Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Modellierung und Deployment von IoT-Umgebungen in der MBP

Amil Imeri

Studiengang:	Softwaretechnik
Prüfer/in:	PD Dr. rer. nat. habil. Holger Schwarz
Betreuer/in:	Ana Cristina Franco da Silva, M.Sc. Dr. rer. nat. Pascal Hirmer
Beginn am:	9. November 2018
Beendet am:	9. Mai 2019

Kurzfassung

Viele Alltagsgegenstände werden ein Teil des Internets, über welches sie Daten miteinander austauschen und autonom funktionieren können. Das Internet wird als *Internet der Dinge (IoT)* bezeichnet und enthält Geräte die typischerweise mit Sensoren und Aktuatoren ausgestattet sind und somit eine intelligente IoT-Umgebung schaffen. Die IoT-Umgebungen können von verschiedenen Typen sein und eine hohe Anzahl an Geräten enthalten. Die, an der Universität Stuttgart entwickelte, *Multi-purpose Binding and Provisioning Platform (MBP)*, erleichtert die Konfiguration und Anbindung von verschiedenen Typen von Geräten aus einer IoT-Umgebung. Die Plattform verwendet Adapter, die an die Geräte geschickt werden, um Sensordaten zu erhalten oder Aktuatoren zu aktivieren. Die automatisierte Anbindung der Geräte durch das Deployment der Adapter, kann bei vielen Geräten in einer IoT-Umgebung sehr aufwendig sein, da die Geräte einzeln konfiguriert und angeschlossen werden müssen. In dieser Bachelorarbeit wird dieser Prozess durch ein grafisches Modellierungswerkzeug vereinfacht, mit dem Modelle von IoT-Umgebungen erstellt und ihre Geräte in einem Schritt angeschlossen werden können. Hierzu wird eine prototypische Implementierung des Werkzeugs realisiert, die in dem vorhandenen Prototyp der MBP integriert wird. Das Ergebnis ist eine Plattform, die durch die Modellierung die physische IoT-Umgebungen widerspiegelt und anschließend ihr Deployment ermöglicht.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Abkürzungsverzeichnis

API Application Programming Interface. 21

CEP Complex Event Processing. 25

GPS Global Positioning System. 17

HTML Hypertext Markup Language. 37

HTTP Hypertext Transfer Protocol. 21

HTTPS Hypertext Transfer Protocol Secure. 40

IoT Internet of Things. 15

IP Internet Protocol. 17

JSON JavaScript Object Notation. 20

LED Light-Emitting Diode. 36

MAC Media Access Control. 28

MBP Multi-purpose Binding and Provisioning Platform. 15

MQTT Message Queuing Telemetry Transport. 21

REST Representational State Transfer. 21

SSH Secure Shell. 21

TOSCA Topology and Orchestration Specification for Cloud Applications. 24

UI User Interface. 39

XML Extensible Markup Language. 20

1 Einführung

Die Menschen sind heutzutage mit vielen verschiedenen Geräten umgeben, die allein oder zusammen bestimmte Ziele erfüllen. Das *Internet der Dinge (kurz: IoT, engl.: Internet of Things)* [FT14; VF13] steht für die Vernetzung dieser Geräte, die sich überall, jederzeit und mit allem und jedem ereignen kann. Die Geräte, die oft mit Sensoren und Aktuatoren ausgestattet sind, kommunizieren miteinander um eine *intelligente* Umgebung zu erschaffen. Diese Umgebung wird auch als IoT-Umgebung bezeichnet und kann Sensoren enthalten, die die physischen Zustände erfassen und Aktuatoren besitzen, welche die physischen Zustände verändern können. So kann beispielsweise das Licht in einem Raum durch Bewegungs- und Helligkeitssensoren oder die Klimaanlage durch Temperatursensoren kontrolliert werden. Es existieren verschiedene Arten von IoT-Umgebungen mit verschiedener Komplexität [BS11; GBMP13; VF13]. Das bekannteste Beispiel ist das *Smart Home* [AZZ+17], welches ein intelligentes Zuhause mit Sensoren und Aktuatoren repräsentiert.

Der Mensch kann mit den IoT-Geräten auf verschiedene Weise interagieren. Das ist meistens von dem Gerätetyp abhängig. In einer IoT-Umgebung wird typischerweise eine Plattform [MMST16; SK17] aufgestellt, die mit einer IoT-Applikation Geräte vernetzt, steuert und die Interaktion mit den Menschen ermöglicht. Die IoT-Plattformen haben verschiedene Aufgaben [MMST16; SK17]: Verwaltung und Speicherung von großen Datenmengen, Datenvisualisierung, Unterstützung heterogener Geräte etc. Sie zeichnen sich auch durch die Protokolle und Technologien aus, die sie für die Kommunikation verwenden.

Um die IoT-Applikationen zu nutzen, müssen zuerst die Geräte in der Umgebung eingerichtet, und für die Nutzung der Sensoren und Aktuatoren, konfiguriert und vernetzt werden. Um diese Schritte zu vereinfachen, wurde an der Universität Stuttgart die *Multi-purpose Binding and Provisioning Platform (MBP)* entwickelt [HBS+16]. Die MBP ermöglicht eine automatisierte Anbindung von verschiedenen Typen von IoT-Geräten, um an die Sensordaten zu gelangen oder die Aktuatoren zu aktivieren. Das ist vor allem vorteilhaft, wenn eine IoT-Umgebung viele Geräte enthält, die manuell konfiguriert werden müssen und somit viel Zeit in Anspruch nehmen. Das Konzept der MBP umfasst die Erstellung der digitalen Repräsentation einer physischen IoT-Umgebung, in der die IoT-Geräte gleichzeitig und mit niedrigem Aufwand angebunden werden. Diese Idee wurde noch nicht umgesetzt und liegt im Fokus dieser Bachelorarbeit.

Ziel dieser Bachelorarbeit ist es, eine grafische Modellierung in der MBP zu ermöglichen, wodurch IoT-Modelle von physischen Umgebungen erzeugt werden können. Die IoT-Modelle sollen für die Anbindung der darin enthaltenen Geräte verwendet werden. Auf diese Weise wird der Prozess der automatisierten Anbindung weiter vereinfacht.

Gliederung

In dieser Bachelorarbeit werden zuerst die Ziele definiert und nachdem der Ansatz für ihre Umsetzung vorgestellt und implementiert wurde, wird das Ergebnis evaluiert. Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 - Grundlagen: Die Grundlagen liefern das Basiswissen für diese Bachelorarbeit. Es wird ein Einblick in das Internet der Dinge vermittelt. Dabei werden einige Anwendungsbereiche vorgestellt und die IoT-Plattform definiert. Im zweiten Teil des Kapitels werden die MBP und ihr Prototyp beschrieben.

Kapitel 3 - Verwandte Arbeiten: Es werden Arbeiten vorgestellt, die im Zusammenhang mit dieser Bachelorarbeit stehen. Zuerst werden Arbeiten über die Modellierung von IoT-Umgebungen vorgestellt und anschließend Arbeiten über das Deployment betrachtet.

Kapitel 4 - Aufgabenstellung: Dieses Kapitel befasst sich mit der Aufgabenstellung und beschreibt das Hauptziel und die zusätzlichen Ziele dieser Bachelorarbeit.

Kapitel 5 - Modellierung und Deployment von IoT-Umgebungen: In diesem Kapitel wird der Ansatz mit den Konzepten und Methoden beschrieben und dabei wird auf das Datenmodell und die Modellierung komplexer Umgebungen eingegangen. Weiterhin werden die notwendigen Anpassungen in der MBP identifiziert. Der zweite Teil des Kapitels beschäftigt sich mit der Implementierung. Vor der Implementierung wird die Analyse der Technologien beschrieben, die für die Implementierung verwendet werden können. Nach der Auswahl der Technologien, wird die Implementierung beschrieben. Anschließend wird das Ergebnis im Bezug auf die Ziele evaluiert.

Kapitel 6 - Zusammenfassung und Ausblick: Das letzte Kapitel fasst die Ergebnisse der Arbeit zusammen und gibt einen Ausblick über die möglichen zukünftigen Erweiterungen und Forschungen.

2 Grundlagen

Im Folgenden werden die Grundlagen dargestellt, die für das Verständnis der nachfolgenden Kapitel notwendig sind. Einerseits wird auf das Internet der Dinge eingegangen und auf der anderen Seite wird die Struktur und die Funktionsweise der MBP verständlich gemacht, die im Mittelpunkt dieser Bachelorarbeit steht.

2.1 Internet der Dinge

Das Internet der Dinge beruht auf dem Konzept *Ubiquitous Computing*, eine Vision von Mark Weiser, in der ein Mensch mit einer Vielzahl von Computern und Geräten umgeben sein wird, die ihre Dienste in eine unaufdringliche Weise anbieten werden [Wei93]. Dabei wird die Vernetzung dieser Geräte im Allgemein als das Internet der Dinge bezeichnet. Das ist eine neue Internet-Revolution, in der die Geräte in der Lage sind miteinander zu kommunizieren, um gemeinsame Ziele zu erreichen [VF13].

Nach Fleisch und Thiesse [FT14] wird IoT als ein Funktionsbündel verschiedener Technologien betrachtet. Nach den Autoren bilden die einzelnen Funktionen und Technologien zusammen eine neue Qualität der Informationsverarbeitung, die unter anderem folgende Eigenschaften der IoT-Geräte voraussetzt:

- *Kommunikation:* Die Möglichkeit zur Vernetzung, um Daten auszutauschen oder Dienste zu nutzen. Die IoT-Geräte können sich über WiFi oder Bluetooth und durch Nutzung standardisierter Protokolle wie z.B. das Internet Protokoll (IP), mit Ressourcen oder untereinander vernetzen.
- *Identifikation und Lokalisierung:* In einer bestimmten Umgebung sind alle IoT-Geräte eindeutig identifizierbar und können durch verschiedene Techniken, wie GPS oder Ultraschall, lokalisiert werden.
- *Speicher:* Die IoT-Geräte sind in der Lage eine bestimmte Menge von Informationen zu speichern und zu verarbeiten.
- *Sensorik:* Die Möglichkeit, verschiedene Informationen über die Umgebung zu sammeln (Temperatur, Helligkeit, Bewegungen etc.).
- *Aktuatorik:* Die Möglichkeit, eine Einwirkung in der Umgebung zu erzielen (Heizung einstellen, Licht einschalten etc.).

- *Benutzerschnittstelle:* Die Interaktion zwischen dem Nutzer und dem IoT-Gerät wird durch verschiedene Schnittstellen realisiert, die sich eventuell von klassischen Rechnerschnittstellen unterscheiden. Es besteht auch die Möglichkeit, über eine zentrale Plattform, alle IoT-Geräte gleichzeitig zu kontrollieren.

Eine Menge von IoT-Geräten mit den aufgeführten Eigenschaften erschaffen eine IoT-Umgebung, die in vielen verschiedenen Bereichen des alltäglichen Lebens wiedergefunden werden kann. Die Anwendung von IoT wird im folgenden Abschnitt beschrieben.

2.1.1 Anwendungsbereiche

Als eine aufstrebende Technologie, findet das IoT in immer mehr Bereichen eine Anwendung. Das kann ein Zimmer in einem Haus, ein Sektor in einer Fabrik oder eine ganze Stadt sein. Dabei kommt IoT in den Bereichen der Gesundheit, des Transports oder der Sicherheit zum Einsatz [BS11; GBMP13; VF13]. In diesem Abschnitt wird kurz auf die IoT-Umgebungen *Smart Home* und *Smart Factory* eingegangen.

Smart Home wird von Satpathy [SU06] als ein intelligentes Zuhause definiert, welches es den Bewohnern ermöglicht, mit Hilfe der Technologie, unabhängig und komfortabel zu leben. Dabei enthält ein solches Zuhause mechanische und digitale Geräte, die ein Netzwerk bilden, miteinander und mit dem Nutzer kommunizieren können und somit einen interaktiven Raum erschaffen. So kann ein Bewegungssensor an der Tür dazu führen, dass das Licht durch einen Aktuator eingeschaltet wird oder dass ein Lichtsensor die Jalousien hoch oder runter fährt. Der Nutzer kann darüber hinaus die Sensordaten lesen oder die Aktuatoren manuell steuern. Smart Home in Kombination mit dem IoT hat ein sehr breites Forschungsspektrum [AZZ+17].

Die vierte industrielle Revolution, auch bekannt unter dem Namen *Industrie 4.0* [KWH13; WSCL13], sieht die Nutzung des IoT in industriellen Prozessen vor. Dabei werden Smart Factorys erschaffen, die mit Sensoren und Aktuatoren ausgestattet sind. Die Sensoren können beispielweise den Zustand eines Produkts oder einer Maschine wiedergeben und die Aktuatoren eine Maschine ein- oder ausschalten. Eine Smart Factory ist in der Lage hohe Komplexität zu beherrschen um Waren effizienter herzustellen, ist zudem weniger anfällig für Störungen und ermöglicht eine einfache Kommunikation zwischen Menschen, Maschinen und Ressourcen [KWH13].

Alle diese IoT-Umgebungen haben oft eine Verwaltungskomponente, in der die Geräte organisiert, kontrolliert und überwacht werden und in der eine Benutzerschnittstelle bereitgestellt wird. Diese wird im folgenden Abschnitt detailliert beschrieben.

2.1.2 IoT-Plattform

Eine *IoT-Plattform* verbindet die IoT-Geräte mit den Nutzern und bezeichnet im Allgemein eine Middleware mit Hardware- und Softwarekomponenten [MMST16; SK17]. Abbildung 2.1 zeigt eine Cloud-basierte und eine lokale Plattform, bei denen die Beziehungen mit den Nutzern und IoT-Geräten dargestellt werden. Mineraud et al. [MMST16] haben eine Analyse der vorhandenen IoT-Plattformen durchgeführt, wobei sich die folgende Eigenschaften einer IoT-Plattform hervorheben:

- *Unterstützung heterogener Geräte:* Die IoT-Geräte unterscheiden sich oft bezüglich der Kommunikationsprotokolle die sie verwenden. Aus diesem Grund ist es nicht möglich alle Geräte direkt mit der Plattform zu verbinden. Um das Problem zu lösen kann ein *Gateway* als eine Zwischenkomponente eingesetzt werden, die eine Verbindung ermöglicht.
- *Typ und Architektur:* Eine IoT-Plattform kann über einer Cloud oder über einen lokalen Server bereitgestellt werden. Bei einer Cloud können beispielsweise Platform-as-a-Service (PaaS) oder Software-as-a-Service (SaaS) verwendet werden. Es gibt auch IoT-Geräte, die ihr eigenes Netzwerk bilden und somit individuell kontrolliert werden können.
- *Datenverwaltung:* Eine IoT-Umgebung kann eine sehr hohe Anzahl an IoT-Geräten mit Sensoren und Aktuatoren enthalten, die ihre Daten zum gleichen Zeitpunkt senden. Die wichtigsten Aspekten sind die Speicherung dieser großen Mengen an Daten, ihre effiziente Verarbeitung, ihre Übertragung und die Kontrolle des Datenzugriffes (Informationssicherheit).

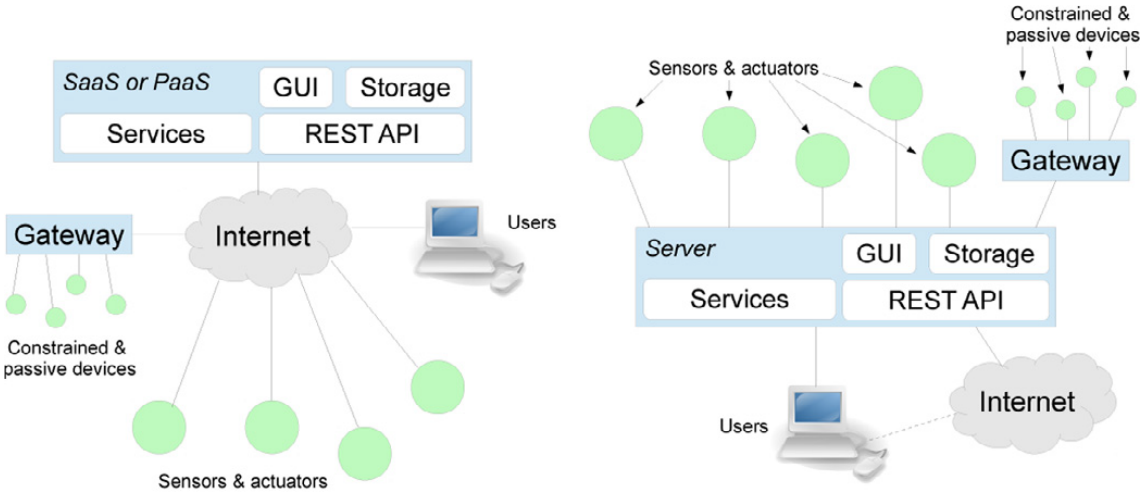

Abbildung 2.1: IoT-Plattformen [MMST16]

Anschließend wird die MBP, als ein Vertreter der IoT-Plattformen, vorgestellt und ihre Funktionsweise beschrieben.

2.2 Multi-Purpose Binding and Provisioning Platform

Die Multi-Purpose Binding and Provisioning Platform (MBP) wurde entwickelt, um die Anbindung der Geräte in der IoT zu automatisieren und deren Überwachung zu ermöglichen [HBS+16]. Die physische IoT-Umgebung hat oft eine digitale Repräsentation [BR16], die ständig in Synchronisation bleiben soll. Die IoT-Geräte ändern oft ihre Position, werden entfernt oder eingefügt und müssen im Normalfall manuell mit der digitalen Umgebung gebunden und registriert werden. Dieser Prozess kann bei vielen heterogenen Geräten komplex, zeitintensiv und teuer sein. Die MBP vereinfacht den Prozess, indem die Registrierung und Anbindung der IoT-Geräte automatisiert wird.

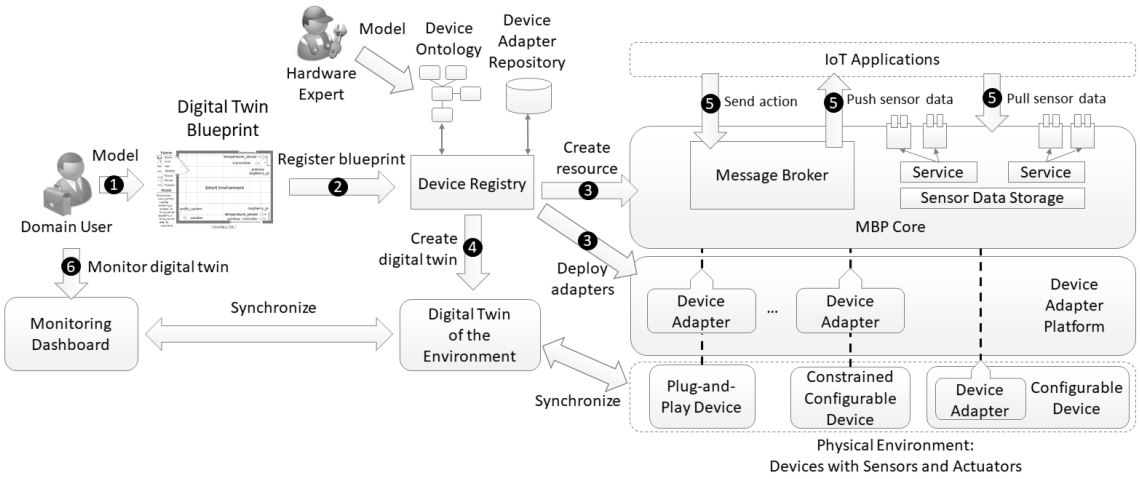

Abbildung 2.2: MBP Systemarchitektur (basiert auf [HBS+16])

Die Architektur der MBP und die Schritte der Verarbeitung sind in Abbildung 2.2 zu sehen. Die Funktionsweise des Systems ist in sechs Schritten aufgeteilt [HBS+16]:

1. **Modellierung der IoT-Umgebung:** Der Nutzer kennt die physische Umgebung und ist in der Lage einen Entwurf zu erstellen. Das dabei verwendete Datenmodell kann auf eine Ontologie basieren oder die Standardformate XML oder JSON verwenden. Der Nutzer kann bei der Modellierung das Layout und die Größe beliebig ändern und die IoT-Geräte durch weitere Parameter spezifizieren. Dieser Schritt wurde in dem aktuellen Prototyp der MBP noch nicht realisiert. Daher liegt hierauf, zusammen mit der Modellregistrierung und anschließendem Deployment, der Schwerpunkt dieser Arbeit.

2. **Modellregistrierung:** Die in dem Modell enthaltende Geräte, Aktuatoren und Sensoren werden registriert. Ihre zusätzliche technische Informationen die für die Anbindung notwendig sind, werden von *Device Ontology* geholt. Dabei wird, aufgrund der Bearbeitbarkeit und Erweiterbarkeit, ein Ontologiemodell mit folgenden Elementen verwendet: physisches Objekt, Gerät, Sensor, Aktuator und Adapter.

3. **Automatisiertes Deployment der Adapter und Anbindung der Geräte:** Ein Adapter ist ein Stück Code, der es möglich macht, eine Verbindung zu dem Gerät aufzubauen, Aktuatoren zu aktivieren, Sensordaten zu extrahieren und in *MBP Core* bereitzustellen (durch verschiedene Protokolle, wie z.B. HTTP oder MQTT[1]). Automatisierte Anbindung bedeutet, den Zugriff der IoT-Applikationen auf die Geräte, durch die Anwendung von Adaptern, zu ermöglichen. Die Adapter werden von *Device Adapter Repository* geholt. Automatisiertes Deployment hängt vom Gerätetyp ab und kann entweder direkt auf dem Gerät oder auf der *Device Adapter Platform* ausgeführt werden, die hier die Rolle eines Gateways übernimmt. Dabei wird SSH oder ein anderer Ansatz für das Deployment angewendet.
4. **Erstellung der digitalen Umgebung:** Nach der Registrierung und Anbindung der Geräte, Sensoren und Aktuatoren, wird die digitale Repräsentation von der physischen Umgebung erstellt. Dabei soll die Synchronisation mit der physischen Umgebung jederzeit gewährleistet sein.
5. **Zugriff der IoT-Applikationen auf die Geräte:** Auf die Sensordaten kann über die *MBP Core* auf zwei Arten zugegriffen werden: über eine Anfrage (pull-basiert) oder über ein Publish-/Subscribe-Prinzip (push-basiert). Der Zugriff auf die Aktuatoren wird ebenfalls über die *MBP Core* durchgeführt. Die Adapter bieten dazu bestimmte Funktionen die von *MBP Core* aufgerufen werden können.
6. **Überwachung der IoT-Umgebung:** Die digitale Umgebung wird auf einem Dashboard graphisch dargestellt. So kann der Nutzer die Geräte überwachen und ein visuelles Feedback über deren Zustand bekommen.

Der Prototyp der MBP ist als eine Open-Source Webanwendung implementiert und wird in den folgenden Abschnitt kurz beschrieben.

2.2.1 Prototyp

Der Prototyp besteht aus einem Java-basierten Backend, der mit dem Spring Framework[2] implementiert ist, und einem AngularJS[3] Frontend. Die aktuelle Implementierung ermöglicht die Registrierung der Geräte, Sensoren und Aktuatoren mit ihren Parametern durch die REST API. Das Deployment der Adapter geschieht über SSH. Nach dem Deployment bekommt die Anwendung die Sensordaten von den Geräten, die gespeichert und graphisch dargestellt werden und somit die Überwachung möglich machen. Die Zustände der Geräte werden ebenfalls in der Anwendung bekannt. Für die Speicherung der Daten wird die

[1] http://mqtt.org/
[2] https://spring.io/
[3] https://angularjs.org/

NoSQL Datenbank MongoDB[4] verwendet. Die Vernetzung der Geräte und das Publish-/Subscribe-Prinzip wird durch das MQTT Protokoll und Mosquitto[5] Broker ermöglicht. In dem Prototyp ist keine Nutzerverwaltung vorhanden, es wird lediglich zwischen einem einfachen Nutzer und Experten unterschieden, der bei der Nutzung größere Rechte hat. Die Benutzerschnittstelle des Prototyps ist in Abbildung 2.3 dargestellt.

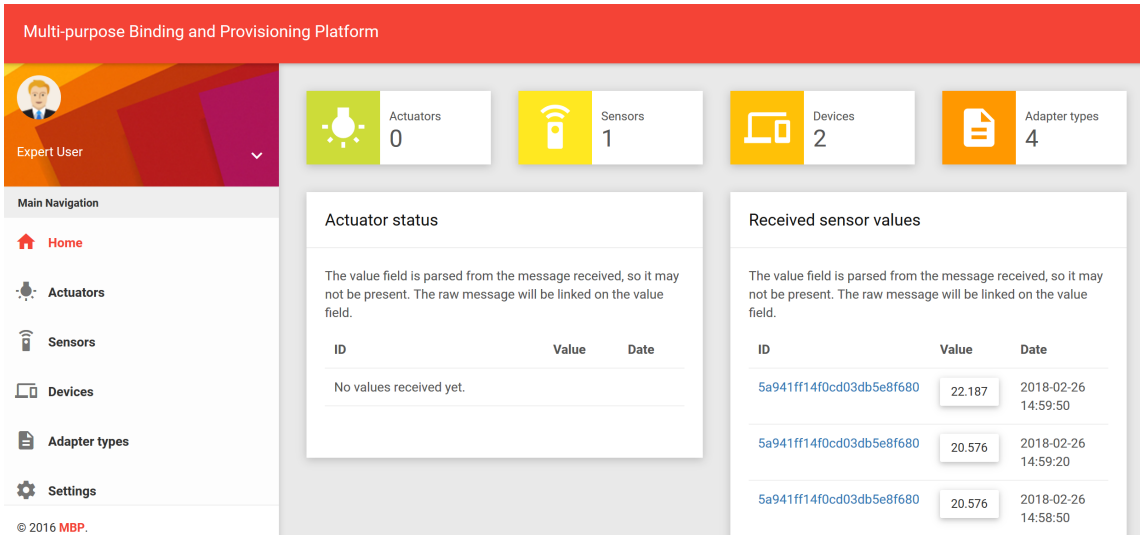

Abbildung 2.3: MBP Benutzerschnittstelle

[4] https://www.mongodb.com/
[5] https://mosquitto.org/

3 Verwandte Arbeiten

Es gibt verschiedene Ansätze, wie die Modellierung und das Deployment von IoT-Umgebungen umgesetzt werden können. Im weiteren Verlauf werden verwandte Arbeiten vorgestellt, wobei einige als Grundlage für diese Bachelorarbeit dienen.

3.1 Modellierung von IoT-Umgebungen

Nugent et al. [NFD+07] haben, als Lösung für die heterogene Darstellungen der Daten eines Smart Homes, ein XML-basiertes Schema mit dem Namen *homeML* erstellt. Das Ziel von homeML ist es, die Darstellung von Daten zu standardisieren und somit ihre Speicherung und ihren Austausch zu erleichtern. HomeML hat eine hierarchische Baumstruktur mit dem Knoten *SmartHome* als Wurzel. Die Wurzel enthält unter anderem Räume und diese wiederum Geräte und andere Parameter. Smart Home hat in der Arbeit von Nugent et al. das Ziel, Menschen bezüglich ihrer Gesundheit zu unterstützen. Daher enthält homeML, zusätzlich zur technischen Dokumentation, auch Informationen über die Einwohner und deren Gesundheitsplan. Es wurden auch einige Werkzeuge entwickelt, die die Nutzung von homeML unterstützen sollen [MNH+13]. Dieser Ansatz dient als Grundlage für die Darstellung einer physischen IoT-Umgebung die in dieser Bachelorarbeit umgesetzt wird.

Eine andere Art der Modellierung von IoT-Umgebungen wurde von Mayer et al. [MIV+14; MIVV14] vorgestellt. Deren Ansatz ermöglicht eine zielorientierte Modellierung von IoT-Umgebungen. Die IoT-Geräte sind bereits in der Umgebung bereitgestellt und der Nutzer kann über ein Modellierungswerkzeug den gewünschten Zustand der Umgebung modellieren. Mit dem Werkzeug wird der Komplexität der zugrundeliegenden Semantik ausgewichen. Die Elemente des erstellten Modells repräsentieren dabei nicht zwingend die konkreten Geräte in der Umgebung, sondern stehen für den Zustand der Umgebung. Das System stellt dann fest, ob der gewünschte Zustand durch die vorhandenen Geräte und ihre Dienste erreicht werden kann.

Der *Jigsaw Editor* [HCH+03] bietet Nutzern die Möglichkeit, die Komponenten in einer Umgebung grafisch zu ordnen und zu kombinieren. Die Komponenten werden als Puzzleteile dargestellt und können unter anderem Geräte, Sensoren oder Aktuatoren sein. Alle Komponenten müssen vorher in einem Datenbereich für eine bestimmte Umgebung angelegt werden. In dem Editor werden die Puzzleteile per Drag & Drop in die Modellierungsfläche eingefügt und miteinander kombiniert. Der Editor gibt zeitgleich Feedback über die

erlaubten Kombinationen mit den jeweiligen Komponenten. Da es sich um Puzzleteile handelt, ist der sequentielle Prozess der Verarbeitung - von der Eingabe bis zu der Ausgabe - leicht zu erkennen.

Eine prozessorientierte Modellierung bietet auch das System *homeBLOX* [RGW+13]. Das Ziel von homeBLOX ist es, die Erstellung von Prozessen für die Hausautomatisierung für die Nutzer zu vereinfachen. Ein Prozess besteht aus einer Sequenz von Knoten und Kanten, bei der die Knoten die Geräte beziehungsweise ihre Aktionen und die Kanten die auszulösenden Ereignisse repräsentieren. Dabei hat jede Sequenz eine Startbedingung die ebenfalls durch den Nutzer festgelegt wird. Eine Android-Applikation bietet die Nutzerschnittstelle für die Modellierung mit Drag & Drop. Der modellierte Prozess wird anschließend durch die darunter liegenden Komponenten übersetzt und ausgeführt.

3.2 Deployment von IoT-Umgebungen

Franco da Silva et al. [SBH+17] stellen einen Ansatz vor, bei dem das Deployment von IoT-Umgebungen automatisiert wird. Die IoT-Umgebung wird dabei durch den TOSCA [OAS13] Standard modelliert und demzufolge wird ein Topologiemodell erstellt. Das Deployment des Modells wird dann über eine TOSCA Laufzeitumgebung durchgeführt. Das Topologiemodell besteht im Allgemeinen aus der IoT-Applikation, der Middleware und dem IoT-Gerät. Im Gegensatz zur IoT-Umgebung in der MBP, schließt die IoT-Umgebung in diesem Ansatz, neben den IoT-Geräten, auch die Middleware und die IoT-Applikation mit ein. Ein Prototyp [SBK+16] wurde als Nachweis für das Deployment nach dem vorgestellten Prinzip gebaut. Der Prototyp besteht aus einem Mosquitto Broker und der OpenTOSCA [BBH+13] Laufzeitumgebung. Das in [SBK+16] beschriebene Beispielszenario verwendet den Mosquitto Broker und zwei Raspberry Pis, die über das MQTT Protokoll mit dem Broker verbunden sind und somit das Publish-/Subscribe-Prinzip verwirklichen. Ein Raspberry Pi hat einen Sensor und schickt Daten zu dem Broker und ein anderes Raspberry Pi, welches einen Aktuator besitzt, registriert sich bei dem Broker und empfängt Anweisungen. Für das Szenario wurde ein TOSCA Topologiemodell erstellt, welches die genannten Komponenten mit Installations- und Ausführungsskripten enthält. Das Deployment des Topologiemodells wird dann über OpenTOSCA und SSH durchgeführt. Das Modell kann auch für andere Szenarien verwendet werden, indem die Skripte oder die Komponenten ausgetauscht werden.

Das *LEONORE Framework* [VSID16] ermöglicht ein weiträumiges Deployment von Applikationen auf heterogene IoT-Geräte. Das Framework ist auf IoT-Geräte zugeschnitten, die ihre eigene Laufzeitumgebung haben. Die Applikation wird zusammen mit den benötigten Bibliotheken, Installations- und Ausführungsanweisungen, sowie anderen Artefakten in einem Paket verpackt. Das Paket wird dann über ein festgelegtes Kommunikationsprotokoll auf das IoT-Gerät geschickt. Dort befindet sich ein Agent der die Pakete verwaltet, diese installiert und ausführt. Der Agent sowie weitere Komponenten, müssen auf dem IoT-Gerät im

Voraus installiert werden. Ebenso muss das Framework genügend Vorabinformationen über das IoT-Gerät haben, um ein passendes Paket zu erstellen. LEONORE hat die Möglichkeit die Komponenten zu replizieren, was sich positiv auf die Skalierbarkeit auswirkt.

SStreaMWare [GRL+08] benutzt, wie die MBP, das Konzept von Adaptern, um die Kommunikation zwischen heterogenen Sensoren mit unterschiedlichen Schnittstellen und der Middleware einzurichten. SStreaMWare basiert auf einer hierarchischen und serviceorientierten Architektur, welche auch Gateways miteinschließt. Diese Gateways werden in bestimmten Umgebungen eingesetzt, um die Sensoren in dieser Umgebung zu verwalten und die Adapter auszuführen. Ganz oben in der Hierarchie befindet sich die Kontrollstelle, die zugleich eine Interaktion mit den Nutzern bietet. Ein wichtiger Aspekt von SStreaMWare ist auch die Erstellung einer einheitlichen Darstellung von Daten, die von verschiedenen Sensoren stammen können. Die Daten sind, zu diesem Zweck, in drei Typen aufgeteilt: Messdaten, Zeitstempel und Sensoreigenschaften.

Ishaq et al. [IHR+12] haben eine Methode entwickelt, um das Deployment von Sensornetzwerken zu erleichtern und die manuellen Konfigurationsschritte zu reduzieren. Das Netzwerk von Sensoren ist dabei über ein Gateway von außen erreichbar. Die Kommunikation wird über eine REST API abgewickelt, wobei Implementierungen am Sensor und dem Gateway notwendig sind. Das Gateway ermöglicht eine einfache Verwaltung, insbesondere die Entdeckung von neu eingefügten Sensoren und den Zugriff auf diese.

Die MBP wurde von Franco da Silva et al. [FHPM18] eingesetzt, um eine verteilte Datenverarbeitung zu ermöglichen. Bei der vorgestellten Methode wird die effiziente Verarbeitung durch *Complex Event Processing* (CEP) [CM12] in der Nähe der Datenquellen erzielt. Anstatt alle Daten verschiedener Quellen an einem zentralen Verarbeitungsort zu sammeln, werden die *CEP-Querys* über die MBP an die geeignete Hardware geschickt und dort ausgeführt. Die Hardware bezeichnet in diesem Fall die IoT-Geräte, die an der MBP gebunden sind. Die Datenquellen und Datensenken müssen ebenfalls mit der MBP verbunden sein. Durch diesen Ansatz wird die Kommunikationslast reduziert und die vorhandene Hardware effizient ausgenutzt.

4 Aufgabenstellung

Mittels der MBP kann eine digitale Repräsentation der physischen IoT-Umgebung erstellt werden. Es sollte für Anwender möglich sein, ein Modell der Umgebung zu erstellen, mit dessen Hilfe die Registrierung und Anbindung der darin enthaltenen Geräte effizient ausgeführt werden kann. Zum Zeitpunkt vor dieser Bachelorarbeit wurden die IoT-Geräte in dem Prototyp einzeln konfiguriert und angeschlossen, was in komplexen IoT-Umgebungen sehr aufwendig sein kann. Ziel dieser Bachelorarbeit ist es, diesen Prozess durch ein grafisches Modellierungswerkzeug zu vereinfachen. Abbildung 4.1 zeigt wie so ein Werkzeug aussehen könnte - mit einer Palette und einem Modellierungsbereich.

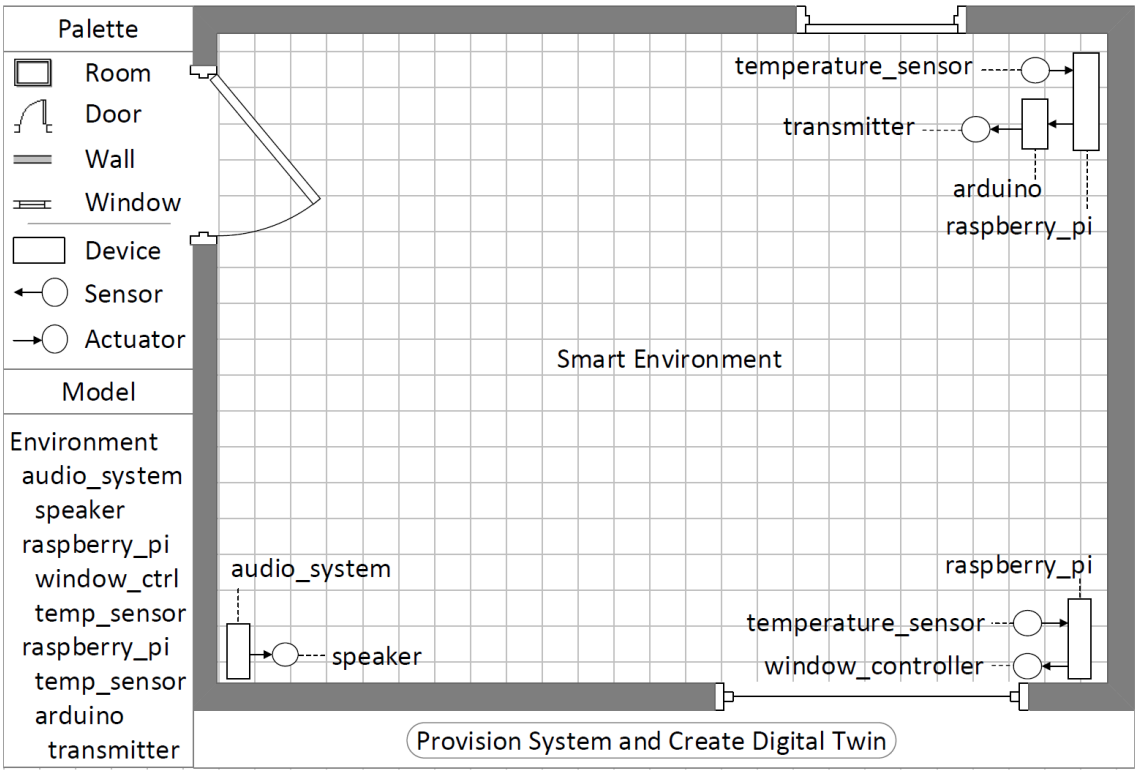

Abbildung 4.1: Modell eines Modellierungswerkzeugs [HBS+16]

Das Werkzeug soll die Möglichkeit bieten, die physische Umgebung zu modellieren und die enthaltenen Geräte in einem Schritt zu konfigurieren und anzuschließen. Dabei ergeben sich bestimmte Anforderungen, die auch spezifische Fälle einbeziehen. Die Ziele und die Anforderungen an das Werkzeug werden im weiteren Verlauf erläutert.

4.1 Ziele und Anforderungen

Bestimmte Ziele müssen erfüllt werden, um die Modellierung und das Deployment von IoT-Umgebungen zu erlauben und für die Nutzer so einfach wie möglich zu machen. Diese Ziele sind gleichzeitig auch Anforderungen an das Modellierungswerkzeug bzw. die verwendete Bibliothek und werden daher bei der Wahl der Technologien mitberücksichtigt. Es wurden insgesamt sieben Ziele festgelegt:

- **Benutzerfreundlichkeit:** Bei den vorgestellten Schritten der Verarbeitung in der MBP steht die Modellierung ganz am Anfang und erfordert eine Interaktion mit den Nutzern. Aus diesem Grund soll das Modellierungswerkzeug benutzerfreundlich sein. Das Werkzeug soll an erster Stelle nach dem Prinzip Drag & Drop realisiert werden. Dabei wird eine Palette verwendet, welche die verschiedenen Typen von Geräten, Sensoren und Aktuatoren enthält und ein Modellierungsbereich bereitgestellt, in dem das Modell erstellt wird. Zusätzlich ist es wichtig, dass das Werkzeug den Nutzern Feedback über die durchgeführten Operationen, Erfolge, Fehler und Zustände der Komponenten gibt.
- **Flexibilität des Layouts:** Eine IoT-Umgebung kann verschiedene Formen haben und unterschiedlich groß sein. In dem Modell soll es möglich sein, die Größe der Komponenten und das Layout der Umgebung beliebig zu ändern. Eine Umgebung kann beispielsweise aus mehreren Räumen und Ebenen bestehen, die mit Sensoren und Aktuatoren ausgestattet sind.
- **Spezifizierung der Komponenten:** Damit das Deployment erfolgreich durchgeführt werden kann, müssen zusätzliche Parameter bei den Geräten, Sensoren und Aktuatoren angegeben werden. Diese zusätzlichen Parameter sind zum Beispiel die IP- und MAC-Adresse bei den Geräten und die Adapter bei den Sensoren/Aktuatoren. Das Modellierungswerkzeug soll eine Möglichkeit bieten diese Informationen bei der jeweiligen Komponente einzugeben und zu speichern.
- **Modellierung komplexer Umgebungen:** In Abschnitt 2.1.1 wurden diverse Anwendungen und IoT-Umgebungen vorgestellt, die nicht immer eine einfache Struktur wie Smart Home haben. Es soll ebenso möglich sein eine komplexere IoT-Umgebung zu modellieren, wie zum Beispiel eine Smart Factory mit vielen IoT-Geräten.
- **Modellierung großer Mengen von Sensoren/Aktuatoren:** Ein IoT-Gerät kann eine große Menge von Sensoren und Aktuatoren enthalten. In diesem Fall erweist sich die grafische Modellierung als schwer und eine andere Vorgehensweise wird benötigt, um diese große Menge grafisch darzustellen. Das Deployment andererseits, muss nicht anders behandelt werden.
- **Einheitliches Datenformat:** Eine IoT-Umgebung soll unter Verwendung eines festgelegten Datenmodells modelliert werden. Das Modell soll dabei in ein standardisiertes Datenformat dargestellt und gespeichert werden. Dieser Ansatz erleichtert den Austausch der Daten, die Wiederverwendung und die Erweiterung.

- **Speicherung, Registrierung und Deployment des Modells:** Beim Speichern des Modells, wird der Nutzer zugewiesen, der das Modell erstellt hat. Nachdem das Modell erstellt wurde, soll die Registrierung und anschließend das automatisierte Deployment des gesamten Modells durchgeführt werden. Dafür soll die vorhandene Funktionalität des Prototyps verwendet werden. Der Nutzer bekommt dabei eine Rückmeldung über den Erfolg oder Misserfolg.

5 Modellierung und Deployment von IoT-Umgebungen

In diesem Kapitel werden die Konzepte und Methoden vorgestellt, die dazu beitragen die im Kapitel 4 definierten Ziele umzusetzen. Einige Anpassungen in der MBP sind für die Erfüllung bestimmter Ziele erforderlich. Die MBP wird diesbezüglich analysiert und die Anpassungen beschrieben. Weiterhin wird eine Analyse der notwendigen Technologien durchgeführt. Zu guter Letzt wird die Implementierung der Anpassungen und des Werkzeugs beschrieben und das Ergebnis im Bezug auf die festgelegten Ziele evaluiert.

5.1 Konzepte und Methoden

Die Funktionalität der MBP wird durch ein Modellierungswerkzeug erweitert, das die Registrierung und Anbindung von ganzen IoT-Umgebungen durchführen kann. Im weiteren Verlauf werden der Ansatz und die Idee der Umsetzung beschrieben. Die wichtigsten Komponenten und ihre Beziehungen sind in Abbildung 5.1 dargestellt.

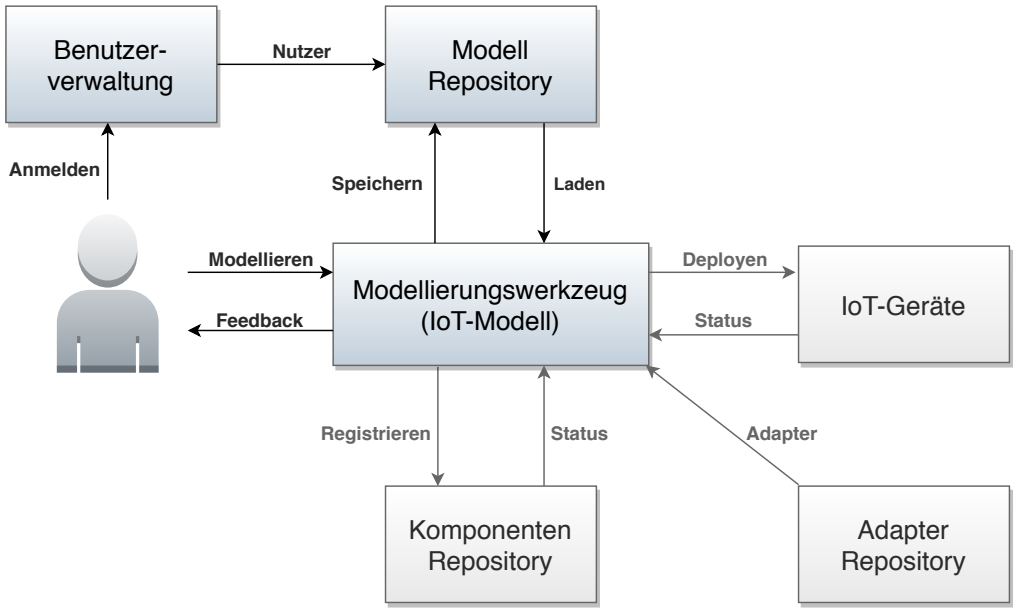

Abbildung 5.1: Modellierungswerkzeug Systemarchitektur

Im Mittelpunkt steht das *Modellierungswerkzeug*, welches eine Schnittstelle zu den Nutzern bietet und die grafische Erstellung von IoT-Modellen ermöglicht. Über die Benutzerschnittstelle kann der Nutzer Modelle erstellen, die Modelle speichern, die darin enthaltenden Komponenten registrieren und die Adapter deployen. Der Nutzer bekommt Feedback über die durchgeführten Operationen. Das Werkzeug bietet für die Modellierung eine Palette mit Grundriss-Objekten, IoT-Geräten, Sensoren und Aktuatoren, die mittels Drag & Drop in einem definierten Bereich eingefügt werden. Auf diese Weise wird die physische IoT-Umgebung widergespiegelt. Diese Komponente ist der Kern des Systems, welcher mit den restlichen Komponenten kommuniziert.

Die *Benutzerverwaltung* sorgt dafür, dass der Nutzer angemeldet ist, bevor er das Werkzeug bedienen kann. Die Komponente verwaltet die Registrierung, die Zugriffe und vergibt unterschiedliche Rechte für unterschiedliche Nutzer. So können IoT-Modelle von anderen Nutzern eingesehen und für andere Nutzer freigegeben werden, was ein Problem der Privatsphäre offenlegt. Dieses liegt aber nicht im Fokus dieser Bachelorarbeit und wird nicht weiter behandelt.

Das *Modell Repository* bewahrt die erstellten Modelle. Beim Speichern und Laden der Modelle wird der aktuell angemeldete Nutzer benötigt, der von der *Benutzerverwaltung* abgefragt wird. Beim Speichern wird der Nutzer dem Modell zugewiesen und beim Laden werden nur die Modelle geladen die diesem Nutzer gehören. Das Repository erlaubt außerdem Modifikation und Aufhebung der Modelle. Wie die Modelle gespeichert werden und welches Datenmodell dabei benutzt wird, wird später detaillierter beschrieben.

Das *Komponenten Repository* enthält die registrierten Komponenten (Geräte, Sensoren und Aktuatoren). Der Nutzer kann die Registrierung starten, sofern alle Komponenten in dem Modell spezifiziert worden sind. Insbesondere ist die Zuweisung der Adapter und der Geräte bei den Sensoren und Aktuatoren wichtig. Das Werkzeug bekommt bei dem Registrierungsprozess eine Rückmeldung über den Erfolg oder eventuellen Fehlern und dabei lässt sich der Zustand der Komponente auslesen. Die Rückmeldung wird zu dem Nutzer weitergeleitet.

Um das Deployment durchzuführen, müssen die Adapter vom *Adapter Repository* in das Werkzeug geladen werden. Die Adapter müssen vorher manuell erstellt und in das Repository gespeichert werden. Sie werden später bei der Modellierung für die Spezifizierung der Sensoren und Aktuatoren gewählt. Daher sollen sie schon beim Start des Werkzeugs geladen werden.

Die letzte wichtige Komponente sind die *IoT-Geräte*. Die Komponente umfasst alle Typen von IoT-Geräten, sowie das Gateway als eine Zwischenkomponente. Außerdem gibt es weitere Bestandteile die das Deployment möglich machen. Das IoT-Modell enthält Sensoren/Aktuatoren mit zugewiesenen Adaptern, die an die IoT-Geräte geschickt werden. Dabei bekommt das Werkzeug den Zustand der Komponenten zurückgegeben, welcher dann wiederum dem Nutzer angezeigt wird. Die Schnittstelle bietet das Deployment und das Undeployment der Modelle.

Die Komponenten *Komponenten Repository*, *Adapter Repository* und *IoT-Geräte* sind in Abbildung 5.1 heller dargestellt, weil sie samt ihrer Schnittstellen bereits in der MBP und dem Prototyp vorhanden sind. Die restlichen Komponenten müssen in dem Prototyp implementiert werden. Die Systemarchitektur in Abbildung 5.1 wurde für besseres Verständnis vereinfacht dargestellt. Es existieren weitere Komponenten und Beziehungen die ebenfalls ein Teil der gesamten Funktionalität sind. So ist neben dem Deployment auch das Undeployment möglich oder das Löschen ergänzend zu dem Speichern der Modelle.

In dem Prozess, von der Modellierung bis zu dem Deployment, gibt es eine bestimmte Reihenfolge in der Verarbeitung, die befolgt werden muss (Abbildung 5.2). Der Nutzer muss angemeldet sein bevor er ein Modell erstellen kann. Die Adapter müssen vor dem Start in das Werkzeug geladen werden, da sie schon während der Modellierung benutzt werden. Der Nutzer startet mit der Modellierung und danach werden die vorhandenen Komponenten registriert. Damit das Deployment durchgeführt werden kann, müssen die Komponenten des Modells registriert sein. Erst nach einem erfolgreichen Deployment kann das Undeployment durchgeführt werden. Das Speichern der Modelle kann unabhängig von der Registrierung und dem Deployment ausgeführt werden. Beim Löschen muss der Zustand der vorhandenen Komponenten berücksichtigt werden, um, falls nötig, gewisse Vorkehrungen zu ergreifen. Zuerst wird überprüft ob die Adapter bei den Geräten ausgeführt werden und anschließend ob die Komponenten registriert sind. Wenn das der Fall ist, müssen das Undeployment und die Deregistrierung durchgeführt werden. So wird verhindert, dass auf die Komponenten, nach dem Löschen im Modell, nicht mehr zugegriffen werden kann.

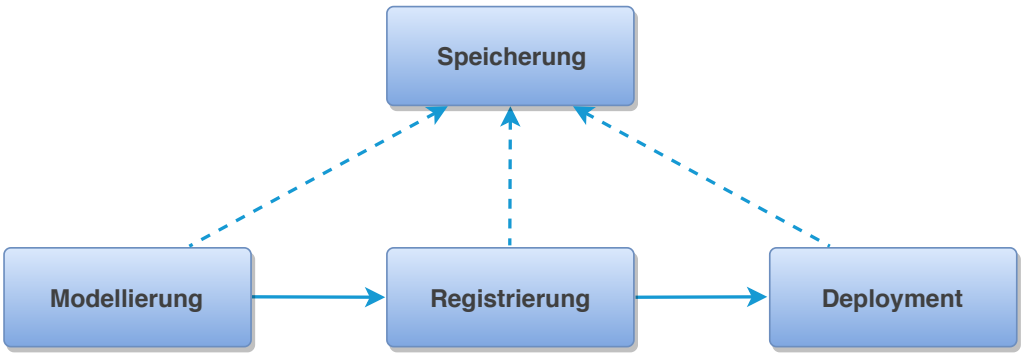

Abbildung 5.2: Verarbeitungssequenz

Wie die Abbildung 5.2 zeigt, kann die Speicherung der IoT-Modelle sofort nach der Modellierung, nach der Registrierung und nach dem Deployment erfolgen. Um Fehler zu vermeiden und den aktuellen Zustand festzuhalten, wird das Modell unmittelbar nach der Registrierung und nach dem Deployment automatisch gespeichert. So ist der Zustand nach der Registrierung in der Datenbank gespeichert bevor das Deployment ausgeführt wird. Die Daten die dabei gespeichert werden, können sich, abhängig von dem Zeitpunkt der Speicherung, unterscheiden. Die Datendarstellung wird im folgenden Abschnitt verdeutlicht.

5.1.1 Datenmodell

Durch homeML [NFD+07] wird ein Smart Home in einer hierarchischen Struktur dargestellt. Smart Home enthält Räume und die Räume enthalten IoT-Geräte. Es existieren verschiedene Arten von IoT-Umgebungen bei denen diese Hierarchie nicht immer angewendet werden kann. Um die Darstellung einer IoT-Umgebung so allgemein wie möglich zu machen, wird die Hierarchie bei der Erstellung des Modells nicht miteinbezogen. Im Mittelpunkt stehen Knoten, die die Komponenten der Umgebung vertreten und Beziehungen zwischen den Knoten. Die Komponenten sind die IoT-Geräte, Sensoren, Aktuatoren und Grundriss-Objekte der Umgebung. Nur die Geräte, Sensoren und Aktuatoren haben Beziehungen zueinander. Alle Komponenten haben, wie bei homeML, bestimmte Eigenschaften, die entweder für die Erstellung des Modells oder für die Registrierung und Deployment relevant sind. Abbildung 5.3 zeigt das Datenmodell, das für die Darstellung der IoT-Umgebungen verwendet wird.

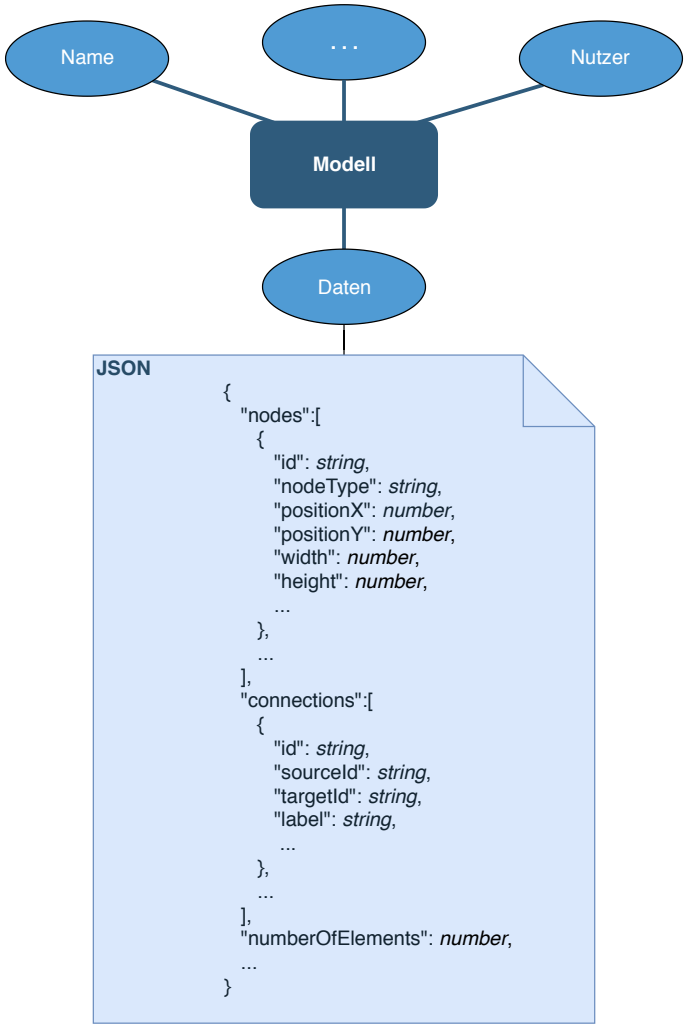

Abbildung 5.3: Datenmodell eines IoT-Modells

Ein Modell der IoT-Umgebung besitzt mindestens drei Attribute: den Namen, den zugewiesenen Nutzer und die eigentlichen Modelldaten. *Nutzer* bezeichnet die Person beziehungsweise ihre ID oder ihr Nutzername, die das Modell erstellt und benannt hat. Die Daten stehen für Knoten, Beziehungen und weitere Informationen die ein Modell auszeichnen. Solche Informationen sind beispielsweise die Position des Knotens, seine Größe oder generell die Anzahl der Knoten. Sie werden im JSON-Format erfasst und gespeichert. JSON[1] eignet sich gut für den Datenaustausch, kann mit JavaScript problemlos verwendet werden und hat eine einfach lesbare Form. Die Datendarstellung ist stark von der verwendeten Bibliothek abhängig und kann daher eine andere Struktur oder Form haben. Ebenso ist es möglich, dass XML statt JSON verwendet wird. Die Modelldarstellung in Knoten und Beziehungen kann auch für komplexe IoT-Umgebungen angewendet werden.

5.1.2 Modellierung komplexer Umgebungen

Der vorgestellte Ansatz und das Datenmodell zeigen wie die Mehrheit der festgelegten Ziele erfüllt werden. Für andere Ziele müssen zusätzliche Methoden entworfen werden. Was eine besondere Behandlung erfordert, ist die Möglichkeit komplexe Umgebungen und große Mengen von Sensoren und Aktuatoren zu modellieren.

Unter einer komplexen Umgebung wird eine physische IoT-Umgebung verstanden, die einen großen geographischen Bereich umfasst, eine hohe Anzahl an IoT-Geräten enthält oder nicht in Räumen dargestellt werden kann. Es soll möglich sein, IoT-Umgebungen wie Smart Factory oder Smart City zu modellieren. Um das zu ermöglichen, wird ein unbegrenzter, am Anfang vollkommen leerer Modellierungsbereich als Teil der Lösung verwendet. Der Modellierungsbereich hat in der Länge und Breite keine Einschränkungen und kann somit auf eine beliebige Größe skaliert werden. Große geographische Bereiche mit vielen IoT-Geräten können auf diese Weise in beliebiger Proportion modelliert werden. Wie das in dem Prototyp umgesetzt wird, wird in Abschnitt 5.4 beschrieben. Zusätzlich können spezielle Grundriss-Objekte, abhängig von der Umgebung, in der Palette eingefügt und bei der Modellierung verwendet werden. Sie ermöglichen den Nutzern, die digitale Repräsentation so genau wie möglich zu erstellen. Ein Raum ist ein Grundriss-Objekt wie jedes andere. Das vorgestellte Datenmodell macht es möglich, verschiedene Arten von IoT-Umgebungen darzustellen.

Ein weiteres Problem bei der Modellierung sind die IoT-Geräte, die eine hohe Zahl angeschlossener Sensoren und Aktuatoren haben. Wegen der hoher Anzahl können nicht alle Sensoren und Aktuatoren in dem grafischen Modell dargestellt werden. Als Lösung für das Problem werden Knoten eingeführt, die eine Menge von Sensoren und Aktuatoren beinhalten können. Sie werden als ein Element in der Palette und in dem Model angelegt und bieten eine beschränkte Sicht, bei der immer nur eine Teilmenge der Sensoren/Aktuatoren angezeigt wird. Der Nutzer kann die Menge durch Einfügen oder Löschen beliebig ändern.

[1] https://www.json.org/

Diese Knoten müssen eventuell bei der Speicherung, Registrierung, Deployment und anderen Operationen des Werkzeugs gesondert behandelt werden. Wie das umgesetzt wird, auch im Bezug auf das Datenmodell, wird in Abschnitt 5.4 genauer beschrieben.

Damit der vorgestellte Ansatz in der MBP umgesetzt werden kann, müssen einige Anpassungen in der MBP und dem Prototyp durchgeführt werden. Aus diesem Grund wird die MBP und die notwendigen Anpassung im folgenden Abschnitt analysiert.

5.2 Analyse der notwendigen Anpassungen in der MBP

Die MBP stellt den Großteil der Funktionalität bereit und ihre Architektur erlaubt eine einfache Integration des Modellierungswerkzeugs. In Abbildung 5.1 befinden sich Komponenten die bereits in der MBP vorhanden sind und weitere Komponenten die integriert werden müssen. Es wird lediglich auf die Erweiterung der Funktionalität der MBP eingegangen. Die Anpassungen beziehen sich ausschließlich auf die Erweiterungen, die als Voraussetzung für die Integration des Werkzeugs gelten und nicht auf die eigentlichen Komponenten des Werkzeugs.

Die MBP unterscheidet zwischen einfachen Nutzern und Experten. Die Aufgabe des einfachen Nutzers ist die Modellierung der physischen IoT-Umgebung und die des Experten die Bereitstellung der technischen Informationen, die für die Anbindung der IoT-Geräte notwendig sind [HBS+16]. Der vorgestellte Ansatz beschreibt eine Nutzerzuweisung bei den IoT-Modellen, was auf die Einrichtung einer Benutzerverwaltung führt. Zusätzlich zu den zwei Nutzertypen, soll die MBP den Nutzern die Möglichkeit bieten, sich bei der MBP zu identifizieren und diese unabhängig zu verwenden. Die Unterscheidung zwischen einfachen Nutzern und Experten soll in der MBP erhalten bleiben. Die Aufgabe der Benutzerverwaltung ist es, die Registrierung und anschließend die Anmeldung der Nutzer zu ermöglichen. Außerdem werden die Zugriffe auf die Ressourcen koordiniert und abhängig von dem Nutzer eingeschränkt. Jeder Nutzer kann zusätzlich ein Administrator sein, der mehr Rechte als andere Nutzer hat. Der Administrator kann zum Beispiel die Nutzer verwalten, neue Nutzer einfügen oder Vorhandene löschen. Das System soll insbesondere zu jedem Zeitpunkt in der Lage sein, den aktuell angemeldeten Nutzer bereitzustellen. All diese Funktionen sind in dem aktuellen Prototyp nicht vorhanden und müssen umgesetzt werden.

Das Modellierungswerkzeug enthält eine Palette mit Grundriss-Objekten und verschiedenen Typen von IoT-Geräten, Sensoren und Aktuatoren. Ein IoT-Gerät kann ein Raspberry Pi[2], Computer, Smartphone, Smartwatch und vieles mehr sein. Beispiele für Sensorentypen sind Bewegungssensor, Temperatursensor, Schallsensor, Kamera etc. Auf der anderen Seite kann ein Aktuator ein Schalter, Lautsprecher, LED oder eine Klimaanlage sein. Die Definition der IoT-Geräte, sowie der Sensoren und Aktuatoren in der MBP schließt den Typ

[2] https://www.raspberrypi.org/

nicht mit ein. Diese Komponenten sollen mit noch einem Typparameter erweitert werden, der bei der Registrierung der Komponenten angegeben werden muss. Die Palette enthält eine begrenzte Anzahl der Komponententypen, die in dieser Bachelorarbeit festgelegt werden. Später soll es möglich sein, diese feste Menge leicht zu erweitern.

Es sind keine große Änderungen in der Systemarchitektur der MBP (Abbildung 2.2) erforderlich. Vor allem die Schritte der Verarbeitung bleiben gleich. Der einzige Unterschied ist, dass es mehrere verschiedene Nutzer gibt, die ihre eigene Entwürfe erstellen. Die identifizierten Anpassung werden in dem Prototyp umgesetzt. Bevor mit der Implementierung begonnen wird, werden die benötigten Technologien und Bibliotheken analysiert und ausgewählt.

5.3 Analyse und Auswahl der Technologien

Der Prototyp der MBP benutzt das Spring Framework[3] für die Implementierung des Backends. Das Open-Source-Framework bietet viele Module, die die Entwicklung leichter machen. So wird Spring Data für die Verknüpfung verschiedener relationaler und NoSQL-Datenbanken eingesetzt, Spring Boot für die Konfiguration, Spring Web für die Webanwendungen und die REST API und Spring Security für die Sicherheit. Das Framework kann ohne weiteres auch für die Ziele dieser Bachelorarbeit verwendet werden. Das Frontend wurde mit AngularJS[4] implementiert. Dies ist ein JavaScript-Webframework, welches die Syntax von HTML erweitert. AngularJS hat viele Vorteile, wie zum Beispiel die automatische Synchronisation zwischen dem View und der Anwendungslogik oder die einheitliche Struktur bei der Entwicklung. Eine wichtige Eigenschaft von AngularJS ist die Möglichkeit, die Datentypen von JavaScript zu verwenden. So können externe Bibliotheken, ohne Zwischenschritte, eingebunden und benutzt werden. Für die Modellierung reicht AngularJS alleine nicht aus und daher werden zusätzliche Bibliotheken eingebunden. Es existieren Bibliotheken, die speziell für diese Anwendungsfälle entwickelt worden sind. Die meisten von ihnen sind kostenpflichtig, wobei ein Teil gleichzeitig eine kostenlose Version mit beschränkten Funktionalität bereitstellt. Im Folgenden werden einige der Bibliotheken vorgestellt und analysiert.

GoJS[5] ist eine JavaScript und TypeScript Bibliothek, die auch mit AngularJS verwendet werden kann. GoJS hat viele Funktionen, die es möglich machen, verschiedene Arten von Diagrammen und Graphen zu erstellen. Außerdem bietet GoJS hunderte von Vorlagen die nur angepasst werden müssen. Die ausführlich dokumentierte API und viele Anwendungsbeispiele erleichtern den Einstieg für neue Nutzer. In Betracht der Anforderungen, bietet GoJS die Möglichkeit eine Palette zu erstellen und die Elemente mit Drag & Drop zu bedienen. Die Elemente können bearbeitet, gedreht, verkleinert oder vergrößert werden und

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

somit problemlos verschiedene Layouts erstellen. Das verwendete Datenmodell basiert auf Knoten und Verbindungen und dabei wird zwischen einer Graphen- und einer Baumstruktur unterschieden. Die Daten können daher leicht in JSON oder XML dargestellt werden. Die Bibliothek erfüllt die Anforderungen und kann für die Umsetzung der Ziele eingesetzt werden. Sie ist jedoch kostenpflichtig und kommt daher für die prototypische Umsetzung in dieser Arbeit nicht in Frage.

Im Gegensatz zur GoJS ist JointJS[6] eine kostenlose JavaScript Bibliothek die eine kommerzielle Version mit dem Namen Rappid hat. Rappid besitzt, genauso wie GoJS, die erforderlichen Funktionen für die Umsetzung der Ziele. Mit JointJS können keine Paletten erstellt werden, aber eine Interaktion mit den Elementen ist immernoch möglich. In diesem Fall ist die Bearbeitung der Elemente beschränkt, indem sie nicht gedreht werden und nicht ihre Größe verändern können. Somit beschränkt sich auch die Modellierung. Die Diagramme können mühelos im JSON-Format exportiert und importiert und die Daten dabei beliebig erweitert werden. Die Nutzung ist ausreichend beschrieben und es existieren ebenfalls Beispiele, die als Vorlagen verwendet werden können.

JsPlumb[7] ist eine weitere JavaScript Bibliothek für die Erstellung von Diagrammen. Die Bibliothek kommt, wie JointJS, in zwei Versionen. Die kostenlose Version ermöglicht eine Interaktion mit den Elementen mittels Drag & Drop. Die zusätzlichen Funktionen, wie die Modifikation der Elemente oder die Verwendung einer Palette sind in dieser Version nicht enthalten. Knoten und Kanten repräsentieren die Diagramme und können beim Speichern in JSON umgewandelt werden. Die angegebenen Vorlagen geben viele Informationen über die Verwendung von jsPlumb und können zudem leicht angepasst und benutzt werden. Alle Vorteile der Bibliothek sind in der Dokumentation gut beschrieben. JsPlumb benutzt Events, um vordefinierte Operationen bei bestimmten Ereignissen, die bei der Interaktion entstehen, auszuführen. Diese Funktionalität erhöht die Qualität der Modellierung.

Ein anderer Typ der Bibliotheken ist die mxGraph[8] Bibliothek. Sie ist Open-Source in GitHub verfügbar. Das Paket beinhaltet, außer den JavaScript Dateien, mehrere Backend Implementierungen in verschiedenen Programmiersprachen. Sie können benutzt werden, um die Diagramme zu speichern oder zu laden. XML wird für die Darstellung der Daten und Diagramme verwendet. Die Bibliothek stellt auch eine vollständige Vorlage mit zahlreichen Funktionen bereit.

Es gib viele andere Bibliotheken, die entweder kommerziell sind, wie die von Syncfusion[9] oder yWorks[10], oder weniger Funktionen als die oben Vorgestellten haben. Bei einigen Bibliotheken liegt der Schwerpunkt in anderen Bereichen. So wird D3[11] meistens für die

[6] https://www.jointjs.com/opensource
[7] https://jsplumbtoolkit.com/
[8] https://github.com/jgraph/mxgraph
[9] https://www.syncfusion.com/javascript-ui-controls/js-diagram
[10] https://www.yworks.com/products/yfiles-for-html
[11] https://d3js.org/

Datenmanipulation und Visualisierung und Mermaid[12] für die Erstellung der Diagrammen durch Text benutzt. Aus diesem Grund werden diese Bibliotheken hier nicht weiter behandelt.

Eine Eigenschaft die bei der Wahl der Bibliothek eine wichtige Rolle spielt ist die Menge der Funktionen der kostenlosen Version. Viele der vorgestellten Bibliotheken erfüllen die Anforderungen, aber wenige von ihnen können kostenlos genutzt werden. Daher wird GoJS nicht im Betracht gezogen, obwohl diese eine sehr beliebte Bibliothek ist, mit Vorlagen die gut zur der Vorstellung des Modellierungswerkzeugs passen. Obwohl mxGraph alle benötigte Funktionen enthält und frei verwendet werden kann, ist die Bibliothek im Vergleich zu den anderen komplexer und die API ist nicht ausreichend dokumentiert. Die Anpassung der vorgegebenen Vorlage für die Ziele dieser Bachelorarbeit wäre in diesem Fall zu aufwendig. Die kostenlose Versionen von JointJS und jsPlumb sind, was die Funktionalität angeht, sehr ähnlich. Sie bieten beinahe die selben Funktionen, was die Wahl schwer macht. Für die Entwicklung des Werkzeugs wird letztendlich jsPlumb benutzt, aufgrund von adäquater Vorlagen die bei der Entwicklung von Nutzen sein könnten. Zu beachten ist, dass JointJS genauso gut den Zweck erfüllen würde. Ein Überblick der Eigenschaften der vorgestellten Bibliotheken ist in Tabelle 5.1 dargestellt.

Bibliothek	Drag & Drop	Palette	Bearbeiten, Drehen, Größe ändern	Aufwand	Kostenlos	Datenformat
GoJS	✓	✓	✓	✓	✗	JSON/XML
JointJS*	✓	✗	✗	✓	✓	JSON
jsPlumb*	✓	✗	✗	✓	✓	JSON
mxGraph	✓	✓	✓	✗	✓	XML

* Open-Source Version

Tabelle 5.1: Vergleich der vorgestellten Bibliotheken

Die kostenlose Version von jsPlumb bietet nur bestimmte Basisfunktionen und daher wird jQuery[13] eingesetzt, um die Möglichkeiten zu erweitern. Mit jQuery können zusätzliche Daten zu den UI-Elementen zugeordnet werden, somit können die Komponenten durch einen festgelegten Eingabebereich spezifiziert werden. Die speziellen Daten können jederzeit, insbesondere beim Speichern, wieder ausgelesen werden. Das Ändern der Größe und das Drehen der Elemente können ebenfalls mit Hilfe von jQuery umgesetzt werden. JsPlumb in Kooperation mit jQuery erfüllt die Anforderung und kann eingesetzt werden, um das entsprechende Modellierungswerkzeug zu implementieren.

[12] https://mermaidjs.github.io/
[13] https://jquery.com/

5.4 Implementierung

Die ausgewählten Technologien wurden angewendet, um den Prototyp der MBP mit einem Modellierungswerkzeug zu erweitern. Das Ergebnis ist eine Webanwendung, die Modellierung und Deployment von IoT-Umgebungen möglich macht. Bei der Implementierung wurde darauf geachtet, dass die vorhandene Richtlinien und der Implementierungsstil eingehalten werden, um die Konsistenz zu gewährleisten. Zuallererst wird die Implementierung der Anpassung beschrieben und dann wird auf die Implementierung des Werkzeugs eingegangen.

5.4.1 Implementierung der Anpassungen

Die, in Abschnitt 5.2, identifizierten Anpassungen der MBP wurden in dem Prototyp umgesetzt. Durch die Benutzeroberfläche können sich die Nutzer anmelden und registrieren. Es wurde eine einfache Registrierung bereitgestellt, bei der die Nutzer zusätzlich zu dem Nutzernamen und Passwort auch ihren Vornamen und Nachnamen eingeben müssen. Die Benutzerverwaltung wurde mit Hilfe von Spring Security[14] implementiert. Spring Security lässt sich einfach mit der REST API verwenden und erleichtert die Authentifizierung und Autorisierung der Nutzer. Für die Authentifizierung wird die *Basisauthentifizierung* verwendet, eine Methode, die wenn sie in Kombination mit HTTPS verwendet wird, eine ausreichende Sicherheit bietet. Die Clients authentifizieren sich mit dem HTTP-Authorization-Header und müssen dabei einen gültigen Nutzernamen und Passwort verwenden. Der Nutzername und das Passwort werden in der Form *Nutzername:Passwort* mit Base64 kodiert. Listing 5.1 zeigt wie der Header mit den Nutzernamen *admin* und das Passwort *admin* aussieht.

```
Authorization: Basic <admin:admin>
Authorization: Basic YWRtaW46YWRtaW4=
```

Listing 5.1: HTTP-Authorization-Header

Diese Anmeldeinformationen sind als Standard festgelegt und in der Datenbank gespeichert. Bei dem Client wird die entsprechende Base64-Kodierung durchgeführt und die Daten, bei einer erfolgreichen Authentifizierung, in einem *Cookie* gespeichert. Auf diese Weise bleibt der Nutzer für eine bestimmte Zeit angemeldet. Auf dem Server werden die Anmeldeinformationen überprüft und die Nutzer gespeichert oder geladen. Bei der Registrierung wird darauf geachtet, dass der eingegebene Nutzername nicht bereits vorhanden ist und dass das Passwort verschlüsselt gespeichert wird. Der Zugriff auf die Ressourcen wird kontrolliert und ggf. verweigert wenn keine ausreichende Autorisierung vorhanden ist. Der Nutzer bekommt zweckgemäß ein Feedback. Spring Security bietet unter anderem die Möglichkeit, die Informationen über den aktuell angemeldeten Nutzer auf eine einfache Weise zu bekommen.

[14] https://spring.io/projects/spring-security

Ein Nutzer kann ein Administrator sein und zusätzliche Rechte besitzen. In der aktuellen Implementierung können die Administratoren andere Nutzer einsehen, neue Nutzer hinzufügen und Vorhandene löschen oder bearbeiten. Dafür wurde ein Bereich in der Benutzeroberfläche bereitgestellt. Die Unterscheidung zwischen einfachen Nutzern und Administratoren ist für das Modellierungswerkzeug nicht von Bedeutung, kann aber für zukünftige Arbeiten von Nutzen sein.

Neben der Implementierung der Benutzerverwaltung, wurden auch die IoT-Geräte, Sensoren und Aktuatoren in dem Prototyp verändert. Die Attribute der Komponenten wurden mit einem zusätzlichen Attribut erweitert, das für den Typ der Komponenten steht. Diese Änderung zieht weitere Anpassung mit sich. So muss bei der Registrierung und dem Deployment, der Typ der Komponenten vorhanden sein. Die Erstellung der Adapter bezieht sich ebenfalls auf die vorher festgelegten Typen. Zum Zweck der Modellierung wurde eine feste Menge der Komponententypen ausgewählt und in der Palette eingefügt. Die Menge ist in Tabelle 5.2 zu sehen.

IoT-Geräte	**Sensoren**	**Aktuatoren**
Raspberry Pi	Camera	Light
Arduino	Sound	LED
NodeMCU	Temperature	Speaker
Computer	Humidity	Buzzer
Laptop	Gas	Vibration
TV	Light	Heater
Smartphone	Motion	Air Conditioner
Smartwatch	Location	Switch
Audio System	Gyroscope	Motor
Voice Controller	Proximity	Default Actuator
Camera	Touch	
Default Device	Vibration	
	Default Sensor	

Tabelle 5.2: Komponententypen

Jeder Eintrag in der Tabelle hat ein Symbol zugewiesen bekommen, das für die Modellierung verwendet wird. Außerdem gibt es jeweils für IoT-Geräte, Sensoren und Aktuatoren ein allgemeines Symbol ohne festgelegten Typ. Dieses Element kann für die Komponenten verwendet werden, dessen Typ nicht in der Palette vorhanden ist. Die Implementierung der Palette und des Werkzeugs wird im folgenden Abschnitt beschrieben.

5.4.2 Implementierung des Werkzeugs

Das Modellierungswerkzeug wurde mit jsPlumb in dem Prototyp der MBP implementiert. Die JavaScript Bibliothek hat sich als eine gute Wahl für diese Aufgabe erwiesen, dennoch mussten mehrere Funktionen von Grund auf implementiert werden. Es gab vor allem keine Schwierigkeiten bei der Verwendung mit AngularJS. Der größte Teil des Werkzeugs ist im Frontend implementiert. Dort befindet sich die notwendige Funktionalität, um IoT-Modelle zu erstellen. Über die REST API wird mit dem Backend kommuniziert, bei dem die IoT-Modelle gespeichert, registriert und deployt werden. Wie das Werkzeug in der Benutzeroberfläche integriert ist, wird in Abbildung 5.4 dargestellt.

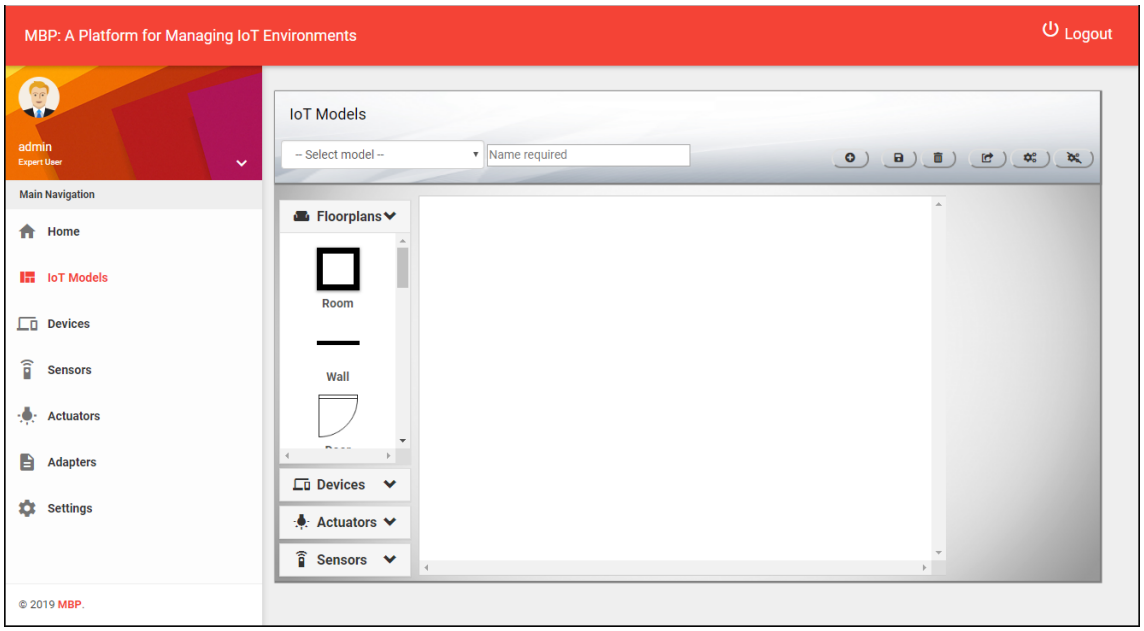

Abbildung 5.4: MBP Modellierungswerkzeug

Sobald sich der Nutzer angemeldet hat, kann er unter *IoT Models* das Werkzeug finden und mit der Modellierung beginnen. Am Anfang wird ein leerer Modellierungsbereich angezeigt, um ein neues Modell zu erstellen. Gleichzeitig werden alle vorhandene IoT-Modelle des Nutzers geladen und in einer Liste zusammengestellt. Die Funktionsweise des Werkzeugs wird durch die Phasen in der Verarbeitung beschrieben, angefangen mit der Modellierung bis zu dem Deployment. Dabei wird ein IoT-Modell als Beispiel für die Illustration verwendet.

Modellierung

Das Werkzeug ist wie folgt aufgebaut: Auf der linken Seite befindet sich die Palette, in der Mitte ist der Modellierungsbereich und auf der rechten Seite werden Informationen über die Komponenten angezeigt. In dem oberen Bereich befindet sich die Werkzeugleiste, in der

die IoT-Modelle des Nutzers in einer Drop-Down Liste angezeigt werden. Der Nutzer kann zwischen den Modellen wechseln und in das Eingabefeld den Namen des Modells eingeben oder ändern. Der Name ist für die Speicherung und andere Operationen erforderlich. Rechts in der Werkzeugleiste sind die Buttons, jeweils einer für die Erstellung eines neuen Modells, Speicherung, Löschung, Registrierung, Deployment und Undeployment. Die Palette funktioniert als ein Container und wurde mit Hilfe von jQuery implementiert. Sie wird durch das Auf- und Zuklappen bedient und zeigt dabei die Elemente der Grundriss-Objekte, IoT-Geräte, Aktuatoren und Sensoren. Sie können mittels Drag & Drop in dem Modellierungsbereich eingefügt werden. Dort kann ihre Größe und ihre Ausrichtung durch Drehen verändert werden. Da jsPlumb diese Funktionen nicht bietet, wurde jQuery als eine Ausweichlösung eingesetzt. Abbildung 5.5 zeigt ein Beispielmodell, das auf diese Weise erzeugt wurde.

Abbildung 5.5: Modellierungswerkzeug - Modellierung

Das Beispielmodell repräsentiert ein Haus mit mehreren Ebenen, die nebeneinander modelliert worden sind. Es enthält mehrere IoT-Geräte, die mit Aktuatoren, Sensoren und anderen Geräten verbunden sind. Bei einer Verbindung besteht die Möglichkeit, den Namen beziehungsweise die Art der Verbindung anzugeben. In dem Modell ist beispielsweise ein Smartphone mit einer Smartwatch über Bluetooth verbunden. Wenn ein Element, das kein Grundriss-Objekt ist, in dem Modell ausgewählt wird, dann erscheinen seine Informationen auf der rechten Seite. Dieser Bereich ermöglicht die Spezifizierung der Komponenten. In der Kopfzeile des Bereiches ist angegeben, ob es sich um ein Gerät, Aktuator oder Sensor handelt. Abhängig von der Komponente werden unterschiedliche Eingabefelder bereitgestellt. Für IoT-Geräte sind das Felder für den Namen, Typen, MAC-Adresse, IP-

Adresse, Nutzernamen und RSA-Schlüssel. Für Aktuatoren/Sensoren muss neben den Namen und Typen auch das Gerät und ein Adapter aus der Liste der geladenen Adapter ausgewählt werden. Das Gerät kann in das Feld nicht eingegeben werden. Wenn eine Verbindung zwischen einem Gerät und einem Aktuator/Sensor in dem Modell erstellt wird, dann wird das Gerät automatisch dem Aktuator/Sensor zugewiesen. Diese Informationen über die Elemente in dem Modell werden im Hintergrund mittels jQuery gespeichert. In der Fußzeile des Bereiches befindet sich der Status über die Registrierung und bei den Aktuatoren und Sensoren wird zusätzlich der Status über das Deployment angezeigt. Dort werden auch Fehlermeldung bei der Registrierung und dem Deployment angezeigt.

Registrierung

Nachdem das Modell erstellt wurde und alle vorhandenen Komponenten spezifiziert worden sind, kann die Registrierung der Komponenten durchgeführt werden. Die Komponenten werden durch eine Iteration einzeln registriert. Dabei wird für jede Komponente eine HTTP-Anfrage an den Server geschickt und dort übernimmt die vorhandene Implementierung die Registrierung der Komponenten. Die Anfrage enthält die Details der Komponente in einer JSON-Darstellung. Zuerst werden die IoT-Geräte registriert, um die generierte ID zu bekommen, die für die Registrierung der Aktuatoren und Sensoren notwendig ist. Danach werden auch die Aktuatoren und Sensoren registriert.

Abbildung 5.6: Modellierungswerkzeug - Registrierung

Wenn der Prozess der Registrierung abgeschlossen wurde, wird das Modell mit den Statusinformationen der Komponenten automatisch gespeichert. Es ist wichtig, dass der Zustand des Modells aktuell bleibt, falls keine manuelle Speicherung durch den Nutzer erfolgt. Das Ergebnis der Registrierung des Beispielmodells ist in Abbildung 5.6 dargestellt. Die Elemente in dem Modell bekommen einen farbigen Rahmen: grün - die Komponente wurde erfolgreich registriert und rot - es ist ein Fehler aufgetreten. Die Fehlermeldung wird in dem Statusbereich angezeigt. Das ausgewählte Element aus Abbildung 5.6 ist ein Bewegungssensor im Flur, bei dem der Name und der Adapter nicht eingegeben worden sind. Die Parameter sind für die Registrierung notwendig und deshalb ist der Fehler aufgetreten. Der Fehler kann korrigiert und die Registrierung erneut ausgeführt werden. Dabei werden nur die Komponenten betrachtet, die noch nicht registriert sind.

Deployment

Die Registrierung ist eine Voraussetzung, um das Deployment durchzuführen. Die Komponenten die nicht erfolgreich registriert sind, werden bei dem Deployment scheitern. Es werden nur Aktuatoren und Sensoren betrachtet, wobei die Reihenfolge nicht relevant ist. Das Modell wird ebenfalls nach dem Deployment automatisch gespeichert. Mit dem Deployment ist der Prozess gemeint, die ausgewählten Adapter zu den Geräten zu schicken, die in das Modell mit den jeweiligen Aktuator/Sensor verbunden sind. Die Komponenten werden, wie bei der Registrierung, iteriert und einzeln deployt. Dabei wird eine HTTP-Anfrage an den Server geschickt, um das Deployment zu starten. Die Funktionalität ist bereits auf dem Server vorhanden: Die Adapter werden mit zusätzlichen Ausführungsskripten über SSH an die Zielgeräte geschickt und dort ausgeführt. Von diesem Zeitpunkt an, fungieren die Geräte als MQTT Clients und schicken Daten zu dem MQTT Broker (Sensoren) oder erhalten Anweisungen von dem Broker (Aktuatoren). Der Server ist ein weiterer MQTT Client, der mit dem Broker kommuniziert, um Sensordaten zu erhalten oder Anweisungen zu schicken. Nachdem alle Adapter verschickt und ausgeführt sind, ist das automatisierte Deployment der ganzen IoT-Umgebung abgeschlossen und die digitale Repräsentation der Umgebung wurde erstellt. Der Nutzer hat, durch das grafische Modell, einen Überblick über die Umgebung und sieht welche Geräte aktiv sind. Das Beispielmodell nach dem Deployment wird in Abbildung 5.7 sichtbar gemacht. Der Status der IoT-Geräte in dem Modell ändert sich nach dem Deployment nicht, sie bleiben als registriert gekennzeichnet. Die Aktuatoren und Sensoren, bei denen das Deployment erfolgreich war, bekommen einen gelben Rahmen. Ein Beispiel des Erfolgs ist der Temperatursensor aus Abbildung 5.7 der mit einem Raspberry Pi im Schlafzimmer verbunden ist. Die Statusinformationen zeigen, dass der Sensor registriert ist und sein Adapter deployt wurde. Wenn ein Fehler bei dem Deployment auftritt dann bekommt das Element einen roten Rahmen und Feedback in den Statusinformationen. Das Deployment kann nur wieder bei den Komponenten ausgeführt werden die noch nicht deployt sind.

Abbildung 5.7: Modellierungswerkzeug - Deployment

Neben dem Deployment ist auch das Undeployment des Modells möglich. Bei den Skripten, die mit dem Adapter geschickt werden, befindet sich auch ein Skript, das die Ausführung des Adapters stoppt. Es wird aktiviert wenn sich der Nutzer für das Undeployment entscheidet. Das Undeployment in das Modellierungswerkzeug funktioniert ähnlich wie das Deployment mit der Ausnahme, dass die passende HTTP-Anfrage geschickt wird und nur die Sensoren und Aktuatoren betrachtet werden die bereits deployt sind. In der grafischen Darstellung wird der gelbe Rahmen mit einem Grünen ersetzt oder einem Roten wenn ein Fehler auftritt.

Speicherung

Die Speicherung kann jederzeit durch den Nutzer erfolgen, sobald der Name eingegeben wurde. Bei dem Prozess werden alle Knoten und Verbindungen aus dem Modell ermittelt und ein JSON-Objekt generiert. In den Knoten werden modellspezifische Informationen eingefügt, wie zum Beispiel Informationen über ihre Position oder Größe. Weiter werden die Informationen aus der Spezifizierung der Komponenten mittels jQuery ausgelesen und in den Knoten platziert. Die Statusinformationen werden ebenfalls eingefügt. Die Verbindungen und ihre zugehörige Knoten werden mit jsPlumb ermittelt und zusammen mit ihrer Beschriftung in dem JSON-Objekt eingefügt. In Listing 5.2 ist ein Ausschnitt der JSON-Darstellung des Beispielmodells dargestellt.

```
{
   "nodes":[
      {
         "nodeType":"sensor",
         "elementId":"canvasWindow146",
         "clsName":"window sensor temperature-sensor custom jtk-node jtk-draggable jtk
          -droppable ui-resizable jtk-endpoint-anchor jtk-connected deployed-element",
         "positionX":756,
         "positionY":371,
         "width":28,
         "height":25,
         "id":"5cc834c5df7d480958169566",
         "name":"Temp Bedroom Floor 2",
         "type":"Temperature",
         "adapter":"5c7cecfa56c6e680f86a303b",
         "device":"Rasp Bedroom Floor 2",
         "deviceId":"5cc834c4df7d480958169560",
         "deployed":true
      },
      ...
   ],
   "connections":[
      {
         "id":"con_1484",
         "sourceId":"canvasWindow144",
         "targetId":"canvasWindow146",
         "label":"NAME",
         "labelVisible":false
      },
      ...
   ],
   "numberOfElements":136,
   ...
}
```


Listing 5.2: Ausschnitt der Datendarstellung

Es sind Informationen über den Temperatursensor aus Abbildung 5.7 zu sehen, der bereits registriert und deployt wurde. Dort ist außerdem seine Verbindung mit dem Raspberry Pi dargestellt. Sobald das JSON-Objekt generiert wurde, wird der eingegebene Name des Modells beigefügt und das Objekt über HTTP an den Server geschickt. Die Spring Implementierung findet den angemeldeten Nutzer, fügt den Nutzer dem Modell bei und speichert das Modell in das Modell Repository. Beim Laden werden nur die Modelle des Nutzers geladen und zu dem Client geschickt, bei dem sie wieder mit jsPlumb gezeichnet werden.

Das Modellierungswerkzeug erlaubt das Löschen der einzelnen Komponenten und des gesamten Modells. Beim Löschen wird geprüft, ob die Komponenten registriert und deployt worden sind, um das Undeployment und die Deregistrierung durchzuführen. Ohne diese Vorkehrung könnte auf die Komponenten und die Adapter nicht mehr zugegriffen werden. Das Gleiche wird beim Sensoren und Aktuatoren durchgeführt wenn in dem Modell die Verbindung zu dem Gerät getrennt wird. In diesem Fall hat der Sensor/Aktuator kein zugewiesenes Gerät mehr und der registrierte Zustand ist nicht mehr valide.

Komplexe Umgebungen

Für komplexe IoT-Umgebungen wird ein unbeschränkter Modellierungsbereich zur Verfügung gestellt. Die Nutzer können mehrere Räume, Ebenen und Gebäude in einem Modell erstellen. Die IoT-Geräte, Sensoren und Aktuatoren können auch außerhalb eines Raums platziert werden, wie in Abbildung 5.8 zu sehen ist. Durch die beschränkte Bildschirmgröße ist immer nur ein Teil des Modells sichtbar. Der Rest kann durch Scrollen eingesehen werden. Eine Verbesserung könnte durch eine Zoom-Funktion erzielt werden, die aber nicht in der kostenlosen Version von jsPlumb vorhanden ist. Aus der Implementierungssicht, ist das Einfügen von neuen Elementen in der Palette ein einfacher Prozess mit niedrigem Aufwand. Abhängig von der Umgebung können spezielle Grundriss-Objekte in der Palette eingefügt und später bei der Modellierung verwendet werden.

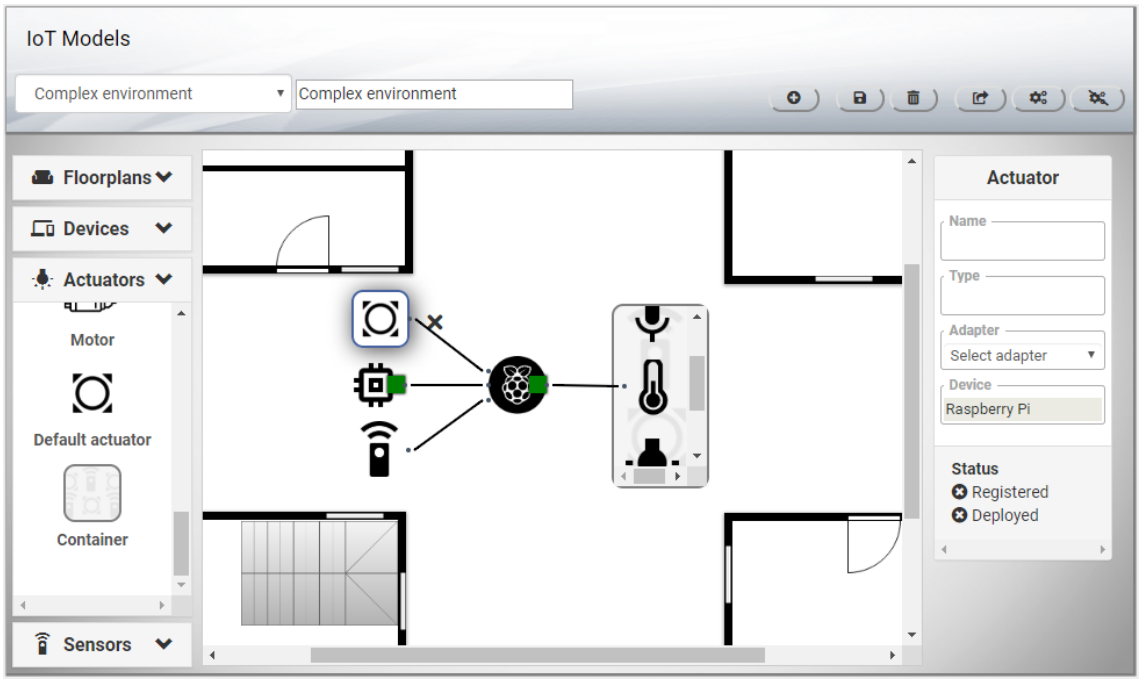

Abbildung 5.8: Modellierungswerkzeug - Komplexe Umgebungen

In Abbildung 5.8 wird eine Lösung für die Modellierung großer Mengen von Sensoren und Aktuatoren vorgestellt. In der Palette, bei den Sensoren und Aktuatoren, kann ein Container ausgewählt werden, der eine beliebige Menge von Sensoren und Aktuatoren aufnehmen kann. Auf diese Weise werden mehrere Elemente in einem kleinen Bereich des Modells eingefügt. In der Abbildung ist ein Raspberry Pi mit einem Container auf der rechten Seite verbunden, der unter anderem einen Schallsensor, Temperatursensor und Licht-Aktuator enthält. Auf der linken Seite befinden sich die Elemente mit allgemeinen Symbolen für einen Aktuator, Gerät und Sensor, bei dem der Typ eingegeben werden muss. Sensoren und Aktuatoren können direkt aus der Palette in dem Container eingefügt und aus dem Container entfernt werden. Durch Scrollen und Änderung der Größe des Containers kann eine beliebige Teilmenge angezeigt werden. Bei der Speicherung wird der Container als ein Knoten betrachtet, der Kindknoten enthält. In dem JSON-Objekt werden die Kindknoten als ein Array-Parameter bei dem Knoten eingefügt. Die Spezifizierung, Registrierung, Deployment und andere Operationen verlaufen gleich wie bei den anderen Elementen, da die Elemente in dem Container einzeln betrachtet werden.

Die Implementierung des Werkzeugs wurde nach den vorgestellten Konzepten und Methoden durchgeführt. Seine Funktionsweise und Verwendung wurde in diesem Abschnitt durch mehreren Schritten verständlich gemacht. Im folgenden Abschnitt werden der Ansatz und das Ergebnis der Implementierung im Bezug auf die definierten Zielen und Anforderungen analysiert und beurteilt.

5.5 Evaluation

Das Hauptziel dieser Bachelorarbeit ist es, die Anbindung der IoT-Geräte in der MBP durch ein Modellierungswerkzeug zu vereinfachen und effizient auszuführen. Zu diesem Zweck wurde ein Ansatz vorgestellt und in dem Prototyp der MBP umgesetzt. Durch das Modellierungswerkzeug ist es möglich, Modelle der physischen IoT-Umgebungen zu erstellen und die darin enthaltenen Geräte effizient zu konfigurieren und anzubinden. Im weiteren Verlauf wird das Ergebnis dieser Arbeit im Bezug auf die, im Kapitel 4, festgelegten Ziele evaluiert:

- **Benutzerfreundlichkeit.** Das Werkzeug wurde auf eine Art implementiert, sodass es leicht und intuitiv bedient werden kann. Für die Modellierung steht eine vordefinierte Menge der Komponenten in der Palette bereit, mit der die IoT-Modelle mit niedrigem Aufwand mittels Drag & Drop erstellt werden können. Zudem bietet das Werkzeug ausreichend Feedback über die ausgeführten Operationen, das den Nutzern in einer unaufdringlichen Weise mitgeteilt wird. Außerdem wird der Zustand der Komponenten visuell in dem Modell dargestellt, sodass der Nutzer den Zustand der IoT-Umgebung schnell einschätzen kann.

- **Flexibilität des Layouts.** Mit dem Werkzeug ist es möglich, ein Modell mit mehreren Räumen und Ebenen zu erstellen. In dem Modell können die Elemente gedreht und ihre Größe verändert werden. Sie können dazu an beliebige Positionen eingefügt werden. Das alles ermöglicht eine flexible Modellierung, bei der, verschiedene Layouts erstellt werden können.
- **Spezifizierung der Komponenten.** Auf der rechten Seite des Werkzeugs befindet sich ein Eingabebereich, bei dem die Komponenten spezifiziert werden können. Dort werden die erforderlichen Informationen eingegeben, die für die Registrierung und das Deployment notwendig sind. Der Nutzer bekommt bei Fehlern eine Rückmeldung über die erforderlichen Eingaben. Mit Hilfe von jQuery werden die eingegebenen Eigenschaften zu den UI-Elementen zugewiesen und später, wenn notwendig, wieder ausgelesen.
- **Modellierung komplexer Umgebungen.** Als eine Lösung für die Modellierung komplexer Umgebungen wurde für die Nutzer ein Modellierungsbereich bereitgestellt, in dem sie beliebige IoT-Umgebungen modellieren können. Die Nutzer können die Größe und Art des Modells frei bestimmen. So können neben dem Smart Home auch andere Umgebungen dargestellt werden.
- **Modellierung großer Mengen von Sensoren/Aktuatoren.** Die Container wurden eingeführt, um große Mengen von Sensoren und Aktuatoren in einem kleinen grafischen Bereich darzustellen. Der Inhalt der Container kann jederzeit geändert werden. Die Container müssen bei der Speicherung anders behandelt werden, aber bei der Registrierung und dem Deployment werden die Komponenten in dem Container wie alle andere in dem Modell betrachtet.
- **Einheitliches Datenformat.** Es wurde ein Datenmodell für die IoT-Modelle vorgestellt. Für die Datendarstellung wurde JSON ausgewählt, ein beliebtes Datenformat für den Datenaustausch und für die Verwendung mit JavaScript. Außerdem ist durch die Knoten und Verbindungen eine flexible Darstellung der Modelle gestattet.
- **Speicherung, Registrierung und Deployment des Modells.** In der MBP werden die IoT-Modelle abhängig von dem Nutzer gespeichert und geladen. Durch das Werkzeug können IoT-Umgebungen modelliert, die Modelle registriert und anschließend deployt werden. In einem Schritt kann eine große Menge von IoT-Geräten mit ihren Sensoren und Aktuatoren angebunden werden. Der Nutzer bekommt einen Überblick über die physische Umgebung und kennt den Zustand der Komponenten. So kann er sehen welche Komponenten aktiv sind und bei welchen Komponenten ein Fehler vorliegt.

6 Zusammenfassung und Ausblick

Im Rahmen dieser Bachelorarbeit wurde die MBP um ein grafisches Modellierungswerkzeug erweitert, mit dem die IoT-Umgebungen modelliert werden können. Es wurde ein Ansatz vorgestellt, bei dem die erzeugten IoT-Modelle bzw. die in dem Modell vorhandenen Geräte in einem Schritt angeschlossen werden können. Dazu wurde eine prototypische Implementierung realisiert, die im GitHub[1] verfügbar ist.

Zu Beginn der Arbeit wurden die Ziele festgelegt. Neben dem Hauptziel existieren weitere Ziele und Anforderungen, die in der Aufgabenstellung beschrieben sind. Der weitere Verlauf der Arbeit fokussiert sich auf die Umsetzung dieser Ziele. Im Kapitel 5 wurde die Systemarchitektur des Modellierungswerkzeugs vorgestellt und die vorhandenen Komponenten und Beziehungen beschrieben. Einige der Komponenten waren bereits in der MBP vorhanden und andere mussten eingefügt oder angepasst werden. Durch die Architektur wurde eine Reihenfolge der Verarbeitungsprozesse identifiziert, die zu dem Deployment der IoT-Umgebung führen. Weiterhin wurde das Datenmodell definiert, eine Datendarstellung die für die Erstellung und Speicherung der IoT-Modelle verwendet wird. Eines der Nebenziele dieser Arbeit war es, die Modellierung von komplexen Umgebungen mit vielen Sensoren und Aktuatoren zu ermöglichen. In diesem Sinne wurden einige Lösungen vorgestellt und später in dem Prototyp umgesetzt. Die Umsetzung wurde im Abschnitt 5.4.2 beschrieben.

Die Implementierung hat die Verwendung von bestimmten Bibliotheken gefordert, mit dessen Hilfe das Modellierungswerkzeug entwickelt werden kann. Zuerst wurden die vorhandenen Technologien in dem Prototyp der MBP untersucht und danach wurden verschiedene Bibliotheken für die Modellierung analysiert und verglichen. Für die Implementierung wurde jsPlumb[2] ausgewählt, eine JavaScript Bibliothek, mit der das Werkzeug in dem Prototyp der MBP implementiert wurde. Bei der Implementierung ist außerdem die Umsetzung der Anpassungen beschrieben, der Benutzerverwaltung und Typisierung der IoT-Geräte, Sensoren und Aktuatoren. Die Funktionsweise des Werkzeugs wurde anhand der Verarbeitungsschritte beschrieben. Die Modellierung nutzt die Elemente aus der Palette die mit Drag & Drop eingefügt werden. Die Registrierung sucht nach IoT-Geräten, Sensoren und Aktuatoren in dem Modell und registriert diese. Das Deployment sucht nach den registrierten Sensoren und Aktuatoren und verschickt die festgelegten Adapter an die

[1] https://github.com/IPVS-AS/MBP
[2] https://jsplumbtoolkit.com/

Zielgeräte. Die Speicherung konstruiert ein JSON-Objekt des IoT-Modells und speichert dieses in das Modell Repository. Zum Schluss wurden die festgelegten Ziele aufgelistet und ihre Umsetzung beurteilt.

Ausblick

Der vorgestellte Ansatz für die Modellierung und das Deployment kann durch weitere Anpassungen vereinfacht werden. Die Modellierung kann übersprungen werden, indem ein System entwickelt wird, das die Geräte in einer IoT-Umgebung erkennt und daraufhin ein Modell automatisch erstellt. Dafür kann *mermaid*[3] eingesetzt werden, eine Bibliothek, die Modelle aus Text erstellen kann. Ebenso kann die vorhandene jsPlumb Implementierung verwendet werden, wenn die Datendarstellung eingehalten wird. Das System muss die identifizierten Geräte in einem bestimmten Textformat als Eingabe bereitstellen.

Die Implementierung eines Modellierungswerkzeugs erfordert Berücksichtigung von vielen Details und Feinheiten, die viel Zeit in Anspruch nehmen. Die in dieser Bachelorarbeit realisierte Implementierung bietet die Grundfunktionalität und hat noch Raum für Verbesserungen. So kann eine Zoom-Funktion die Modellierung erleichtern oder der Zustand der IoT-Geräte präziser dargestellt werden. Zudem kann die Synchronisation mit der physischen IoT-Umgebung verbessert werden. Dabei ist die automatische Aktualisierung des Modells gemeint, wenn ein Gerät beispielsweise eingefügt oder entfernt wird. Dafür muss der Zustand des Geräts kontinuierlich abgefragt werden. Für zukünftige Arbeiten kann die Menge der Informationen über eine physische IoT-Umgebung in dem Modell erweitert werden. So wird für eine dreidimensionale Darstellung, zusätzlich zur Länge und Breite, auch die Höhe benötigt. Ebenso können die Nutzer mehr einbezogen werden, sodass das Modell die Sichtbarkeit und Freigabe für andere Nutzer ebenfalls speichert.

[3] https://mermaidjs.github.io/

Literaturverzeichnis

[AZZ+17] M. Alaa, A. Zaidan, B. Zaidan, M. Talal, M. Kia. „A review of smart home applications based on Internet of Things“. In: *Journal of Network and Computer Applications* 97 (2017), S. 48–65. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2017.08.017. URL: http://www.sciencedirect.com/science/article/pii/S1084804517302801 (zitiert auf S. 15, 18).

[BBH+13] T. Binz, U. Breitenbücher, F. Haupt, O. Kopp, F. Leymann, A. Nowak, S. Wagner. „OpenTOSCA–a runtime for TOSCA-based cloud applications“. In: *International Conference on Service-Oriented Computing*. Springer. 2013, S. 692–695 (zitiert auf S. 24).

[BR16] S. Boschert, R. Rosen. „Digital Twin—The Simulation Aspect“. In: *Mechatronic Futures: Challenges and Solutions for Mechatronic Systems and their Designers*. Hrsg. von P. Hehenberger, D. Bradley. Cham: Springer International Publishing, 2016, S. 59–74. ISBN: 978-3-319-32156-1. DOI: 10.1007/978-3-319-32156-1_5. URL: https://doi.org/10.1007/978-3-319-32156-1_5 (zitiert auf S. 20).

[BS11] D. Bandyopadhyay, J. Sen. „Internet of Things: Applications and Challenges in Technology and Standardization“. In: *Wireless Personal Communications* 58.1 (Mai 2011), S. 49–69. ISSN: 1572-834X. DOI: 10.1007/s11277-011-0288-5. URL: https://doi.org/10.1007/s11277-011-0288-5 (zitiert auf S. 15, 18).

[CM12] G. Cugola, A. Margara. „Processing Flows of Information: From Data Stream to Complex Event Processing“. In: *ACM Comput. Surv.* 44.3 (Juni 2012), 15:1–15:62. ISSN: 0360-0300. DOI: 10.1145/2187671.2187677. URL: http://doi.acm.org/10.1145/2187671.2187677 (zitiert auf S. 25).

[FHPM18] A. C. Franco da Silva, P. Hirmer, R. K. Peres, B. Mitschang. „An Approach for CEP Query Shipping to Support Distributed IoT Environments“. In: *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. März 2018, S. 247–252. DOI: 10.1109/PERCOMW.2018.8480241 (zitiert auf S. 25).

[FT14] E. Fleisch, F. Thiesse. *Internet der Dinge*. Hrsg. von N. Gronau et al. 2014. URL: http://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/technologien-methoden/Rechnernetz/Internet/Internet-der-Dinge (zitiert auf S. 15, 17).

[GBMP13] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami. „Internet of Things (IoT): A vision, architectural elements, and future directions“. In: *Future Generation Computer Systems* 29.7 (2013). Including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services & Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond, S. 1645–1660. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2013.01.010. URL: http://www.sciencedirect.com/science/article/pii/S0167739X13000241 (zitiert auf S. 15, 18).

[GRL+08] L. Gurgen, C. Roncancio, C. Labbé, A. Bottaro, V. Olive. „SStreaMWare: A Service Oriented Middleware for Heterogeneous Sensor Data Management“. In: *Proceedings of the 5th International Conference on Pervasive Services*. ICPS '08. Sorrento, Italy: ACM, 2008, S. 121–130. ISBN: 978-1-60558-135-4. DOI: 10.1145/1387269.1387290. URL: http://doi.acm.org/10.1145/1387269.1387290 (zitiert auf S. 25).

[HBS+16] P. Hirmer, U. Breitenbücher, A. C. F. da Silva, K. Kepes, B. Mitschang, M. Wieland. „Automating the Provisioning and Configuration of Devices in the Internet of Things“. In: *CSIMQ* 9 (2016), S. 28–43 (zitiert auf S. 15, 20, 27, 36).

[HCH+03] J. Humble, A. Crabtree, T. Hemmings, K.-P. Åkesson, B. Koleva, T. Rodden, P. Hansson. „“Playing with the Bits” User-Configuration of Ubiquitous Domestic Environments“. In: Bd. 2864. Okt. 2003, S. 256–263. DOI: 10.1007/978-3-540-39653-6_20 (zitiert auf S. 23).

[IHR+12] I. Ishaq, J. Hoebeke, J. Rossey, E. De Poorter, I. Moerman, P. Demeester. „Facilitating Sensor Deployment, Discovery and Resource Access Using Embedded Web Services“. In: *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. Juli 2012, S. 717–724. DOI: 10.1109/IMIS.2012.48 (zitiert auf S. 25).

[KWH13] H. Kagermann, W. Wahlster, J. Helbig. „Securing the future of German manufacturing industry: Recommendations for implementing the strategic initiative INDUSTRIE 4.0“. In: *Final report of the Industrie* 4.0 (2013) (zitiert auf S. 18).

[MIV+14] S. Mayer, N. Inhelder, R. Verborgh, R. Van de Walle, F. Mattern. „Configuration of smart environments made simple: Combining visual modeling with semantic metadata and reasoning“. In: *2014 International Conference on the Internet of Things (IOT)*. Okt. 2014, S. 61–66. DOI: 10.1109/IOT.2014.7030116 (zitiert auf S. 23).

[MIVV14] S. Mayer, N. Inhelder, R. Verborgh, R. Van de Wallet. „User-friendly configuration of smart environments“. In: *2014 IEEE International Conference on Pervasive Computing and Communication Workshops (PERCOM WORKSHOPS)*. März 2014, S. 163–165. DOI: 10.1109/PerComW.2014.6815188 (zitiert auf S. 23).

[MMST16] J. Mineraud, O. Mazhelis, X. Su, S. Tarkoma. „A gap analysis of Internet-of-Things platforms“. In: *Computer Communications* 89-90 (2016). Internet of Things Research challenges and Solutions, S. 5–16. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2016.03.015. URL: http://www.sciencedirect.com/science/article/pii/S0140366416300731 (zitiert auf S. 15, 19).

[MNH+13] H. McDonald, C. Nugent, J. Hallberg, D. Finlay, G. Moore, K. Synnes. „The homeML suite: shareable datasets for smart home environments“. In: *Health and Technology* 3.2 (Juni 2013), S. 177–193. ISSN: 2190-7196. DOI: 10.1007/s12553-013-0046-7. URL: https://doi.org/10.1007/s12553-013-0046-7 (zitiert auf S. 23).

[NFD+07] C. D. Nugent, D. D. Finlay, R. J. Davies, H. Y. Wang, H. Zheng, J. Hallberg, K. Synnes, M. D. Mulvenna. „homeML – An Open Standard for the Exchange of Data Within Smart Environments“. In: *Pervasive Computing for Quality of Life Enhancement*. Hrsg. von T. Okadome, T. Yamazaki, M. Makhtari. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, S. 121–129. ISBN: 978-3-540-73035-4 (zitiert auf S. 23, 34).

[OAS13] OASIS. *Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0.* 2013 (zitiert auf S. 24).

[RGW+13] M. Rietzler, J. Greim, M. Walch, F. Schaub, B. Wiedersheim, M. Weber. „homeBLOX: Introducing Process-driven Home Automation“. In: *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*. UbiComp '13 Adjunct. Zurich, Switzerland: ACM, 2013, S. 801–808. ISBN: 978-1-4503-2215-7. DOI: 10.1145/2494091.2497321. URL: http://doi.acm.org/10.1145/2494091.2497321 (zitiert auf S. 24).

[SBH+17] A. C. F. da Silva, U. Breitenbücher, P. Hirmer, K. Képes, O. Kopp, F. Leymann, B. Mitschang, R. Steinke. „Internet of Things Out of the Box Using TOSCA for Automating the Deployment of IoT Environments“. Englisch. In: *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER)*. Hrsg. von D. Ferguson, V. M. Mu noz, J. Cardoso, M. Helfert, C. Pahl. Bd. 1. ScitePress. SciTePress Digital Library, Juni 2017, S. 358–367. ISBN: 978-989-758-243-1. DOI: 10.5220000624330358036 7. URL: httpwww2.informatik.uni-stuttgart.decgi-binNCSTRLNCSTRL_view.plid=INPROC-2017-28&engl=0 (zitiert auf S. 24).

[SBK+16] A. C. F. da Silva, U. Breitenbücher, K. Képes, O. Kopp, F. Leymann. „OpenTOSCA for IoT: Automating the Deployment of IoT Applications Based on the Mosquitto Message Broker“. In: *Proceedings of the 6th International Conference on the Internet of Things*. IoT'16. Stuttgart, Germany: ACM, 2016, S. 181–182. ISBN: 978-1-4503-4814-0. DOI: 10.1145/2991561.2998464. URL: http://doi.acm.org/10.1145/2991561.2998464 (zitiert auf S. 24).

[SK17] K. J. Singh, D. S. Kapoor. „Create Your Own Internet of Things: A survey of IoT platforms.“ In: *IEEE Consumer Electronics Magazine* 6.2 (Apr. 2017), S. 57–68. ISSN: 2162-2248. DOI: 10.1109/MCE.2016.2640718 (zitiert auf S. 15, 19).

[SU06] L. Satpathy, M. S. University. *Smart Housing: Technology to Aid Aging in Place. New Opportunities and Challenges*. Mississippi State University, 2006. ISBN: 9781109798623 (zitiert auf S. 18).

[VF13] O. Vermesan, P. Friess, Hrsg. *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*. River Publishers Series in Communication. Aalborg: River, 2013. ISBN: 978-87-92982-73-5. URL: http://www.internet-of-things-research.eu/pdf/Converging_Technologies_for_Smart_Environments_and_Integrated_Ecosystems_IERC_Book_Open_Access_2013.pdf (zitiert auf S. 15, 17, 18).

[VSID16] M. Vögler, J. M. Schleicher, C. Inzinger, S. Dustdar. „A Scalable Framework for Provisioning Large-Scale IoT Deployments“. In: *ACM Trans. Internet Technol.* 16.2 (März 2016), 11:1–11:20. ISSN: 1533-5399. DOI: 10.1145/2850416. URL: http://doi.acm.org/10.1145/2850416 (zitiert auf S. 24).

[Wei93] M. Weiser. „Hot topics-ubiquitous computing“. In: *Computer* 26.10 (Okt. 1993), S. 71–72. ISSN: 0018-9162. DOI: 10.1109/2.237456 (zitiert auf S. 17).

[WSCL13] E. Westkämper, D. Spath, C. Constantinescu, J. Lentes. *Digitale Produktion*. SpringerLink : Bücher. Springer Berlin Heidelberg, 2013. ISBN: 9783642202599 (zitiert auf S. 18).

Alle URLs wurden zuletzt am 08. 05. 2019 geprüft.

Erklärung

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

Ort, Datum, Unterschrift