HAW HAMBURG

Masterarbeit

Alexander M. Sowitzki

Eine Abstraktionsschicht für cyber-physische Systeme am Beispiel einer intelligenten Wohnung

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Alexander M. Sowitzki

Eine Abstraktionsschicht für cyber-physische Systeme am Beispiel einer intelligenten Wohnung

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Kai von Luck
Zweitgutachter: Prof. Dr. Thomas Lehmann

Eingereicht am: 9. April 2019

Alexander M. Sowitzki

Thema der Arbeit

Eine Abstraktionsschicht für cyber-physische Systeme am Beispiel einer intelligenten Wohnung

Stichworte

Cyber Physical Systems, Smart Environment, Smart Home, Abstraktionsschicht, Heimautomatisierung

Kurzzusammenfassung

Im Internet der Dinge (IoT) werden cyber-physische Systeme unterschiedlicher Entwickler zu einem stark heterogenen Zusammenschluss geformt, der die speziellen Aufgaben der Einsatzumgebung hoch konfigurierbar lösen soll. Für diese Problemstellung existieren verschiedene kommerzielle und freie Lösungen, die sich jedoch nur auf Teilaspekte spezialisieren und keine vollwertige Lösung ermöglichen. In dieser Arbeit soll untersucht werden, welche Anforderungen an eine Abstraktionsschicht für die Entwicklung für ein Netzwerk aus cyber-physischen Systemen gegeben sind, welche Funktionen sie bereitstellen sollte und wie sie letztlich umzusetzen ist. Zudem soll eine Architektur für die Elemente eines solchen Systems konzipiert werden.

Alexander M. Sowitzki

Title of Thesis

An Abstraction Layer for Cyber Physical Systems exemplified by an Intelligent Flat

Keywords

Cyber Physical Systems, Smart Environment, Smart Home, Abstraction Layer, Home Automation

Abstract

In the Internet of Things, Cyber Physical Systems of different producers are connected to a strongly heterogeneous system, that has to be highly configurable to fit to its operational environment. Multiple projects exists to conquer this problem but fail to solve every part of it and do not provide a unified solution. This thesis aims to identify, which requirements an abstraction layer for a network of Cyber Phyiscal Systems provides, which functions it should provide and it is to implement. Additionally an architecture for the network of such elements is to be defined.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Tabellenverzeichnis

1 Einleitung

Zunächst wird der Zweck dieser Arbeit motiviert, anschließend folgt eine konkrete Festlegung der Problemstellung. Abschließend wird ein Ausblick auf den folgenden Aufbau der Arbeit gestellt.

1.1 Motivation

Die Studie *Integrierte Forschungsagenda Cyber-Physical Systems* bewertet
Cyber-physische Systeme (CPS) als ein zentrales Feld für die langfristige Entwicklung von Gesellschaft, Wirtschaft und Wissenschaft und sieht das Potenzial, die Lebensqualität von Menschen und die Effizienz von Wirtschaftsprozessen signifikant steigern zu können. So kann über CPS das autonome Fahren umgesetzt werden. Industrie 4.0 kann in der Wirtschaft Einzug halten, wodurch z. B. Fertigungsanlagen optimal ausgelastet werden können. Die Pflege revolutioniert werden, indem z. B. mit Smart Environments effektiver betreute Wohnanlagen pflegebedürftigen Menschen größere Freiheiten und Entfaltungsmöglichen zukommen können. Unabhängig vom Einsatzort zeichnet sich ein CPS dadurch aus, dass es ein System mehrerer Systeme ist, sich also aus einer Vielzahl von Untergruppen zusammen setzt, die gebündelt werden, um gemeinsam ein höheres Ziel zu erreichen. [31]

Über den Ansatz, wie genau CPS umzusetzen ist, wird momentan breit und viel diskutiert, da das Konzept sehr vielseitig einsetzbar ist. So können ganze Städte auf Makroebene optimiert werden, indem bspw. anhand von Wetter- und Verkehrsverhältnissen Ampelschaltungen automatisch angepasst werden, um für einen flüssigeren Verkehr zu sorgen und Umweltverschmutzung zu begrenzen. Auf Kleinstebene können Wohnungen als Smart Home automatisiert werden, um z. B. Energie zu sparen und den Bewohnern zusätzlichen Komfort zu bieten.

Für die Datenverarbeitung und -bündelung in einem solchen Smart Environment stehen zahlreiche existierende Programme (wie z. B. OpenHAB und Node-RED) zur Verfügung, welche über das Netzwerk Informationen von Geräten entgegennehmen, sie nach einer vorgegebenen Logik verarbeiten und wieder über das Netzwerk weitere Geräte steuern. Für die Entwicklung der Software jener Geräte bieten genannte Programme jedoch oft nur geringe Unterstützung.

Viele Sensoren und Aktoren erfordern direkte Interaktion mit Hardwareschnittstellen, also z. B. General-purpose input/output (GPIO), I2C, Serial und Serial Peripheral Interface (SPI). Der Linuxkernel bietet Treiber für jene Schnittstellen, die über sogenannte System Calls in der Programmiersprache C gesteuert werden. Es ist also eine hohe Hardwarenähe erforderlich, die jedoch zwecks Wartbarkeit nicht in einer Erhöhung der Programmkomplexität resultieren darf.

Eine Lösung, welche die Entwicklung eines einheitlichen Ökosystems ermöglicht, das bei der Entwicklung von sowohl hardwarenaher als auch genereller Software unterstützt, die auf unterschiedlichen Ausführungsplattformen betrieben werden kann, fehlt. Um ein solches Ökosystem zu realisieren, ist es also erforderlich, auf mehrere Teillösungen zuzugreifen, was zu einem hohen Wartungsaufwand und zu einer gesteigerten Entwicklungszeit führt. Dies behindert den zügigen Übergang eines CPS vom Konzept zum eigentlichen Einsatz.

1.2 Problemstellung

Aus der o. g. Beschreibung ergibt sich die Frage, wie ein solches System robust, leicht erweiterbar und trotzdem effektiv funktionieren kann. In dieser Arbeit wird das Hauptaugenmerk auf ein einzelnes Smart Home gelegt, welches zu berücksichtigende Besonderheiten aufweist.

Um Anforderungen für die Abstraktionsschicht zu ermitteln, die sich aus einer solchen Konstellation ergeben, ist die Benennung von konkreten Szenarien und deren Simulation hilfreich. Für ein komplexes verteiltes System, wie ein Smart Environment, ist eine Simulation im realen Raum nur mit viel Aufwand möglich. Eine rein virtuelle Simulationslösung vereinfacht zwar den Versuchsaufbau, führt jedoch zusätzliche Komplexität, Fehleranfälligkeit und Implementierungsaufwand ein.

Aus diesem Grund soll ein physikalisches Modell eingesetzt werden, mit welchem ein Smart Home simuliert wird. Anhand dieses Aufbaus sollen Anforderungen für eine Softwareabstraktionsschicht erstellt werden, welches die Steuerung eines verteilten Smart Homes übernimmt und dabei zusätzlich den Schwerpunkt auf Erweiterbarkeit und Ausfallsicherheit legt.

1.3 Aufbau

Diese Arbeit wurde in drei weitere Kapitel aufgeteilt. Zunächst wird mit der Analyse begonnen, in welcher auf die eigentliche Forschungsarbeit eingegangen wird, welche daraus bestand, den aktuellen Stand der Diskussion und der Technik zu untersuchen und daraus Anforderungen an die umzusetzende Implementierung zu stellen. Das Kapitel *Implementierung* befasst sich mit den Entscheidungen, die für das Design der Implementierung getroffen wurden und deren Dokumentation. Im Schlussteil werden ein Fazit und ein Ausblick auf die zukünftige Entwicklung des Themas dargestellt.

2 Analyse

Dieses Kapitel befasst sich mit der Forschungsarbeit, die vor der Konzeption der Architektur der Abstraktionsschicht durchgeführt wird, um eine Übersicht über die Ausgangslage der Problemstellung zu erhalten.

CPSs finden Verwendung in einer Vielzahl von Einsatzgebieten, seien es Maschinen in Fabriken, Systeme innerhalb von Autos oder ein handelsüblicher Drucker. Ein besonders interessantes Forschungsfeld in diesem Kontext ist, wie mehrere CPSs zwecks Kooperation miteinander verbunden werden können. Grundsätzlich kann solch eine Kooperation als ein Multiagentensystem modelliert werden: Einzelne CPSs sind hier Agenten, die eine spezialisierte Aufgabe verfolgen und miteinander arbeiten, um übergeordnete Ziele zu erreichen. Weiterhin umfasst die Problemstellung die Frage, welche Regeln für die Architektur eines solchen Systems aufzustellen sind.

Für diese Arbeit soll das eigentliche Problemfeld auf Smart Homes beschränkt werden, da eine Betrachtung aller möglichen Systeme und deren Anforderungen den Rahmen dieser Arbeit sprengen würde. Trotz dieser Einschränkung können grundlegende Aspekte ermittelt werden, die alle Spezialisierungen von CPSs betreffen. Welche dies sind, wird im Laufe der Analyse erfasst. Smart Homes bieten sich zudem an, da diese ein aufstrebender Bereich der Automatisierung sind und immer mehr Umsetzung im Privatbereich erfahren.

2.1 Begriffsklärung

Als erster Teil der Analyse sollen die verwendeten Begriffe erklärt und eingeordnet werden. Dazu sind vor allem die Begriffe CPS und Multiagentensystem bedeutend. Zusätzlich wird auf IoT eingegangen, da dieser Bereich eine hohe Überlappung mit CPS besitzt.

2.1.1 Internet of Things

Aufgrund seines häufigen Einsatzes als Modewort ist das IoT nur unscharf definiert. Es beschreibt im groben den computergestützten Anschluss von allgemeinen Gegenständen an das Internet, in welchem diese spezifische Dienste bereitstellen. An ihrem Einsatzort sind diese Geräte allgegenwertig, also *ubiquitous*, und oft nicht als Computersystem erkennbar. [19][82]

2.1.2 Cyber-physische Systeme

CPS befasst sich mit der Automatisierung von Systemen im physischen Raum, wobei es häufig für Synonym für IoT verwendet wird, obwohl beide Begriffe unterschiedliche Aspekte derselben Problemlösung beschreiben. [68]

Während beim IoT der Schwerpunkt auf der Interaktion zwischen mehreren Teilnehmern liegt, die die Umgebung beeinflussen oder von ihr beeinflusst werden können, ist bei CPS von größerer Bedeutung, wie diese Beeinflussung umgesetzt wird. Grundsätzlich verfügt jede physikalische Komponente über ein virtuelles Gegenstück, welches dieses imitiert. So ist laut dieser Definition ein netzwerkfähiger Tintenstrahldrucker ein CPS. Kommuniziert ein Desktopcomputer mit diesem Drucker, so präsentiert der Onboardchip diesem nicht die Sammlung von Sensoren und Aktoren, aus denen der Drucker besteht, sondern eine virtuelle, logische Repräsentation des Druckers, der über sich selbst informieren und Befehle entgegen nehmen kann. Kommunizieren mehrere CPS in einem Netzwerk, kann dieses als IoT oder Smart Environment ausgelegt werden. [30, S. 244][68][80, S. 1]

2.1.3 Multiagentensysteme

Ein Multiagentensystem besteht aus einer Sammlung von langlebigen Agenten. Jeder dieser Agenten hat eine spezifische Aufgabe, die er eigenständig erfüllt. Um übergeordnete Ziele zu erfüllen, die von einem einzelnen Agenten nicht umgesetzt werden können, können mehrere von ihnen innerhalb des Multiagentensystems kommunizieren. Um dies zu erreichen, muss das System eine Form von Nachrichtenübermittlung unterstützten, sei es durch den Einsatz einer Middleware oder durch die Bereitstellung eines Discoverydienstes als Hilfsstellung für die direkte Kommunikation zwischen den Agenten. [3][67]

2.1.4 Smart Environments

Ein Smart Environment ist eine Umgebung, welche im physischen Raum angesiedelt und aus mehreren Computersystemen zusammengesetzt ist. Diese können sichtbar und unsichtbar sein und beeinflussen sich wechselseitig mit dem physischen Raum. Dies wird über eine heterogene Menge von Sensoren und Aktoren umgesetzt, welche in einem Netzwerk verbunden sind. Dieses System entspricht Ubiquitous Computing, also dem allgegenwärtigen Einsatz von Computern, die nicht zwangsläufig als solche erkennbar sind, da sie innerhalb von Alltagsgegenständen platziert sind. Für eine solche Umgebung ist auch relevant, wie die Verarbeitung der aus den Sensoren gewonnenen Informationen umgesetzt wird. Dies kann mittels verschiedener Methoden realisiert werden, u. a. Mustererkennung, Fuzzylogik, Complex Event Processing (CEP), neuronale Netzwerke. [21, 58][62, 371]

Smart Homes sind eine Unterform des Smart Environments. Sie finden ihren Einsatz in Wohnbereichen und haben die Aufgabe, u. a. die dortige Wohnqualität zu steigern. Dies kann auf verschiedene Weise erreicht werden, schließt jedoch im Allgemeinen die Beobachtung der Bewohner und die Reaktion auf das beobachtete Verhalten ein. Je nach Spezialisierung der Umgebung kommen konkretere Funktionen hinzu. In Seniorenwohnungen ist bspw. die Überwachung des Gesundheitszustandes und die Erkennung von Notfällen relevant, während in gewöhnlichen Haushalten mehr Wert auf die Automatisierung von Geräten gelegt wird, um z. B. energieintensive Prozesse wie das Betreiben einer Waschmaschine an die Belastung des Stromnetzes anzupassen. [20][49]

Auch können Smart Environments als Multiagentensystem interpretiert werden, indem jedes Computersystem als ein oder mehrere Agenten ausgelegt wird, die als Ziel haben, den vorgegebenen Zweck des Smart Home zu erfüllen. Ein diskussionswürdiger Punkt ist, ob ein Smart Environment wirklich als ein Multiagentensystem betrachtet werden sollte. Zwar sind die Vorteile dieses Ansatzes hinreichend belegbar, eine Anwendung anderer Modelle ist jedoch möglich. So wäre es denkbar, Smart Environments als Aktorsysteme zu interpretieren. [22][38]

Ein Aktor agiert jedoch passiv, bis er von einer anderen Komponente eine Nachricht erhält, auf welche er reagieren kann. Er initiiert also von sich aus keine Aktionen und ist darauf angewiesen, dass ihm externe Systeme Informationen liefern. Dies bedeutet, dass das System nicht eigenständig für die Behandlung von Smart Environments dienen kann, da diese auf Sensorinformationen angewiesen sind, welche dann außerhalb des Aktorsystems gesammelt werden müssten, was kein reines Aktorsystem zulässt. [1][27]

2.2 Einordnung

2.2.1 Vorhergegangene Arbeiten

Für diese Ausarbeitung legen vier Vorarbeiten den Grundstein, auf welche kurz eingegangen wird.

Die Ausarbeitung *Cyber Physical Systems* recherchiert, welche Definition ein CPS aufweist, welche Konzepte es beinhaltet sowie welche soziale Auswirkungen sich aus der Umsetzung dieses Konzepts ergeben. So wird aufgeschlüsselt, welche Rollen CPSs einnehmen können und wie sie in einem Netzwerk angeordnet werden. Diese Erkenntnisse werden für diese Arbeit herangezogen. [68]

In *Eine Infrastruktur für Cyber Physical Systems* wurde analysiert, mit welchen Methoden es möglich ist, hardwarenahe Software in Clusteringsystemen zu betreiben. Dabei wurde überprüft, inwieweit jene Programme in Dockercontainer verpackt und mittels Kubernetes auf mehreren Embeddedsystemen in einer Laborumgebung verwaltet werden können. Der Mehrwert hiervon zeigte sich durch ein zentralisiert gesteuertes Deployment, eine gesteigerte Übersicht sowie verschiedene Hilfsmittel wie die Unterstützung von Rolling Release für den Verwalter. Diese Vorteile erkauft sich die Lösung mit verminderter Leistung. [70]

Den Grundstein für die Aufstellung der Anforderungen einer Abstraktionsschicht legt *Anforderungen an eine Cyber Physical Systems Plattform am Beispiel des MauzFlat Smart Homes*. Hier wird dargelegt, welche Anforderungen an Softwareplattformen zu stellen sind, mit welcher Programme auf CPSs betrieben werden sollen. Diese umfassen u. a. die Unterstützung von Netzwerkfunktionen, die z. B. Kompatibilität erhöhen und Nachrichtenverlust vermindern. Auch ist relevant, dass es Softwareteile durch ihre Entkopplung ermöglichen, das System als Cluster bereitzustellen. [69]

Umsetzung eines smarten Puppenhauses untersucht den Aufbau der physikalischen Simulationsumgebung und die Aufstellung der hier verwendeten Testszenarien. Das Haus wurde von Grund auf als Testumgebung für verteilte Systeme, insbesondere für das Projekt dieser Masterarbeit, entwickelt, die innerhalb einer echten Wohnung agieren sollen und mit verschiedenen Hardwarefunktionen wie z. B. der Erkennung der Präsenz von Bewohnern in einem Raum und der Belüftungsregelung ausgestattet ist. Auf diese Arbeit wird genauer im Analyseteil eingegangen. [71]

2.2.2 Stand der Diskussion

Innerhalb der wissenschaftlichen Gemeinschaft wurden bereits Forschungen durchgeführt, die dieses Thema betreffen. Hier wird nun umrissen, was der aktuelle Stand der Diskussion ist.

Kaibin Bao beschreibt in *A Microservice Architecture for the Intranet of Things and Energy in Smart Buildings* [12] die Möglichkeit, das IoT als Zusammenschlüsse von Microservices zu interpretieren. Neben der Erfassung von benötigten Grundbegriffen beinhaltet dieses Paper eine Untersuchung, welche Methode zur Übertragung von Nachrichten für die Problemstellung am effektivsten ist. Zudem wird überprüft, welche Architektur sich für ein solches verteiltes System anbietet. In *IoT-MAP: IoT Mashup Application Platform for the Flexible IoT Ecosystem* wird analysiert, inwieweit ein IoT System dynamisch um eine Smartphone-Anwendung aufgebaut werden kann. Von besonderem Interesse für dieses Projekt ist hierbei die Analyse, wie Hardwarebausteine von der Systemlogik entkoppelt werden können. [35]

Das Paper *Drivers, Standards and Platforms for the IoT: Towards a Digital VICINITY* [52] befasst sich mit der Entwicklung IoT Komponenten unter Zuhilfenahme von bestehenden Standards. Dazu wird eine Übersicht über aktuelle Softwareplattformen und deren Funktionen aufgestellt und diese gegeneinander aufgewogen. Des Weiteren bietet das Paper zusätzlich eine Übersicht über ausgewählte Embeddedplattformen, die sich generell auch für den Einsatz in diesem Projekt eignen.

The Device Driver Engine - Cloud enabled Ubiquitous Device Integration [39] analysiert die Entwicklung von Treibern innerhalb von IoT Komponenten. Die fokussierte Betrachtung dieses Teilaspektes lässt sich auf eine Architektur übertragen, die sich modular über das lokale Netzwerk erstreckt und wie Treibersubsysteme generalisiert werden können.

Die Betrachtung der Situation von einer anderen Seite bietet *Designing Smart Artifacts for Smart Environments* [72]. Werden Komponenten für Smart Environments entwickelt, so steht die resultierende Funktionalität im Vordergrund. Dies muss bei der Konzeption einer Abstraktionsschicht berücksichtigt werden, um ein Produkt zu erstellen, welches eine größere Zielgruppe bedienen kann.

Die Idee, CPS als Multiagentensysteme anzusehen sind, wurde u. a. schon in *Multi-agent smart environments* [22] behandelt. Grundsätzlich bietet dieser Ansatz den Vorteil, die Architektur anhand von bereits etablierten Konzepten strukturiert und erweiterbar auslegen zu können. Zudem befasst sich die Ausarbeitung mit den Forschungsrichtungen, die zum Zeitpunkt der Veröffentlichung für Smart Environments relevant waren. Dies umfasst u. a. das Problem, dass bei Umgebungen mit mehreren Nutzern ein Konflikt bei der Ermittlung des Sollzustands entstehen kann. Analysiert wird, mit welchen Methoden dieser Konflikt beigelegt werden könnte. Da dies die Architektur dieses Projektes beschreibt, ist das Paper besonders relevant.

The Device Cloud - Applying Cloud Computing Concepts to the Internet of Things [61] untersucht, welche Konzepte sich von Cloudsystemen auf Smart Environments übertragen lassen, was bereits in vorhergegangen Arbeiten angerissen wurde. Hier ist besonders relevant, mit welcher Methodik konkrete Ressourcen in Multiagentensystem abstrahiert werden können, um eine möglichst hohe Kompatibilität mit den Komponenten zu erreichen.

Die Artikel *Secure Control: Towards Survivable Cyber-Physical Systems* [17] zeigt auf, welche Sicherheitsbedenken für CPS einzuräumen sind. CPS können direkten Einfluss auf die physikalische Welt nehmen und u. U. gravierende Schäden verursachen, wenn diese nicht gegen böswillige Manipulationen gesichert sind. Dieser Artikel erfasst mögliche Angriffspunkte und zeigt auf, wie Teilaspekte des Systems fehlertolerant gestaltet werden können.

Um einen Bogen zurück zu Smart Homes zu schlagen, wird hier *Towards the Implementation of IoT for Environmental Condition Monitoring in Homes* [37] analysiert, welches sich mit einem Versuchsaufbau befasst, an welchen die Anwendbarkeit von IoT für Umgebungsüberwachung und -kontrolle überprüft werden soll. Ein Teilaspekt ist die Untersuchung der Verlässlichkeit von drahtlosen Verbindungen, welche im Weiteren beachtet werden soll.

2.2.3 Akteure

An einem Smart Home ist eine Vielzahl von Akteuren vertreten, die über unterschiedliche Interessen und Kenntnisstände verfügen. Die folgend aufgestellten Stakeholder repräsentieren dabei nicht zwangsläufig voneinander isolierte Personengruppen, sondern vielmehr Rollen, von welchen eine Person mehrere einnehmen kann.

Benutzer Die eigentlichen Bewohner der Umgebung agieren als Benutzer des Smart Homes und sind vom Entwicklungsprozess entkoppelt. Ihr primäres Ziel ist die Nutzung der bereitgestellten Funktionen zwecks Komfort und der Steigerung der Wohnqualität. Andere Stakeholder müssen darauf hinarbeiten, dass die Bedürfnisse dieser Zielgruppe erreicht werden.

Administratoren Diese Gruppe kümmert sich um den laufenden Betrieb der Umgebung und stellt sicher, dass dieser reibungslos abläuft. Dies umfasst Wartungsaufgaben wie das Installieren von Updates und den Austausch beschädigter Komponenten. Eine Änderung der Konfiguration findet durch diesen Stakeholder jedoch nicht statt.

Architekten Kümmern sich um die Eingliederung neuer Systeme in die bestehende Umgebung. Dazu werden diese getestet, konfiguriert und an die Benutzer übergeben. Da potenziell während des gesamten Lebens der Umgebung Änderungen im Aufbau des Systems erfolgen, ist dies eine laufende Aufgabe und hilft dabei, das Smart Home auch an neue Gegebenheiten anpassen zu können.

Agentenentwickler Neue Agenten werden durch diese Position entwickelt. Dies umfasst die Konzeption der Hardware sowie der Softwarekomponente und deren Umsetzung. Dadurch nimmt dieser Stakeholder die wichtige Rolle ein, nicht nur für die lokale Umgebung neue Komponenten zu produzieren, sondern auch für andere ein Konzept zur Umsetzung neuer Funktionen bereitzustellen zu können.

2.2.4 Gesellschaftlicher Einfluss

Insgesamt reiht sich die, langsam Fahrt aufnehmende, allgemeine Anwendung von Smart Homes in den Trend der Digitalisierung ein. Da es sich hierbei um einen zentralen und privaten Bereich vieler Menschen handelt, sind bereits jetzt verschiedene Einflüsse auf die Gesellschaft sichtbar.

Wie in den verwandten Arbeiten angesprochen, bieten Smart Homes mit älteren und behinderten Menschen die Möglichkeit, ein selbstbestimmteres und freieres Leben innerhalb ihrer vier Wände zu führen, was mehrere zusätzliche Vorteile mit sich bringt. Ambient Intelligence spielt hierbei eine entscheidende Rolle, da dieses Konzept innerhalb der Umgebung natürliche und einfach verständliche Schnittstellen fördert, welche keine Einweisung benötigen. So kann potenziell Aufwand bei der Pflege eingespart werden, ohne die Qualität dieser zu senken, was dem demografischen Wandel entgegenwirkt.

Viele Wohnungsbesitzer sind nicht in der Lage, komplexe Systeme selber zu entwickeln und zu betreiben, sondern verlassen sich auf kommerzielle Lösung, welche plattformbasiert sind. Dies alleine führt zu dem Vorteil, dass hier ein großes Potenzial für einen wirtschaftlichen Zweig liegt, welcher eine Vielzahl von Arbeitsplätzen bieten kann.

Ein Problem des hier oft verwendeten Angebots einer Plattform, also eines allgemeinen Dienstes wie das HomeKit von Apple, welches relativ einfach einzurichten und leicht zu betreiben ist, in der Erweiterbarkeit und Anpassbarkeit jedoch eingeschränkt ist. Hinzu kommt, dass kommerzielle Unternehmen wirtschaftlich arbeiten müssen, was Regeln für die Lebensdauer verkaufter Geräte erfordert. So sollten Komponenten über ein Mindestmaß an Sicherheit verfügen und über mehrere Jahre mit Aktualisierungen versorgt werden, um dieses Niveau zu halten.

Nachteile dieser Entwicklung sind zudem, dass es sich bei Wohnungen, wie oben beschrieben, um einen sensiblen Privatbereich handelt und Sicherheitslücken ein großes Potenzial für Desaster haben, sollten sie ausgenutzt werden können. Hinzu kommt, dass in vielen Fällen die Umgebung nicht von professionellen Anwendern installiert und gewartet wird, sondern vom Bewohner selbst, der nicht zwangsläufig über Fachkenntnisse verfügt, was das Problem zusätzlich verschlimmern kann, da hier weitere Fehler auftreten können.

2.3 Fallstudie

Um die Behandlung der Problemstellung realitätsnah ausführen zu können, wurde diese Arbeit anhand einer Fallstudie durchgeführt. Dies erforderte die Erstellung einer Testumgebung und das Aufstellen von Szenarien, die mit dieser überprüft werden sollen.

2.3.1 Testumgebung

Als Testumgebung wird ein Smart Home verwendet, in welchem sich CPS Netzwerke wieder finden. Außerdem sollte ein allgemein bekanntes Smart Environment zwecks Anschaulichkeit verwendet werden. Da ein komplettes Smart Home in einer vollwertigen Wohnung zu verortet ist und dies mehrere Probleme einführen würde, die für eine solche Arbeit nur schwer zu rechtfertigen wären, wurde in einer vorhergegangenen Arbeit eine physikalische Simulationsumgebung für Smart Homes erstellt, in welcher getestet werden kann. Ein Schema der Testumgebung ist in 2.1 aufgezeigt. [71]

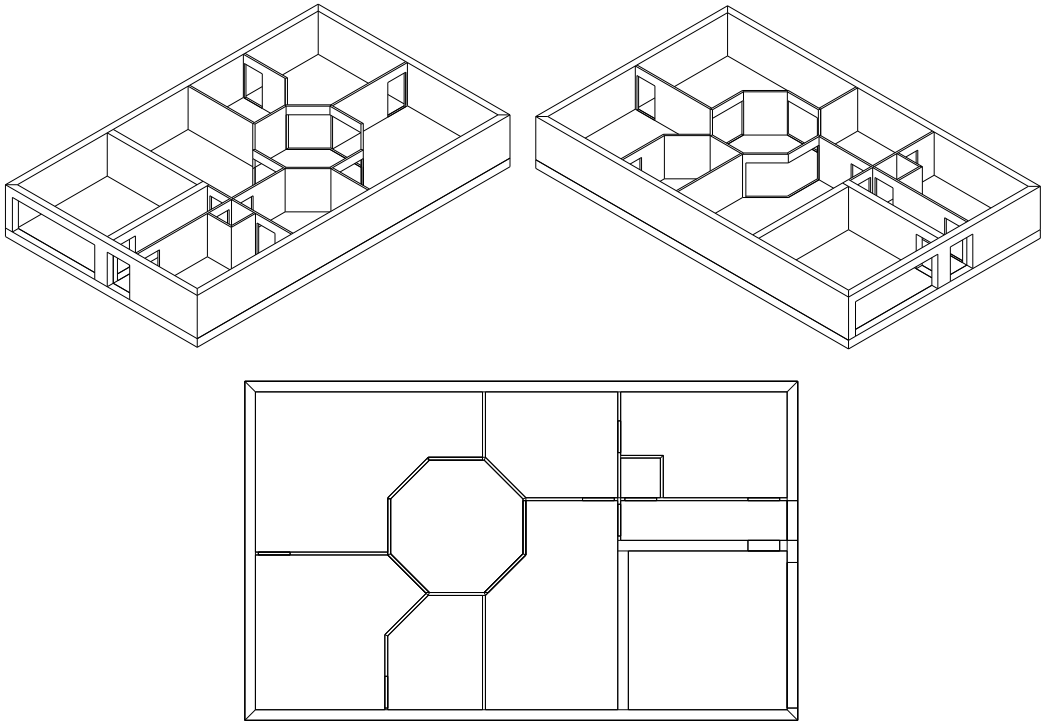


Abbildung 2.1: Schema der Testumgebung

Das Fundament und die Wände des Smart Homes bestehen aus Holz, Glasscheiben wurden mit Polystyrol modelliert, welches ebenfalls verwendet wird, um Aussparungen im Boden abzudecken und eine transparente Decke bereitzustellen, mit welcher das Haus luftdicht gemacht werden kann. Es besteht aus zehn Räumen, die mit verschiedenen Sensoren und Aktoren ausgestattet sind, die wie in einer echten Wohnung Zustände wie die Lufttemperatur messen und Einfluss auf die Umgebung nehmen, bspw. über die Steuerung von Lampen. Angebunden werden diese Komponenten über fünf Raspberry Pis, also voneinander unabhängige eingebettete Systeme, die untereinander über LAN kommunizieren. Grundsätzlich ist diese Installation so eingerichtet, dass die Logik des Smart Homes nicht den Unterschied zu einer echten Wohnung erkennen kann. [71]

2.3.2 Szenarien

Im Artikel zur physikalischen Testumgebung wurden bereits Szenarien aufgestellt, die die Funktionsfähigkeit der Abstraktionsschicht umfänglich beleuchten. Aufgrund ihrer Wichtigkeit für diese Arbeit werden sie im Folgenden erneut aufgelistet. [71]

Aufstehen

Die Bewohnerin des Hauses wacht früh morgens auf und beginnt ihren Arbeitstag. Sobald sie sich im Bett aufsetzt, öffnet sich die Zimmertür und die Beleuchtung des Schlafzimmers und Badezimmers dimmt zur Zielhelligkeit hoch, bis im Raum die Zielhelligkeit erreicht ist und die Tür zum Badezimmer öffnet sich. Die Bewohnerin begibt sich ins Badezimmer und beginnt die Morgenhygiene, während das System die Kaffeemaschine startet.

Sobald die Bewohnerin die Morgenhygiene abgeschlossen hat, begibt sie sich in die Küche, wobei sie andere Räume durchquert. Das Licht dimmt im aktuellen Raum voll und in anliegenden Räumen teilweise hoch. In weiter entfernten Räumen wird die Beleuchtung ausgeschaltet. Die Türen zu angrenzenden Räumen öffnen sich.

Nachdem die Bewohnerin gefrühstückt hat, begibt sie sich in den Eingangsraum. Die Tür schließt sich hinter ihr und das Garagentor öffnet sich, während die Bewohnerin sich für das Verlassen des Hauses vorbereitet. Ist sie fertig, öffnet sich die Tür zur Garage und schließt sich hinter ihr. Sie steigt in das Auto und fährt aus der Garage. Das System schließt daraufhin das Garagentor.

Dies ist in Abbildung 2.2 visualisiert.

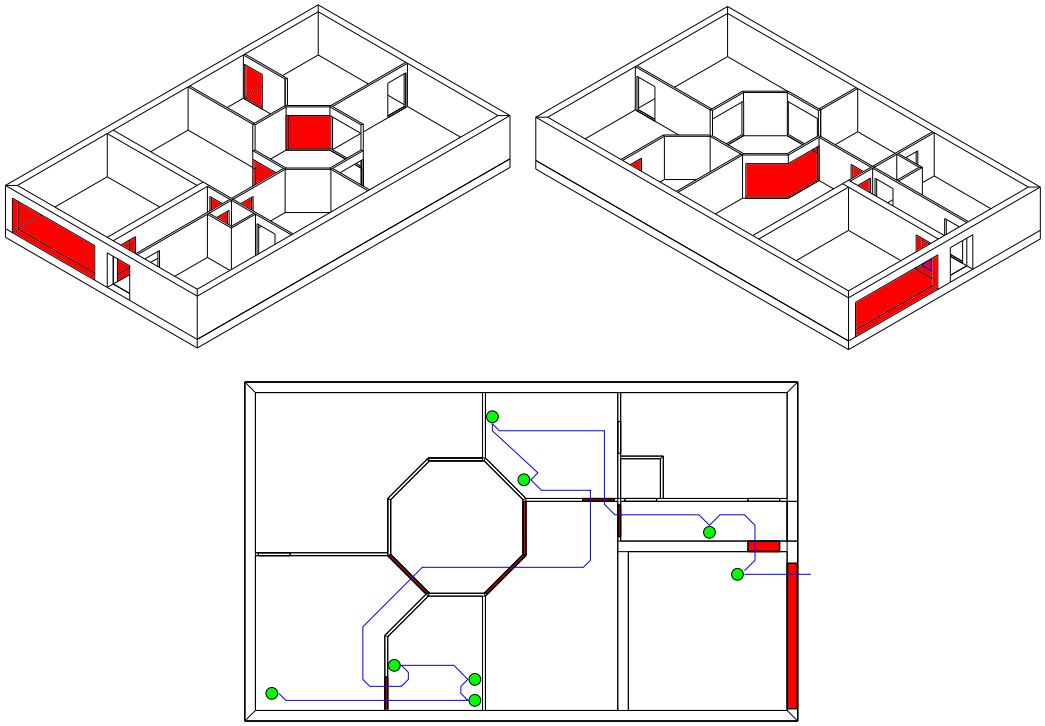


Abbildung 2.2: Weg (blau) der Bewohnerin durchs Haus mit betroffenen Türen (rot) und Interaktionspunkten (grün)

Ventilation

Viele Neubauten sind Niedrigenergiehäuser, welche eine aktive Lüftung erfordern. In diesem Haus soll das System bestimmen können, wie viel Luft in den einzelnen Räumen ausgetauscht wird. Entschieden wird dies anhand von Luftqualitätssensoren, die in den Räumen angebracht sind. Hat ein Raum eine Temperatur, die außerhalb des Sollbereichs liegt oder ist die Luft verbraucht so wird die Belüftung angemessen verstärkt. Sollten die Messwerte stark außerhalb der Toleranzzone liegen, so ist der Benutzer anhand eines Alarms aufzufordern, den Raum zu verlassen.

Da diese Funktion kritisch ist und deren Ausfall zu Schäden an Wohnung und Menschen führen könnte, muss sie gegen Ausfall abgesichert sein. Durch Redundanz soll es möglich sein, einen Teil des Systems außer Betrieb zu nehmen, ohne die Funktionsfähigkeit signifikant zu beeinträchtigen.

Die möglichen Wege der Luftströme sind in Abbildung 2.3 visualisiert.

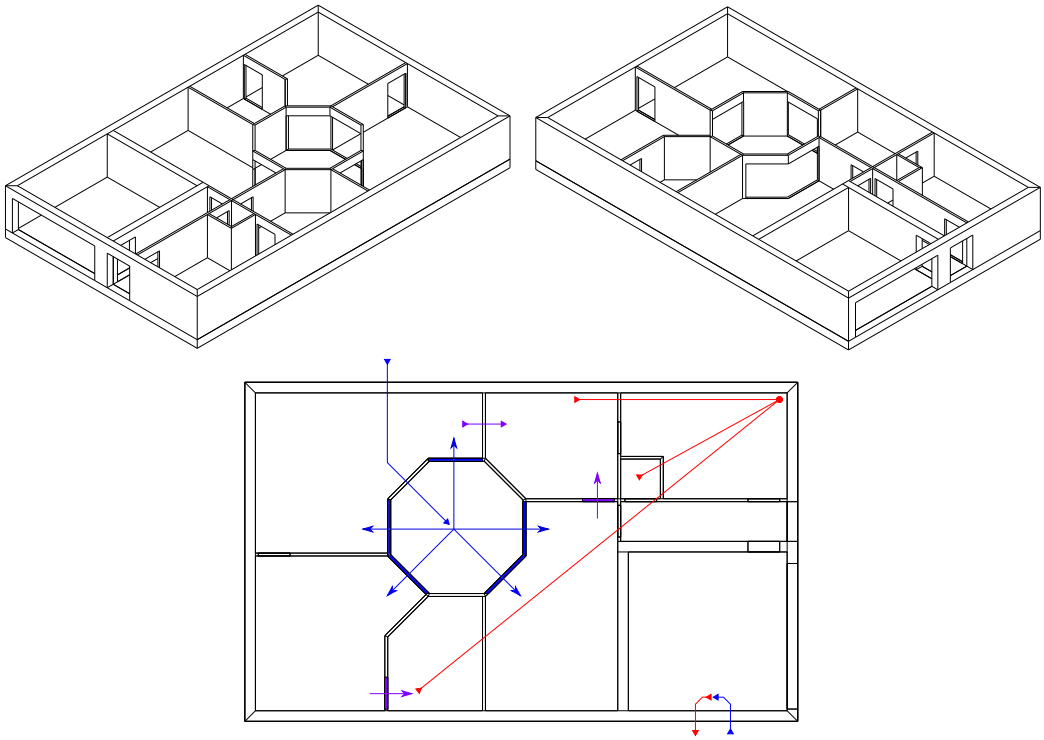


Abbildung 2.3: Ventilationswege der Testumgebung - Zuluft (blau), Umverteilung (violett), Abluft (rot)

Bird Alert

Die Bewohnerin besitzt einen Ziervogel, der sich frei im Haus bewegen kann. Dem Vogel ist der Zugang zum Arbeitszimmer, dem Atrium und dem Wohnzimmer gestattet. Fliegt er in Räume, die anliegend zu verbotenen Räumen liegen, so schließen sich die dahin trennenden Türen. Sollte der Vogel trotzdem in verbotene Räume gelangen, so löst das System einen Alarm aus, den die Bewohnerin einsehen kann. Dabei wird unterschieden, ob der Vogel in einen Raum gelangt ist, dessen Hygiene dadurch beeinträchtigt wird oder einen Raum, durch den der Vogel nach draußen gelangen kann. Erster Fall ist als Warnung zu betrachten, zweiter als kritischer Zustand.

Der Vogel hat hier den Zweck, eine chaotische Komponente in die Fallstudie einzuführen, welche exemplarisch z. B. auf Kleinkinder oder Senioren übertragen werden kann.

Die erlaubten Räume sind Abbildung 2.4 aufgezeigt.

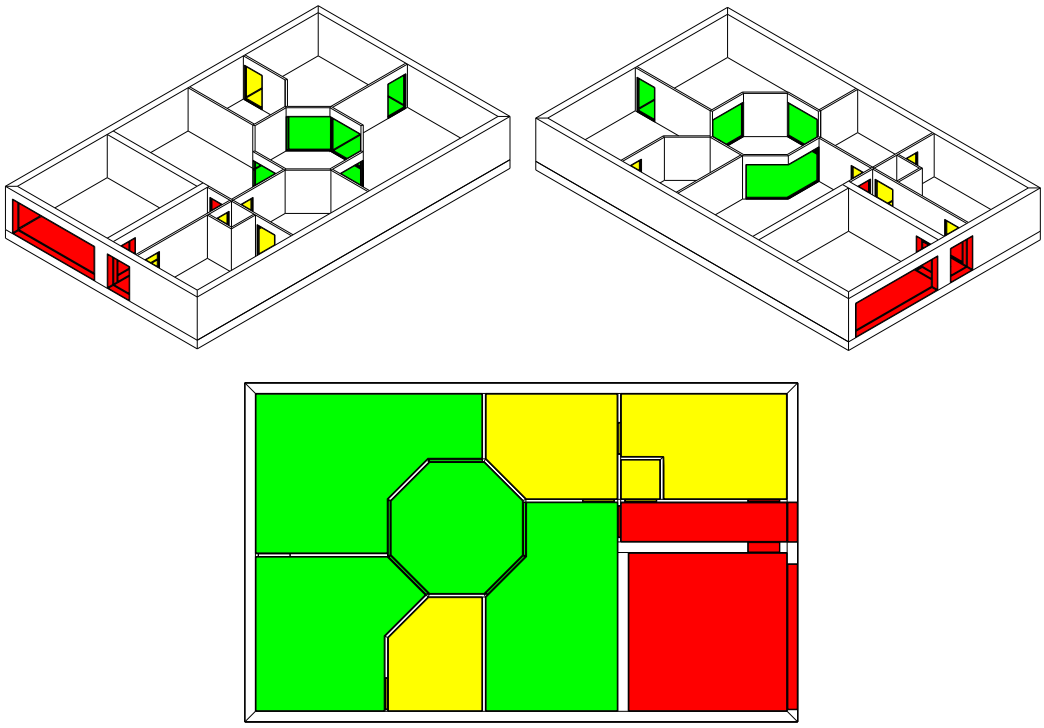


Abbildung 2.4: Für Vögel erlaubte Räume und zugehörige Türen in der Testumgebung - erlaubt (grün), verboten (gelb) aber nicht kritisch, verboten und kritisch (rot)

2.4 Stand der Technik

Zur genaueren Erfassung, was für eine Abstraktionsschicht nötig ist, die praktisch eingesetzt werden soll, wird ein Blick auf den aktuellen Stand der Technik geworfen.

2.4.1 Anwendungsbeispiele

Zur Beschreibung des Stands der Technik sei hier beispielhaft auf drei Anwendungsgebiete für Komponenten eingegangen, und wie diese in einem Netzwerk von CPS zusammenspielen können.

Beleuchtungssteuerung Zur Steuerung der Beleuchtung eines Haushalts sollte auf eine Lösung zugegriffen werden, die Lampen mit einer ausreichenden Leuchtkraft ansteuern kann und dabei nur *CE* geprüfte Produkte einsetzt, da z. B. bei Deckenlampen 220 Volt zum Einsatz kommen. Oft kommunizieren Geräte hier untereinander mit *Zigbee* und lassen sich mittels Gateways an IP-Netzwerke anbinden, über welches sie durch Protokolle wie wodurch sie über Constrained Application Protocol (coap) mit der Abstraktionsschicht kommunizieren können. Hier könnte das *Trådfri* System von *IKEA* eingesetzt werden, welches eine Auswahl an Leuchtmitteln, Schaltern und schaltbaren Steckdosen anbietet. [5]

Erdfeuchtigkeitsmessung Es soll überprüft werden, ob die Erde einer Zimmerpflanze ausreichend bewässert ist. Dazu kann mit Kontakten der elektrische Widerstand der Erde ermittelt werden, um daraus Rückschlüsse auf die Feuchtigkeit zu ziehen. Da viele Controller rein digital arbeiten, ist ein Analog-Digita-Umsetzer nötig, um den analogen Widerstandswert verarbeitbar zu machen. Ein solches Bauteil kann bspw. über I2C angesteuert und durch den Controller mit 3,3 Volt versorgt werden. Ein Konverter, der so eingesetzt werden könnte, ist der *ADS1015*. [6]

Luftmessung Wenn die Luft in einem oder mehreren Räumen überwacht werden soll, können mehrere Messwerte erfasst werden. Die Lufttemperatur kann zum Regel der Heizung dienen, der Luftdruck zur lokalen Wettervorhersage, die Luftfeuchtigkeit zur Ermittlung von Lüftungsbedarf. Auch hier werden Baugruppen von einem Controller mit 3,3 Volt versorgt und über I2C angesteuert. Bspw. der *BME680* ist in der Lage, diese Messwerte zu ermitteln und zusätzlich das Vorkommen von flüchtigen organischen Verbindungen in der Luft zu bestimmen. Dadurch, dass diese Funktionalität in einer Baugruppe vereint ist, minimiert sich der Verdrahtungsaufwand. [4]

2.4.2 Sensorik und Aktorik

Sensorik und Aktorik sind innerhalb von Smart Environments von kritischer Bedeutung. Sensoren ermöglichen dem System, den Zustand von Teilaspekten der verwalteten Umgebung zu erfassen, Aktorik ermöglicht es, den Einfluss auf diese zu nehmen. Somit sind beide zusammen der direkte Weg, um die eigentliche Aufgabe im physischen Raum durchzuführen, weswegen sie ein Kernelement für die Abstraktionsschicht und die benötigte Architektur darstellen. Aufgrund dieser Wichtigkeit soll zuerst beleuchtet werden, wobei es sich bei diesen Gerätschaften handelt und wie mit ihnen interagiert werden kann. Grundsätzlich soll es für die Abstraktionsschicht möglich sein, alle üblichen Sensorik- und Aktorikkomponenten zu unterstützen. Dies erfordert selbstredend die Arbeit eines Entwicklers, welche jedoch durch die Abstraktionsschicht minimiert werden soll. [21, 58][23]

Aktuelle, vollwertige Smart Home Lösungen verfügen über eine Form von Netzwerkanbindung, mit welcher sie an die bestehende Infrastruktur angebunden werden. Dies wird häufig über WLAN realisiert, da diese Netzwerktechnologie in Haushalten weit verbreitet ist, jedoch werden auch *Bluetooth*, *Zigbee* und weitere angewendet, da diese Spezifikationen für die jeweiligen Anwendungsfälle spezialisierter und somit vorteilhafter sein können. Viele dieser Lösungen sind normalerweise darauf ausgelegt, mit verbreiteten Hubs im Smart Home verbunden zu werden, an welche sie Informationen übermitteln oder von dem sie Steuerkommandos empfangen. Je nach verwendetem Hub wird den Besitzern nur die Möglichkeit gegeben, vordefinierte Konfiguration zu aktivieren oder aber mit API-Zugang komplexere Plugins zu implementieren. [16][44]

Neben der Unterstützung für diese Komponenten ist entscheidend, auch jene zu unterstützen, die nicht über eine eigenständige Anbindung an solche Hubs verfügen. Während eben genannte Lösungen über einen festgelegten Bereich von Funktionalität verfügen und aufgrund mehrerer Faktoren, u. a. aufgrund der gesteigerten Komplexität, teurer in der Anschaffung sind, kann mit der Anbindung von elektrischen Baugruppen eine weitere Auswahl von Anwendungsmöglichkeiten zu einem geringeren Preis erreicht werden. Da diese Komponenten von einem Controller gesteuert werden müssen, der in diesem Fall von der Abstraktionsschicht betrieben wird, bieten sie maximale Konfigurierbarkeit und Anpassbarkeit an den Einsatzort. [79][83]

Welche genauen Baugruppen unterstützt werden, ist hinsichtlich des Ziels für Abstraktionsschicht nicht relevant, da für verschiedene Anwendungen oft eine Vielzahl von Herstellern unterschiedliche Modelle anbieten, welche unterschiedliche Herangehensweisen anwenden, jedoch letztlich gleiche Resultate in unterschiedlicher Qualität liefern. Grundsätzlich sei hier angemerkt, dass das Beziehen von Komponenten, seien es vollwertige Komponenten oder Baugruppen, über den Onlinehandel und andere Bezugswege auch für Privatpersonen durchgeführt werden kann, sodass für beide Arten keine Einschränkung beim Einkauf zu erwarten ist.

Die Erfassung eines temporären Zustandes in der technischen Entwicklung kann nicht als Grundlage dafür genommen werden, um für ein langlebiges Programm Anforderungen zu ermitteln. Anforderungen ändern sich rapide und sollten daher so weit wie möglich generalisiert werden, um langfristigen Nutzen zu bringen. So lässt sich hier u. a. extrahieren, dass Entwickler erforderlich sind, die grundlegende Erfahrungen mit dem Umgang von elektrotechnischen Installationen haben und in der Lage sein müssen, Agenten für sich ändernde Hardware anpassen zu können. Zudem sollte ein, auf mehrere Schichten unterteilbares, Abstraktionssystem angewendet werden, um hardwarefernen Programmteilen stabile Schnittstellen zu bieten und langfristig wartbar zu sein.

Entwicklerschnittstellen

Je nach Art des anzubindenden Geräts bestehen unterschiedliche Möglichkeiten, wie als Controller mit diesen interagiert werden kann. Da diese vom Gerätehersteller entsprechend der, sich stets weiterentwickelnden, Anforderungen des Geräts konzipiert werden, kann keine allgemeine Aussage darüber getroffen werden, welche einzelnen Standards generell gegenüber anderen zu präferieren sind. Zwecks Einordnung in das Themenfeld werden hier übliche, nicht industrielle, Anbindungsarten aufgezählt.

IP Netzwerk Wie bereits erwähnt, besteht in vielen Haushalten bereits Zugang zu einem WLAN-Zugangspunkt, welcher es ermöglicht, sowohl mit Teilnehmern des lokalen Netzes als auch mit dem Internet über das *Internet Protocol* und den dazugehörigen Transportprotokollen zu kommunizieren. Moderne Umgebungen verfügen zudem über eine kabelgebundene Anbindung an das lokale Netz über Ethernet. Dies ermöglicht eine strahlungsärmere und ausfallsicherere Verknüpfung von Komponenten. Die Verwendung von IP als allgemeines Kommunikationsmedium führt zu einer hohen Kompatibilität, da Computersysteme üblicherweise von Haus aus über dahin gehende Anbindungsmöglichkeiten verfügen und somit die Applikationsschicht von Programmen für das Szenario angepasst werden muss. Im nicht industriellen Bereich dazu üblich sind Applikationsprotkolle wie Representational state transfer (REST), CoAP, Message Queuing Telemetry Transport (MQTT) und Advanced Message Queuing Protocol (AMQP). Während die Verwendung solcher Protokolle unabhängig von der Programmiersprache gut durch bestehende Softwareprojekte abgedeckt werden kann, ist es die eigentliche Herausforderung, gerätespezifische Anwendungsprotokolle umzusetzen. [8][10][44]

Spezialisierte Funkstandards Während IP-Netzwerke aufgrund ihrer Universalität gut zum Austausch für Daten verwendet werden können, sind die drunter liegenden Transportmedien, Ethernet für Kabelanbindung und WLAN für kabellose Anbindung, nicht sehr energieeffizient und erfordern Hardware, die für kleinere Geräte einen signifikanten Zuwachs an Komplexität darstellt. Hinzukommt, dass WLAN-Netzwerke im üblichen Infrastrukturmodus eine Sterntopologie aufweisen und Geräte mit kleineren Antennen u. U.. im selben Haus keine ausreichende Reichweite aufweisen, um sich mit dem Zugangspunkt zu verbinden. Hier bieten vermaschte Netzwerke wie ZigBee den Vorteil, dass Teilnehmer sich nicht in direkter Reichweite zu einem anderen Endpunkt befinden müssen, um Nachrichten mit diesem auszutauschen. Hierbei handelt es sich in der

Regel um energieoptimierte Kommunikationsmethoden, die ihre Effizienz durch geringere Transferraten erkaufen. Wenn ein Gerät nicht dem kompletten Netzwerk verfügbar gemacht werden, sondern nur mit einzelnen Gegenstellen drahtlos kommunizieren soll, so bietet sich ein Funkstandard wie Bluetooth Low Energy an, welche unter minimaler Energienutzung Entfernungen von bis zu 100 Meter überbrücken können. Dies kann z. B. verwendet werden, um Sensoren außerhalb des Hauses anzubinden, die über eine Batterie versorgt werden und auf Stromsparsamkeit angewiesen sind. [37][50][55]

Bussysteme Die vorhergegangenen Lösungen befassen sich mit der Verbindung von räumlich getrennten Komponenten. Sollen Baugruppen an Controller angebunden werden, sind die Anforderungen anders. Da ein Controller die Verbindung zum Netzwerk aufbaut und darüber die Baugruppen anbindet, stehen für diese Durchsatz und Verlässlichkeit im Vordergrund, da dies potenziell für viele Anwendungsfälle erforderlich ist. Controller verfügen über eine begrenzte Anzahl an Ein- und Ausgängen, weswegen die gemeinsame Nutzung von Leitungen erforderlich ist. Dies wird durch Bussysteme ermöglicht, durch welche mehrere Bauelemente an Controller angebunden werden. Relevante Implementierungen sind hier u. a. I2C und SPI. [33][65]

Verkabelung über GPIO Einfache Bauelemente, wie Schalter und Feinstaubsensoren, verfügen oft über keinen Chip, der die Integration in Bussysteme ermöglicht, weswegen sie exklusiv über I/O des Controllers abgefragt und gesteuert werden müssen. Dies umfasst digitale Signale, die oft direkt oder mit vorgeschalteten Spannungswandlern vom Prozessor des Controllers direkt verarbeitet werden können, und analoge Signale, welche aufgrund der Natur von Prozessoren Digital-Analogwandler und Analog-Digitalwandler benötigen. [46][79]

2.4.3 Smart Environments

Aus dem vorhergegangenen Abschnitt kann geschlossen werden, dass eine heterogene Gruppe von Komponenten angebunden werden soll und dafür Agenten erforderlich sind. Neben diesen ist es nötig, eine Zahl von Hilfsagenten zu betreiben, um eingehende Informationen zu verwerten und zu ausgehenden Informationen umzuschreiben. All diese Komponenten formen ein Smart Environment, welches in diesem Abschnitt genauer beschrieben werden soll. Standardisierte Kommunikation zwischen diesen ist relevant, jedoch führt erst die Kapselung der Schichten dazu, das Einzelne von ihnen ohne größeren Aufwand ersetzt werden können. [57]

Ebenen

Ein Smart Environment ist ein komplexes System, welches sich aus mehreren Ebenen zusammensetzt, die aufeinander aufbauen. Für diese Arbeit sind drei Ebenen relevant, die vom konkretesten zum abstraktesten Fall hin beschrieben werden: elektronische Baugruppen, Agenten und der Kommunikationspool. Diese sind in Abbildung 2.5 dargestellt.

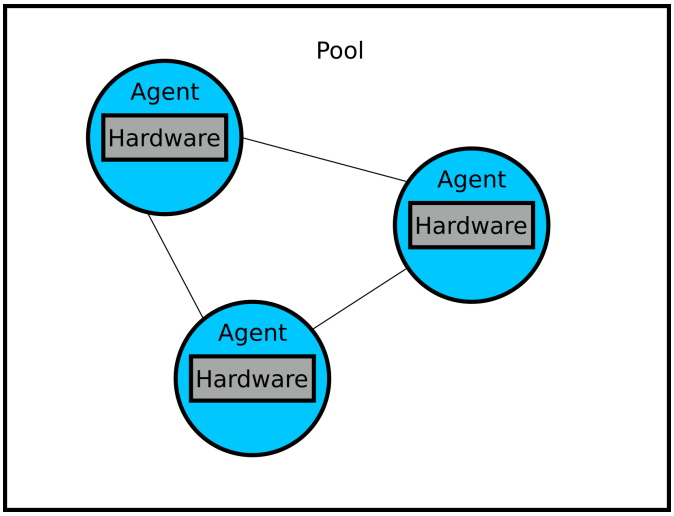


Abbildung 2.5: Ebenen eines Smart Environments

Elektronische Baugruppen Auf unterster Ebene stehen die eigentlichen Baugruppen, welche die Sensorik und Aktorik beinhalten. Diese Komponenten agieren nicht als Teil des Netzwerkes, sondern als Dienstleister für den Controller. Da es sich hierbei um elektronische Bauteile handelt, die für einen konkreten Anwendungszweck optimiert sind, können sie nur schwer abstrahiert werden und müssen auf eine spezifische Weise angesprochen werden.

Agenten Darüber folgen die Agenten, welche wie bereits beschrieben eine eigene Aufgabe erfüllen und innerhalb des Multiagentensystems anderen Zielen zuarbeiten. Ein Agent hat dabei Zugriff auf eine, entsprechend den lokalen Regeln der Agentensynthese, beliebige Anzahl an Baugruppen. Er fungiert dabei als Abstraktionsschicht zwischen den Baugruppen. Während ein Agent konkrete Baugruppen ansteuert und deswegen explizit für diese geschrieben sein muss, macht er die dadurch zugänglichen Ressourcen über eine generelle Schnittstelle verfügbar, die sich beim Wechsel von z. B. dem Modell der Baugruppe nicht ändert. [24][54]

Kommunikationspool In diese Schicht werden die eben genannten Ressourcen eingespeist. Auf dieses Medium haben alle Agenten des Multiagentensystems Zugriff und können Informationen mit einem festgelegten Protokoll teilen. Je nach Anwendungsfall bietet es sich hierbei an, Informationen in diesem Netz persistent bereitzuhalten oder Agenten Informationen voneinander anfordern zu lassen. [54][60]

Dezentralisierung

Es ist üblich, Agenten eines Smart Environments auf nur einer Plattform laufen zu lassen. Dazu werden diese auf einer einzelnen Ausführungsplattform gesammelt, erhalten dort Sensorinformationen und konvertieren diese in Steuerinformationen. Die Plattform muss aufgrund der Bündelung höhere Rechenkapazitäten aufweisen, erspart jedoch den administrativen Aufwand, der durch die Verteilung der Agenten auf mehrere Rechner entstehen würde. Zum Nachteil wird, dass diese Plattform dadurch zum Single Point of Failure des Systems wird. Fällt sie aus, so ist der Großteil der Umgebungsfunktionen nicht mehr verfügbar. [81]

Eine Verteilung der Agenten auf mehrere Plattformen im Netzwerk bringt mehrere Vorteile. Dadurch, dass diese Agenten nach beliebigen Kriterien aufgeteilt werden können, kann z. B. anstelle einer einzelnen Plattform mit hoher Kapazität ein Cluster von kleineren Systemen mit geringer Leistung verwendet werden, was u. U.. zu einem geminderten Stromverbrauch führt. Auch erhöht sich, eine entsprechende Kommunikationsform vorausgesetzt, die Ausfallsicherheit des Gesamtsystems, da der Ausfall einer Plattform nur einen Teil der Agenten betrifft. Mithilfe eines Clustermanagers können dann betroffene Agenten auf andere, noch lauffähige, Plattformen umverteilt werden. [12, S. 4]

Abstrahierbares

Für Agenten eines Smart Environments gibt es viele generische Aufgaben, die Agenten von bestimmten Gruppen gleichermaßen ausführen müssen. Hier soll eine Übersicht darüber gegeben werden, welchen Kategorien diese Aufgaben entsprechen und inwieweit sie sich abstrahieren lassen.

Konfiguration Sämtliche Agenten müssen konfiguriert werden, da zumindest deren Identität festgelegt werden muss und wie dieser mit dem System interagiert. Hinzu kommen Informationen, die zum Betrieb des Agenten notwendig sind, bspw. die Busadresse einer Baugruppe, die der Agent ansteuern soll und die Senke, in welche Sensorinformationen abgelegt werden sollen. Es sollte möglich sein, den Start eines Agenten zu verschieben, wenn kritische Konfigurationsparameter noch nicht gesetzt sind.

Peripheriezugriff Hardwarenahe Agenten müssen zwangsläufig Zugriff auf die Peripherie des Betriebssystems nehmen, um mit den Baugruppen zu kommunizieren. Dies kann wie erwähnt über eine Vielzahl von Bussystemen oder anderen Methoden ablaufen. Der Zugriff auf diese sollte zumindest im Groben vereinfacht werden. Häufig benutzte Bussysteme könnten vollständig abstrahiert werden, sodass Agenten keinen Bezug auf das unterliegende System nehmen müssen. Für seltenere Peripherie, deren vollständige Unterstützung sich nicht lohnt, könnte der Zugriff auf Betriebssystemschnittstellen vereinfacht werden, damit Agenten zumindest teilweise keinen konkreten Bezug auf das System nehmen müssen. Nichtsdestotrotz wird es nicht möglich sein, eine komplette Abstrahierung vom Betriebssystem und der Zielhardware vorzunehmen, da es sich hierbei um hardwarenahe Agenten handelt, die entweder exotische Zugriffsmethoden oder Direktzugriff zwecks Performance benötigten könnten. [18]

Zustandshaltung Ein Agent durchschreitet über die Dauer seiner Ausführungszeit mehrere Zustände. Es sollte dem Entwickler abgenommen werden, für jeden Agenten programmatisch sicherzustellen zu müssen, ob ein Agent vom System noch erfordert wird und ob sich Konfigurationsparameter geändert haben. Außerdem soll es möglich sein, Input / Output (I/O)-Fehler, wie sie bei Peripheriezugriff öfter auftreten können, automatisch in einem definierten Ablauf behandeln zu können. [32]

Monitoring Für die Entwicklung ist es relevant, den Entwicklern Möglichkeiten zu Debuggen und Verfolgen von Fehlern zu bieten. Dafür können verschiedene Strategien angewandt werden. So ist das Bündeln von Logausgaben aller Agenten und das Erfassen des Gesamtzustands des Systems an einer Station eine Möglichkeit, unnötige Zerlegung relevanter Informationen auf mehrere Ausführungsplattformen zu vermeiden. [9]

Tasking Agenten müssen Aufgaben meistern, die sich u. U.. an zeitliche Dimensionen anpassen müssen. So muss ein Sensor in regelmäßigen Intervallen abgefragt werden oder ein neues Ereignis generiert werden, wenn eine Gegenstelle innerhalb einer vorgegebenen Zeit nicht geantwortet hat. Wie dies für Agenten umzusetzen ist, hängt stark von der verwendeten Programmiersprache und ihrem Threadingmodell ab. Eine Abstraktion ist hier also angemessen, wenn die Sprache benötigte Funktionen nicht bieten kann.

2.4.4 Complex Event Processing

Eine wichtige Aufgabe in Smart Environments ist die Verarbeitung von Daten. Dies umfasst die Serialisierung von Daten, also das Verpacken von Objekten in einen Bytestrom und die Rückwandlung in ein Objekt. Die Kernaufgabe hierbei ist jedoch die Anreicherung von Daten mit zusätzlichem Kontext, das Aggregieren von verschiedenen Daten und das Erkennen von Mustern in einem Datenstrom. [67]

Serialisierung

Die Abstraktionsschicht muss für die korrekte Verarbeitung von Daten diese auch Serialisieren können. Da es eine begrenzte Anzahl von Formaten und Serialisierungsformen gibt, sollte es für Entwickler möglich sein, eigene Serialisierer zu erstellen und sie zu der Sammlung von Serialisierern hinzufügen, die die Abstraktionsschicht von sich aus unterstützt.

Für eine Einführung in diese Datenarten siehe Abschnitt A.1. Um als Entwickler Schnittstellen definieren zu können und Agenten die Möglichkeit zu geben, empfange Daten auf ihre Kompatibilität zu überprüfen, sollte neben der eigentlichen Nutzlast auch bekannt gemacht werden, welcher Datentyp in einem bestimmten Nachrichtenkanal übertragen wird und was dieser repräsentiert.

Verarbeitungsarten

Die serialisierten Daten müssen verarbeitet werden, um aus ihnen Schlüsse ziehen zu können. Für diese Verarbeitung können verschiedene Methoden angewendet werden, von denen diejenigen folgend genauer beschrieben werden, die für dieses Projekt relevant sind. [67]

Kontextanreicherung In der Datenverarbeitung innerhalb von CPSs ist Kontext ein entscheidender Faktor. Die Informationen, dass der digitale Eingang auf einem bestimmten Rechner einen niedrigen Pegel ausweist, ist an sich nicht aussagekräftig. Erst durch die Verbindung dieser Informationen mit Kontext kann diese Angabe interpretiert werden. So kann diesem Eingang zugeordnet werden, dass es sich dabei z. B. um den Zustand eines bestimmten Fensterkontakts handelt. Hinzukommen kann die Information, für wie lange das sich Fenster bereits in diesem Zustand, also offen oder geschlossen, befindet. Diese Anreicherung stellt für Rohdaten den ersten Schritt zur Verarbeitung dar.

Konvertierung Unter Zuhilfenahme von Regeln können Informationen umgewandelt werden. So kann die Temperatur eines Raumes mit entsprechender Regel in den Wahrheitswert umgewandelt werden, ob die Temperatur geringer als 20 Grad ist. Neben Logikoperationen können Werte zwischen verschiedenen Formaten konvertiert werden.

Aggregierung Die oben genannten, einzelnen Informationen, können verwendet werden, um primitive Logik innerhalb des CPSs zu realisieren, was jedoch nicht die nötige Tiefe erreicht, um systemweit Entscheidungen zu treffen. Durch die Kombination verschiedener Messwerte können komplexere Annahmen überprüft werden und dementsprechend gehandelt werden. So tragen bspw. die Luftfeuchtigkeit und -qualität zur Entscheidung bei, wie stark ein Raum belüftet werden muss.

Umsetzungsweise

Die Implementierung der Systemlogik lässt sich auf mehrere Weisen umsetzen. Folgend soll auf eine Auswahl dieser Möglichkeiten eingegangen und ihre Vor- und Nachteile aufgeschlüsselt werden.

Spezialisierte Oberflächen Ein häufig angewandter Ansatz ist der Einsatz von Weboberflächen, in welcher der Benutzer die gewünschte Logik eintragen kann. Dies kann grafisch in Form von Blockschaltbildern oder textuell mit der Eingabe von z. B. Quellcode erfolgen. Vorteil hiervon sind eine gesteigerte Übersicht, höherer Komfort durch bessere Zugänglichkeit und die Möglichkeit, Rechte für individuelle Nutzer festzulegen. Nachteil ist, dass komplexere Regeln wie Mustererkennung in Datenströmen in der Regel von solchen Oberflächen nicht angeboten werden, wodurch weniger Szenarien umgesetzt werden können. Populäre Beispiele hierfür sind NodeRED und OpenHAB. [25]

Complex Event Processing Engine Die Datenverarbeitung in Smart Environments lässt sich gut mit CEP abbilden, also durch die Interpretation, Kontextanreicherung und Aggregierung von Ereignisströmen mithilfe eines Regelwerks und der Generierung von neuen Ereignissen. Im professionellen Umfeld werden dafür Complex Event Processing Engines (CEP Engines) eingesetzt, also Programme, die auf die Durchführung von CEP spezialisiert sind. Solche Implementierungen sind darauf optimiert, sogenannte *High Velocity* Datenströme, also flüchtige Ereignismengen mit einer hohen Durchsatzrate, zu verarbeiten und dabei ressourceneffizient zu arbeiten. [67]

Ein Vorteil dieser Methode ist, dass die Regeln dieser CEP-Engine oft in einer sehr kompakten Domänenspezifische Sprache (DSL) formuliert werden können, was zu einer gesteigerten Wartbarkeit führt. Einige Sprachen unterstützen zudem das Referenzieren und Kombinieren mit anderen Regeln, wodurch eine hohe Wiederverwendbarkeit gegeben ist. Als Beispiel sei eine Regel für das Produkt *Esper CEP* in Listing 2.1 angegeben, die die Durchschnittwerte aller Werte bildet, die innerhalb der letzten 60 Sekunden innerhalb eines Ereigniskanals *Channel* eingegangen sind. Der resultierende Durchschnittswert, der bei jeder Änderung neu generiert wird, wird im Ereigniskanal *AverageValue* veröffentlicht. [67]

```
select avg(value) as AverageValue from Channel.win:time(60 sec)
```

Listing 2.1: Beispielregel für die Complex Event Processing Engine *Esper CEP*

Eine CEP-Engine verfügt dazu über eine eigene Speicherverwaltung, welche für das Regelwerk relevante Ereignisse vorhält, bis sie nicht mehr benötigt werden. Dies wird in der Regel mit einem sogenannten *Sliding Window* realisiert und kann, wie im o. g. Beispiel, zeitlich oder durch die Anzahl an Ereignissen beschränkt werden. Neben der Aggregation können auch Informationen zueinander korreliert werden, also in einen Kontext gebracht werden. Auch können Kausalitäten festgestellt werden, z. B. ob ein Ereignis eines Kanals ein Anderes ausgelöst hat. Diese Zusammenhänge werden in Abbildung 2.6 veranschaulicht. [67]

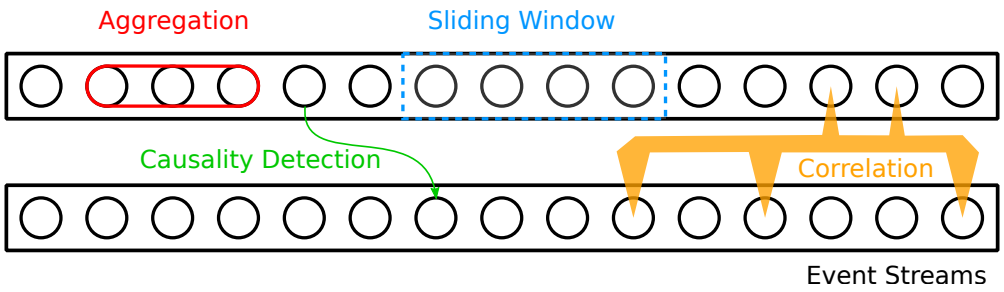


Abbildung 2.6: Verzeichnisthemen für Metadaten

Nachteilig ist, dass viele Lösungen nur in kommerziellen Versionen Clustering angeboten werden, was dazu führt, dass die komplette Datenverarbeitung bei einem Großteil der Nutzer pro Regelwerk auf nur einem Rechner erfolgen muss. Da Smart Homes auf einen verteilten Betrieb ausgelegt sein sollten und oft keine Ausführungsplattform ausweisen, in welcher sich die Rechenkapazität konzentriert, ist diese Variante nur schwer einsetzbar. [67]

Eigene Agenten Wenn man das Multiagentenschema auf diesen Teilaspekt anwendet, ergibt sich die Möglichkeit, jeden Teil der Datenverarbeitung als Agenten zu modellieren. So könnten Datenkonvertierer und Aggregatoren generische Agenten sein, die über ihre Konfiguration Regeln erhalten, wie Eingangs- zu Ausgangsnachrichten umgesetzt werden können. Dies führt zu einer geminderten Performance und höherem Konfigurationsaufwand, durch welche jedoch eine Entkopplung der einzelnen Verarbeitungsschritte und die Unabhängigkeit von Drittbibliotheken erkauft werden.

2.4.5 Agenteninteraktion

Nach der Analyse der Bedingungen für einzelne Agenten folgt die Betrachtung des Gesamtnetzwerkes, also wie Agenten miteinander kommunizieren und welche Anforderungen an die Kommunikation zu stellen sind.

Anforderungen

Um die günstigste Kommunikationsart für den Anwendungsfall zu ermitteln, ist zunächst grob einzugrenzen, aus welcher Klasse von Kommunikationsmethoden eine Variante ausgewählt werden soll. Dazu müssen Kernanforderungen aufgestellt werden. [12, S. 2]

Qualitätsgüte Für die Kommunikation innerhalb eines Smart Environments ist es für einige Nachrichten kritisch, garantiert übermittelt zu werden. Wenn z. B. eine Alarmanlage dem restlichen System signalisieren soll, dass ein Einbruch stattfindet, so ist die Nachricht nicht zu verwerfen, wenn Systemkomponenten ausfallen, sondern wiederholt erneut zuzustellen.

Im Kontrast dazu gibt es Themen, die eine hohe Nachrichtenfrequenz aufweisen. Der Verlust von einzelnen Nachrichten ist dort jedoch nicht kritisch. Diese Nachrichten mit einer solchen Garantie auszustatten, würde unnötig viele Ressourcen konsumieren und das System verlangsamen. [31, S. 232]

Aus diesem Grund muss es möglich sein, für verschiedene Themen unterschiedliche Dienstgüten zu konfigurieren. Für die Übermittlung sollte es dabei zumindest möglich sein, die Garantien *At-Most-Once* und *Exactly-Once* einstellen zu können. Dies lässt sich zwar auch dadurch implementieren, dass kommunizierende Agenten ein eigenes Kommunikationsprotokoll auf Applikationsebene einsetzen, um zu validieren, ob ein Zustellversuch erfolgreich war. Dies führt jedoch zu einer engen Kopplung und einem Bruch des Schichtenmodells, weswegen dieser Ansatz verworfen wird. Grundsätzlich sollte für alle Themen zusätzlich gegeben sein, dass Nachrichten in der richtigen Reihenfolge und korruptionsfrei übermittelt werden. [41]

Sicherheit Da innerhalb eines Smart Environments sensible Daten anfallen und übertragen werden können, ist es von Bedeutung, diese sicher zu handhaben.

Daher ist es wichtig, dass Daten verschlüsselt übertragen werden, um das Mitschneiden oder gar verändern dieser zu vermeiden. Quellen führen hier an, dass sich Smart Environments in einem geschützten Netzwerk befinden und somit keine Verschlüsselung benötigen. Dies ist jedoch ein Antipattern und nicht auf einen langfristigen Betrieb ausgelegt, in welchem sich solche Gegebenheiten ändern können. In Smart Homes bspw. teilen sich CPS und Arbeitsplatzrechner der Bewohner den Zugriff, wodurch Sicherheitsbrüche aus diesem Umfeld Einfluss auf das Smart Environment nehmen können. [48, S. 1508]

Weiterhin ist relevant, dass legitime Teilnehmer des Smart Environments nur Zugriff auf für sie relevante Daten erhalten, um die Auswirkungen von Sicherheitsbrüchen zu minimieren. Erlangt ein Angreifer bspw. Zugriff auf die Ausführungsplattform, auf der ein Agent für die Steuerung der Beleuchtung betrieben wird, so kann der Angreifer nur z. B. auf dieses Subsystem nehmen und nicht bspw. auf Überwachungskameras. Durch Angriffslücken kann u. U.. auch dies umgangen werden, jedoch wird so die Komplexität eines Angriffs höher, wenngleich auch der Konfigurationsaufwand dadurch leicht ansteigt. [48]

Beide Anforderungen können dadurch angegangen werden, dass die Transportverschlüsselung Transport Layer Security (TLS) eingesetzt wird. Dies erfüllt den ersten Punkt, die Verschlüsselung der Verbindung, vollständig. Zudem können dadurch beide Teilnehmer der Kommunikationsverbindung sich gegenseitig authentifizieren, wobei zusätzlich sichergestellt werden muss, dass Dateninhaber Abfragen gegen konfigurierte Zugriffsregeln validieren. [42]

Selektierbarkeit Wie erwähnt, können in einem Smart Environment eine große Menge an Kommunikationsdaten auftreten, die Themen zugeordnet werden können. So könnte ein Thema Temperaturmesswerte beinhalten, die alle 5 Minuten aktualisiert werden, ein anderes Thema den Zustand einer LED-Kette, die mit 30 Hertz aufgefrischt wird. Wenn beide Themen nicht trennbar ausgelegt werden, müssen leichtgewichtige Agenten, die nur an der Temperaturmessung interessiert sind, auch das andere Thema verarbeiten und verwerfen, was zu verschwendeter Rechenleistung führt. Die Aufteilung der Daten in verschiedene Themen oder Kanäle ist deswegen zwingend. Die Aufteilung kann auf mehrere Wege umgesetzt werden. So können Daten zwecks Trennung über verschiedene Verbindungskanäle gesendet werden und mit der Metainformation versehen werden. Dies kann dadurch realisiert werden, dass vermerkt wird, zu welchem Thema sie gehören oder mittels eines Taggingmechanismus Attributen zugeordnet werden können. [75]

Zustandshaltung Je nach Art der Komponenten im Smart Environment, können diese kurzlebig sein oder das Netzwerk aufgrund ihrer Mobilität kurzfristig betreten oder verlassen. So könnte sich auf dem Smart Phone eines Benutzer eine Anwendung befinden, die eine Übersicht über den aktuellen Zustand des Smart Homes unmittelbar nach dem Öffnen aufzeigt. Wenn Informationen innerhalb der Umgebung direkt von den Quellen angefordert werden müssen, muss die Anwendung nach dem Start zuerst eine Reihe von Komponenten in der Umgebung kontaktieren, um den Zustand erfassen zu können. Dazu muss gewährleistet sein, dass diese Komponenten aktuell erreichbar sind und sich in einem Zustand befinden, der das Erfragen der Informationen erlaubt. Ist dies nicht der Fall, sind die Informationen temporär nicht zugänglich. [34]

Aus diesem Grund ist es vom Vorteil, dass der Zustand des Smart Environments innerhalb der Kommunikationsschicht vorgehalten wird, um z. B. ausgefallenen Agenten ein schnelles Sammeln von benötigten Informationen zu bieten und damit einen schnellen Start zu ermöglichen. Dies erfordert eine Kommunikationsmethode, die über Clustering benötigte Informationen auf mehrere Plattformen replizieren kann oder einen Serverdienst, über welchen Agenten kommunizieren. Eine direkte Peer-to-Peer Kommunikation ist so nur schwer möglich, auch wenn Ansätze dafür vorhanden sind. [64]

Kommunikationsweise

Wie eben ermittelt, bietet sich Peer-to-Peer Kommunikation für den Anwendungsfall nicht an, wodurch Kommunikationspattern wie Remote Procedure Call (RPC), Notification und Message Queuing nicht weiter zur Wahl stehen. Es ist also ein Serverdienst erforderlich, der die Kommunikation zwischen den Agenten durchführt. Diese Form von Dienst ist allgemein als Messagebroker bekannt. [28]

Ein Kommunikationspattern, welches potenziell alle eben aufgestellten Anforderungen erfüllen kann, ist Publish/Subscribe. Mit diesem Prinzip bestehen zwei Arten von Kommunikationsteilnehmern. Publisher veröffentlichen Nachrichten zu bestimmten Themen. Subscriber abonnieren Themen und erhalten in diesen veröffentliche Nachrichten. Alle Nachrichten und Themen befinden sich in einem Verzeichnis, welche in der Praxis von einem Messagebroker verwaltet wird. [12, 2]

Je nach Implementierung ist es möglich, Nachrichten für bestimmte Themen vorzuhalten. Nach festgelegen Regeln wird bestimmt, welche Nachricht der aktuelle Stand des Themas ist. Wird durch einen Subscriber ein Thema abonniert, erhält dieser sofort die aktuelle Nachricht, ohne auf eine neue Veröffentlichung warten zu müssen. [12, 3]

Datenzugriff

Nach der Klärung der Kommunikationsweise ist zu ermitteln, was für die effektive Nutzung von Publish/Subscribe unter der Berücksichtigung der Problemstellung nötig ist. Wie in Abschnitt 2.4.4 beschrieben, treten in einem Smart Environment eine Vielzahl von verschiedenen Datentypen auf. Dies beinhaltet Wahrheitswerte, Ganzzahlen, Gleitkommazahlen, Strings, Listen sowie verschachtelte Datentypen, die u. a. mit JSON und YAML dargestellt werden können.

An dieser Stelle ist relevant, dass für maximale Kompatibilität darauf geachtet werden sollte, Informationen aufgabenangemessen zu strukturieren. So kann ein einzelner Luftdruckmesswert als ein JavaScript Object Notation (JSON) String serialisiert werden, um auf einem Zielsystem mit einer Hochsprache wie JavaScript ohne die Notwendigkeit für Metainformationen oder Zusatzbibliotheken aus dem Bytestrom deserialisiert werden, da dies bereits durch den Sprachstandard unterstützt wird. Auf einem Embeddedsystem mit begrenzten Hardwareressourcen gestaltet sich dieser Prozess komplexer und führt zu einer längeren Entwicklungszeit. Serialisiert man den Messwert als Integer, so können beide Zielplattformen den Wert effizient und direkt deserialisieren, solange zwischen Quelle und Senke Konsens über das Datenformat herrscht. Dieser Konsens könnte während der Kopplung der Agenten händisch überprüft werden, was jedoch zu einer potenziellen Fehlerquelle führt. Bei der Konzeption der Architektur ist deswegen besonderes Augenmerk darauf zu legen, wie die Validierung des Formats automatisch durchgeführt werden kann. [47]

Zusätzlich muss festgelegt werden, welche Informationen zu einem Thema gruppiert werden sollen. Wenn für eben genannten Messwert ein Zeitstempel relevant ist, sollte dieser sich innerhalb der Nachricht des Messwertes befinden und nicht getrennt, damit partielle Aktualisierungen des Publish/Subscribe-Verzeichnisses nicht zu inkonsistenten Informationen führen. [28]

2.4.6 Ausführungsplattformen

Die Abstraktionsschicht wird in Form von Software ausgeliefert, welche auf einer Ausführungsplattform eingesetzt wird. Um die Entscheidung treffen zu können, welche Arten unterstützt werden sollen, muss zuerst eine Übersicht über relevante Auswahlfaktoren aufgestellt werden. [52, 5 f.]

Universelle Komponenten

Prozessor Ein jedes Computersystem benötigt einen Prozessor, welchen Funktionsumfang dieser bereitstellen ist jedoch variabel. Server und Desktopcomputer setzen heutzutage Prozessoren mit *x86_64* Architektur ein, welche hohe Leistung bieten, jedoch sehr wenig Unterstützung für die Integration in ein Embeddedsystem aufweisen. Dies erweist sich für hardwarenahe Agenten als Nachteil, ist jedoch für rechenintensive Aufgaben innerhalb des Smart Environments wie Bildverarbeitung von Vorteil. [14]

Auf der gegenüberliegenden Seite des Leitstungsspektrums liegen Mikrocontroller (MCU), welche einen festen Programmcode einprogrammiert bekommen und diesen mittels eines eingebauten Mikroprozessors ausführen. Da das vollständige System innerhalb eines Chips untergebracht werden muss, verfügen diese Chips über geringe Ressourcen und werden üblicherweise mit leichtgewichtigen Echtzeitbetriebssystemen betrieben, um diese effizient auszunutzen. Für Aktualisierungen des Programms muss der Chip entweder über ein Over the Air (OTA) Update verändert oder aus der lokalen Schaltung entfernt und das Programm mittels eines Programmers überschrieben werden. Vor allem Ersteres erfordert zusätzliche Sicherheitsmaßnahmen, um den neuen Programmcode zu validieren, was sich aufgrund der Hardwarebeschränkungen als schwierig erweisen kann. Im Gegenzug dazu sind Microcontroller in der Lage, in elektronische Schaltungen eingebettet zu werden. Dies bedeutet, dass sie Zustände innerhalb dieser Schaltung erfassen und Einfluss auf sie nehmen können, was eine Voraussetzung für CPS ist. [56]

Einen Mittelweg bieten Einplatinencomputer, welche nicht auf nur einen Chip beschränkt sind, sondern sich über ein Printed Circuit Boards (PCB), also eine Platine, erstrecken. Die zusätzliche Ausdehnung ermöglicht eine Ausstattung, die vom Umfang her, mehr denen eines Desktopcomputers ähnelt. Aufgrund der zusätzlichen Ressourcen ist es zudem möglich, vollwertige Betriebssysteme zu verwenden, um eine größere Kompatibilität für bereits existierende Programme herzustellen. Wie Mikrochips sind auch Einplatinencomputer in der Lage, in Schaltungen eingebettet zu werden. Sie bieten also ein ausgewogenes Verhältnis zwischen Integrierbarkeit und Leistung und sollten für die Entwicklung als Hauptziel ins Auge gefasst werden. [29]

Betriebssystem Wie im vorhergegangenen Abschnitt bereits angerissen, wird auf allen, für diese Anwendung relevanten, Prozessoren ein Betriebssystem eingesetzt. Der Funktionsumfang von diesem besteht je nach Einsatzumgebung aus mindestens einem Scheduler, Speicherverwaltung sowie Peripheriezugriff und kann mit einer vollständigen Distribution bis hin zu einer grafischen Oberfläche sowie einem Ökosystem von Programmen und Kernelmodulen reichen. [74]

Die Verwaltung des Peripheriezugriffs ist sinnvoll, um den Zugriff auf diese effizient und performant zu halten. Ein direkter Zugriff auf jene ist über Methoden wie Direct Memory Access (DMA), also die Kommunikation mit Peripherie über festgelegte Adressen im Hauptspeicher möglich, erhöht jedoch die Wahrscheinlichkeit für Fehler und deren negative Auswirkungen. Zudem verfügen einige Betriebssysteme über eine umfangreiche Sammlung von Treibern in Form von Kernelmodulen, mit welchen z. B. der Zugriff auf Bussysteme auf Anwendungsseite vereinfacht werden kann. [74]

Kompaktere Distributionen bringen den benötigten Basisumfang von Bibliotheken nicht mit, wie sie z. B. für die Unterstützung von TLS erforderlich ist, weswegen bei diesen Systemen die Bibliothek in die Auslieferung des Programms eingebunden werden muss. Damit wird die Aktualisierung von teils sicherheitskritischen Bibliotheken vom Aktualisierungszyklus des Programms abhängig gemacht, wodurch kritische Updates u. U.. erst spät eingepflegt werden. Für eine Abstraktionsschicht, die um Netzwerkfunktionen herum zentriert und somit für Angriffe exponiert ist, ist dies nicht hinnehmbar. Aus diesem Grund sollten Betriebssystemdistributionen ins Auge gefasst werden, die die einfach Aktualisierung von Bibliotheken ermöglichen und eine gute Unterstützung für Zielplattformen bieten. [74]

Netzwerkanbindung Wie in 2.4.2 aufgezeigt, bestehen für Geräte unterschiedliche Möglichkeiten, an das lokale Netzwerk angebunden zu werden. Im diesem Kontext sind auch die Instanzen der Abstraktionsschicht ein Gerät, welches integriert werden und somit die Integrationsmethode geplant werden muss. Die Schicht soll u. a. die Möglichkeit anbieten, Daten zu verarbeiten, die eine hohe laufende Bandbreite verursachen und auf geringe Latenzen angewiesen sind. Hinzu kommt, dass auf Dienste aus dem Internet zugegriffen werden soll, sei es das Abrufen von aktuellen Börsendaten oder die Kommunikation mit anderen Smart Environments über das Internet. Aufgrund dieser beiden Bedingungen ist es ratsam, die Instanzen über Ethernet (also kabelgebunden) ans lokale Netzwerk anzubinden. Zum einen können Instanzen damit direkt aufs Internet zugreifen, zum anderen sind dadurch die Latenz minimal und die Durchsatzrate, entsprechende Hardware vorausgesetzt, im Gigabitbereich. Zudem ist eine höhere Störungssicherheit als mit WLAN gegeben. WLAN selbst eignet sich jedoch als Alternative, wenn keine Kabelanbindung zur Verfügung steht, wenn die Nachteile hinreichend berücksichtigt werden. [84]

Weiterhin sei hier angemerkt, dass für die Verbindung von Agenten keine Bussysteme, wie CAN-Bus, in Betracht gezogen werden. Dies liegt daran, dass diese Systeme, neben ihrer Inkompatibilität zum Internet und den dadurch entstehenden Komplikationen, Anforderungen bedienen, die für dieses Szenario nicht kritisch sind. Für die Kommunikation zwischen Agenten ist Echtzeitfähigkeit nicht kritisch, da jeder Agent für seine Basisfunktionalität inklusive der kritischen Ausführungsabschnitte selbstständig ausführen kann. Zudem verfügen Bussysteme nur selten über eingebaute Sicherheitsmechanismen gegen böswillige Manipulation, welche unter Beachtung der Anwendungssituation kritisch sind. [84]

Gruppen

Neben den Unterschieden, sie sich aus der Konzeption der Komponenten der Plattformen ergeben, ergibt sich eine Gruppierungsmöglichkeit anhand des primären Zwecks der Plattform und den daraus resultierenden Besonderheiten. Grundsätzlich lassen sich die hier relevanten Plattformen in drei Gruppen einteilen:

Server Server zeichnen sich dadurch aus, dass sie von allen Gruppen potenziell über die meisten Kapazitäten von Ressourcen wie z. B. Prozessorleistung, Hauptspeicher und Festplattenschreibrate verfügen, und sich dadurch als das Mittel der Wahl für komplexe Berechnungen anbieten. Server betreiben Anwendung, die so weit wie möglich von der darunter liegenden Hardware abstrahiert oder virtualisiert ist. Dadurch ergeben sich Vorteile wie verbesserte Portabilität und höhere Kompatibilität.

Embeddedgeräte Während Server, oft in relativ geringer Anzahl für eine Aufgabe zuständig sind, besteht ein Smart Environment aus einer Vielzahl von eingebetteten Systemen. Wie erwähnt verfügen sie dabei über beschränkte Ressourcen und ihre Lokalität ist entscheidend, da sie in Schaltungen eingebaut werden. Dadurch ergibt sich, dass Software auf Embeddedsystemen stark an die unterliegende Hardware gekoppelt wird, um diese korrekt ansprechen zu können. Während Server auf Leistung optimiert sind, verfügen Embeddedsysteme also nur über geringe Ressourcen, weswegen diese nur wenn nötig eingesetzt werden sollten.

Arbeitsplatzrechner/Mobile Geräte Arbeitsplatzrechner und mobile Geräte, also bspw. Smart Phones und Tablets, orientieren sich von den Ressourcen her zwischen den beiden anderen Gruppen und vereinigen einige ihrer Eigenschaften. Sie können sich mit ihrer Leistungsfähigkeit mit denen der Server messen, im Falle von mobilen Geräten ist jedoch auch der Stromverbrauch entscheidend. Ihre Besonderheit ist, dass sie eine direkte, grafische Schnittstelle zum Benutzer haben, um Informationen anzuzeigen und Befehle entgegenzunehmen. Soll ein Agent als Nutzerschnittstelle verwendet werden, so ist er potenziell kurzlebig, da er bedarfsweise vom Nutzer aufgerufen wird, während die Agenten der anderen Gruppen langlebig sind.

Schlussfolgerung Wie in Abbildung 2.7 aufgezeigt, liegen für die drei erwähnten Betrachtungspunkte teilweise widersprüchliche Anforderungen vor. Da die Abstraktionsschicht nicht auf eine einzelne Gruppe beschränkt werden soll, ist es erforderlich, alle erwähnten Gruppen zu unterstützen. Dies kann dadurch realisiert werden, dass den Entwicklern Optionen angeboten werden, die Entwicklung von Agenten in Hinsicht einer oder mehrerer Kriterien zu vereinfachen, ohne ihnen jedoch die Entscheidung abzunehmen, in welche Richtung optimiert werden soll.

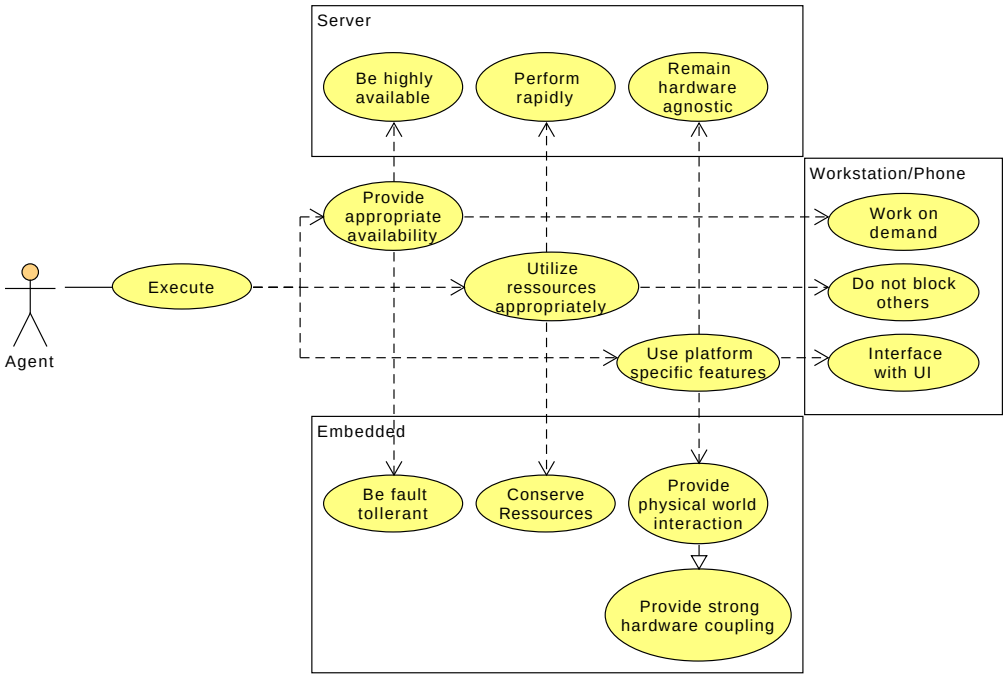


Abbildung 2.7: Unterschiede zwischen Plattformgruppen

2.4.7 Erwarteter Entwicklungsablauf eines Agenten

Die Entwicklung eines Agenten läuft generell nach demselben groben Schema ab, welches zum besseren Verständnis hier analysiert werden soll. Grundsätzlich richtet sich dieses Schema nach üblichen Entwicklungsmustern, welche für den Sonderfall Smart Environment angepasst wurden. Grundsätzlich sollte diese Entwicklung spiralförmig durchgeführt werden, d. h. bei der Einführung neuer Anforderungen beginnt dieser Ablauf von Neuem. [15]

Projektplanung Zu Beginn des Projektes muss der Entwickler eines Agenten Planungen anstellen. Dies umfasst die Ermittlung der benötigten Funktionen, ggf. die nötige Performance, die benötigte Hardware und die Interaktion mit anderen Agenten. Je nach Entwicklungsmodell können in den folgenden Schritten zusätzliche Planungs- und Überarbeitungsphasen eingelegt werden, eine initiale Konzepterstellung ist aber auf jeden Fall erforderlich. Diese umfasst insbesondere die Planung, wie hinzukommende Agenten mit den bereits bestehenden interagieren und welche Fehlerquellen auftreten können.

Hardwareumsetzung Um möglichst schnell validieren zu können, ob die entwickelte Software korrekt mit der benötigten Hardware interagiert, ist diese möglichst früh in der Entwicklung anzuschaffen, frühestens jedoch nachdem überprüft wurde, dass die Hardware die gegebenen Anforderungen erfüllen kann. Ist dies gegeben, muss diese beschafft und auf Fehlerfreiheit getestet werden.

Betriebssystemeinrichtung Insofern nicht ein bereits bestehender Controller für die Anbindung von Peripherie verwendet wird, muss ein neu angeschaffter Controller zunächst mit einem Betriebssystem eingerichtet werden. Je nach Hersteller und Modell stehen verschiedene Distributionen zur Verfügung, aus welchen gewählt werden kann. Nach der Wahl muss dieses System konfiguriert werden. Dies beinhaltet die Anweisung an den Kernel, benötigte Treibermodule zu laden und Schnittstellen zur Peripherie zur Verfügung zu stellen. Hinzu kommen die Installation der Abstraktionsschicht und die Konfiguration der Anbindung an das Netzwerk. Anschließend muss eine Instanz als Systemdienst eingerichtet werden.

Konfiguration Wenn ein Agent anhand eines konkreten Einsatzfalls implementiert und direkt Tests am Zielsystem durchgeführt werden sollen, so ist der Agent vor der Fertigstellung zu konfigurieren. Die Konfiguration umfasst die Definition und Erstellungsanweisung für den Agenten, Einstellungen spezifisch zur Problemstellung, z. B. wie oft pro Sekunde ein Sensor abgefragt wird und Kopplungsinformationen, also unter welchem Thema der Agent z. B. Sensorwerte im Netzwerk veröffentlichen soll.

Automatisierte Tests Test-driven Development als Vorgehen für die Entwicklung von Software zwecks Steigerung der Codequalität und Wartbarkeit beinhaltet das Erstellen von automatisierten Tests, optimalerweise vor Beginn der Entwicklung, um diese frei von Annahmen umsetzen zu können. Dies kann unter Zuhilfenahme von Mockups umgesetzt werden, wodurch ein isoliertes Testen des Agenten ermöglicht wird.

Nutzlastimplementierung Auf die Vorarbeit folgt die Umsetzung des Agenten. Dazu wird innerhalb eines Moduls unabhängig von der Distribution des Agenten die Implementierung angelegt und entwickelt. Um dem Multiagentenschema treu zu sein, kann es nötig sein, mehrere eigenständige und möglichst wiederverwertbare Agenten zu entwickeln, um eine komplexere Problemstellung zu lösen.

Integrationstests Nach den Umsetzungen und möglicherweise ersten Tests während der Entwicklung ist es wichtig, einen vollständigen Integrationstest durchzuführen, um eventuelle Konflikte innerhalb des Netzwerkes frühzeitig beseitigen zu können. Dies bedeutet, dass das Verhalten der neuen Agenten bei einem Testlauf mit allen betroffenen, bereits existierenden, Agenten evaluiert werden sollte.

Wartung Neben der regulären Aktualisierung der Ausführungsplattform ist es unter Umständen nötig, den Agenten im Laufe seines Lebenszyklus anzupassen. So kann Funktionalität erforderlich sein, die zur Erstellungszeit noch nicht berücksichtigt wurde oder vom Agenten benötigte Drittbibliotheken erfahren eine Aktualisierung, die Anpassungen der Agenten erfordert.

2.4.8 Ähnliche Projekte

Zur Abgrenzung sei hier auf weitere Projekte verwiesen, welche ein ähnliches Ziel wie dieses hier verfolgen.

IoTAgent *IoTAgent* hat das Ziel, die Anbindung von IoT Systemen an Cloudsysteme zu vereinfachen und die Verarbeitung von Big Data zu entkomplizieren und hat dabei einen starken Bezug auf Smart Cities. Das Projekt versucht dabei, für unerfahrene Programmierer geringe Einstiegshürden zu liefern und möglichst alle Anwendungsfälle aus diesem Bereich abzudecken. Teil des Projekts ist eine Vielzahl von Agenten, die als Schnittstelle zwischen dem *Fiware* Protokoll und *NGSI* dienen. Daten werden textbasiert und in der Regel als eine, auf Comma-separated values (CSV) basierenden DSL, angegeben. Das Projekt erweist sich jedoch im Bereich von Smart Environments als einschränkend, da durch die eben genannten Designentscheidung Projekte mit sog. *High Velocity Data* nur schwer übertragen werden können und Ausfallsicherheit von Entwickler selbst umgesetzt werden muss. Wird ein Agent über das native Protokoll angebunden, wird mit dem *Request-Response* verfahren kommuniziert, was potenziell eine ständige Kommunikationsbereitschaft des Agenten erfordert und zu einem hohen Stromverbrauch führt. Diese Arbeit zeigt, inwieweit sich das Agentenmodell praktisch auf das Umfeld von Smart Environments anwenden lässt. [59]

IoTivity *IoTivity* wird von der *Linux Foundation* verwaltet und von der *Open Connectivity Foundation* gesponsert. Es erfüllt den *OCF*-Standard, welcher das Ziel hat, die Kommunikation zwischen IoT Komponenten zu definieren. Dieser verwendet ein auf User Datagram Protocol (UDP) basierendes Verfahren und kann ebenfalls mit Brückenagenten an andere Message Brokers angebunden werden. *IoTivity* liefert native Unterstützung für *OCF* für die Programmiersprache C und Bindings für weitere. Neben dieser Funktion überlässt das Projekt die vollständige Entwicklung der Agenten den Entwicklern selbst. Hier wird aufgezeigt, wie sich eine Umgebung realisieren lässt, die einem festgelegten Standard folgt, was kritisch für langlebige Systeme ist. [73]

mBed *mBed* ist eine Plattform für die Entwicklung von IoT Systemen. Sie ist dabei auf Ausführungsplattformen spezialisiert, die auf *ARM*-Prozessoren basieren. Hardwareferne Agenten können jedoch auch auf jeder Linuxumgebung ausgeführt werden. Sie bietet zudem die Möglichkeit, den Zugriff auf Hardware und Netzwerk zu abstrahieren und Betriebssystemfunktionalität bereitzustellen. Leider ist diese Lösung konzeptionell stark an die eigene Cloudlösung gekoppelt, was eine Internetverbindung voraussetzt und für eine verminderte Robustheit sorgt. Dieses Projekt wird in vielen praktischen Projekten angewandt und gibt hinweise darauf, welche Funktionalität durch eine Abstraktionsschicht angeboten werden muss. [76]

Schlussfolgerung Die genannten Projekte bedienen ähnliche Anforderungen, unterscheiden sich aber in ihrer Designphilosophie grundsätzlich von dieser Lösung, weswegen eine Erweiterung der Projekte nur schwierig umzusetzen ist. Daher rechtfertigt sich die Lösung der Problemstellung als neues Projekt.

2.5 Fazit

Die Erkenntnisse der Analyse werden nun zusammengefasst und ein Ausblick darauf gegeben, welche Aspekte in Folgearbeiten ausgebaut werden können.

2.5.1 Zusammenfassung

Die Analyse hat ergeben, dass die Problemstellung durch einen technischen Ansatz gelöst werden kann. So sollte die Abstraktionsschicht unterstützen, dass mehrere Instanzen innerhalb eines Netzwerkes verschlüsselt miteinander kommunizieren können, um komplexe Aufgaben zu erfüllen. Jede Instanz soll einen oder mehrere Agenten beherbergen, welche über lokale Kommunikationsmittel mit Sensoren und Aktoren interagieren oder losgelöst von der Umgebung arbeiten können.

Eine dezentrale Architektur ist entscheidend, damit das Smart Environment auch bei partiellem Ausfall seine Funktionalität noch bereitstellen kann. Dazu gehört eine Methode, mit welcher Informationen dezentral verarbeitet werden könnten, um die entstehende Last auf mehrere Plattformen verteilen zu können. Für die Synthese von Agenten und deren Kommunikation muss ein Leitfaden erstellt werden, an welchem die Größe eines Agenten, sowie die Aufteilung von Daten und deren Serialisierung ermitteln zu können, um die Fragmentierung eines bestehenden Systems zu verhindern.

Für Agenten in Smart Environments existieren viele wiederkehrende Aufgaben, die durch eine Abstraktionsschicht übernommen werden sollten und in der Analyse aufgestellt wurden. So sind einfaches Debugging, einheitliche Konfiguration, Peripheriezugriff, Zustandsverwaltung und Aufgabenmanagement entscheidende Punkte.

Wie genau diese Anforderungen umgesetzt werden können, wird im folgenden Kapitel *Architektur und Implementierung* beschrieben.

2.5.2 Ausblick

Während die Abstraktionsschicht bereits eingesetzt wird und gezeigt hat, dass sie effektiv das geplante Soll umsetzen kann, können Erweiterungen vorgenommen werden.

Wie bereits erwähnt, kann die Konfiguration des Systems durch eine grafische Oberfläche deutlich vereinfacht werden. Die bestehende Testumgebung kann erweitert werden, um mit weniger Implementierungsaufwand mehr potenzielle Fehlerquellen testen zu können. Ein Toolkit für die Programmiersprache C kann umgesetzt werden, um eine deutlich größere Anzahl an Embeddedplattformen zu unterstützen.

Zudem kann genauer untersucht werden, wie die Performance der Abstraktionsschicht weiterhin verbessert werden kann und welche allgemeinen Optimierungen vorgenommen werden können, um die Schicht zu noch mehr Anwendungsfällen kompatibel zu machen.

3 Architektur und Implementierung

Dieses Kapitel befasst sich mit der Architektur der erstellten Lösung. Die Beschreibung wird vom Publish/Subscribe-Verzeichnis aus bis hin zum Agenten hinab beschrieben.

3.1 Publish/Subscribe-Verzeichnis

Während in der Analyse in Abschnitt 2.4.5 ermittelt wurde, dass ein Publish/Subscribe-Verzeichnis sich für das Projekt eignet, muss hier das konkrete Protokoll und mögliche Implementierungen für den Messagebroker besprochen und eine Struktur für die Datenhaltung und die Speicherung von Metadaten gefunden werden.

3.1.1 Protokoll

Es existieren mehrere Protokolle, welche Publish/Subscribe für IP Netzwerke anbieten. Um aus diesen das Vorteilhafteste zu ermitteln, wird ein Vergleich angestellt. [12, 3]

AMQP AMQP erfreut sich weiter Verbreitung und verfolgt das Ziel, alle häufig auftretenden Problemstellungen im Bereich der Nachrichtenübermittlung bedienen zu können. Nachrichten können entweder Themen zuordnet oder auch mit Tags versehen werden, die die Zuordnung zu verschiedenen Eigenschaften ermöglichen. Nachrichten werden auf der Seite des Messagebrokers Queues zugeordnet. Je nach Modellierung können mehrere Subscriber Nachrichten aus einer Queue z. B. für Round Robin Datenverarbeitung verwenden oder jeder Subscriber erhält seine eigene Queue mit einer Kopie jeder eingehenden Nachricht.

Während sich AMQP als universelles Nachrichtenprotokoll eignet, ergeben sich für diesen Anwendungsfall Probleme. Ohne Erweiterungen, die die Anpassung von sowohl Messagebroker als auch Clienten voraussetzen, ist es bspw. nicht möglich festzustellen, ob bestimmte Clienten erreichbar sind, da u. U.. keine Testamentnachrichten unterstützt werden. Zudem kann die Dienstgüte nur als *At-Most-Once* und *At-Least-Once* garantiert werden. [53]

MQTT MQTT ist ein leichtgewichtiges und platzeffizientes Protokoll mit besonderem Augenmerk auf die Anbindung von Embeddedsystemen. Der Standard umfasst nur eine geringe Anzahl von Nachrichten, wodurch das Entwickeln von neuen Clienten schnell umgesetzt werden kann. Integriert sind alle erforderlichen Dienstgüten und die Möglichkeit, sog. Testamentnachrichten zu hinterlegen, die bei der unsauberen Trennung des Clienten vom Messagebroker veröffentlicht werden. Zudem kann über Flags jeder Nachricht angegeben werden, ob diese für ihr Thema als letzter gültiger Wert zurückgehalten werden soll. Während jede Nachricht einem expliziten Thema zugeordnet werden muss, das wie ein Verzeichnispfad strukturiert werden kann, können Subscriber mittels Wildcards mehrere Themen gleichzeitig abonnieren. [11]

Neues Protokoll Die komplette Neuentwicklung eines Protokolls hätte den Vorteil, dass die Auswahl an Funktionen komplett an das Szenario angepasst werden kann. Leider ergibt sich dadurch ein hoher Implementierungs- und Wartungsaufwand für alle zu unterstützenden Plattformen. Zudem würde dadurch die Anbindung von Drittkomponenten nur schwer möglich sein. Aus diesen Gründen wird diese Option nicht weiter in Betracht gezogen. [12]

STOMP STOMP ist wie HTTP ein textbasiertes Protokoll zum Austausch von textbasierten Nachrichten. Messagebroker ordnen Nachrichten anhand von *Destinations* zu, deren Semantik der Serverimplementierung überlassen ist. Ebenso freigelassen ist die Übermittlungssemantik, also z. B. mit welcher Dienstgüte Nachrichten übertragen werden. Da Aufgrund dieser Unsicherheit die Abstraktionsschicht gegen eine konkrete Implementierung konzipiert werden müsste und dadurch eine hohe Kopplung zu dieser entsteht, entfällt diese Lösung. [45]

XMPP Extensible Messaging and Presence Protocol (XMPP) zeichnet sich durch seine universelle Verwendbarkeit zur Übertragung von Nachrichten aller Art aus. Entwickler können zur Erweiterung des Standards Vorschläge einreichen, die innerhalb der Entwicklergemeinschaft besprochen werden und nach ausreichender Bearbeitung ein Teil des Sprachstandards werden. Zwar dient dieses Protokoll hauptsächlich wie IRC zum Chatten zwischen zwei oder mehreren Teilnehmern, ein Erweiterungsvorschlag definiert jedoch ein Unterprotokoll für Publish/Subscribe. Leider hat dieser den Draft-Status seit über 10 Jahren nicht verlassen und wird von den wenigsten Clienten unterstützt. Bis hier wesentliche Fortschritte gemacht werden, sollte ein Protokoll gewählt werden, das konkreter auf die Problemstellung zugeschnitten ist. [63]

Entscheidung Grundsätzlich lassen sich sowohl AMQP als auch MQTT für den Anwendungsfall einsetzen, da beide die Anforderungen aus Abschnitt 2.4.5 erfüllen. Aufgrund des Fokus von MQTT auf Embeddedanwendungen und größeren Möglichkeiten zur Behandlung von Fehlerzuständen ist sich für dieses Protokoll entschieden worden.

3.1.2 Messagebroker Implementierung

Durch die Festlegung des Protokolls sind alle Informationen bekannt, die zur Anbindung an eine beliebige Messagebrokerimplementierung des Protokolls nötig sind. Trotzdem wird hier zur Einordnung auf zwei Messagebroker eingegangen, um aufzuzeigen, welche Auswahlmöglichkeiten hier bei der Zusammenstellung eines Systems bestehen. [66]

mosquitto Der MQTT Messagebroker mosquitto ist vor allem im privaten Umfeld weit verbreitet, da er universell auf verschiedenen Plattformen einsetzbar ist und trotz des relativ kleinen Quellcodeumfangs eine Vielzahl von Funktionen wie ACLs, Persistenz und Optionen zur Optimierung der Performance anbietet. Mehrere Instanzen des Messagebrokers können über sog. Bridges verbunden werden: Eine Instanz meldet sich Client bei der anderen Instanz an und überträgt ausgewählte Themen an und von diesem. [43]

emq EMQ ist mit der Programmiersprache erlang geschrieben und ist, im Gegensatz zu mosquitto, auf den Betrieb innerhalb eines Clusters ausgelegt. Mehrere Instanzen kommunizieren direkt miteinander, um ihren Zustand zu synchronisieren. Dadurch kann dieser Messagebroker vor allem in größeren Installationen, die Ausfallsicherheit erfordern, nützlich sein. Leider unterstützt EMQ den Einsatz auf Embeddedplattformen wie dem Raspberry Pi nur eingeschränkt, weswegen für die Evaluationsumgebung abzuwägen ist, ob die erhöhte Ausfallsicherheit den kontinuierlichen Wartungsaufwand rechtfertigt. [58]

3.1.3 Struktur

Wie bereits erwähnt, werden Themen bei MQTT wie ein Verzeichnispfad angeben und Verzeichnisse durch Separatoren getrennt. Die Bezeichnung und Struktur für die Ablage der Daten des Multiagentensystems ist an diesem Punkt also völlig offen. Um keine Situationen zu provozieren, die zu Konflikten innerhalb dieses Verzeichnisses führen, ist es jedoch wichtig, allgemeingültige Grundregeln festzulegen. Die Informationen im Verzeichnis lassen sich in mehrere Gruppen einteilen, die als erste Ebene des Pfades fungieren. Zur Veranschaulichung dient das Schaubild in Abbildung 3.1.

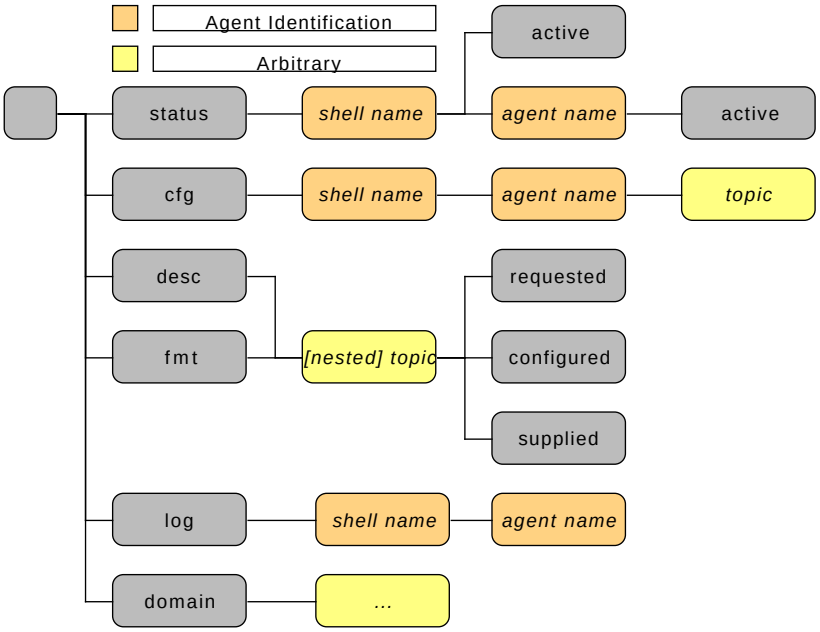


Abbildung 3.1: Aufbau des Stammes des Publish/Subscribe-Verzeichnisses

Agentenkonfiguration - cfg

Wie erwähnt benötigen Agenten u. U. zahlreiche Konfigurationsoptionen, welche unter dem Verzeichnis CFG/ platziert werden sollen. Diese werden im Verzeichnis unter dem Namen der Shell und des Agenten abgelegt. Die Optionen weisen keine weiteren Unterverzeichnisse auf.

Statusinformationen - status

Der Administrator des Systems soll schnell die Übersicht über den Ausführungszustand aller Agenten im System erhalten können. Dazu veröffentlicht jeder Agent in diesem Verzeichnis die Information, ob sein Ausführungszustand aktiv ist. Dies wird durch einen einfachen Wahrheitswert beschreiben. Durch den Aufbau des Verzeichnisses lassen sich, zu einem späteren Zeitpunkt, zusätzlich weitere Informationen hinzufügen. Auch wird für jede Shell, der Prozess in welchem ein oder mehrere Agenten untergebracht sind, der Zustand ebenfalls angegeben. Durch die Verwendung von Testamentnachrichten markiert der Messagebroker Shells automatisch als inaktiv, sobald sie sich unsauber trennen, wie z. B. nach einem Absturz oder Timeout durchs Netzwerk.

Logging - log

Unterhalb des Loggingverzeichnisses kann jeder Agent Logausgaben veröffentlichen, die ihn betreffen. Da viele Programmiersprache Loggingsysteme aufweisen, mit welchen Logeinträge einem Agenten und einer Priorität zugeordnet werden können, kann hier die Abstraktionsschicht automatisch Logeinträge veröffentlichen, ohne dass sich die Agentenimplementierung darum kümmern muss. Nachrichten in diesen Kanälen sollten nicht zurückgehalten werden, da die beinhaltete Information nur eine sehr begrenzte zeitliche Gültigkeit hat.

Daten - domain

Unter diesem Verzeichnis werden Nachrichten veröffentlicht, die zwischen Agenten ausgetauscht werden sollen. Das könnten z. B. Kalibrierungsdaten sein, die zwischen zwei Sensoragenten geteilt werden oder ein Temperaturmesswert, der auf einer Oberfläche dem Benutzer präsentiert werden soll. Für dieses Verzeichnis wird keine strikte Regel ausgegeben, da dessen Struktur von Einsatzgebiet und der Präferenz des Administrators abhängt.

Allgemeine Regeln für dieses Verzeichnis können dennoch angegeben werden, siehe dazu Abbildung 3.2. Als Beispiel sei ein Smart Home gegeben. Ähnlich wie bei Konzepten wie der strukturierten Verkabelung, die festlegt, wie Gebäude mit Netzwerkanschluss ausgestattet werden sollten, bieten sich als oberste Ebene das Gebäude, Stockwerk und der Raum an. Da die Evaluationsumgebung der Fallstudie über nur eine Etage verfügt, wird diese weggelassen, wodurch sich die, im Diagramm rot dargestellte, Obergruppe ergibt.

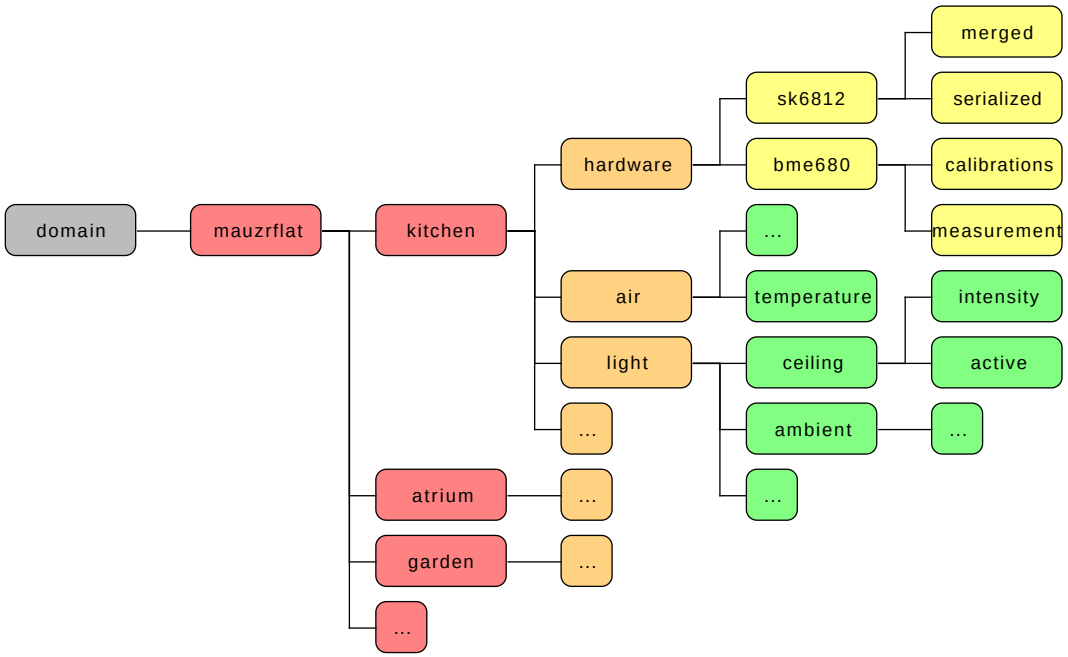


Abbildung 3.2: Beispielhafte Struktur des DOMAIN/ Verzeichnisses für Smart Homes

Die orange Gruppe darunter ist domänenspezifisch und orientiert sich am Inhalt, der in diesen Gruppen übertragen wird. So beinhaltet das Verzeichnis *air* Informationen, die die Luft betreffen, also deren Temperatur und weitere Daten.

Innerhalb des Verzeichnisses *light* sind die Leuchtgruppen *ambient*, also das Stimmungslicht, und *ceiling*, womit die Deckenleuchte gemeint ist. Für diese kann die Intensität eingestellt werden, also wie stark das Licht gedimmt wird. Ohne die Intensität zu beeinflussen, kann zudem das Licht komplett an oder ausgeschaltet werden. All diese Themen vereint, dass es sich um Informationen handelt, die unabhängig von speziellen Agenten sind. Sie repräsentieren eine Anwendungsschnittstelle, mit welcher interagiert werden kann, ohne genaueres Wissen über andere Agenten im System haben zu müssen, weswegen diese Themen in grün dargestellt werden.

Für die Kommunikationen zwischen mehreren konkreten Agenten wird das Unterverzeichnis *hardware*, markiert in gelb, verwendet. So tauschen die beiden Agenten, die für das Ansprechen des *BME680* in der Küche zuständig sind, Kalibrierungsdaten und unverarbeitete Messwerte im entsprechenden Unterverzeichnis aus. [4]

Aus diesem Aufbau lässt sich ableiten, dass das Verzeichnis zunächst nach physischer Lokalität und anschließend nach Kopplungsstärke unterteilt werden sollte. Beide Gruppierungsoptionen sollten auf alle Smart Environment generalisierbar sein, da dies elementare Eigenschaften sind, die sich verschiedenste Anwendungsfälle teilen dürften.

Themenbeschreibung - desc

Für jedes der eben genannten Themen wird der jeweilige Pfad mit dem Präfix DESC/ versehen und in diesem Pfad ein String veröffentlicht, der den Inhalt und Zweck der eigentlichen Nachrichten beschreibt. Dies soll dazu dienen, dass das System eine Selbstauskunft für verwendete Themen liefern kann. So könnte eine Verwaltungsoberfläche alle aktuellen Themen auflisten und für jedes Thema die zugehörige Beschreibung anzeigen. Dadurch können Entwickler und Konfiguratoren Funktionen direkt ermitteln, ohne umständlich in externen Dokumentationen suchen zu müssen. Da Teilnehmer mit unterschiedlichem Bezug zu einzelnen Themen stehen, sind einige Besonderheiten zu beachten, auf die im später folgenden Abschnitt 3.4.1 eingegangen wird.

Formatbeschreibung - fmt

Wie bei DESC/ handelt es sich bei FMT/ um ein Metaverzeichnis, in welchem für alle Oberverzeichnisse, außer DESC/ und FMT/, ein Unterthema angelegt wird. Als String wird hier die Information veröffentlicht, mit welchem Format Daten im beschriebenem Thema serialisiert werden. Bei diesem handelt es sich, ähnlich wie bei MIME-Types, um eine generell verstandene Beschreibung, mit der ein interagierender Agent automatisch zuordnen kann, mit welcher Methode die Daten zu behandeln sind. Damit wird sich im später folgenden Abschnitt 3.3.4 befasst. Auch hier gilt, dass verschiedene Parteien auf ein Thema zugreifen und deswegen Besonderheiten zu beachten sind. Obwohl es technisch möglich ist, auf einem Thema unterschiedlich formatierte Daten zu übertragen, wird in diesem Projekt darauf verzichtet und die einheitliche Formatierung innerhalb eines Themas erzwungen, da dies die Fehleranfälligkeit reduziert.

Externe Anwendungen - ext

Wenn externe Komponenten über die Möglichkeiten verfügen, eine Verbindung zum Messagebroker aufzubauen und mit dem Publish/Subscribe-Verzeichnis zu interagieren, soll dies möglich sein, ohne kompatible Agenten einzuschränken. So sollte unter DOMAIN/ die Annahme gelten, dass Format- und Funktionsbeschreibung für jedes Thema existieren, um effizient Agenten entwickeln zu können. Da die externen Systeme unter Umständen properietär sind und deswegen möglicherweise nur geringfügige Anpassungen wie z. B. das Ändern des Zieltopics ermöglichen, muss hier eine Sonderlösung gefunden werden. Eine Möglichkeit ist, bei der Konfiguration des Systems benötigte Metadaten manuell hinzuzufügen. Für den Fall, dass externe Komponenten einen dynamischen Bereich an Themen benötigen, wird das Verzeichnis EXT/ reserviert und deren willkürliche Benutzung erlaubt.

3.2 Provisioning und Deployment

Da der Betrieb der Abstraktionsschicht auf einem korrekt konfigurierten Betriebssystem stattfinden muss, ist der Einsatz eines Konfigurationsverwalters ratsam. Bei der herkömmlichen Installation von Ausführungsplattformen wird ein Großteil der Arbeitsschritte manuell ausgeführt. Konfigurationen werden für jede Systeminstanz von Hand angepasst und jeder Systemdienst einzeln konfiguriert. Ein Konfigurationsverwaltungswerkzeug hat das Ziel, automatisierbare Schritte in diesem Ablauf nach einem Regelwerk auf angebundene Systeme anzuwenden. [13][51]

3.2.1 Initiale Einrichtung

Bevor der Konfigurationsverwalter eingesetzt werden kann, ist eine initiale Einrichtung eines Grundsystems notwendig, welches diesen betreiben kann. Dies kann vereinfacht werden, indem dieses Grundsystem z. B. als klonbares Abbild bereitgehalten wird. Zusätzlich dazu sind weitere nicht generalisierbare, Schritte erforderlich, welche für das Zielsystem angepasst durchgeführt werden müssen:

Systemkonten In Embeddedsystemen ist häufig anzutreffen, dass hardwarenahe Userspace-Programme mir maximalen Rechten ausgeführt werden, damit diese den direkten Zugriff auf Hardware erhalten. Dies stellt ein erhebliches Sicherheitsrisiko dar und ist zu vermeiden. Das Konfigurationsverwaltungswerkzeug kann verwendet werden, um ein Systembenutzerkonto für die Abstraktionsschicht automatisch anzulegen und benötigte Privilegien je nach Aufgabe der Agenten in der Instanz entsprechend zugewiesen werden.

Systemkonfiguration Je nach Funktion der gewünschten Agenten sind Einstellungen im Betriebssystem erforderlich. So müssen z.B Kernelmodule geladen oder Device-Tree Einstellungen vorgenommen werden. Diese zusätzlichen Aktionen werden effizient vom Werkzeug aufgeführt, da dessen Regelwerk eine Übersicht über die komplette Systemeinstellung bietet und es somit Konflikte potenziell verhindert werden können.

TLS Zertifikate Die Kommunikation des Konfigurationsverwaltungswerkzeugs wird üblicherweise vom selbigen gesichert. Die Verbindung der Abstraktionsschicht mit dem Message Broker erfordert hingegen die Verwaltung einer Zertifizierungsstelle, um signierte Zertifikate und Verschlüsselung bereitstellen zu können. Dies und die Verteilung der Zertifikate kann das Tool übernehmen. Über Einträge im Regelwerk wird festgelegt, auf welchen Systemen Zertifikate für Abstraktionsschichtinstanzen bereitgestellt werden sollen.

Abhängigkeiten Die Abstraktionsschichtinstanzen setzen voraus, dass bestimmte Programme und Bibliotheken auf dem System installiert sind. Das Tool wird also konfiguriert, benötigte Abhängigkeiten über den Paketmanager des Systems zu installieren und aktuell zu halten.

Dienstkonfiguration Agenten sollen im Normalfall vom Prozessmanager des Betriebssystems verwaltet werden. Das Tool wird eingesetzt, um mittels Regelwerk die eingestellten Instanzen als Dienste zu registrieren und für den automatischen Start einzutragen. Innerhalb der erstellen Dienstdefinitionen muss vermerkt sein, welches Zertifikate-Paar jede Instanz zur Verbindung mit dem Message Broker zu verwenden hat, da sich sowohl Instanz als auch der Message Broker die Identität der Instanz aus dem Zertifikat ableiten.

Message Broker Für den Messagebroker, sei es eine einzelne Instanz auf einer Plattform oder ein Cluster von mehreren, fallen ebenfalls Konfigurationsaufgaben an. So müssen die serverseitigen Zertifikate und Konfiguration installiert werden, die sich entsprechend den eingesetzten Agenten ändern. Sollen z. B. ACLs zum Einsatz kommen, um Agenten nur Zugriff auf Themen zu gewähren, die sie für ihre Ausführung benötigen, so ist eine solche Liste automatisch zu generieren, um Übertragungsfehler zu minimieren und Aufwand einzusparen. Dies kann über entsprechend angepasste Regelwerke des Tools umgesetzt werden.

3.2.2 Aktualisierung

Neben der einmaligen Einrichtung von Agenten und Zusatzdiensten ist, wie in der Analyse angemerkt, eine regelmäßige und stetige Aktualisierung der Systeme notwendig. Dies beinhaltet mehrere Teilschritte, die in unterschiedlicher Häufigkeit ausgeführt werden müssen.

Systemupdates

Die Aktualisierung des Systems geschieht in der Regel über den Paketmanager der Distribution. Je nach System kann manuelle Intervention eines Administrators nötig sein, weswegen hier auf eine vollständige Automatisierung verzichtet werden sollte. Vielmehr sollte mittels Teilautomatisierung der Aufwand des Prozesses verkürzt werden. So kann die Aktualisierung mehrerer Plattformen manuell und zentral im Tool angestoßen werden.

Bereitstellung der Abstraktionsschicht

Damit eine automatische Aktualisierung durch das Tool erfolgen kann, muss die Abstraktionsschicht durch sie beziehbar sein. Da das Projekt mit Open Source Lizenz öffentlich verfügbar ist, ist eine private Verteilung nicht nötig. Anstelle dessen ist es kritisch, dass das bezogene Softwarepaket von den Maintainern der Software signiert wurde, um das Risiko von Manipulieren des Paketes zu verhindern.

Die Software soll automatisiert getestet sowie die Dokumentation automatisch generiert, und als Webseite verfügbar gemacht werden können. Um dies zu gewährleisten, muss von Anfang an darauf geachtet werden, dass das Projekt modular bereitgestellt wird. Wenn Agenten bspw. die Bildverarbeitungsbibliothek *OpenCV* verwendet sollen, sollten diese nicht mit der Abstraktionsschicht bereitgestellt werden, sondern in einem separaten Paket, da so die übrigen Programmteile ohne die Installation dieser umfangreichen Bibliothek vollständig getestet werden können.

Aktualisierung der Abstraktionsschicht

Das Tool wird für die Aktualisierung angewiesen, eine bestimmte (oder die neuste) Version des Programms von einer festgelegten Quelle zu beziehen und die Signatur zu validieren. Dazu wird ein Systempaket daraus gebaut und mittels des Paketmanagers des Systems installiert. Abschließend werden alle Agenten neu gestartet, um die Aktualisierung möglichst zeitnahe anwenden zu können.

3.3 Abstraktionsschicht

Nach der Beschreibung der Vorarbeit, dem Spezifizieren des Publish/Subscribe-Verzeichnisses und der Definition des Deployments wird sich nun mit der Abstraktionsschicht selbst beschäftigt. Dabei ist zu ermitteln, wie die Konfiguration des Gesamtsystems erfolgen soll, auf welche Weise sie umgesetzt wird, auf welchen Plattformen die Abstraktionsschicht lauffähig sein soll und aus welchen Komponenten sie besteht.

3.3.1 Konfiguration

Ein besonderes Augenmerk ist auf die Konfiguration des Gesamtsystems zu legen. Jeder Agent benötigt eine eigene, auf den konkreten Anwendungsfall zugeschnittene, Konfiguration. In komplexeren Systemen kann dies zu einer großen Sammlung an Konfigurationseinträgen führen, über welche der Benutzer die Übersicht behalten muss. Aus diesem Grund muss festgelegt werden, wie die Konfiguration abläuft und wie der Benutzer Einträge in dieser definiert.

Ablauf

Die Konfiguration der Agenten befindet sich zu jeder Zeit im Publish/Subscribe-Verzeichnis. Zur Erstellung dieser Konfiguration sind mehrere Verfahren möglich, bspw. über die Verwendung einer GUI. Zwecks Komplexitätsminderung wird in dieser Arbeit ein Script verwendet, welches ein strukturiertes Dokument mit entsprechenden Informationen in Publish/Subscribe-Verzeichnis überträgt. Diese Lösung ermöglicht die Versionierung der Konfiguration mit einem Versionskontrollsystem und bietet sich für Systeme ohne grafische Oberfläche an.

Der Nachteil dieser Lösungen zeigt sich beim Anwenden einer Konfigurationsänderung. Werden mehrere Optionen eines Agenten verändert, so sollten diese nicht einfach sequenziell im System veröffentlicht werden, da so unnötige Übergangszustände für den Agenten auftreten könnten. Daher wird für den Konfigurator ein defensiver Ansatz gewählt: Zum Start wird das komplette Konfigurationsverzeichnis bereinigt. Dazu wird zunächst ein Wildcard-Abonnement auf das Verzeichnis ausgeführt und mehrere Sekunden gewartet, um auch bei eingeschränkter Netzwerk- und Systemleistung alle Einträge zu empfangen. MQTT verfügt über keine Funktion zum rekursiven Löschen von Themen oder einer Auflistung von zurückgehaltenen Themen, weswegen dieser Extraschritt erforderlich ist. Anschließend wird auf jedes Thema, für das eine Nachricht empfangen wurde und nicht in der Konfiguration vorhanden ist, eine leere Nachricht versendet, um den letzten zurückgehaltenen Wert zu löschen. Daraufhin wird erneut gewartet, damit alle Agenten Zeit haben die Änderung zu empfangen, sich zu deaktivieren und hinter sich aufzuräumen. Nun wird die Veröffentlichung der eigentlichen Werte durchgeführt. Neben den Werten werden für jedes Thema zudem Metadaten veröffentlicht, als der Dokumentations- und Format-String.

Aufbau

Für die Formatierung der Konfigurationsdateien wird YAML verwendet, da diese Beschreibungssprache Referenzen unterstützt, wodurch Redundanzen vermieden werden können. Der Konfigurator erlaubt die Aufteilung der Gesamtkonfiguration auf mehrere Dateien, um Übersichtlichkeit zu wahren. Innerhalb der Datei wird der Schlüssel CFG erwartet, unterhalb welchem alles als Konfiguration interpretiert und veröffentlicht wird. Der Schlüssel *_value* beinhaltet für das darüber liegende Thema die Information für das Veröffentlichen in Form einer Liste, welche sich aus dem Konfigurationseintrag, sowie das Serialisierungsformat und der Dokumentation des Eintrags zusammensetzt. Andere Schlüssel stehen zur freien Verfügung.

Ein Beispiel für die Überwachung eines Magnetkontakts für ein Küchenfenster ist in Listing 3.1 zu sehen: Unter *topics* ist das Thema konfiguriert, unter welchem der Zustand des Fensters veröffentlicht wird. Unter CFG ist die Shell *gpio@kitchen.example.com* konfiguriert. Diese Shell bündelt sämtliche Agenten, die Zugriff auf das GPIO Subsystem nehmen. Der Gruppierung steht nichts im Wege, solange die I/O Schnittstelle nicht für hochfrequente Operationen verwendet wird. Innerhalb der Shell ist ein einzelner Agent definiert, der den Namen *window* erhält. Unter dem Pfad der Shell wird im Unterthema mit dem Namen des Agenten der Pfad zur Fabrikmethode des Agenten veröffentlicht. Es folgen drei Optionen: Identifikator der GPIO-Pins, ob der Eingangswert logisch invertiert werden soll und ob der integrierte Pullup- oder Pulldown-Widerstand aktiviert werden soll. Als Letztes folgt die Verbindung des Fensterthemas mit dem Ausgang des Agenten. [78]

```
topics:
  - &open:
    topic: domain/kitchen/window
    qos: 0
    retain: True
    fmt: "struct/?"
    desc: If kitchen window is open

cfg:
  gpio@kitchen.example.com:
    window:
      _value: ["mauzr.agents.gpio:Input", str, Agent path]
      identifier:
        _value: [18, struct/B, Identifier of the GPI]
      invert:
        _value: [True, "struct/?", If value shall be inverted]
      pull:
        _value: [up, str, Pullup setting. May be one of up none down]
      input:
        _value: [*open, topic, Publish topic]
```

Listing 3.1: Beispielkonfiguration für einen Agenten

Validierung

Wie in vorhergegangen Kapiteln beschrieben, greifen drei Interessentengruppen auf ein einzelnes Thema zu, siehe dazu auch Abbildung 3.3. Der Administrator legt über das Veröffentlichen der Konfiguration zu einem Thema fest, wie dieses genutzt werden soll, weswegen die Formatbeschreibung in der Konfiguration explizit definiert werden muss. Dem entgegen stehen die Agenten, die konfiguriert werden, eine bestimmte Information auf diesem Thema zu veröffentlichen und dabei der Formatbeschreibung zu entsprechen. Signalisiert ein Agent der Abstraktionsschicht, dass er auf einem Thema Nachrichten veröffentlichen möchte, so übergibt er dabei einen regulären Ausdruck. Die Abstraktionsschicht abonniert das entsprechende Thema, um den vom Konfigurator festgelegten Formatstring zu erhalten, welcher mit dem regulären Ausdruck des Agenten verglichen wird. Kann der Ausdruck das Format abbilden, so ist davon auszugehen, dass der Agent in der Lage ist, konform dazu Nachrichten zu serialisieren. In diesem Fall wird der Ausdruck zusätzlich im Verzeichnis veröffentlicht, um im Fehlerfall mehr Ansatzpunkte zur Fehlerfindung zu bieten. Für die dritte Gruppe, den Abonnenten eines Themas wird ähnlich verfahren. Abonniert ein Agent ein Thema, muss ebenso ein regulärer Ausdruck angegeben werden, welcher mit dem Format des Administrators verglichen und veröffentlicht wird, falls kein Fehler auftritt. Um die Informationen der drei Gruppen zu trennen, werden diese wie in Abbildung 3.3 dargestellt auf die Unterthemen CONFIGURED, SUPPLIED und REQUESTED des eigentlichen Themas verteilt.

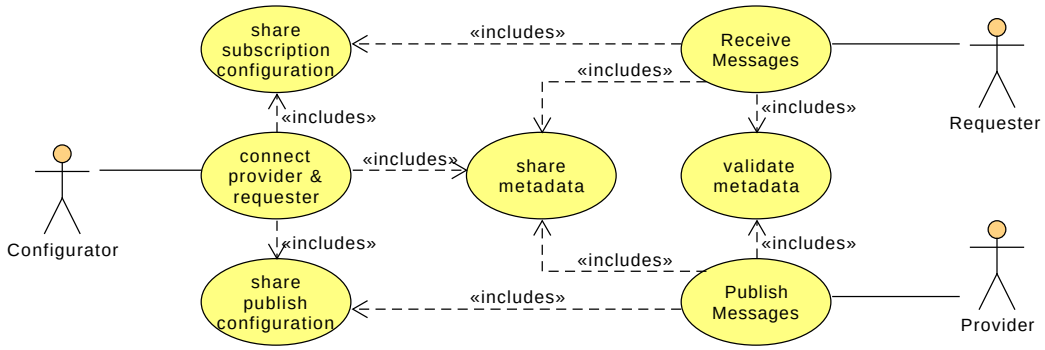


Abbildung 3.3: Anwendungsfalldiagramm für Interessenten eines Kommunikationskanals

3.3.2 Programmiersprachen

Wie vorhergehend angemerkt, muss die Abstraktionsschicht selbst in mindestens einer Programmiersprache verfasst werden. Aus diesem Grund muss ermittelt werden, welche Umsetzungsweise für die Abstraktionsschicht am geeignetsten ist und in Folge, welche Sprache am ehesten infrage kommt.

Umsetzungsweise

Bevor eine Auswahl vorgenommen werden kann, muss entschieden werden, in welcher Form die Abstraktionsschicht umgesetzt werden soll. In Hinblick auf die Entwicklung von Agenten bestehen folgende Möglichkeiten zur Umsetzung:

Unterstützung mehrerer Sprachen Zwar besteht die Möglichkeit, dieselbe Funktionalität in Abstraktionsschichten verschiedener Sprachen anzubieten, jedoch würde dies dazu führen, dass auch die Agenten für alle unterstützten Sprachen portiert werden müssten. Während der Aufwand für die Portierung der Abstraktionsschicht akzeptabel wäre, wirft die Portierung der Agenten mehr Probleme auf. So ist davon auszugehen, dass Agenten von Entwicklern erstellt werden, die nicht an der Abstraktionsschicht arbeiten und ihren Agenten entsprechend der vorhandenen Bedingungen für eine konkrete Plattform entwickeln. Anwender anderer Programmiersprachen würden so nur konzeptionell von solchen Implementierungen profitieren. Letztlich sinkt somit die Wiederverwendbarkeit von Code und spaltet die Nutzergemeinde in Untergruppen.

Domänenspezifische Sprache Es könnte eine DSL eingesetzt werden, mit welcher das Verhalten von Agenten definiert anstatt programmiert wird. Dazu könnten bestehende Lösungen verwendet werden oder eine Eigenlösung konzipiert werden. Ein Vorteil einer DSL wäre, dass die Implementierung des Agenten selbst nicht portiert werden muss. Jedoch weist dieser Ansatz auch zwei Nachteile auf: Zum einen muss der Interpreter u. U. auf neue Plattformen portiert werden, was potenziell einen hohen Aufwand darstellt. Hinzukommend muss dieser Interpreter mit stark eingeschränkten Ressourcen umgehen können. Zum anderen müsste es möglich sein, externe Bibliotheken einzubinden, wie es z. B. für die Analyse von Bilddaten oder das Anbieten einer Weboberfläche nötig wäre.

Wenn bspw. ein laufender Bildstrom von einer Überwachungskamera auf Bewegung analysiert werden soll, müssen verschiedene Bildfilter angewendet werden, deren Auswahl je nach konkretem Anwendungsfall unterschiedlich sein kann. Folglich wäre es nötig, nicht nur Teile, sondern die vollständige von den Agenten benötigte Funktionalität zu abstrahieren. Da dies nicht umsetzbar ist, müsste in der DSL die Möglichkeit gegeben sein, auf Elemente der darunter liegenden Sprache zurückzufallen. Dies würde nicht nur zu einem gesteigerten Overhead führen, sondern wegen der Mischform auch die Lesbarkeit verringern, was die Wartbarkeit reduziert. [77]

Umsetzung in einer Sprache mit Kompatibilität Die direkteste Möglichkeit ist Umsetzung der Abstraktionsschicht in einer Programmiersprache, die gleichzeitig möglichst universell, aber auch effektiv zu nutzen ist. Dadurch kann ein Großteil der Agenten mit dieser Schicht entwickelt werden. Es wird Grenzfälle geben, welche eine andere Umgebung erfordern. Für diese ist eine hohe Kompatibilität mit integrierten Agenten zu gewährleisten. Es könnten z. B. Programmierkits für andere häufig genutzte Plattformen angeboten werden, die eine Basisfunktionalität bereitstellen. Eine andere Möglichkeit ist, die Anwendungsschnittstellen so zu gestalten, dass eine einfache Integration von externen, unabhängigen Komponenten möglich ist. Diese Komponenten arbeiten ohne oder mit einer anderen Abstraktionsschicht, was zu einer deutlich höheren Flexibilität bei deren Implementierung führt, da somit passgenau auf die Anforderungen des Projekts eingegangen werden kann. [77]

Auswahlkriterien

Zur Auswahl einer geeigneten Sprache ist ein Vergleich erforderlich. Für das Projekt soll eine zugängliche Sprache verwendet werden. Um dies zu gewährleisten, soll die gesuchte Sprache populär sein, was sich durch ihre Präsenz in den oberen Bereichen von Sprachrankings bemerkbar macht. Um innerhalb dieser den besten Kandidaten zu finden, werden Vergleichsfaktoren herangezogen.

System Calls Agenten sollen die Möglichkeit haben, direkt die Funktionen des Betriebssystems zu nutzen. Da diese je nach Gerätefamilie unterschiedlich sind, widerspricht diese Anforderung zwar dem Abstraktionsgedanken, ist jedoch zur Anbindung von Hardware über Peripherie nicht zu vermeiden. Dieser Bruch kann dadurch minimiert werden, dass mit der Abstraktionsschicht Funktionsgruppen ausgeliefert werden, die dynamisch

anhand der Ausführungsplattform den Zugriff für andere Agenten abstrahieren und dadurch keinen direkten Zugriff mehr benötigen. Je nach System kann der Zugriff auf das Betriebssystem auf unterschiedliche Weisen stattfinden, üblich ist jedoch die Verwendung von sog. System Calls. Diese sind in der Regel Funktionsaufrufe, die in der Sprache des Betriebssystems, üblicherweise C, ausgeführt werden. Programmiersprachen, die C Funktionen nicht direkt aufrufen können, lösen dies in der Regel durch die Implementierung von Wrappermethoden, die Aufrufe der Sprache in C Aufrufe umwandeln. Positiv bewertet wird die Möglichkeit, Systemcalls nativ aufzurufen, neutral, wenn alle oder ein Großteil der Systemcalls über Wrapper erreichbar sind, negativ, wenn keine der beiden Möglichkeiten existiert.

Portabilität Weiterhin relevant ist die Portabilität des Quellcodes. Negativ bewertet wird, wenn eine Programmiersprache nur auf einer einzelnen Plattform verfügbar ist. Neutral bewertet werden Sprachen, die auf verschiedenen Plattformen verfügbar sind, jedoch für die entsprechende Plattform individuell kompiliert werden müssen. Positiv zu bewerten sind Sprachen, die ohne Anpassung allgemein auf unterstützten Plattformen lauffähig sind.

Dynamische Ausführung Die Agenten sollen ihre Konfiguration übers Netzwerk erhalten. Während diese zum Großteil aus primitiven Daten bestehen wird, stechen zwei Besonderheiten heraus. Kritisch ist die Möglichkeit, Agenten und deren Ausführungscode dynamisch zu laden, was erfordert, dass die Ausführungsumgebung Bibliotheken zur Laufzeit einbinden kann. Kann sie dies nicht, ist sie ungeeignet und das Kriterium wird negativ bewertet. Ein weiterer Punkt ist, dass nicht jede Einstellung als Datum effektiv repräsentiert werden kann. Wenn bspw. die Heizung einer Wohnung unter Beachtung der Außentemperatur anhand einer mathematischen Funktion angepasst werden soll, bietet sich die Beschreibung dieser Logik als Lambdamethode als direktester Weg an. Wenn die zu untersuchende Sprache das Kompilieren oder Interpretieren von Quellcode zur Laufzeit unterstützt, ist dies als positiv anzusehen.

Generalität Die Abstraktionsschicht soll die Entwicklung von Agenten vereinfachen und nicht einschränken. Aus diesem Grund soll es einfach möglich sein, auf Funktionen und Bibliotheken zuzugreifen, die von der Abstraktionsschicht nicht implementiert wurden. Dies lässt sich mit der Verwendung einer sog. General Purpose Language erreichen, also einer Sprache, die nicht auf eine spezielle Problemstellung spezialisiert ist, sondern das Ziel hat, möglichst viele allgemeine Anwendungsfälle bedienen zu können. Positiv bewertet werden diese General Purpose Languages, negativ das Gegenteil, domänenspezifische Sprache, also Sprachen, die nur auf eine konkrete Problemgruppe zugeschnitten sind.

Lizenz Weiterhin entscheidend ist die Lizenz der Sprache. Die Sprache selbst und benötigte Komponenten wie Kernbibliotheken und Compiler/Interpreter sollten für alle Interessenten frei unter einer Open Source Lizenz verfügbar sein. Dies ist als positiv zu bewerten. Bestehen Einschränkungen für kommerzielle Nutzer, so ist dieser Punkt als neutral abzuwerten. Vollständig eingeschränkte Nutzungsrechte führen zu einer negativen Bewertung. Dies ist wichtig, da die Abstraktionsschicht veröffentlicht werden soll, damit Entwickler aus aller Welt an dem Projekt mitarbeiten können. Eine eingeschränkte Lizenz würde zu einem hohen Verlust von potenziellen Entwicklern führen, da dies Mitarbeitende zusätzlich mit Einstiegskosten für Lizenzen von Drittanbietern belasten würde. Zudem bedeutet proprietäre Software, dass Modifikationen nur schwer umsetzbar sind und nicht weiterverbreitet werden können.

Vergleich

Wie anfangs beschrieben, sollen populäre Sprachen verwendet werden, um eine breite Nutzerbasis erreichen zu können. Um jene Sprachen zu ermitteln, wird der Tiobe Sprachindex, ein Ranking der beliebtesten Programmiersprachen vom 5. November 2018, herangezogen. Aus diesem Ranking wurde die Top 15 auf die aufgestellten Kriterien überprüft. Die Erfüllung eines Kriteriums konnte positiv, neutral oder negativ ausfallen, was zu einer numerischen Wertung von -1, 0 und +1 führt. Für jede Sprache werden die Bewertungen aller Kriterien summiert als Gesamtwertung erfasst. Für die Erfassung dieser Kriterien bietet sich die Tabellenform an. In Tabelle 3.1 repräsentiert jede Zeile eine begutachtete Sprache, die Spalten beinhalten Namen der Sprachen, die getesteten Kriterien, Wertungssumme und Platz im Ranking. Die Zeilen wurden erst nach Wertungspunkten, dann nach Rankingplatzierung sortiert. [2]

Zwecks Übersichtlichkeit wurden die Sprachen C, C++ und Objective C zusammengefasst, da diese dieselbe Wertung aufweisen und zur selben Familie gehören. Als Platz im Ranking wurde die höchste Platzierung im Tiobe Index (C mit Platz 2.) verwendet. Der Tiobe Index führt auf Platz 13. Assembly an, welche generell für die Verwendung von Assemblersprache steht und nicht für eine konkrete Prozessorimplementierung.

Ergebnis

Die Vergleichstabelle 3.1 zeigt auf, dass sich Python und JavaScript mit einer Wertungssumme von vier Punkten am geeignetsten erweisen. Beide Sprachen erfreuen sich großer Beliebtheit und ermöglichen die rapide Entwicklung von hardwarenahen Projekten. Neben der Möglichkeit, auf Low Level APIs des Betriebssystems zuzugreifen, sind beides Interpretersprachen, die dynamische Codeausführung unterstützen und als generelle Programmiersprachen über eine große Anzahl von Bibliotheken verfügen, die viele Möglichkeiten zum Entwickeln von Agenten bieten.

Sprache	Syscalls	Port.	Dynamik	Generalität	Lizenz	$\sum$	Rank
Python	o	+	+	+	+	4	4
JavaScript	o	+	+	+	+	4	8
C/C++/Obj C	+	o	o	+	+	3	2
C#	+	o	o	+	+	3	6
PHP	-	+	+	+	+	3	7
Swift	+	o	o	+	+	3	10
Go	o	o	+	+	+	3	12
Assembly	+	-	o	+	+	2	13
R	+	+	o	-	+	2	14
Java	-	+	o	+	o	1	1
Visual Basic	+	o	o	+	-	1	5
SQL	-	o	+	-	+	0	9
Matlab	-	+	-	-	-	-3	11

Tabelle 3.1: Vergleich der möglichen Programmiersprachen

Beide Sprachen bieten sich also für die Umsetzung der Abstraktionsschicht an. Python befindet sich im Ranking auf Platz 4., JavaScript auf Platz 8., weswegen die Entscheidung auf Python fällt. Python verfügt über eine große Gemeinschaft und wird auf vielen Embeddedplattformen, wie z. B. dem Raspberry Pi, gut unterstützt. Projekte wie Micropython ermöglichen es zudem, Python-Code auf Microchips wie dem ESP32 mit Echtzeitfähigkeiten auszuführen. [36][40]

3.3.3 Plattformen

Nach der Festlegung der Programmiersprache ist es möglich, die Zielplattform einzugrenzen. Python unterstützt offiziell Unixderivate, MacOS und Windows, es existiert jedoch eine Vielzahl von inoffiziellen Anpassungen für Embeddedsysteme, auf welchen jene Betriebssysteme nicht laufen. Somit kann Python auch z.B unter Android ausgeführt werden oder mittels *micropython* auf Mikrocontrollern wie dem ESP32. Die inoffiziellen Anpassungen weichen u. U. von dem offiziellen vollen Sprachstandard, den Standardbibliotheken oder Interna wie z. B. der Umsetzung der Garbage Collection ab. So verfügt das *micropython* Derivat über keine Synchronisationsmechanismen wie Mutexe und bietet keine Optionen für schwache Objektreferenzen. [36][40]

Die Abstraktionsschicht könnte zu solchen Plattformen kompatibel sein, indem nur ein eingeschränkter Umfang der Features der Sprache verwendet wird. Dies würde jedoch zulasten des Entwicklungskomforts gehen und Agentenentwickler u. U. stark einschränken, weswegen keine Rücksicht auf entsprechende Einschränkungen durch jene genommen wird.

3.3.4 Komponenten

Die Abstraktionsschicht ist aus mehreren Komponenten zusammengesetzt, auf welche nun eingegangen wird. Das vereinfachte Klassendiagramm in Abbildung 3.4 zeigt die einzelnen Komponenten und deren Relationen auf.

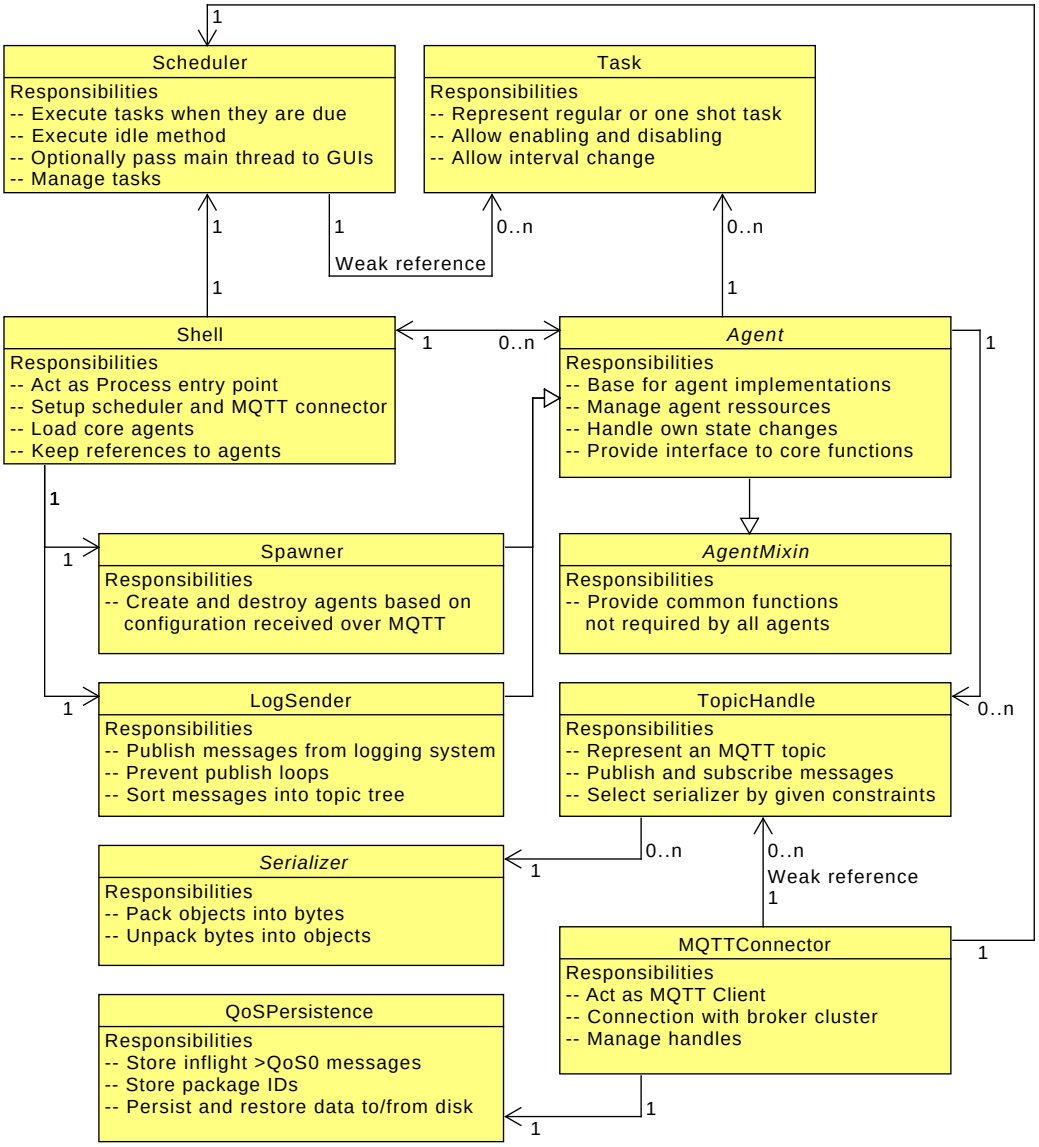


Abbildung 3.4: Vereinfachtes Klassendiagramm der Abstraktionsschicht

Shell

Eine Instanz der Abstraktionsschicht wird Shell genannt. Die gleichnamige Klasse hat die Aufgabe, bei Start des Programms mithilfe der Programmargumente die benötigten Komponenten zu starten und diese sowie die erstellten Agenten während der Ausführung zu verwalten. Wird das Programm mittels Signal zur Beendigung aufgefordert, werden alle verwalteten Komponenten deinitialisiert.

Wird die Instanz als Dienst gestartet, kann die Instanz direkt mit der Diensteverwaltung kommunizieren, um den Zustand des Dienstes ohne Umweg über die Standardausgabe zu kommunizieren. Dazu überprüft die Shell, ob ein entsprechender Kommunikationskanal über Prozessmanager wie z. B. *systemd* (ein häufig genutzter Prozessmanager unter Linux) festgelegte Umgebungsvariable definiert wurde. Ist dies der Fall, wird der Diensteverwaltung in regelmäßigen Abständen der Zustand der Instanz mitgeteilt. Bleibt diese Aktualisierung aus, geht die Verwaltung von einem Versagen des Dienstes aus und startet die Fehlerbehandlung.

Tasker

Agenten haben eine einzelne primäre Aufgabe, die sich in Teilschritte zerlegen lässt. Diese müssen u. U. eine bestimmte zeitliche Distanz zueinander aufweisen. So muss bei der Steuerung eines Sensors zwischen dem Start einer Messung und dem Anfordern des Resultates eine Mindestzeit vergehen, damit der Messwert Gültigkeit hat.

Agenten sollten dabei Zugriff auf drei Semantiken haben: Zwischen der Anforderung und der Ausführung einer Aufgabe soll eine bestimmte Zeit vergehen (*after*), nach Anforderung soll die Aufgabe mit einem festgelegten Abstand zueinander regelmäßig ausgeführt werden (*every*), und die Ausführung einer Aufgabe zu einem bestimmten Zeitpunkt (*at*).

Grundsätzlich sollten Tasks aufgrund der Besonderheit von Python von einem einzelnen Thread verwaltet werden, um den Overhead durch Threadwechsel zu minimieren. Dies erfordert, dass Aufgaben von Seite der Agenten aus als nicht blockierend entworfen werden müssen. Obgleich dies das geschickte Behandeln von I/O Operationen erfordert, ermöglichen nicht blockierende und nicht preemptive Aufgaben den großteiligen Verzicht auf Synchronisierungsmechanismen. Während dies für den größten Teil der Aufgaben gut realisierbar ist, sind für Sonderfälle Ausnahmen möglich. Es soll unterstützt werden, den

Hauptthread des Programmes exklusiv für GUI bereitzustellen, da viele Implementierungen dies erfordern. Dazu wird beim Initialisieren dieser Komponente ein neuer Thread gestartet, der die Verwaltung der Aufgaben übernimmt. Der Hauptthread wird anschließend an die GUI weitergereicht. Weiterhin ist es für die Anbindung an den Message Broker relevant, eine Methode aufzurufen, sobald keine Aufgaben akut zur Ausführung anstehen. Für eine hohe Reaktionsfähigkeit ist es relevant, dass Nachrichten so schnell wie möglich empfangen werden, weswegen hier ein Intervallaufruf nicht angemessen ist. Der aufzurufenden Methode wird mitgeteilt, für wie lange sie den Thread exklusiv in Anspruch nehmen kann.

Grundsätzlich sind Aufgaben langlebig und müssen anpassbar sein, um sich ändernden Situationen entsprechen zu können. Wenn eine Aufgabe beim Erstellen eines Agenten registriert wird, sollte sie noch nicht aktiviert sein, da der Agent bspw. noch Konfigurationsoptionen erhalten muss oder z. B. im Fall einer Aufgabe, die regelmäßig einen Messwert abfragen soll, wenn der Sensor erst initialisiert werden muss. Die Aktivierung und Deaktivierung ist von anderen Aufgaben des Agenten durchzuführen. Dies ist in Abbildung 3.5 veranschaulicht. Auch kann die Zeitangabe einer Aufgabe durch eine Konfigurationsoption angegeben werden, welche sich während des Betriebs ändern kann, was eine dynamische Änderung erfordert.

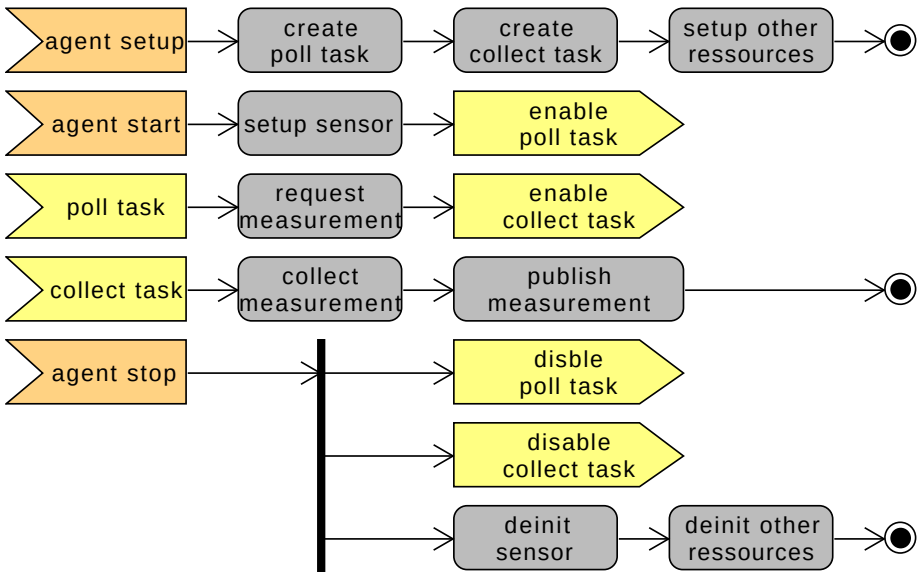


Abbildung 3.5: Die Verwendung von Aufgaben im Lebenszyklus eines Agenten

Aufgabenmanagement Um zu vermeiden, dass Entwickler unbeabsichtigt Aufgaben nicht deinitialisieren, wird eine Aufgabe automatisch deaktiviert und gelöscht, sobald sie von keinem Agenten mehr referenziert wird. Dies wird umgesetzt, indem der SCHEDULER schwache Referenzen auf die Aufgaben hält, wodurch der Garbage Collector in diesem automatisch nicht mehr benötigte Aufgaben beseitigt.

Serialisierer

Das Serialisierungsmodul des Projekts baut sich um die Formatbeschreibung auf. Der Aufbau der Beschreibung orientiert sich an Multipurpose Internet Mail Extensions (MIME-Types) und hat die Form DATACLASS/SPECIFICATION. Wenn ein generischer Datentyp gehandhabt wird, sollten Agenten nicht mit den konkreten Implementierungen der Serialisierer hantieren, sondern ausschließlich mit deren Beschreibung. Die Abstraktionsschicht übernimmt das Nachschlagen und Bereitstellen dieser.

Zur Auslieferung der Abstraktionsschicht gehören mehrere Serialisierer, die bis auf zwei Ausnahmen keinen Spezifikationsteil in der Formatbeschreibung benötigen. Dabei handelt es sich um Serialisierer für Datentypen wie UTF8-Strings und JSON, die keine Zusatzinformationen zum Betrieb benötigen.

Ein Serialisierer, der im Umfeld der Simulationsumgebung besonders häufig verwendetet wird, ist STRUCT, welcher das gleichnamige Pythonmodul verwendet, um eine Liste umzuwandeln, die aus einem oder mehreren primitiven Daten besteht. Um bspw. einen Temperaturmesswert als Gleitkommazahl mit einem Zeitstempel zu serialisieren, wird der Formatstring STRUCT/!FQ angewendet. Dies weist STRUCT an, in den Datenstrom mit Big-Endian Bytereihenfolge, repräsentiert als Ausrufezeichen, zuerst eine 32-bit Gleitkommazahl und einen vorzeichenlosen 64-bit Integer zu verpacken.

Wenn ein Agent einen, speziell auf einen Anwendungszweck zugeschnittenen, Serialisierer nutzen möchte, so kann dessen Auflösung überbrückt werden. Ein spezieller Serialisierer innerhalb der Abstraktionsschicht ist dafür zuständig, Informationen zu einem Thema zu serialisieren, also das Thema selbst sowie Quality of Service (QoS) und Rückhalteeinstellungen. Für die weitere Verwendung ist es erforderlich, dieses Thema beim MQTT Clienten zu registrieren, um es entweder zu abonnieren oder darauf zu veröffentlichen. Da dies während des Serialisierungsprozesses durchgeführt wird und Fehler vorzeitig erkannt werden können, entschlackt dies die Handhabung von Themen im weiteren Verlauf.

Messaging

Wie im Protokollvergleich aufgezeigt, existieren Clientimplementierungen für MQTT in vielen Sprachen. Unter Python ist die Bibliothek *Paho* von der *Eclipse Foundation* eine häufig genutzte Umsetzung. Leider verfügt diese Implementerung über mehrere Nachteile. So unterstützt die Bibliothek zwar die Weitergabe von Anforderungen an die Qualitätsgüte, kann diese aber selbst nicht garantieren. Fällt das Programm zwischen der Übergabe einer Nachricht und dem asynchron folgendem Senden dieser zum Message Broker aus, so ist die Nachricht verloren. Auch wird die Anbindung an hoch verfügbare Message Broker nicht unterstützt, was die Ausfallsicherheit stark reduziert. Es ist unklar, wann die Bibliothek die neue Version, MQTT 5, unterstützen wird.

Implementierung Da andere Bibliotheken ähnliche Schwächen aufweisen und bei der Protokollauswahl wert auf ein kompaktes Protokoll gelegt wurde, ist die Umsetzung einer eigenen Anbindung effektiver. Dadurch kann diese an nötigen Stellen enger mit der Abstraktionsschicht gekoppelt werden, um eine einheitliche Schnittstelle für Agentenentwickler zu bieten.

Handles Für gewöhnlich werden Themen mittels eines Strings adressiert, der keine Zusatzinformationen enthält. Zusatzinformationen wie die QoS Garantie müssen jeder Methode übergeben werden, die Zugriff auf diese benötigt. Abonnements werden durch einen Methodenaufruf deklariert und müssen ebenso wieder abgebrochen werden. Für dieses Projekt wurde sich dafür entschieden, Themen und Abonnements als Ressourcen zu behandeln. Themen werden durch sog. Handle repräsentiert, die entweder durch die Angabe des absoluten Themas oder durch einen Pfad relativ zu einem anderen Handle mit den nötigen Zusatzinformationen erstellt werden können. Das Handle verwaltet sämtliche Funktionen und Ressourcen, die mit dem Thema interagieren können, so können Daten mit diesem Thema veröffentlicht und Abonnements erstellt werden. Im Gegensatz zu den meisten Implementierungen ist ein Abonnement die Kombination des Themas mit dem angegebenen Callback. Wird ein Abonnement von einem Agenten angefordert, gibt das Handle mit dem Aufruf ein Objekt zurück, welches das Abonnement repräsentiert. Ist kein Agent mehr im Besitz des Objekts, wird es durch das Handle verworfen. Entsprechend der Existenz dieser Objekte propagiert das Handle Abonnementstatus an den Message Broker weiter. Instanzen der Handle werden zudem vorgehalten, um kurzlebiges aber stetiges Interesse an Themen handhaben zu können.

High Availability Für einen reibungslosen Betrieb ist es kritisch, die Zeit, die ein MQTT Client ohne Verbindung zum Message Broker verbringt, möglichst kurz zu halten. Wird, wie üblich, nur ein einzelner Server genutzt, besteht keine Ausweichlösung für den Clienten, wenn dieser ausfällt, weswegen die erwähnten MQTT Broker mit Clusterfunktion verwendet werden können. Diese erfordern, dass der Client mit dieser Situation umgehen kann und bei dem Zusammenbruch der Verbindung zu einem Server einen weiteren probiert. Um die Konfiguration davon effektiv zu gestalten, soll auf eine dafür übliche Technologie zurückgegriffen werden. Im Standard von Domain Name System (DNS) ist der Eintrag *SRV* definiert. Dieser Eintrag ermöglicht, innerhalb einer Domäne für einen Dienst eine Liste zuständiger Server und deren Verbindungsparameter anzugeben. Für jeden Eintrag kann zudem eine Priorität und Gewichtung festgelegt werden, wodurch sowohl Hochverfügbarkeit als auch Lastverteilung umgesetzt werden können. Für den Sonderfall, dass beim Ausfall des primären Message Brokers ein Reservedienst verwendet wird, der sich nicht mir dem Primärdienst in einem Cluster befindet und somit ein unabhängiges Publish/Subscribe-Verzeichnis aufweist, wurden die HANDLE angepasst. Wird für ein Thema eine Nachricht versendet, die vom Message Broker vorgehalten werden soll, hält das HANDLE diese Nachricht selber vor und veröffentlicht sie erneut, sobald eine Verbindung zu einem neuen Message Broker aufgebaut wird. Dieses Vorgehen macht sich zunutze, dass vorgehaltene Nachrichten einen Zustand repräsentieren, sie also im Gegensatz zu nicht vorgehaltenen Nachrichten bis auf Weiteres gültig sind und nicht eine zeitlich lokale Information repräsentieren.

Quality of Service Die Implementierung soll die Garantien für die Vermittlung von Nachrichten ebenso wie der Message Broker innerhalb von technisch möglichen Maßstäben garantieren können. Dies bedeutet, dass ein Agent von der Einhaltung der Garantie ausgehen kann, sobald der Methodenaufruf zum Veröffentlichen der Daten vom HANDLE zurückkehrt. Dazu ist es zuerst nötig, festzustellen, <an welchen Stellen Nachrichten verloren gehen können. In Abbildung 3.6 sind diese in gelb markiert. Es können die Abstraktionsschichten des Senders und Empfängers ausfallen, der Verbindungen zum Message Broker verloren gehen oder dieser selbst ausfallen. Damit die Abstraktionsschicht des Senders dies sicherstellen kann, muss der Aufruf des Agenten zum Senden der Nachricht vollständig ausgeführt werden. Auch kann der Ausfall des empfangenden Agenten behandelt werden. Um dagegen vorgehen zu können, ist es nötig, dass jede dieser Etappen Nachrichten mit entsprechender Garantie persistiert, bis sie von der folgenden Etappe einen erfolgreichen Empfang signalisiert bekommt. Hier ist zu beachten, dass sowohl Haupt- als auch Persitierungsspeicher überlaufen können, wenn z. B. über einen längeren Zeitraum keine Verbindung zu einem Message Broker aufgebaut werden kann. Um dies zu kompensieren, könnte eine künstliche Beschränkung wie z. B. eine Begrenzung der Anzahl gepufferter Nachrichten eingeführt werden. Dies würde jedoch dazu führen, dass die feste QoS Garantie aufgegeben werden müsste, weswegen diese nicht eingesetzt wird.

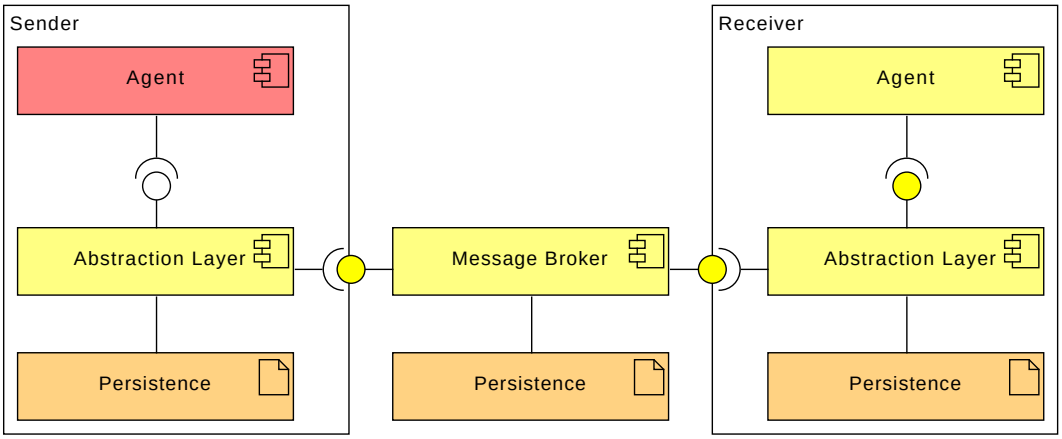


Abbildung 3.6: Gegen Ausfall abgesicherte Abschnitte der Nachrichtenvermittlung

3.3.5 Mitgelieferte Agenten

Mit der Abstraktionsschicht werden Agenten ausgeliefert. Bei diesen Agenten handelt es sich zum einen um Komponenten, die für die korrekte Funktion einer Programminstanz erforderlich sind. Zum anderen bestehen im Multiagentensystem häufig vorkommende Aufgaben, deren Implementierung zur Vermeidung von Coderedundanzen von der Distribution übernommen werden sollte.

Spawner

Damit Agenten auf der Zielinstanz der Abstraktionsschicht eingesetzt werden können, müssen sie zur Laufzeit geladen werden. Dies wird durch den SPAWNER vorgenommen, der alle Konfigurationsthemen direkt unterhalb des Namens der Instanz abonniert, also CFG/<INSTANCE>/+. Das letzte Element des Themas wird als Name des Agenten verwendet, sodass die Konfiguration des zukünftigen Agenten unter CFG/<INSTANCE>/<AGENT> zu finden ist. Als Nutzlast des Themas wird ein String im Format <MODULE>:<METHOD> erwartet, aus welchem eine Fabrikmethode oder ein Konstruktor abgeleitet werden kann, mit welchen der Agent instantiiert wird.

Der SPAWNER versucht mit diesen Informationen, den Agenten zu erstellen und fügt ihn bei Erfolg der SHELL hinzu. Soll ein Agent entfernt werden, so ist die zurückgehaltene Nachricht dieses Themas zu löschen. Dies wird bei MQTT durch das Veröffentlichen einer leeren Nachricht in diesem Thema durchgeführt. Der SPAWNER greift bei Erhalt dieser Nachricht auf die SHELL zu und weist den entsprechenden Agenten an, sich zu deaktivieren und zu zerstören. Ist die Erstellung eines Agenten nicht möglich, weil z. B. die Fabrikmethode nicht abrufbar ist oder diese fehlerhaft ist, so wird der Fehler protokolliert und der Agent bis auf Weiteres ignoriert.

Logger

Ein weiterer Agent übernimmt das Veröffentlichen von Logausgaben. Dazu wird das Modul LOGGING der Standardbibliothek von Python erweitert, dass auch die Ausgaben von Drittbibliotheken ohne Änderung dieser erfasst werden können. Die Shell definiert dazu einen Stammlogger, welcher als Senke, neben der Standardausgabe, diesen Agenten erhält.

Die Basisklasse für Agenten erstellt für jeden Agenten einen Unterlogger auf Basis des Loggers der SHELL. Dadurch können Agenten individuelle Einstellungen an ihrer Loggerinstanz vornehmen, wie das Anpassen des Loglevels anhand von Konfigurationsoptionen über das Publish/Subscribe-Verzeichnis. Zudem lässt sich dadurch eindeutig feststellen, welche Ausgabe zu welchem Agenten gehört.

Sämtliche Ausgaben werden durch den Stammlogger an den Logagenten geleitet, welcher dieser im Thema LOG/<INSTANCE>/<AGENT> als flüchtige Nachricht veröffentlicht. Dabei sind Besonderheiten zu beachten, um Schleifen bei der Veröffentlichung zu vermeiden. So werden bspw. Logausgaben der Messagingkomponente nicht veröffentlicht, da im geringsten Loglevel eine Ausgabe erzeugt wird, sobald eine Nachricht veröffentlicht wird.

Neben diesem Sendeagenten, der in jeder Instanz automatisch erstellt wird, existiert das Gegenstück zu diesem, welcher empfangene Logausgaben in das lokale Loggingsystem einspeist, damit der Anwender diese konzentriert an seinem Arbeitsplatz begutachten und ggf. zentral filtern kann. Dadurch kann Debugging, Analyse und Tracing des Systems effizienter durchgeführt und einfach eine Übersicht über den Gesamtzustand des Systems erstellt werden.

Complex Event Processing Engine

Wie erwähnt eignet sich eine CEP-Engine, um einen Großteil der nötigen Verarbeitung von Daten umzusetzen. Da in vorhergegangen Arbeiten die Engine *Esper CEP* behandelt wurde, soll hier eine Anbindung an diese umgesetzt werden. *Esper CEP* läuft, wie auch ein Großteil der restlichen CEP-Engines auch, unter Java und ist somit über einen externen Agenten anzubinden. Dieser verwendet nicht die Abstraktionsschicht, sondern läuft als eigenständiger Prozess. Er erhält seine vollständige Konfiguration, inklusive Regelwerk, über das Publish/Subscribe-Verzeichnis und abonniert dynamisch Themen, die für die CEP-Engine relevant sind. Generierte Ereignisse werden wieder in das Publish/Subscribe-Verzeichnis eingespeist.

Datenaggregatoren

Sollten Entwickler den vollen Funktionsumfang einer CEP-Engine nicht benötigen, bietet sich ein leichtgewichtigerer Ansatz an. Es werden mehrere Agenten bereitgestellt, die jeweils eine einzelne generische Verarbeitungsmethode durchführen, um von Entwicklern zu komplexeren Verarbeitungsketten zusammengesetzt werden zu können, ohne dass diese dafür eigens Agenten schreiben müssen. Dazu werden ein oder mehrere Eingänge und ein Ausgang sowie spezifische Optionen konfiguriert.

Converter Dieser Agent hat einen Eingang, dessen Daten anhand einer Regel umgewandelt werden und über den Ausgang wieder veröffentlicht werden. Die Regel wird als Lambdamethode ausgedrückt, welche als String konfiguriert wird. Dies kann für eine Vielzahl von Anwendungen genutzt werden, wie z. B. das Anreichern einer Information mit Kontext oder der Umsetzung einer logischen Implikation.

Delayer Auch dieser Agent verfügt über je einen Ein- und Ausgang, hat jedoch die Aufgabe, eine empfangene Nachricht nach einer konfigurierten zeitlichen Verzögerung unverändert im Ausgang zu veröffentlichen. Trifft während der Wartezeit eine neue Nachricht ein, wird der Timer zurückgesetzt und die neue Nachricht nach Ablauf der Zeit veröffentlicht. Aktiviert der Benutzer bspw. die Alarmanlage einer Wohnung, so ist diese erst nach einer gewissen Zeit zu aktivieren, da der Benutzer die Wohnung zuerst verlassen muss.

Aggregator Der AGGREGATOR agiert ähnlich wie der CONVERTER, verfügt aber über mehrere Eingänge. Die Konvertierungsmethode erhält, neben dem sich aktuell änderndem Thema und dessen neuen Wert auch ein Verzeichnis mit den letzten Zuständen aller anderen Eingänge, wodurch die Aggregation aller Eingangsdaten zum Ausgang kompakt durchgeführt wird. So kann z. B. der Öffnungszustand von mehreren Fensterkontakten zu einem Wahrheitswert zusammengefasst werden, der aussagt, ob alle Fenster geschlossen sind.

Toggler Obwohl seine Komplexität alle anderen Agenten dieser Kategorie übersteigt, ist dieser Agent für manche Funktionen innerhalb von Smart Environments notwendig. Der Ausführungszustand vieler Aktoren in Smart Environments lässt sich durch einen Wahrheitswert ausdrücken. Auf diesen Zustand können verschiedene Komponenten Einfluss nehmen. Neben Informationen, die als Zustände ausgedrückt werden, also z. B. *der Benutzer ist in der Wohnung*, sind auch kurzlebige Ereignisse relevant wie *Der Taster für die Beleuchtungssteuerung wurde betätigt*. Diese repräsentieren zwei Informationsarten, nämlich Bedingungen, die einen oder mehrere mögliche Zustände des Aktors vorschreiben oder Anfragen, die einen bestimmten Zustand fordern, aber von den Bedingungen außer Kraft gesetzt werden sollen. Für die anfragenden Komponenten ist zudem relevant, welche Anfragen aktuell gestellt werden können, damit sie Benutzern entsprechende Informationen präsentierten können. Dies wird von Toggler gehandhabt.

3.4 Agent

Eine Zusammensetzung von CPSs zu einem Smart Environment lässt sich effektiver darstellen, wenn das resultierende System als Multiagentensystem interpretiert wird. Eine CPS-Komponente kann dabei, je nach Komplexität, einen oder mehrere Agenten beinhalten. So wäre eine netzwerkfähige Glühbirne als ein Agent zu modellieren. Dieser Agent hätte die Fähigkeit, den Zustand des Leuchtmittels in der Glühbirne zu verändern. Ein intelligentes Heizungsthermostat könnte aus drei Agenten bestehen: Ein Agent regelt den Wasserdurchfluss der Heizung, ein Weiterer erfasst die Temperatur der Umgebungsluft und ein Dritter verwendet die beiden anderen Agenten, um das Zimmer auf eine, zu einer bestimmten Uhrzeit geltenden, Solltemperatur zu temperieren. Werden diese Komponenten in ein Smart Environment zusammengeschlossen, so werden die beinhalteten Agenten mit anderen gekoppelt, um höhere Funktionalität zu gewährleisten. Der Agent der Glühbirne kann z. B. mit einem Lichtschalter gekoppelt werden und der Kontrollagent des Thermostats mit einem weiteren Thermometer im Raum, um genauere Informationen über den Raum zu erhalten und effizienter zu heizen.

3.4.1 Konfiguration

Agenten werden, wie vorher beschrieben, vollständig über das Publish/Subscribe-Verzeichnis konfiguriert. Dies wird dadurch realisiert, dass bei der Implementierung des Agenten sog. Optionen angegeben werden. Diese können jeweils einen der folgenden Verwendungszwecke haben:

Einstellung Für verschiedene Einstellungen innerhalb des Agenten sind Werte erforderlich. Dabei kann es sich um das Intervall handeln, in welchem ein Sensor abgefragt wird oder eine Liste von RGB-Farben, welche auf einer *SK6812* Kette anzuwenden sind. Im Normalfall ist das genaue Format bei dem Erstellen der Option bekannt und wird deswegen bei der Definition der Option angegeben. Wird eine Option auf dem Thema veröffentlicht, die nicht der Formatdefinition entspricht, wird dies protokolliert und die Nachricht verworfen. Ein Agent kann erst starten, wenn alle als erforderlich markierten Optionen eingegangen sind. Sollte eine Option während des Betriebs neu veröffentlicht werden, so wird der Agent neu gestartet, sofern dies bei der Definition nicht anders angegeben wurde. Optionen können automatisch als Attribut des Agenten gesetzt werden oder über ein angegebenes Callback verarbeitet werden.

Wahlfreier Nachrichteneingang Um die dynamische Zuordnung von Themen zu Agenten zu ermöglichen ist es nötig, dass ein Agent dieses Thema als Konfiguration erhält, was sich ähnlich wie die Definition einer Option verhält. Auch hier wird angegeben, welchen Namen der Eingang aufweist, wofür der Eingang verwendet wird und welches Callback bei Nachrichtenempfang aufzurufen ist. Zudem kann ein Agent erst starten, wenn alle benötigten Eingänge konfiguriert wurden. Da Agenten u. U. flexibler bei der Nachrichtenformatierung sind, z. B. kann der CONVERTER-Agent je nach Konvertierungsmethode völlig unterschiedliche Eingangsdaten aufnehmen, weswegen hier das unterstütze Format als Regulärer Ausdruck (Regex) angegeben wird, welches mit dem vom Konfigurator definierten Konvertierungsformat abgeglichen wird. Erfüllt das Format den regulären Ausdruck nicht, so wird die Konfiguration protokolliert und verworfen.

Wahlfreier Nachrichtenausgang Ebenso wie Eingänge lassen sich Ausgänge definieren. Sobald entsprechende Konfiguration eingegangen ist, werden diese selbstredend nicht abonniert, sondern als Attribut abgelegt, sodass der Agent direkt auf die zugreifen kann, um Nachrichten zu veröffentlichen.

Umsetzung Wenn eine Option definiert wird, abonniert die Basisklasse des Agenten automatisch das entsprechende Thema und nimmt auf diesem eingehende Nachrichten entgegen. Sobald die letzte benötigte Option eingeht, löst diese den Start des Agenten aus. Dies beinhaltet den Aufruf von Kontextmanagern des Agenten und der Abonnierung aller Eingangstopics. Stoppt der Agent werden diese Abonnements automatisch aufgelöst, bei Zerstörung des Agenten ebenso die Optionsabonnements.

3.4.2 Aufteilung

Es gibt bei der Aufteilung von Agenten keine allgemeine Regel für den Umfang. So könnten die drei Agenten des Heizungsthermostats auch als ein einzelner Agent mit einem allgemeineren Ziel, nämlich nur dem Regeln der Raumtemperatur, interpretiert werden. Auch möglich wäre eine weitere Aufteilung des Kontrollagenten in einen Agenten, der anhand der aktuellen Uhrzeit die Solltemperatur festlegt, und einen, der diese Solltemperatur einstellt. Je nach Anwendungsfall eignet sich eine unterschiedliche Granularität.

Entkopplung und Kompatibilität

In diesem Projekt ist es entscheidend, die Kompatibilität für verschiedenste Geräte zu ermöglichen. Da die Abstraktionsschicht selbst jedoch in einer Programmiersprache geschrieben werden muss, können durch sie selbst nicht alle Anwendungsszenarien unterstützt werden. So kann eine Lösung, die auf Mikrochips mit kleinsten Ressourcen lauffähig sein muss, zwangsläufig nicht den vollen Funktionsumfang von Betriebssystemen nutzen. Eine Lösung, die dem Entwickler Programmieraufwand erspart und dafür z. B. die Funktionen einer Hochsprache nutzt, ist wahrscheinlich nicht auf Mikrochips lauffähig.

Um hier einen Kompromiss zu finden, ist effektives Agentendesign vom Vorteil. Als Beispiel dient hier die Entwicklung eines Treibers für den Umweltsensor BME280. Dieser Chip ermittelt Temperatur, Luftfeuchtigkeit und -druck. Dazu sind im Groben die in Abbildung 3.7 gezeigten Arbeitsschritte nötig. Alle Arbeitsschritte bis auf die Berechnungen der Werte stellen hierbei I/O Operationen dar, die relativ einfach umzusetzen sind. Die Berechnungen selbst stellen mehrere mathematische Formeln dar, von denen nur alleine die Berechnung des Luftdrucks in der Referenzimplementierung des Herstellers in der Programmiersprache C 25 Zeilen einnimmt. [7]

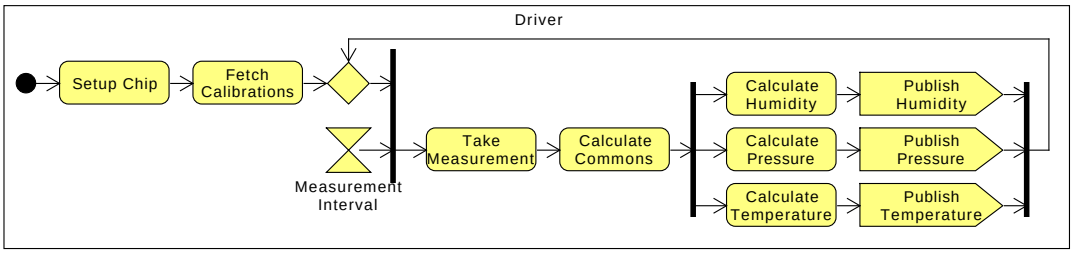


Abbildung 3.7: Aktivitätsdiagramm eines BME280 Treibers ohne Teilung

Wenn nun auf einer neuen Plattform dieser Chip angebunden werden soll, muss die Implementierung unter Umständen komplett von Grund auf neu durchgeführt werden. Um die Vorteile des Multiagentensystems nutzen zu können, bietet es sich an, diesen Agenten in zwei Teile zu spalten. Ein Agent kümmert sich um die I/O Operationen und läuft direkt auf der Plattform mit dem angebundenen *BME280*. Der andere Agent ist in einer Sprache verfasst, die allgemein auf möglichst vielen Systemen ausgeführt werden kann, also bspw. eine Interpretersprache, und kann überall im Multiagentensystem ausgeführt werden. [7]

Diese Interpretation führt zu den Arbeitsschritten in Abbildung 3.8. Bei neu hinzukommenden Plattformen muss so nur ein kleiner Teil des Agenten angepasst werden, welcher insgesamt am spezifischsten für die aktuelle Plattform ist. [7]

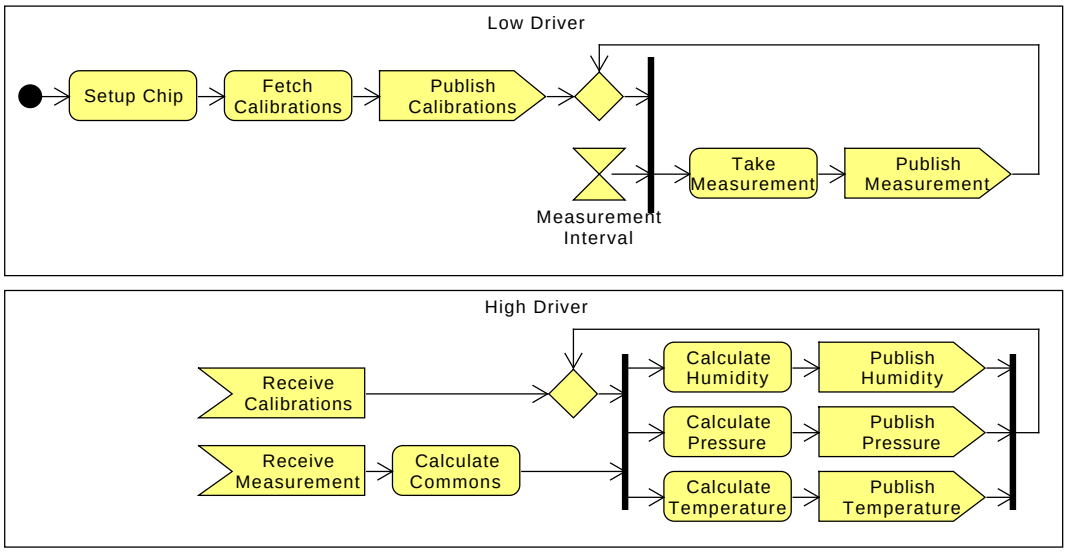


Abbildung 3.8: Aktivitätsdiagramm eines auf zwei Teile aufgeteilten BME280 Treibers

Wenn Komponenten angebunden werden sollen, die bereits vom Hersteller für andere Ökosysteme entworfen wurden, ist eine Integration notwendig. Dies kann durch die Erstellung eines Agenten durchgeführt werden, der als Brücke zwischen beiden Systemen fungiert. Um dem Agentenmodell treu zu bleiben, bietet es sich hier an, dass ein Brückenagent die Funktionen eines einzelnen Agenten in das jeweilige System überträgt.

Gruppierung

Da die Abstraktionsschicht innerhalb eines Betriebssystems läuft, wird eine Instanz von ihr als Systemprozess dargestellt. Konzeptionell stellt sich hier die Frage, in welcher Relation die Abstraktionsschicht mit den Agenten verbunden ist. So kann die Schicht einen einzelnen Agenten bedienen oder mehrere.

Nur einen einzelnen Agenten pro Instanz zu verwenden, würde zu einem größeren Overhead an Ressourcen führen. So müsste für jede Instanz eine eigene Verbindung zum Message Broker aufgebaut und gehalten werden und ein neuer Prozess bereitgestellt und verwaltet werden. Des Weiteren muss jede Instanz konfiguriert und ebenfalls verwaltet werden. So muss für jede Instanz ein Zertifikatepaar erstellt und installiert sowie der Dienst registriert werden. Dies kann wie angemerkt mit einem Configuration Management Tool vereinfacht werden, bei einer großen Anzahl von Agenten wird jedoch auch das Regelwerk des Tools u. U. unübersichtlich.

Die Bündelung von mehreren Agenten in einer Instanz kann zu mehr Ordnung und Übersichtlichkeit bei der Bereitstellung des Multiagentensystems sorgen. So können sie anhand ihrer allgemeinen Funktion oder der räumlichen Zugehörigkeit im realen Raum gruppiert werden.

Auch kann nach Leistungsaufwand gruppiert werden, indem sequenzielle, aufwendige Arbeitsschritte exklusiv einen Agenten erhalten. So könnten bspw. die Bildaufnahmen von mehreren Überwachungskameras im Haus verarbeitet werden. Die eingehenden Bilder sollen dazu zuerst rotiert, skaliert, binarisiert und anschließend auf Bewegung überprüft werden. Bei einer moderaten Aktualisierungsrate ist davon auszugehen, dass in dieser Kette maximal ein Schritt zur selben Zeit ausgeführt wird. Für jede Überwachungskamera wird deswegen für die Verarbeitung eine Instanz bereitgestellt, damit die Last gleichmäßig auf einzelne Prozesse verteilt wird. So können, trotz Gruppierung, die Hardwareressourcen optimal genutzt werden, ohne die Ausfallsicherheit zu verringern, da alle gruppierten Agenten voneinander abhängen.

3.4.3 Zustände und Fehlerbehandlung

Wie bereits angerissen, verfügt der Agent über mehrere Zustände, die er über den, in Abbildung 3.9 definierten Automaten erreichen kann. Während der Instanziierung des Agenten kann der Agentenentwickler die nötigen Optionen sowie Ein- und Ausgänge anzugeben und diesen anschließend als fertig konfiguriert markieren. Anschließend werden alle nötigen Optionen gesammelt und die Startprozedur ausgeführt, um den Agenten in den Ausführungszustand zu leiten.

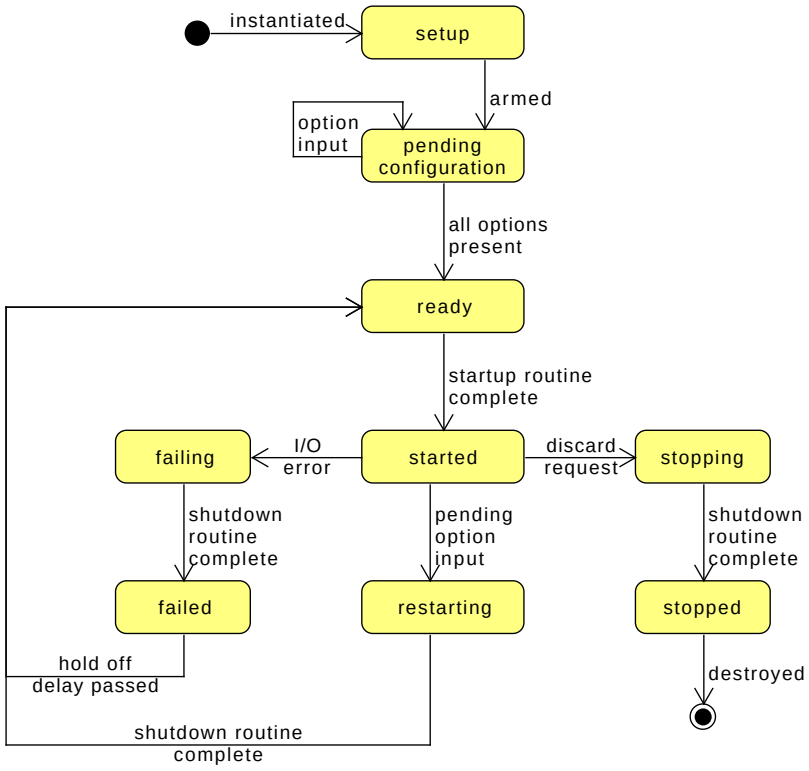


Abbildung 3.9: Automat für die Ausführungszustände des Agenten

Während der Ausführung ist zu erwarten, dass Fehler während I/O-Prozeduren auftreten. Dies kann z. B. aufgrund von minderwertigem Design von Baugruppen auftreten. Wenn der Agent dies anfordert, werden solche Fehler automatisch von der Basisklasse abgefangen und führen zum Stopp des Agenten. Nach einer festgelegten Vorhaltezeit wird dieser erneut gestartet. Je nach Anwendungsfall kann die Abstraktionsschicht konfiguriert werden, wie allgemein nicht abgefangene Ausnahmen aus Agenten behandelt werden sollen.

Eine Möglichkeit ist, dass diese Fehler zu einer Beendigung der Abstraktionsschicht führen, wodurch dem Dienstemanager sofort ersichtlich ist, dass ein Fehler aufgetreten ist. Dadurch können Fehler nur schwer übersehen werden, die restlichen Agenten der Instanz werden jedoch durch den Fehler mitgerissen und ihre Verfügbarkeit beeinträchtigt. Die andere Möglichkeit ist, für den Prozess nicht kritische Fehler abzufangen und nur den auslösenden Agenten zu deaktivieren, wodurch sich Vor- und Nachteile der vorherigen Lösung umkehren. Die besondere Behandlung von diesen Fehlern innerhalb dieses Projektes fußt auf der Verteiltheit und Asynchronität des Multiagentensystems. Tritt ein Fehler auf, so ist das Tracing der Ursache über mehrere Agenten und potenziell Ausführungsplattformen nötig, weswegen der dafür notwendige Aufwand weitestgehend eingedämmt werden sollte.

Zudem kann der Agent einen Neustart ohne Verzögerung ausführen, wenn dies z. B. aufgrund von geänderter Konfiguration angefordert wird. Externe Akteure wie der Spawner können den Agenten zudem anweisen, sich permanent zu deaktivieren. Dazu stoppt der Agent zuerst, gibt dann alle Ressourcen frei und entfernt sich aus der Shell, wodurch er für den Garbagecollector freigegeben wird.

3.4.4 Tests

Das Testen von Agenten in verteilten Systemen unterscheidet sich vom Testen einzelner Module in einem isolierten Programm. So findet die Kommunikation zwischen Agenten über Nachrichten statt und nicht über das Aufrufen von bspw. Methoden. Da das Kommunikationsmedium in der Regel agnostisch gegenüber der Nutzlast ist, kann diese auch nicht sicherstellen, dass Kommunikationsschnittstellen eingehalten werden. Daher liegt diese Aufgabe bei den Sendern und Empfängern, was gründliches Testen erfordert. [26]

Für die Entwicklung von Agenten bietet die Abstraktionsschicht vorgefertigte Mockups an, mit welchen automatisiert überprüft werden kann, ob ein Agent sich wie gefordert verhält. Dazu lässt sich überprüfen, ob dieser bestimmte Aufrufe auf Komponenten der Schicht in der der richtigen Reihenfolge und mit den richtigen Parametern ausführt. Dazu stellt die Abstraktionsschicht Mockups für das Anlegen von Tests zur Verfügung. Während der Definition des Tests definiert der Entwickler den Sollzustand dieser Ereignisliste sowie eventuelle zeitliche Abstände. Bei Ausführung des Tests protokollieren die Mockups Aufrufe und gleichen diese mit dem Sollzustand ab. Tritt ein Ereignis auf, welches nicht definiert wurde oder bleibt es aus, so bricht der Test mit einer Fehlermeldung ab.

3.4.5 Mixins

Verschiedene Agenten teilen sich öfter Teilfunktionalität, deren Umfang keine Abtrennung in einen neuen Agenten erfordert. So benötigen viele Agenten einen einzelnen, konfigurierbaren Timer, der eine Aufgabe wie das regelmäßige Abrufen von Sensormesswerten durchführt oder den Zugriff auf ein I2C Gerät. Dieser kann dadurch die Implementierung in jeden entsprechenden Agenten realisiert werden, führt jedoch zu hoher Redundanz. Deshalb wird die Verwendung von Mixins unterstützt, also dem Erben von Funktionalität durch abstrakte Zusatzklassen.

3.5 Evaluation

Nach der Beschreibung der Implementierung ist zu evaluieren, ob das in der Analyse aufgestellte Ziel erreicht wurde. Dazu wird zuerst überprüft, ob die gesetzten Szenarien angemessen erfüllt werden können. Folgend wird erfasst, mit welchen nicht funktionalen Funktionen die Schicht unterstützt. Anschließend wird anhand des Fazits der Analyse ermittelt, ob alle, in der Analyse aufgestellten Funktionalitäten gut umgesetzt werden konnten.

3.5.1 Szenarien

Die drei aufgestellten Szenarien sind so konzipiert, dass jede von ihnen eine gesteigerte Komplexität aufweist, damit mögliche Probleme bei deren Umsetzung besser identifiziert werden können. Diese wurden innerhalb der vorgestellten Testumgebung ausgeführt und analysiert.

Aufstehen

Für die Interaktion mit den verwendeten *SK6812* Pixeln wurden mehrere Agenten erstellt. Für die Beleuchtung jeden Raums ist ein spezialisierter Agent zuständig, der anhand von mehreren Eingängen, der Helligkeit der Umgebung, dem Zustand der Präsenzmelder und dem aktuell gewünschten Farbthema den Zustand jedes einzelnen Pixels im Raum bestimmt.

Diese Informationen werden an einen Merger weitergegeben, der die Listen der Farbwerte von mehreren Räumen zu einer aggregiert, da die Pixel mehrerer Räume von nur einem Controller besteuert werden. Es folgt ein Agent, der die Farbewerte in ein Bytearray konvertiert, welches vom nächsten Agenten mit der richtigen Taktrate über SPI zu den Pixeln übertragen wird. Zusätzlich zu der Raumbeleuchtung erhält der Merger von Konvertern einzelne Farbwerte, die den Zustand der Türen repräsentieren, da in der Evaluationsumgebung mechanischen Türen durch Leuchten repräsentiert werden sollen.

In der Evaluationsumgebung nutzt der Präsenzmelder die vorhanden, von welchen es zwei bis drei pro Raum gibt. Jeder Sockel verfügt über zwei digitale Eingänge zur Identifikation von Steckern, wodurch drei Teilnehmer voneinander unterschieden werden können. Obwohl für GPIO bereits Agenten zur Verfügung stehen, die einen einzelnen Eingang handhaben können, wird ein spezialisierter Agent für das Einlesen eines einzelnen Sockels verfasst, damit durch eine teilweise Aktualisierung keine ungültigen Zustände entstehen können. Ein weiter Agent ordnet der, vom vorherigen Agenten ausgegebenen, ID einen Klartextnamen zu, der von anderen Agenten einfacher verwendet werden kann. Die restlichen Funktionen lassen sich über GPIO realisieren, welche von entsprechender Logik beschaltet werden.

Dieses Szenario diente der Überprüfung, ob das implementierte System in der Lage ist, verteilt die Werte eines Sensors entgegenzunehmen, die Information logisch zu verarbeiten und mehrere Aktoren zu steuern und wurde vom System erfüllt.

Ventilation

Im letzten Szenario soll überprüft werden, wie sich ein komplexerer Regelkreis mit dem System umsetzen lässt. Jeder, nicht in einen anderen Raum mündender, Raum verfügt über Ventilation, die Luft aus dem Raum absaugt. Frische Luft strömt ausschließlich über angrenzende Räume aus dem Atrium nach. Das System muss anhand der Umweltsensoren jedes Raums ermitteln, mit welcher Intensität jeder einzelne Ventilator betrieben werden muss. Dabei ist in fast jedem Fall ein Kompromiss zwischen den Räumen im Belüftungspfad eines jeden Ventilators zu finden.

Je nach Nutzerkonfiguration wird dabei entweder der Raum mit dem höchsten Bedarf an Lüftung als Referenz genommen oder der Durchschnitt aller betroffenen Räume gebildet. Mit dieser Information wird die Differenz zum Sollwert gebildet. Überschreitet die Differenz einen definierten Wert, werden die Ventilatoren auf höchster Leistung gestartet und so lange degressiv angepasst, bis der Zielwert erreicht ist.

Unabhängig von der vorherigen Konfiguration wird bei einer sehr niedrigen Luftqualität in mehr als einem Raum davon ausgegangen, dass eine für die Bewohnerin gefährliche Situation vorliegen kann. In so einem Fall wird ein Alarm ausgelöst. Weisen Präsenzmelder zudem darauf hin, dass diese sich im Haus befinden, lüften alle Ventilatoren mit voller Leistung. Wird gemeldet, dass die Bewohnerin das Haus verlassen hat, stoppt die Belüftung, um z. B. Situationen wie einen Hausbrand nicht zu verschlimmern.

Um sicherzustellen, dass eine möglichst hohe Ausfallsicherheit gewährleistet werden kann, werden die für die Logik zuständigen Agenten als Dienste innerhalb eines Docker Swarms installiert, welcher alle Embeddedsysteme der Testumgebung beinhaltet. Dadurch kann bei einem partiellen Ausfall der Plattform durch Umverteilung des Dienstes auf andere Plattformen ein Funktionsausfall vermieden werden. Zudem kann die elektronische Steuereinheit der Ventilatoren von mehreren Schnittstellenagenten angesprochen werden, wodurch ein Ausfall von einem dieser Agenten kompensierbar ist.

Der Einsatz von Clusteringlösungen auf Embeddedsystemen stellt nach wie vor aufgrund der dort begrenzten Ressourcen eine Herausforderung dar, da viele Clusterimplementierungen hauptsächlich auf vollwertigen Servern eingesetzt werden. Ohne diese Lösungen ist jedoch das Umsetzen von ausfallsicheren Ökosystemen eine Herausforderung, wenn nicht deutliche Einschnitte bei anderen Kriterien hingenommen werden. [70]

Bird Alert

Dieses Szenario hat das Ziel, die im vorherigen Szenario eingerichtete Architektur mit komplexeren Tests zu versehen, um realistischere Anwendungsfälle zu simulieren. Dazu werden die bereits bestehenden Agenten für Beleuchtung, Türen und Präsenzmelder übernommen.

Zusätzlich wird mittels der implementierten Logikagenten das Regelwerk erweitert. Für jede Tür, die an einen, für den Vogel verbotenen Raum grenzt, wird ein Agent eingesetzt, der anhand des Präsenzmelders bestimmt, ob die Tür zu schließen ist.

Selbiger Agent öffnet die Tür wieder, sobald der Vogel nicht mehr im entsprechenden Raum ist. Für jeden verbotenen Raum wird zudem ein Konverter angelegt, der bei Präsenz des Vogels seine Warnstufe ausgibt. Ein Aggregator bündelt alle Warnungen und gibt die kritischste Warnstufe weiter. Diese Warnstufe wird von anderen Agenten verwendet, um die Räume entsprechend zu beleuchten, um den Zustand zu visualisieren.

Bei diesem Szenario zeigt sich, wie kritisch die Übersichtlichkeit über die Konfiguration des Multiagentensystems ist. Das hier verwendete Konfigurationstool ermöglicht die Aufteilung der Agentenkonfiguration in Dateien und Ordner und bietet die Möglichkeit, Themendefinitionen von der eigentlichen Konfiguration zu trennen. Trotzdem würde sich hier der Einsatz einer grafischen Oberfläche anbieten, um den Benutzer direkten Zugriff auf relevante Informationen zu bieten und Änderungen schnell umsetzbar gestalten zu können. Die Entwicklung einer solchen Oberfläche könnte in Folgearbeiten umgesetzt werden.

3.5.2 Nicht funktionale Funktionen

Neben der Überprüfung der funktionalen Funktionalität anhand der Szenarien erfüllt die Abstraktionsschicht nicht funktionale Aspekte, die Entwicklern beim Umsetzen von eigenen Projekten zugutekommen.

Reduzierte Entwicklungszeit Eines der hauptsächlichen Ziele der Abstraktionsschicht, der Verkürzung der Entwicklungszeit von CPS Projekten, wird u. a. durch die umgesetzte Vereinfachung des Hardwarezugriffs und der Konfiguration über das Publish/Subscribe-Verzeichnis erreicht.

Interoperabilität Durch die Umsetzung der Kommunikation über MQTT und vielseitigen Serialisierungsmethoden wird die Anbindung, vor allem von Geräten mit beschränkten Hardwareressosurcen, effizient umsetzbar gestaltet.

Erweiterbarkeit Durch das Design als Multiagentensystem ist bereits architektonisch vorgesehen, das System in entkoppelte Komponenten auszuteilen. Die Menge dieser kann durch das Hinzufügen weiterer Agenten ergänzt werden, um neue Funktionalität bereitzustellen oder bestehende zu erweitern.

Wartbarkeit Agenten können aufgrund ihrer Entkopplung bereits unabhängig voneinander getestet und integriert werden. Durch das dynamische Laden von Agenten durch die SHELLS ist es zudem möglich, Implementierungen ohne Neustart von Prozessen durchzuführen und zu integrieren, was die Wartbarkeit vereinfacht.

Benutzbarkeit Es wurde eine verbreitete und universell einsetzbare Programmiersprache für die Abstraktionsschicht verwendet, um möglichst vielen Entwicklern die Verwendung der Schicht zugänglich zu machen. Durch die Plattformunabhängigkeit dieser entfallen zudem komplexe Buildprozesse, was ebenfalls die Benutzbarkeit steigert.

3.5.3 Fazit

Dieses Kapitel hat gezeigt, dass eine Umsetzung der vorher aufgestellten Anforderungen sehr gut durchführbar ist. Für die Kommunikationsform bietet sich das Protokoll MQTT an, für welches zahlreiche Implementierungen zur Wahl stehen, die teilweise clusteringfähig sind.

Provisioning und Configuration Management Tools ermöglichen es, einen Großteil des anfallenden Konfigurations- und Wartungsaufwands abzufedern und durch Automatisierung Fehlerquellen zu reduzieren, welche ansonsten im späteren Verlauf der Entwicklung zu Komplikationen führen würden.

Indem die komplette Agentenkonfiguration in das Publish/Subscribe-Verzeichnis eingelagert wird, erhalten Agenten immer den aktuellsten Stand der Konfiguration und können dynamisch rekonfiguriert werden. Als Programmiersprache wurde Python gewählt, welche sich durch eine hohe Kompatibilität sowohl zur Hardwareschicht als auch zu komplexen Berechnungsaufgaben eignet.

Durch die Beschränkung der Ausführungsplattform auf vollwertige Betriebssysteme kann den Entwicklern die größte Freiheit für die Entwicklung von Agenten geboten werden, ohne den generellen Zugang zu Embeddedprojekten signifikant zu erschweren.

Durch die Umsetzung eines eigenen MQTT Clienten kann sichergestellt werden, dass die Dienstgüte bei der Nachrichtenübertragung eingehalten wird. Mitgelieferte Agenten kümmern sich um Grundfunktionen der Abstraktionsschicht, wodurch diese Funktionalität einfach erweitert oder ausgetauscht werden kann.

Mit der Aufstellung von Kriterien für die Aufteilung von Agenten und dem Einordnen von Informationen in das Publish/Subscribe-Verzeichnis ist ein Leitfaden gegeben, anhand welchem Entwickler ihr Projekt effizient planen können. Durch die Behandlung des Ausführungszykluses und eventuellen Fehler können zudem Ausfallzeiten reduziert werden.

Während der Bearbeitung hat sich gezeigt, dass das Projekt einen hohen Umfang für eine einzelne Person aufweist. Um es langfristig bereitstellen zu können, sollte das Entwicklerteam der Abstraktionsschicht erweitert werden. Dadurch könnte auch angegangen werden, eine grafische Oberfläche zum Konfigurieren des Systems anzubieten, wodurch zusätzlich Aufwand reduziert werden könnte.

4 Schluss

Abschließend wird in diesem Kapitel eine Zusammenfassung dargelegt und eine Perspektive für weiterführende Arbeiten skizziert.

4.1 Zusammenfassung

In der Einleitung wurde erfasst, das für die Entwicklung von Softwarekomponenten für Smart Environments dereit keine einheitliche Lösung existiert, die eine unterschiedliche hardwarenähe für einzelne Komponenten unterstützt. Aus diesem Grund soll als Problemstellung behandelt werden, wie mit einer einzelnen Abstraktionsschicht jene Softwarekomponenten, die als in einem Netzwerk kommunizierende Elemente fungieren, entwickelt werden können. Diese agieren als ein System von Systemen und entsprechen einem CPS. Dazu wurde als Forschungsthema behandelt, welche Anforderungen sich an eine solche Abstraktionsschicht stellen und wie diese zu erfüllen sind.

In der Analyse wurde die Untersuchung relevanter Begrifflichkeiten und des Diskussionsstandes erfasst, welcher mit einer Übersicht über relevante Artikel ergründet wurde. Es folgt die Aufstellung einer Fallstudie, in welcher anhand einer Simulationsumgebung für Smart Environments getestet wurde, ob eine entwickelte Abstraktionsschicht realistischen Anforderungen genügen kann. Dazu wurden drei Szenarien aufgestellt, in denen verschiedene Anforderungsgruppen geprüft wurden. Anschließend wurde der Stand der Technik untersucht und Bezug auf verwendbare Sensorik- und Aktorikkomponenten genommen sowie die Besonderheiten von Smart Environments aufgeschlüsselt. Wie Softwarekomponenten miteinander interagieren, Daten mittels CEP verarbeitet werden und welche Ausführungsplattformen unterstützt werden können, wird vor dem Fazit beschrieben, in welchem abschließend relevante Erkenntnisse zusammengefasst werden.

Im Kapitel *Architektur*, wird die Umsetzung der Abstraktionsschicht beschrieben. Hier wurde zuerst ermittelt, mit welcher Kommunikationsform einzelne Softwareagenten miteinander interagieren können, indem anhand der in der Analyse aufgestellten Anforderungen eine Auswahl aus möglichen Umsetzungen getroffen wurde. Zudem wurde behandelt, mit welcher Struktur Daten einsortiert werden sollen. Des Weiteren wurde die Provisionierung der Abstraktionsschicht behandelt, da dieses einen kritischen Faktor bei der Anwendbarkeit dieser darstellt. Der folgende Abschnitt befasst sich mit der Beschreibung der Abstraktionsschicht, also der Konfiguration, Struktur und Umsetzung dieser. Zudem werden einige kritische Agenten mit der Schicht ausgeliefert und hier erklärt. Daraufhin erfolgt die Definition eines Agenten, welcher von den Entwicklern erstellt werden soll. Es wird beschrieben, wie ein Agent Optionen erhält, welchen Umfang ein einzelner Agent aufweisen soll, welche Laufzeitzustände er aufweisen kann, wie Tests vorgenommen und wie generische Funktionalität geteilt werden können. Abschließend erfolgt die funktionale Evaluation, in welcher anhand der Szenarien festgestellt wird, ob die erstellte Abstraktionsschicht realitätsnah einsetzbar ist. Abschließend erfolgt ein Fazit, in welchem behandelt wird, was das entwickelte Softwareprojekt für Vorteile aufweist und was verbessert werden kann.

4.2 Ergebnis

Die hier konkret für Smart Homes ermittelten Erkenntnisse lassen sich auch auf andere CPSs und Smart Environments generalisieren. Dazu gehören neben Wohnbereichen Produktionsumgebungen, welche mit Sensorik und Aktorik ausgestattet wurden und Officeumgebungen. Auch hier gilt das gleiche Prinzip wie in der Fallstudie, eine robuste und dezentrale Architektur ohne Single Point of Failure trägt zu einem effektiven System bei, welches mit Fehlersituationen umgehen und schneller an geänderte Umstände angepasst werden kann.

Durch das Schichtenmodell und die Unterstützung von Pluginfähigkeit ist zudem eine einfach umsetzbare Erweiterbarkeit und Rekonfigurierbarkeit gegeben. Dadurch, dass eine Vielzahl potenziell leistungsschwacher Geräte an einem Standort existiert, können diese nur durch einen verteilten Ansatz vollständig ausgenutzt werden.

Ein zentralisierter Ansatz vereinfacht jedoch u. U. die Bereitstellung und Wartung eines solches Systems. Der Einsatz einer Abstraktionsschicht ist ebenso sinnvoll, um auch über einen längeren Zeitraum wartbare Komponenten bereitstellen zu können, wie dies vor allem in solchen komplexen Umgebungen von Nöten ist. Die Generalisierbarkeit konstruierter Elemente ist dabei entscheidend, um sie in geänderten Einsatzumgebungen ebenso einsetzen zu können. Dazu gehört, dass die Abstraktionsfähigkeit für mehrere Schichten, seien sie hardwarenahe oder -fern, erfolgen muss. Diese Architektur lässt sich in Zukunft auch auf andere Bereiche übertragen, welche ähnlichen Faktoren ausgesetzt sind.

4.3 Reflexion

Die vorgestellte Lösung erfüllt die aufgestellten Anforderungen und verfügt über Funktionen, die das Entwickeln von CPS Komponenten beschleunigen. Sie muss sich jedoch außerdem gegen bestehende Projekte durchsetzen. Da CPSs, je nach Einsatzzweck, teils sehr unterschiedliche Anforderungen aufweisen, die sich zudem noch im Wandel befinden, ist ein Vergleich anhand von einzelnen Kriterien nicht zielführend. Welche Lösung sich für einzelne Problemstellungen eignet, ist vielmehr anhand der gewünschten Architektur Mindestfunktionsmenge zu bestimmen.

Die hier verwendete Agentensicht ist besonders für komplexere Ökosysteme vorteilhaft, die aus einer größeren Menge von Agenten bestehen. Für Szenarien, die nur eine sehr kleine Anzahl von Agenten benötigen, ist der Overhead durch die Einarbeitungszeit in das Konzept und u. a. ausgiebiges Testen höher, als wenn bspw. klassische Lösungen wie die Kommunikation einzelner Programme über REST verwendet werden, da das Konzept Entwicklern allgemein bekannt und das System aufgrund dessen Einfachheit anhand ihrer Funktionalität getestet werden können.

Die Erfassung der Anforderungen fand anhand einer reinen Simulationsumgebung statt. Wie beschrieben hat dies den Vorteil, dass so deutlich effektiver Einfluss auf das Testsystem genommen werden kann, als es bspw. innerhalb einer echten Wohnung möglich wäre. Als Nachteil weist diese Variante jedoch auf, dass Szenarien eigens entwickelt und durchgeführt werden müssen, während im Verlauf eines Langzeittests in einem bewohnten Smart Home Situationen auftreten können, durch welche neue Erkenntnisse abgeleitet werden können. Der Einsatz der Lösung muss also in weiteren Arbeiten in realen Umgebungen auf die Probe gestellt werden.

4.4 Aussicht

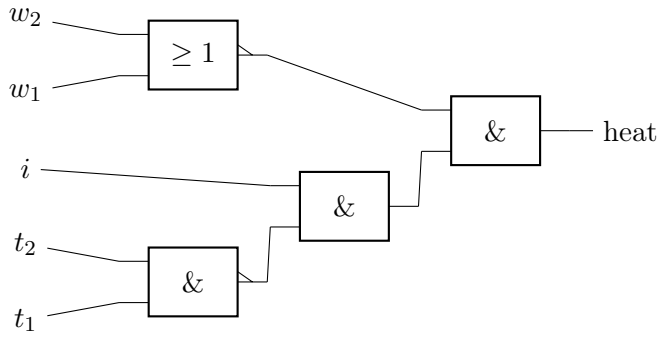
Als wie robust sich das ermittelte Modell sich in der allgemeinen Anwendung erweisen wird, ist noch nicht absehbar, da die Technologie des Forschungsumfelds rapide Entwicklung erfährt und Technologiesprünge absehbar sind, die die Anforderungen signifikant ändern können. Mit einem endgültigen Ergebnis ist zu rechnen, sobald sich dieser Trend stabilisiert und Smart Environments verbreitete Anwendung in den Einsatzgebieten erfahren.

A Anhang

A.1 Datenarten

Booleans

Komponentenzustände werden vielfach durch Wahrheitswerte bestimmt, sei es die Informationen, ob z.B eine Lampe eingeschaltet sein soll oder ein Türkontakt offen ist. Um diese Informationen zu ermitteln oder weiterzuverarbeiten, ist Boolsche Logik bestimmt, also der Aggregierung dieser Werte über z. B. Verundung und Veroderung. Soll bspw. ermittelt werden, ob eine Gasttherme den Heizkreislauf erwärmen soll, ergibt sich dies aus den Informationen, ob in den Räumen der Wohnung die Zieltemperatur unterschritten ist, ob der Bewohner anwesend ist und ein oder mehrere Fenster geöffnet sind. Eine solche Logik kann in Gleichungsform, programmatisch oder visuell angegeben werden, wie in Abbildung A.1 dargestellt.




```
heat = not (t1 and t2) and not (w1 or w2) and i
```

$heat = \neg(t_1 \wedge t_2) \wedge \neg(w_1 \vee w_2) \wedge i$

Abbildung A.1: Beispiel für Boolsche Logik, dargestellt als Gatterschaltbild, Python Code und Boolsche Algebra

Numerik

Um eben genannte Wahrheitswerte zu ermitteln, ist oft die Analyse von numerischen Werten von Nöten. Im vorherigen Beispiel ist u. a. zu ermitteln, ob die Zieltemperatur eines Raumes den Zielwert unterschreitet. Durch den Einsatz von Vergleichsoperation kann hier aus numerischen Werten ein Wahrheitswert generiert werden. Eine weitere Anwendung umfasst auch die Generierung neuer Zahlenwerte durch andere Operationen, z. B. zum Ermitteln der Durchschnittstemperatur aller Räume. Zudem können numerische Werte Enumeratoren repräsentieren, die für die Wahl eines Elements aus einer Sammlung von nicht numerischen Werten stehen, z. B. für den aktuellen Tag der Woche.

Die Besonderheit dieser Datenart ist, dass je nach Anwendungsfall die korrekte Repräsentation auf Byte-Ebene gefunden werden muss. Während Wahrheitswerte durch ein einzelnes Byte repräsentiert werden (üblicherweise steht der Wert Null für falsch, alle anderen Werte für wahr), gibt es unterschiedliche Darstellungsformen für Ganzzahlen, die vorzeichenbehaftet oder -los sein können. Für Gleitkommazahlen existiert eine IEEE Norm, die eine Serialisierungsform mit zwei Präzisionen beschreibt, mit welcher diese Zahlen näherungsweise beschrieben werden können.

Für einen Anwendungsfall muss also entweder vorher ermittelt werden, welchem Wertebereich und Präzision sie umfassen soll, oder die Zahl muss als andere Datenart, wie z. B. als String serialisiert werden. Dies führt zwar zu einem höherem Entwicklungskomfort, vermindert jedoch die Geschwindigkeit der Serialisierung und verschlechtert die Platzeffizienz.

Listen

Eben genannte Datentypen repräsentieren in der Regel eine einzelne Information, jedoch können komplexe Datentypen aus mehreren Teilinformationen bestehen. So besteht ein RGB Farbwert aus den einzelnen Farbwerten der jeweiligen Kanäle, also Rot, Grün und Blau, üblicherweise dargestellt als je ein Byte. Zur Darstellung der Farbe ist es nicht zielführend, die Farbkanäle unabhängig voneinander zu betrachten, da dadurch die eigentliche Information fragmentiert wird. Ein solcher Fall erfordert die Darstellung von Werten als Liste, d. h. primitive Datentypen werden direkt hintereinander in den Byte-Strom serialisiert. Dies entspricht dem komplexen C Datentyp *struct* und erfordert die Zusatzinformation, welche Datentypen in welcher Reihenfolge serialisiert wurden. Neben dem Aufwand, diese Information zusätzlich zu übergeben, bietet dieses Format maximale Platzeffizienz.

Strings

Ein String ist eine zusammenhängende Liste von einzelnen Zeichen und Symbolen und bietet eine Möglichkeit für die universelle Darstellung jeglicher Daten, potenziell zum Preis von Performance und Platzeffizienz. Während zu Beginn der Informatikära Strings über die sog. ASCII Tabelle codiert wurden, welche jedoch nur einen sehr beschränkten Zeichensatz unterstützte, werden Strings heutzutage mittels Unicode codiert, einem Standard, der das Ziel hat, alle relevanten Zeichen der Welt zu unterstützen. Realisiert wird dies, indem jedes Zeichen, nicht wie bei ASCII eine feste Länge von einem Byte hat, sondern indem, je nach Position in der Kodierungstabelle, durch eine unterschiedliche Bitlänge repräsentiert wird. Strings haben, unabhängig von der Codierung, eine variable Anzahl von Zeichen, außer sie werden durch Konventionen begrenzt.

Strukturierte Daten

Zu der eben genannten Möglichkeit, komplexe Informationen als Listen darzustellen, existieren Formate, die durch Ihren Aufbau die Beschreibung der Serialisierung beinhalten.

JSON bspw. ist ein Format, welches die Kombination von Strings und Zahlen zu Listen und Verzeichnissen ermöglicht. Serialisierer ermitteln beim Entpacken automatisch die benötigten Datentypen und ersparen hier Konfigurationsaufwand. Die meisten Serialisierer verfügen jedoch nicht über die Möglichkeit, die Struktur der Eingabe zu validieren, weswegen beim Weiterverarbeiten des Datensatzes Vorsichtsmaßnahmen zu ergreifen sind. Damit unterscheidet sich JSON von Extensible Markup Language (XML), welches sich mit XSD Schema Beschreibungen validieren lässt.

Erwähnenswert ist zudem YAML, mit welcher JSON kompatibel ist. Dieses Format zeichnet sich, neben den Fähigkeiten von JSON, dadurch aus, innerhalb des Dokumentes Referenzen zu ermöglichen, was redundante Informationen reduziert.

Themen

Wie beschreiben, soll die Abstraktionsschicht Ein- und Ausgänge dynamisch an Agenten anbinden können. Um diese Kernkomponente zu realisieren, ist es notwendig, diese Informationen auch serialisieren zu können. Zusätzlich zur Information, welches Thema zu abonnieren ist, sind Informationen wie die Dienstgüte, welche mit diesem Thema verknüpft werden soll, relevant. Da es sich hierbei um einen Datentypen handelt, der eine geringe Aktualisierungsrate aufweist und einfach konfigurierbar sein soll, wird diese Information in JSON serialisiert. Explizit einzuplanen ist die Möglichkeit, dass eine einzelne Themakonfiguration mehrere einzelne Themen beinhalten kann, wie es z. B. bei Aggregatoren der Fall sein kann.

Blobs

Neben den bisher, von ihrer Gesamtgröße her überschaubaren Daten, soll es auch möglich sein, große binäre Objekte zu übertragen. Dies könnten Fotos, Videos oder Audio sein. Für diese Arten existieren bereits Beschreibungen wie JPEG, Theora und Vorbis. Diese Formate verwenden ausgefeilte Kompressionsalgorithmen, um die zu übertragende Datenmenge zu minimieren. Da dies ein komplexer Vorgang ist, ist hierbei auf Bibliotheken zuzugreifen, welche die aktuelle Version des Standards fehlerfrei handhaben können.

A.2 Use Cases

Zur Übersicht, welche Funktionalität die Abstraktionsschicht bietet, sei auf Abbildung A.2 verwiesen, welche diese in einem Anwendungsfalldiagramm anordnet und in Kategorien einteilt.

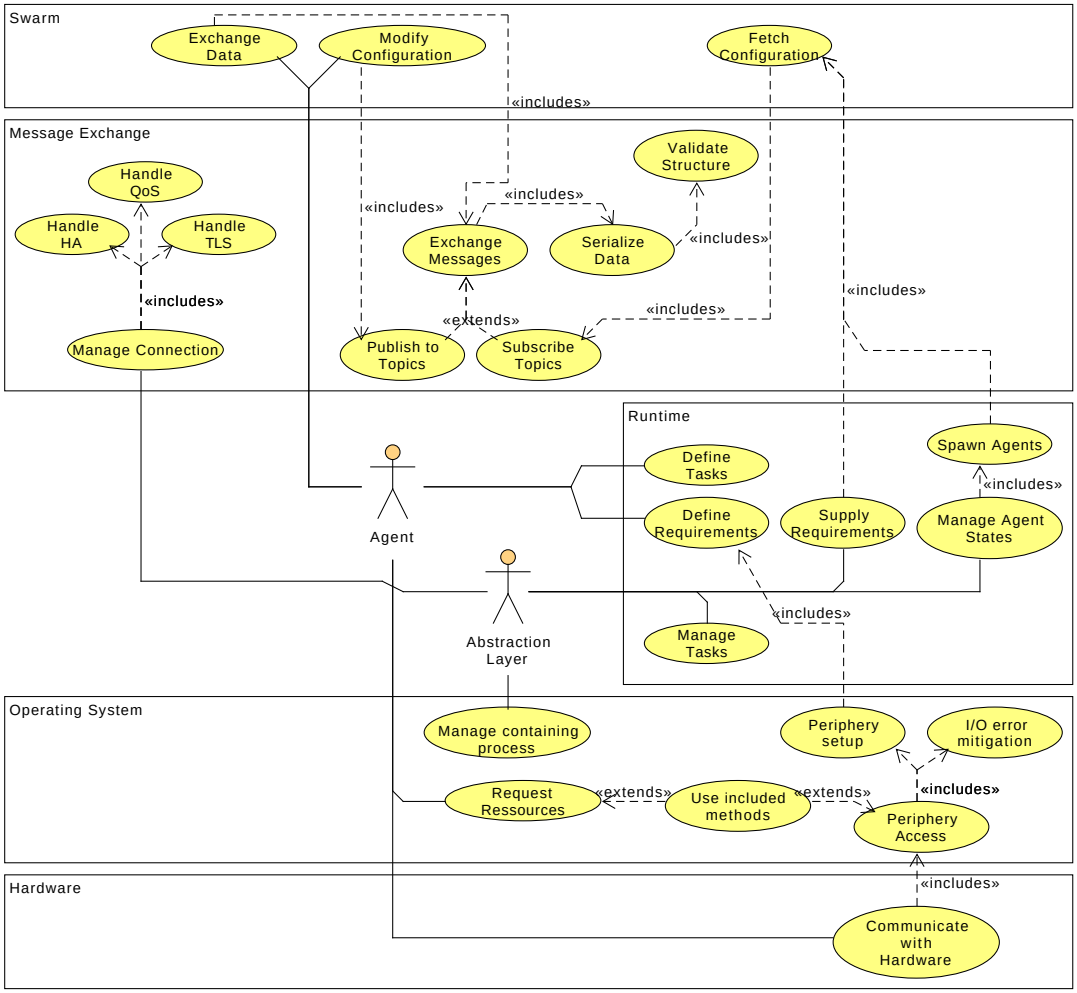


Abbildung A.2: Anwendungsfälle innerhalb des Projekts

Literaturverzeichnis

[1] : *How the Actor Model Meets the Needs of Modern, Distributed Systems*. – URL https://doc.akka.io/docs/akka/2.5/guide/actors-intro.html. – Zugriffsdatum: 2019-01-02

[2] : *TIOBE Index for November 2018*. – URL https://www.tiobe.com/tiobe-index. – Zugriffsdatum: 2018-11-05

[3] : *VDE/VDI 2653*

[4] *BME680 Low power gas, pressure, temperature & humidity sensor*, 2017. – URL https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BME680-DS001.pdf. – Zugriffsdatum: 2019-02-01

[5] *IKEA Smart lighting*, 2017. – URL https://www.ikea.com/gb/en/doc/general-document/ikea-ikea-smart-lighting-pdf__1364484962639.pdf. – Zugriffsdatum: 2019-02-01

[6] *ADS101x Ultra-Small, Low-Power, I2C-Compatible, 3.3-kSPS, 12-Bit ADCs with Internal Reference, Oscillator, and Programmable Comparator*, 2018. – URL https://www.ti.com/lit/ds/symlink/ads1015.pdf. – Zugriffsdatum: 2019-02-01

[7] *BME280 Combined humidity and pressure sensor*, 2018. – URL https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BME280-DS002.pdf. – Zugriffsdatum: 2019-02-01

[8] Al-Fuqaha, Ala ; Guizani, Mohsen ; Mohammadi, Mehdi ; Aledhari, Mohammed ; Ayyash, Moussa: Internet of things: A survey on enabling technologies, protocols, and applications. In: *IEEE Communications Surveys & Tutorials* 17 (2015), Nr. 4, S. 2347–2376

[9] Armstrong, Joe: *Making reliable distributed systems in the presence of software errors*, Mikroelektronik och informationsteknik, Dissertation, 2003

[10] Bandyopadhyay, Soma ; Sengupta, Munmun ; Maiti, Souvik ; Dutta, Subhajit: Role of middleware for internet of things: A study. In: *International Journal of Computer Science and Engineering Survey* 2 (2011), Nr. 3, S. 94–105

[11] Banks, Andrew ; Gupta, Rahul: MQTT Version 3.1. 1. In: *OASIS standard* 29 (2014), S. 89

[12] Bao, Kaibin ; Mauser, Ingo ; Kochanneck, Sebastian ; Xu, Huiwen ; Schmeck, Hartmut: A microservice architecture for the intranet of things and energy in smart buildings. In: *Proceedings of the 1st International Workshop on Mashups of Things and APIs* ACM (Veranst.), 2016, S. 3

[13] Benson, James O. ; Prevost, John J. ; Rad, Paul: Survey of automated software deployment for computational and engineering research. In: *Systems Conference (SysCon), 2016 Annual IEEE* IEEE (Veranst.), 2016, S. 1–6

[14] Blem, Emily ; Menon, Jaikrishnan ; Sankaralingam, Karthikeyan: Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures. In: *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on* IEEE (Veranst.), 2013, S. 1–12

[15] Boehm, Barry W.: A spiral model of software development and enhancement. In: *Computer* 21 (1988), Nr. 5, S. 61–72

[16] Byun, Jinsung ; Jeon, Boungju ; Noh, Junyoung ; Kim, Youngil ; Park, Sehyun: An intelligent self-adjusting sensor for smart home services based on ZigBee communications. In: *IEEE Transactions on Consumer Electronics* 58 (2012), Nr. 3

[17] Cardenas, Alvaro A. ; Amin, Saurabh ; Sastry, Shankar: Secure control: Towards survivable cyber-physical systems. In: *Distributed Computing Systems Workshops, 2008. ICDCS'08. 28th International Conference on* IEEE (Veranst.), 2008, S. 495–500

[18] Celesti, Antonio ; Mulfari, Davide ; Fazio, Maria ; Villari, Massimo ; Puliafito, Antonio: Exploring container virtualization in IoT clouds. In: *Smart Computing (SMARTCOMP), 2016 IEEE International Conference on* IEEE (Veranst.), 2016, S. 1–6

[19] Cenedese, Angelo ; Zanella, Andrea ; Vangelista, Lorenzo ; Zorzi, Michele: Padova smart city: An urban internet of things experimentation. In: *World of Wire-*

less, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a IEEE (Veranst.), 2014, S. 1–6

[20] Chan, Marie ; Campo, Eric ; Estève, Daniel ; Fourniols, Jean-Yves: Smart homes - current features and future perspectives. In: *Maturitas* 64 (2009), Nr. 2, S. 90–97

[21] Chan, Marie ; Estève, Daniel ; Escriba, Christophe ; Campo, Eric: A review of smart homes - Present state and future challenges. In: *Computer methods and programs in biomedicine* 91 (2008), Nr. 1, S. 55–81

[22] Cook, Diane J.: Multi-agent smart environments. In: *Journal of Ambient Intelligence and Smart Environments* 1 (2009), Nr. 1, S. 51–55

[23] Cook, Diane J. ; Crandall, Aaron S. ; Thomas, Brian L. ; Krishnan, Narayanan C.: CASAS: A smart home in a box. In: *Computer* 46 (2013), Nr. 7, S. 62–69

[24] Das, Sajal K. ; Cook, Diane J. ; Battacharya, Amiya ; Heierman, Edwin O. ; Lin, Tze-Yun: The role of prediction algorithms in the MavHome smart home architecture. In: *IEEE Wireless Communications* 9 (2002), Nr. 6, S. 77–84

[25] Dobler, A ; Uckelmann, D ; Lückemeyer, G: 4 SIMPLIFY OPENHAB UTILIZATION IN PUBLIC INSTITUTIONSUSING CLOUD TECHNOLOGIES. In: *Smart Public Building 2018* (2018), S. 31

[26] Eichler, Tobias: Agentenbasierte middleware zur Entwicklerunterstützung in einem Smart-Home-Labor. (2014)

[27] Esposito, Antonio ; Loia, Vincenzo: Integrating concurrency control and distributed data into workflow frameworks: an actor model perspective. In: *Systems, Man, and Cybernetics, 2000 IEEE International Conference on* Bd. 3 IEEE (Veranst.), 2000, S. 2110–2114

[28] Eugster, Patrick T. ; Felber, Pascal A. ; Guerraoui, Rachid ; Kermarrec, Anne-Marie: The many faces of publish/subscribe. In: *ACM computing surveys (CSUR)* 35 (2003), Nr. 2, S. 114–131

[29] Ferdoush, Sheikh ; Li, Xinrong: Wireless sensor network system design using Raspberry Pi and Arduino for environmental monitoring applications. In: *Procedia Computer Science* 34 (2014), S. 103–110

[30] Geisberger, E: *Agenda CPS. Integrated research agenda cyber-physical systems. Acatech Study.* 2012

[31] Geisberger, Eva ; Broy, Manfred: *agendaCPS: Integrierte Forschungsagenda Cyber-Physical Systems.* Bd. 1. Springer-Verlag, 2012

[32] Haegg, Staffan: A sentinel approach to fault handling in multi-agent systems. In: *Australian Workshop on Distributed Artificial Intelligence* Springer (Veranst.), 1996, S. 181–195

[33] Hafeez, Ayesha ; Kandil, Nourhan H. ; Al-Omar, Ban ; Landolsi, Taha ; Al-Ali, AR: Smart home area networks protocols within the smart grid context. In: *journal of Communications* 9 (2014), Nr. 9, S. 665–671

[34] Hakiri, Akram ; Berthou, Pascal ; Gokhale, Aniruddha ; Abdellatif, Slim: Publish/subscribe-enabled software defined networking for efficient and scalable IoT communications. In: *IEEE communications magazine* 53 (2015), Nr. 9, S. 48–54

[35] Heo, Sehyeon ; Woo, Sungpil ; Im, Janggwan ; Kim, Daeyoung: IoT-MAP: IoT mashup application platform for the flexible IoT ecosystem. In: *Internet of Things (IOT), 2015 5th International Conference on the* IEEE (Veranst.), 2015, S. 163–170

[36] Jain, Sarthak ; Vaibhav, Anant ; Goyal, Lovely: Raspberry Pi based interactive home automation system through E-mail. In: *Optimization, Reliabilty, and Information Technology (ICROIT), 2014 International Conference on* IEEE (Veranst.), 2014, S. 277–280

[37] Kelly, Sean Dieter T. ; Suryadevara, Nagender K. ; Mukhopadhyay, Subhas C.: Towards the implementation of IoT for environmental condition monitoring in homes. In: *IEEE Sensors Journal* 13 (2013), Nr. 10, S. 3846–3853

[38] Klein, Laura ; Kwak, Jun-young ; Kavulya, Geoffrey ; Jazizadeh, Farrokh ; Becerik-Gerber, Burcin ; Varakantham, Pradeep ; Tambe, Milind: Coordinating occupant behavior for building energy and comfort management using multi-agent systems. In: *Automation in construction* 22 (2012), S. 525–536

[39] Kliem, Andreas ; Köner, Marc ; Weissenborn, Sören ; Byfield, Marvin: The device driver engine-cloud enabled ubiquitous device integration. In: *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2015 IEEE Tenth International Conference on* IEEE (Veranst.), 2015, S. 1–7

[40] Kodali, Ravi K. ; Mahesh, Kopulwar S.: Low cost ambient monitoring using ESP8266. In: *Contemporary Computing and Informatics (IC3I), 2016 2nd International Conference on* IEEE (Veranst.), 2016, S. 779–782

[41] Lee, Shinho ; Kim, Hyeonwoo ; Hong, Dong-kweon ; Ju, Hongtaek: Correlation analysis of MQTT loss and delay according to QoS level. In: *Information Networking (ICOIN), 2013 International Conference on* IEEE (Veranst.), 2013, S. 714–717

[42] Lesjak, Christian ; Hein, Daniel ; Hofmann, Michael ; Maritsch, Martin ; Aldrian, Andreas ; Priller, Peter ; Ebner, Thomas ; Ruprechter, Thomas ; Pregartner, Günther: Securing smart maintenance services: Hardware-security and TLS for MQTT. In: *Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on* IEEE (Veranst.), 2015, S. 1243–1250

[43] Light, Roger A.: Mosquitto: server and client implementation of the MQTT protocol. In: *The Journal of Open Source Software* 2 (2017), Nr. 13, S. 265

[44] Lin, Hsien-Tang: Implementing smart homes with open source solutions. In: *International Journal of Smart Home* 7 (2013), Nr. 4, S. 289–295

[45] Magnoni, L: Modern messaging for distributed sytems. In: *Journal of Physics: Conference Series* Bd. 608 IOP Publishing (Veranst.), 2015, S. 012038

[46] Maksimović, Mirjana ; Vujović, Vladimir ; Davidović, Nikola ; Milošević, Vladimir ; Perišić, Branko: Raspberry Pi as Internet of things hardware: performances and constraints. In: *design issues* 3 (2014), S. 8

[47] McArthur, Stephen D. ; Davidson, Euan M. ; Catterson, Victoria M. ; Dimeas, Aris L. ; Hatziargyriou, Nikos D. ; Ponci, Ferdinanda ; Funabashi, Toshihisa: Multi-agent systems for power engineering applications—Part I: Concepts, approaches, and technical challenges. In: *IEEE Transactions on Power systems* 22 (2007), Nr. 4, S. 1743–1752

[48] Miorandi, Daniele ; Sicari, Sabrina ; De Pellegrini, Francesco ; Chlamtac, Imrich: Internet of things: Vision, applications and research challenges. In: *Ad hoc networks* 10 (2012), Nr. 7, S. 1497–1516

[49] Mo, Yilin ; Kim, Tiffany Hyun-Jin ; Brancik, Kenneth ; Dickinson, Dona ; Lee, Heejo ; Perrig, Adrian ; Sinopoli, Bruno: Cyber–physical security of a smart grid infrastructure. In: *Proceedings of the IEEE* 100 (2012), Nr. 1, S. 195–209

[50] Moravcevic, Vladimir ; Tucic, Milan ; Pavlovic, Roman ; Majdak, Aleksandar: An approach for uniform representation and control of ZigBee devices in home automation software. In: *Consumer Electronics-Berlin (ICCE-Berlin), 2015 IEEE 5th International Conference on* IEEE (Veranst.), 2015, S. 237–239

[51] Myers, Colton: *Learning saltstack*. Packt Publishing Ltd, 2016

[52] Mynzhasova, Aida ; Radojicic, Carna ; Heinz, Christopher ; Kölsch, Johannes ; Grimm, Christoph ; Rico, Juan ; Dickerson, Keith ; García-Castro, Raúl ; Oravec, Victor: Drivers, standards and platforms for the IoT: Towards a digital VICINITY. In: *Intelligent Systems Conference (IntelliSys), 2017* IEEE (Veranst.), 2017, S. 170–176

[53] Naik, Nitin: Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In: *2017 IEEE international systems engineering symposium (ISSE)* IEEE (Veranst.), 2017, S. 1–7

[54] Niazi, Muaz ; Hussain, Amir: Agent-based computing from multi-agent systems to agent-based models: a visual survey. In: *Scientometrics* 89 (2011), Nr. 2, S. 479

[55] Nieminen, Johanna ; Gomez, Carles ; Isomaki, Markus ; Savolainen, Teemu ; Patil, Basavaraj ; Shelby, Zach ; Xi, Minjun ; Oller, Joaquim: Networking solutions for connecting bluetooth low energy enabled machines to the internet of things. In: *IEEE network* 28 (2014), Nr. 6, S. 83–90

[56] Nugroho, Andri P. ; Okayasu, Takashi ; Horimoto, Masafumi ; Arita, Daisaku ; Hoshi, Takehiko ; Kurosaki, Hidehito ; Yasuba, Ken-ichiro ; Inoue, Eiji ; Hirai, Yasumaru ; Mitsuoka, Muneshi u. a.: Development of a field environmental monitoring node with over the air update function. In: *Agricultural Information Research* 25 (2016), Nr. 3, S. 86–95

[57] Poslad, Stefan ; Buckle, Phil ; Hadingham, Rob: The FIPA-OS agent platform: Open source for open standards. In: *proceedings of the 5th international conference and exhibition on the practical application of intelligent agents and multi-agents* Bd. 355, 2000, S. 368

[58] Rajput, Pruthvish ; Chaturvedi, Manish ; Patel, Pankesh: Advanced urban public transportation system for Indian scenarios. In: *Proceedings of the 20th International Conference on Distributed Computing and Networking* ACM (Veranst.), 2019, S. 327–336

[59] Ramparany, Fano ; Marquez, Fermin G. ; Soriano, Javier ; Elsaleh, Tarek: Handling smart environment devices, data and services at the semantic level with the FI-WARE core platform. In: *2014 IEEE International Conference on Big Data (Big Data)* IEEE (Veranst.), 2014, S. 14–20

[60] Razzaque, Mohammad A. ; Milojevic-Jevric, Marija ; Palade, Andrei ; Clarke, Siobhán: Middleware for Internet of Things: A survey. In: *IEEE Internet of Things Journal* 3 (2016), Nr. 1, S. 70–95

[61] Renner, Thomas ; Kliem, Andreas ; Kao, Odej: The device cloud-applying cloud computing concepts to the internet of things. In: *Ubiquitous Intelligence and Computing, 2014 IEEE 11th Intl Conf on and IEEE 11th Intl Conf on and Autonomic and Trusted Computing, and IEEE 14th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UTC-ATC-ScalCom)* IEEE (Veranst.), 2014, S. 396–401

[62] Römer, Kay ; Domnitcheva, Svetlana: Smart playing cards: A ubiquitous computing game. In: *Personal and Ubiquitous Computing* 6 (2002), Nr. 5-6, S. 371–377

[63] Saint-Andre, Peter: Extensible messaging and presence protocol (XMPP): Core. 2011. – Forschungsbericht

[64] Shang, Wentao ; Yu, Yingdi ; Droms, Ralph ; Zhang, Lixia: Challenges in IoT networking via TCP/IP architecture. In: *Technical Report NDN-0038. NDN Project* (2016)

[65] Singh, Kiran J. ; Kapoor, Divneet S.: Create Your Own Internet of Things: A survey of IoT platforms. In: *IEEE Consumer Electronics Magazine* 6 (2017), Nr. 2, S. 57–68

[66] Soni, Dipa ; Makwana, Ashwin: A survey on MQTT: a protocol of Internet of Things (IoT). In: *Proceeding of the International Conference on Telecommunication, Power Analysis and Computing Techniques, Chennai: IN*, 2017

[67] Sowitzki, Alexander M.: *Integration von Complex Event Processing in ein Multiagentensystem für Smart Environments.* 2016. – URL https://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/bachelor/sowitzki.pdf. – Zugriffsdatum: 2019-01-02

[68] Sowitzki, Alexander M.: Cyber Physical Systems. (2017). – URL https://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2017-gsem/Sowitzki/bericht.pdf. – Zugriffsdatum: 2018-11-05

[69] Sowitzki, Alexander M.: Anforderungen an eine Cyber Physical Systems Abstraktionsschicht am Beispiel von Smart Environments. (2018). – URL https://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2018-hsem/sowitzki/bericht.pdf. – Zugriffsdatum: 2018-11-05

[70] Sowitzki, Alexander M.: Eine Infrastruktur für Cyber Physical Systems. (2018). – URL https://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2018-hsem/sowitzki/bericht.pdf. – Zugriffsdatum: 2018-11-05

[71] Sowitzki, Alexander M.: Umsetzung eines smarten Puppenhauses. (2018). – URL https://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2018-proj/sowitzki2.pdf. – Zugriffsdatum: 2018-11-05

[72] Streitz, Norbert A. ; Rocker, Carsten ; Prante, Thorsten ; Van Alphen, Daniel ; Stenzel, Richard ; Magerkurth, Carsten: Designing smart artifacts for smart environments. In: *Computer* 38 (2005), Nr. 3, S. 41–49

[73] Subash, Ashok: Iotivity–connecting things in iot. In: *TIZEN Development Summit* (2015)

[74] Tanenbaum, Andrew S.: *Modern operating system*. Pearson Education, Inc, 2009

[75] Tantitharanukul, Nasi ; Osathanunkul, Kitisak ; Hantrakul, Kittikorn ; Pramokchon, Part ; Khoenkaw, Paween: Mqtt-topic naming criteria of open data for smart cities. In: *Computer Science and Engineering Conference (ICSEC), 2016 International* IEEE (Veranst.), 2016, S. 1–6

[76] Toulson, Rob ; Wilmshurst, Tim: *Fast and effective embedded systems design: applying the ARM mbed*. Newnes, 2016

[77] Van Deursen, Arie ; Klint, Paul ; Visser, Joost: Domain-specific languages: An annotated bibliography. In: *ACM Sigplan Notices* 35 (2000), Nr. 6, S. 26–36

[78] Vujovic, Vladimir ; Maksimovic, Mirjana: Raspberry Pi as a Wireless Sensor node: Performances and constraints. In: *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2014 37th International Convention on* IEEE (Veranst.), 2014, S. 1013–1018

[79] Vujović, Vladimir ; Maksimović, Mirjana: Raspberry Pi as a Sensor Web node for home automation. In: *Computers & Electrical Engineering* 44 (2015), S. 153–171

[80] Wang, Lihui ; Törngren, Martin ; Onori, Mauro: Current status and advancement of cyber-physical systems in manufacturing. In: *Journal of Manufacturing Systems* 37 (2015), S. 517–527

[81] Wu, Jiankun ; Huang, Linpeng ; Wang, Dejun ; Shen, Fei: R-OSGi-based architecture of distributed smart home system. In: *IEEE Transactions on Consumer Electronics* 54 (2008), Nr. 3

[82] Xia, Feng ; Yang, Laurence T. ; Wang, Lizhe ; Vinel, Alexey: Internet of things. In: *International Journal of Communication Systems* 25 (2012), Nr. 9, S. 1101–1102

[83] Zhu, Qian ; Wang, Ruicong ; Chen, Qi ; Liu, Yan ; Qin, Weijun: Iot gateway: Bridgingwireless sensor networks into internet of things. In: *Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on* Ieee (Veranst.), 2010, S. 347–352

[84] Ziegler, Sébastien ; Crettaz, Cedric ; Ladid, Latif ; Krco, Srdjan ; Pokric, Boris ; Skarmeta, Antonio F. ; Jara, Antonio ; Kastner, Wolfgang ; Jung, Markus: Iot6–moving to an ipv6-based future iot. In: *The Future Internet Assembly* Springer (Veranst.), 2013, S. 161–172

Glossar

Agent „Ein Agent ist ein Computersystem, das sich in einer bestimmten Umgebung befindet und welches fähig ist, eigenständige Aktionen in dieser Umgebung durchzuführen, um seine (vorgegebenen) Ziele zu erreichen.“[3].

Aktor Ein Smart Object welches sein Umfeld beeinflussen kann, bspw. Rollosteuerung.

AMQP Ein Zugriffsprotokoll für nachrichten-basierte Middlewares.

Bluetooth Low Energy Eine Untergattung des Bluetooth Standards, welche auf geringe Energienutzung optimiert ist.

CEP Behandelt die Analyse, Erkennung, Gruppierung und Verarbeitung von zusammenhängenden Ereignissen.

CEP-Engine Ein Programm, welches für die Durchführung von CEP zuständig ist.

CoAP Ein auf Hypertext Transfer Protocol (HTTP) basierendes Protokoll zum übertragen kleinster Datenmengen zwischen Embeddedsystemen und anderen.

CPS Die Kombination von Geräten mit Elektronische Datenverarbeitung (EDV) Komponenten und der Zusammenführung dieser in ein Netzwerk.

CSV Eine Liste bestehend aus einem oder mehreren Einträgen, die durch ein definiertes Zeichen weiter unterteilt sind..

DMA Der Zugriff auf Peripheriegeräte über direkte Anbindung im Hauptspeicher..

DNS Ein für die Auflösung von Domänennamen und Auffindung von Ressourcen zuständige Dienstgruppe.

Docker Eine Virtualisierungslösung, welche vorgefertigte Container nutzt, um diverse Vorteile für Administratoren zu erzielen.

DSL Eine Programmiersprache, die auf einen bestimmten Anwendungsfall konkretisiert ist.

EDV *Elektronische Datenverarbeitung*, das Bearbeiten und Erfassen von Daten durch computergestützte Systeme.

Ethernet Eine weitläufig genutzte Technologie zur kabelgebunden Übertragung von Daten.

GPIO Ein generalisierter I/O Port an einem Computersystem.

HTTP *Hypertext Transfer Protocol*, ein zustandsloses Protokoll zur Übertragung von annotierten Daten. Kernbestandteil des heutigen Internets.

I/O Die Kommunikation zwischen einem Computersystem und der Umgebung.

I2C Ein Busprotokoll für die Ansteuerung von Teilkomponenten innerhalb einer Hardwarebaugruppe. Es werden insgesamt zwei Adern zum Austausch benötigt, Komponenten werden über eine einzigartige Adresse angesprochen. Die Kommunikation findet im Halbduplex-Bebtrieb statt.

IoT Die Vernetzung von Smart Objects mit dem Internet zur Steigerung ihrer Funktionalität.

JSON Ein leichtgewichtiges Datenformat, welches von Menschen leicht lesbar und von Maschinen leicht verarbeitbar sein soll.

Message Broker Ein Programm, dessen Aufgabe es ist, Nachrichten zwischen verschiedenen Programmen über unterschiedliche Protokolle auszutauschen. Eine Unterart der Middleware.

Middleware Ein universelles Verbindungsprogramm zwischen verschiedenen Anwendungen.

MIME-Type Ein, auf einer Zeichenkette basierender, Klassifikator für übertragbare Ressourcen.

MQTT Protokoll zur Übertragung von Nachrichten von und zu einem Publish/Subscribe-Verzeichnis.

Multiagentensystem Eine Umgebung, welche aus mehreren Agenten besteht, die kooperativ ein gemeinsames Problem lösen.

PCB Eine gedruckte Leiterplatine..

Publish/Subscribe Ein Softwarepattern für den Nachrichtenaustausch zwischen mehreren Teilnehmern. Nachrichten werden von Quellen nicht direkt an Senken gesendet, sondern in bestimmten Kanälen eines Dienstes veröffentlicht. Teilnehmer, die entsprechende Nachrichten erhalten sollen, abonnieren entsprechende Kanäle.

Publish/Subscribe-Verzeichnis Die Datenhaltung, in welcher Publish/Subscribe Systeme Daten hierarchisch ablegen können.

QoS Das Angebot einer Funktion mit garantierten Parametern im Bereich der Dienstgüte..

Regex Eine Beschreibungssprache für Zeichenketten zur Aufstellung von syntaktischen Regeln.

REST Ein Paradigma zum Austausch von Informationen zwischen Webservices.

RPC Eine Prozedur zum Aufrufen von Funktionen eines Programms durch andere Programme auf dem selben Rechner oder im Netzwerk.

Serial Eine Schnittstelle zum Austausch von Daten zwischen elektronischen Systemen.

Smart City Eine vernetzte Stadt, die durch Sensoren gewonnen Informationen nutzt, um ihre Lebensqualität zu verbessern..

Smart Environment Eine Umgebung mit vernetzten automatisierbaren Geräten und Computern als Ubiquitous Computing.

Smart Home Ein Haushalt mit Smart Environment Fähigkeiten.

Smart Object Ein Objekt, welches die Fähigkeit besitzt mit anderen Smart-Objekts zu kommunizieren und mit seiner Umgebung zu interagieren.

SPI Ein Busprotokoll für die Ansteuerung von Teilkomponenten innerhalb einer Hardwarebaugruppe. Es werden drei Adern für den Fullduplex-Betrieb benötigt sowie eine für jedes Gerät je eine Ader zum Selektieren.

System Call Ein programmatischer Weg für ein Programm, Dienste vom Betriebssystemkernel anzufordern.

TLS Ein Verschlüsselungsprotokoll zur Datenübertragung, weitläufig eingesetzt im Internet.

Ubiquitous Computing Der Einsatz von Computern in allen Gegenständen im Gegensatz zum traditionellen Ansatz, Computer als separate Geräte anzusehen.

UDP *User Datagram Protocol*, ein unverlässliches Transportprotokoll zum Übermitteln von Daten in einem Netzwerk.

XML Eine Beschreibungssprache für hierarchisch zusammengesetzte Daten.

XMPP Ein offenes Kommunikationsprotokoll, welches für Chats eingesetzt wird, aber durch Erweiterungen auch für andere Anwendungsfälle werden kann..

YAML Eine robuste Datenbeschreibungssprache, die eine Obermenge zu JSON darstellt.

ZigBee Eine Spezifikation für den energieeffizienten Austausch von Daten mit geringem Volumen..

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Ort	Datum	Unterschrift im Original