Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Monitoringsystem für modellgetriebene IoT-Anwendungen

Arne Bartenbach

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Betreuer/in:	Daniel Del Gaudio, M.Sc.
Beginn am:	15. Januar 2022
Beendet am:	15. Juli 2022

Gendererklärung

Um eine bessere Lesbarkeit zu gewährleisten werden in dieser Bachelorarbeit nicht die unterschiedlichen geschlechterspezifischen Sprachformen gleichzeitig verwendet. Dabei soll kein Geschlecht hervorgehoben werden, sondern alle Formulierungen gleichermaßen für alle Geschlechter gelten.

Kurzfassung

Das Internet der Dinge (engl. Internet of Things, kurz IoT) gewinnt immer mehr an Bedeutung, da neben bekannten Bereichen wie Smart-Homes auch Smart-Cities, Smart-Factories, Smart-Health und weitere IoT-Anwendungen entstehen. IoT-Umgebungen und deren Anwendungen werden immer komplexer, da die Anzahl an Geräten und verschiedenen Anwendungsfällen steigt. Durch den modellgetriebenen Ansatz zur Anwendungsentwicklung lassen sich Anwendungen leicht auf verschiedene Umgebungen übertragen und können einfach an neue Anforderungen angepasst werden. Dies ist durch Modellierungstools, wie das IoT-Application-Modeling-Tool (IAMT) vom IPVS der Universität Stuttgart, möglich. Die in IoT-Anwendungen entstehenden Fehler, wie Überlastung oder Ausfall von Geräten und falschen Sensorwerten, müssen erkannt werden, um die Funktionalität und Zuverlässigkeit der IoT-Anwendungen zu gewährleisten. Für modellgetriebene IoT-Anwendungen ist dabei eine ebenfalls modellgetriebene Lösung zur Fehlererkennung notwendig. Es werden Monitoringfunktionen zu den Anwendungsmodellen hinzugefügt. Dabei sollen neben dem Monitoring für technische Eigenschaften der Geräte auch operationale Fehler erkannt werden. Operationale Fehler sind Fehlverhalten, die beispielsweise durch falsche Sensorwerte entstehen. In dieser Arbeit wird ein Konzept für ein Monitoringsystem entwickelt, das prototypisch umgesetzt wird und in das IAMT integriert wird.

Inhaltsverzeichnis

1. Einleitung **13**
- 1.1. Motivation . . . 14
- 1.2. Ziele der Bachelorarbeit . . . 15
- 1.3. Gliederung . . . 17

2. Grundlagen Anwendungsmodellierung und Monitoring **19**
- 2.1. Anforderungen an IoT-Anwendungen . . . 19
- 2.2. Modellgetriebene IoT-Anwendungen . . . 20
- 2.3. Monitoring für IoT-Anwendungen . . . 22
- 2.4. Modellierungstool zur Erweiterung um eine Monitoringfunktion . . . 23

3. Verwandte Arbeiten **25**

4. Konzept für ein Monitoringsystem für modellgetriebene IoT-Anwendungen **29**
- 4.1. Anforderungen an das Monitoringsystem . . . 29
- 4.2. Fehlererkennung . . . 30
- 4.3. Ablauf von Sensor- und Monitoringdatenerhebung . . . 34
- 4.4. Architektur . . . 40

5. Implementierung **49**
- 5.1. Modellierungstool . . . 49
- 5.2. IoT-Runtime-System - Multi-purpose Binding and Provisioning Platform . . . 57
- 5.3. Monitoring - InfluxDB2.x . . . 57
- 5.4. Kontextdatenbank - Apache Jena Fuseki . . . 58
- 5.5. Evaluierung der Anforderungen an das Monitoringsystem . . . 61

6. Zusammenfassung und Ausblick **65**

A. Anhang **69**

Literaturverzeichnis **71**

Abbildungsverzeichnis

1.1. Heterogene Geräte in einem Smart Home 14

2.1. Modellgetriebene Anwendungsentwicklung - aus „Towards Feedback Loops in Model-Driven IoT Applications“ [DH21] 21

4.1. Modellierung & Fehlererkennung durch den Benutzer 34
4.2. Ablauf modellgetriebene IoT-Anwendungmodellierung mit Monitoringfunktion - Erweiterung aus „Towards Feedback Loops in Model-Driven IoT Applications“ [DH21] 35
4.3. Ablauf der Anwendung eines IoT-Modells 36
4.4. Datenübertragung mit Kontextdatenbank 39
4.5. Architektur zur Integration des Modellierungstools 40
4.6. Schematisches Beispiel für ein Kontextmodell 42
4.7. Exemplarischer Ablaub Modellerstellung und Monitoring 44

5.1. Anwendungsmodell - Erstellung 50
5.2. Deploymentmodell - Initialisierung 51
5.3. Monitoringmodell - Monitoring konfigurieren 53
5.4. Regelerstellung 54
5.5. Monitoring Übersicht 56

A.1. Modellierungstool gesamtes Fenster Übersicht 69

Verzeichnis der Listings

5.1. SPARQL Query - Aktuelle Daten filtern 55
5.2. SPARQL Query - MBP-Daten auslesen 59
5.3. SPARQL Query - Monitoringdaten auslesen 60
5.4. SPARQL Query - Sensordaten auslesen 61

1. Einleitung

Die steigende Anzahl und Komplexität von IoT-Umgebungen in der heutigen Zeit bringt neue Anforderungen und Schwierigkeiten mit sich [Mol21]. Für IoT-Umgebungen gibt es viele Anwendungsbereiche, wie Smart Homes [CKLP21], Smart Cities [Yıl21] oder Smart Factories [KRA22]. Diese IoT-Umgebungen bestehen aus einer Vielzahl unterschiedlicher Geräte, die verschiedene Aufgaben übernehmen. Ebenfalls unterscheiden sich die Geräte durch unterschiedliche Kommunikationsschnittstellen, um sowohl zwischen IoT-Gerät und Server als auch zwischen den Geräten direkt Informationen auszutauschen. Da im IoT viele kleine Geräte verwendet werden, müssen Programme zur Verarbeitung oder zum Auslesen von Daten, Kommunikationsprotokolle oder auch weitere Softwarekomponenten an die Eigenschaften der Geräte angepasst werden. Dafür sind für das IoT angepasste Konzepte notwendig, um beispielsweise bestehende Ressourcen effizient auszunutzen, aber dabei Geräte nicht zu überlasten [DH20a]. Solche Systeme sind fehleranfällig und benötigen deshalb Überwachung und Struktur, um die geforderte Funktionalität zu gewährleisten [Xin20]. Beim modellgetriebenen Ansatz für IoT-Anwendungen wird in der Regel ein passendes Modell der Anwendung verwendet oder muss erst entworfen werden [FH20]. Modellgetriebene IoT-Anwendungen erleichtern das Erstellen, Erweitern und Überwachen der IoT-Umgebung. Benutzer können durch Modelle IoT-Anwendungen leicht realisieren. Hierfür werden Programme benötigt, die einem Benutzer die Modellierung ermöglichen. Dies kann durch grafische Anwendungen, wie beispielsweise FlexMash [HB17], die minimale Systemkenntnisse erfordern, umgesetzt werden, sodass eine leichte Bedienung ermöglicht wird und mehr Benutzer dadurch die Anwendung verwenden können. Dadurch werden die erstellten Modelle anschaulich und nachvollziehbar dem Benutzer präsentiert. Um fehleranfällige IoT-Anwendungen kontrollieren zu können, erfordert es ständige Überwachung, so dass Fehler schnell erkannt und behandelt werden. Fehler, wie beispielsweise ausfallende Geräte, können leicht entstehen und müssen bemerkt werden, um das System intakt zu halten. Zudem kommen verschiedene Arten an Fehlern vor. Fehlerhafte Sensorwerte, deren Erkennung nicht immer eindeutig ist, sind ein Beispiel dafür. Diese Fehler gehören zu der Kategorie der operationalen Fehler, welche von vielen bestehenden Monitoringsystemen bisher nicht erkannt werden können [DH21]. Um trotzdem eine Fehlererkennung umzusetzen, werden Kriterien und Verfahren zur Erkennung benötigt. Monitoring ist eine schwierige Aufgabe im IoT, da es viele nicht vom Benutzer kontrollierbare Einflüsse auf die Umgebung und daher auch auf die Geräte gibt. Hierfür werden übersichtliche Darstellungen des aktuellen Systemkontexts benötigt, um die Fehlersuche zu ermöglichen. Der Benutzer soll die Möglichkeit haben, Modelle zu erstellen und in diesem die Fehlerfälle der Komponenten zu definieren, welche für ihn zur Laufzeit automatisch überprüft werden. Diese Arbeit

beschreibt, wie eine Anwendung zur Erstellung von IoT-Modellen mit Monitoringfunktion realisiert wird.

1.1. Motivation

Für IoT-Anwendungen, die Anforderungen an Strukturierung und Überwachung haben, existieren viele Anwendungsgebiete. Eine Struktur und Überwachung hilft den Anwendungen langfristig stabil zu laufen. Um Sicherheiten in IoT-Anwendungen zu garantieren, müssen individuelle Kriterien an jede Komponente gestellt werden. Smart Homes sind ein Beispiel, welches zeigt, dass dieses Szenario eine große Anzahl an individuellen Systemen betrifft, die jedem im alltäglichen Leben begegnen können. In einem Smart Home gehören viele verschiedene IoT-Geräte zur Umgebung, die vernetzt arbeiten sollen, jedoch individuelle Anforderungen mit sich bringen.

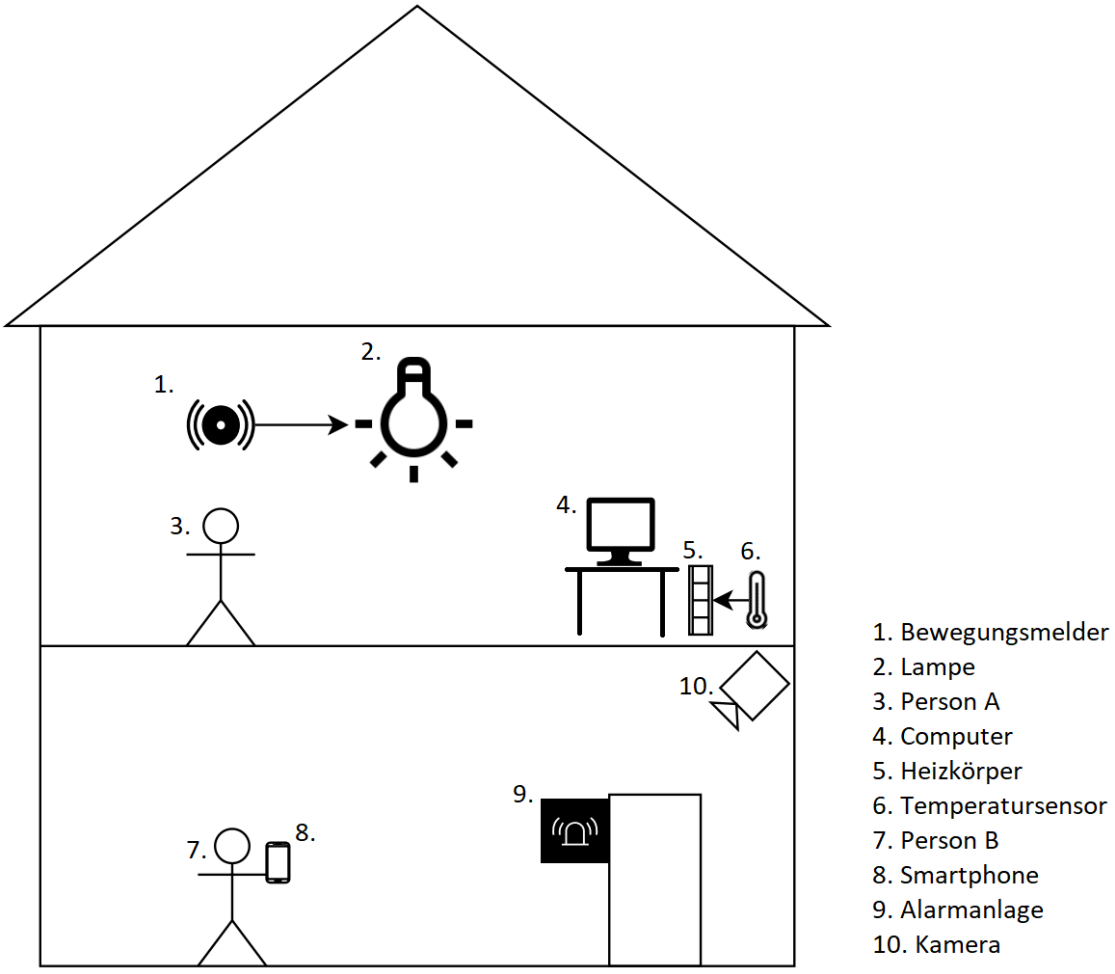

Abbildung 1.1.: Heterogene Geräte in einem Smart Home

Abbildung 1.1 zeigt ein Smart Home mit unterschiedlichen Geräten, die individuelle Kriterien erfüllen müssen, um zu funktionieren. Der Bewegungsmelder erkennt Person A und schaltet die Lampe an. Die Heizung wird mithilfe eines Temperatursensors gesteuert. Der Computer und das Smartphone können verwendet werden, um das Smart Home zu konfigurieren oder manuelle Steuerung zu übernehmen. Die Alarmanlage und Überwachungskamera dienen der Sicherheit, wodurch man diesen Geräten vertrauen können muss. Eine Alarmanlage benötigt daher eine höhere Garantie an Zuverlässigkeit, als ein Bewegungsmelder, welcher für eine Person das Licht einschaltet, wenn diese einen Raum betritt. Mobile Geräte, wie Smartphones, verlassen das Netzwerk regelmäßig und sind daher nicht immer erreichbar, auch wenn die Geräte selbst funktionsfähig sind. Andere Geräte weisen möglicherweise Fehler auf, wenn sie zum Beispiel veraltet oder von schlechter Qualität sind. Fehler sind hierbei auf unterschiedliche Weise zu erkennen. Gerätefehler, die zu einem Absturz der Komponente führen, die damit nicht mehr erreichbar ist, lassen sich eindeutiger erkennen als sogenannte operationale Fehler. Operationale Fehler lassen sich in der Regel erst auf semantischer Ebene wahrnehmen. Am Beispiel eines Temperatursensors würde das System im Fall eines nicht mehr verfügbaren Gerätes keine Werte mehr empfangen, aber im Fall eines operationalen Fehlers würden weiterhin Werte übermittelt, welche jedoch nicht der Wahrheit entsprächen. Dies kann auf verschiedene Ursachen, wie zum Beispiel einer Überlastung der Komponente, zurückzuführen sein. Diese verschiedenen Fehlertypen müssen erkannt werden. Eine Schwierigkeit dabei ist, dass die Geräte zu unterschiedlich sind und daher oft ein einziges Programm nicht ausreicht, um alle Fehlertypen für jedes Gerät zu erkennen. Verschiedene Smartphones, Computer, Überwachungskameras, Heizungssteuerungen, usw. haben unterschiedliche Betriebssysteme, Rechenleistungen oder auch Anforderungen an den Grad der Überwachung. Daher muss Monitoring individuell konfigurierbar sein, um diesen unterschiedlichen Geräten, Fehlern und Anforderungen gerecht zu werden.

1.2. Ziele der Bachelorarbeit

Diese Bachelorarbeit hat das Ziel, das Monitoring für IoT-Entwickler durch ein Modellierungstool zu ermöglichen. Folgende Teilaufgaben sind der Kern der Arbeit.

- Ziel 1: Monitoring für technische Eigenschaften von IoT-Geräten
- Ziel 2: Erkennung operationaler Fehler
- Ziel 3: Integration in bestehende Modellierungsumgebung

In den folgenden Abschnitten werden die einzelnen Ziele genauer erklärt.

1.2.1. Ziel 1: Monitoring für technische Eigenschaften von IoT-Geräten

Technische Eigenschaften können durch verschiedene, bereits vorhandene, Tools gemessen oder ausgelesen werden. Hierzu gehören Eigenschaften wie CPU-Auslastung oder freier Speicher. Da nicht jedes Monitoringprogramm für alle Gerätetypen zur Verfügung steht oder nicht alle gesuchten Eigenschaften abbilden kann, muss garantiert werden, dass Messdaten in einheitlicher Form bereitgestellt werden. Wenn dies nicht garantiert ist, muss die Monitoringkomponente des Modellierungstools für jede neue Monitoringsoftware erweitert werden, was einen großen Aufwand darstellt. Aus diesem Grund sollte das Monitoring flexibel sein, so dass der Benutzer verschiedene Monitoringanwendungen verwenden kann, um zwischen diesen wechseln zu können. Ebenfalls können, für unterschiedliche Bereiche, verschiedene dieser Anwendungen zur Verfügung stehen. Durch eine modellgetriebene Monitoringlösung soll die Flexibilität gewährleistet werden, da das Modell losgelöst von der spezifischen Umsetzung in einer IoT-Umgebung ist und damit leicht angepasst werden kann. Das Monitoringmodell baut dabei auf das Anwendungsmodell eines Systems auf und ist dabei wie auch dieses Modell einfach auf neue Umgebungen übertragbar [DH21]. Somit soll eine flexible Messung technischer Daten der Geräte ermöglicht werden. Da die Messwerte nicht manuell analysiert werden sollen, wird zuvor vom Benutzer definiert, wann er gewarnt werden möchte, um Fehler erkennen oder vorbeugen zu können. Die daraus resultierenden Ergebnisse und möglichen Warnungen sollen dem Benutzer in Echtzeit angezeigt werden.

1.2.2. Ziel 2: Erkennung operationaler Fehler

Operationale Fehlverhalten in der IoT-Anwendung entstehen beispielsweise durch Messfehler, welche teils eindeutig, jedoch oft nicht klar zu bestimmen sind. Als Beispiel kann ein negativer Messwert eines Ultraschallsensors bereits durch eine Beschränkung des Wertebereichs auf nur positive Zahlen abgefangen werden, da ein negativer Abstand nie eintreten kann. Wenn allerdings bei einem Temperatursensor fälschlicherweise Sprünge in der Messreihe wahrgenommen werden, ohne eine tatsächliche Änderung der Umgebungstemperatur, kann dieser Fehler schwer bestimmt werden. Dies liegt daran, dass die Werte trotzdem in einem validen Wertebereich liegen können, ohne dabei der Wahrheit zu entsprechen. Daher soll der Benutzer die Option haben, individuell für jede Komponente eigene Regeln zu definieren. Diese Regeln sollen, wie auch bei den technischen Eigenschaften, in Echtzeit für jedes Update überprüft werden. Dass die Regelerstellung individuell für jede Komponente konfigurierbar sein soll, ist wichtig, da unterschiedliche Geräte unterschiedliche Messungen durchführen. Ebenfalls müssen für die Komponenten unterschiedliche Sicherheiten gegeben werden, ab wann ein Fehler erkannt werden soll, da dies nicht in jedem Fall eindeutig ist.

1.2.3. Ziel 3: Integration in bestehende Modellierungsumgebung

Die Integration der Monitoringkomponente ist erforderlich, da diese auf der Modellierung der IoT-Umgebung basiert und damit dieses Modell erweitert werden soll. Hierbei soll die bestehende Anwendung nicht eingeschränkt werden, da ein IoT-Modell auch ohne Monitoring die geforderte Funktionalität bereitstellen kann. Das Monitoring soll dem Benutzer als Option angeboten werden. Die Ausgabewerte der verschiedenen Monitoringsoftwares sollen in einem einheitlichen Stil direkt in dem vom Benutzer zuvor erstellten Modell angezeigt werden. Dies erleichtert die Zuordnung, und kann die Fehlersuche beschleunigen und vereinfachen. Bisher wird in dem Modellierungstool IoT-Application-Modelling-Tool (IAMT) von dem IPVS der Universität Stuttgart die automatisierte Installation und Ausführung der Software auf den einzelnen IoT-Geräten von einem separaten Konfigurator [Rei19] übernommen. Dieses Programm muss ebenfalls für Monitoringsoftware erweitert werden. Zusätzlich sind weitere Korrekturen zur Kompatibilität mit der aktuellen Version des verwendeten IoT-Runtime-Systems notwendig. Daher wird die Funktionalität zur Installation und Ausführung von sowohl Anwendungssoftware als auch Monitoringsoftware in das Modellierungstool integriert. Damit soll das bestehende Modellierungstool so erweitert sein, dass aufbauend auf das Anwendungsmodell ein Monitoringmodell für das IoT-System erstellt und realisiert werden kann.

1.3. Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Grundlagen Anwendungsmodellierung und Monitoring: Zuerst werden die Grundlagen zu modellgetriebenen IoT-Anwendungen beschrieben und darauf folgend wird genauer auf Monitoringansätze für das IoT eingegangen.

Kapitel 3 – Verwandte Arbeiten: Es werden verwandte Arbeiten vorgestellt, welche grundlegende oder weiterführende Konzepte beschreiben, die ebenfalls in dieser Umsetzung zu finden sind oder noch in Zukunft angewendet werden können. Dabei werden die Unterschiede abgegrenzt und beschrieben, welche Anpassungen und Verbesserungen vorgenommen wurden.

Kapitel 4 – Konzept für ein Monitoringsystem für modellgetriebene IoT-Anwendungen: Das Konzept umfasst alle Pläne zur Architektur des Gesamtsystems und der Struktur der entwickelten Modellierungsanwendung.

Kapitel 5 – Implementierung: Hier wird die explizite Umsetzung präsentiert und erläutert, wie die Pläne aus dem Konzept prototypisch realisiert wurden. Abschließend werden die Anforderungen, die im Konzept gestellt werden, evaluiert.

Kapitel 6 – Zusammenfassung und Ausblick Schließlich wird die Arbeit zusammengefasst und Ausblicke zur Weiterentwicklung und Forschung aufgezeigt.

2. Grundlagen Anwendungsmodellierung und Monitoring

In diesem Kapitel werden die Grundlagen für das entworfene Konzept dieser Arbeit beschrieben. Zuerst werden in Abschnitt 2.1 allgemeine Anforderungen und Schwierigkeiten für IoT-Anwendungen erläutert. In Abschnitt 2.2 wird das Konzept für modellgetriebene Anwendungsentwicklung für das IoT dargestellt, da dieses Konzept verwendet wird, um Monitoringmodelle ebenfalls in die Anwendung zu integrieren. Dafür wird in Abschnitt 2.3 das Monitoring für die erstellten IoT-Anwendungen vorgestellt. Dabei werden Unterschiede zu klassischen Monitoringlösungen anhand der Schwierigkeiten für das Monitoring im IoT beschrieben. Der darauf folgende Abschnitt 2.4 stellt das Modellierungstool IAMT vor, welches im Rahmen dieser Arbeit um die Monitoringfunktion erweitert werden soll.

2.1. Anforderungen an IoT-Anwendungen

In *Internet of things: converging technologies for smart environments and integrated ecosystems.* [VF13] werden umfassende Konzepte und Anforderungen für vielseitige IoT-Umgebungen präsentiert. Anhand vieler verschiedener Anwendungsszenarien werden diese Konzepte so weiterentwickelt, dass größere und komplexere IoT-Umgebungen gebaut werden können. Jedes Smart Home bringt seine individuellen Anforderungen mit sich und stellt damit eine eigene IoT-Umgebung dar. Allerdings kann die Idee hinter einem Smart Home noch größer skaliert werden, um Städte zu Smart Cities weiterzuentwickeln oder letztendlich auch diese zu erweitern. Hier werden Ansprüche an den Energieverbrauch und die Verteilung steigen. Auch die Datenverarbeitung bringt neue Schwierigkeiten mit sich, da Systeme weiter ausgedehnt werden und damit dezentrale Konzepte eingeführt werden müssen. Weitere Bereiche, wie die Medizin oder die Landwirtschaft bieten wiederum neue Anwendungsgebiete, die eigene Anforderungen definieren. Somit wird deutlich, dass sich hinter der Idee des IoTs eine ganze Welt an Anwendungsfällen öffnet, die eine Vernetzung unterschiedlicher Bereiche auslöst. Zunächst bringt dies viele Vorteile mit sich, da durch das Monitoring der gesamten Umgebung im Alltag neue Erkenntnisse gewonnen werden und Optimierungen in allen Bereichen vorgenommen werden können. Um diese Vision umzusetzen, werden viele auch sensible Daten produziert. Da diese in einer stark vernetzen Welt schnell erreichbar sind, spielt Sicherheit im IoT eine

zentrale und erhebliche Rolle. Persönliche Daten müssen geschützt werden, aber trotzdem für berechtigte Zwecke im IoT verfügbar gemacht werden, da durch eine höhere Bereitstellung an Daten eine höhere Qualität und Aussagekraft der IoT-Umgebung erzielt werden kann. Dies soll jedoch nicht bedeuten, dass sensible Daten öffentlich gemacht und Sicherheitsanforderungen nicht mehr erfüllt werden.
Durch die vielen unterschiedlichen Anwendungsbereiche, Sicherheitsanforderungen und Komplexität der Systeme werden strukturierte Ansätze zur Erstellung von IoT-Anwendungen benötigt. Dies kann durch modellgetriebene Anwendungsentwicklung umgesetzt werden.

2.2. Modellgetriebene IoT-Anwendungen

Modellgetriebene IoT-Anwendungen zu entwickeln, beschreibt einen Ansatz, der die Entwicklung auf eine höhere Abstraktionsebene anhebt. Dadurch werden komplexe, heterogene Systeme aus vielen unterschiedlichen Gerätetypen strukturiert und leichter nachvollziehbar beschrieben. Dabei kann das Konzept für modellgetriebene Anwendungsentwicklung, das in „Towards Feedback Loops in Model-Driven IoT Applications“ [DH21] vorgestellt wird, angewendet werden. Dort wird der Modellierungsprozess zuerst in eine Umgebungsmodellierung, danach eine Anwendungsmodellierung aufgeteilt, wodurch durch Model-Mapping ein Deployment-Modell erstellt wird. Dieses Deployment-Modell wird schließlich für weitere Konfiguration, Installation und Ausführung des Modells angewendet. Das Modell wird dabei automatisch in ein ausführbares System in der IoT-Umgebung übersetzt [NTBG15]. Ein Modell beschreibt die für den Benutzer wichtigen Informationen, so dass daraus eindeutig auf das dargestellte System geschlossen werden kann. Um dies zu ermöglichen, können Modelle von IoT-Umgebungen sowohl verwendete Hardware, als auch Informationen über oder von der Software beinhalten. Die Struktur der Anwendung bezieht sich im Wesentlichen auf die verwendeten Geräte und die dort jeweils verfügbaren Hardwarekomponenten wie Sensoren oder Aktoren. Zu den Geräten können ebenfalls die zu installierenden Softwarekomponenten zugeordnet werden, da auch diese in der Regel nur durch manuelle strukturelle Änderungen durch einen Benutzer vorgenommen werden. Nachdem das Modell erstellt wurde, soll es nicht nur als Repräsentation für den Benutzer dienen, sondern auch die Realisierung des modellierten Systems übernehmen. Dafür müssen alle Programme auf den Geräten installiert und schließlich ausgeführt werden. Nach dem Start der Programme werden zur Laufzeit des Systems große Mengen an neuen Daten, wie beispielsweise Messdaten von Sensoren oder Monitoringdaten, produziert. Der aktuelle Zustand des Systems wird durch Echtzeitdaten beschrieben und kann sich daher ständig ändern. Änderungen werden dabei durch Sensoren detektiert und wenn diese veränderte Messwerte wahrnehmen, liegt ein anderer Zustand vor und andere Reaktionen durch Aktoren können notwendig sein. Dies kann an dem Beispiel eines Thermostats verdeutlicht werden. Der Sensor des Thermostats nimmt eine zu geringe Temperatur wahr, wodurch der Zustand des Systems sich geändert hat und als „zu kalt“ beschrieben werden kann. Daraufhin reagiert das System, indem die Heizungssteuerung die Heizung stärker heizen lässt. Dies kann ständig passieren und Korrekturen des Systems werden

ebenfalls ständig ausgelöst. Die Kombination aus Struktur und Zustand wird als der Kontext des Systems bezeichnet. Die komplexen Daten über den Kontext des Systems werden durch das Modell übersichtlich dargestellt und sollen es dem Benutzer ermöglichen, die Anwendung durch das Modell verstehen, erstellen und verändern zu können.

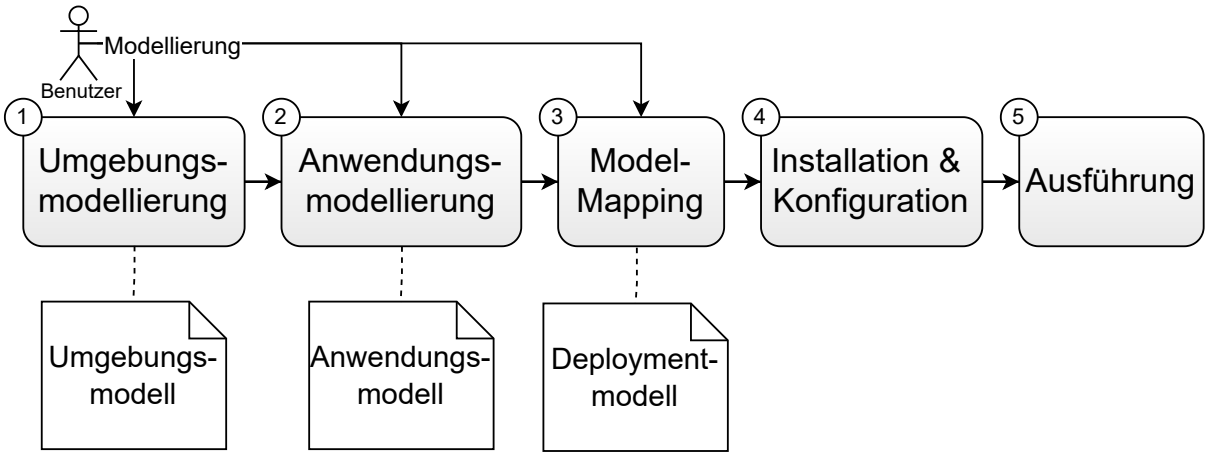

Abbildung 2.1.: Modellgetriebene Anwendungsentwicklung - aus „Towards Feedback Loops in Model-Driven IoT Applications“ [DH21]

Die Entwicklung der Umgebungsmodelle und Anwendungsmodelle, um IoT-Systeme zu beschreiben, muss komplexe Systeme abstrahieren und auf die für die Entwicklung wichtigen Informationen reduzieren. Durch modellgetriebene IoT-Awendungsentwicklung kann dabei das Modell losgelöst von der spezifischen IoT-Umgebung erstellt werden und kann dadurch einfach für weitere IoT-Umgebungen angepasst und eingesetzt werden [HBS+16]. Diese Modellierung von IoT-Umgebungen umfasst kleine Geräte, welche Limitierungen an Rechenleistung oder Speicher haben. Gleichzeitig umfasst die Modellierung Geräte wie Computer oder Smartphones mit eigenen Funktionalitäten und Limitierungen. Da Geräte in einer IoT-Umgebung direkt zusammenhängen können, müssen diese Verbindungen ebenfalls modelliert werden. Abhängigkeiten können als Signal zum Erkennen kritischer systemrelevanter Punkte im Modell genutzt werden [CCD+17]. Abhängigkeiten können beispielsweise entstehen, wenn ein Gerät keine direkte Verbindung zum Server hat, oder zu einer anderen Komponente, die für die Datenverarbeitung verwendet wird. Dann besteht eine Abhängigkeit zu dem Gerät, welches für die Weitergabe der Daten an den Server zuständig ist. Einfach sind dabei Abhängigkeiten zum Beispiel zwischen einem Sensor und Aktor zu erkennen, denn wenn die Messungen des Sensors Fehler aufweisen oder nicht verfügbar sind, kann der Aktor keine korrekte Funktion gewährleisten. Dies kann bei einem Thermostat eintreten, wenn der Sensor zu niedrige Temperaturen misst, reagiert die Heizungssteuerung mit stärkerem Heizen, was in dem Szenario nicht gewollt ist. Da viele individuelle Fehler und zusätzlich die durch Abhängigkeiten resultierende Fehler erkannt werden müssen, wird Monitoring in die modellgetriebene IoT-Anwendungsentwicklung eingeführt.

2.3. Monitoring für IoT-Anwendungen

IoT-Anwendungen haben eine hohe Fehleranfälligkeit, die auf vielerlei Ursachen zurückzuführen sind. Typische Fehler entstehen durch die Verwendung von sehr kleinen Geräten, die im Gegensatz zu Servern nur geringe Robustheit aufweisen. Ebenfalls entstehen Netzwerkfehler durch die Verteilung des Systems auf eine große Anzahl an Geräten. Neben den genannten Fehlerursachen sollen auch noch weitere erkannt werden können, um sicherzustellen, dass einzelne Fehler nicht das System als Ganzes beeinflussen. Da nicht alle Fehler vermeidbar sind, kann im Fall eines erkannten Fehlers immer die Qualität und Aussagekraft der Anwendung für den Zeitraum des Fehlers neu bewertet werden. Dass die Bewertung von Monitoringeigenschaften nicht trivial für jedes System ist, ist schon anhand einfacher Szenarien zu verstehen. Wenn beispielsweise mobile Geräte zu einer IoT-Umgebung gehören, können diese Geräte aus dem Netzwerk entfernt werden und somit nur begrenzte Zeit zur Verfügung stehen. Falls ein solches Gerät aus dem Netzwerk entfernt wird, ist es nicht eindeutig, wie das System damit umgehen soll. Aus der Monitoringsicht antwortet ein Gerät nicht mehr. Ob dies jedoch auf einen Fehler schließen lässt und das Gerät kaputt ist oder ob es nur wie im Beispiel temporär bewusst aus dem Netzwerk entfernt wurde, stellt schon eine Schwierigkeit für die Fehlersuche dar. Die Verwendung von mobilen Geräten in IoT-Umgebungen und damit auch die Probleme der Verfügbarkeit werden in „Challenges of Using Edge Devices in IoT Computation Grids“ [DMPP13] adressiert. Dort wird ein Konzept beschrieben, wie Rechenleistung von mobilen Geräten in IoT-Umgebungen genutzt werden kann und somit das Gerät als Ressource für die Umgebung zur Verfügung steht. Um besser mit Problemen der Fehlererkennung und damit auch der Fehlersuche umzugehen, gibt es verschiedene Ansätze, dies zu bewerkstelligen [FY20] [WSZ+16]. Monitoringlösungen sind bereits weit verbreitet und einige Eigenschaften der Geräte können oft direkt ausgelesen werden, ohne eigene Messungen zu starten. Allerdings wird diese Art von Monitoring meist nur auf das eigene Gerät angewendet und nicht auf viele eigenständig operierende IoT-Geräte. Das Monitoring einzelner Komponenten muss zusammengeführt werden, da sonst viele verschiedenartige Daten verteilt auf Geräten manuell gesammelt und analysiert werden. In „Cloud monitoring: A survey“ [ABdP13] werden einige verschiedene Optionen für eine Cloud-Monitoring-Lösung präsentiert und verglichen. Cloud-Monitoring ermöglicht die Datenverarbeitung der Monitoringdaten in der Cloud, wodurch die Daten schnell verarbeitet werden und für alle verbundenen Komponenten leicht verfügbar sind. Dabei ist zu erkennen, dass hierbei mit mehreren verschiedenen Geräten in dem Netzwerk gearbeitet wird und die gesammelten Daten automatisch zusammengeführt werden. Um Cloud-Monitoring in der eigenen IoT-Umgebung zu realisieren, muss der Benutzer zunächst spezifizieren, welche Komponenten zu seinem Netzwerk dazugehören und wie diese überwacht werden sollen. Da wichtige Daten für das Monitoring über Geräte in den Umgebungsmodellen und über die zu überwachenden Anwendungskomponenten in den Anwendungsmodellen enthalten sind, wird ersichtlich, dass eine Monitoringfunktion in einem Modellierungstool leicht realisierbar ist. Damit kann ein Monitoringmodell auf diese Modelle aufbauen.

2.4. Modellierungstool zur Erweiterung um eine Monitoringfunktion

Das Modellierungstool (IoT-Application-Modeling-Tool - IAMT), das im Rahmen dieser Arbeit, wie später in Kapitel 5 beschrieben, erweitert wurde, verfügt über Funktionen zur Modellierung der Geräte und der Software, die für funktionale Operatoren auf dem IoT-Gerät verwendet werden. Über eine grafische Benutzeroberfläche lassen sich die Komponenten aus einer Übersicht von allen verfügbaren Komponenten auswählen und in Form von Knoten dem Modell hinzufügen. Dabei können weitere Daten, wie Beschreibungen oder Parameter zur Ausführung der Komponente, eingegeben werden. Daraufhin soll das Modellierungstool automatisiert das Modell in einer IoT-Umgebung realisieren, indem alle Softwarekomponenten auf den Geräten installiert und ausgeführt werden.

Da dieses Modellierungstool erweitert werden soll, müssen Anpassungen gemacht werden. Funktionen zur Installation und Ausführung für die weiteren Softwarekomponenten müssen bereitgestellt werden und der Benutzer benötigt die Option zur Modellierung der Monitoringspezifikationen. Um sicherzustellen, dass das Modellierungstool leicht verwendbar ist, sollen möglichst geringe Anforderungen über spezifische Systemkenntnisse gefordert werden. Konzepte für die Entwicklung von Benutzeroberflächen für modellgetriebene IoT-Anwendungen werden auch in „Model-driven development of user interfaces for IoT systems via domain-specific components and patterns“ [BUA17] präsentiert. Hier wird eine Benutzeroberfläche für Smartphones gezeigt, dass eine stark abstrahierte Darstellung der IoT-Geräte ermöglicht. Jedoch unterscheidet sich das in dieser Arbeit verwendetet Modellierungstool, indem es dem Benutzer eine flexiblere Möglichkeit gibt komplexe Modelle zu bauen. Die Modelle ermöglichen leichte Darstellungen für Zusammenhänge zwischen Komponenten und eine einfache Verteilung von Software auf vielseitige IoT-Geräte. Durch die modellgetriebene Entwicklung der Anwendung in dem Modellierungtool lassen sich Geräte auch einfach austauschen oder hinzufügen, da Veränderungen in dem Modell automatisch umgesetzt werden können und beispielsweise die Software von einem defekten Gerät leicht auf ein Neues übertragbar ist [DH20b].

3. Verwandte Arbeiten

Monitoring in IoT-Umgebungen bringt neue Schwierigkeiten mit sich, auf die in vielen Arbeiten mit unterschiedlichen Schwerpunkten eingegangen wird. Dabei ist das Ziel, die modellgetriebene Entwicklung von IoT-Umgebungen gemeinsam mit passenden Monitoringlösungen zu ermöglichen. Aufgrund der Vielseitigkeit von IoT-Umgebungen, muss auf Flexibilität geachtet werden, um die Konzepte universell für viele Anwendungsfälle verwenden zu können. Methoden zur Verteilung der Monitoringkomponenten oder Datenanalyse sind wichtige Bereiche, da große Mengen an Daten produziert werden und diese analysiert und verarbeitet werden müssen. Für die Verteilung von Operatoren zur Datenverarbeitung in IoT-Umgebungen gibt es verschiedene Ansätze, wie von Franco da Silva et al. in „Model-Based Operator Placement for Data Processing in IoT Environments“ [FHM19], welches die Datenverarbeitung nah an den Geräten selbst vorsieht. Diese geschieht durch Data-Stream-Processing Modelle, welche von einem Benutzer für jede Komponente definiert werden. Hierbei werden verschiedene Algorithmen verglichen, um die Data-Stream-Processing Modelle den Geräten zuzuordnen. Somit werden Verarbeitungsschritte direkt auf den IoT-Geräten vorgenommen, welches verteilte dezentrale Datenverarbeitung ermöglicht und damit eine besser Ausnutzung der Hardware darstellt und Netzwerklast verringern kann. Von diesen Vorteilen kann jede IoT-Umgebung profitieren. Jedoch ist dabei wichtig zu erkennen, in welcher Form diese Verteilung der Operatoren zur Datenverarbeitung umgesetzt werden kann. Wenn beispielsweise, wie im Ansatz dieser Arbeit in Abschnitt 4.3.3, die gesamten Daten in Form eines Kontextmodells letztendlich zusammen bereitstehen, liegen alle benötigten Daten für eine zentrale Datenverarbeitung gemeinsam vor [GABS22]. Daher muss eindeutig für jedes Szenario klar sein, welche verarbeitenden Operationen, wie Vorfilterung der Daten, sinnvoll direkt an den Geräten vorgenommen werden können.

Monitoring einer IoT-Umgebung kann Vorteile, wie beispielsweise eine höhere Ausfallsicherheit oder bessere Auslastung der Geräte mit sich bringen. Abhängig von den Beweggründen, warum ein Gerät oder Programm überwacht werden soll, werden hier unterschiedliche Eigenschaften ausgelesen oder gemessen. In „IoT Based Approach for Load Monitoring and Activity Recognition in Smart Homes“ [FMKA21] wird Monitoring angewendet, um die Auslastung verschiedener IoT-Geräte zu überwachen. Dort werden Auslastungen anhand des Energieverbrauchs gemessen, wodurch Metadaten erkannt werden können, die auf Aktivitäten in der IoT-Umgebung schließen lassen. Mit diesen gemessenen und auch daraus abgeleiteten Daten können Optimierungen für ein intelligentes Stromnetz vorgenommen, Auslastungen überwacht oder regelmäßige Abläufe von Aktivitäten in einem Smart Home erkannt werden.

In dieser Arbeit zur Erweiterung um eine Monitoringfunktion lässt diese neben dem Energieverbrauch beliebig weitere Eigenschaften der Geräte von Monitoringsoftware überprüfen. Dies wird zwar von der Monitoringfunktion ermöglicht, hängt jedoch von der ausgewählten Monitoringsoftware ab. Das Monitoring in dieser Arbeit erkennt im Gegensatz zu „IoT Based Approach for Load Monitoring and Activity Recognition in Smart Homes“ den Zustand der IoT-Umgebung anhand der von dem Gerät gezielt übermittelten Daten. Dabei wird regelmäßig die Verfügbarkeit und damit der verbundene Status abgefragt, oder es werden direkt Messwerte der Monitoringsoftware übertragen, so dass dadurch auf den Zustand der Umgebung geschlossen werden kann. Allerdings muss abgesehen von der allgemeinen Verfügbarkeit der Geräte der Benutzer eigenständig die Aktivitäten in der IoT-Umgebung aus den Daten in der Monitoringübersicht auslesen.

Manuelles Monitoring für IoT-Geräte ist aufgrund hoher Anzahl und unterschiedlicher Gerätetypen sehr aufwändig. Deshalb setzt sich Cloud-Monitoring in IoT-Umgebungen immer stärker durch. Je weiter sich Cloud-Monitoring verbreitet, umso wichtiger wird es, mit dessen neuen Herausforderungen umzugehen. Dabei gibt es verschiedene Anbieter, die einem Cloud-Monitoring Lösungen bereitstellen. Um dabei keine Abhängigkeiten von einzelnen Produkten aufzubauen, sollten Benutzer losgelöst von den Anbietern arbeiten können, da sonst der Wechsel von großen Cloud-Anwendungen eine risikoreiche und kostenintensive Entscheidung werden kann. Zu diesem Problem werden in „Avoiding Vendor-Lockin in Cloud Monitoring Using Generic Agent Templates“ [MHSM20] Lösungen durch das Verwenden von Templates dargestellt. Diese Templates können von dem Benutzer verwendet werden, um universelle Monitoringspezifikationen zu modellieren. Die Templates können dann für eine Vielzahl an Cloud-Monitoringsystemen verwendet werden und erleichtern somit das Wechseln zwischen Systemen. Auch die in dieser Arbeit in Abschnitt 5.3 verwendete Monitoringlösung mit InfluxDB2.x[1] kann über die InfluxDB Cloud verwendet werden und damit von der Verwendung der Templates profitieren. Die Monitoringfunktion des Modellierungstools dieser Arbeit bietet bereits die Flexibilität, schnell zwischen Monitoringprogrammen zu wechseln. Dies liegt daran, dass angefragten Daten nicht für die verwendete InfluxDB in dem Tool spezifisch definiert werden müssen. Allerdings werden keine komplexeren Datenverarbeitungsschritte innerhalb der von IndfluxData bereitgestellten Monitoringlösung verwendet, was weitere Konfiguration erfordert. Mehr Verwendung der Cloud-Monitoringfunktionalität kann für das Modellierungstool Vorteile bringen. Durch die Verwendung der vorgestellten Templates gegen Vendor-Lockin kann die Erweiterbarkeit für verschiedene Monitoringsysteme vereinfacht werden.

IoT-Anwendungen verwenden viele Geräte und produzieren große Mengen an Daten, wie zum Beispiel Messungen. Sowohl in zentralen, als auch dezentralen Systemen werden große Mengen an Daten übertragen. Abhängig davon, ob kleine Geräte direkt untereinander kommunizieren oder ein Gerät seine Daten an einen zentralen Server schickt, werden geeignete Kommunikationsprotokolle benötigt. Da außerdem in einer immer stärker vernetzten Welt auch IoT-Systeme

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

miteinander verbunden werden, werden in beispielsweise all-IP Systemen weitere Protokolle integriert [AXM04]. Schließlich sollen auch veraltete Systeme an größere IoT-Umgebungen angeschlossen werden, da nicht jedes aktuell verwendetet lauffähige System an eine einheitliche Form angepasst werden kann. Nicht jedes System ermöglicht ein globales einheitliches Kommunikationsprotokoll, da sowohl der Datenaustausch direkt zwischen Geräten, aber auch über Clouds oder Servern implementiert werden muss. Diese Verbindungen zwischen Geräten werden auch in den Modellen des Modellierungstools IAMT dargestellt. Dabei werden lediglich Abhängigkeiten zwischen den Komponenten dargestellt, die jedoch wichtig für die Monitoringerweiterung sind. Ausfallende Kommunikation zwischen Geräten kann dabei Auswirkungen auf weitere Komponenten oder sogar die gesamte IoT-Umgebung haben. Wenn Abhängigkeiten zwischen Geräten bestehen, ist ein Versagen einer Komponente ein größeres Problem, da weitere Funktionen eingeschränkt werden, auch wenn die anderen Geräte keine eigenen Fehler aufweisen. Da dies auch für kleine leistungsschwache IoT-Geräte gilt, müssen diese Konzepte an diese Eigenschaften der IoT-Umgebungen angepasst werden. Dabei sollen verfügbare Ressourcen verwendet werden, aber gleichzeitig Geräte nicht durch Protokolle mit großem Overhead unnötig mit aufwendiger Kommunikation überlastet werden [DH20a]. Dafür wird eine Messaging Engine zur Kommunikation zwischen den Geräten entworfen, die eine dezentrale Datenverarbeitung ermöglicht. Diese Kommunikation ist für zeitkritische Anwendungen ein Vorteil, da Daten direkt übermittelt werden und dabei Datenverarbeitungsschritte direkt an den Komponenten ausgeführt werden können. Die Kommunikation wird dabei durch Data-Flow Modelle beschrieben, welche die Abhängigkeiten beschreiben und damit für das Monitoring wichtig sind. Für diese Arbeit zur Erweiterung um eine Monitoringfunktion ist dabei das Monitoring zu Erkennung der Auslastung der Geräte durch weitere Kommunikationslösungen, aber auch das Monitoring der Kommunikation selbst wichtig, da fehlerhafte oder ausfallende Datenübertragung weitere Fehler in abhängigen Komponenten auslösen kann. Das Monitoring der Arbeit lässt sich dabei durch geeignete Monitoringsoftware dafür konfigurieren, aber der Benutzer muss die Abhängigkeiten eigenständig erkennen, modellieren und schließlich das Monitoring dazu passend konfigurieren.

Abhängigkeiten spielen gerade in sicherheitskritischen Systeme eine große Rolle. Das Monitoring, welches in dieser Arbeit hinzugefügt wird, kann dabei flexibel auf verschiedene Geräte angewendet werden. Diese Geräte müssen zunächst als kritische Komponenten erkannt werden, damit ihre Ausfallsicherheit besser gewährleistet wird. Dazu werden in dem Modellierungstool IAMT Verbindungen zwischen Komponenten hergestellt, so dass dabei Abhängigkeiten erkannt werden können und das Monitoring leicht darauf angepasst werden kann. Herausforderungen für sicherheitskritische Systeme werden in „Model-Driven Engineering for Mission-Critical IoT Systems“ [CCD+17] beschrieben und durch im Modell definierte Sicherheitseigenschaften universell für verschiedene Gerätetypen behandelt. Diese Systeme reagieren durch Kontextinformationen auf aktuelle Begebenheiten und passen nach Bedarf das Modell an, um die geforderten Funktionalitäten wieder herzustellen. Dafür müssen Ressourcen in der IoT-Umgebung verfügbar sein, die die Anforderungen des betroffenen Gerätes erfüllen. Um das Modell automatisch auf passende alternative Hardware anzupassen, müssen diese Ressourcen erkannt werden und im Fehlerfall verfügbar sein. Dies wird durch das Monitoring

dieser Arbeit ermöglicht, da freie Ressourcen festgestellt werden können und Monitoring leicht an neue Begebenheiten adaptiert werden kann. Dabei werden nicht universell für alle Geräte gleiche Anforderungen spezifiziert, da viele heterogene Geräte verwendet werden, sondern der Benutzer kann individuell die Konfiguration schnell ändern.

Die Multi-purpose Binding and Provisioning Platform[2] (MBP) ist eine Open-Source IoT-Plattform, die zum Sammeln, Verarbeiten und Visualisieren von Daten verwendet werden kann [FHS+20]. Darüber hinaus lassen sich IoT-Geräte managen, sodass diese individuell mit Softwarekomponenten konfiguriert werden können. Dazu gehört neben der funktionalen Software für die Anwendung der IoT-Umgebung auch Monitoringsoftware. Die Monitoring-software lässt sich auf die Plattform hochladen und dann einfach auf verfügbaren IoT-Geräten verteilen. Die Messwerte der Monitoringsoftware werden außerdem zur Laufzeit dem Benutzer in der MBP angezeigt. Diese Arbeit baut auf die MBP auf und nutzt die Funktion zum Registrieren von IoT-Geräten, Anwendungsoftware und Monitoringsoftware. Außerdem wird die Funktion zur Installation und Ausführen der Software verwendet. Jedoch wird dies nicht von dem Benutzer direkt in der MBP ausgeführt, sondern die Funktion wird automatisch von dem Modellierungstool gestartet, wenn diese für das vom Benutzer erstelle Modell benötigt wird. Die Monitoringdaten werden ohne die MBP an das Modellierungstool übermittelt und angezeigt. Dadurch werden alle Daten übersichtlich direkte in dem Anwendungsmodell, welches um die Monitoringfunktion erweitert wurde, den Komponenten zugeordnet angezeigt.

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

4. Konzept für ein Monitoringsystem für modellgetriebene IoT-Anwendungen

IoT-Anwendungen können aufgrund vieler zusammenhängender Komponenten eine komplexe Struktur haben. Deshalb werden Modellierungstools verwendet, die einem ermöglichen, diese Struktur abstrahiert abzubilden [IRMP21]. Die entstehenden Modelle werden dann verwendet, um die Geräte in der IoT-Umgebung zu steuern und diese mit Software auszustatten, sodass die gewünschte Funktionalität gewährleistet wird. Da IoT-Umgebungen vielseitige Fehler aufweisen können, welche in Abschnitt 4.2 erklärt werden, müssen diese Systeme kontrolliert und überprüft werden [NYO+18]. Jedes Gerät weist unterschiedliche Schwachstellen auf und Systeme, in denen sie verwendet werden, können sich durch Größe, Komplexität und Anwendungsgebiet stark unterscheiden. Daher muss die Überwachung des Systems auf den Anwendungsfall angepasst sein. In dem Modellierungstool, welches im Rahmen dieser Arbeit erweitert wird, soll daher die Monitoringfunktion so integriert werden, dass weiterhin eine große Flexibilität in dem System zugelassen wird. Damit soll das Modellierungstool in verschiedensten, vielseitigen IoT-Umgebungen verwendet werden können. Hierfür werden in diesem Abschnitt konkrete Anforderungen an die Entwicklung gestellt und ein Konzept für ein flexibles Monitoring präsentiert. Zuerst werden in Abschnitt 4.1 die Anforderungen an die Anwendung definiert, um die in Abschnitt 1.2 beschriebenen Ziele umzusetzen. In Abschnitt 4.2 werden zu den verschiedenen Fehlertypen die passenden Möglichkeiten der Fehlererkennung beschrieben. Schließlich wird in Abschnitt 4.3 der Gesamtablauf des Modellierungstools erklärt und in Abschnitt 4.4, wie es in die Architektur der Umgebung integriert wird.

4.1. Anforderungen an das Monitoringsystem

Aus den in Abschnitt 1.2 beschriebenen Zielen wurden Anforderungen extrahiert, die in diesem Abschnitt beschrieben werden. Anhand dieser Anforderungen wird das Konzept für das Monitoring des Modellierungstools entworfen. Der Benutzer soll über eine grafische Benutzeroberfläche mit dem Modellierungstool interagieren können, um ihm eine leichte Bedienung zu ermöglichen (Anforderung 1). Dort soll der Benutzer die Möglichkeit haben, IoT-Geräte und Softwarekomponenten individuell zu konfigurieren, um die Software auf dem zugeordneten Gerät zu installieren und auszuführen (Anforderung 2). Da auch in der Monitoringfunktion auf Skalierbarkeit geachtet werden muss, soll die Verteilung der Software auf den Geräten, sowie die Installation und Ausführung automatisiert möglich sein (Anforderung 3). Ohne diese

Automatisierung würde der Aufwand zum initialen Aufsetzen der IoT-Umgebung und erneut für jede Anpassung, bei immer größer werdenden Systeme einen zu hohen Zeitaufwand erfordern. Die Softwarekomponenten sollen beispielsweise Funktionalitäten wie die Messungen von verfügbaren Sensoren zurückgeben, oder die Überwachung der Geräteauslastung und der Verfügbarkeit der Geräte bieten. Dem Benutzer sollen diese Daten übersichtlich angezeigt werden (Anforderung 4). Um die Daten nicht händisch auswerten zu müssen, sollen Regeln definiert werden können, wie diese Werte gefiltert werden. Durch die erstellten Regeln werden Fehlerkriterien für jede Komponente individuell definiert. Die Fehlerkriterien können dabei auf unterschiedliche Kategorien an Fehlertypen abzielen, welche im folgenden Abschnitt 4.2 erklärt werden (Anforderung 5). Schließlich muss der Benutzer über erkannte Fehler informiert werden, um eine Fehlerbehandlung zu ermöglichen (Anforderung 6).
Im Folgenden werden alle genannten Anforderungen zur Übersicht nochmal aufgelistet:

- Anforderung 1: Leichte Bedienbarkeit
- Anforderung 2: Geräte und Software individuell konfigurieren
- Anforderung 3: Automatische Installation und Ausführung der Monitoringsoftware
- Anforderung 4: Übersicht aller aktuellen Daten
- Anforderung 5: Verschiedene Fehlertypen erkennen
- Anforderung 6: Warnung vor Fehlern

Diese festgelegten Anforderungen dienen im Weiteren zur Umsetzung des Konzepts und der Implementierung und werden am Ende der Implementierung in Abschnitt 5.5 evaluiert.

4.2. Fehlererkennung

Um eine funktionierende Fehlererkennung umzusetzen, müssen Fehler zunächst definiert werden. Auch wenn es in vielen Fällen aus der Sicht des Benutzers eindeutig ist, wenn ein Fehler eintritt, muss dieser eindeutig definiert sein, um von einem System automatisch erkannt zu werden. Fehler können aus vielen Ursachen entstehen und machen sich durch unterschiedliche Arten von Fehlverhalten der Anwendung bemerkbar. Auch der Rückschluss eines Fehlverhaltens auf den Fehler ist nicht in jedem Fall eindeutig. Monitoringprogramme können lediglich Fehlverhalten oder Anzeichen für mögliche Fehlerquellen detektieren und von den Ergebnissen kann dann gegebenenfalls auf den Fehler im System geschlossen werden. Daher muss unterschieden werden, ob das erkannte Ergebnis möglicherweise nur vor einer hohen Auslastung des Systems warnen soll oder ob ein eindeutiger Fehler vorliegt. Zum Beispiel kann dies ein Wert außerhalb des Wertebereichs von einem Sensor sein, der einen Fehler darstellt, da dieses Fehlverhalten eindeutig ist. Außerdem kann die Fehlererkennung für verschiedene Gerätetypen unterschiedlich aussehen, da diese beispielsweise von anderen Monitoringsystemen überprüft

werden müssen. Manche Informationen müssen aus den empfangenen Werten bezogen werden, aber auch andere Messdaten können direkt von Systemen ausgelesen und weitergegeben werden. Da die Systeme mehrere Geräte umfassen, wird Kommunikation über ein Netzwerk benötigt, um die Anwendungsdaten, aber auch die Monitoringdaten zu sammeln. Dabei können Netzwerkfehler bei der Übertragung entstehen oder Geräte können nicht mehr erreichbar sein. Unterschiedliche Fehler haben unterschiedliche Auswirkungen und damit hat auch nicht jeder Fehler gleich viel Einfluss auf das Gesamtsystem. Deswegen muss nach der Erkennung eines Fehlers entschieden werden, wie darauf reagiert werden soll [NYO+18]. Dabei können verschiedene Fehlertypen durch unterschiedliche Abfragen überprüft werden. Beispiele für Fehlertypen, die auf unterschiedliche Weise erkannt und dann auch unterschiedlich behandelt werden müssen, sind:

- Nichterreichbarkeit des Gerätes
- Falsche Messwerte
- Überlastung des Gerätes (z.B. CPU Auslastung)

[DH21]

Diese Fehlertypen sollen mithilfe der Monitoringerweiterung erkannt werden können. Wie dies jeweils umgesetzt werden kann und wie mit den Fehlern umgegangen wird, ist in den folgenden Abschnitten beschrieben.

4.2.1. Nichterreichbarkeit des Gerätes

Nicht erreichbare Geräte stellen eine Schwierigkeit dar, da keine Daten mehr empfangen werden können und daher der aktuelle Zustand des Gerätes nicht verfügbar ist. Ob ein Gerät erreichbar ist, lässt sich leicht durch eine Anfrage an das Gerät mithilfe eines Timeouts überprüfen. Dies kann durch ein Signal erreicht werden, welches in regelmäßigen Abständen an jedes Gerät geschickt wird [DH20b]. Dann wird eine feste Zeitspanne auf eine Antwort gewartet, um festzustellen, ob das Gerät reagiert. Durch diese Überprüfung kann zwar die Nichterreichbarkeit festgestellt werden, allerdings kann dies wieder verschiedene Ursachen haben. Da ein fehlendes Antwortsignal nicht immer einen Defekt im Gerät signalisiert, kann auch hier nicht eindeutig festgestellt werden, was passiert ist. Es ist möglich, dass die Hardware einen Defekt aufweist und ein menschlicher Eingriff in das laufende System erforderlich ist. Andererseits kann das Gerät temporär überlastet sein und daher nach dem Abfallen der Last wieder funktionieren. In diesem Fall muss ebenfalls reagiert werden, indem beispielsweise das Gerät durch ein leistungsstärkeres ausgetauscht wird oder die Ursache für die hohe Auslastung eingeschränkt wird. Schließlich kann, wenn der Fehler nicht mehr akut festgestellt wird, trotzdem ein Problem bestehen, das in Zukunft wieder eintreten kann, oder sogar für weitere Probleme sorgen kann. Eine weitere Option ist, dass es sich um ein mobiles Gerät handelt und es lediglich aus dem Netzwerk temporär entfernt wurde. Dies ist kein Fehler, sondern möglicherweise ein erwartetes Verhalten und erfordert daher keinen Eingriff, da erst, wenn das Gerät wieder in das Netzwerk zurückkehrt, Kommunikation aufgenommen werden muss.

Ob ein Gerät verfügbar ist, stellt in jedem Fall eine grundlegende Eigenschaft dar, weshalb dies standardmäßig für jedes Gerät überprüft werden soll. Die Nichtverfügbarkeit eines Gerätes kann der Auslöser sein, ob weitere Funktionalitäten eingeschränkt werden, wie zum Beispiel das Senden von Sensorwerten. Daher ist die Nichterreichbarkeit zwar nicht in jedem Fall ein Fehler, aber sollte trotzdem beachtet werden, da auch ohne Fehler auf die Erkennung der Nichterreichbarkeit reagiert werden können muss.

4.2.2. Falsche Messwerte

Falsche Messwerte können in IoT-Umgebungen schnell entstehen, da oft kleine, unzuverlässige Hardware verwendet wird. Da hohe Zuverlässigkeit nicht für jeden Anwendungsfall gleich wichtig ist, kann trotzdem mit diesen Werten umgegangen werden. Dabei kann es ausreichen, einzelne Werte herauszufiltern und damit zu ignorieren. Wenn jedoch ein Problem besteht, welches durch Korrekturen in dem System behoben werden kann, müssen diese Anpassungen vorgenommen werden, um das System zu verbessern und die Fehler in Zukunft zu vermeiden. Falsche Messwerte können unterschiedliche Reaktionen erfordern und sind je nach Genauigkeit schwierig festzustellen, da zunächst klar sein muss, was einen falschen Wert ausmacht. Hier können zum Beispiel komplexere Algorithmen zur Ausreißererkennung angewendet werden, was jedoch nicht weiterer Fokus dieser Arbeit ist [RLP08]. Dem Benutzer soll trotzdem die Möglichkeit gegeben werden, individuelle Regeln zu definieren, welche im einfachsten Fall Schwellwerte sind, die den akzeptierten Bereich an empfangenen Werten eingrenzt.

Beispiel 1 - Regel mit Schwellwerten für Messwert x min_x, max_x:

$$x_{min} < x < x_{max}$$

Beispiel 2 - Regel mit Abhängigkeit von dem vorherigen Wert:

$$(x_{min} < x_n < x_{max}) \wedge (x_{n-1} * 2 < x_n)$$

In IoT-Geräten können nicht überall hochwertige, sehr genaue Sensoren verwendet werden. Die meisten Geräte sind klein und können nur geringe Genauigkeit der Messungen garantieren. Nicht jedes Gerät weist Probleme bei der Qualität der Messwerte auf. Allerdings ist es ohnehin in vielen Szenarien nicht nötig, eine sehr hohe Garantie für die Genauigkeit der Messwerte zu haben. Daher muss die Hardware auf den Anwendungsfall abgestimmt sein. Abhängig davon, ob Temperaturen in einer Chemieanlage mit einer hohen Präzision gemessen werden müssen oder ob zu Hause am Kühlschrank überprüft wird, ob dieser seine Temperatur hält, müssen geeignete Komponenten ausgewählt werden. Um dies umsetzen zu können, muss die Definition für falsche Werte dementsprechend angepasst werden. Falsche Werte, die eindeutig unmöglich zu erreichen sind, lassen sich dabei klarer erkennen, als ungenaue Werte. Zur Erkennung von Fehlern durch zu hohe Ungenauigkeit müssen ebenfalls unterschiedliche Ansprüche an Sicherheitsgarantien erfüllt werden. Der Benutzer muss auch in diesem Fall für jede Komponente die Möglichkeit haben individuell die Konfiguration anzupassen, um

gegebenenfalls in wichtigen Fällen lieber einmal zu viel gewarnt zu werden und in anderen Szenarien nicht mit irrelevanten Warnungen gestört zu werden.

4.2.3. Überlastung des Gerätes

Überlastete Geräte sind fehleranfälliger und haben eingeschränkte Funktionalität. Vor diesem Problem soll ebenfalls gewarnt werden, obwohl alleinig eine hohe Auslastung keinen Fehler darstellt und damit auch kein Problem für das Gerät vorliegen muss. Trotzdem ist es sehr wichtig, diese Eigenschaften zu überwachen [NYO+18]. Fehler treten schneller auf und können daher möglicherweise präventiv verhindert werden. Um Überlastung eines Gerätes zu erkennen, erfordert es das Nutzen von Monitoringsoftware, die zusätzlich auf dem Gerät installiert wird. Selbst wenn sich in der Anwendung keine Fehler bei einer hohen Last bemerkbar machen, gehen aus Monitoringdaten der Auslastung eines Gerätes immer noch relevante Informationen hervor. Bei einer hohen Auslastung, die jedoch keinen Fehlerfall auslöst, können die aktuellen Daten des Gerätes markiert werden, so dass sicherheitskritische Systeme diese Markierung als Vorwarnung ansehen können. Außerdem ist es für den Benutzer ein Zeichen, dieses Gerät nicht für weitere Anwendungen zu verwenden oder möglicherweise die Hardware durch eine leistungsstärkere Komponente auszutauschen. Ebenfalls kann aus hoher Auslastung der Fehler „Nichterreichbarkeit des Gerätes“ 4.2.1 folgen, was zum Beispiel durch Überhitzen geschehen kann. In diesen beschriebenen Situationen ist bis jetzt nur allgemein von der Auslastung die Rede. Als erste Metrik kann dafür die CPU-Auslastung verwendet werden, jedoch sollten weitere Eigenschaften ebenfalls überwacht werden. Freier Arbeitsspeicher und ebenfalls persistenter Speicher auf dem Gerät können das System einschränken. Der persistente Speicher des Gerätes ist hierbei in den meisten Fällen nicht der Auslöser für Einschränkungen des Gerätes, da Speicherlimitierung im Wesentlichen bestimmt, ob Platz für weitere Programme auf dem Gerät ist. Da Speicher billiger ist, als beispielsweise Prozessorleistung, sind meistens andere Eigenschaften der Hardware, aufgrund der Kosten, stärker limitiert, als der Speicherplatz. Je nach Monitoringsoftware können hier noch viele weitere Eigenschaften unter Betracht gezogen werden. Der Benutzer kann daher bei der Konfiguration des Monitorings in dem Modellierungstool entscheiden, welche Eigenschaften überwacht werden. Mögliche Zusammenhänge zwischen hohen Auslastungen und falschen Messwerten können gegebenenfalls auch gefunden werden. Dies erfordert zwar das Erkennen dieser Korrelation, aber kann eine wichtige Information bei der Suche nach Fehlerquellen sein. Um dies alles zu bewerkstelligen, sollen auch für diese Messwerte der Monitoringsoftware dem Benutzer die Option gegeben werden, Regeln analog zu den Sensorwertregeln zu erstellen. Diese Regel legen fest, ab wann der Benutzer vor hohen Auslastungen gewarnt wird.

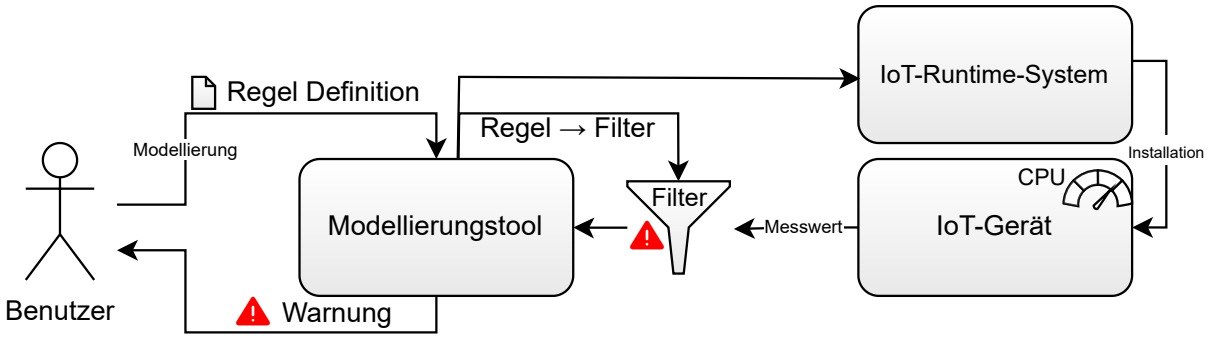

Abbildung 4.1.: Modellierung & Fehlererkennung durch den Benutzer

4.3. Ablauf von Sensor- und Monitoringdatenerhebung

Der Prozess, relevante Daten, von sowohl Monitoringsoftware als auch Sensoren, zu erhalten, erfordert eine klare Struktur, um auch bei komplexeren Anwendungen zu jeder Komponente die korrekten Daten zuzuordnen. Die Struktur der Komponenten ist in Abbildung 4.1 zu erkennen, welche die Fehlererkennung darstellt. Dabei wird der modellorientierte Ansatz zur Anwendungsentwicklung aus „Towards Feedback Loops in Model-Driven IoT Applications“ [DH21] für die Modellierung erweitert. Zuerst wird das Umgebungsmodell definiert, das beispielsweise Informationen über verfügbare Geräte enthält. Dieses Modell wird zum Erstellen des Anwendungsmodells genutzt. Die Funktionalität der Anwendung ist die Motivation zur Implementierung eines IoT-Systems in die Umgebung. Nachdem dies definiert ist, wird in dieser Arbeit für den Ablauf der Monitoringdatenerhebung das Monitoringmodell integriert. In diesem Modell werden die Monitoringprogramme dargestellt und Spezifikationen zu Fehlererkennung hinzugefügt, die während der Laufzeit überprüft werden. Danach geht es wieder mit dem Konzept aus „Towards Feedback Loops in Model-Driven IoT Applications“ [DH21], mit dem Model-Mapping, für die Installation und Ausführung des Modells, weiter. Die Installation wird vorgenommen und die Softwarekomponenten für einerseits die Anwendung und andererseits für das Monitoring werden gestartet.

Der Ablauf dieses Prozesses zur Erstellung der verschiedenen Modelle wird dabei ab der Anwendungsmodellierung bis zum Start der Installation von dem Modellierungtool übernommen, das in dieser Arbeit erweitert wird. Außerdem werden die aktuellen Daten während der Ausführung in einer Übersicht für das Monitoring dargestellt. Um dies zu ermöglichen, ist das Modellierungstool in verschiedene Bereiche aufgeteilt, in denen schrittweise die erforderlichen Konfigurationsdaten eingebunden werden. Die Struktur des dieser Arbeit zugrunde liegenden Modellierungstools IAMT ist hierfür bereits vorgegeben und folgt dem zuvor beschriebenen modellgetrieben Ansatz zur Umgebungs-, Anwendungs- und jetzt auch Monitoringmodellierung [DH21]. Danach wird durch Model-Mapping ein Deployment-Modell erstellt. Um den Entwicklungsprozess zu erläutern, wird an dieser Stelle das Tool vollständig beschrieben, um

auch Problemstellungen und Schwierigkeiten nachvollziehen zu können. Zuerst werden die Modelle schrittweise wie der Abbildung 4.2 erstellt.

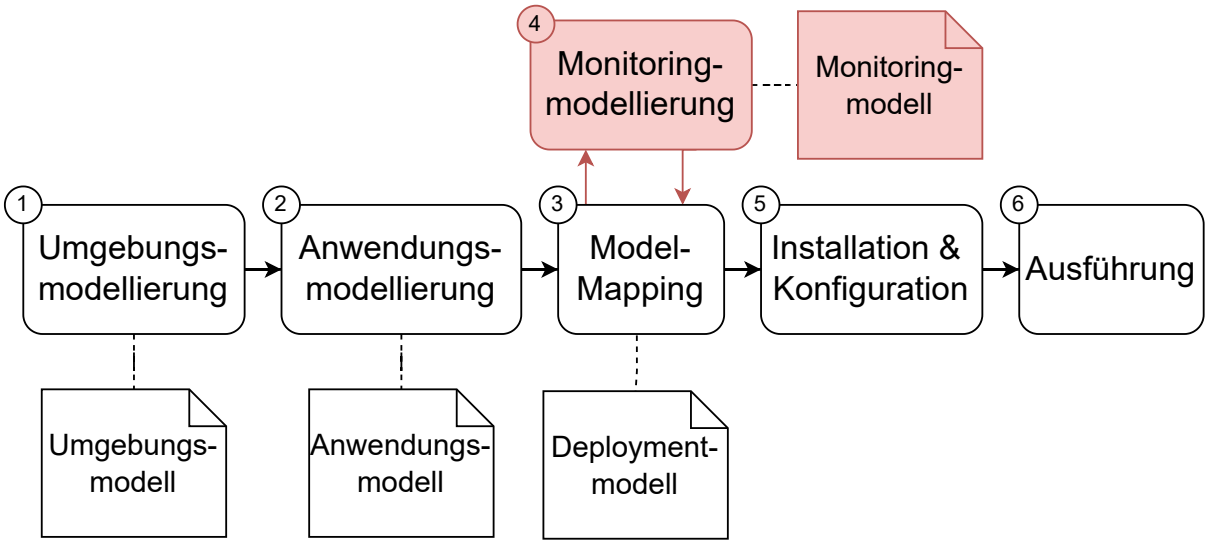

Abbildung 4.2.: Ablauf modellgetriebene IoT-Anwendungmodellierung mit Monitoringfunktion - Erweiterung aus „Towards Feedback Loops in Model-Driven IoT Applications“ [DH21]

Abbildung 4.3 zeigt in drei Schritten schematisch, wie die mit dem Modellierungstool erstellten Modelle angewendet werden sollen. Dabei werden im ersten Schritt für das Monitoring neben den Monitoringagenten auch Filter zur Fehlererkennung definiert. Monitoringagenten sind Komponenten, die zum Auslesen oder Messen der Monitoringeigenschaften auf den Geräten für die spezifischen Monitoringlösungen installiert werden. Im zweiten Schritt wird das Modell in der Umgebung angewendet, wofür alle modellierten Anwendungs- und Monitoringssoftwarekomponenten auf den zugeordneten Geräten installiert und gestartet werden müssen. Nachdem dies geschehen ist und alle Komponenten ausgeführt werden, werden in Schritt drei Daten von den einzelnen Geräten wieder zurück an das Modellierungstool übermittelt. An dieser Stelle lassen sich weitere Zwischenschritte einfügen, die je nach Anwendungsfall notwendig sind. Ein Beispiel dafür ist der Filter in der Abbildung, der in diesem Fall verwendet wird, da nicht direkt Rohdaten angezeigt werden sollen, sondern der Filter die Daten validiert oder wie in Abschnitt 4.2 Fehler erkennt. Da auch dieser Filter nicht universell für jede Komponente gleich aussehen kann, muss der Benutzer eine Spezifikation der Kriterien für den Filter einbinden. Wie für diese Anforderungen ein Modellierungstool aussehen kann, wird nun schrittweise beschrieben.

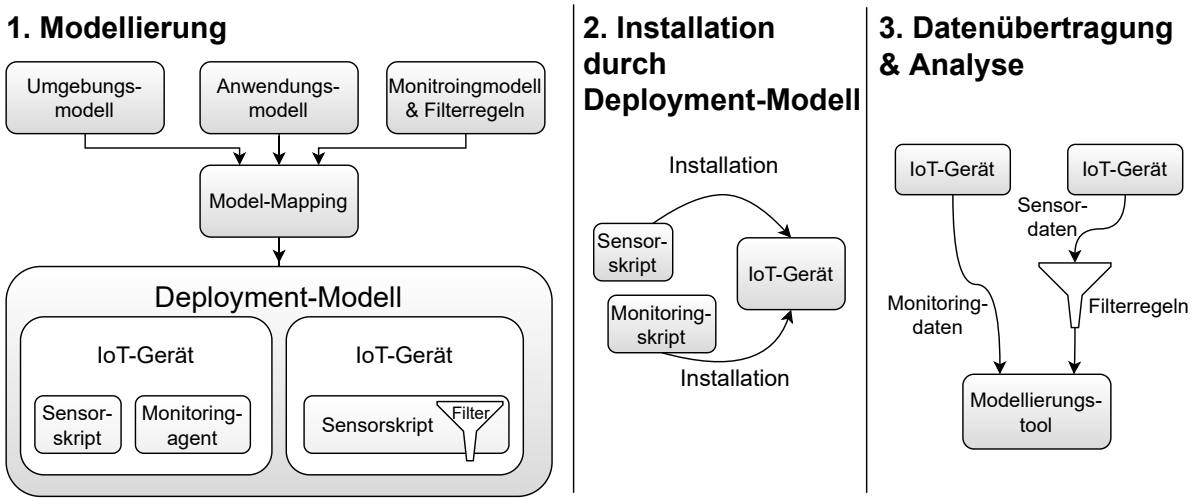

Abbildung 4.3.: Ablauf der Anwendung eines IoT-Modells

4.3.1. Modellierung

Die Modellierung umfasst mehrere Schritte. Dabei wird mit der IoT-Umgebung angefangen, die die verfügbaren Geräte enthält. Darauf folgt die Anwendungsmodellierung, in der die Funktionsweise der Anwendung beschrieben wird. Um diese Anwendung zu realisieren wird Model-Mapping angewendet, wodurch die Anwendungskomponenten Geräten zugewiesen werden und ein Deploymentmodell entsteht. Danach wird in dieser Arbeit das Monitoring modelliert. Dieser Teil der Modellierung ist der Fokus dieser Arbeit, da Konzepte für die Umgebungs- und Anwendungsmodellierung bereits umgesetzt sind und das Modellierungstool IAMT lediglich um die Monitoringfunktion erweitert werden soll. Da ebenfalls die Monitoringsoftware auf Geräte verteil werden muss, wird diese nach der Modellierung ebenfalls dem Deploymentmodell hinzugefügt. Die Teile zur Erstellung des Deploymentmodells, welches für die Installation benötigt wird, sind in Abbildung 4.3 im ersten Abschnitt zu sehen. In den folgenden Abschnitten werden alle benötigten Modell beschrieben.

Umgebungsmodell

Zunächst werden in dem Umgebungsmodell nur die verfügbaren Geräte eingefügt. Damit wird die grundlegende Struktur beschrieben, die verwendet werden kann, um Anwendungen in der Umgebung zu realisieren. Die Hardware gehört zu den statischen Daten des Systems, da diese sich selten oder in der Regel nur den manuellen Eingriff in das System ändert. Die modellierten Hardwarekomponenten stehen damit für die Anwendung zur Verfügung.

Anwendungsmodell

Dieses Modell beschreibt die Funktion, die von dem System umgesetzt werden soll. Dazu werden Softwarekomponenten in das Modell eingefügt. Abhängigkeiten zwischen den Komponenten werden ebenfalls modelliert. Abhängigkeiten können entstehen, wenn Komponenten beispielsweise Messwerte von anderen Komponenten benötigen. In diesem Fall wird gegebenenfalls direkte Kommunikation zwischen den Komponenten benötigt und eine Messaging-Engine kann dem Modell hinzugefügt werden.

Deployment-Modell

Durch Modell-Mapping werden allen Softwarekomponenten Geräte zugewiesen, wodurch das Deployment-Modell erstellt wird. Damit werden die Informationen festgelegt, um das Modell automatisch realisieren zu können. Da, in diesem Modellierungsschritt festgelegt wird, welche Geräte verwendet werden, wird erst danach das Monitoringmodell erstellt. Allerdings muss nach der Erstellung des Monitoringmodells auch die Monitoringsoftware dem Deploymentmodell hinzugefügt werden, weshalb das Deploymentmodell erst danach fertiggestellt wird.

Monitoringmodell

Da zum Erstellen des Monitoringmodells die Funktionalität des Systems bereits modelliert ist und alle Informationen über verwendetet Geräte und mögliche verfügbare Messwerte bekannt sind, kann das Monitoring hinzugefügt werden. In diesem Modellierungsschritt, der in diese Arbeit hinzugefügt wird, werden weitere Softwarekomponenten, die für das Monitoring verwendet werden, hinzugefügt. Dies sind Monitoringagenten, die auf den IoT-Geräten die Monitoringdaten sammeln und weitergeben, sodass diese dem Benutzer angezeigt werden. Außerdem werden in diesem Schritt Filter definiert, die für die Fehlererkennung angewendet werden, sobald Messwerte empfangen werden. Wenn die Kriterien für einen Fehler erfüllt werden, wird dies dadurch erkannt.

Der Benutzer soll die Möglichkeit haben, weitere Daten den Modellen hinzuzufügen, da diese die Aussagekraft steigern und Funktionalitäten ermöglichen können. Zusätzliche Informationen können für Funktionalitäten gebraucht werden, wie beispielsweise IP-Adressen für Kommunikationen. Oder nichtfunktionale Beschreibungen der Komponenten können dem Benutzer helfen, Modelle besser zu verstehen. Jedes Gerät wird daher individuell beschrieben und dadurch an die Begebenheiten der IoT-Umgebung, der zugeordnete Software und auch dem gesamten restlichen Modell angepasst.

4.3.2. Installation und Ausführung der Monitoringkomponenten

Nach der Modellierung folgt die Installation und Ausführung. Die Installation und Ausführung der Monitoringsoftware und Anwendungssoftware verläuft ähnlich, da in beiden Fällen eine Software auf ein Gerät geladen werden muss und nach dem Start der Ausführung Daten empfangen werden können. In diesem Schritt soll, solange das System keine Fehler erkennt, der Benutzer keine Aufgabe haben. Nachdem das Modell vollständig mit allen notwendigen Informationen konfiguriert wurde, wird die Monitoringsoftware automatisch auf dem zuvor zugeordneten Gerät installiert und gestartet. Um dies umzusetzen, muss selbstverständlich für jedes Gerät zuvor die Funktion gegeben sein, dass dieser Zugriff automatisch erfolgen darf. Beispielsweise kann es notwendig sein, für den Zugriff eine SSH-Verbindung zu erlauben. Durch die Automatisierung von Installation und Ausführung der Komponenten wird Skalierbarkeit garantiert. In größeren Systemen muss sonst manuell zunächst auf jedes Gerät zugegriffen werden, die Software auf das Gerät geladen und erst danach die Installation und Ausführung gestartet werden. Dies würde einen hohen zeitlichen Aufwand darstellen, da gerade in der Entwicklung und Konfigurationszeit Systeme häufiger verändert werden und damit gehäuft der Extraaufwand durch manuelle Installationen ins Gewicht fallen würde. Jedoch müssen in dem Installationsprozess ebenfalls die Schwächen einer IoT-Umgebung beachtet werden. Die zuvor präsentierten Fehlerfälle „Nichterreichbarkeit des Gerätes“ und „Überlastung des Gerätes“ können bereits während der Konfiguration der Anwendung eintreten. Erneut kann es dazu kommen, dass beispielsweise mobile Geräte zum Zeitpunkt der Installation nicht im Netzwerk sind oder Geräte während des Installationsvorgangs ausfallen und daher nicht erreichbar sind. Hier muss überprüft werden, ob die Installation und der Start der Software erfolgreich war oder ob gegebenenfalls die Installation beispielsweise nicht anfangen konnte oder ob das Programm danach nicht ausgeführt wird. Gegebenenfalls muss der Schritt zu einem späteren Zeitpunkt wiederholt werden oder mit abgebrochenen Installationen umgegangen werden. Auch wenn eine Überlastung der Geräte möglicherweise eine geringere Bedeutung in diesem Bereich hat, da der Zustand nicht regelmäßig eintritt, kann dies während dem Installationsprozess eintreten und wieder die Nichterreichbarkeit verursachen. Überlastungen sind zu diesem Zeitpunkt kaum zu erkennen, da die Monitoringsoftware erst installiert werden soll, wodurch Fehler schwer automatisch zu erfassen sind. Daher ist an dieser Stelle meistens ebenfalls der automatische Neustart des Installationsprozesses nach einem Timeout die gewählte Option. Durch eine Benachrichtigung an den Benutzer kann, im Fall von mehreren Fehlschlägen der Installation, das Problem weiter untersucht werden.

4.3.3. Datenübertragung und Analyse

Jedes Gerät soll die Möglichkeit haben, erzeugte oder gemessene Daten an die Monitoringfunktion des Modellierungstools zu übermitteln. Somit bekommt das Tool die Informationen, die durch zuvor zugeordneter Software bereitgestellt wird. Im einfachsten Fall kann jedes Gerät direkt seine Daten an das Modellierungstool übermitteln, wie es in Abbildung 4.3 im dritten Abschnitt mit den Monitoringdaten dargestellt ist. Diese Art der Kommunikation kann für

manche Anwendungsfälle ausreichend sein. Da der Benutzer in vielen Szenarien weiterführendes Interesse an den Daten hat, können diese auch an andere Empfänger, wie beispielsweise eine Datenbank übermittelt werden, wie es in Abbildung 4.4 zu sehen ist. Dies bietet den Vorteil, dass das Modellierungstool besser an unterschiedliche Systeme angepasst werden kann, da alle Informationen immer in gleicher Form aus der Datenbank ausgelesen werden können. Für echtzeitkritische Systeme stellt der Umweg über eine Datenbank möglicherweise einen zu großen Zeitverlust dar, bis die Daten in der Benutzeroberfläche der Monitoringansicht des Modellierungstools angezeigt werden. In anderen Umsetzungen können die Daten sowohl an das Modellierungstool, aber auch an weitere Anwendungen oder eine Datenbank geschickt werden. Dies hat wiederum den Nachteil einer höheren Netzwerkauslastung, da das IoT-Gerät die doppelte Anzahl an Nachrichten übermitteln muss. Jede dieser exemplarischen Varianten muss daher zu dem Anwendungsgebiet passen. In diesem Konzept wird eine Kontextdatenbank verwendet, die alle gesammelten Daten den Komponenten zugeordnet speichert.

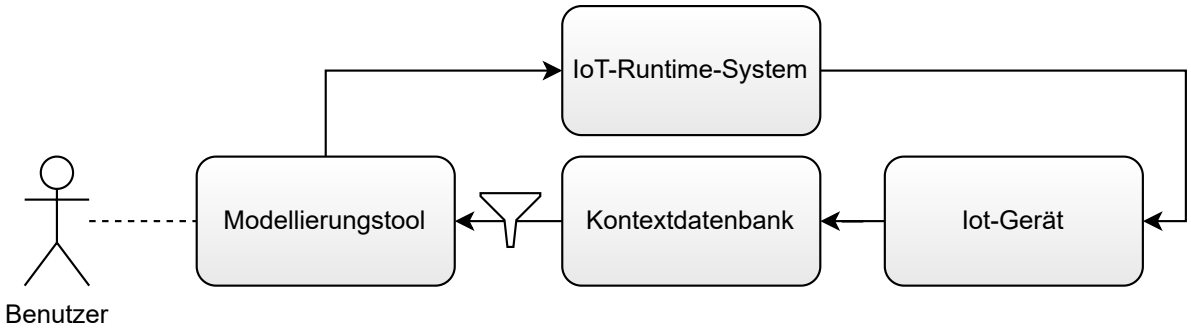

Abbildung 4.4.: Datenübertragung mit Kontextdatenbank

Alle Daten der Geräte werden mittels angepassten Adaptern eingelesen und stehen somit in einer, für das Modellierungstool nutzbaren Form auf Anfrage zur Verfügung. Daher können auch zukünftig weitere neue Komponenten, die auf den gleichen Daten arbeiten, ebenfalls einfach auf diese Daten zugreifen. Da durch das Nutzen einer Datenbank auch der Verlauf der IoT-Umgebung abbildbar ist, können ebenfalls weitere Analysen, basierend auf Zeitreihendaten, erfolgen. Alle Daten stehen in einer einheitlichen, vordefinierten Form zu Verfügung [GABS22]. Daher kann ab diesem Zeitpunkt die Analyse der Daten angewendet werden. Vom Benutzer definierte Filter können direkt beim Auslesen der Daten aus der Datenbank genutzt werden. In den Anfragen kann direkt nur nach den Werten gefragt werden, die die vom Benutzer geforderten Eigenschaften erfüllen oder alle neuen Daten aus der Datenbank werden abgefragt, um den aktuellen Systemkontext festzustellen. Um komplexe Filterregeln anzuwenden, sind oft ganze Datensätze nötig, da Werte nicht einzeln analysiert werden, sondern beispielsweise der Verlauf auf unrealistische Sprünge überprüft wird [HRM19]. Jedoch ermöglichen viele Datenbanken, beispielsweise durch SQL-ähnliche Anfragen, schon für viele Fälle ausreichend mächtige Filteroptionen. In jedem Fall sollen die vom Benutzer definierten Filter angewendet werden können, da dies auch ein weiterer Automatisierungsschritt der Analyse ist. Sonst

müsste der Benutzer manuell auf den Rohdaten arbeiten und diese auf ihre Korrektheit und Aussagekraft selbst überprüfen.

4.4. Architektur

Um das Modellierungstool für ein lauffähiges System zu verwenden, müssen Schnittstellen bereitgestellt und zugeordnet werden. Diese Architektur realisiert die zuvor vorgestellten Konzepte und ist damit Grundlage für die Implementierung des in Abschnitt 5 beschriebenen Modellierungstools. Die vorgestellte Architektur in Abbildung 4.5 soll hierbei die notwendigen Anforderungen aus Abschnitt 4.1 erfüllen, aber auch flexibel für Weiterentwicklung sein. Dies ist wichtig, da zum Beispiel die Daten aus der Umgebung noch weiterverwendet werden können und weitere Analysen oder Funktionalitäten hinzugefügt werden können. Daher enthält diese Architektur die Kontextdatenbank, welche für den Benutzer den Systemkontext speichert, aber nicht für die minimalen Anforderungen notwendig ist [GABS22]. Genaueres über die Kontextinformationen wird im folgenden Abschnitt 4.4.1 Schnittstelle Modellierungstool - Kontextdatenbank beschrieben.

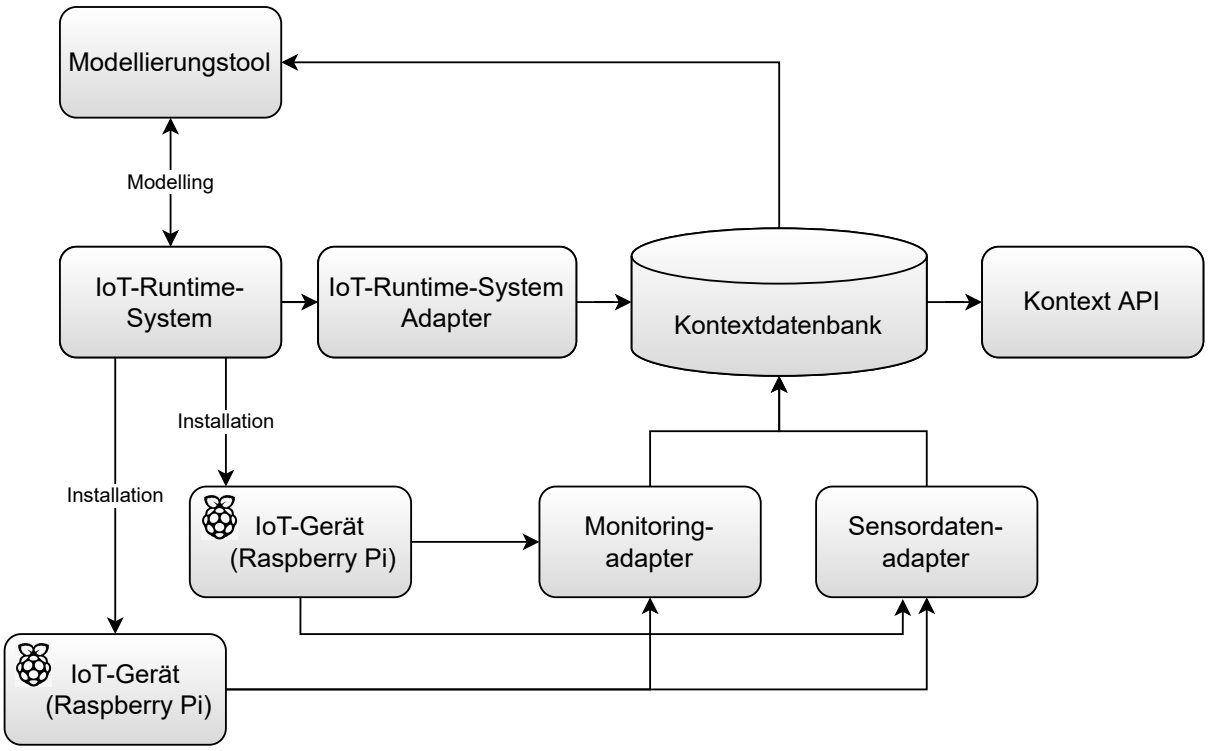

Abbildung 4.5.: Architektur zur Integration des Modellierungstools

4.4.1. Integration des Modellierungstools

Die Integration des Modellierungstools soll das gesamte bisherige System verbessern und Funktionen der anderen Bestandteilen der Architektur verbinden und nutzen. Um dies zu realisieren, sind an dieser Stelle zunächst die direkten Schnittstellen des Modellierungstools zu beschreiben. Die Schnittstellen stellen die Anforderungen für die Integration, wodurch jedes System, dass die Anforderungen erfüllt, mit dem Modellierungstool verbunden werden kann.

Schnittstelle Modellierungstool - Iot-Runtime-System

Da nicht alle Daten direkt in das Modellierungstool selbst eingelesen werden können, wird dafür ein IoT-Runtime-System verwendet [DH20b] [FHS+20]. Das Modellierungstool soll nur die bereits angelegten Komponenten modellieren und zusammenführen und nicht die Komponenten aus dem IoT-Runtime-System erneut anlegen. Durch ein solches IoT-Runtime-System stehen dem Modellierungstool die im Netzwerk registrierten Geräte für die Modellerstellung zur Verfügung. Außerdem werden alle Softwarekomponenten in das IoT-Runtime-System hochgeladen, um auch diese für die Modellierung verwenden zu können. Daher bekommt das Modellierungstool wichtige Informationen aus dem IoT-Runtime-System, die die Grundlage der Modelle darstellen. Diese Informationen aus dem IoT-Runtime-System müssen nach Bedarf aktualisiert werden können, da die Geräte oder auch die Software verändert werden können. Sonst werden möglicherweise Geräte, die nicht mehr zur Verfügung stehen sollen, in Modelle eingebunden, was zu Fehlern führen kann. Allerdings ist diese Kommunikation zwischen dem Modellierungstool und dem IoT-Runtime-System in beide Richtungen wichtig, da auch Funktionen des IoT-Runtime-Systems nach der Modellierung verwendet werden können. Nach der Modellierung aller Teilmodelle im Modellierungstool, wie in Abbildung 4.2, steht eindeutig fest, welche Software auf welches Gerät installiert werden soll. Daher werden diese Zuordnungen wieder zurück an das IoT-Runtime-System übermittelt, sodass die Installation und das Ausführen der Softwarekomponenten über das IoT-Runtime-System automatisiert erfolgen kann.

Schnittstelle Modellierungstool - Kontextdatenbank

In dieser Architektur wird eine Datenbank verwendet, welche den gesamten Kontext der Umgebung speichert. Daher kommuniziert das Modellierungstool nicht direkt mit jedem Geräte einzeln, sondern kann alle gesammelten Daten gebündelt aus der Kontextdatenbank auslesen. Es ist auch möglich, eine Architektur zu implementieren, die diesen Zwischenschritt über eine Kontextdatenbank nicht verwendet. Allerdings bietet die Verwendung der Datenbank Vorteile, da anhand des gespeicherten Systemkontexts weitere Funktionen in Zukunft hinzugefügt werden können. Das Vereinheitlichen der Daten in eine vordefinierte, flexible Struktur erleichtert es dem Modellierungstool, die Daten für das Monitoring zu verwenden [GABS22].

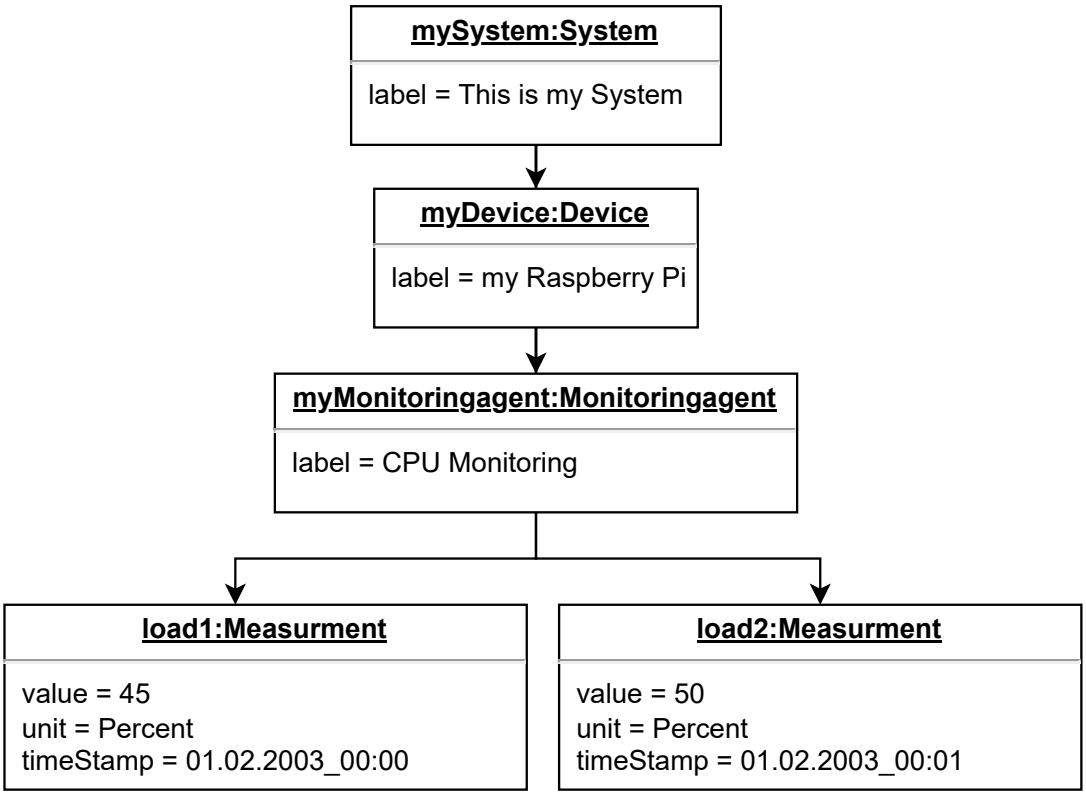

Abbildung 4.6.: Schematisches Beispiel für ein Kontextmodell

Die Datenbank, eine Kontextdatenbank, erlaubt es, den aktuellen Zustand abzurufen, der in der Regel von der Monitoringfunktion des Modellierungstools zur Überwachung benötigt wird. Außerdem kann der Verlauf vergangener Messdaten verfolgt und analysiert werden. Zur Effizienzsteigerung kann ein Kontextmodell verwendet werden, dass nur die Metainformationen der IoT-Umgebung speichert. Für die Echtzeitdaten werden dann Zeitreihendatenbanken verwendet, da dadurch effizientere Speicherung ermöglicht wird. In dieser Architektur wurde der Einfachheit halber darauf verzichtet, da das Kontextmodell aus „A live context model for semantic reasoning in IoT applications“ [GABS22] alle erforderlichen Informationen enthält und somit alle Daten der IoT-Umgebung gemeinsam verfügbar gemacht werden. Das Modellierungstool sendet in einem gewünschten Intervall Anfragen an die Kontextdatenbank, um die neuesten verfügbaren Daten zu bekommen. Hier können die Anfragen für jede Komponente angepasst werden, um immer die gewünschten Informationen zu erhalten. Dies ist vom Modellierungstool aus leicht auszuführen, da durch das Kontextmodell der Kontextdatenbank vordefiniert ist, in welche Form und Struktur die Daten bereitgestellt werden, wie es in dem vereinfachten Beispiel in Abbildung 4.6 zu sehen ist. Wenn im alternativen Architekturansatz eine solche Datenbank nicht verwendet wird, müssen die Daten immer in korrekter Form direkt an das Modellierungstool übermittelt werden. Dann hat das Modellierungstool die Aufgabe, mit den Fehlern hierbei umzugehen. Für jede Art der Datenquelle muss daher garantiert werden, dass die Daten lesbar und somit verwendbar für das Modellierungstool sind. Daher

bringt die Kontextdatenbank den Vorteil, dass das Modellierungstool weniger abhängig von der Umgebung ist und damit leichter für verschiedene Umgebungen verwendet werden kann.

Kontextdaten einlesen

Um die Daten in die Kontextdatenbank strukturiert einzufügen, werden Adapter benötigt. Die Adapter lesen zunächst die statischen Daten des IoT-Runtime-Systems ein und fügen darauf aufbauend zu jeder Komponente die dazugehörigen Echtzeitdaten hinzu. Wie genau die Adapter realisiert werden, hängt von der Art des Kontextmodells ab, aber auch in welcher Form und über welche Schnittstelle diese abgerufen oder empfangen werden können. Dies kann beispielsweise durch Anfragen an eine Datenbank geschehen, welche in diesem Fall zwischen Adapter und IoT-Gerät eingefügt werden müsste, oder auch durch einen MQTT-Broker, welcher die Echtzeitdaten entgegennimmt. Die Monitoringdaten werden dabei von Monitoringagenten auf den Geräten ausgelesen, sodass diese Echtzeitdaten mithilfe der passenden Adapter in die Kontextdatenbank eingelesen werden und von der Monitoringfunktion im Modellierungstool verwendet werden können. Die zuvor beschriebene Schwierigkeit, dass die Daten in passender Form bereitgestellt werden müssen, wird an dieser Stelle durch ein Kontextmodell behandelt. Die Schwierigkeit wird dabei nicht umgangen, da die Daten an dieser Stelle der Architektur für die Kontextdatenbank in die passende Form des Kontextmodells gebracht werden müssen. Jedoch ist dieses Problem dadurch losgelöst vom Modellierungstool. Dieses kann ohne Störungen weiterarbeiten, falls Daten nicht passend bereitgestellt werden, da es aus Sicht des Tools Fehler durch fehlerhafte Form der Daten nie gab und es lediglich keine neuen Daten gibt. Die Datenbank hat außerdem ein bekanntes, bereitgestelltes Modell als Grundlage [GABS22]. Dadurch eignet sich die Datenbank besser zur Vereinheitlichung der Daten als das Modellierungstool, welches jede Antwort auf seine Anfragen selbst auf Korrektheit überprüfen müsste. Außerdem stehen die Daten aus der Datenbank für mögliche weitere Komponenten zu Verfügung, ohne eine komplett neue Kommunikation von jedem IoT-Gerät einzeln zu der neuen Komponente herstellen zu müssen. Somit ist diese Variante mit einer Kontextdatenbank eine Option, die für die meisten Systeme von Vorteil ist. Allerdings kann in spezifischen Fällen auch die direkte Kommunikation implementiert werden, beispielsweise falls die Daten nicht weiter benötigt werden, nachdem sie im Monitoringtool angezeigt wurden und daher nicht gespeichert werden müssen.

4.4.2. Struktur der Modellerstellung und des Monitorings

Der vorgestellte Ablauf, den ein Benutzer in dem Monitoringtool durchläuft, soll schrittweise an das Modell heranführen. Trotzdem ist dabei zu beachten, nicht die Flexibilität einzuschränken, wie das zu erstellende Modell aussehen soll. Daher wird im folgenden Abschnitt beschrieben,

wie der Ablauf des Modellierungstools aussieht und an welcher Stelle die Erweiterung, um Monitoring hinzuzufügen, integriert werden soll. Der gesamte Ablauf ist zusätzlich in Abbildung 4.7 dargestellt.

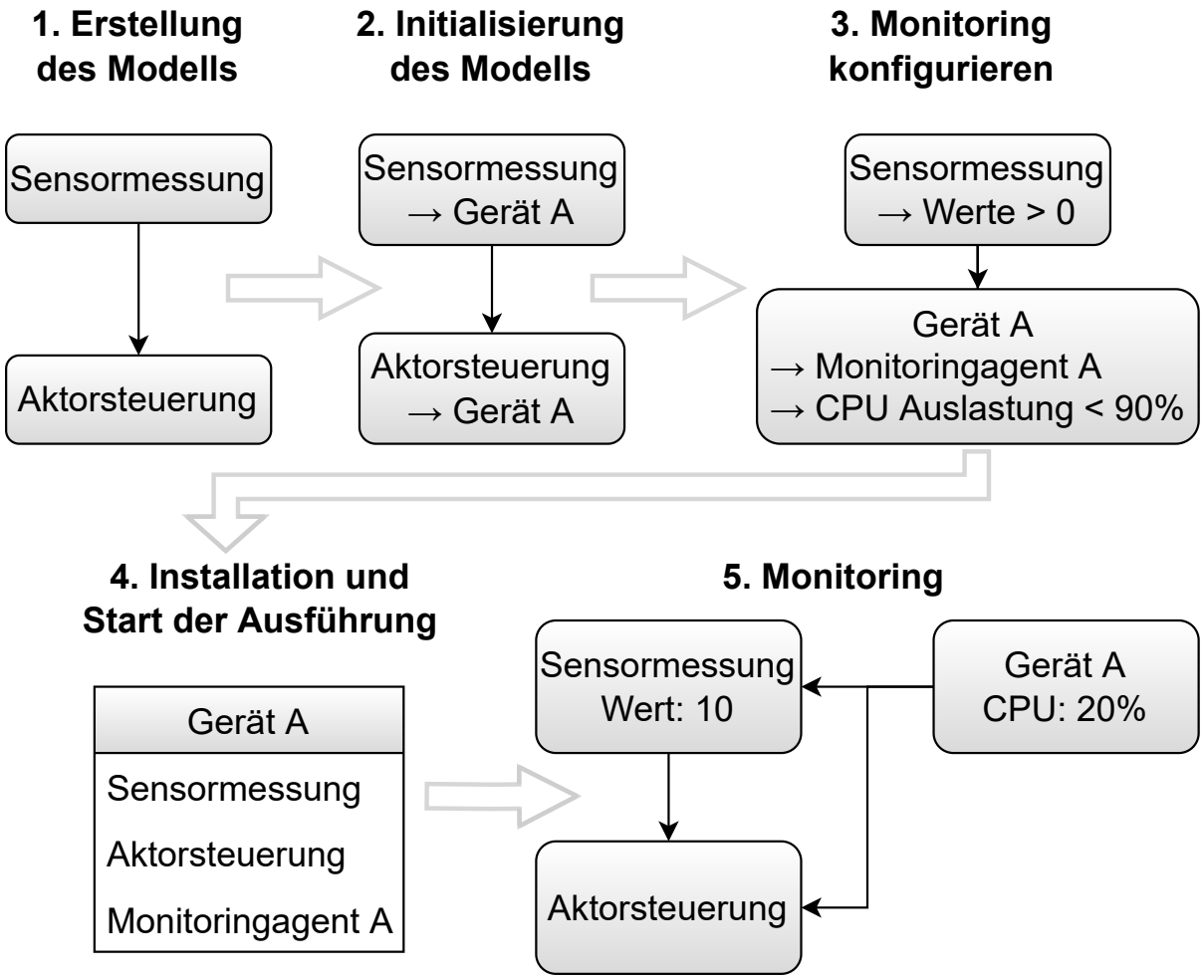

Abbildung 4.7.: Exemplarischer Ablaub Modellerstellung und Monitoring

Schritt 1: Erstellung des Modells

Der Benutzer startet das Modellierungstool mit der Erstellung des Modells, indem er die gewünschten Funktionalitäten in der Umgebung auswählt. Dafür verwendet der Benutzer beispielsweise Sensorskripte zum Auslesen von Messwerten oder Programme zur Steuerung von Aktoren. Der Benutzer fügt alle Sensorskripte und andere funktionalen Softwarekomponenten zu dem Modell hinzu. Die funktionalen Softwarekomponenten umfassen alle Programme, die Aufgaben erfüllen, um die gewünschten Aktionen oder Messungen in der IoT-Umgebung

auszuführen. Programme, die nur zur Überwachung des Systems erforderlich sind, werden in Abschnitt 4.4.2 optional zum Modell hinzugefügt. Beim Hinzufügen der Softwarekomponenten in der Erstellung des Modells können weitere Informationen mitgegeben werden, welche die Komponente spezifizieren oder kategorisieren. Beispiele hierfür wären das Hinzufügen von Beschreibungen oder die Einordnung, ob es sich um eine Software für einen Sensor oder Aktor handelt. Wenn alle Informationen in diesem Abschnitt eingegeben wurden, ist die gesamte Funktionalität der Umgebung beschrieben. Allerdings, ist die Initialisierung des Modells mit verfügbarer Hardware nötig, um diese Funktionalität in der IoT-Umgebung umzusetzen.

Schritt 2: Initialisierung des Modells

Wenn der Benutzer im Initialisierungsabschnitt ist, initialisiert er das Modell so, dass jede zuvor hinzugefügt Software einem Gerät zugewiesen wird, um diese auszuführen. Dieser Abschnitt dient der eindeutigen Zuweisung der Softwarekomponenten zu der Hardware, da nicht jedes Gerät gleiche Messungen vornehmen soll oder überhaupt die Möglichkeit hat, die Skripte oder Programme auszuführen. Dafür bekommt der Benutzer die in dem IoT-Runtime-System registrierten Geräte angezeigt und kann somit die Software auf alle Geräte verteilen. Dieser Abschnitt ist getrennt von der Erstellung des Modells mit den Softwarekomponenten, da so erst definiert werden kann, welche Funktionen in seiner IoT-Umgebung bereitgestellt werden sollen. Erst dann wird die Umsetzung der Funktionen modelliert, da schließlich die geforderte Funktionalität der Grund für die Umsetzung des Systems ist. Somit kann der Benutzer die Übersicht haben, welches Gerät wofür verwendet wird und gegebenenfalls schnell Änderungen vornehmen, falls ein Gerät überlastet wird und für zu viele Programme benötigt wird. Bei diesen Änderungen ist trotzdem das gleiche Modell die Grundlage, wodurch automatisch sichergestellt ist, dass die Funktionalität sich nicht ändert.

Schritt 3: Monitoring konfigurieren

Da bereits definiert ist, was in der IoT-Umgebung passieren soll und wie dies umgesetzt wird, kann diesem Schritt das Monitoring hinzugefügt werden. In diesem Abschnitt, welcher den Kernbestandteil dieser Arbeit beschreibt, wird die Option gegeben, in dem betreffenden System Monitoringsoftware hinzuzufügen. Dies ist optional, da es nicht die Funktionalität erweitert, sondern nur der Überwachung dient. Wie wichtig die Überwachung eines Systems ist und welche Arten von Fehlern eintreten und somit erkannt werden können, wird in Abschnitt 4.2 beschrieben. Das Monitoring wird in zwei Kategorien aufgeteilt, da einerseits die Rückgabewerte der bereits verwendeten Software analysiert werden, aber auch weitere Komponenten hinzugefügt werden können, welche zusätzliche Eigenschaften der Geräte messen oder auslesen. Dies deckt verschiedene Bereiche der Fehlererkennung ab. Die Informationen, die dem Modellierungstool sowieso schon zur Verfügung stehen, werden für das Monitoring verwendet, aber auch neue Informationen, die gezielt gesammelt werden. Die Messwerte, die als ein Teil der ursprünglichen Funktionalität gesammelt werden, haben geringere Aussagekraft über die

Auslastung der Hardware, aber können auch selbst Fehler aufweisen. Somit werden nicht die Geräte durch die Messwerte überprüft, sondern die Werte selbst auf Korrektheitskriterien getestet. Damit kann die Qualität der Messungen verbessert werden, da weniger fehlerhafte Daten ausgegeben werden. Da hierfür keine weitere Komponente auf den Geräten installiert werden muss, sondern nur eine Definition der Korrektheitskriterien gefordert ist oder wie diese überprüft werden sollen, bekommt der Benutzer die Option, Regeln zu definieren. Diese Regeln müssen für jede Komponente individuell konfigurierbar sein, da Rückgabewerte von unterschiedlichen Sensoren unterschiedliche Formen haben können und andere Kriterien erfüllen müssen, um diese als korrekt anzusehen.
Die andere Art des Monitorings zur Überprüfung der Geräteauslastung muss in diesem Abschnitt mittels weitere Software auf den Geräten umgesetzt werden. Diese Software kann der Benutzer zu jedem Gerät, das überwacht werden soll, zuordnen. Diese Monitoringagenten messen oder lesen verschiedene Eigenschaften der Geräte, an denen Überlastungen erkannt werden können, aus. Wie auch bei der Zuordnung der Geräte zu den Softwarekomponenten hat der Benutzer alle im IoT-Runtime-System registrierten Monitoringkomponenten zur Auswahl, um diese zu verwenden. Hierbei ist zu beachten, dass viele funktionale Softwarekomponenten zu einem Gerät zugeordnet werden können, aber das Monitoring der Hardware nur einmal für das Gerät geschehen muss. Daher darf eine Monitoringsoftware nur direkt zum Gerät und nicht zu den anderen Softwarekomponenten zugewiesen werden. Auch für diese Werte kann der Benutzer mit Regeln definieren, vor welchen Werten er gewarnt werden möchte, um für jedes Gerät individuelle Toleranzen festlegen zu können, falls diese unterschiedlich hohe Garantien an die Ausfallsicherheit erfüllen müssen. Damit ist das Modell schließlich vollständig definiert und kann verwendet werden.

Schritt 4: Installation und Start der Ausführung

Durch das Deploymentmodell können alle Anwendungs- und Monitoringkompoenten auf den Geräten autoamtisch installiert werden. Daher wird dieser Schritt mit dem abschließen der Konfiguration des Monitorings, oder fall kein Monitoring verwendet werden soll nach der Initialisierung des Modells gestartet. Dafür wird für jede Software überprüft welches Gerät zugewiesen ist. Diese Zusweisungen werden an das IoT-Runtime-System übermittelt. Dort wird damit der Start der Installation, wie in Abbildung 4.5, ausgelöst. Dafür werden zunächst alle notwendigen Dateien auf die Geräte kopiert und die installation wird auf dem Gerät gestartet. Nach der Fertigstellung der Installation wird ebenfalls autoamtisch die Software selbst gestartet, so dass die Funktionalität der Anwendung und das Monitoring ausgeführt wird.

Schritt 5: Monitoring

In diesem finalen Schritt erhält der Benutzer eine Übersicht über das Modell. In dieser Übersicht sind alle zuvor im Modellierungstool erstellten Komponenten zu sehen, um ihren aktuellen

Status in der Umgebung wiedergeben zu können. Sobald die Installation aus dem vorherigen Schritt fertiggestellt ist können Daten empfangen werden. Dabei werden die von ihm definierten Regeln geprüft, um den Benutzer, wenn nötig, vor einem Regelbruch zu warnen. Die Warnungen sollen hierbei eindeutig den Geräten zugewiesen werden können. Dies muss leicht nachvollziehbar dargestellt werden, da im Falle eines gesamten Systemversagens viele Fehlermeldungen gleichzeitig eintreten können, wodurch dies sonst schnell unübersichtlich für den Benutzer wird. Solange keine automatisierten Reaktionen auf registrierte Fehler verwendet werden, muss der Benutzer selbst erkennen können, was manuell zu machen ist, um erkannte Fehler zu beheben. Aktuelle Informationen über Erreichbarkeit der Komponenten müssen ebenfalls immer bereitgestellt werden, da diese Information Grundlage für weitere empfangene Daten ist. Wenn ein Gerät nicht im Netzwerk verfügbar ist, können keine weiteren Messwerte erwartet werden. Auch wenn dieser Zusammenhang klar ist, sollte dies dem Benutzer so dargestellt werden, dass auch intuitiv erkannt wird, dass präsentierte Werte nicht mehr aktuell sein können und somit nicht die gleiche Bedeutung haben wie Messwerte anderer verfügbarer Komponenten. Trotzdem sollten die somit veralteten Daten noch einsehbar bleiben, da beispielsweise ein gerade erkannter Fehler Auslöser oder Anzeichen für die verlorene Verbindung sein könnte. Für alle diese Daten gilt, dass sie stets aktuell gehalten werden müssen, da sonst Probleme zu spät erkannt werden oder auch vor Problemen, die nicht mehr bestehen, immer noch gewarnt wird.

5. Implementierung

In diesem Kapitel wird die prototypische Implementierung beschrieben. Zuerst wird in Abschnitt 5.1 das IoT-Application-Modeling-Tool (IAMT) vorgestellt, auf welchem die Implementierung zu dieser Arbeit aufbaut. Dann wird beschrieben, wie die Erweiterung der Monitoringmodellierung und Monitoringübersicht integriert wird und wie diese an dem Ablauf der Anwendungsmodellerstellung anknüpft. Danach werden in den folgenden Abschnitten alle weiteren Komponenten vorgestellt, die für die Integration und Realisierung der Monitoringerweiterung verwendet werden.

5.1. Modellierungstool

Das IoT-Application-Modelling-Tool (IAMT) ermöglicht Benutzern das Erstellen von Modellen für IoT-Umgebungen. Das Tool wird mit dem Framework Angular[1] erstellt und ist damit als Web-Anwendung verfügbar. Dieses Tool wurde in dieser Arbeit erweitert, da bisher Modellerstellung und Realisierung der Anwendung möglich sind und die Modellierung von Monitoringlösungen hinzugefügt werden soll. Durch eine grafische Benutzeroberfläche kann der Benutzer mit dem Tool interagieren, um neue Modelle zu erstellen oder bestehende zu bearbeiten. Diese Modelle können nach der Modellierung verwendet werden, um sie automatisiert in der Umgebung zu realisieren. Für eine bessere Kontrolle der modellierten IoT-Umgebungen wird das Tool im Rahmen dieser Arbeit um eine Monitoringfunktion erweitert. Somit kann Monitoringsoftware in das Modell integriert werden, welches ebenfalls in dem Modellierungstool erstellt wurde. Nach der Modellierung wird das gesamte Modell automatisiert in der IoT-Umgebung realisiert. Dafür werden die Funktionen zur Installation und Ausführung der Softwarekomponenten des IoT-Runtime-Systems verwendet. Dies wird in dieser Implementierung durch REST-Anfragen an die API des IoT-Runtime-Systems ungesetzt. Dadurch soll die gewünschte Funktionalität gewährleistet und zudem das gesamte System überwacht werden können. Die einzelnen Bereiche, die vom Beginn der Modellerstellung bis zur Ausführung nötig sind, werden in den folgenden Abschnitten beschrieben.

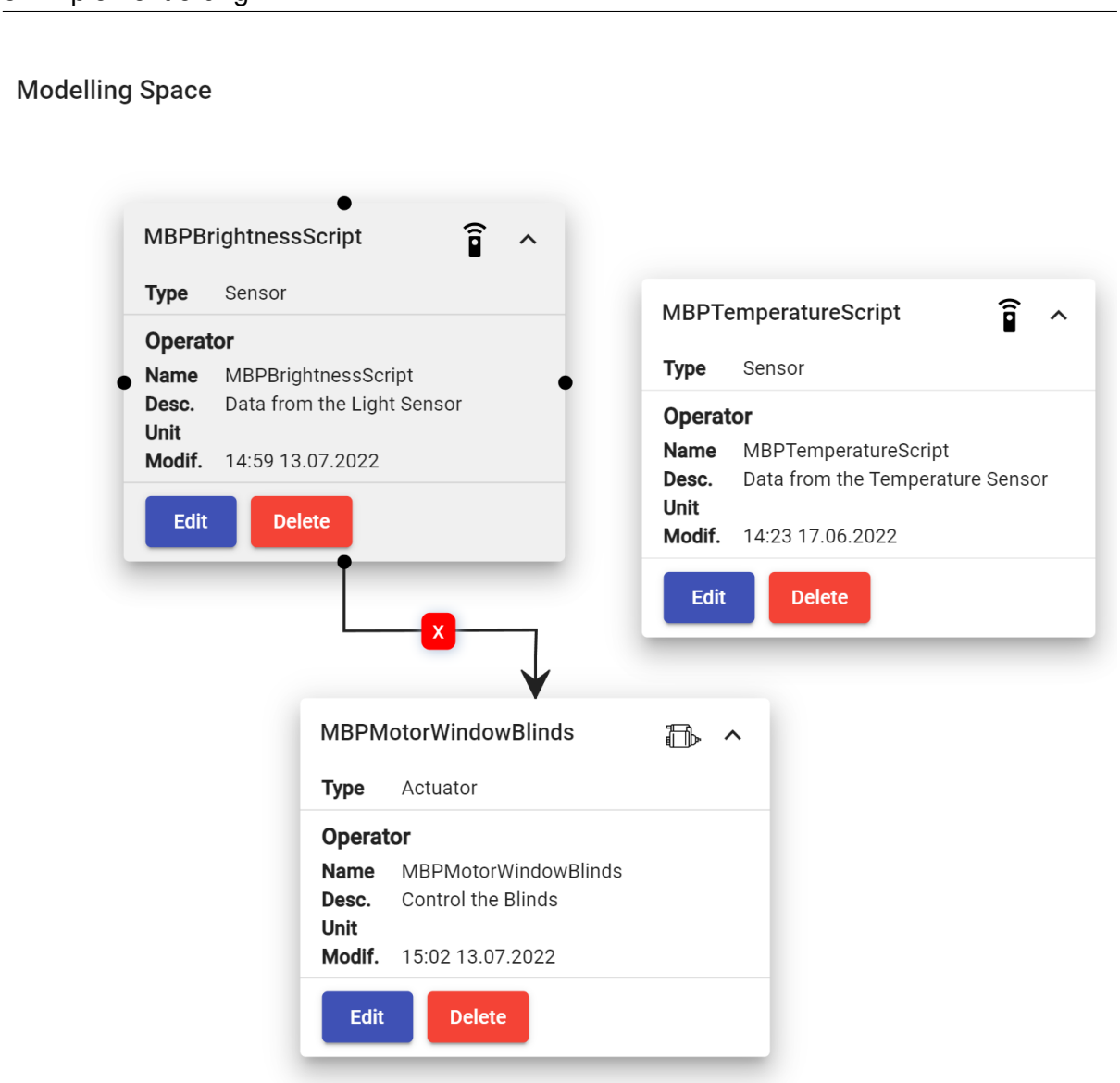

Abbildung 5.1.: Anwendungsmodell - Erstellung

5.1.1. Modellerstellung

Nachdem der Benutzer ein neues leeres Modell geöffnet hat, kann er, wie in Abbildung A.1 im Anhang zu sehen ist, aus der Auswahl an Operatoren auf der linken Seite die Gewünschten auswählen, um sie wie in Abbildung 5.1 dem Modell hinzuzufügen. Diese Operatoren wurden zuvor im IoT-Runtime-System (hier MBP) registriert [FHS+20]. Operatoren können zudem in dem Modell verknüpft werden, um Zusammenhänge zu signalisieren. Beim Hinzufügen der Operatoren bekommt der Benutzer zusätzlich die Option, weitere Informationen in einem

[1]https://angular.io/

Dialogfenster einzugeben. Dazu können beispielsweise einfache Beschreibungen für den Benutzer gehören. Außerdem können Operatoren entweder für Sensoren oder Aktoren verfügbar gemacht werden.

5.1.2. Initialisierung der Modelle

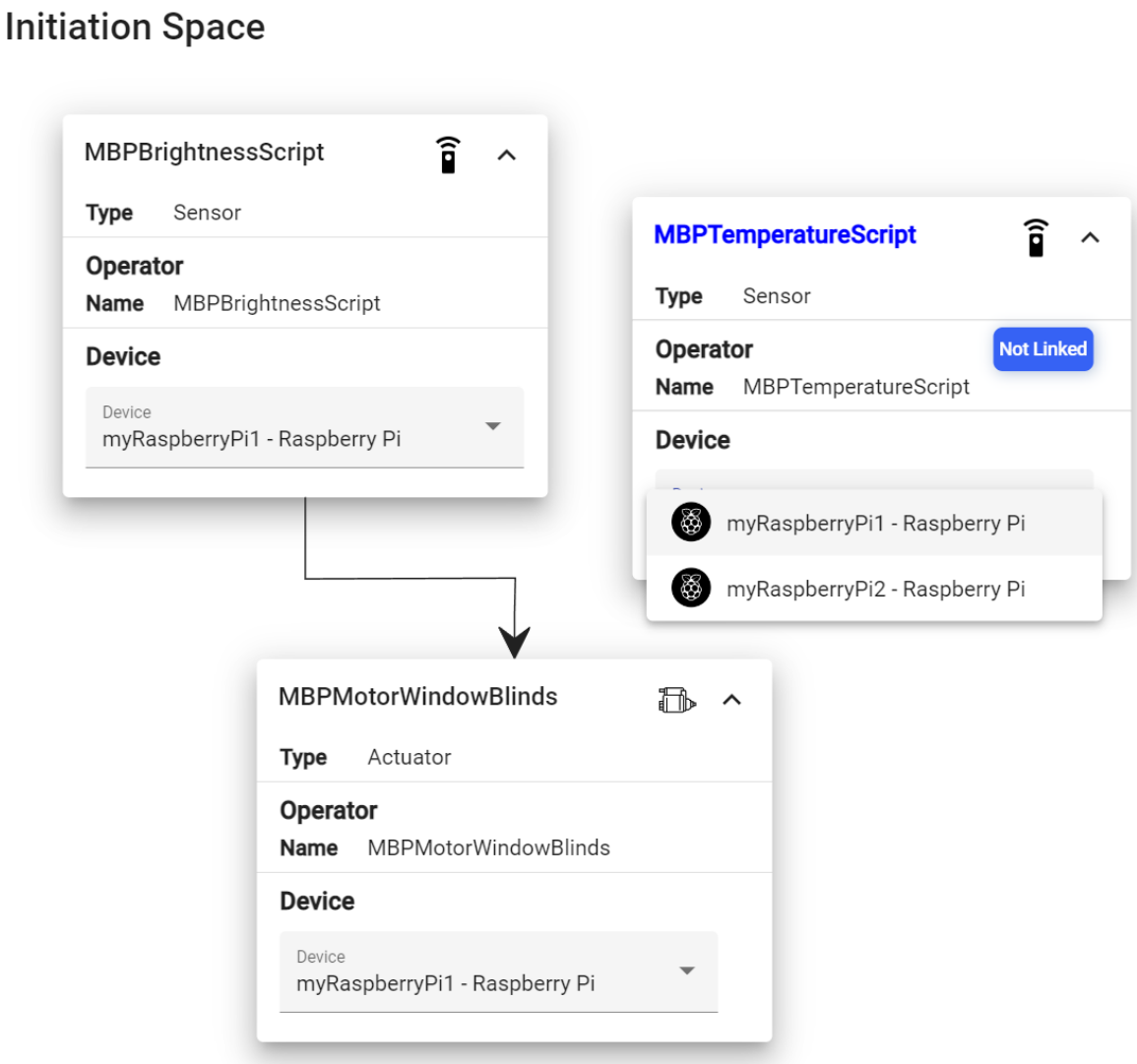

Abbildung 5.2.: Deploymentmodell - Initialisierung

Da bis jetzt nur alle Softwarekomponenten, welche die Funktionalität der Anwendung beschreiben, zum Modell in der Erstellung hinzugefügt wurden, wird jetzt jeder Softwarekomponente ein Gerät zugewiesen. Erst damit wird definiert, von welchem Gerät die Messungen oder

Datenverarbeitungen übernommen werden. Außerdem kann die Software gleichmäßig verteilt werden, damit Ressourcen der Geräte verwendet, aber auch diese nicht überlastet werden. Dem Benutzer werden hier alle zuvor im IoT-Runtime-System registrierten Geräte zur Auswahl gegeben. Jeder Knoten wird exakt einem Gerät zugewiesen, da die Software ohne Recheneinheit keine Funktionalität bieten kann und kein Teil der IoT-Umgebung wäre. Für Szenarien, in denen von mehreren Geräten die gleichen Operatoren ausgeführt werden sollen, können diese in der Modellerstellung in Abschnitt 5.1.1 mehrfach hinzugefügt werden. Dadurch werden in der Initialisierung keine Operatoren hinzugefügt, sodass die Funktionalität der Umgebung nach der Anwendungsmodellerstellung nicht verändert wird und jeder Operatorknoten genau einem Gerät zugewiesen wird. Erst wenn alle Knoten ein Gerät zugewiesen bekommen haben, hat der Benutzer die Möglichkeit das Modell umzusetzen und dafür die Software zu installieren und zu starten. An dieser Stelle kann jedoch der Benutzer die in dieser Arbeit hinzugefügte Option nutzen, indem er Monitoring für sein Modell hinzufügt. Wenn dies gewählt wird gelangt er zu der „Monitoring initialisieren“-Ansicht, die in Abbildung 5.3 zu sehen ist und im folgenden Abschnitt 5.1.3 erklärt wird. Alternativ kann ohne Verwendung von Monitoring nach diesem Initialisieren Abschnitt die Installation bereits gestartet werden.

5.1.3. Monitoring konfigurieren

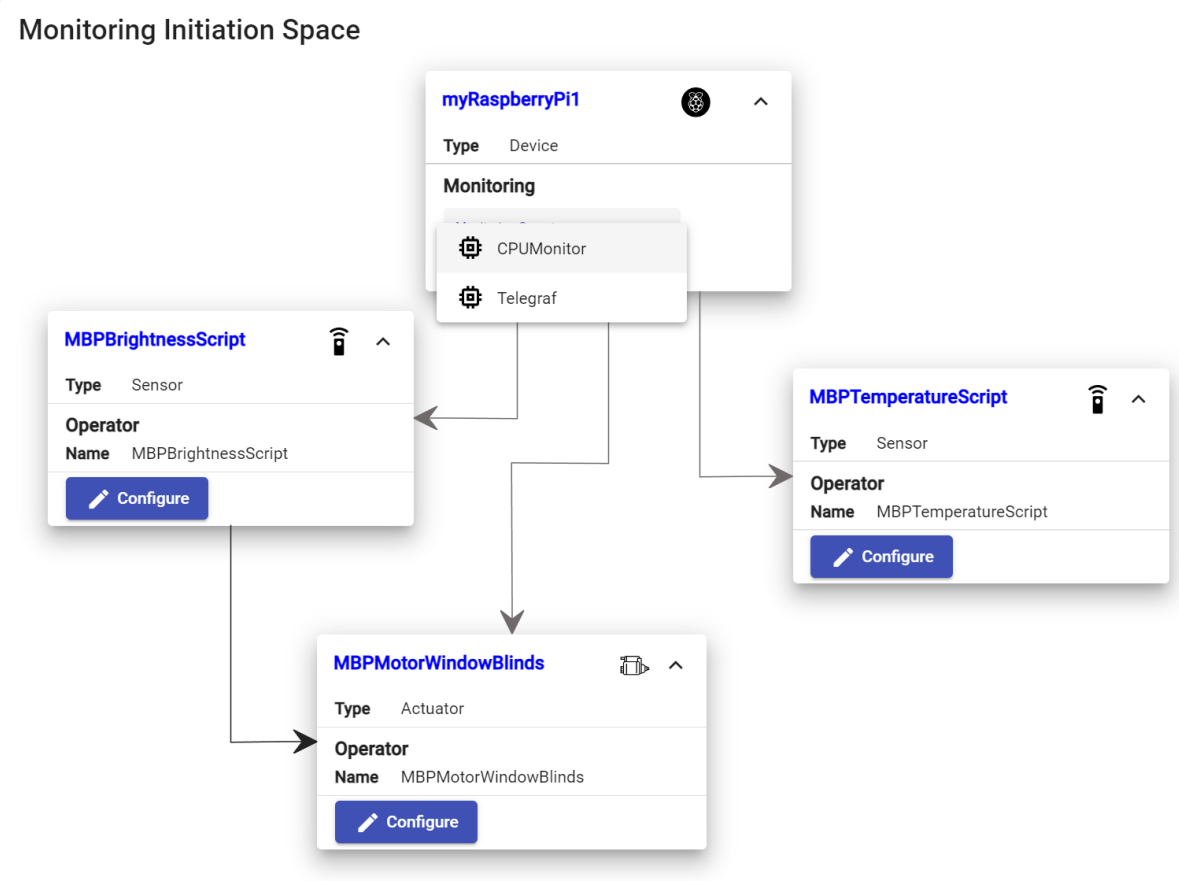

Abbildung 5.3.: Monitoringmodell - Monitoring konfigurieren

Dieser Teil ist optional auswählbar. Der Benutzer kann für eine beliebige Auswahl an Komponenten, welche sowohl Software als auch Hardware beinhalten kann, weitere Monitoringeigenschaften überprüfen. Durch den "Configure" Knopf kann bei Softwarekomponenten direkt eine Regel erstellt werden. Damit gelangt der Benutzer zu der Regelerstellung aus Abbildung 5.4. Bis jetzt sind Geräte keine eigenständigen Knoten im Modell. Jedoch soll in diesem Schritt das Monitoring sowohl für Geräte, als auch für Software ermöglicht werden. Daher wird, um die Knoten nicht zu überladen, für jedes verwendete Gerät ein weiterer Knoten automatisch erstellt. Dies stellte in dem bisherigen Tool eine Schwierigkeit dar, da bisher nicht mehrere Knotenklassen vorgesehen waren. Da nun Geräteknoten benötigt werden, wird das Programm so weiter verändert, dass es eindeutig ist, ob gerade alle Knoten des Modells benötigt werden oder nur die bisherigen Softwarekomponenten oder Geräteknoten. Ohne diese Option würde ein Gerät in mehreren Knoten vorkommen können, welches ein unübersichtliches oder sogar uneindeutiges Monitoring erzeugen würde. Das Monitoring der Geräte unterscheidet sich in der Durchführung von der Überprüfung der Messwerte. Für Geräte aus dem Modell kann

eine weitere Monitoringkomponente installiert werden, um gezielte Hardwareeigenschaften auszugeben. Die ist gefordert, da dadurch wichtige Informationen über den Zustand der Geräte gesammelt werden können. Die Monitoringsoftware wurde hierbei ebenfalls wie Geräte und Sensor- oder Aktorsoftware im IoT-Runtime-System registriert.

5.1.4. Regelerstellung

In der Regelerstellung kann der Benutzer spezifizieren, nach welchen Kriterien die Werte überprüft werden sollen. Beispiele für Filter wurden in 4.2 beschrieben. Diese Option lässt trotzdem alle gemessenen Werte zu, um sie dem Benutzer anzuzeigen, jedoch werden die Werte mit einer Warnung markiert und dem Benutzer in der Monitoringübersicht angezeigt.

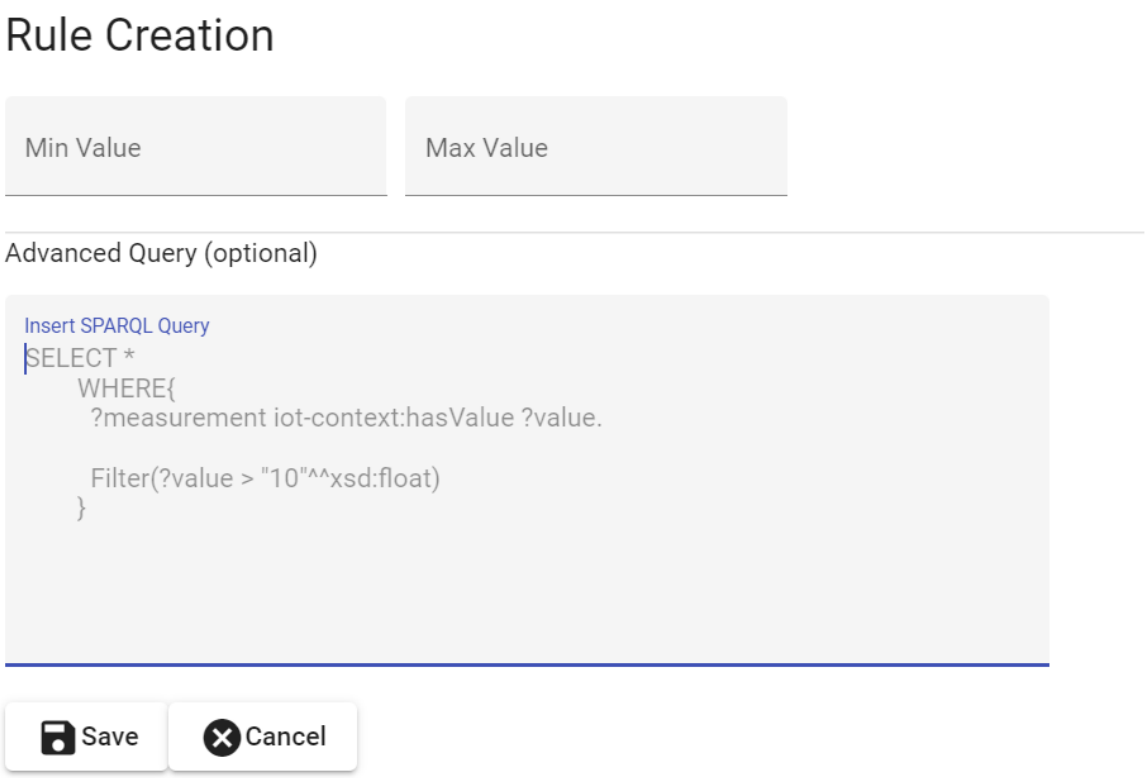

Abbildung 5.4.: Regelerstellung

Als Beispiel für einfache Regeln können die Werte der Messungen auf einen gültigen Wertebereich überprüft werden. Da die hier eingegeben Informationen in eine SPARQL-Query übersetzt werden, ist die optionale Auswahl durch selbst geschriebene SPARQL Queries möglich. Dies

soll gewährleisten, dass die gesamte Mächtigkeit der Sprache zur Verfügung steht und nicht durch Vereinfachung der grafischen Benutzeroberfläche die Funktionalität eingeschränkt wird. An dieser Stelle kann die Monitoringkomponente erweitert werden, um dem Benutzer mehr Optionen für Filter zu bieten, ohne das Wissen über SPARQL vorauszusetzen. Die erstellten Regeln werden dann für jeden Knoten gespeichert und während des Monitorings angewendet.

```
PREFIX iot-context: <http://www.uni-stuttgart.de/2021/iot-context#>
PREFIX iot-lite: <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#>

SELECT *
WHERE{
   ?Sensor iot-lite:id "sensorID".
   ?Sensor iot-lite:hasSensingDevice ?SensingDevice.
   ?SensingDevice iot-context:hasMeasurement ?measurement.
   ?measurement iot-context:hasValue ?value.
   ?measurement iot-context:hasTimeStamp ?timeStamp.

  Filter not exists {
      ?measurement2 iot-context:hasTimeStamp ?timeStamp2.
            ?SensingDevice iot-context:hasMeasurement ?measurement2.
    Filter(?timeStamp2 > ?timeStamp)
}

  Filter(?value > "10"^^xsd:float)
}
```

Listing 5.1: SPARQL Query - Aktuelle Daten filtern

5.1.5. Monitoring

Die Monitoringübersicht ist der Bereich, in dem das System nach dem Festlegen der Monitoringlogik überwacht werden kann. Es wird der Systemstatus anhand der zuvor definierten Regeln und Monitoringeigenschaften angezeigt. Der Benutzer muss nicht weiter in das Modellierungstool eingreifen, sondern diese Übersicht dient lediglich der Präsentation der Daten. Dies wird ermöglicht, indem alle Komponenten automatisiert installiert und gestartet werden, sobald der Benutzer mit der Modellierung fertig ist. Das System soll ab diesem Zeitpunkt selbstständig laufen und nur im durch Monitoringmeldungen erkannten Fehlerfall soll das Eingreifen eines Benutzers erforderlich werden. Sobald der Benutzer das Monitoringfenster verlässt und zu der Modellierungsumgebung zurückkehrt, werden die laufenden Programme auf den Geräten angehalten. Somit lässt sich alles wieder neu aufsetzen, da der Benutzer nun wieder Änderungen vornehmen kann. Da dies auf jedem Gerät parallel geschehen kann, ist das Neustarten der Umgebung kein großer zeitlicher Mehraufwand bis jedes Gerät wieder bereit

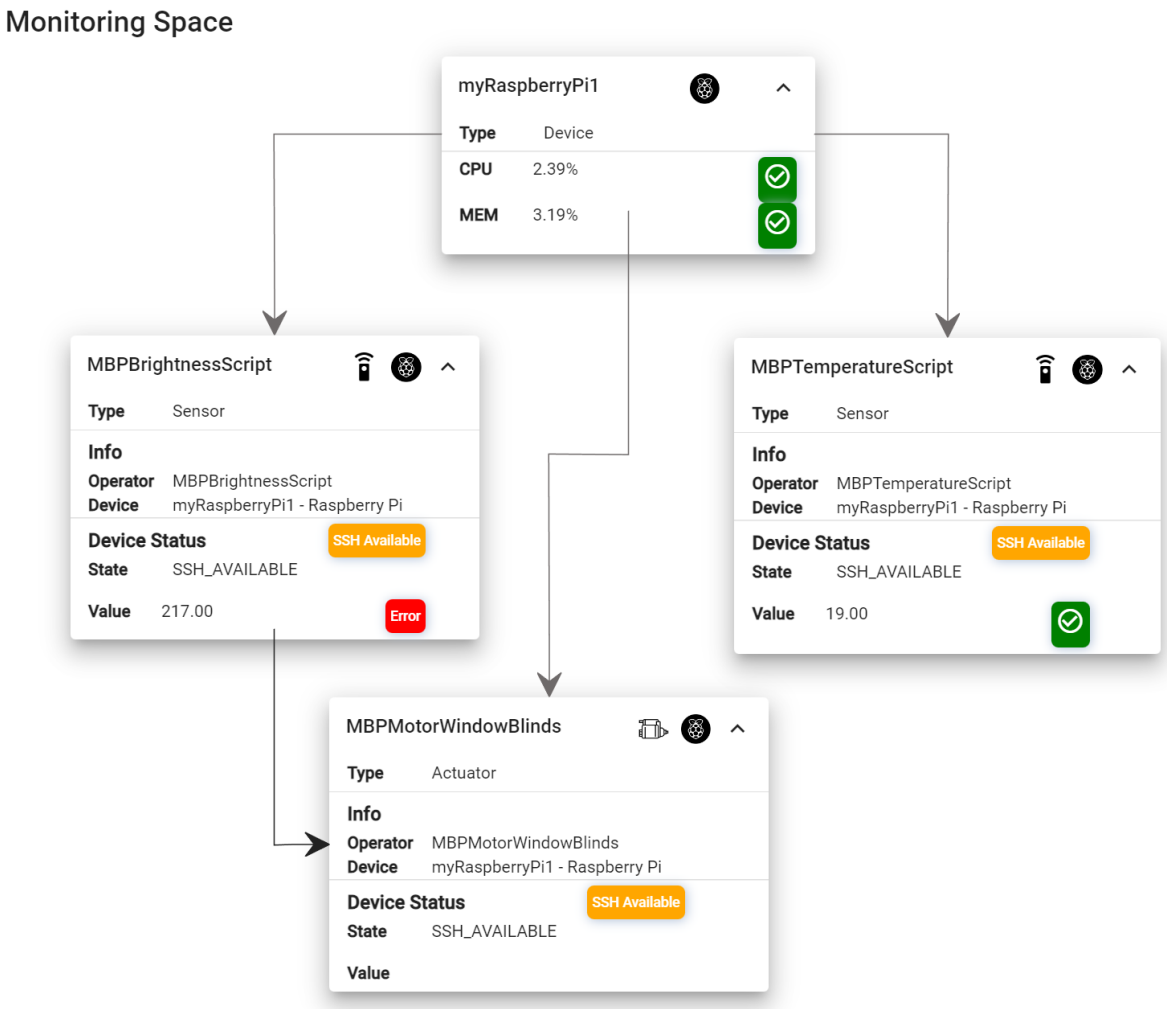

Abbildung 5.5.: Monitoring Übersicht

ist. Daher hängt der Mehraufwand von dem Gerät, welches am längsten braucht, ab und nicht von der Anzahl der Geräte. In der Monitoringumgebung werden dem Benutzer nur die für ihn wichtigen Knoten angezeigt. Falls das Tool ohne die neue Monitoringoption verwendet wird, ist es nicht notwendig, für Geräte wieder eigene Knoten hinzuzufügen, da auf diesen keine weitere Monitoringsoftware installiert wurde. Daher werden auch in der Monitoringübersicht diese Knoten nicht angezeigt, da bereits alle Informationen in den Operatorknoten enthalten sind. Sobald die neue Monitoringoption des Modellierungstools verwendet wird, wird jedes Gerät als eigenständiger Knoten hinzugefügt. Dabei wird nicht unterschieden, ob auf jedem Gerät Monitoring verwendet wird, damit das Modell einheitlich ist und Geräte in einer eindeutigen Form dargestellt werden.

5.2. IoT-Runtime-System - Multi-purpose Binding and Provisioning Platform

Die Multi-purpose Binding and Provisioning Platform (MBP) ist ein IoT-Runtime-System, das vom IPVS-AS der Universität Stuttgart entwickelt wird [FHS+20]. Die Plattform erleichtert die Entwicklung von IoT-Umgebungen durch Funktionen zum Installieren und Ausführen von Softwarekomponenten und dem Managen von IoT-Geräten. Hierfür lassen sich Hardwarekomponenten aus dem IoT registrieren und Software als Operator einfügen, welche dann auf die jeweiligen Geräte zu verteilen ist. Während der Laufzeit können Daten gesammelt und verarbeitet werden, die dann wieder in der MBP abrufbar sind. Außerdem gibt es die Möglichkeit, Daten zu visualisieren.
In dieser Arbeit wird das Anlegen der Komponenten verwendet, um diese dann über die MBP zu installieren und auszuführen. Hierfür werden passende Schnittstellen bereitgestellt, um gezielt jede Komponente auszulesen und zu kontrollieren. Das Projekt ist Open Source zur Verfügung gestellt.

5.3. Monitoring - InfluxDB2.x

Die weit verbreitete InfluxDB wird als Datenbank verwendet, da die InfluxDB2.x eine vielseitige, leistungsstarke Monitoringlösung bietet. Als Nachfolger des TICK Stacks werden die bisherigen Funktionalitäten des Chronografs und Kapacitors integriert und somit ist nur noch der Telegraf als Monitoringagent separat auf jeder Komponente, von der Daten gesammelt werden sollen, zu verteilen. Die InfluxDB2.x ermöglicht es, Daten von Geräten zu überwachen, indem diese vom Telegraf ausgelesen werden und direkt in der InfluxDB gespeichert werden. Hier kann mit Flux, einer funktionalen Programmiersprache, auf die Daten zugegriffen werden. Flux bietet Optionen für komplexe Filterungsregeln. Für einfache Fälle lässt sich eine Query sehr leicht über eine grafische Benutzeroberfläche erstellen, sodass in den meisten Fällen kein konkretes Wissen über Details der Sprache notwendig ist. Dies ist für diese Arbeit zwar ein weiterer Zwischenschritt, da die Daten auch weiter in der Kontextdatenbank gespeichert werden, jedoch wird hier der Support für eigenständige Monitoringzeitreihendatenbanken dargestellt. Damit wird gezeigt, dass es mit der InfluxDB2.x als Monitoringlösung umsetzbar ist, schränkt jedoch andere Monitoringlösungen, die beispielsweise mithilfe direkter Kommunikation zur Kontextdatenbank noch leichter umsetzbar sind, nicht ein. Andere Monitoringlösungen können ebenfalls mit nur geringem Aufwand in die IoT-Anwendung integriert werden. Jedoch wurde für diese Implementierung aufgrund der Popularität, des laufenden Support und der Vielseitigkeit an messbaren Monitoringeigenschaften die InfluxDB2.x als Beispiel ausgewählt.

5.4. Kontextdatenbank - Apache Jena Fuseki

Apache Jena[2] Fuseki ist ein SPARQL Server für den Zugriff auf die TDB, eine Tripplestore-Datenbank [IG18] ebenfalls von Apache Jena. Diese Datenbank wird verwendet, um Ontologien zu speichern, die durch RDFS (Resource Description Framework Schema) repräsentiert werden. Dies ermöglicht in allgemeiner erweiterbarer Struktur große Mengen an Daten sortiert zu speichern [GABS22]. Eine Kontextdatenbank erlaubt dem Benutzer zu jedem Zeitpunkt den aktuellen Zustand und die Struktur des Systems auszulesen. Genau diese Informationen werden im Modellierungstool für das Monitoring benötigt. Es wäre auch möglich, alle Daten direkt an die Schnittstelle des Modellierungstools zu schicken, jedoch wäre damit keine Flexibilität zwischen unterschiedlichen Formen der Daten verschiedener Quellen gewährleistet. In der Kontextdatenbank liegen alle Daten strukturiert vor und Apache Jena Fuseki bietet die Möglichkeit, mit SPARQL, einer SQL-ähnlichen Sprache, Daten auszulesen und zudem zu filtern. Dies wird für die Regelerstellung zur Konfiguration des Monitorings im Modellierungstool verwendet. Dabei ist die Struktur der Daten in der Kontextdatenbank mithilfe einer Ontologie vordefiniert, was sowohl zum Auslesen vom Modellierungstool, als auch zum Hinzufügen von Daten verwendet wird [GABS22]. Das Verwenden einer Datenbank, ermöglicht den Verlauf der Daten zu analysieren, da automatisch jede Information gespeichert werden kann. Die Flexibilität, welche die Kontextdatenbank dem Modellierungstool bietet, wird beim Eingeben der Daten in die Datenbank hergestellt, da an dieser Stelle Adapter verwendet werden. Diese Adapter müssen für unterschiedlichen Quellen angepasst werden. Die verwendeten Adapter werden im folgenden Abschnitt beschrieben.

In jedem der Adapter werden lediglich die Messdaten oder andere Daten für die Datenbank in die Form der SPARQL-Query gebracht, weshalb das Anpassen für neue Schnittstellen keinen großen Aufwand darstellt.

5.4.1. Schnittstellen Adapter

Die Adapter sind dafür da, die Daten in eine einheitliche Form zu transformieren. Dementsprechend liegen die Daten nach dem Anwenden der Adapter einheitlich vor und können zum Einspeichern in die Datenbank verwendet werden. Drei verschiedene Optionen zum Einlesen der Daten sind hier als Beispiel für die Implementierung aus einer vorherigen Arbeit bereitgestellt, welche in den folgenden drei Abschnitten beschrieben werden. Diese Adapter sind dabei wichtig für die Monitoringfunktion, da damit alle Echtzeitdaten passend bereitgestellt werden. Es handelt sich lediglich um Programme, die aus unterschiedlichen Datenquellen die Informationen auslesen und in eine Kontextdatenbank einlesen. In diesem Fall wird dies für Apache Jena Fuseki mittels SPARQL umgesetzt [GABS22].

[2] https://jena.apache.org/

MBP-Adapter

Der MBP-Adapter verarbeitet alle statisch verfügbaren Daten. Zu den statischen Daten der IoT-Umgebung gehören alle Hardware- und Softwarekomponenten, da diese in der Regel sich zur Laufzeit nicht ändern. Änderungen müssen durch Benutzer vorgenommen werden, wenn beispielsweise ein neues Gerät integriert werden soll. Diese ändern sich nur selten, da sich die Daten nur ändern, wenn das Modell verändert wird. In der Regel sollte dies lediglich in der Entwicklungszeit der IoT-Umgebung geschehen. Trotzdem sind Änderungen nicht ausgeschlossen, um die verwendete Konfiguration der IoT-Geräte regelmäßig zu korrigieren und anzupassen zu können, da Veränderungen erkannt werden und der Systemzustand in der Datenbank stets aktuell gehalten wird.

```
1  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  PREFIX iot-lite:  <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#>
3  PREFIX ssn:    <http://purl.oclc.org/NET/ssnx/ssn#>
4
5  SELECT *
6  WHERE{
7        ?device rdf:type ssn:Device.
8        ?device iot-lite:id "deviceID".
9        ?sensingOperator rdf:type ssn:Service.
10       ?sensingOperator iot-lite:id "sensingOperatorID".
11       ?monitoringOperator rdf:type ssn:Service.
12       ?monitoringOperator iot-lite:id "monitoringOperatorID".
13 }
```

Listing 5.2: SPARQL Query - MBP-Daten auslesen

Monitoringadapter

Der Monitoringadapter ist in dieser Arbeit ein InfluxDB Adapter, da die Monitoringinformationen durch den Telegraf gesammelt werden und in der InfluxDB gespeichert sind. Diese Daten werden mittels Flux, der Programmiersprache für die Datenbank, ausgelesen und dann in die Form der SPARQL-Syntax gebracht. Hierbei werden die Monitoringechtzeitdaten zu ihren Komponenten zugeordnet, welche von dem MBP-Adapter bereits eingespeichert wurden. Daher können bei der Abfrage der Daten aus der InfluxDB die registrierten statischen Daten aus der Kontextdatenbank verwendet werden, um nur nach den Informationen zu fragen, die zu der Umgebung gehören.

```
PREFIX iot-context: <http://www.uni-stuttgart.de/2021/iot-context#>
PREFIX iot-lite: <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#>
PREFIX ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT *
WHERE{
   ?device rdf:type ssn:Device.
   ?device iot-lite:id "deviceID".
   ?device iot-context:hasMonitoringComponent ?Monitor.
   ?Monitor iot-context:hasMeasurement ?measurement.
   ?Monitor rdfs:label "monitoringType".
   ?measurement rdf:type iot-context:Measurement.
   ?measurement iot-context:hasValue ?value.
   ?measurement iot-context:hasTimeStamp ?timeStamp.
}
```

Listing 5.3: SPARQL Query - Monitoringdaten auslesen

MQTT-Adapter

Der MQTT-Adapter ist die allgemeine Schnittstelle, die universell für Echtzeitdaten verwendet werden kann. In dieser Implementierung werden über den MQTT-Broker die Sensordaten empfangen und mithilfe des MQTT-Adapters in die Kontextdatenbank geladen. Hierbei sollte eine weitere Schnittstelle verwendet werden, da dies die Erweiterbarkeit für weitere Datenquellen zeigen soll. In diesem Fall wird die Information wieder aus den empfangenen Nachrichten gelesen und wie in jedem Adapter für die Kontextdatenbank in die SPARQL-Syntax für das Kontextmodell konvertiert. Da es sich bei den Monitoringinformationen erneut um Echtzeitdaten handelt, sollen auch diese ihren zugehörigen statischen Hardwarekomponenten zugeordnet werden.

```
PREFIX iot-context: <http://www.uni-stuttgart.de/2021/iot-context#>
PREFIX iot-lite: <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#>

SELECT *
WHERE{
   ?Sensor iot-lite:id "sensorID".
   ?Sensor iot-lite:hasSensingDevice ?SensingDevice.
   ?SensingDevice iot-context:hasMeasurement ?measurement.
   ?measurement rdf:type iot-context:Measurement.
   ?measurement iot-context:hasValue ?value.
   ?measurement iot-context:hasTimeStamp ?timeStamp.
}
```

Listing 5.4: SPARQL Query - Sensordaten auslesen

5.5. Evaluierung der Anforderungen an das Monitoringsystem

In diesem Abschnitt werden die Anforderungen, die in Abschnitt 4.1 gestellt werden, evaluiert. Dafür wird das entworfene Konzept und die Umsetzung in der Implementierung überprüft, durch welche Lösungen die Anforderungen adressiert werden.

Anforderung 1: Leichte Bedienbarkeit

Leichte Bedienbarkeit wird in dem Konzept adressiert, indem der Benutzer schrittweise durch das Modellierungstool bis zum Monitoring geführt wird. Das Modell wird zuerst durch die Funktionalität beschrieben, dann wird das Modell mit der Initialisierung für die Realisierung der Anwendung in der Umgebung vorbereitet. Schließlich kann zu jeder Komponente Monitoring hinzugefügt werden. Die Schritte sind voneinander getrennt, so dass es leicht ist, das Gesamtmodell schrittweise zu erstellen. Damit wird diese Anforderung adressiert, aber die Qualität der Bedienbarkeit kann stark von der spezifischen Implementierung abhängen. Das Konzept gibt nur die strukturelle Grundlage für eine leichte Bedienbarkeit vor. In der Regelerstellung für die Monitoringkomponenten werden dem Benutzer für eine leichte Bedienbarkeit einfache Filteroptionen gegeben, die ohne spezifisches SPARQL Wissen anwendbar sind. Änderungen im Monitoringmodell sind für den Benutzer leicht umsetzbar, da alte Konfigurationen automatisch verworfen werden und das Modellierungstool die Änderungen für den Benutzer realisiert. Daher wird die Leichte Bedienbarkeit in der Implementierung ebenfalls adressiert, aber kann noch weiter verbessert werden, wenn zum Beispiel die Regelerstellung mehr Auswahl für Regeln gibt, die ohne Kenntnisse über SPARQL verwendet werden können.

Anforderung 2: Geräte und Software individuell konfigurieren

In dem Konzept wird in der Modellerstellung für die Konfiguration der Monitoringfunktionen darauf geachtet, dass jedes Gerät individuell spezifische Monitoringsoftware verwenden kann. Außerdem können Toleranzen für die Warnung vor Fehlern auf das Gerät angepasst definiert werden. Diese Anforderung wird in der implementierung umgesetzt, indem der Benutzer alle verwendeten Geräte als Knoten in dem Monitoringmodell angezeigt bekommt und jeweils eigene Monitoringsoftware zuweisen kann. Ebenfalls werden Regeln einzeln für jede Komponenten spezifiziert, so dass die Flexibilität nicht eingeschränkt wird und nicht für eine Monitoringsoftware auf mehreren Geräten die gleichen Filter angewendet werden.

Anforderung 3: Automatische Installation und Ausführung der Monitoringsoftware

Um diese Anforderung zu erfüllen, wird in Abschnitt 4.3.2 die automatische Installation und Ausführung durch ein IoT-Runtime-System umgesetzt. In welcher Form die automatische Installation für die spezifische Implementierung gestartet wird, hängt dabei von dem verwendeten IoT-Runtime-System ab. In dieser Implementierung wird dies durch Anfragen an die API der MBP ausgelöst. Dafür werden die Paare aus Software und Gerät an die API geschickt, die automatisch die Umsetzung ausführt. Alle notwendigen Informationen dafür werden für die Umsetzung vom Modellierungstool bereitgestellt. Damit ist diese Anforderung erfüllt, indem das IoT-Runtime-System (MBP) automatisch die Zuordnungen der Software zu den Geräten vom Modellierungstool erhält und diese Anforderung für den Benutzer erfüllen kann.

Anforderung 4: Übersicht aller aktuellen Daten

Um diese Anforderung zu erfüllen, wird dem Benutzer eine Übersicht über alle aktuellen Daten während des Monitorings, das in Abschnitt 4.4.2 beschrieben wird, angezeigt. Dafür werden, wie in Abschnitt 4.3.3, alle aktuellen Daten über die IoT-Umgebung dem Benutzer im zuvor erstellten Modell angezeigt. Alle Sensor- und Monitoringdaten werden dabei direkt den Komponenten zugeordnet, um in komplexen Modellen eine übersichtliche Repräsentation des Zustandes zu gewährleisten. Dies wird zunächst von dem Konzept beschrieben, aber muss in der Implementierung übersichtlich dargestellt werden. Eine Übersicht kann dabei von Benutzern unterschiedlich bewertet werden. Daher kann der Benutzer in dieser Implementierung die Anordnung der Knoten bearbeiten, so dass die Übersicht individuell angepasst werden kann. Alle Daten werden dem Benutzer direkt bei den Knoten, die die Software oder das Gerät repräsentieren, angezeigt. Damit wird die Zuordnung der Daten dem Benutzer erleichtert. Die Übersicht könnte möglicherweise durch mehr Optionen zur individuellen Anpassung verbessert werden, indem beispielsweise Daten einzeln auszublenden sind, falls die Menge der anzeigten Informationen zu groß wird.

Anforderung 5: Verschiedene Fehlertypen erkennen

Die behandelten Fehlertypen werden im Abschnitt 4.2 ausführlich adressiert und werden vom Monitoringsystem erkannt. Dazu werden geeignete Monitoringagenten zum Sammeln der Daten verwendet. Dabei werden technische Fehler der Geräte erkannt, die Erreichbarkeit regelmäßig überprüft und Messwerte analysiert, um die Korrektheit zu gewährleisten. Die Anforderung wird in Kombination mit der zweiten Anforderung, der individuellen Konfiguration der Komponenten, erfüllt. Die verschiedenen Fehlertypen werden durch individuelle Fehlerdefinitionen für jedes Gerät oder deren Software erkannt, was ebenfalls in der Implementierung des IAMT für jeden Knoten möglich ist. Mit diesen Fehlerdefinitionen werden dann regelmäßig die Daten der IoT-Umgebung überprüft.

Anforderung 6: Warnung vor Fehlern

Die Warnung vor Fehlern ist der Teil der Anforderung, der von der Umsetzung der anderen Anforderung am meisten abhängt. Warnungen werden durch leicht bedienbare Konfiguration der Monitoringkriterien ermöglicht, da sonst keine Fehler erkannt werden. Die Fehlererkennung deckt verschieden Fehlertypen ab, die differenziert voneinander erkannt werden können. Wenn Fehler erkannt werden, wird eine Warnung ausgelöst, die dem Benutzer in der Übersicht über den aktuellen Zustand der IoT-Umgebung angezeigt wird. Durch die Konfiguration des Modells wird die Anforderung der Fehlererkennung und der Darstellung des Systemzustands ermöglicht. Die Warnungen werden den Komponenten in der Übersicht zugeordnet, um dem Benutzer die Fehlersuche zu erleichtern. Da diese Implementierung die Kontextdatenbank verwendet, werden die Warnungen langsamer angezeigt, als wenn Messwerte direkt übertragen werden. Da die aktuellen Daten, aus denen die Warnungen entstehen, durch regelmäßige Anfragen an die Datenbank an das Modellierungstool übertragen werden, kann dabei das Intervall erhöht werden, um schnelle Datenübertragung zu gewährleisten. Die erhöht jedoch ebenfalls die Auslastung der Hardware des Modellierungstools und Kontextdatenbank, sowie die Netzwerklast. Konzepte zur dezentralen Datenverarbeitung und damit Erkennung von Fehlern, die Warnungen auslösen, werden in dieser Implementierung nicht verfolgt. Damit könnte in Zukunft möglicherweise die Warnung vor Fehler beschleunigt werden, indem im Fehlerfall eine zusätzliche Warnungsnachricht direkt an das Modellierungstool übertragen wird. Trotzdem ist allgemein die Anforderung erfüllt, jedoch kann in spezifischen Anwendungsfällen eine angepasste Realisierung Vorteile bringen.

6. Zusammenfassung und Ausblick

Im Rahmen dieser Bachelorarbeit wurde ein Konzept für ein Monitoringsystem für modellgetriebene IoT-Anwendungen entwickelt. Dadurch können IoT-Modelle leicht um Monitoringfunktionen erweitert und leicht für verschiedene IoT-Umgebungen konfiguriert und erweitert werden.

Der Bedarf für Monitoringlösungen für IoT-Anwendungen steigt mit wachsender Anzahl an Smart Homes, Smart Cities oder Smart Factories. Die IoT-Umgebungen bestehen aus vielen heterogenen Geräten, die verschiedene Softwarekomponenten, Kommunikationsprotokolle oder Betriebssysteme verwenden, wodurch spezifische Anforderungen an die Monitoringlösung erfüllt werden müssen. Diese Systeme sind aufgrund wenig robuster Hardware und Verteilung der Systeme auf viele kleine Geräte fehleranfällig und benötigen daher Überwachung durch eine passende Monitoringlösung.
Mit dem entworfenen Konzept können operationale und technische Fehler erkannt werden. Dafür werden verschiedene Fehlertypen, wie Nichterreichbarkeit von Geräten, falsche Messwerte oder Überlastung der Geräte, adressiert. Um Fehlverhalten zu behandeln, werden Konzepte zur Erkennung für jeden vorgestellten Fehlerfall präsentiert. Da unterschiedliche Fehler verschiedene Auswirkungen auf das System haben, muss die Fehlerbehebung je nach Fehlertyp passend umgesetzt werden. Durch die Integration einer Monitoringlösung in ein Modellierungstool wird von den Vorteilen modellgetriebener Anwendungsentwicklung profitiert, so dass das Modell erweiterbar ist, und Anpassungen im Modell leicht in eine IoT-Umgebung umgesetzt werden können. Um das Konzept mit einem Modellierungstool anzuwenden, wurde eine Architektur in Abschnitt 4.4 entworfen, die dem Modellierungstool die Informationen über die verfügbaren Komponenten in der IoT-Umgebung über ein IoT-Runtime-System zur Verfügung stellt. Für die Datenübertragung wurden verschiedene Ansätze verglichen und für die Verwendung einer Kontextdatenbank zwischen Geräten und Modellierungstool entschieden. Die Funktionalität des Modells ist dabei von der spezifischen IoT-Umgebung losgelöst und lässt sich individuell konfigurieren. Dabei können verschiedene Monitoringlösungen auf den Geräten verwendet werden. Die Daten liegen dann einheitlich für das Modellierungstool in einer Kontextdatenbank in Form eines Kontextmodells bereit. Auf die Daten der Kontextdatenbank lassen sich Analysen für die Fehlererkennung anwenden. Für die Fehlererkennung werden Regeln vom Benutzer definiert, um die Fehlerfälle zu spezifizieren. So können operationale Fehler erkannt werden, wenn beispielsweise falsche Sensorwerte außerhalb des definierten Wertebereichs liegen. Technische Fehler werden durch Monitoringlösungen erkannt, die verschiedene Eigen-

schaften der Geräteauslastung überprüfen. Dieses Monitoring lässt sich im Modellierungstool konfigurieren, so dass die Konzepte zur Fehlererkennung anwendbar sind.

Das Monitoringsystem wurde in das Modellierungstool IAMT integriert, um das vorgestellte Konzept zu realisieren. Für die Integration in das bestehende Modellierungstool wird der Prozess zur Modellerstellung angepasst und erweitert. Durch die modellorientierte Anwendungsentwicklung ist das Modell leicht anzupassen und ermöglicht es dem Benutzer eine strukturierte Übersicht über das System zu erhalten, in welche die Monitoringinformationen integriert wurden. Dafür erstellt der Benutzer seine individuellen Regeln zur Fehlererkennung, so dass alle Monitoringdaten automatisch analysiert werden. Das erstellte Modell der Anwendung und des Monitorings kann nach der Fertigstellung in verschiedenen IoT-Umgebungen angewendet werden. Dafür werden Installation und Ausführung der Softwarekomponenten sowohl für die Funktionalität als auch für das Monitoring automatisiert umgesetzt, wodurch die Skalierbarkeit der Anwendungen verbessert wird. Der Benutzer bekommt das gesamte Modell mit allen aktuellen Daten der Anwendung und deren Monitoringinformationen grafisch dargestellt. Sobald die von ihm definierten Fehler eintreten, werden außerdem Warnungen ausgegeben, um den Status des Systems anzuzeigen und Fehlerbehandlung zu ermöglichen.

In dieser Arbeit konnte die Monitoringerweiterung für ein Modellierungstool umgesetzt werden. Das Konzept ist flexibel anwendbar, aber kann für spezifische Anwendungsfälle, wie beispielsweise echtzeitkritische Systeme, die besonders kurze Reaktionszeiten erfordern, angepasst werden. Wenn dies gefordert ist, kann beispielsweise wie in Abschnitt 4.3.3 beschrieben, zu direkter Kommunikation von dem Modellierungstool zu den Geräten gewechselt werden.

Ausblick

In dieser Arbeit wurde eine zentrale Kontextdatenbank gewählt, in der alle Kontextinformationen gesammelt vorliegen. Dies erleichtert die Analyse der Daten, jedoch kommen an dieser Stelle auch weitere Konzepte zur dezentralen Datenanalyse in Frage. Dafür lassen sich Ressourcen der IoT-Geräte verwenden. Nötige Informationen über freie Rechenleistung können durch die Ergebnisse dieser Arbeit bereits überprüft werden. Durch Dezentralisierung wird somit weitere Skalierbarkeit gewährleistet.

Durch das Verbinden von Knoten ermöglicht das Modellierungstool IAMT dem Benutzer die grafische Modellierung von Zusammenhängen zwischen Geräten. Allerdings ist in der aktuellen Version des Programms keine Funktionalität an die Verbindungen geknüpft. Das bedeutet, dass die Verbindung in diesem Modell lediglich eine grafische Darstellung für den Benutzer ist. Daher wird die Verbindung bei der automatisieren Umsetzung des Modells nicht betrachtet, sondern beschreibt lediglich einen Zusammenhang, der durch eine andere Komponente oder durch den Benutzer hergestellt wird. Die Verbindung könnte beispielsweise direkte Kommunikation zwischen Geräten automatisch verfügbar machen [DH20a]. Dafür

kann bei der Installation der Anwendungssoftware für alle verbundenen Geräte zusätzlich eine Messaging Engine installiert und konfiguriert werden. Die Modellierung der Kommunikation zwischen den Geräten kann ebenfalls für dezentrale Anwendungsszenarien verwendet werden. Als Beispiel können Anwendungsbereiche, die Complex-Event-Processing (CEP) verwenden, von einer dezentralen Datenverarbeitung profitieren [SHWM16]. Wenn in IoT-Anwendungen eine Situationserkennung benötigt wird, die aus mehreren Teilereignissen besteht, können mit einem dezentralen Ansatz Teilereignisse direkt auf dem Gerät durch Filter erkannt werden. Die Teilereignisse werden an das nächste Gerät direkt übermittelt, welches daraus ein komplexeres Ereignis erkennen kann. Dies lässt sich wiederholen, wodurch in jedem Schritt Rechenleistung der IoT-Geräte ausgenutzt wird und das System nicht von einer zentralen Komponente abhängt, mit der jedes Gerät kommunizieren muss.

Die Regelerstellung bietet die Mächtigkeit von SPARQL zur Filterung. Dafür ist bei komplexeren Anfragen spezifische Fachkenntnis über SPARQL erforderlich. Deswegen kann der Bau der Query durch vereinfachte Bausteine in der grafischen Benutzeroberfläche ein Vorteil sein. Dadurch wird mehr Benutzern die Erstellung komplexerer Filterregeln ermöglicht. Weitere Ansätze zur komplexeren Datenfilterung und Analyse können ebenfalls erforscht und gegebenenfalls in das Monitoring für modellgetriebene Anwendungsentwicklung integriert werden. Dafür können Konzepte mit der Verwendung von künstlicher Intelligenz oder Algorithmen zur Ausreißererkennung getestet werden.

Das Modellierungstool kann um einen Sprachsupport für verschiedene Sprachen erweitert werden. Da die Vision des IoTs, einer vollständig vernetzen Welt, Systeme überall zusammenführen soll, sollte das Modellierungstool keine sprachliche Barriere aufbauen.

A. Anhang

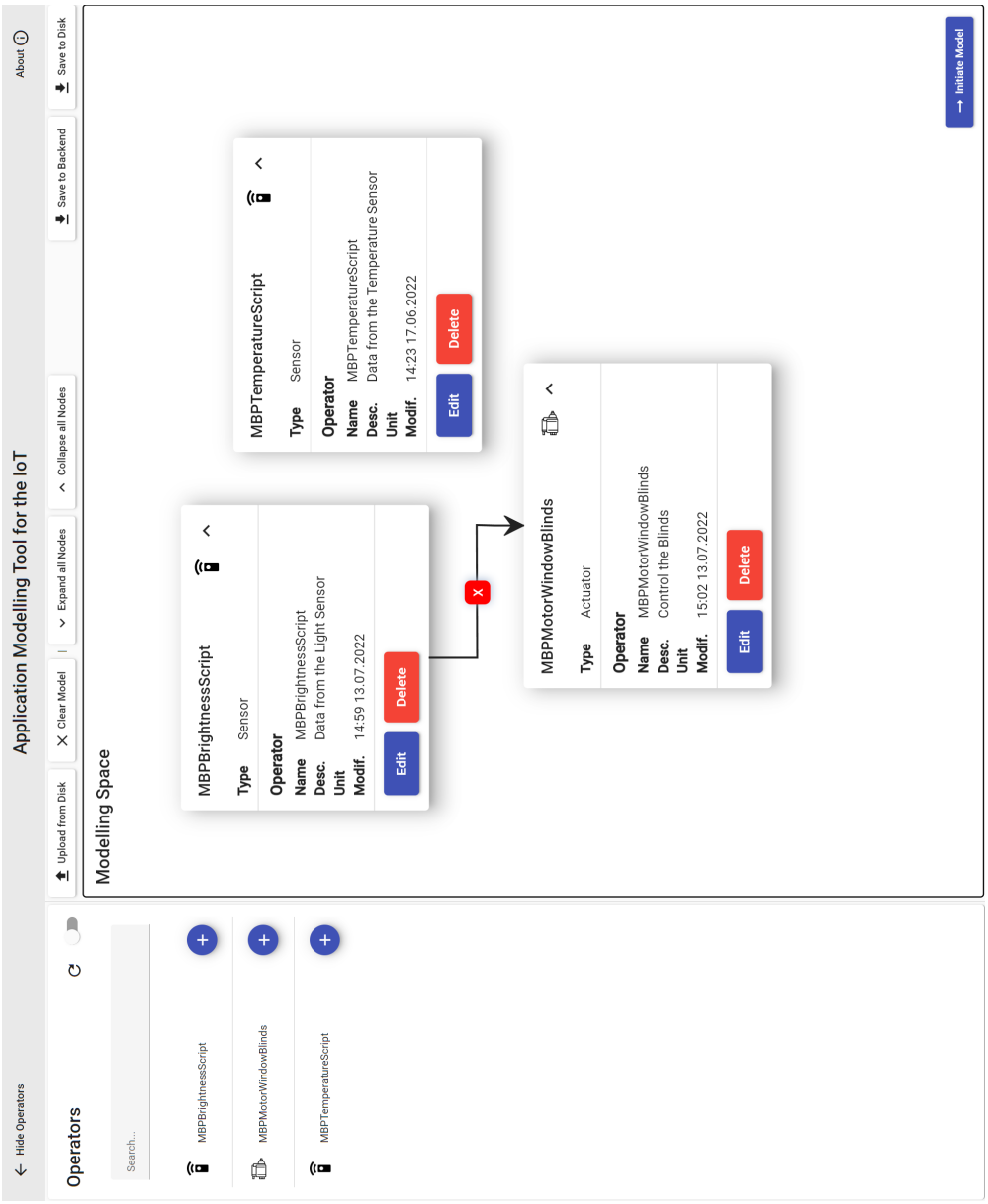

Abbildung A.1.: Modellierungstool gesamtes Fenster Übersicht

Literaturverzeichnis

[ABdP13] G. Aceto, A. Botta, W. de Donato, A. Pescapè. „Cloud monitoring: A survey“. In: *Computer Networks* 57.9 (2013), S. 2093–2115. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2013.04.001. URL: https://www.sciencedirect.com/science/article/pii/S1389128613001084 (zitiert auf S. 22).

[AXM04] I. Akyildiz, J. Xie, S. Mohanty. „A survey of mobility management in next-generation all-IP-based wireless systems“. In: *IEEE Wireless Communications* 11.4 (2004), S. 16–28. DOI: 10.1109/MWC.2004.1325888 (zitiert auf S. 27).

[BUA17] M. Brambilla, E. Umuhoza, R. Acerbis. „Model-driven development of user interfaces for IoT systems via domain-specific components and patterns“. In: *Journal of Internet Services and Applications* 8.1 (Sep. 2017), S. 14. ISSN: 1869-0238. DOI: 10.1186/s13174-017-0064-1. URL: https://doi.org/10.1186/s13174-017-0064-1 (zitiert auf S. 23).

[CCD+17] F. Ciccozzi, I. Crnkovic, D. Di Ruscio, I. Malavolta, P. Pelliccione, R. Spalazzese. „Model-Driven Engineering for Mission-Critical IoT Systems“. In: *IEEE Software* 34.1 (2017), S. 46–53. DOI: 10.1109/MS.2017.1 (zitiert auf S. 21, 27).

[CKLP21] W. Choi, J. Kim, S. Lee, E. Park. „Smart home and internet of things: A bibliometric study“. In: *Journal of Cleaner Production* 301 (2021), S. 126908. ISSN: 0959-6526. DOI: https://doi.org/10.1016/j.jclepro.2021.126908. URL: https://www.sciencedirect.com/science/article/pii/S0959652621011276 (zitiert auf S. 13).

[DH20a] D. Del Gaudio, P. Hirmer. „A lightweight messaging engine for decentralized data processing in the Internet of Things“. In: *SICS Software-Intensive Cyber-Physical Systems* 35.1 (Aug. 2020), S. 39–48. ISSN: 2524-8529. DOI: 10.1007/s00450-019-00410-z. URL: https://doi.org/10.1007/s00450-019-00410-z (zitiert auf S. 13, 27, 66).

[DH20b] D. Del Gaudio, P. Hirmer. „Seamless integration of devices in industry 4.0 environments“. In: *Internet of Things* 12 (2020), S. 100321. ISSN: 2542-6605. DOI: https://doi.org/10.1016/j.iot.2020.100321. URL: https://www.sciencedirect.com/science/article/pii/S2542660520301529 (zitiert auf S. 23, 31, 41).

[DH21] D. Del Gaudio, P. Hirmer. „Towards Feedback Loops in Model-Driven IoT Applications“. In: *Service-Oriented Computing*. Hrsg. von J. Barzen. Cham: Springer International Publishing, 2021, S. 100–108. ISBN: 978-3-030-87568-8 (zitiert auf S. 13, 16, 20, 21, 31, 34, 35).

[DMPP13] S. Dey, A. Mukherjee, H. S. Paul, A. Pal. „Challenges of Using Edge Devices in IoT Computation Grids“. In: *2013 International Conference on Parallel and Distributed Systems*. 2013, S. 564–569. DOI: 10.1109/ICPADS.2013.101 (zitiert auf S. 22).

[FH20] A. C. Franco da Silva, P. Hirmer. „Models for Internet of Things Environments—A Survey“. In: *Information* 11.10 (2020). ISSN: 2078-2489. DOI: 10.3390/info11100487. URL: https://www.mdpi.com/2078-2489/11/10/487 (zitiert auf S. 13).

[FHM19] A. C. Franco da Silva, P. Hirmer, B. Mitschang. „Model-Based Operator Placement for Data Processing in IoT Environments“. In: *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*. 2019, S. 439–443. DOI: 10.1109/SMARTCOMP.2019.00084 (zitiert auf S. 25).

[FHS+20] A. C. Franco da Silva, P. Hirmer, J. Schneider, S. Ulusal, M. T. Frigo. „MBP: Not just an IoT Platform“. In: *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. 2020, S. 1–3. DOI: 10.1109/PerComWorkshops48775.2020.9156156 (zitiert auf S. 28, 41, 50, 57).

[FMKA21] P. Franco, J. M. Martínez, Y.-C. Kim, M. A. Ahmed. „IoT Based Approach for Load Monitoring and Activity Recognition in Smart Homes“. In: *IEEE Access* 9 (2021), S. 45325–45339. DOI: 10.1109/ACCESS.2021.3067029 (zitiert auf S. 25, 26).

[FY20] K. Fang, G. Yan. „IoTReplay: Troubleshooting COTS IoT Devices with Record and Replay“. In: *2020 IEEE/ACM Symposium on Edge Computing (SEC)*. 2020, S. 193–205. DOI: 10.1109/SEC50012.2020.00033 (zitiert auf S. 22).

[GABS22] D. D. Gaudio, B. Ariguib, A. Bartenbach, G. Solakis. „A live context model for semantic reasoning in IoT applications“. In: *2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. 2022, S. 322–327. DOI: 10.1109/PerComWorkshops53856.2022.9767267 (zitiert auf S. 25, 39–43, 58).

[HB17] P. Hirmer, M. Behringer. „FlexMash 2.0 – Flexible Modeling and Execution of Data Mashups“. In: *Rapid Mashup Development Tools*. Hrsg. von F. Daniel, M. Gaedke. Cham: Springer International Publishing, 2017, S. 10–29. ISBN: 978-3-319-53174-8 (zitiert auf S. 13).

[HBS+16] 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.“ In: *Complex Syst. Informatics Model. Q.* 9 (2016), S. 28–43 (zitiert auf S. 21).

[HRM19] V. Hirsch, P. Reimann, B. Mitschang. „Data-Driven Fault Diagnosis in End-of-Line Testing of Complex Products“. In: *2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. 2019, S. 492–503. DOI: 10.1109/DSAA.2019.00064 (zitiert auf S. 39).

[IG18] B. IANCU, T. M. GEORGESCU. „Saving Large Semantic Data in Cloud: A Survey of the Main DBaaS Solutions“. In: *Informatica Economica* 22 (1/2018 2018). ISSN: 14531305. DOI: 10.12948/issn14531305/22.1.2018.01 (zitiert auf S. 58).

[IRMP21] F. Ihirwe, D. D. Ruscio, S. Mazzini, A. Pierantonio. „Towards a modeling and analysis environment for industrial IoT systems“. In: *CoRR* abs/2105.14136 (2021). arXiv: 2105.14136. URL: https://arxiv.org/abs/2105.14136 (zitiert auf S. 29).

[KRA22] R. Kumar, S. Rani, M. A. Awadh. „Exploring the Application Sphere of the Internet of Things in Industry 4.0: A Review, Bibliometric and Content Analysis“. In: *Sensors* 22.11 (2022). ISSN: 1424-8220. DOI: 10.3390/s22114276. URL: https://www.mdpi.com/1424-8220/22/11/4276 (zitiert auf S. 13).

[MHSM20] M. Mormul, P. Hirmer, C. Stach, B. Mitschang. „Avoiding Vendor-Lockin in Cloud Monitoring Using Generic Agent Templates“. In: Juli 2020, S. 367–378. ISBN: 978-3-030-53336-6. DOI: 10.1007/978-3-030-53337-3_27 (zitiert auf S. 26).

[Mol21] K. Moltrecht. *Das intelligente Zuhause: Smart Home 2021*. 2021. URL: https://www.bitkom.org/Bitkom/Publikationen/Das-intelligente-Zuhause-Smart-Home-2021 (zitiert auf S. 13).

[NTBG15] X. T. Nguyen, H. T. Tran, H. Baraki, K. Geihs. „FRASAD: A framework for model-driven IoT Application Development“. In: *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. 2015, S. 387–392. DOI: 10.1109/WF-IoT.2015.7389085 (zitiert auf S. 20).

[NYO+18] Y. Nishiguchi, A. Yano, T. Ohtani, R. Matsukura, J. Kakuta. „IoT fault management platform with device virtualization“. In: *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*. 2018, S. 257–262. DOI: 10.1109/WF-IoT.2018.8355109 (zitiert auf S. 29, 31, 33).

[Rei19] M. Reichel. „Automatisierte Integration von IoT-Geräten in IoT-Umgebungen“. B.S. thesis. 2019. DOI: 10.18419/opus-10691. URL: http://elib.uni-stuttgart.de/handle/11682/10708 (zitiert auf S. 17).

[RLP08] S. Rajasegarar, C. Leckie, M. Palaniswami. „Anomaly detection in wireless sensor networks“. In: *IEEE Wireless Communications* 15.4 (2008), S. 34–40. DOI: 10.1109/MWC.2008.4599219 (zitiert auf S. 32).

[SHWM16] A. C. F. da Silva, P. Hirmer, M. Wieland, B. Mitschang. „SitRS XT-towards near real time situation recognition“. In: *Journal of Information and Data Management* 7.1 (2016), S. 4–4 (zitiert auf S. 67).

[VF13] O. Vermesan, P. Friess. *Internet of things: converging technologies for smart environments and integrated ecosystems*. River Publishers, 2013 (zitiert auf S. 19).

[WSZ+16] U. Wetzker, I. Splitt, M. Zimmerling, C. A. Boano, K. Römer. „Troubleshooting Wireless Coexistence Problems in the Industrial Internet of Things“. In: *2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*. 2016, S. 98–98. DOI: 10.1109/CSE-EUC-DCABES.2016.167 (zitiert auf S. 22).

[Xin20] L. Xing. „Reliability in Internet of Things: Current Status and Future Perspectives“. In: *IEEE Internet of Things Journal* 7.8 (2020), S. 6704–6721. DOI: 10.1109/JIOT.2020.2993216 (zitiert auf S. 13).

[Yıl21] B. Yıldız. *Internet of Things and Smart Cities: A Bibliometric Analysis*. Kastamonu Universitesi İktisadi ve İdari Bilimler Fakültesi Kuzeykent KASTAMONU, 2021 (zitiert auf S. 13).

Alle URLs wurden zuletzt am 14. 07. 2022 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