenenfalls anzupassen. Infolgedessen können realistische Werte erzeugt werden. Durch diesen Testansatz wird die Notwendigkeit der Simulation von Daten hervorgehoben. Der beschriebene Ansatz der Wiederverwendung öffentlicher Datensätze ist jedoch schwer umsetzbar. Ein Grund dafür ist die Suche nach einem passenden Datensatz für ausgewählte Testfälle, welche sich als zeitintensiv erweisen kann. Demnach stellt die Quelle lediglich eine Basis und Motivation für die Entwicklung eines funktionierenden realitätsnahen Sensor-Simulator dar.

3.2. Testwerkzeuge für IoT-Applikationen

Popereshnyak et al. [29] beschreiben die zunehmende Wichtigkeit von wirksamen automatisierten Tests für das IoT, sowie verschiedene Bereiche auf die hierbei ein Augenmerk gelegt werden sollte. Ein Grund für die Dringlichkeit der Tests ist die rasante Entwicklung und Vielzahl an unterschiedlichen Lösungen auf diesem Gebiet. Dadurch entstehen zahlreiche Probleme sowohl innerhalb der IoT-Applikationen als auch für die Sicherheit des Menschen und der Umwelt durch Anomalien der einzelnen IoT-Geräte und generellen Fehlfunktionen. Aufgrund der engen Verbindung zwischen IoT-Applikationen und der physischen Welt stellt ein Hauptbereich das Testen von IoT-Applikationen dar. Dabei sollten laut Popereshnyak et al. Simulatoren verwendet werden, um unter anderem effizient auf die Funktionalität, Sicherheit, Netzwerkkonnektivität sowie viele weitere Punkte testen zu können. Jedoch kann sich der Ansatz für IoT-Tests je nach Architektur unterscheiden, weshalb eine Vielzahl an unterschiedlichen Testtypen existieren. Durch diese Beschreibung wird die Wichtigkeit der Thematik des Testens von IoT-Applikationen verdeutlicht, wobei zusätzlich die Notwendigkeit der Sensor-Simulationen für ein effektives Testen bestätigt wird. In dieser Quelle wird kein bestimmtes Testwerkzeug vorgestellt, weshalb sie lediglich zur Verdeutlichung der Wichtigkeit der Tests von IoT-Applikationen dient, was das grundlegende Thema dieser Bachelorarbeit darstellt.

Durch den von Kim et al. [30] beschriebenen IoT-Test-As-A-Service (TaaS) Ansatz, wird ebenfalls die Notwendigkeit des Testens von IoT-Applikationen verdeutlicht. Durch diesen Ansatz werden ein allgemeingültiges Konzept sowie eine Entwurfsmethode bereitgestellt, um heterogene IoT-Applikationen standardmäßig auf Funktionsmerkmale sowie die semantische und syntaktische Korrektheit innerhalb der Kommunikation zu überprüfen. Im Zuge dessen werden drei Schlüsseltechnologien vorgestellt, welche das Testen auf verschiedenen Ebenen ermöglichen. Diese drei Technologien können als verschiedene Phasen betrachtet und als Konformitätstest gefolgt von Semantikvalidierungstests und dem abschließenden Interoperabilitätstest definiert werden. Der hier beschriebene semantische Validierungstest zielt auf die Überprüfung der versendeten Nachrichten der IoT-Geräte ab und soll die syntaktische und semantische Richtigkeit dieser bestätigen, wobei die semantische Interoperabilität ebenfalls eine Rolle spielt. Diese Phase korreliert mit dem Fokus, auf welchen bei der Erstellung des Testwerkzeuges dieser Bachelorarbeit Wert gelegt wird. Die Semantik zielt hierbei auf das Aufkommen von Ausreißern und unrealistischen Werten ab, wobei die Syntax beispielsweise Fokus auf falsch versendete Wertetypen legt. Dieser Ansatz kann mit seinen Beschreibungen als Wissensgrundlage für das in dieser Bachelorarbeit zu entwickelnde Testwerkzeug dienen.

Mit dem Ziel, öffentliche und private Dienstleistungen durch IoT-Tests zu verbessern, wurde durch Bosmans et al. [8] ein auf Hybridsimulationen basierender Testansatz beschrieben. Der Fokus bei diesem Ansatz liegt auf dem Testen der Auswirkungen des Verhaltens, der miteinander interagierenden lokalen Entitäten wie Sensoren, Aktuatoren und dem menschlichen Verhalten auf das

Gesamtsystem. Dabei sollen die lokalen Entitäten durch eine hybriden Ansatz simuliert werden. Vorteile des Simulationstests bilden die Anpassbarkeit, die Kosteneffizienz, eine hohe Skalierbarkeit sowie die Wiederverwendbarkeit der Simulationen. Dabei werden verschiedene Techniken zur Erhebung simulierter Daten beleuchtet. Der hier vorgestellte Testansatz beschreibt den Einsatz einer Kombination aus realen und simulierten Daten. Während der Entwicklungsphase sollte lediglich auf simulierte Daten zurückgegriffen werden, wohingegen im weiteren Verlauf immer mehr reale Daten verwendet werden sollten. Für Tests sollte somit zuerst ein stabiler Simulator geschaffen werden, welcher reale Werte generiert. Dieser Ansatz korreliert mit der Thematik dieser Bachelorarbeit, welche ebenfalls auf die Prüfung der dynamischen Auswirkungen von IoT-Applikationen mit Hilfe von Sensor-Simulationen abzielt. Das Verhalten der einzelnen IoT-Komponenten, welche sich auf das Verhalten des Gesamtsystems auswirken, umfasst ebenfalls ein Fehlverhalten und Anomalien dieser. Aus dieser Quelle können Motivationen und Ansätze extrahiert werden, welche für die folgende Arbeit hilfreich sind.

Von Reetz et al. [7] wird hervorgehoben, dass Anders als in klassischen Testverfahren innerhalb von IoT-Applikationen die Interaktion der einzelnen IoT-Komponenten mit der physischen Umgebung berücksichtigt werden muss. Da das effiziente Testen, wie in den vorherigen Quellen ebenfalls beschrieben, durch die Heterogenität der IoT-Komponenten leidet, wird der Prozess an einer zügigen Entwicklung und dem Einsatz von IoT-Applikationen gehindert. Aus diesem Grund wird hier ein halb-automatisierter Prozess zum Testen von IoT-Applikationen auf der Grundlage der *Code-Insertion*-Methodik zur Simulation von Sensoren beschrieben. Da es einen hohen Aufwand erfordert, Sensorwerte der physischen Geräte durch externe Eingriffe zu manipulieren, sollen die IoT-Komponenten mit ihren verschiedenen Verhaltensweisen simuliert werden. Dies schließt unter anderem unerwartetes Verhalten und spezifische Werte mit ein. Die Testumgebung, welche hier beschrieben wird, besteht aus drei Komponenten. Eine Komponente stellt die *Test Design Engine* dar, welche die Fähigkeit besitzt Testfälle aus der semantischen Beschreibung des Dienstes abzuleiten. Die weiteren Komponenten bilden die *Test Execution Engine*, welche unter einer kontrollierbaren Umgebung für die Ausführung der Tests verantwortlich ist, und die *Sandbox-Umgebung*, welche die Möglichkeit bietet, verschiedene Komponenten in Bezug auf ihre Kommunikation untereinander, die Systemlast und dem Netzwerkverhalten zu simulieren. Übereinstimmend zur Thematik dieser Bachelorarbeit wird hier ein Ansatz für das Testen der Interaktion innerhalb von IoT-Applikationen beschrieben, welche mit all ihren unerwarteten Verhaltensweisen und Anomalien in Bezug auf die Daten simuliert werden sollen. Aufgrund dessen bietet diese Arbeit eine Wissensgrundlage für das hier entstehende Testwerkzeug.

3.3. Tool-Auswahl

Die Integration des zu entwickelnden Testwerkzeugs mit Sensor-Simulator in die bestehende IoT-Plattform MBP sorgt dafür, dass der Fokus der Auswahl an Tools auf der Kompatibilität und der hürdenlosen Integration in diese liegt.

Für die Simulation einzelner Sensoren von IoT-Applikationen konnte keine Quelle für ein leicht zugängliches Tool mit einem akzeptablem Maß an Anpassungen ausgemacht werden, welches für die Zwecke dieser Arbeit verwendet werden könnte. Ein Grund dafür besteht zum einen in der Tatsache, dass Simulatoren häufig lediglich dazu bestimmt sind, den normalen Verlauf von Änderungen der externen Umwelt zu simulieren und nicht auf deren Anomalien eingegangen wird.

Zum anderen unterschieden sich die Architektur und Kommunikationssysteme von der innerhalb der MBP verwendeten. Infolge der geringen Dokumentation und Unübersichtlichkeit innerhalb der Implementierungen der vorgestellten Simulatoren ist die Einarbeitungszeit für eine Anpassung selbst für eine geringe Menge an Simulatoren erheblich. Aus diesem Grund bietet es sich an, eigene für diesen Anwendungsfall sowie zur IoT-Plattform MBP passende Sensor-Simulatoren für das Testen von IoT-Applikationen zu erstellen. Einige der genannten Arbeiten enthalten wiederverwendbare Methodiken und sind hilfreich für den Aufbau des Simulators.

Die in Abschnitt 3.2 vorgestellten Quellen in Bezug auf bereits existierende Testwerkzeuge für IoT-Applikationen verdeutlichen jeweils die Wichtigkeit und Notwendigkeit dieser Tests aufgrund der steigenden Anzahl an heterogenen IoT-Geräten und Größe von IoT-Applikationen. Die Simulation der Sensoren innerhalb von IoT-Applikationen wird durch die meisten Quellen aufgrund verschiedener Vorteile vorgeschlagen. Jedoch konnte kein konkretes Testwerkzeug, welches direkt in die MBP integrierbar wäre, gefunden werden. Grund dafür ist die Vielfalt an Lösungen der Architektur und Kommunikation von IoT-Applikationen und Plattformen, für die jeweils verschiedene Ansätze gefunden werden müssen. Die vorgestellten Quellen tragen jedoch dazu bei den richtigen Fokus für diese Art von Tests innerhalb des IoT zu finden und bieten somit eine Wissensgrundlage. Im Verlauf der Arbeit wurde aus diesen Gründen ein eigenes Testwerkzeug entwickelt, welches für die Architektur und das Kommunikationssystem der MBP geeignet ist.

4. Entwicklung eines adaptives Testwerkzeugs für IoT-Applikationen

In diesem Kapitel wird die Entstehung des adaptiven Testwerkzeugs für IoT-Applikationen sowie die Simulation der Sensoren und Aktuatoren erläutert. Da das Testwerkzeug und die Simulatoren in die MBP eingepflegt werden sollen, wird zunächst der Aufbau und die grobe Architektur der Plattform beleuchtet. Im Anschluss wird das Konzept der Integration des Testwerkzeugs und der Simulatoren in die MBP erläutert. Schließlich wird die Erstellung und Ausführung eines Tests mit Hilfe von Sensor-Simulatoren schrittweise anhand einer bestimmten Testmethodik erklärt.

4.1. Aufbau und Architektur der MBP

Wie bereits in Abschnitt 2.3 beschrieben, können innerhalb der MBP IoT-Applikationen modelliert werden. Hierfür ermöglicht die Plattform ein einfaches Anbinden von Sensoren und Aktuatoren von IoT-Geräten sowie die automatische Interaktion zwischen diesen durch benutzerdefinierte Regelungen. Für die Kommunikation zwischen den einzelnen Komponenten ist der *MBP-Core* zuständig [19]. Dieser nutzt für den Nachrichtenaustausch das in Abschnitt 2.2 beschriebene Kommunikationsprotokoll MQTT [19]. Abbildung 4.1 zeigt die Architektur und das Kommunikationssystem der MBP auf eine stark vereinfachte Weise.

Um eine IoT-Applikation zu modellieren, müssen die benötigten Sensoren, Aktuatoren und Regeln zur deren Interaktion an die Plattform angebunden und registriert werden. Um die Sensoren und Aktuatoren registrieren zu können, müssen im Voraus das zugehörige IoT-Gerät und die sogenannten benutzerdefinierten Extraction/Control Operators (Operatoren) registriert werden, welche diesen zugeordnet werden. Die Operatoren werden im Zuge des Deployments auf das jeweilige IoT-Gerät des Sensors oder Aktuators gelegt und mit Hilfe von *Lifecycle-Management-Skripts install.sh*, *start.sh*, *running.sh* und *stop.sh* automatisiert ausgeführt [31]. Durch die Verwendung der Operatoren ist es beispielsweise möglich, die durch den Sensor extrahierten Daten der externen Umgebung automatisch über das Kommunikationssystem des MBP-Core an die MBP zu senden, aber auch umgekehrt Befehle für Aktuatoren an den einzelnen Geräten zu erhalten [5]. Die Operatoren dienen somit als MQTT-Clients und stellen weitere vom Nutzer definierte Funktionen bereit. Bei der Implementierung der Operatoren steht die Wahl der Programmiersprache frei [31].

Da IoT-Geräte per Definition genau adressierbar und identifizierbar sein sollten, müssen bei der Registrierung dieser an die MBP unter anderem Angaben wie ein eindeutiger Name, der Gerätetyp, die IP-Adresse und der Private-Schlüssel einer RSA-Verschlüsselung gemacht werden [5]. Nachdem Sensoren und Aktuatoren angelegt wurden, können diese deployed, gestartet, gestoppt und, wie bereits erwähnt, über das Anlegen von benutzerdefinierten Regeln zu logischen Applikationen zusammengeschlossen werden. Diese Regeln basieren auf dem *event-condition-action*-Prinzip und können vom Nutzer einfach über die Benutzeroberfläche definiert werden [5]. Im Hintergrund

werden diese Regeln zu CEP-Queries, umgewandelt, wodurch die in die MBP eintreffenden Sensordaten kontinuierlich und gegenwartsnah im Hintergrund vom CEP-System verarbeitet werden können [5][15]. Erkennt das CEP-System im komplexen Datenstrom der Sensoren eine Verletzung der Regeln, so kann mit Hilfe der Aktuatoren reagiert werden [15]. Die MBP sendet über den MBP-Core die Informationen zur Handlung, welche zuvor durch die Regeln definiert wurde, an den spezifischen Aktuator weiter. Damit ist das Zusammenspiel von Sensoren und Aktuatoren über die MBP geregelt. Alle von Benutzern angegebenen Informationen innerhalb der Schritte zur Modellierung einer IoT-Applikation werden in der NoSQL-Datenbank MongoDB[1] gespeichert.

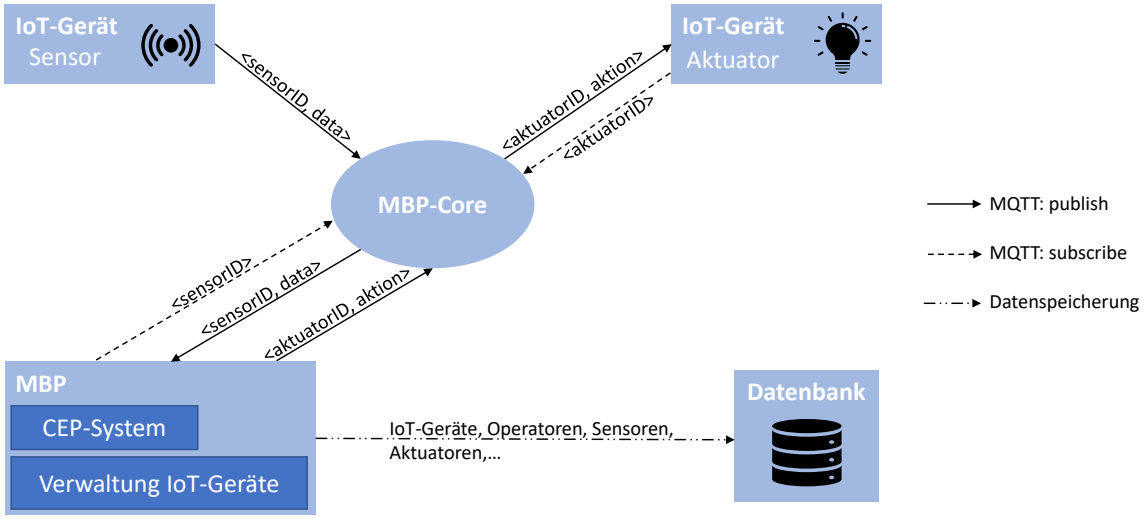


Abbildung 4.1.: Grobe Architektur und Funktionsweise der MBP

[1] https://www.mongodb.com/

4.2. Konzeption

In diesem Kapitel werden die beiden Konzepte zur Lösung des Haupt- und Teilproblems, welche in Abschnitt 1.1 beschrieben wurden, vorgestellt. Dabei wird zuerst auf die Entwicklung der Simulatoren und anschließend auf die des Testwerkzeugs eingegangen.

4.2.1. Simulatoren

Wie in Abschnitt 1.2 beschrieben, sollten Sensoren für verschiedene Anwendungsfälle simuliert werden, um ein effizientes Testen mit passenden Eingabedaten für verschiedene Testfälle gewährleisten zu können. Des Weiteren ist es sinnvoll für Tests von IoT-Applikationen einen Simulator für einen Aktuator bereitzustellen, welcher keinerlei Aktionen ausführt. Diese Simulatoren sollten für die Gewährleistung der Interaktion untereinander kommunikationsfähig sein.

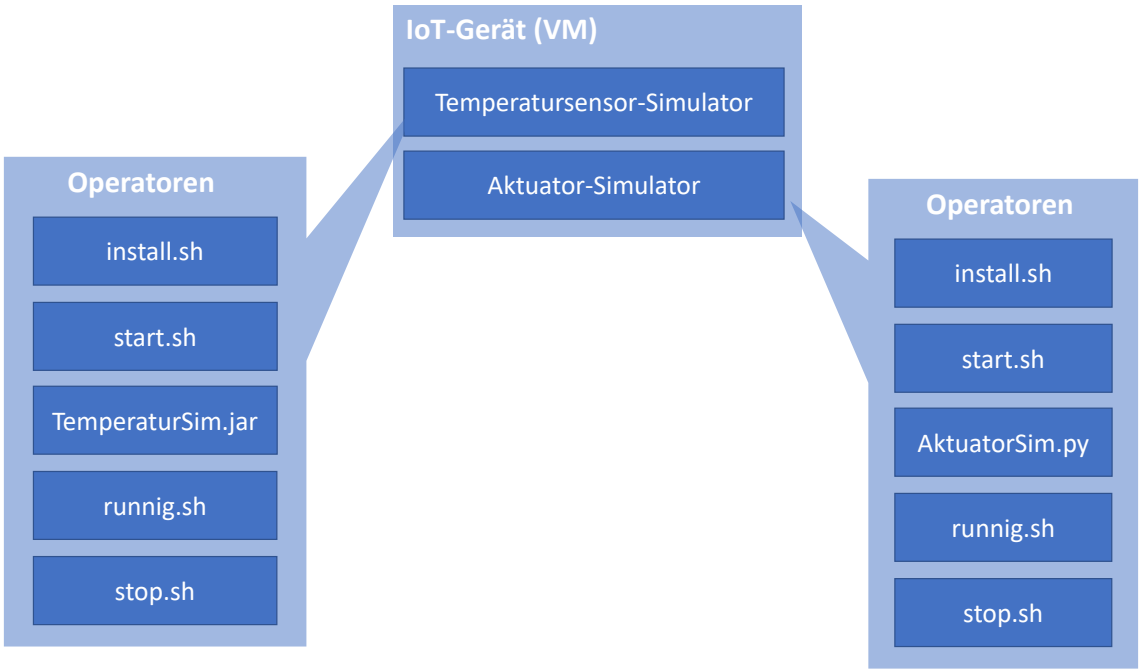


Abbildung 4.2.: Konzept Sensor- und Aktuator-Simulator

Durch Abbildung 4.2 wird das im Folgenden vorgestellte Prinzip mit Hilfe eines Beispieles visualisiert. Sie zeigt die Bereitstellung eines Simulators für einen Temperatursensor sowie für einen Aktuator-Simulator mit Hilfe einer Virtuellen Maschine (VM) und Operatoren. Für das Anbinden von Sensoren oder Aktuatoren an die MBP müssen wie in Abschnitt 4.1 beschrieben, IoT-Geräte und Operatoren zur Verfügung gestellt werden. Um im Zuge der Simulatoren kein echtes IoT-Gerät bereitstellen zu müssen, wird, wie in Abbildung 4.2 zu sehen, auf eine VM zurückgegriffen. Eine VM bringt alle Eigenschaften mit sich, welche für das Anlegen eines IoT-Gerätes innerhalb der MBP verlangt werden. Auf dieser soll die eigentliche Simulation über die Operatoren bereitgestellt werden.

Dafür werden bei der Simulation von Sensoren neben den Lifecycle-Management-Skripts selbst innerhalb dieser Arbeit implementierte Java[2]-JAR-Dateien zur Verfügung gestellt, welche als MQTT-Client und jeweiliger Simulator fungieren. Diese werden beim Deployment auf die VM gelegt und ausgeführt. Mit Hilfe von Parametern, welche über die Benutzeroberfläche an die Operatoren mitgegeben werden können, besteht die Möglichkeit die Simulatoren nach benutzerdefiniertem Anwendungsfall zu konfigurieren.

Für den Aktuator-Simulator wird dasselbe Prinzip angewandt, mit dem Unterschied, dass eine Python[3]-Datei bereitgestellt wird, welche ebenfalls als MQTT-Client dient, jedoch keinerlei Aktionen auslöst. Wie in Abbildung 4.2 zu sehen, können beide Simulatoren mit ihren jeweiligen Operatoren auf dieselbe VM gelegt werden.

Durch diese Arbeit werden Simulatoren durch Operatoren für verschiedene Arten von Sensoren sowie ein Aktuator-Simulator, welcher keinerlei Aktionen ausführt, bereitgestellt. Unter den Sensor-Simulatoren befinden sich sowohl eindimensionale als auch dreidimensionale Sensoren, wobei die Möglichkeit besteht die Dauer der Simulation sowie die Anzahl an Ereignissen und Anomalien zu bestimmen. Um Tests unter denselben Bedingungen wieder ausführbar zu machen, wird eine XML-Datei mit den zuletzt generierten Simulationswerten und den dazugehörigen Informationen des jeweiligen Simulators auf der VM abgelegt.

4.2.2. Testwerkzeug für IoT-Applikationen

Das zu entwickelnde Testwerkzeug soll innerhalb der Benutzeroberfläche der MBP für den Nutzer zur Verfügung stehen und die Möglichkeit bieten, Tests zu verwalten. Dies beinhaltet das Anlegen, Starten und Stoppen sowie das Löschen der Tests durch den Nutzer. Bevor ein Test über eine IoT-Applikation definiert werden kann, müssen die dazugehörigen Sensor-Simulatoren, der Aktuator-Simulator und die dementsprechenden Regeln über die Plattform angelegt werden. Beim anschließenden Anlegen des Tests werden durch das Testwerkzeug alle Informationen verlangt, welche für die Durchführung und Entscheidung eines erfolgreichen Zusammenspiels innerhalb von IoT-Applikationen notwendig sind. Dazu gehören Angaben wie:

- Sensor-Simulator
- Art der Simulation
- Wiederverwendung der letzten Sensordaten
- geplante oder zufällige Durchführung der Simulation
- Regeln, welche verletzt beziehungsweise nicht verletzt werden sollen
- Definition über den Erfolg des Tests

Wurden alle Angaben getätigt, kann der Test gestartet werden. Der Sensor-Simulator sendet nun dem Anwendungsfall entsprechende Daten über den MBP-Core an die MBP. Dort prüft die CEP zeitnah, ob die einzelnen Daten der Simulation gegen die definierten Regeln verstoßen und informiert gegebenenfalls den Aktuator-Simulator über den MBP-Core darüber. Um dem Nutzer nach

[2] https://www.java.com/
[3] https://www.python.org/

Ende des Tests einen aufschlussreichen Testbericht ausgeben zu können, werden im Hintergrund Detail-Informationen gesammelt. Zu diesen gehören unter anderem Informationen darüber, ob die ausgewählten Regeln durch die Simulation verletzt wurden oder nicht. Diese Information entscheidet darüber, ob der Test erfolgreich war beziehungsweise ob die definierte IoT-Applikation wie gewünscht arbeitet. Zusätzlich werden dem Nutzer weitere Detail-Informationen geliefert, welcher einer besseren Nachvollziehbarkeit des Testergebnisses dienen. Der Testbericht soll dem Nutzer nach dem Ende oder Stoppen des Tests über die Benutzeroberfläche zur Verfügung stehen. Abbildung 4.3 zeigt die Architektur der MBP mit der Einbindung beider Konzepte in diese. Die MBP erhält mit dem Testwerkzeug eine weitere Komponente wobei der benötigte Sensor und Aktuator der Applikation über eine VM, welche als IoT-Gerät fungiert, simuliert werden. Der bereits verwendeten Datenbank MongoDB werden zudem alle Informationen über die Tests übermittelt.

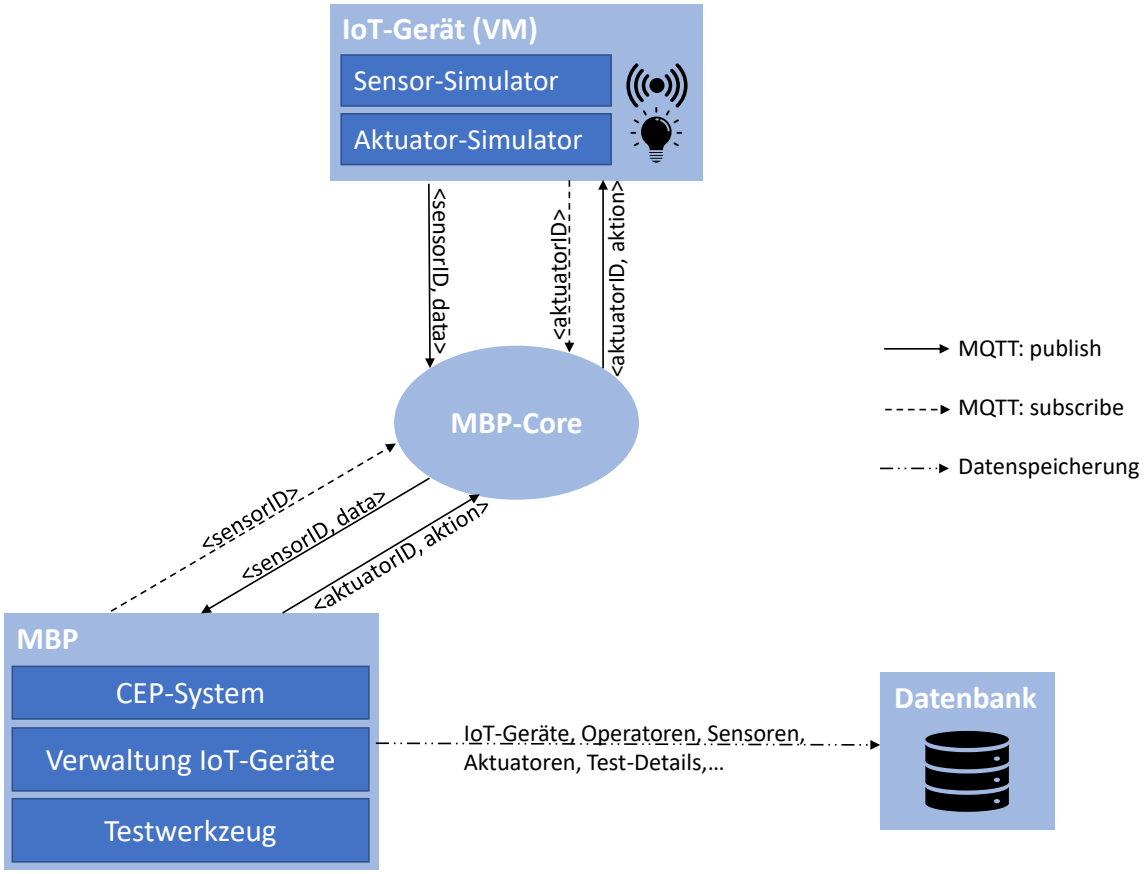


Abbildung 4.3.: Funktionsweise der MBP mit Testwerkzeug und Simulatoren

Die Implementierung des Testwerkzeugs wurde direkt im bestehenden System der MBP vorgenommen. Aus diesem Grund wurde die bestehende Auswahl der Tools zur Entwicklung übernommen. Während für die Weboberfläche das Open-Source-Framework AngularJS[4] zu Verwendung kam, wurde das Java-Backend mit Hilfe des Spring-Boot-Frameworks[5] implementiert.

[4] https://angularjs.org/
[5] https://spring.io/

4.3. Testmethodik

Mit der in Abbildung 4.4 veranschaulichten Testmethodik wird das strukturierte Vorgehen zur Erstellung und Durchführung eines Tests mit dem in dieser Arbeit entwickelten Testwerkzeug und der MBP für IoT-Applikationen vorgestellt. Wie in Abbildung 4.5 zu sehen, ist das Testwerkzeug innerhalb der MBP über den letzten Menüpunkt *Testing-Tool* zu erreichen, wobei auf der Hauptseite aufgeführt wird, wie viele Tests bisher registriert wurden.

An erster Stelle der Testmethodik in Abbildung 4.4 steht der Nutzer, welcher mit der MBP und dem darin integrierten Testwerkzeug interagiert. Durch ihn werden vorerst Testfälle definiert und weitere Vorbereitungen innerhalb der MBP getroffen. Zu den Vorbereitungen des Tests einer IoT-Applikation gehört die Registrierung des Sensor- und Aktuator-Simulators mit Hilfe der VM und der spezifischen Operatoren. Sind diese Schritte getan, werden die Regeln definiert, welche die Interaktion der beiden Komponenten organisiert. Im Anschluss an diese vorbereitenden Schritte kann der entsprechende Test über das integrierte Testwerkzeug verwaltet werden. Wurde der Test gestartet und abgeschlossen oder gestoppt, ist es möglich den Testbericht einzusehen. Über diesen wird der Nutzer informiert, ob der Test erfolgreich abgeschlossen wurde und liefert weitere detaillierte Hintergrundinformationen darüber. Abschließend liegt die Entscheidung beim Nutzer, auf das Ergebnis des Tests zu reagieren, Maßnahmen zu ergreifen oder gegebenenfalls einen erneuten Test durchzuführen.

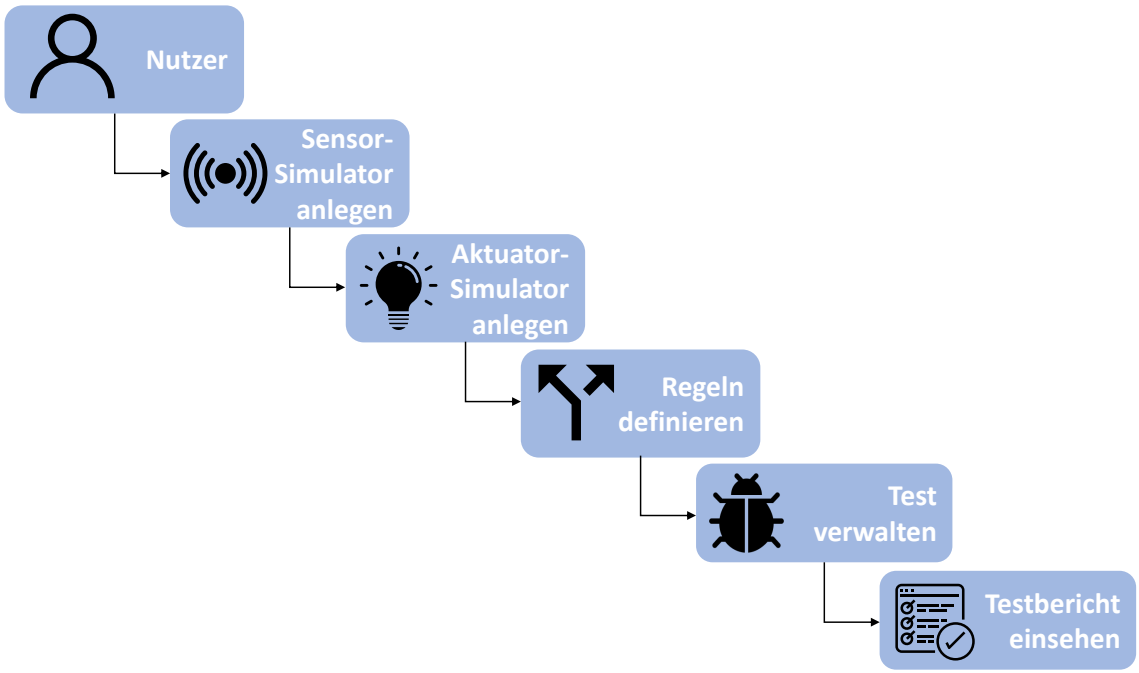


Abbildung 4.4.: Testmethodik

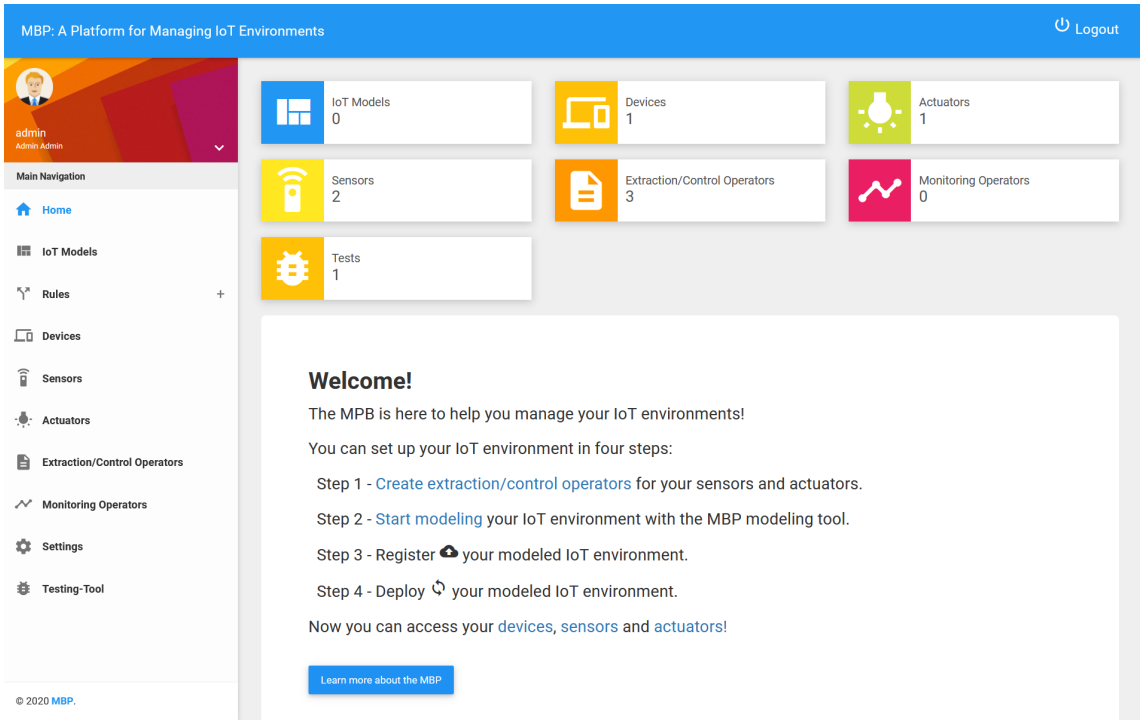


Abbildung 4.5.: Benutzeroberfläche der MBP mit integriertem Testwerkzeug

4.3.1. Nutzer

Der Nutzer steht an erster Stelle des Testverlaufs und führt alle drauf folgenden Schritte aus. Ihm wird vorerst die Aufgabe zuteil, gewisse Vorbereitungen zu treffen, um einen effektiven und strukturierten Test durchführen zu können [21]. Zu diesen Vorbereitungen zählt, wie bereits in Abschnitt 2.4 erwähnt, die Definition eines Testfalls [21]. Ist dieser definiert, so steht der Nutzer in Interaktion mit der MBP und legt als weiteren vorbereitenden Schritt die zu testende IoT-Applikation an. Im Anschluss kann mit dem in der Benutzeroberfläche integrierten Testwerkzeug interagiert und der gewünschte Test verwaltet werden. Im Folgenden wird ein Beispiel-Testfall angelegt, welcher in den Schritten der weiter beschriebenen Testmethodik zum besseren Verständnis fortgeführt wird.

Ein im IoT-Bereich unerfahrener Nutzer möchte mit Hilfe einer IoT-Applikation innerhalb der MBP-Plattform die optimale Raumtemperatur für sein Büro innerhalb des Wohnhauses gewährleisten. Angenommen wird, dass die optimale Temperatur für diese Räumlichkeit zwischen 20.0-23.0°C liegt. Werden die Grenzen unter- beziehungsweise überschritten, soll entweder die Heizung oder die Klimaanlage aktiviert werden. Der Nutzer legt zunächst für sich sinnvolle Regeln fest, um die Interaktion zwischen Sensoren und Aktuatoren zu koordinieren und eine IoT-Applikation zu definieren. Mit Hilfe des Testwerkzeugs soll die von ihm entworfene IoT-Applikation auf Stabilität und Korrektheit geprüft werden. Dabei möchte der Nutzer mögliche Anomalien, welche der Sensor mit sich bringt, nicht außer Acht lassen. Die folgenden Informationen werden damit durch den Nutzer für den Testfall festgelegt:

- **Sensor-Simulator**: Temperatursensor
- **Simulationsart**: Temperaturabfall auf 18.0°C
- **Anomalie**: Ausreißer-Wert
- **Regeln**:
  - Steigt die Temperatur auf >23.0°C an, schalte die Klimaanlage ein
  - Sinkt die Temperatur auf <20°C ab, schalte die Heizung ein
- **Erfolgreich, wenn**: Aktuator-Aktion wird lediglich bei der Über- oder Unterschreitung von 23°C oder 20°C ausgelöst. Die simulierte Anomalie hat keinen Einfluss auf die Interaktion der IoT-Applikation.

4.3.2. Sensor-Simulator anlegen

Wurde der Testfall definiert, so befindet sich der Nutzer in Interaktion mit der MBP. In diesem Schritt, wird der Sensor-Simulator angelegt, was die Registrierung der VM als IoT-Gerät und die zum gewählten Sensor spezifischen Operatoren als Simulatoren beinhaltet. Die VM ist eindeutig identifizierbar und wird mit Hilfe von IP-Adresse, Gerätetyp, Nutzername, dem Private-RSA-Schlüssel sowie einem Namen angelegt.

Um einen Sensor auf der VM gemäß Abschnitt 4.2.1 simulieren zu können, wurden ausführbare Java-JAR-Dateien zu verschiedenen Arten von Sensoren entwickelt. Da jedoch eine Vielzahl an verschiedenen Sensoren existiert, welche für die Simulation der Sensordaten jeweils unterschiedlich behandelt werden müssten, wurde die Anzahl der Sensoren, welche im Rahmen dieser Bachelorarbeit zum Einsatz kommen, beschränkt. Entwickelt wurden eindimensionale Temperatur- und Feuchtigkeitssensoren, aber auch dreidimensionale GPS- und Beschleunigungssensoren. All diese generieren Sensorwerte mit dem Datentyp Floating-Point. Die entwickelten Simulatoren sind dazu in der Lage realitätsnahe Änderungen der externen Umgebung nachzustellen und diese gegebenenfalls mit Anomalien zu versehen. Als eine Anomalie innerhalb der Sensordaten werden

- *Ausreißer-Werte*, welche über beziehungsweise unter der maximalen Änderungsrate von aufeinanderfolgenden Werten liegen,
- *falsche Wertetypen*, welche statt eines Floating-Points beispielsweise einen String liefern,
- *Ausfälle*, wobei Werte nicht nach der eigentlichen Frequenz geliefert werden,

angesehen. Für jeden Sensor wurden zunächst durch Recherchen Informationen beschafft, welche für eine realitätsnahe Simulation von Bedeutung sind. Obwohl jeder Sensor einen anderen Anwendungsbereich besitzt, können einige Parameter definiert werden, welche für jeden Simulator von Bedeutung sind:

- **Frequenz**: Frequenz, mit der Sensoren Werte der externen Umgebung messen
- **Messbereich**: Die Ober- und Untergrenze der Messwerte von Sensoren
- **Simulationsart**: Welche Änderung der externen Umgebung simuliert werden soll
- **Anomalien**: Art der Anomalie, die simuliert werden soll

- **Maximale Änderungsrate**: Größter zulässiger Unterschied zwischen zwei aufeinander folgenden Messungen, um reelle Werte simulieren zu können
- **Simulationszeit**: Wie lange die Simulation für einen Test andauern soll
- **Anzahl der Ereignisse**: Wie oft die Änderung der externen Umgebung simuliert werden soll
- **Anzahl der Anomalien**: Wie oft die Anomalie des Sensors simuliert werden soll

Parameter wie die Frequenz, der Messbereich und die maximale Änderungsrate werden im Hintergrund für die einzelnen Sensoren definiert. Für die Frequenz und den Messbereich wurden Internetrecherchen und Vergleiche verschiedener gleichartiger Sensoren angestellt und die hier gängigen Werte übernommen. Im Gegensatz dazu konnten für die maximale Änderungsrate der unterschiedlichen Sensoren keine Quellen gefunden werden, welche realistische Werte dieser vorgeben. Aus diesem Grund wurden für diesen Parameter jeweils eigene realitätsnahe Annahmen getroffen, um möglichst reelle Sensorwerte generieren zu können.

Die Simulationsart, die Art der Anomalie, sowie die Simulationszeit und Anzahl der Ereignisse und Anomalien müssen im Gegensatz dazu durch den Nutzer innerhalb der MBP an die Operatoren mitgegeben werden. Die Angaben über die Simulationszeit sowie die Anzahl an Ereignissen und Anomalien sind optional. Durch sie können Tests geplant durchlaufen werden. Wird diese Konfiguration jedoch nicht in Anspruch genommen, so laufen die Simulationen jeweils nur über einen kurzen Zeitabschnitt mit jeweils einem Ereignis und einer Anomalie.

Im Anschluss folgt eine kurze Erklärung über den Aufbau und Anwendungsbereich der einzelnen Simulatoren, um einen Einblick zu erhalten, für welche Anwendungsfälle diese einsetzbar sind. Für alle Simulatoren wird angenommen, dass sie für die Verwendung von IoT-Applikationen eines Smart-Homes verwendet werden sollen.

Temperatursensor

Um wie im Beispiel-Testfall IoT-Applikationen mit einem Temperatursensor aufbauen zu können, wurde ein Simulator für diesen entwickelt. Dieser Simulator ist beispielsweise für IoT-Applikationen einsetzbar, welche die optimale Raumtemperatur der verschiedenen Räumlichkeiten innerhalb eines Smart-Homes garantieren.

Grundlegend existieren zwei verschiedenen Ereignisse, welche durch einen solchen Sensor aus der externen Umwelt extrahiert werden können. Zum einen der Temperaturanstieg, zum anderen der Temperaturabfall. Die Simulation dieser Ereignisse soll jeweils mit den genannten Anomalien kombiniert werden können, um die Applikationen auf Stabilität und Korrektheit prüfen zu können. Um den Simulator realitätsnah aufbauen zu können, wurden die folgenden Parameter festgelegt:

- **Frequenz**: Die Frequenz eines Temperatursensors liegt üblicherweise bei 1 Hz. Dies entspricht einer periodischen Messung pro Sekunde.
- **Maximale Änderungsrate**: Es wurde eine Annahme über eine realistische Änderung der Raumtemperatur getroffen welche besagt, dass die Raumtemperatur um nicht mehr als 1.5°C pro Minute sinkt oder steigt. Aus dieser Annahme heraus kann die maximale Änderungsrate von 0.025°C pro 1 Hz berechnet werden.

- **Messbereich**: Gängige Temperatursensoren sind dazu in der Lage, Temperaturen von etwa -55.0°C bis 125.0°C zu messen. Diese Grenzen werden hier übernommen.

Soll eine vom Nutzer ungeplante Simulation über einen Temperaturabfall bzw. -anstieg stattfinden, so startete dieser bei einem zufälligen Wert kurz über bzw. unter der jeweiligen Grenze der optimalen Raumtemperatur des Raumes innerhalb des Smart-Homes. Die Grenzen der optimalen Raumtemperatur wurden ebenfalls durch Internetrecherchen und Vergleiche verschiedener Angaben im Hintergrund definiert. Um eine Reihe an aufeinanderfolgenden, realitätsnahen Werten zu simulieren, wurde folgende Berechnung (4.1) vorgenommen. Auf den aktuellen Wert wird eine zufällige Zahl zwischen Null und der maximalen Änderungsrate addiert beziehungsweise subtrahiert, um den darauffolgenden Temperaturwert zu erhalten. Die Berechnung 4.1 wird so lange fortgeführt, bis der jeweilige Grenzwert über- beziehungsweise unterschritten wurde:

(4.1) Folgender Wert = aktueller Wert ± random(0, maximale Änderungsrate)

Nimmt der Nutzer die Möglichkeit einer geplanten Simulation wahr, kann die Simulationslänge, aber auch die Anzahl an Ereignissen und Anomalien konfiguriert werden. Dabei wird eine periodische Temperaturkurve generiert, welche innerhalb der zeitlichen Rahmenbedingungen alle vom Nutzer gewünschten Ereignisse simuliert. Im Zuge dessen finden im Hintergrund verschiedene Berechnungsschritte statt, welche beispielsweise die zeitliche Berechnung der Simulation eines einzelnen Ereignisses beinhaltet. Um unrealistische Temperaturabfälle oder -anstiege innerhalb der Gesamtzeit der Simulation zu verhindern, wird eine maximale Temperaturänderung zwischen Anfang und Start von 15°C definiert. Soll somit innerhalb einer Stunde ein einziger Temperaturabfall simuliert werden, sinkt die Temperatur nicht in unrealistische Tiefen, sondern maximal um 15°C ab.

Anomalien werden an zufälligen Stellen innerhalb der generierten Sensorwerte platziert. Wird ein Ausreißer gewählt, wird ein zufälliger, weit über der maximalen Änderungsrate liegender Wert berechnet. Damit wird garantiert, dass der Ausreißer-Wert kein gültiger Folge-Wert ist. Die Simulation eines falschen Wertetyps gibt im Gegensatz dazu statt einer gültigen Floating-Point Zahl ein String (*error*) aus. Soll die Anomalie eines fehlenden Wertes simuliert werden, so handelt es sich um einen kurzen Ausfall des Sensors. Wird diese Art der Anomalie gewählt, werden an zufälligen Stellen leere Messungen übertragen.

Feuchtigkeitssensor

Der Simulator eines Feuchtigkeitssensors wurde entwickelt, um die IoT-Applikationen definieren zu können, welche die optimale Raumluftfeuchtigkeit innerhalb der verschiedenen Räumlichkeiten eines Smart-Homes garantieren. Dabei erfolgt die Generierung der Simulation analog zum Simulator des Temperatursensors. Gleichermaßen kann der Nutzer zwischen einer geplanten und ungeplanten Simulation wählen, in welcher zwei Ereignisse in möglicher Kombination mit Anomalien generiert werden. Diese Ereignisse stellen den Feuchtigkeitsanstieg und -abfall dar. Die Parameter der Frequenz, die maximale Änderungsrate sowie der Messbereich müssen jedoch angepasst werden:

- **Frequenz**: Die Frequenz eines Feuchtigkeitssensors liegt üblicherweise bei 0.5 Hz. Dies entspricht einer periodischen Messung zweimal pro Sekunde.

- **Maximale Änderungsrate**: Es wurde eine Annahme über eine realistische Änderung der Luftfeuchtigkeit innerhalb eines Raumes getroffen, welche besagt, dass diese um nicht mehr als 0.07% pro Minute fällt oder steigt. Dies entspricht einer maximalen Änderungsrate von 0.035% pro Messung.
- **Messbereich**: Gängige Feuchtigkeitssensoren sind dazu in der Lage, die Feuchtigkeit im Bereich von etwa 20.0% bis 95.0% zu messen. Diese Grenzen werden hier übernommen.

GPS-Sensor

Bei einem GPS-Sensor handelt es sich um einen dreidimensionalen Sensor, wobei die Dimensionen durch Koordinaten (Breitengrad, Längengrad, Höhe) in Dezimalgrad-Darstellung repräsentiert werden. Da die Simulatoren für die Verwendung innerhalb von IoT-Applikationen eines Smart-Homes vorgesehen sind, wurden die Ereignisse dementsprechend festgelegt. Es soll sowohl die Entfernung als auch die Annäherung eines Hausbewohners innerhalb eines gewissen Radius um dieses simuliert werden können, um mit Hilfe einer IoT-Applikation darauf reagieren zu können. Dabei wird davon ausgegangen, dass sich der Nutzer zu Fuß nähert beziehungsweise entfernt.

Um einen realistischen Weg im Koordinatensystem für die Simulation dieser Ereignisse generieren zu können, wurde die Methode der Wegpunktprojektion gewählt. Mithilfe dieser ist es möglich, von einem Ausgangspunkt ausgehend mit einem bestimmten Offset den neuen Punkt im Koordinatensystem zu bestimmen [32]. Dieses Offset besteht aus 2 Komponenten: der Entfernung, wie viele Meter vom ersten Punkt bis zum zweiten Punkt zurück gelegt werden sollen, und dem Richtungswinkel, welcher in Grad angibt, in welche Himmelsrichtung der Weg simuliert werden soll (Norden: 0°, Osten: 90°, Süden: 180°, Westen: 270°) [32]. Diese Methode wird in den meisten GPS-Geräten verwendet [33]. Erneut werden Parameter definiert, welche für eine realistische Simulation notwendig sind:

- **Frequenz**: Die Frequenz eines GPS-Sensors liegt üblicherweise bei 50 Hz. Dies entspricht einer periodischen Messung 50 mal pro Sekunde.
- **Maximale Änderungsrate**: Die maximale Änderungsrate wird im Offset der Wegpunktprojektion benötigt, um zu definieren wie viele Meter maximal vom Hausbewohner zurückgelegt werden. Da die Schrittgröße einer Person mit der Größe von 1,70-1,95m ungefähr bei einem Meter liegt, wurde die maximale Änderungsrate pro Messung wie folgt berechnet: Es wird angenommen, dass pro Sekunde ein Schritt mit der Länge eines Meters zurückgelegt wird. Da pro Sekunde 50 Messungen durchgeführt werden, ergibt sich die maximale Änderungsrate von 0.02 Meter (2 cm) pro Messung.

Wird bei einer ungeplanten Simulation die Entfernung von einem Smart-Home gewünscht, so wird mit Hilfe der Wegpunktprojektion ein gerader Weg in eine zufällige Himmelsrichtung simuliert. Die Länge dieses Weges wird über den vom Nutzer festgelegten Radius, ab welchem innerhalb der IoT-Applikation reagiert werden soll, bestimmt. Schließlich werden immer 2 cm pro Messung in eine gleich bleibende Himmelsrichtung gegangen, bis der Radius überschritten wurde. Gleiches passiert bei der Simulation der Annäherung an das Smart-Home, nur in die entgegengesetzte Richtung.

Soll die Simulation geplant verlaufen, so wird die Annäherung und Entfernung des Bewohners periodisch innerhalb des zeitlichen Rahmens simuliert. Dabei wird gewährleistet, dass die gewünschte Anzahl der Ereignisse generiert wird.

Wird vom Nutzer definiert, dass die Simulation eines Ereignisses mit einer Anomalie kombiniert werden soll, hat dies die folgenden Bedeutungen: Im Falle der Auswahl eines Ausreißer-Wertes, wird ein zufälliger Punkt zwei Kilometer weit entfernt in eine beliebige Himmelsrichtung simuliert. Dies stellt eine unrealistische Bewegung dar, da es einem Menschen nicht möglich ist 2 Kilometer innerhalb von 20 Millisekunden zu Fuß zurückzulegen. Sollen fehlende Werte simuliert werden, welche beispielsweise ein GPS-Funkloch darstellen, werden an beliebigen Stellen keine Koordinaten erzeugt. Bei der Anomalie, dass ein falscher Wertetyp generiert wird, wird wiederum ein String (*error*) statt eines Floating-Point Wertes ausgegeben. Diese Anomalien werden an zufälligen Stellen eingebunden.

Beschleunigungssensor

Ein weiterer dreidimensionaler Sensor, welcher durch einen Simulator entwickelt wurde, ist der Beschleunigungssensor. Beschleunigungssensoren sind häufig Teil eines Alarmanlagensystems und können auch in Kombination mit bestimmten Aktuatoren innerhalb einer IoT-Applikation als eine solche fungieren. Dieser Sensor-Simulator soll für eine IoT-Applikation verwendet werden können, welche ein einziges Objekt bewacht.

Wieder werden zwei Ereignisse detektiert, welche durch den Simulator generiert werden sollen. Zum einen die Bewegung des Objekts, welches den Fall eines Diebstahls simuliert und zum anderen das Objekt in Ruhe. Diese Ereignisse können gleich wie in den anderen Simulatoren mit Anomalien kombiniert werden. Da sich ein Objekt in einem dreidimensionalen Raum befindet, muss der Sensor eine Beschleunigung in jede Richtung wahrnehmen können. Das bedeutet, dass auf

- der X-Achse Bewegungen nach links oder rechts,
- der Y-Achse Bewegungen nach vorne oder nach hinten
- und der Z-Achse Bewegungen nach oben oder unten

wahrgenommen werden. Auf ein Objekt in Ruhe wirkt sich lediglich die Fallbeschleunigung entlang der Z-Achse aus. Diese beträgt $9.81\frac{m}{s^2}$ (1g) [34]. Steht das Objekt ohne Bewegungen auf einer Oberfläche, lassen sich mit dieser Information die drei Achsen mit folgenden Werten darstellen:

$$[0, 0, 1]$$

Das Objekt, welches bewacht werden soll, befindet sich in seiner Ausgangssituation in Ruhe. Wird es einer Bewegung ausgesetzt, welche einen Diebstahl darstellen könnte, so soll ein Alarm ausgelöst werden. Hierbei ist zu beachten, dass das zu bewachende Objekt auch anderen Bewegungen, wie zum Beispiel Vibrationen eines vorbeifahrenden LKWs, ausgesetzt sein kann. Die Alarmanlage muss somit erkennen, wann das Objekt einer tatsächlichen Bewegung ausgesetzt ist und wann es sich lediglich um eine Bewegung durch andere Einflüsse der externen Umwelt handelt. Des Weiteren müssen wiederum die Parameter Frequenz und maximale Änderungsrate festgelegt werden:

- **Frequenz**: Die Frequenz eines Beschleunigungssensors liegt üblicherweise bei etwa 40 Hz. Dies entspricht einer periodischen Messung 40 mal pro Sekunde.
- **Maximale Änderungsrate**: Als maximale Änderungsrate werden hier 3g angenommen, was ungefähr 14 m/s entspricht.

Zur Simulation einer Bewegung soll vom Nutzer eingestellt werden können, mit welcher Sensitivität innerhalb der IoT-Applikation auf diese reagiert werden soll. Wird eine Bewegung simuliert, wird von einer konstanten Beschleunigung ausgegangen. Im Hintergrund wird dazu berechnet, wie viel Kraft aufgebracht werden muss, um einen Gegenstand mit benutzerdefinierter Masse so zu beschleunigen, dass ein Alarm ausgelöst wird. Dazu kann das zweite Newtonsche Gesetz verwendet werden, welches besagt, dass die Beschleunigung a der Quotient aus Kraft F und Masse m ist ($a = \frac{F}{m}$) [35]. Die Kraft F, welche aufgewendet werden muss, lässt sich durch die Einheit Newton $N = \frac{m}{s^2} * kg$ berechnen [36].

Die konstante Beschleunigung für die Simulation wird als zufälliger Wert zwischen der berechneten Beschleunigung, ab welcher ein Alarm ausgelöst wird, und der maximalen Obergrenze gewählt. Als weiteres Indiz eines Diebstahls soll angegeben werden können, ab wie vielen zurückgelegten Metern es sich um einen solchen handelt. Die Simulation endet, wenn mit dieser Beschleunigung die angegebene Meterzahl überschritten wird.

Soll die Simulation geplant durchlaufen werden, kann der Nutzer die Simulationszeit sowie Anzahl der Ereignisse und Anomalien angeben. Dadurch werden im Wechsel eine konstante Beschleunigung, welche über der Sensitivitätsgrenze liegt und die Grenze in Metern überschreitet, sowie das Objekt in Ruhe simuliert.

Definiert der Nutzer, dass die Simulation eines Ereignisses mit einer Anomalie kombiniert werden soll, kann ein Umwelteinfluss, welcher eine Beschleunigung auslöst aber keinen Diebstahl darstellt, simuliert werden. Dieser Umwelteinfluss wird durch das Stoßen einer Fliege gegen das Objekt mit einer Beschleunigung über der maximalen Änderungsrate dargestellt. Des Weiteren können Ausreißer-Werte, welche eine Beschleunigung von 25g darstellen, sowie falsche Wertetypen wie ein String (*error*) statt einer Floating-Point Zahl, integriert werden. Wie durch die vorher beschriebenen Simulatoren werden diese Anomalien an zufälligen Stellen der simulierten Daten eingepflegt.

Für die eindimensionalen Temperatur- und Feuchtigkeitssensor-Simulatoren werden jeweils eine ausführbare Java-JAR-Datei für die geplante und ungeplante Simulation erstellt. Da die MBP lediglich eindimensionale Sensoren unterstützt, werden für den GPS- und Beschleunigungssensor-Simulator drei Java-JAR-Dateien erstellt, welche jeweils eine andere Achse repräsentieren, ebenfalls für die geplante und ungeplante Simulation. Dementsprechend müssen drei verschiedene Sensoren innerhalb der MBP angelegt und die Regeln entsprechend angepasst werden. Die benötigten Parameter der Simulatoren werden bei der Einbindung der Operatoren angegeben und über das Testwerkzeug definiert. Diese werden bei Deployment und Ausführung der Sensoren an die Java-JAR-Datei übertragen. Damit kann mit Hilfe der Parameter definiert werden, welche Daten ein Simulator generieren soll. Die vom Simulator auf der VM generierten Daten werden über den MBP-Core an die MBP gesendet, sowie innerhalb einer XML-Datei auf der VM gespeichert.

Wird der in Abschnitt 4.3.1 beschriebene Testfall fortgeführt, ist es die Aufgabe des Nutzers, den Simulator eines Temperatursensors innerhalb der MBP anzulegen. Hierfür werden eine VM und die bereitstehenden Operatoren des Temperatursensor-Simulators registriert. Um die Simulationsart und weitere Informationen innerhalb des Testwerkzeuges definieren und für die Weiterverarbeitung an den Simulator senden zu können, müssen beim Registrieren der Operatoren folgende Parameter angelegt werden:

- *event* (Number): Gibt die Art des zu simulierenden Ereignisses an
- *anomaly* (Number): Gibt die Art der Anomalie an
- *room* (Text): Gibt den Raum an, in dem die Temperatur gemessen werden soll
- *useNewData* (Switch): Gibt an, ob die zuletzt simulierten Daten wiederverwendet werden sollen

4.3.3. Aktuator-Simulator anlegen

Der dritte Schritt der Testmethodik stellt das Anlegen des Aktuators-Simulators dar. Hierfür werden wie im ersten Schritt (Abschnitt 4.3.2) die VM, welche als IoT-Gerät fungiert, und bestimmte Operatoren benötigt. Neben den vorausgesetzten Lifecycle-Management-Skripts wird innerhalb der Operatoren eine Python-Datei bereitgestellt, welche lediglich als MQTT-Client dient und sich auf die ID des Aktuators innerhalb der MBP subscribed. Damit ist die Kommunikationsfähigkeit zwischen Sensoren und Aktuatoren über die MBP gewährleistet. Wird also eine Regel durch die an die Plattform gesendeten Simulationsdaten verletzt, wird der Aktuator-Simulator über den MBP-Core darüber informiert, jedoch keine Aktion ausgelöst.

Innerhalb des Beispiel-Testfalles wird der Aktuator-Simulator durch den Nutzer angelegt. Hierfür kann dieselbe VM verwendet werden, welche für die Registrierung des Sensor-Simulators angelegt wurde, sowie die hier beschriebenen Operatoren. Damit kann der Aktuator-Simulator für diesen Testfall sowohl als Klimaanlage also auch als Heizung fungieren.

4.3.4. Regeln definieren

Nachdem die Simulatoren für Sensoren und Aktuatoren in der MBP angelegt wurden, müssen innerhalb der Benutzeroberfläche benutzerdefiniert Regeln erstellt werden, um die Komponenten zu sinnvollen IoT-Applikationen zu verknüpfen. Dieser Schritt stellt eine Voraussetzung für das effiziente Verwenden des Testwerkzeuges dar. Innerhalb des *event-condition-action*-Prinzips, durch das die Regeln aufgebaut sind, hat dies die Bedeutung, dass

- das *event* durch die Simulation der einzelnen Sensordaten,
- die *condition* durch die gewünschten Regelungen, bei welchem Event reagiert werden soll,
- und die *action* durch den Aktuator-Simulator ohne Aktionen

repräsentiert wird [5].

In Bezug des Beispiel-Testfalles werden vom Nutzer zwei einfach gehaltene Regeln, wie in Abschnitt 4.3.1 definiert, innerhalb der Benutzeroberfläche der MBP-Plattform angelegt. Diese Regeln sind wie folgt aufgebaut:

1. Regel:

- *Event*: Einzelne Werte des Temperatursensor-Simulators
- *Condition*: 1 Event >23°C
- *Action*: Schalte die Klimaanlage an (Aktuator-Simulator)

2. Regel:

- *Event*: Einzelne Werte des Temperatursensor-Simulators
- *Condition*: 1 Event <20°C
- *Action*: Schalte die Heizung an (Aktuator-Simulator)

4.3.5. Interaktion mit dem Testwerkzeug

Wurden alle vorbereitenden Schritte getätigt, so können die zu den IoT-Applikationen definierten Tests über die Benutzeroberfläche der MBP verwaltet werden. Über den Menüpunkt *Testing-Tool* gelangt man zu dem in Abbildung 4.6 zu sehenden Testwerkzeug. Der Nutzer wird zu Beginn der Seite über die Vorgehensweise aufgeklärt, mittels derer einen Test anzulegen und auszuführen ist. Dabei wird die Verwaltung der Tests in Tabellenform gehandhabt, welche im unteren Bereich der Seite zu finden ist. Dem Nutzer steht hier die Möglichkeit einen Test

- anzulegen,
- zu löschen,
- zu starten,
- zu stoppen,
- zu wiederholen

und den resultierenden Testbericht einzusehen zur Verfügung.

Soll ein neuer Test einer IoT-Applikation angelegt werden, öffnet sich über den Plus-Button, welcher in Abbildung 4.6 oben rechts in der Tabelle gefunden werden kann, ein Popup-Fenter, worüber die Testdetails definiert werden können. Dazu gehören:

- ein eindeutiger Testname,
- der zu simulierende Sensor mit dynamischen Drop-Down-Feldern zur Angabe der Simulationsart,
- die Auswahl, welche Regeln der IoT-Applikation durch die Simulation verletzt/nicht verletzt werden sollen.

Der angelegte Test erscheint nach der Registrierung in der Verwaltungs-Tabelle. Von hier aus besteht schließlich die Möglichkeit den Test zu starten.

Nach Start des Tests durch den Nutzer werden im Hintergrund nacheinander der Aktuator- und der Sensor-Simulator deployed und gestartet. Beim Deployen des Sensor-Simulators werden die vom Nutzer definierte Simulationsart über die bei der Registrierung der Operatoren definierten Parameter an die VM übermittelt. Durch den Start des Sensor- sowie Aktuator-Simulator und die angelegten Regeln wird dann das Zusammenspiel der IoT-Applikation unter bestimmten Bedingungen im Hintergrund simuliert. Dabei werden alle Informationen gesammelt, welche für einen informativen Testbericht notwendig sind. Dieser wird erstellt, sobald die Simulation und somit der Test durchlaufen beziehungsweise gestoppt wird und der Test somit beendet wurde.

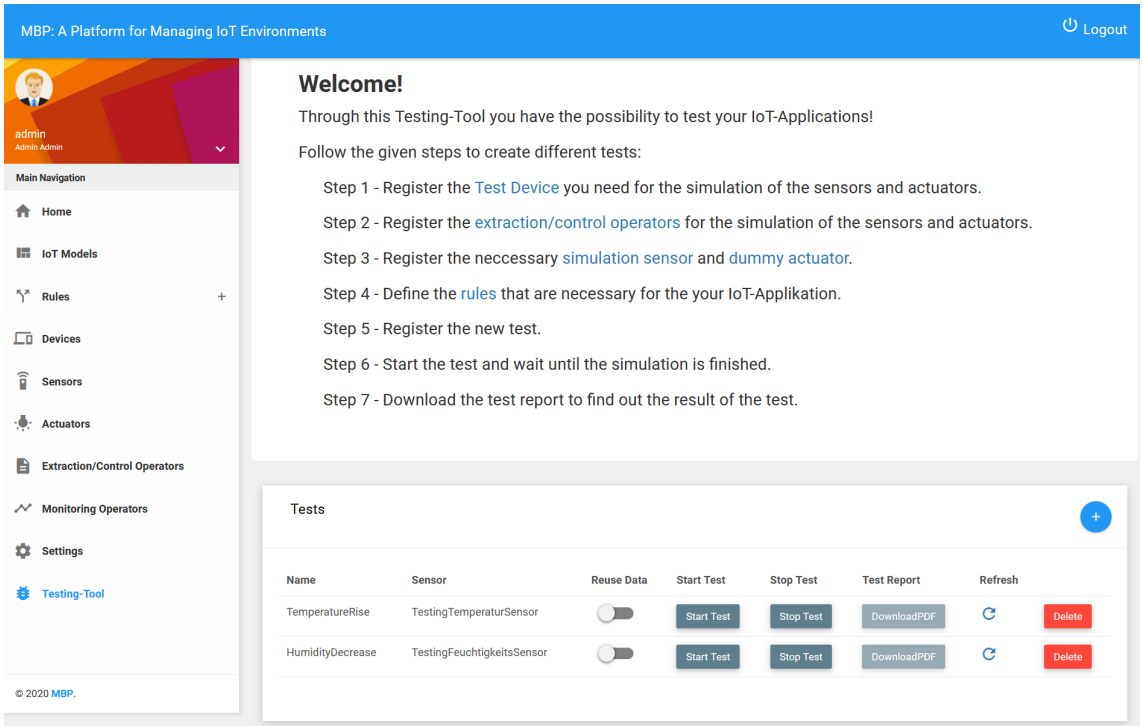


Abbildung 4.6.: Benutzeroberfläche Testwerkzeug

In Abbildung 4.7 wird das Anlegen des Tests im Zuge des Beispiel-Testfalls dargestellt. Dabei wurden alle zum Testfall notwendigen Informationen definiert, wobei festgelegt wurde, dass durch die Simulation in diesem Test lediglich die Regel *Temp<20°C* verletzt und der Aktuator-Simulator informiert werden soll. Die IoT-Applikation würde somit trotz Ausreißer richtig funktionieren, wenn bei einem Temperaturabfall lediglich diese Regel verletzt wird.

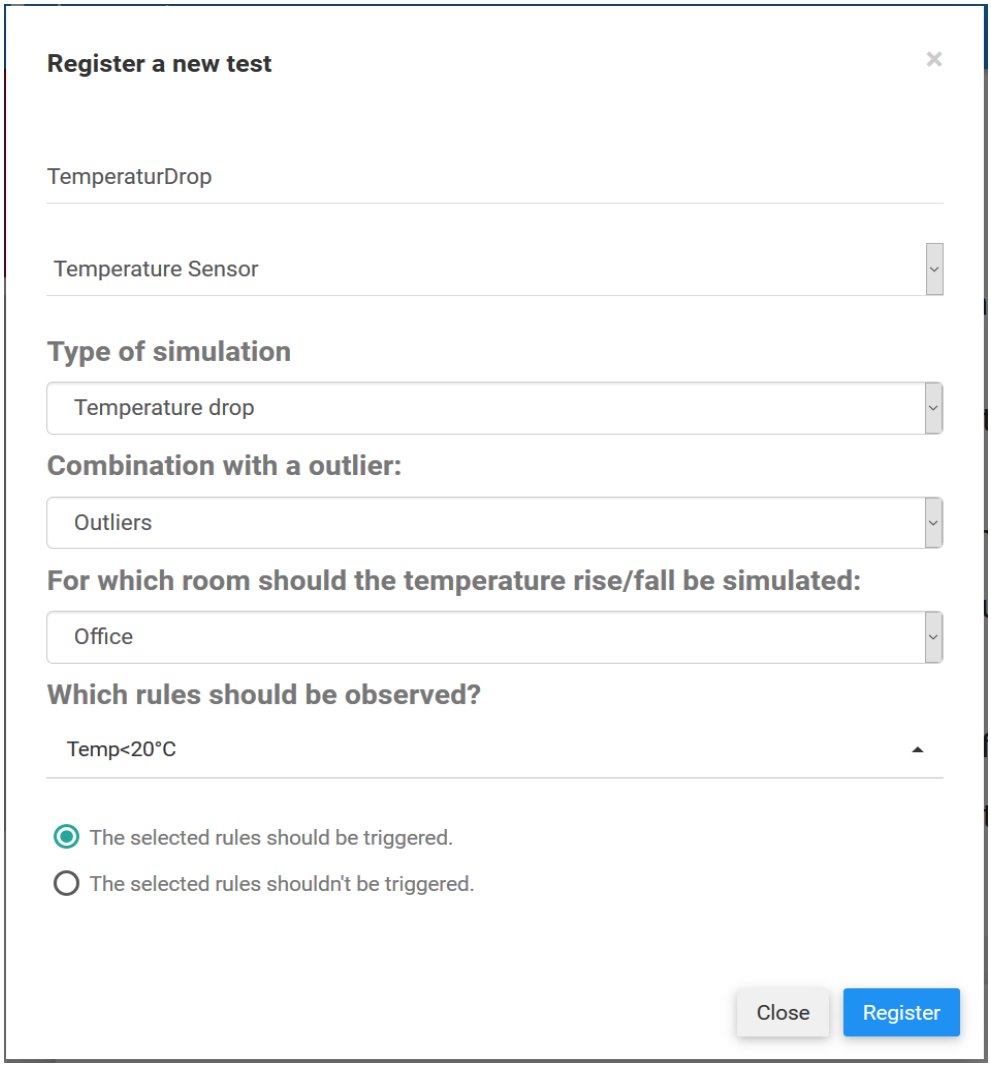


Abbildung 4.7.: Anlegen des Beispiel-Tests

Nach der Registrierung startet der Nutzer diesen Test, um die IoT-Applikation, welche die optimale Raumtemperatur innerhalb des Büros gewährleisten soll, auf Stabilität und Korrektheit zu prüfen.

4.3.6. Testbericht einsehen

Nach Ablauf oder Stoppen des Tests besteht für den Nutzer die Möglichkeit, den im Hintergrund erstellten Testbericht einzusehen. Durch diesen erhält er Auskunft darüber, ob die angelegte IoT-Applikation im gewählten Anwendungsfall wie erwartet reagiert. Um den Erfolg eines Tests bewerten zu können, wird der Vergleich von Soll- und Ist-Resultaten herangezogen. Stimmen diese miteinander überein, so gilt der Test als erfolgreich. Im Zuge der Tests von IoT-Applikationen werden Soll-Resultate durch Regeln, welche durch die Sensor-Simulation verletzt oder nicht verletzt werden

sollen, festgelegt. Die zu vergleichenden Ist-Resultate beziehen sich auf die tatsächliche Verletzung der Regeln der IoT-Applikation durch die Sensor-Simulation. Zum besseren Verständnis des Nutzers über das Ergebnis des Tests werden zusätzliche Detail-Informationen geliefert.

Der Testbericht wird als PDF-Datei bereitgestellt und kann über die Benutzeroberfläche geöffnet und gespeichert werden. Auf der ersten Seite des Reports erhält der Nutzer zum einen allgemeine Informationen über den Erfolg und die Dauer des Tests und zum anderen eine visuelle Darstellung der vom Sensor simulierten Daten. Weitere Detail-Informationen liefert die zweite Seite des Reports. Dabei werden Informationen über die vom Nutzer gewählte Simulationsart sowie Informationen über die Regeln der IoT-Applikation geliefert. Unter anderem werden die Werte aufgeführt, welche für die Regelverletzungen durch die Simulation verantwortlich waren. Wurde ein Test als nicht erfolgreich definiert, besteht hier die Möglichkeit genau zu analysieren welche Werte dafür ausschlaggebend waren. Durch das Diagramm und die weiteren Detail-Informationen kann das Ergebnis durch den Nutzer besser nachvollzogen werden. Nach der Analyse des Testberichts durch den Nutzer kann dieser eine Entscheidung über das weitere Vorgehen treffen, um die gewünschte Funktionalität der IoT-Applikation trotz Ausnahmefällen zu gewährleisten.

Anhang A.1 zeigt den zum Beispiel-Testfall gehörenden Testbericht, welcher nach Ablauf des Tests durch das Testwerkzeug bereitgestellt wurde. Aus diesem kann herausgelesen werden, dass der Test nicht erfolgreich war, was bedeutet, dass die definierte IoT-Applikation im Falle eines Ausreißer-Wertes innerhalb der Sensorwerte nicht wie gewünscht arbeitet. Der Graph zeigt an, dass, wie beim Anlegen des Tests definiert, ein Temperaturabfall mit einem Ausreißer-Wert als Anomalie vom Simulator generiert wurde. Das definierte Soll-Resultat besagt, dass durch diesen Testfall lediglich die Regel *Temp<20°C* verletzt werden soll. Durch den Graph kann bereits vermutet werden, dass die ebenfalls zur IoT-Applikation gehörende Regel *Temp>23°C* durch den Ausreißer-Wert verletzt wurde. Dieser Vermutung kann auf der zweiten Seite nachgegangen werden. Unter den Informationen der Regeln ist zu sehen, dass beide zur IoT-Applikation gehörenden Regeln *Temp<20°C* und *Temp>23°C* verletzt wurden, statt wie vom Nutzer gewünscht nur die Regel *Temp<20°C*. Grund für die Verletzung der Regel *Temp>23°C* stellt der Ausreißer-Wert *26.27°C* dar. Durch diese Informationen kann der Nutzer reagieren und beispielsweise die zuvor einfach gehaltenen Regeln, welche sich lediglich auf einen Wert beziehen, anpassen. Statt einzelner Werte wäre es sinnvoll, diese über einen gewissen Zeitabschnitt zu beobachten und erst zu reagieren, wenn deren Durchschnitt oberhalb oder unterhalb der Grenzen der optimalen Raumtemperatur liegen würde. Durch solch eine Regel wird die Sensibilität für Ausreißer-Werte stark verringert, wobei eine höhere Stabilität und Korrektheit der IoT-Applikation gewährleistet werden könnte.

5. Fazit

In diesem Kapitel wird ein Fazit über die Erfüllung der in dieser Bachelorarbeit definierten Ziele mit Hilfe einer kurzen Zusammenfassung gezogen. Im Anschluss werden limitierende Punkte dieser Arbeit geschildert. Schließlich wird ein Ausblick darüber gegeben, wie sich die Ergebnisse dieser Arbeit für die weitere Forschung eignen und an welchen Punkten weitergearbeitet werden kann.

5.1. Zusammenfassung

Das Ziel dieser Bachelorarbeit bestand darin, ein Testwerkzeug für IoT-Applikationen innerhalb der IoT-Plattform MBP bereitzustellen, welches das Testen der Anforderungen Stabilität und korrekter Funktionsweise dieser ermöglicht. Um einen solchen Test effektiv und geplant nach Anwendungsfall ausführen zu können, entstand das Nebenziel der Realisierung von Simulatoren für die Komponenten Sensor und Aktuator einer IoT-Applikation.

Diesbezüglich wurden im ersten Schritt Recherchen innerhalb der Literatur angestellt, um bereits bestehende Lösungen zur Erreichung der beiden Ziele auszumachen. Im Zuge dessen konnten jedoch keinerlei Quellen gefunden werden, welche passende oder im moderaten Maß anpassbare Simulatoren für die Simulation verschiedener Anwendungsfälle bereitstellen. Gleichermaßen konnte durch die Recherche in Bezug auf Testwerkzeuge für IoT-Applikationen kein für diese Arbeit passendes Tool gefunden werden. In diesen Quellen wurden lediglich theoretische Ansätze beschrieben und die Wichtigkeit dieser Tests verdeutlicht. Aufgrund dieser Ergebnisse fiel die Entscheidung auf eine selbstständige Entwicklung der Simulatoren und des Testwerkzeuges, wobei die Quellen als Wissensgrundlage und Motivation genutzt werden konnten.

Um ein Testwerkzeug und Simulatoren entwickeln zu können, welche in die bestehende IoT-Plattform MBP integriert werden sollten, wurde im nächsten Schritt der Aufbau und die Architektur dieser Plattform beleuchtet. Auf dieser Grundlage konnten im Anschluss Konzepte erstellt werden, welche im Verlauf der Arbeit umgesetzt wurden.

Für die Simulation von Sensoren und Aktuatoren wurden aufgrund der Vielzahl an Arten Beschränkungen getroffen. Daher wurden innerhalb dieser Arbeit Simulatoren für die eindimensionalen Temperatur- und Feuchtigkeitssensoren und zum anderen die dreidimensionalen GPS- und Beschleunigungssensoren entwickelt. Im Falle der Aktuator-Simulation wurde ein Aktuator festgelegt, welcher zwar kommunikationsfähig ist, jedoch keine Aktion ausführt. Durch das Konzept der Simulatoren wurde definiert, dass eine VM als ein IoT-Gerät fungieren soll, auf welchem Operatoren der Simulatoren abgelegt werden können. Diese Operatoren stellen die Simulatoren als einen MQTT-Client dar, durch welchen die Kommunikationsfähigkeit innerhalb der IoT-Applikationen gewährleistet wird. Mit Hilfe von Parametern, welche über die Benutzeroberfläche der MBP an die Operatoren übermittelt werden können, kann der jeweilige Anwendungsfall, welcher simuliert

werden soll, definiert werden. Diese sind jeweils mit verschiedenen Anomalien, welche durch Sensoren beispielsweise auftreten können, kombinierbar. Um einen Test unter denselben Bedingungen wiederholt ausführen zu können, werden die zuletzt simulierten Daten auf der VM abgespeichert und im Zuge einer Wiederholung von dort übernommen.

Durch die Konzeption des Testwerkzeugs werden dessen Funktionsweise innerhalb der Benutzeroberfläche definiert und die im Hintergrund notwendigen Schritte zur Durchführung des Tests beschrieben. Für das bessere Verständnis über die Funktionsweise und das Vorgehen zur Ausführung eines Tests wurde eine Testmethodik aufgestellt, welche in den einzelnen Schritten beschrieben und anhand eines fortlaufenden Beispiels zusätzlich verdeutlicht wird. An erster Stelle dieser Testmethodik steht der Nutzer, welcher zunächst einen Testfall definiert und im Anschluss in Interaktion mit der MBP und dem darin integrierten, durch diese Arbeit entwickelten Testwerkzeug steht. Bevor ein Test angelegt und ausgeführt werden kann, muss zunächst die IoT-Applikation angelegt werden, welche gegen die definierten Anforderungen geprüft werden soll. Dazu gehört das Anlegen des zum Anwendungsfall benötigten Sensor- und Aktuator-Simulator. Um diese Komponenten miteinander zu einer IoT-Applikation verknüpfen zu können, werden benutzerdefinierte Regeln angelegt, welche beschreiben, wann und durch welche Sensorwerte eine Aktuator-Aktion ausgeführt werden soll.

Wurden diese Schritte getätigt, kann innerhalb der MBP auf die Benutzeroberfläche des Testwerkzeugs gewechselt werden. Innerhalb dieser können Tests verwaltet werden, was das Anlegen und Löschen, Starten und Stoppen, die wiederholte Ausführung eines Tests sowie die Einsicht in die jeweiligen Testberichte beinhaltet. Wird ein Test angelegt, so erfolgt zum einen eine Konfiguration darüber, welcher Anwendungsfall mit welchem Sensor simuliert werden soll. Zum anderen wird festgelegt, welche Regeln der IoT-Applikation durch diese Simulation erwartet, verletzt beziehungsweise nicht verletzt werden sollen. Bei Start des Tests werden der Sensor- und Aktuator-Simulator deployed und gestartet. Der Sensor-Simulator sendet über den MBP-Core mittels MQTT die zum Anwendungsfall passenden Werte an die MBP, wo mit Hilfe des CEP auf Verletzungen der definierten Regeln der Aktuator-Simulator in Kenntnis gesetzt wird. Im Hintergrund werden alle zur IoT-Applikation gehörenden Regeln beobachtet und Informationen gesammelt, um einen für den Nutzer aufschlussreichen Testbericht zu erstellen. Dieser wird nach Beendigung oder Stoppen des Tests generiert und für den Nutzer bereitgestellt. Der Testreport liefert dem Nutzer Informationen darüber, ob die IoT-Applikation bei einem definierten Anwendungsfall wie gewünscht funktioniert, sowie weitere Detail-Informationen zur besseren Nachvollziehbarkeit des Ergebnisses. Dabei wird der Test als erfolgreich definiert, wenn lediglich die vom Benutzer innerhalb des Tests definierten Regeln der IoT-Applikationen verletzt beziehungsweise nicht verletzt werden.

Umfassend betrachtet konnten durch die Entwicklung der Simulatoren und des Testwerkzeugs für IoT-Applikationen die definierten Ziele vollständig erreicht werden. Dem Nutzer wird die Möglichkeit zuteil, effektive Tests mit zum Testfall passenden Sensorwerten durchzuführen. Dadurch kann erkannt werden, ob die definierte IoT-Applikation auch in Ausnahmefällen, ausgelöst durch Anomalien der Sensorwerte, stabil und korrekt arbeitet. Darüber hinaus ist der Nutzer mit Hilfe des Testberichts dazu in der Lage, zu erkennen, ob und weshalb ein Fehlverhalten der Applikationen aufgetreten ist, wobei infolgedessen Maßnahmen wie die Anpassung der Regelungen der Applikationen vorgenommen werden können. Somit können stabile und korrekt arbeitende Applikationen bereitgestellt werden.

5.2. Limitationen

Aufgrund der zeitlichen Beschränkung einer Bachelorarbeit und der Vielzahl an unterschiedlichen Sensoren mussten im Zuge der Entwicklung der Simulatoren Einschränkungen getroffen werden. Als eine Einschränkung wurden lediglich zwei ein- und zwei dreidimensionale Simulatoren von Sensoren, welche jeweils Werte des Typs Floating-Point generieren, entwickelt.

Mit Hilfe der Simulatoren werden jeweils zwei Ereignisse generiert, um Anwendungsfälle innerhalb eines Smart-Homes abdecken zu können. Dabei wird, wie in Abschnitt 4.3.2 beschrieben, im Falle des GPS-Sensors lediglich das Zurücklegen eines geraden Weges des Hausbewohners zu Fuß simuliert. Hierbei werden Simulationen eines nicht geraden Weges sowie das schnellere oder langsamere Zurücklegen dieses Weges außer Acht gelassen. Des Weiteren wird durch den Simulator des Beschleunigungssensors lediglich eine konstante Beschleunigung im Falle einer Bewegung simuliert. Da das reelle Pendant nicht nur dazu in der Lage ist, konstante Beschleunigung wahrzunehmen, kann hier eine weitere Limitation festgestellt werden.

Innerhalb von Sensorwerten unterschiedlicher Sensoren kann eine Vielzahl an verschiedenen Anomalien auftreten. Aufgrund dieser Tatsache wurde auch hier die Entwicklung der Simulatoren auf eine geringe Anzahl beschränkt.

Eine weitere Limitation innerhalb dieser Arbeit besteht darin, dass lediglich IoT-Applikationen getestet werden können, welche mit einem einzigen Sensor-Simulator ausgestattet sind. Dadurch können keine Applikationen getestet werden welche reelle Sensoren oder Netzwerke dieser enthalten.

In Folge dieser Limitationen können mit Hilfe der Simulatoren nicht auf alle Anwendungsfälle eingegangen werden. Zudem sind lediglich ein Bruchteil der existenten und definierbaren Applikationen durch das Testwerkzeug effektiv testbar.

5.3. Ausblick

Wie in Abschnitt 5.2 beschrieben, konnten aufgrund der Vielzahl an Sensoren und des zeitlich beschränkten Rahmens dieser Arbeit lediglich vier verschiedene Sensor-Simulatoren zur Verwendung innerhalb der Tests von IoT-Applikationen entwickelt werden. Um jedoch möglichst viele Pendants zu reellen, häufig eingesetzten Sensoren zu schaffen und so ein breitflächiges Testen vieler IoT-Applikationen gegen Stabilität und Korrektheit abdecken zu können, wäre es sinnvoll, weitere Sensor-Simulatoren für diesen Einsatz zu entwickeln. Zudem könnten die bereits bestehenden Simulatoren erweitert werden, um die verschiedenen Anwendungsfälle umfangreicher abdecken zu können.

Ebenfalls in Abschnitt 5.2 beschrieben, ermöglicht das in dieser Arbeit erstellte Testwerkzeug das Testen von IoT-Applikationen, in welche jeweils lediglich ein Sensor-Simulator eingebunden ist. Viele IoT-Applikationen bestehen jedoch unter anderem aus großen Netzwerken verschiedener Sensoren. Hierbei trägt jeder Wert bei der Entscheidungsfindung bei, ob eine Aktuator-Aktion ausgelöst werden soll oder nicht. An diese Arbeit anknüpfend könnten zukünftige Arbeiten die Möglichkeit bieten, IoT-Applikationen mit Sensornetzwerken durch eine Erweiterung des entstandenen Testwerkzeuges ebenfalls gegenüber den definierten Anforderungen zu testen.

Nachdem die IoT-Applikationen mit Hilfe von Simulatoren auf verschiedene Anwendungsfälle durch das Testwerkzeug getestet und Anpassungen beispielsweise innerhalb der Regelungen vorgenommen wurden, wäre es sinnvoll die Applikation ebenso mit dem reellen Pendant des Sensors testen zu können. Dabei könnten Tests über einen längeren Zeitrahmen durchgeführt werden, um die Möglichkeit zu erhalten, verschiedene Anwendungsfälle der Applikation betrachten zu können. Da derzeit lediglich IoT-Applikationen mit Sensor-Simulatoren testbar sind, wäre diese Erweiterung der Entwicklung ein weitere Aufgabe für zukünftige Arbeiten.

Literaturverzeichnis

[1] F. Xia, L. T. Yang, L. Wang, A. Vinel, „Internet of things," *International journal of communication systems*, Jg. 25, Nr. 9, S. 1101, 2012 (zitiert auf S. 11, 15).

[2] P. Suresh, J. V. Daniel, V. Parthasarathy, R. Aswathy, „A state of the art review on the Internet of Things (IoT) history, technology and fields of deployment," in *2014 International conference on science engineering and management research (ICSEMR)*, IEEE, 2014, S. 1–8 (zitiert auf S. 11).

[3] S. A. Ahmed, N. F. Alwan, A. M. Ali, „Overview for Internet of Things: Basics, Components and Applications," *Journal of university of Anbar for Pure science*, Jg. 12, Nr. 3, S. 47–58, 2018 (zitiert auf S. 11, 15).

[4] K. E. Skouby, P. Lynggaard, „Smart home and smart city solutions enabled by 5G, IoT, AAI and CoT services," in *2014 International Conference on Contemporary Computing and Informatics (IC3I)*, IEEE, 2014, S. 874–878 (zitiert auf S. 11).

[5] A. C. Franco da Silva, P. Hirmer, J. Schneider, S. Ulusal, M. Tavares Frigo, „MBP – Not Just an IoT Platform," in *Proceedings of the 18th Annual IEEE Intl. Conference on Pervasive Computing and Communications Demonstrations*, 2020 (zitiert auf S. 11, 16, 17, 27, 28, 40).

[6] J. Mineraud, O. Mazhelis, X. Su, S. Tarkoma, „A gap analysis of Internet-of-Things platforms," *Computer Communications*, Jg. 89, S. 5–16, 2016 (zitiert auf S. 12, 16).

[7] E. S. Reetz, D. Kuemper, K. Moessner, R. Tönjes, „How to test iot-based services before deploying them into real world," in *European Wireless 2013; 19th European Wireless Conference*, VDE, 2013, S. 1–6 (zitiert auf S. 12, 24).

[8] S. Bosmans, S. Mercelis, J. Denil, P. Hellinckx, „Testing IoT systems using a hybrid simulation based testing approach," *Computing*, Jg. 101, Nr. 7, S. 857–872, 2019 (zitiert auf S. 12, 22, 23).

[9] A. Akbar, F. Carrez, K. Moessner, A. Zoha, „Predicting complex events for pro-active IoT applications," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, IEEE, 2015, S. 327–332 (zitiert auf S. 15, 16).

[10] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann, D. Schumm, „Vino4TOSCA: A Visual Notation for Application Topologies Based on TOSCA," in *OTM 2012, Part I*, Ser. Lecture Notes in Computer Science (LNCS), Bd. 7565, Springer-Verlag, 2012, S. 416–424. DOI: 10.1007/978-3-642-33606-5_25. Adresse: http://dx.doi.org/10.1007/978-3-642-33606-5_25 (zitiert auf S. 15).

[11] T. Itu, „Series y: global information infrastructure, internet protocol aspects and nextgeneration networks," *Rec. ITU-T Y*, Jg. 2720, 2009 (zitiert auf S. 15).

[12] E. Fleisch, F. Thiesse, *Internet der Dinge*, Hrsg.: Lehrstuhl für Wirtschaftsinformatik, Universität Potsdam, https://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/technologien-methoden/Rechnernetz/Internet/Internet-der-Dinge, Sep. 2014 (zitiert auf S. 15, 16).

[13] D. Miorandi, S. Sicari, F. De Pellegrini, I. Chlamtac, „Internet of things: Vision, applications and research challenges," *Ad hoc networks*, Jg. 10, Nr. 7, S. 1497–1516, 2012 (zitiert auf S. 15).

[14] G. Fortino, P. Trunfio, *Internet of things based on smart objects: Technology, middleware and applications*. Springer, 2014 (zitiert auf S. 16).

[15] M. Eckert, F. Bry, „Aktuelles schlagwort"complex event processing (cep)"," *Informatik-Spektrum*, Nr. 2, S. 163–167, 2009 (zitiert auf S. 16, 28).

[16] U. Hunkeler, H. L. Truong, A. Stanford-Clark, „MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks," in *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE'08)*, IEEE, 2008, S. 791–798 (zitiert auf S. 16, 17).

[17] N. Naik, „Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," in *2017 IEEE international systems engineering symposium (ISSE)*, IEEE, 2017, S. 1–7 (zitiert auf S. 16).

[18] Y. Xu, M. Veeramani, S. Radhakrishnan, „Towards SDN-based fog computing: MQTT broker virtualization for effective and reliable delivery," *2016 8th International Conference on Communication Systems and Networks (COMSNETS)*, S. 1–6, 2016 (zitiert auf S. 16, 17).

[19] P. Hirmer, U. Breitenbücher, A. C. F. da Silva, K. Képes, B. Mitschang, M. Wieland, „Automating the Provisioning and Configuration of Devices in the Internet of Things.," *CSIMQ*, Jg. 9, S. 28–43, 2016 (zitiert auf S. 17, 27).

[20] Q. Yang, J. J. Li, D. M. Weiss, „A survey of coverage-based testing tools," *The Computer Journal*, Jg. 52, Nr. 5, S. 589–597, 2009 (zitiert auf S. 18).

[21] J. Ludewig, H. Lichter, *Software Engineering: Grundlagen, Menschen, Prozesse, Techniken*. dpunkt. verlag, 2013 (zitiert auf S. 18, 19, 33).

[22] M. Kaur, R. Kumari, „Comparative study of automated testing tools: Testcomplete and quicktest pro," *International Journal of Computer Applications*, Jg. 24, Nr. 1, S. 1–7, 2011 (zitiert auf S. 19).

[23] P. Giménez, B. Molína, C. E. Palau, M. Esteve, „SWE Simulation and Testing for the IoT," in *2013 IEEE International Conference on Systems, Man, and Cybernetics*, IEEE, 2013, S. 356–361 (zitiert auf S. 21).

[24] T. Pflanzner, A. Kertész, B. Spinnewyn, S. Latré, „MobIoTSim: Towards a mobile IoT device simulator," in *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, IEEE, 2016, S. 21–27 (zitiert auf S. 21).

[25] *Simple Sensor Simulator*, https://atomiton.atlassian.net/wiki/spaces/TQLDocs/pages/15728650/Simple+Sensor+Simulator, Accessed: 2020-03-10 (zitiert auf S. 22).

[26] S. N. Han, G. M. Lee, N. Crespi, N. Van Luong, K. Heo, M. Brut, P. Gatellier, „Dpwsim: A devices profile for web services (DPWS) simulator," *IEEE internet of things journal*, Jg. 2, Nr. 3, S. 221–229, 2015 (zitiert auf S. 22).

[27] *DPWSim*, https://github.com/sonhan/dpwsim, Accessed: 2020-03-10 (zitiert auf S. 22).

[28] J. Fernandes, M. Nati, N. Loumis, S. Nikoletseas, T. P. Raptis, S. Krco, A. Rankov, S. Jokic, C. M. Angelopoulos, S. Ziegler, „IoT Lab: Towards co-design and IoT solution testing using the crowd," in *2015 International Conference on Recent Advances in Internet of Things (RIoT)*, IEEE, 2015, S. 1–6 (zitiert auf S. 22).

[29] S. Popereshnyak, O. Suprun, O. Suprun, T. Wieckowski, „IoT application testing features based on the modelling network," in *2018 XIV-th International Conference on Perspective Technologies and Methods in MEMS Design (MEMSTECH)*, IEEE, 2018, S. 127–131 (zitiert auf S. 23).

[30] H. Kim, A. Ahmad, J. Hwang, H. Baqa, F. Le Gall, M. A. R. Ortega, J. Song, „IoT-TaaS: Towards a prospective IoT testing framework," *IEEE Access*, Jg. 6, S. 15 480–15 493, 2018 (zitiert auf S. 23).

[31] *Multi-purpose Binding and Provisioning Platform (MBP)*, https://github.com/IPVS-AS/MBP, Accessed: 2020-03-29 (zitiert auf S. 27).

[32] *Wegpunktprojektion*, https://www.cachewiki.de/wiki/Wegpunktprojektion, Accessed: 2020-03-30 (zitiert auf S. 37).

[33] *OC10191 Käsching-Grundlagen - Peilung / WegpunktProjektion - Geocaching mit Opencaching*. Adresse: https://www.opencaching.de/viewcache.php?cacheid=169026 (zitiert auf S. 37).

[34] M. von Beschleunigungssensoren, W. Beschleunigungssensoren, „Beschleunigungs sensoren," *Sensoren im Kraftfahrzeug*, S. 67, (zitiert auf S. 38).

[35] J. Baker, „Newtons Bewegungsgesetze," in *50 Schlüsselideen Physik*, Springer, 2009, S. 8–11 (zitiert auf S. 39).

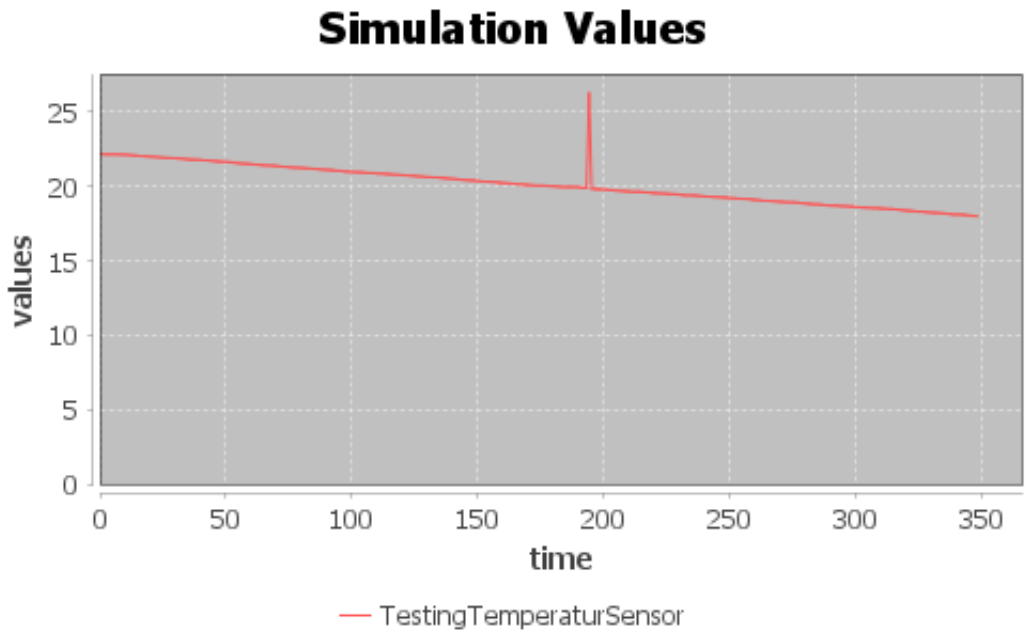
[36] D. Teile, V. des Meters, „Einheiten," *Handbuch Maschinenbau: Grundlagen und Anwendungen der Maschinenbau-Technik*, S. 5, 2010 (zitiert auf S. 39).

Alle URLs wurden zuletzt am 12. 04. 2020 geprüft.

A. Anhang

Test-Report: TemperatureDrop

Successful	No
Start-Time	23-03-2020 13:44:09
End-Time	23-03-2020 13:50:34



Test-Details:

Simulated Sensor	
Sensor-Type	Temperature Sensor
Test-Case	Temperature drop
Combination	Outliers

Simulated actuator
The actuator used for the tests does not trigger any actions if the corresponding rule is triggered. It functions as a dummy.

Rule-Informations	
The selected rules should be executed by the test	
Rules, which should be triggered	Rules, which were triggered
Temp<20°C	Temp>23°C, Temp<20°C
Temp<20°C	
Number of executions before the test	Number of executions after the test
1063	1229
Last execution before the test	Last execution after the test
23-03-2020 13:42:12	23-03-2020 13:50:31
Trigger-Values	
19.99, 19.98, 19.97, 19.97, 19.97, 19.97, 19.97, 19.97, 19.95, 19.93, 19.91, 19.9, 19.86, 19.84, 19.83, 19.83, 19.82, 19.8, 19.79, 19.77, 19.76, 19.74, 19.73, 19.72, 19.7, 19.69, 19.68, 19.67, 19.65, 19.65, 19.65, 19.65, 19.63, 19.62, 19.61, 19.59, 19.57, 19.56, 19.55, 19.54, 19.53, 19.53, 19.51, 19.5, 19.49, 19.49, 19.47, 19.45, 19.43, 19.41, 19.41, 19.41, 19.4, 19.4, 19.38, 19.37, 19.35, 19.33, 19.32, 19.32, 19.31, 19.29, 19.29, 19.28, 19.27, 19.26, 19.25, 19.23, 19.21, 19.2, 19.19, 19.18, 19.17, 19.17, 19.16, 19.14, 19.12, 19.1, 19.09, 19.07, 19.05, 19.04, 19.04, 19.03, 19.01, 19.0, 18.99, 18.98, 18.96, 18.95, 18.94, 18.94, 18.94, 18.92, 18.91, 18.89, 18.87, 18.86, 18.84, 18.83, 18.81, 18.8, 18.8, 18.78, 18.76, 18.74, 18.73, 18.72, 18.72, 18.72, 18.7, 18.69, 18.68, 18.68, 18.66, 18.66, 18.64, 18.63, 18.61, 18.59, 18.58, 18.58, 18.56, 18.56, 18.55, 18.55, 18.53, 18.52, 18.51, 18.5, 18.49, 18.49, 18.47, 18.46, 18.44, 18.42, 18.4, 18.39, 18.37, 18.36, 18.34, 18.32, 18.31, 18.31, 18.29, 18.27, 18.26, 18.25, 18.23, 18.23, 18.21, 18.21, 18.19, 18.17, 18.15, 18.13, 18.12, 18.12, 18.12, 18.1, 18.09, 18.07, 18.05, 18.03, 18.02, 18.0	
Temp>23°C	
Number of executions before the test	Number of executions after the test
286	287
Last execution before the test	Last execution after the test
23-03-2020 13:35:31	23-03-2020 13:47:57
Trigger-Values	
26.27	

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