Universität Regensburg

Entwicklung und Evaluation einer End-User-Programming-Schnittstelle für Sensornetzwerke im Rahmen des Projekts FUSION

Masterarbeit im Fach Medieninformatik am Institut für Information und Medien, Sprache und Kultur (I:IMSK)

Vorgelegt von: Andreas Schmid, B.A.
Adresse: (für Publikation entfernt)
E-Mail (Universität): Andreas.Schmid@ur.de
E-Mail (privat): (für Publikation entfernt)
Matrikelnummer: (für Publikation entfernt)
Erstgutachter: Dr. Raphael Wimmer
Zweitgutachter: Prof. Dr. Niels Henze
Betreuer: Dr. Raphael Wimmer, Andreas Gschossmann
Laufendes Semester: Sommersemester 2019
Abgegeben am: 02.05.2019

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Zusammenfassung

Sensornetzwerke sind ein wichtiges Werkzeug um Daten über die Umwelt zu sammeln, jedoch müssen sie oft an einen bestimmten Anwendungsfall angepasst werden, wofür fortgeschrittene Kenntnisse in der Programmierung vonnöten sind. Da Sensornetzwerke häufig in der Ökologie, der Geologie oder für die Heimautomatisierung eingesetzt werden, haben viele ihrer AnwenderInnen nicht die dafür benötigten Programmierkenntnisse. Aus diesem Grund wurde in dieser Arbeit ein Sensornetzwerk mit einer dazugehörigen Programmierschnittstelle erstellt, die sich durch starke Abstrahierung von technischer Komplexität speziell an AnwenderInnen mit geringen Programmierkenntnissen richtet. Anforderungen an die Schnittstelle wurden durch eine ausführliche Literaturrecherche und einer Fokusgruppenstudie erhoben. Nach der Implementierung wurde die Benutzbarkeit der Programmierschnittstelle in einer Nutzerstudie evaluiert und mit einem ähnlichen System verglichen. Das Sensornetzwerk wurde in Feldexperimenten auf seine praxistauglichkeit untersucht.

Abstract

Sensor networks are an important tool for acquiring data about the environment. Creating and configuring such networks, as it is often necessary in practical use, requires advanced programming skills. Because sensor networks are often used in ecology, geology or home automatation, many of their users do not have sufficient programming knowledge to do so. Therefore, a programming interface for sensor networks, explicitly aimed at users with modest programming skills by abstraction of technical complex components, has been created as part of this thesis. A thorough literature review, as well as a user study with focus groups have been conducted to establish requirements for this system. After the implementation, the usability of the programming interface has been evaluated and compared to a similar system in another user study. Furthermore, a technical evaluation was conducted to determine the practical suitability of the sensor network affiliated with the programming interface.

1 Einleitung

Durch ein breites Informationsangebot im Internet und der Möglichkeit, eigene Projekte über soziale Medien zu präsentieren, erfährt die seit den 1950er Jahren bestehende Do-it-yourself-Bewegung momentan eine Renaissance (Dougherty, 2012). Immer kleinere und kostengünstigere Bauteile (Moore, 1965), Open-Source-Software, sowie Technologien wie der 3D-Druck, ermöglichen die Umsetzung von privaten Hardwareprojekten. Bauanleitungen und fertige Projekte werden im Internet mit gleichgesinnten geteilt[1][2] und auf Messen[3] präsentiert. Die daraus hervorgegangene *Maker-Bewegung* (Dougherty, 2012) erfreut sich seit dem Jahr 2011 immer größerer Beliebtheit (Hartmann & Mietzner, 2017) und weltweit entstehen offene Werkstätten[4], die einen Ort zum gemeinsamen Umsetzen von Projekten bieten.

Mit der vierten industriellen Revolution (*Was ist Industrie 4.0?*, o. J.) und dem Beginn der Ära des *Ubiquitous Computing* (Weiser, 1991; Greenfield, 2010) spielt die Vernetzung von Geräten, zum Beispiel in Form von Sensornetzwerken, eine immer größere Rolle. Auch in der Maker-Szene rücken vernetzte Systeme, beispielsweise für die Heimautomatisierung, immer mehr in den Fokus.

Da es sich bei Sensornetzwerken um komplexe Systeme handelt, die auf verschiedene Hardwarekomponenten, sowie drahtloser Kommunikation untereinander und häufig einer Anbindung an das Internet aufbauen, gestaltet sich auch die Entwicklung von Software für solche Systeme als schwierig. Das dafür benötigte Fachwissen geht dabei über die durchschnittlichen Kenntnisse von Autodidakten, die einen Großteil der Anhänger der Maker-Szene stellen, hinaus. So müssen Programme für Sensorknoten, die meist über wenig Rechenleistung verfügen und über einen Akku mit Strom versorgt werden, sehr effizient und robust geschrieben werden, um eine lange Betriebsdauer und geringe Fehleranfälligkeit zu gewährleisten. Angeschlossene elektronische Komponenten werden oft über digitale Protokolle angesteuert, die fortgeschrittene Konzepte der Programmierung wie manuelle Speicherverwaltung erfordern. Auch für die Kommunikation zwischen den Knoten eines Sensornetzwerks sind technisch komplexe Aufgaben wie Serialisierung und

[1] https://www.thingiverse.com/
[2] https://makezine.com/
[3] https://makerfaire.com/
[4] https://www.fablabs.io/

Deserialisierung von Daten, Synchronisation zwischen mehreren Geräten, die Verwendung von Sockets und das Aufrechterhalten der Verbindung nötig.

In einigen Naturwissenschaften wie der Geologie und der Ökologie werden Sensornetzwerke bereits heute eigesetzt, um Messdaten zu erheben (Hart & Martinez, 2006). Durch die Anbindung des Netzwerks an das Internet kann global auf diese Daten zugegriffen werden. Oft wird proprietäre Software, wie die visuelle Programmierumgebung *LabVIEW*[5]verwendet, um Sensoren auszulesen und Messdaten zu verarbeiten. Solche Software ist jedoch häufig teuer[6] und nicht quelloffen. Dadurch wird die Konfiguration und Erweiterung des Systems, wie es beispielsweise beim Einsatz selbst entwickelter Hardware nötig ist, erschwert.

Ein weiteres Einsatzgebiet von Sensornetzwerken ist die Heimautomatisierung (Hussain et al., 2009), bei der Geräte in der Wohnung auf Basis von Messdaten an- und ausgeschaltet werden. Auch das Auslesen von Daten und das Steuern von Geräten über das Internet oder einen Sprachassistenten ist bei Smart-Home-Anwendungen üblich. Existierende kommerzielle Lösungen beschränken sich in ihrem Funktionsumfang oft auf einfache Geräte wie Lampen, Lüftung und Heizung. Außerdem ist es Herstellern möglich, Daten der Geräte einzusehen (Day et al., 2019), was im Fall von Systemen zu Heimautomatisierung einen gravierenden Eingriff in die Privatsphäre von AnwenderInnen darstellt.

Bei NutzerInnen von Sensornetzwerken in beiden eben genannten Anwendungsfällen kann von einer gewissen technischen Affinität ausgegangen werden. Allerdings ist tieferes Wissen im Bereich der Programmierung, der Softwareentwicklung und der Elektronik bei diesen Nutzergruppen oft nicht vorhanden.

In dieser Arbeit soll deshalb eine einfach zu bedienende Plattform geschaffen werden, mit der EndnutzerInnen selbst Projekte auf Basis von Sensornetzwerken umsetzen können. Dazu sollen keine fortgeschrittene Konzepte der Programmierung und komplexes technisches Hintergrundwissen vonnöten sein. Die Plattform soll offen aufgebaut sein und mehr Flexibilität bieten als proprietäre Lösungen. Außerdem sollen mit der Plattform erstellte Sensornetzwerke autark betrieben werden können, um Unabhängigkeit von den Diensten Dritter zu gewährleisten.

[5]https://www.ni.com/de-de/shop/labview.html
[6]http://www.ni.com/de-de/shop/labview/select-edition.html

2 Ziele der Arbeit

In dieser Arbeit soll durch einen nutzerzentrierten Ansatz eine Programmierschnittstelle für Sensornetzwerke entwickelt werden, die auch für NutzerInnen mit wenig Programmier- und Elektronikerfahrung zugänglich ist.

Die Bedürfnisse und Ziele von NutzerInnen bei der Entwicklung von Projekten für das Internet der Dinge (*Internet of Things*, IoT) stellen einen wichtigen Aspekt für die Umsetzung der Programmierschnittstelle dar (Krajewski, 2017). Deshalb sollen diese erhoben, zusammengefasst und präsentiert werden.

Im Rahmen einer Nutzerstudie soll die Programmierschnittstelle evaluiert und mit ähnlichen Projekten verglichen werden. Die Ergebnisse dieser Studie sollen zusammengefasst und ausgewertet werden, um Probleme der Schnittstelle aufzudecken und Wünsche der NutzerInnen zu erkennen, sodass ein Leitfaden für die über diese Arbeit hinausgehende Weiterentwicklung des Systems erstellt werden kann.

Auf Basis der in Anforderungserhebung und Evaluation aggregierten Daten und der im Verlauf der Entwicklung gesammelten Erfahrungen sollen allgemeine Designempfehlungen für den Aufbau von Programmierschnittstellen - insbesondere im Bereich der Sensorik und des *Internet of Things* - erstellt und präsentiert werden.

Diese in dieser Arbeit entwickelte Programmierschnittstelle ist Teil des vom Sensorik-Applikationszentrum[7] der *OTH Regensburg* durchgeführten *FUSION*-Projekts[8]. Im Rahmen dieses Projekts soll ein Open-Source-Framework zum Erstellen modularer Sensorplattformen entstehen, das zahlreiche Hardwareplattformen unterstützt und Machine-Learning, sowie Sensorfusion integriert.

[7] https://www.sappz.de/
[8] http://www.fusion-project.io/

3 Vorgehen

Zu Beginn der Arbeit wurde eine umfassende Literaturrecherche durchgeführt, mit deren Hilfe ein Überblick über die Thematik des *End User Programming* und des Internet der Dinge geschaffen wurde.

Durch eine Nutzerstudie mit Fokusgruppen sollten die Anforderungen potentieller Endnutzer erhoben werden. Dabei waren vor allem die Bedürfnisse und Ziele der NutzerInnen bei der Entwicklung von IoT-Projekten, sowie deren Vorwissen im Bereich der Programmierung, Elektronik und Sensorik interessant. Diese Studie wird in Kapitel 5 näher beschrieben.

Neben den Anforderungen der NutzerInnen sind auch technische Aspekte und Designentscheidungen bei der Entwicklung der Programmierschnittstelle relevant. IoT-Projekte bestehen aus zahlreichen Komponenten, da mehrere Geräte mit einem bestimmten Protokoll über eine Drahtlosschnittstelle miteinander kommunizieren.

Aufgrund dieser komplexen Anforderungen muss im Rahmen der Durchführung von IoT-Projekten entschieden werden, welche Hardware, Programmiersprache, Kommunikationsprotokolle und Schnittstellen verwendet werden. Die im Rahmen dieser Arbeit verwendeten Komponenten wurden auf Basis von Fachlitertur, etablierten Paradigmen und verwandten Projekten definiert.

Aus den auf diese Weise erhobenen Anforderungen ergab sich ein Grundgerüst für den Aufbau der Programmierschnittstelle, auf Basis dessen eine Referenzimplementierung eines Sensornetzwerks und einer dazu gehörigen API umgesetzt wurde.

Nach der Fertigstellung und Veröffentlichung der Programmierschnittstelle wurde in einer Nutzerstudie mit sieben ProbandInnen aus der Zielgruppe die Bedienbarkeit der dieser API evaluiert und mit einem ähnlichen System verglichen (siehe Kapitel 9.1). Dabei wurden Möglichkeiten zur Verbesserung der API qualitativ erhoben, auf Basis derer die Programmierschnittstelle weiterentwickelt werden kann.

Darüber hinaus wurden die im Laufe dieser Arbeit gesammelten Erkenntnisse zum Design von Programmierschnittstellen zusammengefasst (siehe Kapitel 10.1).

4 Related Work

In diesem Kapitel wird zunächst einschlägige Literatur zum Internet der Dinge, Sensornetzwerken, sowie End User Programming vorgestellt. Anschließend werden existierende Projekte präsentiert, die End User Programming für Sensornetzwerke ermöglichen und deren Vor- und Nachteile diskutiert. Aufbauend auf diese Literaturrecherche und die in Kapitel 5 beschriebene Vorstudie werden Anforderungen an die in dieser Arbeit entwickelte Programmierschnittstelle für Sensornetzwerke definiert, die in Kapitel 6 aufgeführt sind.

4.1 Internet of Things

Das bereits 1991 von Mark Weiser in seinem Aufsatz "The Computer for the 21st Century" (Weiser, 1991) antizipierte und 1999 von Kevin Ashton (Ashton, 2009) benannte Konzept des *Internet of Things* (IoT) bezeichnet eine Infrastruktur, bei der Objekte der physischen Welt miteinander vernetzt sind und Daten über sich selbst und ihre Umgebung erheben und versenden können. Dies stellt einen gravierenden Paradigmenwechsel dar, der von Ashton in seiner Wichtigkeit mit dem der Einführung des Internets gleichgesetzt wird. Ashton begründet die Wichtigkeit eines Internets der Dinge mit der Fähigkeit von Computern, selbstständig Daten über die physische Welt zu sammeln, da Menschen, von denen ein Großteil des momentanen Inhalts des Internets stammt, dazu aufgrund von begrenzter Zeit und Präzision nur eingeschränkt in der Lage sind. Das automatisierte Sammeln und Interpretieren von Daten über die Umwelt führt zu optimierten Prozessen und einer daraus resultierenden Reduzierung von Kosten und Abfall:

> «If we had computers that knew everything there was to know about things—using data they gathered without any help from us—we would be able to track and count everything, and greatly reduce waste, loss and cost. We would know when things needed replacing, repairing or recalling, and whether they were fresh or past their best.»
>
> (Ashton, 2009)

Mattern & Flörkemeier (2010) führen diesen Gedanken weiter aus und legen drei

wesentliche Bereiche fest, die von einem Internet der Dinge profitieren würden:

Kommerziell Logistische Prozesse können durch detaillierte Daten besser automatisiert und dadurch optimiert werden.

Sozial und Politisch Verbesserte Lebensqualität durch Assistenzsysteme und intelligente Infrastruktur.

Persönlich Produkte und Dienste für Unterhaltung, Sicherheit und Unterstützung im persönlichen Alltag.

Konkret können über Sensoren aktuelle Daten zum Zustand eines Systems erhoben werden. Diese können verwendet werden, um bestehende Prozesse zu optimieren, oder besser auf kritische Situationen reagieren zu können. Im allgemeinen sehen Mattern & Flörkemeier (2010) Sensoren als einen grundlegenden Bestandteil des Internet der Dinge:

> «Using sensors, they are able to perceive their context, and via built-in networking capabilities they would be able to communicate with each other, access Internet services and interact with people.»
>
> (Mattern & Flörkemeier, 2010, S. 2)

Frühe Ansätze zum Internet der Dinge, beispielsweise von Weiser (1991), basieren auf Technologien wie Barcodes oder *RFID*, die auf räumliche Nähe von Objekten zueinander angewiesen sind. Mattern & Flörkemeier (2010), sowie Chong & Kumar (2003), beschreiben hingegen bereits Sensornetzwerke, deren Knoten durch Funkprotokolle wie *WLAN* und *ZigBee*[9] über große Distanzen miteinander kommunizieren. Solche Netzwerke werden unter anderem in Naturwissenschaften wie der Geologie und der Biologie verwendet. Beispiele dafür zeigen Hart & Martinez (2006) in einem Überblick über verschiedene *Environmental Sensor Networks* im praktischen Einsatz. Während bereits 1939 die erste automatische Wetterstation entwickelt wurde (Diamond & Hinman, 1940), funktionierten Umweltsensoren meist passiv und mussten manuell ausgelesen werden. Ein Defekt wurde dadurch erst

[9] https://www.zigbee.org/zigbee-for-developers/zigbee-pro/

beim nächsten Auslesen bemerkt, wodurch Daten verloren gingen. Durch Sensornetzwerke mit Anbindung an das Internet können Messdaten von Sensoren an abgelegenen Orten global und in Echtzeit ausgelesen werden. Sind die Sensorknoten mit einer autarken Stromversorgung wie einer Solarzelle ausgestattet, so verringert sich der Aufwand der Instandhaltung auf ein Minimum: Die Sensorknoten müssen einmalig vor Ort platziert und nur im Falle eines Defekts wieder aufgesucht werden. Dies vereinfacht den Einsatz von Sensoren an entlegenen oder gefährlichen Orten. Aufgrund dessen und der Tatsache, dass durch Echtzeitdaten früh vor eventuellen Katastrophen gewarnt werden kann, bezeichnen Hart & Martinez (2006) Sensornetzwerke als ein wichtiges Werkzeug für Umweltwissenschaftler:

> «Environmental Sensor Networks will become a standard research tool for future Earth System and Environmental Science.»
>
> (Hart & Martinez, 2006, S. 1)

Während Hart und Martinez Sensornetzwerke für den wissenschaftlichen Einsatz vorstellen und Mattern und Flörkemeier sich vor allem auf industrielle Prozesse konzentrieren, ist auch die Heimautomatisierung für private AnwenderInnen ein Teilbereich des Internet der Dinge. Dadurch entstehen neue Herausforderungen für das Design der Nutzerschnittstellen des Systems, da von NutzerInnen mit wenig technischem Verständnis auszugehen ist. Krajewski (2017) gibt in ihrem Artikel Empfehlungen für nutzerzentriertes Design im Bereich des Internet der Dinge. Da die Interaktion mit Artefakten des IoT oft nicht über eine konkrete Nutzerschnittstelle stattfindet, sondern auf einem Zusammenspiel aus Sensorik und Algorithmen basiert, sollte beim *User Experience Design* für IoT-Anwendungen die Rolle eines Objekts in einer größeren Prozesskette berücksichtigt werden. Durch das individuelle Zusammenstellen verschiedener *Smart-Home-* und IoT-Produkte sind NutzerInnen selbst in der Lage, Einfluss auf ihre Nutzererfahrung zu nehmen.

> «[Designers] are no longer responsible for a particular experience, predefined by design. Instead they should equip users with tools that turn them into makers of their own individual experiences.»
>
> (Krajewski, 2017)

Krajewski (2017) bezeichnet *Smart Home* als momentan noch zu teuer für die Allgemeinheit, allerdings sind bereits zahlreiche kommerzielle Produkte wie "smarte" Lampen[10], Sprachassistenten[11] oder vollständige *Smart-Home*-Systeme verfügbar.

Am Ende ihres Artikels warnt Krajewski (2017) vor dem Missbrauch von IoT-Geräten durch die Industrie, um Daten über NutzerInnen zu sammeln. Dieser Aspekt wurde auch schon von Mattern & Flörkemeier (2010) behandelt, die von einem Abwägen zwischen Sicherheit und Freiheit, sowie Komfort und Privatsphäre sprechen.

Dass Hersteller von IoT-Geräten tatsächlich Daten ihrer NutzerInnen in Form von Sprachaufnahmen erheben und auswerten, wird von Day et al. (2019) beschrieben. So lässt die Firma Amazon MitarbeiterInnen Tonspuren transkribieren, die mit dem Sprachassistentensystem "Amazon Alexa" aufgezeichnet wurden. Diese enthalten teilweise sensible Inhalte wie Gespräche von Kindern oder Kontodaten.

NutzerInnen haben durch diverse quelloffene Produkte und Projekte (Brown, 2016) die Möglichkeit, IoT- und *Smart-Home*-Systeme zu verwenden, ohne kommerziellen Unternehmen Zugriff auf ihre persönlichen Daten zu gewähren. Diese erlauben auch die Anpassung und Erweiterung der Systeme nach eigenen Bedürfnissen. Nach Krajewski (2017) ist diese Anpassbarkeit auch ein Zeichen für gutes *User Experience Design*:

> «[Designers] facilitate the environment to enable users to autonomously create their own individual experiences.»
>
> (Krajewski, 2017)

4.2 End User Programming

Unter *End User Programming* oder *End User Development* versteht man die Konfiguration und Erweiterung bestehender Softwareartefakte durch deren NutzerInnen. Durch gezielte Modifikationen können Programme so an spezifische Anwendungsfälle angepasst werden, die von generischer Software nicht abgedeckt werden. Ko et al. (2011) heben in ihrer Definition den persönlichen Nutzen dieser Änderungen hervor:

[10] https://www2.meethue.com/de-de
[11] https://www.amazon.com/Amazon-Echo-And-Alexa-Devices/b?ie=UTF8&node=9818047011

> «[End User Programming is] programming to achieve the result of a program primarily for personal, rather public use.»
>
> (Ko et al., 2011, S. 4)

Dies steht im Gegensatz zu professioneller Softwareentwicklung, bei der die EntwicklerInnen von Programmen nicht zwingend auch deren AnwenderInnen sind.

Das Gebiet des End User Programming umfasst neben der von Nardi (1993) untersuchten Verwendung von Spreadsheets auch Skriptsprachen, *Low-Code*-Plattformen, Visuelle Programmierung und Programmierparadigmen wie *Natural Programming* und *Programming by Example* (Lieberman et al., 2006). Dabei werden selten vollständige Programme erzeugt, sondern der Funktionsumfang bestehender Programme erweitert oder Arbeitsabläufe dieser automatisiert. Dies kann über dedizierte Skriptsprachen (z.B. *MatLab*[12]), visuelle Programmierumgebungen (z.B. *LabVIEW*[13]), einer Kombination von beidem in Form von Spreadsheets (z.B. *Excel*[14]) oder der Verarbeitung von Anweisungen in natürlicher Sprache (*Natural Programming*, z.B. *Wolfram Alpha*[15]) umgesetzt sein.

Anforderungen an ein Softwareartefakt können vor der Implementierung nicht vollständig erhoben werden und können sich auch während der Benutzung ändern, da AnwenderInnen, und damit auch Bedürfnisse, sich weiterentwickeln (Costabile et al., 2003). Dadurch, dass NutzerInnen nicht auf Softwareentwickler angewiesen sind, wenn sie Funktionalität benötigen, welche existierende Software nicht bereitstellt, können durch End User Programming Zeit und Kosten eingespart werden. Durch die Möglichkeit, bestehende Systeme anzupassen, entfällt für NutzerInnen darüber hinaus das Erlernen neuer Software.

Für die Entwicklung von Software für sehr spezielle Anwendungsgebiete, wie beispielsweise in den Naturwissenschaften, kann domänenspezifisches Fachwissen vonnöten sein. Hier ist es oft sinnvoll, End-User-Programming-Plattformen wie zum Beispiel visuelle Programmierumgebungen einzusetzen, sodass NutzerInnen mit fachlichem Hintergrundwissen ihre Anwendungen selbst entwicklen können (Mayhew, 1992; Paternò, 1999).

[12] https://www.mathworks.com/products/matlab.html
[13] https://www.ni.com/de-de/shop/labview.html
[14] https://products.office.com/en/excel
[15] https://www.wolframalpha.com/

Obwohl der eigene Nutzen beim End User Programming im Vordergrund steht, können fertige Programme und gewonnenes Wissen trotzdem mit der Öffentlichkeit geteilt werden (Costabile et al., 2006; Pipek & Kahler, 2006). Diese Art der Softwareentwicklung bildet eine Grundlage für die Open-Source-Bewegung und brachte viele wichtige Softwareartefakte wie den Linux-Kernel (*The mind behind Linux | Linus Torvalds*, 2016, 2:35 bis 3:05) und die Versionsverwaltungssoftware *git*[16] (*Tech Talk: Linus Torvalds on git - YouTube*, 2007) hervor.

Um End User Programming zu unterstützen, sollte ein System einfach zu verstehen, zu lernen, zu lehren und zu testen sein (Lieberman et al., 2006). Eine flache Lernkurve spielt dabei eine entscheidende Rolle (Dertouzos, 1998; MacLean et al., 1990; Wulf & Golombek, 2001), da NutzerInnen des Systems meist keine Erfahrung in der Softwareentwicklung haben.

Damit ein einfacher Einstieg in ein System gewährleistet werden kann, ohne erfahrene AnwenderInnen durch einen begrenzten Funktionsumfang einzuschränken, empfiehlt sich ein modularer Aufbau mit mehreren Stufen der Komplexität (Henderson & Kyng, 1992; Mørch, 1995; Stiemerling, 2000). Dies wird beispielsweise bei *Arduino* durch stark abstrahierte, quelloffene Programmbibliotheken realisiert. Weniger komplexe Projekte können alleine durch Kontrollstrukturen und einfache Funktionsaufrufe umgesetzt werden, während erfahrene NutzerInnen auch Zugriff auf technisch anspruchsvollere Funktionalität wie manuelle Speicherverwaltung und Zeiger haben, oder sogar eigene Programmbibliotheken schreiben können. Die visuelle Programmierumgebung *Node-RED* stellt Module für die wichtigsten Funktionen bereit, neue Module aus der Community können aber über einen Paketmanager eingebunden werden. Eigene Module für *Node-RED* können mit JavaScript implementiert werden.

Da es sich bei der Programmierung um ein vielschichtiges und komplexes Aufgabengebiet handelt (Brooks, 1986), ist es bei der Umsetzung einer End-User-Programming-Schnittstelle besonders wichtig, das Vorwissen, die Ziele und die Bedürfnisse der zukünftigen NutzerInnen zu kennen und zu berücksichtigen.

> «There are only two ways to [...] bridge the gap between goals and sys-

[16] https://git-scm.com/

> tem: move the system closer to the user; [or] move the user closer to the system»
>
> (Norman & Draper, 1986, S. 43)

Laut Smith et al. (1996) funktioniert es im Bereich der Programmierung jedoch selten, NutzerInnen dazu zu bringen, sich an ein System anzupassen, weshalb nur die Möglichkeit bleibt, Systeme nutzerzentriert zu gestalten und so aufzubauen, dass sie dem mentalen Modell der NutzerInnen möglichst ähnlich sind.

Dies kann durch eine ausführliche Anforderungserhebung und einen iterativen Entwicklungsprozess mit regelmäßiger Evaluation des Systems erreicht werden.

Myers & Stylos (2016) heben die Wichtigkeit der Benutzbarkeit von Programmierschnittstellen hervor. So arbeiten AnwenderInnen von gut bedienbaren APIs effektiver und lösen Aufgaben schneller. Klassen sollten die Funktionalität enthalten, die man von ihnen erwartet und Funktionen sollten verständliche Namen haben.

Nach Blackwell (2006) sind Paradigmen nach dem Ansatz des *Natural Programming* für unerfahrene ProgrammiererInnen einfacher zu erlernen. Unter Natural Programming versteht man eine Programmiersprache, deren Syntax sich an natürlichen Sprachen orientiert (Miller, 1981).

> «The intention is that, if people naturally describe computational concepts in a certain way, even before they have ever seen a programming language, then that form of description will be the most appropriate for an end-user programming language.»
>
> (Blackwell, 2006, S. 10)

Den an den Standard für nutzerzentriertes Design (*ISO 9241-210*, 2010) angelehnten Designprozess für *Natural-Programming*-Systeme legen Pane & Myers (2006) folgendermaßen fest:

- Identifikation der Zielgruppe und deren Fachgebiet
- Verstehen der Zielgruppe, ihrer Probleme und ihres momentanen Arbeitsablaufs
- Entwurf eines neuen Systems, aufbauend auf diese Erkenntnisse

- Evaluierung des Systems, um neue Probleme zu identifizieren
- Überarbeitung des Systems wenn nötig

Da die Struktur eines Sensornetzwerks stark vom individuellen Anwendungsfall und der verwendeten Hardware abhängt, müssen Softwareschnittstellen für Sensornetzwerke offen und flexibel aufgebaut sein. Daher bietet sich der Ansatz des End User Programming an, durch den NutzerInnen nicht auf externe SoftwareentwicklerInnen angewiesen sind, um neue, möglicherweise selbstgebaute Hardware an das System anzubinden, sondern dies selbst erledigen können. Durch End User Programming können darüber hinaus ohne größeren Aufwand Veränderungen am System vorgenommen werden, wodurch ein iterativer Entwicklungsprozess ermöglicht wird. Die von Krajewski (2017) empfohlene Umgebung, in der NutzerInnen selbst ihre eigenen Erfahrungen kreieren können, wird durch die Flexibilität von End User Programming gewährleistet.

Bei Systemen, die von NutzerInnen durch End User Programming selbst erstellt wurden, ist der Weg der erhobenen Daten zudem transparent. Das Sammeln und Verarbeiten dieser Daten durch die Herstellerfirma oder Dritte, wie es bei kommerziellen, geschlossenen Systemen häufig der Fall ist, wird dadurch ausgeschlossen. Ein weiterer Vorteil selbst entwickelter Systeme ist die Unabhängigkeit von Diensten externer Anbieter. So wurde beispielsweise das *Smart-Home*-System *Revolv* von dessen Herstellerfirma im Jahr 2016 eingestellt (Price, 2016). Die von Servern des Herstellers abhängigen Geräte sind seitdem nicht mehr funktionsfähig.

Sensornetzwerke werden größtenteils von WissenschaftlerInnen für Forschungszwecke oder interessierten Laien für private Projekte verwendet, deshalb kann von einem Mindestmaß an technischem Vorwissen in der Zielgruppe ausgegangen werden.

4.2.1 Node-RED

Node-RED[17] ist eine browserbasierte Oberfläche zur graphischen Programmierung. Es verwendet die JavaScript-Laufzeitumgebung *Node.js*[18] und kann lokal auf einem

[17] https://nodered.org/
[18] https://nodejs.org/

PC, einem SoC-Computer oder in der Cloud ausgeführt werden. Diverse Protokolle und Plattformen werden von *Node-RED* standardmäßig unterstützt und eigene Module können durch in JavaScript geschriebene Plugins eingebunden werden.

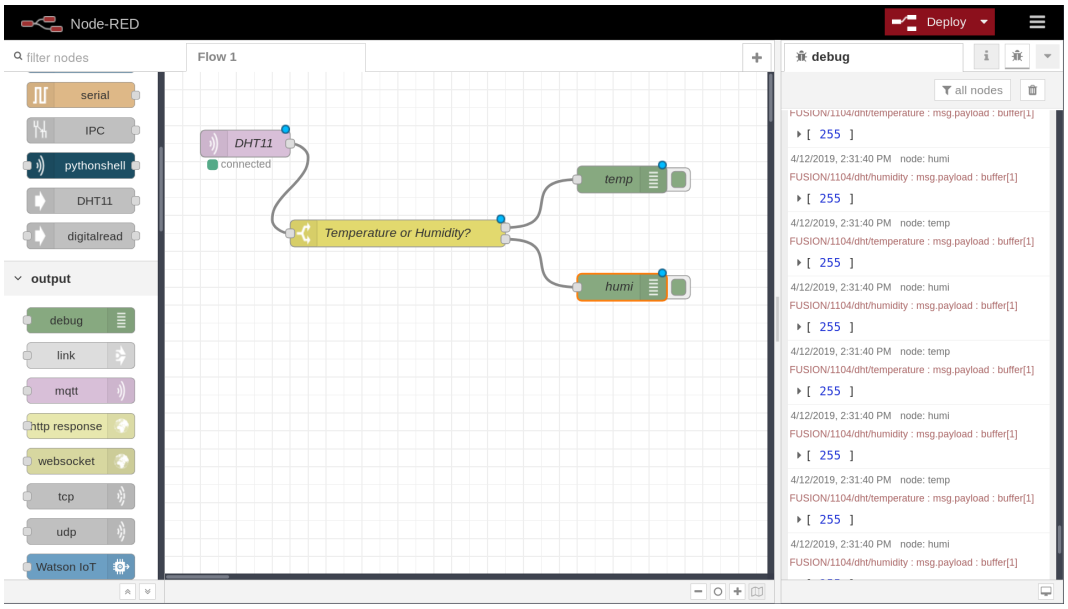

Abbildung 1: Benutzeroberfläche von Node-RED

Node-RED stellt verschiedene Bausteine (*Nodes*) mit Ein- und Ausgängen bereit, die per Drag&Drop auf eine Arbeitsfläche gezogen und miteinander verbunden werden können (siehe Abbildung 1). Die Daten einer Input-Node werden über deren Ausgang an damit verbundene Nodes weitergegeben, von diesen verarbeitet und schließlich über eine Output-Node ausgegeben.

4.2.2 Arduino

Die 2005 am Interaction Design Institute *Ivrea* als kostengünstige und offene Alternative zu kommerziellen Löungen entwickelte Mikrocontrollerplattform *Arduino* hat sich mittlerweile zu einem Standard für die DIY- und Makerszene entwickelt (Kushner, 2011; *Arduino Day 2019: Thank you 659 times!*, 2019).

Die Hardware eines *Arduino* besteht aus einer kleinen Platine (*Arduino Nano*: 4,3 cm * 1,8 cm, *Arduino Mega*: 10,1 cm * 5,3 cm (*List of Arduino boards and compatible systems*, 2019)), auf der sich ein Mikrocontroller und mehrere Input- und Output-Pins befinden. Es existieren mehrere verschiedene Versionen des *Arduino*, die sich in Größe, Rechenleistung, Anzahl der Pins und eingebauter Hardware unterscheiden (zwei Beispiele in Abbildung 2).

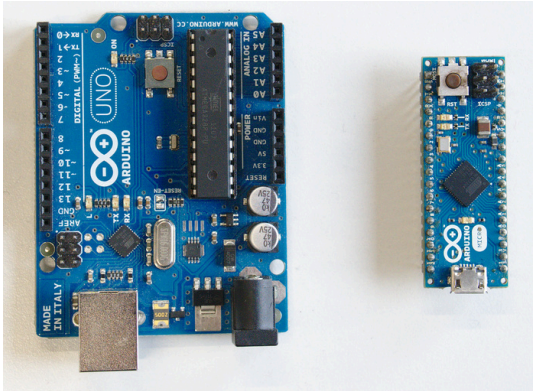

Abbildung 2: *Arduino Uno* (links) und *Arduino Micro* (rechts)

Mithilfe einer auf *Processing*[19] basierten Entwicklungsumgebung (*Arduino IDE*) können Programme für den *Arduino* in der Programmiersprache C++ entwickelt werden. Die IDE unterstützt auch diverse andere Mikrocontollerplattformen und bietet Funktionalität zum Installieren von Programmbibliotheken und einen seriellen Monitor zum Debugging.

```
1  // the setup function runs once when you press reset or power the board
2  void setup() {
3    // initialize digital pin LED_BUILTIN as an output.
4    pinMode(LED_BUILTIN, OUTPUT);
5  }
6
7  // the loop function runs over and over again forever
8  void loop() {
9    digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the
         voltage level)
10   delay(1000);                       // wait for a second
11   digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the
         voltage LOW
12   delay(1000);                       // wait for a second
13 }
```

Codebeispiel 1: *Arduino* "Blink Sample"

Programme für den *Arduino* basieren auf zwei Funktionen: `setup()` wird beim Start des Geräts einmalig ausgeführt, darin können beispielsweise angeschlossene Komponenten initialisiert werden. Danach läuft die `loop()`-Funktion in einer Schleife bis der *Arduino* neu gestartet wird. Diese Funktion enthält meist den Hauptbestandteil der Programmlogik. *Arduino* stellt eine Programmbibliothek zur Verfügung, die einfache Funktionen zum Ansteuern der *GPIO*-Pins und der Kommuni-

[19] https://processing.org/

kation über serielle Protokolle enthält. Das Codebeispiel 1 aus der offiziellen *Arduino*-Dokumentation[20] zeigt eine einfache Anwendung, die eine eingebaute LED des *Arduino* im Sekundentakt blinken lässt.

Aufgrund der geringen Kosten und der flachen Lernkurve ist *Arduino* sehr beliebt für die Umsetzung privater Elektronikprojekte.

[20]https://www.arduino.cc/en/tutorial/blink

5 Anforderungserhebung

Um die nötigen Anforderungen an eine Programmierschnittstelle für EndnutzerInnen zu erheben, wurde neben der Aufarbeitung einschlägiger Literatur eine Nutzerstudie durchgeführt. Zuerst wurde durch Experteninterviews ein grober Rahmen definiert, aus dem sich spezifischere Fragen für die zweite Iteration der Studie ergaben.

Diese bestand aus zwei Fokusgruppen mit potentiellen NutzerInnen der Programmierschnittstelle. Die TeilnehmerInnen an den Fokusgruppen wurden gezielt nach ihrer Vorerfahrung im Programmieren unterteilt, sodass sich eine Gruppe mit erfahrenen und eine Gruppe mit unerfahrenen TeilnehmerInnen zusammenfand. Mit dieser Unterteilung sollte erreicht werden, dass die ProbandInnen der jeweiligen Fokusgruppe durch ihr ähnliches Vorwissen auf einer Augenhöhe kommunizieren können. Dadurch wird auch das mögliche Problem vermieden, dass erfahrene TeilnehmerInnen die Diskussion dominieren und sich Menschen mit weniger Vorerfahrung eventuell dadurch eingeschüchtert fühlen. Darüber hinaus sollen die Aussagen von ProbandInnen mit verschiedenen großer Programmiererfahrung miteinander verglichen werden um mögliche von dieser abhängige Unterschiede feststellen zu können (Krueger, 2014, S. 5).

5.1 Fragestellung

Ziel der Vorstudie war es, herauszufinden wer die potentiellen NutzerInnen der im Rahmen dieser Arbeit entwickelten Programmierschnittstelle sind. Dazu gehören alle, die mit Sensornetzwerken arbeiten (wollen), aber nur eingeschränkte oder gar keine Programmierkenntnisse haben. Während einer Literaturrecherche konnten bereits zwei potentielle Nutzergruppen ermittelt werden. Zum einen werden in bestimmten Bereichen der Naturwissenschaften (zum Beispiel Ökologie und Geologie) Sensornetzwerke eingesetzt werden, um Messdaten zu erheben (Porter et al., 2005). Zum anderen existieren in der Maker- und Startup-Szene viele Projekte im Bereich der Heimautomatisierung und des Internet of Things (Giezeman, o. J.; Coleman, 2017), die ebenfalls Sensorik und drahtlose Kommunikation verwenden.

Neben dem Eingrenzen der Zielgruppe ist es wichtig, die von AnwenderInnen

gewünschten Verwendungszwecke der Programmierschnittstelle herauszufinden. Daraus lassen sich insbesondere technische Anforderungen an das darunter liegende Sensornetzwerk ableiten. Zudem muss der Aufbau der Programmierumgebung an die Komplexität der geplanten Projekte angepasst werden.

Des Weiteren sollte erhoben werden, wie viel Erfahrung diese potentiellen NutzerInnen bereits im Programmieren, der Softwareentwicklung und der Elektronik haben. Auch dies wirkt sich auf die Gestaltung der Programmierschnittstelle aus, da weniger erfahrene NutzerInnen stärker angeleitet werden müssen, während Anwender mit mehr Erfahrung vielleicht schon mit vielen Aspekten von Sensornetzwerken vertraut sind.

Letztlich galt es noch zu entscheiden, ob für die in dieser Arbeit erstellte Referenzimplementierung einer Programmierschnittstelle für Sensornetzwerke ein herkömmlicher textbasierter Ansatz zur Programmierung zum Einsatz kommen sollte, oder ob die NutzerInnen eine graphische Oberfläche bevorzugen.

5.2 Fokusgruppen

Wie bereits in Kapitel 5 beschrieben, wurden zwei Fokusgruppen durchgeführt, bei denen die TeilnehmerInnen mit viel Vorerfahrung im Programmieren in die erste (im Folgenden Gruppe A), und die TeilnehmerInnen mit wenig Vorerfahrung in die zweite Fokusgruppe (Gruppe B) eingeteilt wurden. Beide Fokusgruppen wurden am selben Ort - dem Besprechungsraum 1.101 in der *TechBase* - durchgeführt. Dieser Raum ist reizarm gestaltet und bietet eine neutrale Atmosphäre. Versuchsleiter und TeilnehmerInnen saßen gemeinsam an einem großen Besprechungstisch (siehe Abbildung 3).

Abbildung 3: Besprechungsraum 1.104 in der TechBase

Da während der Studie Bild und Ton aufgezeichnet wurden, mussten die ProbandInnen vor Beginn der Fokusgruppe einer Datenschutzerklärung zustimmen, bei der sie die Rechte am aufgezeichneten Material für Forschungszwecke freigaben (siehe Anhang A[21]).

Außerdem wurde ein kurzer Fragebogen ausgefüllt, mit dem demographische Daten und eine Selbsteinschätzung der eigenen Programmierfähigkeiten abgefragt wurden. Der *Dunning-Kruger-Effekt* (Kruger & Dunning, 1999) bewirkt, dass Personen mit wenig Fachwissen ihre Fähigkeiten auf einem Gebiet überschätzen. Um dies zu vermeiden, wurden die Programmierfähigkeiten nicht durch eine abstrakte Skala, sondern durch eine Liste konkreter Fertigkeiten in der Programmierung abgefragt:

keine Programmierkenntnisse

AnfängerInnen Beherrscht Programmablauf (Schleifen, Verzweigungen),Variablen, Ein- und Ausgabe

Fortgeschritten Kennt wichtige Datenstrukturen, kann einfache Algorithmen implementieren, beherrscht die Standardbibliothek mindestens einer Sprache

ExpertInnen gute Kenntnisse mehrerer Sprachen, kann sich neue Sprachen schnell aneignen, hält best practices ein

Darüber hinaus wurde erfragt, wie lange die TeilnehmerInnen bereits programmieren können (siehe Anhang A[22]).

Gruppe A setzte sich aus sechs TeilnehmerInnen zusammen, davon vier Studierende und zwei wissenschaftliche MitarbeiterInnen, jeweils aus dem Gebiet der Informatik. Die Vorerfahrung im Programmieren liegt im Bereich von drei bis 15 Jahren (Median 4, Durchschnitt 6,3). Zwei von ihnen bezeichnen sich selbst als fortgeschritten, während sich die restlichen vier als ExpertInnen einschätzen.

Gruppe B bestand aus sieben TeilnehmerInnen, darunter ein Studierender im Bereich der Informatik, zwei Studierende im Bereich der angewandten Wissenschaften, drei Ingenieure und ein Naturwissenschaftler. In dieser Gruppe lag die Vorer-

[21] Anforderungserhebung/einverständniserklärung_fokusgruppe_01.pdf
[22] Anforderungserhebung/fragebogen_fokusgruppe_01.pdf

fahrung im Programmieren im Bereich von 1,5 bis neun Jahren (Median 3, Durchschnitt 3,6). Vier TeilnehmerInnen sahen sich selbst als Anfänger im Programmieren, die anderen drei bezeichneten sich als fortgeschritten.

Die Fokusgruppen wurden durch den Versuchsleiter moderiert, welcher offene Fragen stellte, um das Rahmenthema der Studie einzuhalten. Dabei wurde Raum für Diskussion zwischen den ProbandInnen gelassen. Gegen Mitte der etwa neunzigminütigen Studie wurde den TeilnehmerInnen die Aufgabe gestellt, ein intelligentes, automatisiertes Gewächshaus zu entwerfen. Sie sollten den Aufbau des Gewächshauses skizzieren, nötige Sensoren und Aktoren darin platzieren und die dahinterstehende Logik auf die von ihnen bevorzugte Art und Weise ausdrücken. Nach dieser Aufgabe wurde im Plenum über die Ergebnisse diskutiert und es wurden einige knappe abschließende Fragen gestellt. Der Versuchsleiter griff gelegentlich in die Diskussion ein, falls die ProbandInnen zu weit vom Thema abkamen oder um vertiefende Fragen zu stellen.

Nachdem den ProbandInnen knapp Hintergrund und Ziel des *FUSION*-Projekts und insbesondere der in dieser Masterarbeit entwickelten Programmierschnittstelle dargelegt wurde, wurden folgende Fragen während der Fokusgruppe behandelt:

- Würden Sie eine Programmierschnittstelle verwenden, die es ermöglicht, Sensoren auszulesen, Daten zu verarbeiten und Aktionen auszuführen, ohne dass technisches Hintergrundwissen vorausgesetzt wird?

  Wenn ja, wofür? In welchem Kontext? Inwiefern würden Sie davon profitieren?

  Wenn nein, warum nicht? Was müsste daran verändert werden, damit Sie es dennoch in Erwägung ziehen?

- Geben Sie ein Beispiel für einen Anwendungsfall, bei dem Sie ein Sensornetzwerk einsetzen würden.

- Welche Features sind für Sie bei einer Programmierschnittstelle für Sensornetzwerke wichtig?

- Wäre es Ihnen wichtig, dass Sie gemeinsam mit anderen kollaborativ am selben Projekt arbeiten können?

– Welche Ängste oder Bedenken haben Sie bezüglich Sensornetzwerken, dem Internet of Things und einer Programmierschnittstelle für Endnutzer?

5.3 Ergebnisse der Vorstudie

Alle ProbandInnen der beiden Fokusgruppen waren am Projekt interessiert und gaben an, eine einfache Programmierschnittstelle für Sensornetzwerke im privaten Kontext für "Bastelprojekte" oder Smart Home verwenden zu wollen. Zwei Personen hielten die Verwendung einer solchen Schnittstelle mit dem dazugehörigen Sensornetz in der Forschung, beispielsweise in der Ökologie und der Medizin für denkbar. Die Anwendung im industriellen Sektor wurde von allen ProbandInnen angezweifelt. Begründet wurde dies dadurch, dass in der Industrie meist proprietäre Systeme verwendet werden, deren Hersteller bereits passende Messsysteme anbieten. NutzerInnen solcher Messsysteme beschrieben diese als "Blackbox", die manuell konfiguriert werden können und Fachwissen voraussetzen.

Die Frage, welche Features einer Programmierschnittstelle für Sensornetzwerke den ProbandInnen wichtig wären, führte zu folgendem Ergebnis:

Online-Daten Es sollte eine einfache Möglichkeit geben, Daten aus dem Internet abzufragen und in Anwendungen zu verwenden. Beispiele dafür wären Wetterdaten oder die aktuelle Uhrzeit.

Benachrichtigungen Anwendungen sollten in der Lage sein, Nachrichten über Messenger-Dienste oder E-Mail versenden zu können, um NutzerInnen über den aktuellen Zustand des Systems zu informieren.

Kamera Für viele Anwendungen wäre die Anbindung einer Kamera zur Übertragung eines Livebilds oder im Zusammenspiel mit Bildverarbeitungsalgorithmen wünschenswert.

Konfiguration Einige NutzerInnen hielten es für wichtig, Parameter über einfach zugängliche Konfigurationsdateien zu definieren. Im Idealfall sollte dies auch im laufenden Betrieb möglich sein. Auch das automatisierte Anpassen von Thresholds wurde von TeilnehmerInnen der Fokusgruppen angesprochen.

Speichern von Daten Das langfristige Speichern von Daten in Form von Logfiles

oder Datenbanken sollte laut TeilnehmerInnen der Fokusgruppen vom System unterstützt werden.

Visualisierung Die Schnittstelle sollte die Möglichkeit bieten, Daten zu visualisieren. Die Visualisierung soll je nach Art der Daten anpassbar sein.

Externer Zugriff Einige ProbandInnen wünschten sich die Möglichkeit, über ein Webinterface oder eine Smartphone-Applikation auf Anwendungen zuzugreifen und Daten einzusehen.

Aktuatorik Nicht nur das Auslesen von Sensordaten, sondern auch das Ansteuern von Aktuatoren sollte vom Sensornetzwerk und der dazugehörigen Programmierschnittstelle unterstützt werden.

Zusätzlich wurden von den TeilnehmerInnen der Fokusgruppen noch diverse Anforderungen gesammelt, welche für die Benutzbarkeit des Systems in der Praxis unabdingbar sind:

Niedrige Einstiegsschwelle Die Programmierschnittstelle soll auch für wenig erfahrene Programmierer benutzbar sein. Langwieriges Einlesen in die Dokumentation soll nicht nötig sein, bevor man mit der Benutzung beginnen kann. Für häufig vorkommende Anwendungsfälle sollen Codebeispiele vorliegen. Technische Komplexität wie Threading oder Exceptions sollen von der Schnittstelle gekapselt werden.

Abstraktionsebenen Durch die Kapselung von technisch anspruchsvollem Code kann es sein, dass erfahrene AnwenderInnen eingeschränkt werden. Deshalb soll es trotzdem möglich sein, komplexe Programmstrukturen zu verwenden. Dies könnte durch eine Aufteilung der Schnittstelle auf verschiedene Abstraktionsebenen umgesetzt werden.

Schnelles Prototyping Die Programmierschnittstelle soll die Möglichkeit bieten, simple Anwendungen schnell umsetzen zu können. Vor allem erfahrene ProgrammiererInnen kritisierten in den Fokusgruppen die Menge an "Boilerplate-Code", also benötigte Codestrukturen ohne Funktionalität (Lämmel & Peyton Jones, 2003), die in vielen etablierten APIs für simple Programme nötig ist.

Drahtlose Hardware Die Hardwareelemente, die ein Sensornetzwerk aufbauen, sollen Daten drahtlos senden und empfangen können. Außerdem sollen sie nicht auf eine externe Stromversorgung angewiesen sein.

Geringe Latenz Je nach Anwendungsfall kann es wichtig sein, Sensordaten in Echtzeit auzulesen. Deshalb sollte bei der Entwicklung des Systems auf geringe Latenzen geachtet werden.

Sicherheit Das Sensornetzwerk soll vor Fremdzugriff geschützt sein und Daten sollen anonym und verschlüsselt übertragen werden können. Außerdem soll es möglich sein, ein solches Netzwerk ohne Internetanbindung aufzubauen.

Die TeilnehmerInnen der Fokusgruppen waren sich einig, dass sie Projekte gerne im Team umsetzen, allerdings bezog sich diese Aussage eher auf handwerkliche Arbeit an Prototypen. Beim Programmieren arbeiten die ProbandInnen ausnahmslos lieber alleine, einige fänden es aber gut, wenn die Option der kollaborativen Zusammenarbeit trotzdem verfügbar ist. Darüber hinaus sollte sich bei der Umsetzung von Projekten mit der Programmierschnittstelle schnell ein funktionierender Prototyp erstellen lassen, der stabil läuft und wenig Wartung bedarf. Die nachträgliche Änderung von Parametern und das Hinzufügen von Features war den ProbandInnen wichtig. Die TeilnehmerInnen der Fokusgruppe mit wenig Programmiererfahrung legten größeren Wert auf gut kommentierte Codebeispiele als auf eine ausführliche Dokumentation der Schnittstelle, da sie ihre Programme gerne aus Codebausteinen zusammenfügen.

6 Komponenten

Auf Basis der in Kapitel 4 diskutierten Literatur und der in Kapitel 5 beschriebenen Vorstudie wurden Anforderungen an die zu implementierende Programmierschnittstelle und das dazugehörige Sensornetzwerk definiert.

Nicht alle Anforderungen waren im Rahmen dieser Arbeit - besonders im Hinblick auf die letztendlich verwendeten Bauteile und Frameworks - umsetzbar, beziehungsweise für einen Prototyp notwendig. Diese Aspekte werden später in Kapitel 10.2 diskutiert.

Für die Umsetzung wurde, wenn möglich, auf etablierte, offene und kostengünstige Systeme zurückgegriffen. Dies bietet den Vorteil, dass es für deren Komponenten oft bereits eine Community und dadurch eine gute Dokumentation und Support gibt. Zudem können sie bei Bedarf auch angepasst werden. Darüber hinaus soll die Hardware auch für private NutzerInnen erschwinglich sein, da interessierte Laien eine Zielgruppe des in dieser Arbeit entwickelten Systems darstellen und für ein Sensornetzwerk meist mehrere Hardwarekomponenten benötigt werden.

In diesem Kapitel wird beschrieben, welche Komponenten auf Basis der Anforderungserhebung für die Umsetzung des Sensornetzwerks und der Programmierstelle ausgewählt wurden. In Kapitel 7 wird auf Rolle dieser Komponenten im System eingegangen.

6.1 Modularität

Die Anforderungserhebung ergab, dass potentielle NutzerInnen gerne ein "Baukastensystem" mit einzelnen, in sich funktionalen, Komponenten verwenden möchten, die sie möglichst einfach miteinander verbinden können. Dies entspricht im Kern der Unix-Philosophie, die Doug McIlroy folgendermaßen zusammenfasst:

> «[...] Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.»
>
> (Doug McIlroy, nach Raymond (2003))

Auch in der objektorientierten Programmierung wird Wert auf die so genannte

Kapselung von Objekten und das Verbergen von Information gelegt. Dabei werden Objekte so gestaltet, dass sie nach Möglichkeit nur eine Aufgabe haben (*Single-Responsibility*-Prinzip) und die interne Funktionalität von außen nicht einsehbar oder veränderbar ist (*Encapsulation*, *Information Hiding*) (Meyer, 1988).

Der Aufbau eines Systems nach diesen Richtlinien entspricht den Prinzipien der "Modularität". Diese wird von Baldwin & Clark (2000) folgendermaßen beschrieben:

> «[...] modules are units in a larger system that are structurally independent of one another, but work together. The system as a whole must therefore provide a framework - an architecture - that allows for both independence of structure and integration of function.»
>
> Baldwin & Clark (2000)

Software und Hardware des in dieser Arbeit erstellten Sensornetzwerks und der dazugehörigen Programmierschnittstelle sollen nach diesen Prinzipien entworfen und umgesetzt werden.

6.2 API

Die Programmierschnittstelle (API) soll aufbauend auf Designvorschlägen aus relevanter Fachliteratur (Myers & Stylos, 2016; Bari et al., 2013; Krajewski, 2017) und in der Nutzerstudie erhobenen Anforderungen nach den im letzten Abschnitt beschriebenen Prinzipien der Modularität entwickelt werden. Im Mittelpunkt steht dabei eine einfache Benutzbarkeit, wodurch es auch Anfängern möglich sein soll, eigene Projekte umzusetzen.

6.2.1 Design

Ein wichtiger Bestandteil der API ist die Kapselung komplexer Funktionalität wie Speicherverwaltung, Parallelisierung, Exception Handling und Kommunikation über serielle Schnittstellen oder Netzwerkprotokolle. Im Idealfall sollen NutzerInnen davon nichts mitbekommen, da diese Funktionalität vom System automatisch im Hintergrund ausgeführt wird.

Die Schnittstelle soll so gestaltet sein, dass Projekte damit auf Basis von Funktionsaufrufen und einfacher Kontrollstrukturen umgesetzt werden können. Die Fokusgruppenstudie ergab, dass auch NutzerInnen mit wenig Programmiererfahrung das Konzept von Bedingungen und Schleifen verstehen. Viele von ihnen haben diese auch bei der praktischen Aufgabe während der Fokusgruppe aktiv eingesetzt.

6.2.2 Features

Die Schnittstelle soll die Möglichkeit bieten, bestehende Programmbibliotheken einzubinden und zu verwenden. Dies erweitertet den Funktionsumfang erheblich und hat den Vorteil, dass NutzerInnen ihnen bereits bekannte Komponenten benutzen können und dadurch weniger Neues erlernen müssen.

Aufbau und Funktionalität der Schnittstelle sollen ausführlich dokumentiert sein. Darüber hinaus ergab die Fokusgruppenstudie, dass unerfahrene NutzerInnen großen Wert auf mitgelieferte Codebeispiele legen, auf denen sie ihre Projekte aufbauen können.

Außerdem soll die Schnittstelle es ermöglichen, NutzerInnen über den Zustand des Systems zu benachrichtigen. In den Fokusgruppen wurde vorgeschlagen, diese Nachrichten per E-Mail, SMS oder Instant-Messenger-Dienste wie *WhatsApp* zu verschicken.

Insbesondere TeilnehmerInnen der Fokusgruppen mit großer Programmiererfahrung wünschten sich die Möglichkeit, gemessene Sensordaten durch Filter, Sensorfusion oder Machine-Learning-Algorithmen zu verarbeiten. In der Referenzimplementierung soll dies beispielhaft für einfache Filter umgesetzt werden. Dies hat den Grund, dass der Implementierungsaufwand für komplexe Algorithmen den Umfang dieser Arbeit sprengen würde und die Rechenleistung des als zentralen Gateway agierenden *Raspberry Pi* nicht ausreicht, solche Algorithmen in Echtzeit auszuführen.

Die Schnittstelle soll so aufgebaut sein, dass zur Konfiguration des Systems und der damit entwickelten Programme kein Eingriff in den Quellcode nötig ist, sondern Konfigurationsdateien dafür verwendet werden können. ProbandInnen der Fokusgruppen schlugen vor, dass diese Parameter sich auch während des laufenden Betriebs verändern lassen sollten. Diese Anforderung wurde jedoch in der Re-

ferenzimplementierung aus technischen Gründen nicht umgesetzt.

6.3 Hardware

Sensornetzwerke bestehen aus mehreren so genannten Sensorknoten (*Nodes*). Dies sind meist Mikrocontroller, die Sensoren auslesen und deren Messwerte an andere Knoten im Netzwerk und/oder einen zentralen Rechner - das so genannte Gateway - weiterleiten. Dieses sammelt und verarbeitet die empfangenen Daten und reagiert darauf, beispielsweise indem es sie visualisiert, Benachrichtigungen sendet oder Aktuatorik ansteuert.

Viele TeilnehmerInnen der Fokusgruppenstudie merkten an, dass es sinnvoll wäre, Aktuatoren auch als eigenständige Knoten im Netzwerk umzusetzen, sodass diese über das selbe Protokoll wie die Sensorknoten angesteuert werden können.

Die Kommunikation zwischen den Nodes soll drahtlos über eine angemessene Distanz erfolgen und es soll möglich sein, die Nodes ohne eine externe Stromversorgung zu betreiben.

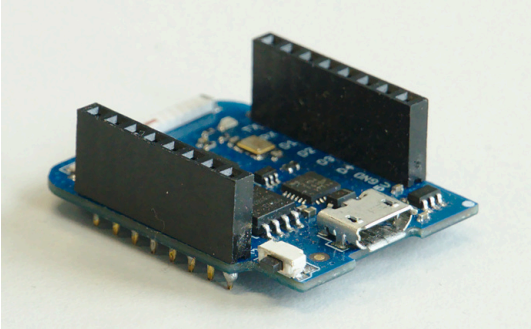

Abbildung 4: Wemos D1 mini

Der *Wemos D1 mini* (siehe Abbildung 4), eine sehr kostengünstige (4-7€) Mikrocontrollereinheit, erfüllt diese Anforderungen:

Er ist mit einem *WiFi*-Chip ausgestattet, dessen Reichweite mittels einer externen Antenne erhöht werden kann. Der *Wemos D1 mini* kann durch einen Akku mit Strom versorgt werden. Darüber hinaus kann er über die relativ einsteigerfreundliche *Arduino-IDE* programmiert werden und mit ihm verbundene Komponenten über GPIO-Pins ansteuern. Der Chip unterstützt diverse serielle Protokolle, enthält einen Analog-Digital-Wandler und kann über die Pins pulsbreitenmodulierte Si-

gnale erzeugen. Dadurch ist er mit einer breiten Auswahl an Sensorik und Aktuatorik kompatibel.

Deshalb werden alle Nodes des in dieser Arbeit umgesetzten Sensornetzwerks auf den *Wemos D1 mini* aufgebaut.

Als zentrales Gateway soll ein *Raspberry Pi*[23] verwendet werden. Dies ist ein *System-on-a-Chip*-Computer (SoC), auf dem die Linux-Distribution *Raspbian*[24] läuft. Er verfügt über mehr Rechenleistung als die als Nodes verwendeten Mikrocontroller. Auf dem Gateway soll die eigentliche Programmierschnittstelle laufen, deren Benutzeroberfläche über einen Webbrowser zugänglich ist.

6.4 Kommunikation

Die Kommunikation zwischen den Nodes und dem Gateway soll drahtlos erfolgen, um eine hohe Reichweite, große Flexibilität und geringen Aufwand zum Aufbau des Netzwerks zu gewährleisten.

In manchen Anwendungsfällen ist eine geringe Latenz bei der Kommunikation wichtig, vor allem wenn schnell auf ein Ereignis reagiert werden muss oder Daten in Echtzeit visualisiert werden sollen. Deshalb sollte ein leichtgewichtiges Nachrichtenprotokoll mit geringem Overhead für die Kommunikation gewählt werden.

Eine stabile Verbindung ist bei drahtloser Kommunikation - insbesondere bei der Anwendung im Freien - nicht immer gegeben. Aus diesem Grund muss das System in der Lage sein, bei Verlust der Verbindung entsprechend zu reagieren und versuchen, die Verbindung automatisch wieder aufzubauen.

Aufgrund der geringen Paketgröße und einem gezielt für instabile Verbindungen ausgelegten Aufbau, eignet sich das *MQTT*-Protokoll[25] sehr gut für den Einsatz im IoT-Bereich und wird deshalb als Nachrichtenprotokoll für das im Rahmen dieser Arbeit implementierte Sensornetzwerk verwendet.

Während *MQTT* auf dem Netzwerkprotokoll TCP/IP und damit WiFi basiert, ist es denkbar, auch andere Funkstandards zu unterstützen. So zeichnet sich insbesondere *LoRaWAN*[26] durch eine sehr hohe Reichweite und geringem Stromverbrauch aus. Deshalb soll die Schnittstelle so gestaltet werden, dass es ohne großen Auf-

[23] https://www.raspberrypi.org/
[24] https://www.raspbian.org/
[25] http://mqtt.org/
[26] https://www.thethingsnetwork.org/docs/lorawan/

wand möglich ist, bei Bedarf ein anderes Netzwerkprotokoll oder einen anderen Funkstandard zu verwenden.

6.5 Frontend

Für das User Interface der Programmierschnittstelle haben sich in der Vorstudie diverse Anforderungen ergeben. Dedizierte Entwicklungsumgebungen weisen häufig einen großen Funktionsumfang auf. Dies sorgt für eine steile Lernkurve und macht sie deshalb ungeeignet für NutzerInnen mit wenig Programmiererfahrung. Wie auch *Processing* oder die darauf aufbauende *Arduino-IDE* sollte das User Interface der Programmierschnittstelle auf die nötigsten Features reduziert sein.

TeilnehmerInnen der Fokusgruppenstudie war es wichtig, Messdaten zu visualisieren und extern auf die Schnittstelle zugreifen zu können. Diese beiden Punkte sprechen für eine browserbasierte Lösung.

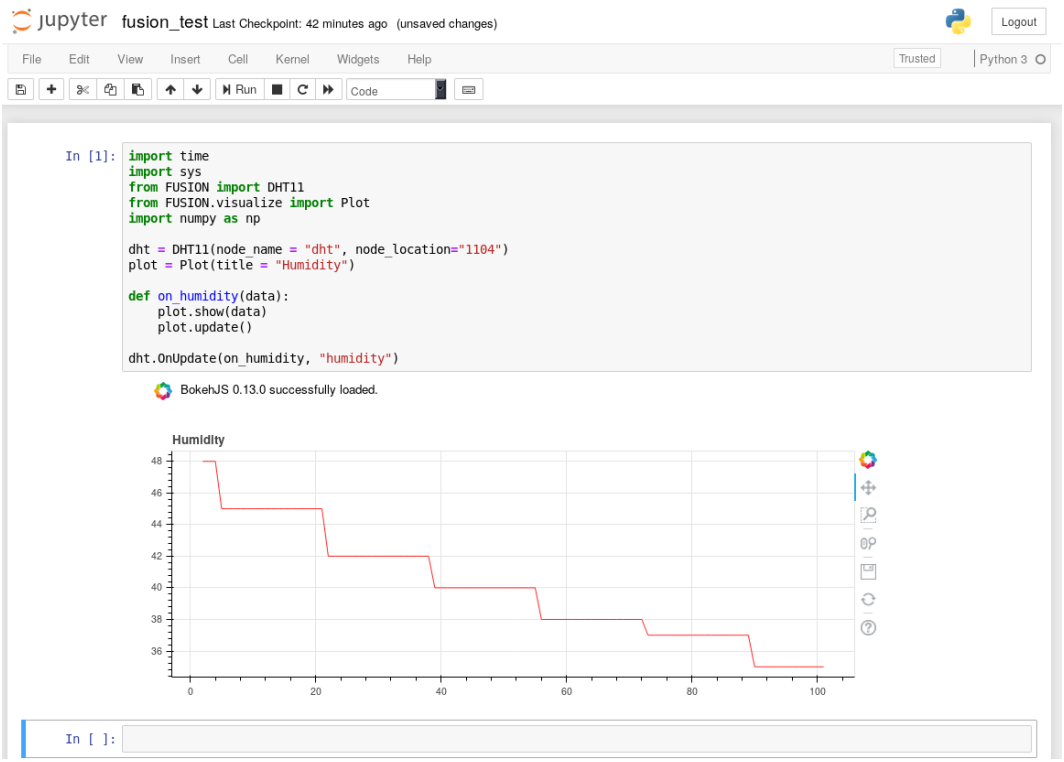

Abbildung 5: Benutzeroberfläche von Jupyter Notebooks

Die aus dem *IPython*-Projekt[27] entstandenen *Jupyter Notebooks*[28] stellen eine browserbasierte Programmierumgebung für verschiedene Programmiersprachen - unter

[27]https://ipython.org/
[28]https://jupyter.org/

anderem Python - dar. Die Notebooks werden auf einem Server gehostet und können im Browser über eine URL aufgerufen werden. Dadurch wird kollaboratives Arbeiten ermöglicht und es kann plattformunabhängig auf die Anwendung zugegriffen werden. Code kann abschnittsweise in so genannten Zellen geschrieben und ausgeführt werden und die Ausgabe findet im selben Browserfenster statt. Dies ermöglicht es NutzerInnen, explorativ zu arbeiten. Verschiedene Python-Pakete zur Datenverarbeitung und -visualisierung stehen zur Verfügung und interaktive Bedienelemente können eingebunden werden (siehe Abbildung 5).

Durch die plattformagnostische Natur einer browserbasierten Lösung, die Möglichkeiten zur Visualisierung und der sehr einsteigerfreundlichen Programmiersprache Python, eignen sich *Jupyter Notebooks* gut als graphische Oberfläche für die in dieser Arbeit umgesetzte Programmierschnittstelle für Sensornetzwerke.

6.5.1 Visuelle Programmierung: Vor- und Nachteile

Neben herkömmlichen, textbasierten Programmierumgebungen, existieren auch Ansätze der visuellen Programmierung. Diese basieren meist auf einem modularen System, bei dem einzelne funktionale Bausteine mit der Maus auf eine Arbeitsfläche gezogen und miteinander verbunden werden können. Visuelle Programmierumgebungen richten sich häufig an NutzerInnen mit wenig Programmiererfahrung, beispielsweise Kinder (*Scratch*[29]), Künstler (*Reaktor*[30], *Blender*[31]) oder Naturwissenschaftler und Ingenieure (*LabVIEW*[32], *Simulink*[33]).

Eine speziell für das Internet of Things ausgelegte visuelle Programmierumgebung ist das bereits im Kapitel 4.2.1 beschriebene *Node-RED*, welches das *MQTT*-Protokoll bereits unterstützt.

Visuelle Programmierumgebungen haben den Vorteil, dass der gesamte Funktionsumfang für NutzerInnen sichtbar und der Arbeitsablauf offensichtlich ist, wodurch die kognitive Anstrengung bei der Benutzung reduziert wird. Das Umsetzen größerer Projekte kann dadurch jedoch umständlich sein, da Module einzeln auf die Arbeitsfläche gezogen und mit anderen Modulen verbunden werden müssen.

[29] https://scratch.mit.edu/
[30] https://www.native-instruments.com/en/products/komplete/synths/reaktor-6/
[31] https://www.blender.org/
[32] https://www.ni.com/labview
[33] https://www.mathworks.com/products/simulink.html

Der Funktionsumfang ist zudem auf die verfügbaren Module begrenzt. Das Hinzufügen eigener Module ist oft zwar möglich, diese müssen aber wiederum "klassisch", also in einer textbasierten Programmiersprache implementiert werden. Darüber hinaus neigen visuelle Programmierumgebungen mit steigendem Funktionsumfang oft zur Unübersichtlichkeit. Whitley & Blackwell (1997) zeigt in einer vergleichenden Studie, dass verschiedene Nutzergruppen unterschiedliche Ansprüche an eine Programmiersprache haben. So seien beispielsweise Forscher mehr an den neuen Möglichkeiten bei der Verwendung einer neuen Programmiersprache interessiert, während professionelle Programmierer Werkzeuge bevorzugen, die sie bereits kennen. Ein Teilnehmer der Fokusgruppenstudie mit großer Programmiererfahrung gab an, dass eine gut entworfene und dokumentierte Programmbibliothek sich nicht zu stark von einem visuellen Ansatz unterscheide. Er berichtete zudem von schlechten Erfahrungen mit visuellen Programmierumgebungen und *Low-Code*-Plattformen und dass die erste für ihn zufriedenstellende Lösung das *Blueprint*-System der 2014 erschienenen (*Unreal Engine 4.0 Update Notes*, 2014) *Unreal Engine 4*[34] war. Auch Brooks (1986) kritisiert visuelle Programmierumgebungen in der Hinsicht, dass Flussdiagramme eine schlechte Darstellungsform von Programmcode sind, da Monitore zu klein und viele Programmstrukturen zu komplex seien, um sie visuell darzustellen.

Im Rahmen der Evaluation dieser Arbeit soll eine Nutzerstudie (siehe Kapitel 9.1.3) durchgeführt werden, in der das auf *Jupyter Notebooks* aufbauende, klassische Design der Programmierschnittstelle mit dem visuellen Ansatz über *Node-RED* verglichen wird. In dieser Studie soll mit Rücksicht auf die Vorerfahrung der ProbandInnen im Programmieren die Benutzbarkeit der beiden Oberflächen qualitativ und die Zeit, die NutzerInnen brauchen, um eine Aufgabe zu lösen (*Task Completion Time*) quantitativ erhoben werden.

6.6 Weggelassene Komponenten

Da es sich bei der im Rahmen dieser Arbeit implementierten Programmierschnittstelle und dem dazugehörigen Sensornetzwerk nur um Prototypen handelt, können nicht alle erhobenen Anforderungen umgesetzt werden. Insbesondere Aspekte, die

[34] https://www.unrealengine.com/

sich nicht auf die Benutzbarkeit der Programmierschnittstelle auswirken, großen Implementierungsaufwand darstellen oder für die die Rechenleistung der verwendeten Hardware nicht ausreicht, werden bei der Entwicklung nicht berücksichtigt.

So stellt beispielsweise die Sicherheit des Netzwerks gegenüber Fremdzugriff keinen wichtigen Aspekt der Schnittstelle im Hinblick auf die Benutzbarkeit dar. Trotz ihrer unbestrittenen Wichtigkeit für den Einsatz von Sensornetzwerken in der Praxis werden aus diesem Grund keine Sicherheitsheitsfeatures in der Referenzimplementierung umgesetzt.

Auch der Stromverbrauch der Sensorknoten im Akkubetrieb hat keinen Einfluss auf die Benutzbarkeit der Programmierschnittstelle an sich. Durch die Implementierung des vom Wemos D1 mini unterstützten *"Deep Sleep"*-Modus könnte die Akkulaufzeit stark erhöht werden (Foxworth, 2017), dieser Aspekt wird aber im Rahmen dieser Arbeit nicht umgesetzt.

Auf den von TeilnehmerInnen der Fokusgruppen vorgeschlagenen Einsatz von Sensorik als *Wearable Computer* (Mann, o. J.) soll bei der Umsetzung des Sensornetzwerks keine Rücksicht genommen werden, da dafür benötigte Features wie das lokale Zwischenspeichern von Sensordaten mit Zeitstempel auf den Nodes, solange keine Verbindung zum Gateway besteht, einen enormen Implementierungsaufwand darstellen würde.

Auch das Übertragen von Kamerabildern über das Sensornetzwerk und anschließende Bildverarbeitung auf dem Gateway wird von dem in dieser Arbeit entwickelten System nicht unterstützt, da die verwendete Hardware nicht über die dafür benötigte Rechenleistung verfügt und diese Operationen sich nicht weit genug abstrahieren lassen, als dass sie für NutzerInnen mit wenig Hintergrundwissen verständlich wären.

7 Implementierung

Um die im vorherigen Kapitel beschriebene Flexibilität zu gewährleisten, wurde das System modular in mehreren, voneinander unabhängigen Ebenen aufgebaut. Diese kommunizieren untereinander über standardisierte Protokolle und können so mit geringem Aufwand je nach Anwendungsfall ausgetauscht werden. Der modulare Aufbau dient darüber hinaus dazu, komplexe Funktionalität von unerfahrenen NutzerInnen abzukapseln, während erfahrene NutzerInnen die Module bei Bedarf ändern können.

Die verschiedenen Ebenen sind folgendermaßen aufgeteilt:

1. Nodes
2. Publisher-Subscriber-System
3. API
4. Frontend

Bei den Nodes handelt es sich um netzwerkfähige Mikrocontoller, die Sensoren beziehungsweise Aktuatoren ansteuern und drahtlos mit einem zentralen Gateway kommunizieren, auf dem die eigentliche Software des Sensornetzwerks läuft.

Als Nachrichtenprotokoll für die Kommunikation wird *MQTT* (*Message Queuing Telemetry Transport*) verwendet. Dabei handelt es sich um ein *Publisher-Subscriber-System*, das durch geringe Paketgrößen und Robustheit bei schlechter Verbindung für die Nutzung im IoT-Kontext optimiert ist (Seidel, 2013). *MQTT* verwendet das *TCP/IP*-Netzwerkprotokoll, ist allerdings auch (unter dem Namen *MQTT-SN*) für andere Netzwerkprotokolle verfügbar.

Auf einem zentralen Gateway läuft ein Dienst (*MQTT-Broker*), der die Kommunikation innerhalb eines Netzwerks moderiert. Sogenannte *Publisher* können Nachrichten mit einem bestimmten *Topic* versenden. Diese werden vom Broker an alle Knoten des Netzwerks weitergeleitet, die sich als *Subscriber* für dieses *Topic* registriert haben. Dadurch ist eine bidirektionale, asynchrone Kommunikation zwischen Nodes und Gateway möglich.

Andere Netzwerkprotokolle wie *UDP* oder *ZigBee* können an das System angebunden werden, indem auf dem Gateway ein im Hintergrund laufendes Programm

(*Daemon*) gestartet wird, welches Nachrichten über das jeweilige Protokoll empfängt und als *Publisher* an den *MQTT-Broker* weitergibt.

Auf Seite der API wird die Kommunikation von einer Basisklasse (`FUSION_MQTT`) übernommen, von der alle anderen Module erben. Durch diesen Ansatz wird technisch anspruchsvoller Code zur Kommunikation vollständig gekapselt und NutzerInnen der Programmierschnittstelle kommen mit diesem nicht in Kontakt. Zudem ist es möglich, das Nachrichtenprotokoll des Systems auszutauschen, indem die `FUSION_MQTT`-Klasse umgeschrieben wird.

Bei den bereits angesprochenen Modulen handelt es sich um virtuelle Repräsentationen von Nodes, die deren Funktionalität nach den Prinzipien der objektorientierten Programmierung modellieren. Es stehen sowohl Methoden zum Versenden von Daten, als auch Callback-Funktionen, die beim Empfangen neuer Daten ausgelöst werden, zur Verfügung. Datenpakete werden beim Empfangen deserialisiert und in eine Datenstruktur gespeichert.

Darüber hinaus gibt es diverse Module, die keine Nodes modellieren, sondern weitere Funktionalität bereitstellen. Beispiele dafür sind die Verarbeitung und Visualisierung von Daten und das Versenden von E-Mails. Die API kann durch das Hinzufügen weiterer Klassen um Funktionalität erweitert werden.

Da es sich bei der API um ein herkömmliches Python-Package handelt, kann von beliebigen Python-Skripts darauf zugegriffen werden. Daher ist es naheliegend, *Jupyter Notebooks* als Entwicklungsumgebung für die EndnutzerInnen bereitzustellen, da der Zugriff darauf über den Webbrowser erfolgt und die Ergebnisse von Programmen einfach visualisiert werden können.

Die benötigte Software für das Sensornetzwerk und die Programmierschnittstelle sind als Repository auf *GitHub*[35] verfügbar. Die Verzeichnisstruktur des Repository ist folgendermaßen aufgebaut:

additional_files/ Enthält einige Konfigurationsdateien und Utility-Skripte.

FUSION/ Enthält das Python-Paket, das den Hauptbestandteil der Programmierschnittstelle darstellt.

nodes/ Enthält Unterverzeichnisse für einzelne Nodes, in denen sich wiederum de-

[35] https://github.com/A-Schmid/FUSION_EUP

ren Programmcode befindet.

nodes/libraries/FUSION/src/ Enthält Programmbibliotheken für Sensornodes, die die Kommunikation mit Hardwarekomponenten übernehmen und im Programmcode der Nodes verwendet werden.

nodes/pio/ Dieses Verzeichnis wird vom später beschriebenen Upload-Skript (siehe 8.2) verwendet.

test_scripts/ Enthält Programmbeispiele, die mithilfe der Programmierschnittstelle erstellt wurden.

fusion.conf Eine globale Konfigurationsdatei, die von diversen Skripts ausgelesen wird und Parameter wie die Zugangsdaten für das Sensornetzwerk enthält.

pio_compile.sh Upload-Skript, das den Programmcode kompiliert und auf Nodes installiert. Dieses Skript wird im Kapitel 8.2 genauer beschrieben.

setup.sh Installationsskript, das beim Einrichten des Sensornetzwerks einmalig aufgerufen werden muss. Dieses Skript wird im Kapitel 8.1 genauer beschrieben.

7.1 Nodes

Jede Node besteht aus einem Mikrocontroller und einem Sensor oder Aktor, der vom Mikrocontoller angesteuert wird. Die Nodes kommunizieren drahtlos mit dem zentralen Gateway. Diese Funktionen übernehmen sie eigenständig; die Kommunikation mit dem Gateway findet lediglich durch kurze Datenpakete über das *MQTT*-Protokoll statt. Abhängig von der Aufgabe der Nodes werden diese grob in folgende Kategorien unterteilt:

Sensornodes Diese Nodes lesen den angeschlossenen Sensor in einem bestimmten Intervall aus und senden das Ergebnis an das Gateway.

Aktornodes Diese Nodes warten auf eine Anweisung vom Gateway, entsprechend derer sie den angeschlossenen Aktor ansteuern.

Two-Way-Nodes Diese Nodes können Nachrichten vom Gateway empfangen und Antworten senden. Ihr Aufbau ist verglichen mit den anderen beiden Kate-

gorien komplexer. Sie werden für spezifische Module verwendet, die auf den Dialog zwischen Gateway und Mikrocontoller angewiesen sind.

Die Implementierung der Nodes basiert auf der *Arduino*-Plattform, da sich diese mit über 850.000 aktiven Nutzern im Jahr 2018 (*Arduino Day 2019: Thank you 659 times!*, 2019) mittlerweile zum de-facto-Standard für private Hardwareprojekte entwickelt hat und deshalb bereits viele Komponenten durch Bibliotheken unterstüzt werden. Außerdem sind bereits viele der potentiellen NutzerInnen mit der Plattform vertraut. Die eigentliche Funktionalität der Nodes befindet sich in einer C++-Klasse, welche wiederum von einer Basisklasse (`FUSION_MODULE`) erbt. Die Funktionen zur Kommunikation mit dem Gateway liegen gekapselt in einer eigenen Klasse vor, wodurch das Nachrichtenprotokoll bei Bedarf ohne großen Aufwand ausgetauscht werden kann.

Im `.ino`-File der Node wird das entsprechende Modul importiert, eine Instanz der Klasse angelegt und diverse Parameter festgelegt. In der `loop()`-Funktion des `.ino`-File kann die Funktionalität der Klasse aufgerufen werden. Die geschieht bei weniger komplexen Nodes über einen Aufruf von deren Memberfunktion `update()`.

Der Programmablauf wird am Beispiel eines Sensorknotens mit einem Button im Codebeispiel 2 und in Abbildung 6 dargestellt.

```
1  #include "FUSION_Button.h"
2
3  // use pin defined by compile parameters, if it is not defined, use the
       default pin of wemos d1 mini button shields
4  #ifndef PIN
5      #define PIN 0
6  #endif
7
8  uint8_t buttonPin = PIN;
9  FusionButton button(buttonPin);
10
11 void setup(){
12     button.initialize();
13 }
14
15 void loop(){
16     button.update();
17     delay(update_time);
18 }
```

Codebeispiel 2: Aufbau einer simplen Node

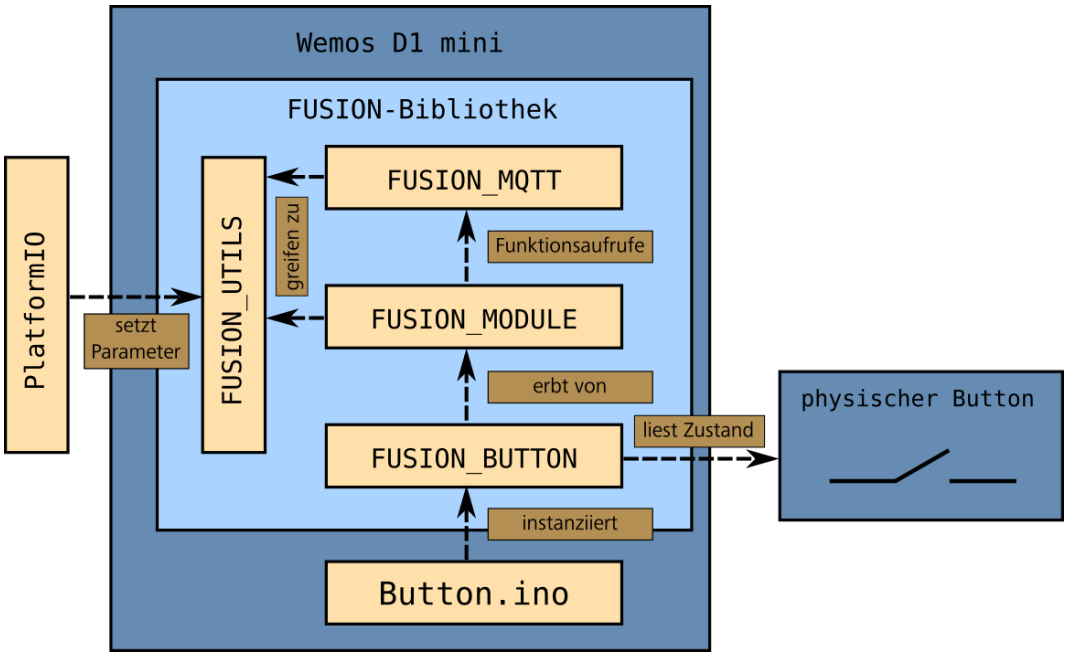

Abbildung 6: Programmablauf der Nodes am Beispiel eines Buttons

Das Kompilieren und Hochladen der Software für die Mikrocontroller der Nodes erfolgt über ein Wrapper-Skript, das auf das Kommandozeileninterface von *PlatformIO*[36] zugreift. Über dieses Skript können dem Programm auch vordefinierte Parameter wie zum Beispiel das *MQTT*-Topic oder die Wartezeit zwischen Messungen des Sensors festgelegt werden. Durch die Verwendung von *PlatformIO* wird eine Vielzahl von Mikrocontrollerplattformen unterstützt.

Erfahrene NutzerInnen können eigene Nodes implementieren, indem sie Module auf Basis der `FUSION_MODULE`-Klasse und ein dazugehöriges `.ino`-File schreiben.

Diese Klasse enthält die überladene Funktion `sendData`, welche wiederum eine Funktion des ausgewählten Nachrichtenprotokolls aufruft, die Daten an das Gateway sendet. Da die Kommunikation auf diese Weise gekapselt ist, kann das verwendete Protokoll bei Bedarf ohne großen Aufwand ausgetauscht werden.

7.1.1 ESP8266

Die Nodes der Referenzimplementierung des Sensornetzwerks basieren auf dem *Wemos D1 mini*[37] (siehe Abbildung 4). Dabei handelt es sich um eine Plattform für den *ESP8266*, einem WiFi-Chip mit eingebautem Mikrocontoller.

[36] https://platformio.org/
[37] https://wiki.wemos.cc/products:d1:d1_mini

Der *Wemos D1 mini* verfügt über 11 digitale *GPIO*-Pins, einen Analog-Digital-Wandler und unterstützt die seriellen Protokolle *I²C*, *SPI*, *UART* (*D1 mini [WEMOS Electronics]*, 2018). Der Aufbau des *Wemos D1 mini* ermöglicht die Verbindung zu kompatiblen Modulen über ein Shield-System, bei dem die Komponenten zusammengesteckt werden können, ohne dass sie verlötet werden müssen.

Durch das Anbringen einer externen Antenne kann die Reichweite des WiFi-Chips erhöht werden.

Der *ESP8266* ist kompatibel mit der *Arduino*-Plattform. Dies ermöglicht die Verwendung von bereits existierenden Programmbibliotheken und erleichtert den Umgang für viele NutzerInnen, da für sie das Hochladen der Software auf den Chip einen bereits bekannten Prozess darstellt.

7.1.2 Unterstützte Sensoren und Aktoren

Um das Sensornetzwerk und die dazugehörige Programmierschnittstelle zu testen, wurden Module für eine Auswahl an Sensoren und Aktoren implementiert. Diese Module setzen sich zusammen aus einem Programm für den Mikrocontoller und einem entsprechenden Modell in der Python-API.

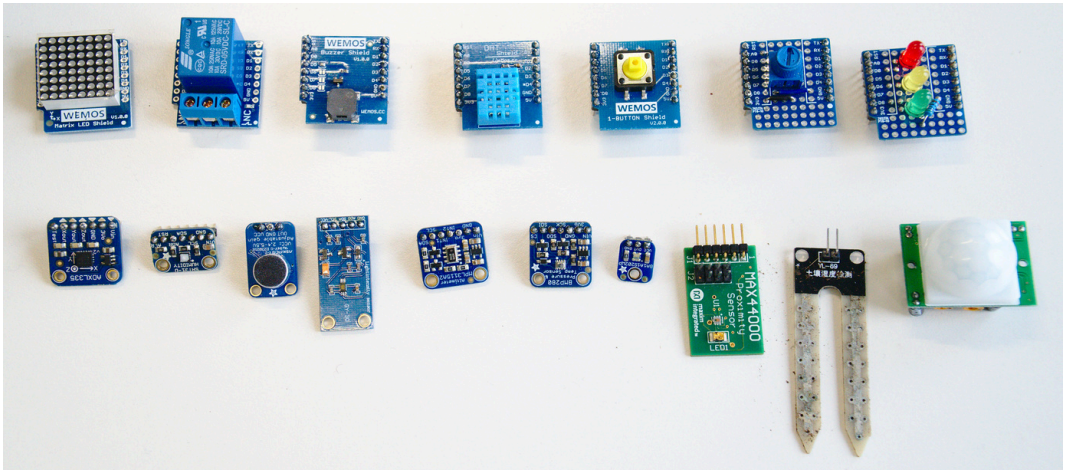

Abbildung 7: Kommerzielle (links oben) und selbstgebaute (rechts oben) Shields für den Wemos und diverse andere kompatible Sensoren (unten)

Pin Über dieses Modul können die diversen *GPIO*-Pins des Mikrocontollers angesteuert werden. Es stellt Funktionen für das digitale Auslesen und Beschreiben der Pins, sowie der analogen Ein- und Ausgabe über den eingebauten

ADC beziehungsweise PWM bereit. Außerdem können Callbackfunktionen auf Interrupts der Pins definiert werden. Alleine durch dieses Modul wird eine breite Auswahl von einfachen Sensoren und Aktoren unterstützt. Die Kommunikation mit diesem Modul erfolgt bidirektional über *TCP*.

DHT11 Der *DHT11* ist ein Sensor für Umgebungstemperatur und Luftfeuchtigkeit, der als Shield für den *Wemos D1 mini* verfügbar ist.

Button Dieses Modul sendet ein Event an das Gateway, sobald ein an einem von den NutzerInnen definierten Pin angeschlossener Taster oder Schalter seinen Zustand ändert. Der eventbasierte Ansatz hat gegenüber dem *GPIO*-Modul den Vorteil, dass die Node selbstständig eine Nachricht sendet und nicht erst vom Gateway ausgelesen werden muss. Dieses Modul kann auch für selbstgebaute Buttons, wie etwa Kupferkontakte an einer Tür, verwendet werden (siehe Abbildung 8).

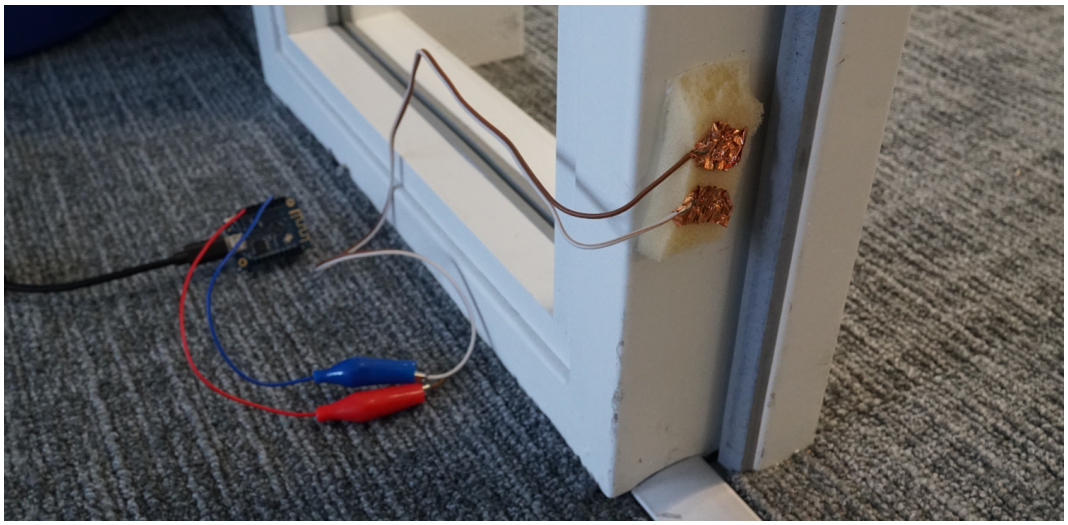

Abbildung 8: Selbstgebauter Sensor, der erkennt, ob eine Tür geöffnet oder geschlossen ist. Beim schließen der Tür werden die Kupferkontakte miteinander verbunden.

BH1750 Der *BH1750* ist ein Lichtintensitätssensor, der über den I^2C-Bus angesteuert wird.

Pir Modul für *PIR*-Bewegungsmelder, das beim Betreten und Verlassen des überwachten Bereichs Events auslöst.

7.2 Gateway

Die eigentliche Logik des Sensornetzwerks läuft auf einem zentralen Gateway. Dieser kommuniziert mit den Nodes, verarbeitet deren Nachrichten und stellt die Programmierumgebung in Form eines browserbasierten Frontends zur Verfügung. Prinzipiell kann ein beliebiges Linux-System als Gateway eingesetzt werden, indem die nötige Software darauf installiert wird (siehe Kapitel 8.1). Die Referenzimplementierung des Sensornetzwerks verwendet einen *Raspberry Pi 3*[38], da es sich dabei um ein verbreitetes und kostengünstiges System handelt, mit dem potentielle NutzerInnen möglicherweise schon Erfahrung haben.

Das Gateway erstellt automatisch ein WiFi-Netzwerk, mit dem sich Nodes verbinden können. Ein im Hintergrund laufender *MQTT-Broker* ermöglicht die Kommunikation zwischen Nodes und API.

In einer Konfigurationsdatei (`fusion.conf`) sind diverse globale Parameter wie zum Beispiel die Zugangsdaten des WiFi-Netzwerks und das *MQTT*-Topic definiert. Diese Datei wird auch vom Upload-Skript (beschrieben in Kapitel 7.1) ausgelesen, sodass die Parameter auf die Nodes übertragen werden können.

Die API des Sensornetzwerks liegt auf dem Gateway als Python-Modul vor, wodurch die Logik des Netzwerks in Form einfacher Python-Skripte umgesetzt werden kann. Jedoch empfiehlt es sich vor allem für unerfahrene NutzerInnen, eine browserbasierte Entwicklungsumgebung (beispielsweise *Jupyter-Notebooks*) für die Programmierung zu verwenden, da diese ein benutzerfreundliches, plattformunabhängiges und remote erreichbares Frontend bietet.

7.3 Kommunikation

7.3.1 Eigenes Nachrichtenprotokoll

In früheren Iterationen des Sensornetzwerks wurde ein eigenes Nachrichtenprotokoll verwendet, das aus Paketen mit einem kurzen Header mit Metadaten und der eigentlichen, binär codierten Nachricht bestand. Ein im Hintergrund laufendes Programm (*Daemon*) übernahm auf dem Gateway die Kommunikation mit den Nodes. Dieses Programm wurde durch *Inter Process Communication* via *Unix Domain Sockets*

[38] https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

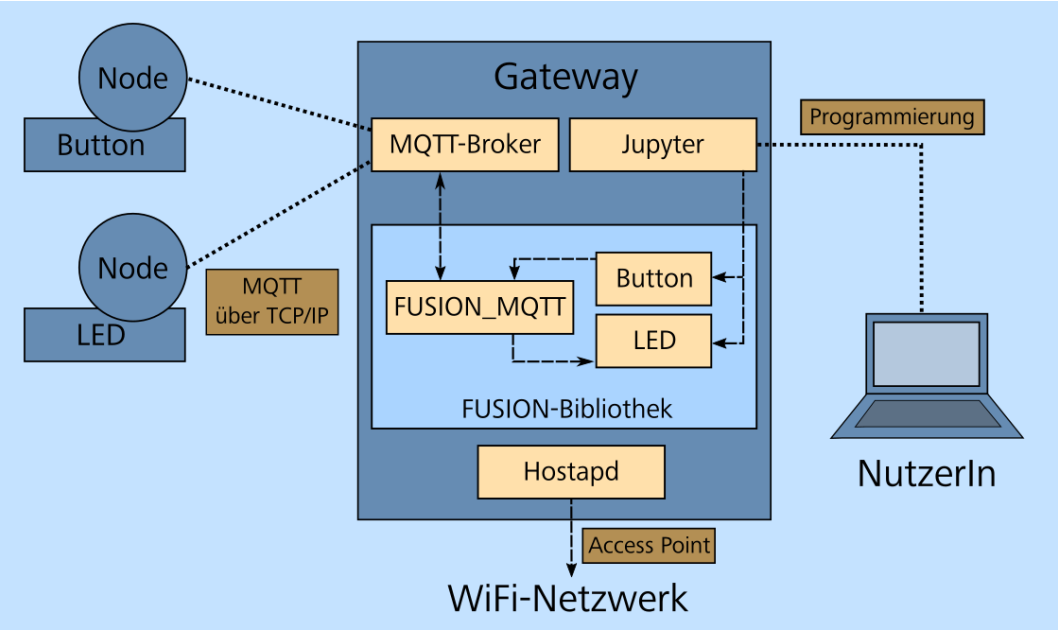

Abbildung 9: Aufbau und Kommunikation des Gateways

von der API angesteuert. Dadurch konnte das System einfach um die Kompatibilität zu neuen Netzwerkprotokollen erweitert werden, indem ein Kommunikationsdaemon für das jeweilge Protokoll erstellt wurde.

In späteren Iterationen wurde dieses Nachrichtenprotokoll durch das verbreitete *MQTT*-Protokoll ersetzt, da dieses als etablierter Standard eine größere Kompatibilität zu anderen Plattformen bietet und deutlich robuster gegenüber Verbindungsproblemen ist.

7.3.2 MQTT

Die Kommmunikation zwischen Nodes und Gateway findet vollständig über das Nachrichtenprotokoll *MQTT* statt. Dieses Protokoll wurde mit besonderer Rücksicht auf Robustheit bei unzuverlässigen Netzwerken entwickelt und gewährleistet sichere Datenübertragung trotz hoher Latenzen, geringer Bandbreite und potentiellem Verlust der Verbindung (Lampkin, 2012). Im Jahre 2014 wurde *MQTT* von der *Organization for the Advancement of Structured Information Standards* (OASIS) als internationaler Standard für das Internet of Things festgelegt (OASIS, 2014).

MQTT-Nachrichten bestehen aus einem leichtgewichtigen Header, einem so genannten *MQTT-Topic* in Textform und den eigentlichen Daten (*Payload*). Das *Topic* besteht aus mehreren, durch Schrägstriche getrennten *Subtopics*, die von NutzerIn-

nen festgelegt werden können und beispielsweise Ort, Art oder Datentyp der übertragenen Daten modellieren.

MQTT funktioniert nach dem *Publisher-Subscriber*-Prinzip. Das heißt, dass alle Nachrichten von so genannten *Publishern* unter einem bestimmten *Topic* veröffentlicht werden. *Subscriber* können diese *Topics* gezielt abonnieren und empfangen dann alle darunter veröffentlichten Nachrichten. Durch Wildcards können auch höhere Hierarchieebenen der *Topics* ungeachtet der *Subtopics* abonniert werden.

Die Referenzimplementierung eines Sensornetzwerks verwendet einen festen Aufbau für *MQTT-Topics*:

```
MQTT_NETWORK/MQTT_LOCATION/NODE_NAME/DATA_TYPE
```

MQTT_NETWORK Die niedrigste Hierarchieebene des *Topics* beschreibt den Namen des Netzwerks, der in der Konfigurationsdatei `fusion.conf` festgelegt wird. Die eindeutige Identifikation eines Netzwerks über dieses *Subtopic* ermöglicht es, mehrere voneinander unabhängige Netzwerke vom selben Gateway betreiben zu lassen.

MQTT_LOCATION Dieses in `fusion.conf` definierte *Subtopic* beschreibt die physische Position einer Node. So können sich beispielsweise mehrere gleiche Sensoren im selben Netzwerk befinden und anhand ihres Ortes identifiziert werden.

NODE_NAME Der Name einer Node dient deren eindeutiger Identifikation. Er wird beim Hochladen eines Programms auf die Node über Parameter des Upload-Skrips festgelegt. Eine Benennung nach der Aufgabe der Node (zum Beispiel “Thermometer“) ist sinnvoll.

DATA_TYPE Das letzte *Subtopic* beschreibt die Art der Daten, die in einem *MQTT*-Paket übertragen werden. Dieses wird vom auf der Node laufenden Programm festgelegt und an das *Topic* angehängt. Dadurch kann eine Node mehrere Datentypen unabhängig voneinander versenden, die dabei eindeutig identifizierbar und mit semantischem Kontext versehen sind.

Dieser Aufbau bietet neben der eindeutigen Beschreibung einer Node - sowohl am System, als auch in der physischen Welt - eine einfache Möglichkeit zum Filtern

von Daten auf der *Subscriber*-Seite. So können mittels Wildcards im abonnierten *MQTT-Topic* beispielsweise alle Nachrichten eines Netzwerks empfangen werden, die eine Temperatur enthalten:

```
NETZWERK/#/#/temperature
```

Der am Anfang des Kapitels bereits kurz beschriebene *MQTT-Broker* - ein Programm, das auf dem Gateway im Hintergrund läuft - empfängt alle von den Nodes gesendeten Nachrichten und leitet sie an die entsprechenden Subscriber weiter. Die in dieser Arbeit implementierte Programmierschnittstelle stellt Klassen bereit, die als *MQTT-Subscriber* agieren. Darüber hinaus können andere Schnittstellen, die *MQTT* unterstützen, einfach an das System angebunden werden.

7.4 API

Das Herzstück der Programmierschnittstelle stellt die als Python-Modul implementierte API dar. Diese besteht aus zahlreichen Klassen (Module), die jeweils eine Node repräsentieren und deren Funktionalität modellieren. Alle diese Klassen erben von der Basisklasse `FUSION_MQTT`, welche grundlegende Aufgaben wie die bereits beschriebene Kommunikation über *MQTT* übernimmt. Dazu werden Funktionen zum Serialisieren und Deserialisieren von Daten, dem Aufbauen und Aufrechterhalten der Verbindung, sowie dem Senden und Empfangen von Nachrichten zur Verfügung gestellt. Des Weiteren können über die Basisklasse Callbackfunktionen registriert werden, die beim Empfangen von Nachrichten mit einem bestimmten *Topic* aufgerufen werden.

Module werden mit einem Namen (`node_name`) und einem Ort (`node_location`) initialisiert. Diese entsprechen den im vorherigen Kapitel beschrieben *MQTT-Subtopics* und dienen der eindeutigen Identifizierung der dazugehörigen Nodes.

Die Funktion `OnUpdate(callback, data_entry)` registriert die als ersten Parameter übergebene Funktion als Callback für den als zweiten Parameter übergebenen Datentyp. Sobald eine Nachricht der korrespondierenden Node mit dem entsprechenden Datentyp empfangen wird, wird die so registrierte Callbackfunktion aufgerufen und die gesendeten Daten werden als Parameter übergeben. Darüber hinaus werden die empfagenen Daten auch im dictionary `data` der Basisklasse mit

dem entsprechenden Datentyp als Key gespeichert und können über die Funktion `get(data_entry)` abgerufen werden.

```
1  from .FUSION_MQTT import *
2
3  class Minimal(FUSION_MQTT):
4      def __init__(self, node_name, node_network=MQTT_DEFAULT_NETWORK,
           node_location=MQTT_DEFAULT_LOCATION):
5          FUSION_MQTT.__init__(self, node_name, node_network,
               node_location)
6          self.add_data_entry("value", int)
7
8      def info(self):
9          print("minimum working example")
```

Codebeispiel 3: Beispiel der Implementierung eines minimalistischen Moduls

Die Funktion `add_data_entry(data_entry, data_type)` fügt einen neuen Eintrag zur Liste von Datentypen (z.B. “temperature“, “humidity“, ...) hinzu, die vom Modul abonniert sind. Diese sind nicht zu verwechseln mit dem im zweiten Parameter übergebenen Datentyp der serialisierten Daten (z.B. `int`, `string`, ...), die als Payload der Nachricht übertragen werden. Dieser ist nötig um die Daten korrekt deserialisieren zu können, da *MQTT* Daten ungeachtet ihres Typs überträgt.

Um ein neues Modul zu erstellen, muss dieses lediglich von der `FUSION_MQTT`-Klasse erben und im Konstruktor müssen vom Modul abonnierte Datentypen mithilfe von `add_data_entry` festgelegt werden. Weitere Funktionalität kann bei Bedarf zum Modul hinzugefügt werden.

```
1  from FUSION import Minimal
2
3  # initialize the module
4  sensor = Minimal(node_name = "mini", node_location="1104")
5
6  # define a callback
7  def on_update(data):
8      print(data)
9
10 # register the callback
11 sensor.OnUpdate(on_update, "value")
```

Codebeispiel 4: Verwendung eines simplen Moduls

Mit der Funktion `send_message(topic, payload)` kann eine Nachricht über *MQTT* versendet werden. Dabei gibt der Parameter `topic` das letzte *Subtopic* der Nachricht

an - Netzwerk, Ort und Name wurden bereits bei der Initialisierung des Moduls festgelegt. Mit den Parameter `payload` werden die Daten übergeben, die mit der Nachricht versendet werden.

Darüber hinaus enthält die API auch Klassen und Funktionen, die keine Nodes modellieren. Diese dienen der Verarbeitung und Visualisierung von Daten, der Kommunikation mit diversen Prozessen auf dem Gateway, oder als Wrapper für bestehende Python-Bibliotheken:

E-Mail Dieses Modul stellt eine einfache Funktion bereit, um E-Mails über *SMTP* zu versenden.

Visualize Ein Wrapper für das Python-Paket *Bokeh*[39], mit dem Daten visualisiert werden können.

Filter Eine Sammlung diverser gängiger Filter zur Datenverarbeitung.

Log Stellt Funktionen bereit, mit denen Daten in einer Log-Datei gespeichert werden können.

Die Klassen der API sind nach den Richtlinien der objektorientierten Programmierung aufgebaut. Technische Aspekte wie Netzwerkcode zur Kommunikation, *Exception-Handling* und die Deserialisierung von Daten in abstrakte Objekte, wurden wenn möglich in eigene Klassen ausgelagert, sodass die Module schlank und lesbar gestaltet werden konnten. Verständliche Variablen- und Funktionsbezeichner, sowie ausführliche Kommentare tragen weiter zur Lesbarkeit des Quellcodes bei. Außerdem enthält jedes Modul die Funktion `info()`, welche eine Beschreibung der Funktionen des Moduls ausgibt.

Durch die Unterteilung der API in verschiedene Ebenen bleibt technisch anspruchsvoller Quellcode vor unerfahrenen NutzerInnen verborgen, während erfahrenen NutzerInnen der Zugriff darauf nicht verwehrt wird. Diese können den Umfang der Programmierschnittstelle durch das Schreiben eigener Module erweitern. Externe Python-Bibliotheken können auf dem herkömmlichen Weg über den Paketmanager *PIP* installiert und durch die `import`-Anweisung eingebunden werden.

[39] https://bokeh.pydata.org/

7.4.1 *Python als Programmiersprache*

Python ist eine weit verbreitete Programmiersprache, die sich durch ihre simple und logische Syntax auszeichnet und dabei den natürlichen Sprachen näher kommt als andere Programmiersprachen. Da die Sprache fast vollständig auf Boilerplate verzichtet (siehe Codebeispiel 5 und 6 für einen Vergleich von Python und C) und funktionale Blöcke durch Einrückung gruppiert, ist Python-Code auch für AnfängerInnen leicht zu lesen und verstehen (Fangohr, 2004).

```
1 print("Hallo Welt!")
```

Codebeispiel 5: Hallo-Welt-Programm in Python

```
1 #include <stdio.h>
2
3 int main()
4 {
5    printf("Hallo Welt!");
6    return 0;
7 }
```

Codebeispiel 6: Hallo-Welt-Programm in C

Trotzdem unterstützt Python auch komplexe Funktionalität und kann für vollständig objektorientierte Programmierung verwendet werden.

Python ist eine interpretierte Sprache, wodurch es kompatibel zu den meisten verbreiteten Plattformen ist. Darüber hinaus können Python-Interpreter bei Laufzeitfehlern sehr aussagekräftige Fehlermeldungen mit Traceback ausgeben.

Durch Programmbibliotheken kann der Funktionsumfang der Sprache erweitert werden, wodurch sie fächerübergreifend einsetzbar ist. So existieren beispielsweise Bibliotheken für mathematische Operationen (*NumPy*[40]), Visualisierung von Daten (*Matplotlib*[41]), Statistik (*SciPy*[42]), Bildverarbeitung (*Pillow*[43]), *Natural Language Processing* (*NLTK*[44]) und Machine Learning (*TensorFlow*[45]). Dadurch wird mit einer gemeinsamen Sprache ein Funktionsumfang abgedeckt, für den einst zahlreiche ver-

[40] https://www.numpy.org/
[41] https://matplotlib.org/
[42] https://www.scipy.org/
[43] https://pillow.readthedocs.io/en/stable/
[44] https://www.nltk.org/
[45] https://www.tensorflow.org/

schiedene Programme benötigt wurden (Langtangen, 2008). Diese mussten von AnwenderInnen einzeln erlernt werden und waren untereinander nicht immer kompatibel.

Aufgrund der niedrigen Einstiegsschwelle und des breiten Angebots an Programmbibliotheken ist Python eine vielseitige und leicht erlernbare Programmiersprache, die sich deshalb gut für End User Programming eignet. Darüber hinaus besteht durch *Jupyter Notebooks* (siehe Kapitel 6.5) die Möglichkeit, Python-Programme in einem Webbrowser zu schreiben und auszuführen, wodurch keine dedizierte Entwicklungsumgebung installiert werden muss und kollaborative Arbeit an Projekten ermöglicht wird.

8 Inbetriebnahme und Verwendung

Um die Programmierschnittstelle verwenden zu können, muss zuerst das Sensornetzwerk auf einem Linux-System mit WiFi-Unterstützung, das später als Gateway dient, installiert werden. Es empfielt sich, dafür einen *System-on-a-Chip*-Computer, wie beispielsweise einen *Raspberry Pi* zu verwenden.

8.1 Installation

Zuerst muss das entsprechende Repository[46] heruntergeladen werden. Danach navigiert man in das Verzeichnis des Repository und führt das Installationsskript `setup.sh` aus. Dadurch werden benötigte Komponenten automatisch installiert und der Service zum Erstellen eines *WiFi Access Point* für das Sensornetzwerk eingerichtet. Das Setup besteht aus folgenden Schritten:

1. Installation von benötigten und empfehlenswerten Paketen über den Paketmanager apt
2. Einrichten einer Python-Umgebung und Installation von benötigten Python-Paketen über Pythons Paketmanager *PIP*
3. Einrichten des *Command Line Interface* der *PlatformIO*-Entwicklungsumgebung
4. Konfiguration von *Jupyter Notebooks*
5. Konfiguration der Services *dnsmasq* und *hostapd* zum automatischen Erstellen eines *WiFi Access Point*

8.2 Benutzung

Nach der Installation kann mithilfe des Upload-Skripts `pio_compile.sh` ein Programm aus dem *nodes*-Verzeichnis auf den Mikrocontroller einer Node hochgeladen werden. Dieses Programm wird beim Bootvorgang der Node automatisch gestartet und läuft bis zu einem Reset des Geräts oder dem Trennen der Stromversorgung. Die Verbindung zum Gateway und das Senden von Nachrichten an den

[46] https://github.com/A-Schmid/FUSION_EUP

MQTT-Broker geschieht automatisch. Das Upload-Skript liest die Konfigurationsdatei `fusion.conf` und kann mit diversen Parametern ausgeführt werden. Diese Daten werden der Node beim Hochladen des Programms übergeben.

Folgende Parameter können dem Upload-Skript übergeben werden:

NODE_NAME Einzigartiger Name der Node, auf der das Programm läuft. Stellt das vierte *Subtopic* des *MQTT-Topic* dar.

DELAY Zeit in Millisekunden, die das Programm nach jedem Durchlauf wartet.

PROTOCOL Verwendetes Nachrichtenprotokoll. Der aktuelle Stand der Schnittstelle unterstützt nur *MQTT*.

ACTION Übergibt man hier "debug", so wird das Programm nur kompiliert und nicht auf die Node hochgeladen.

Folgende Parameter werden aus der Konfigurationsdatei `fusion.conf` ausgelesen:

WIFI_SSID SSID des WiFi-Netzwerks, das für das Sensornetzwerk verwendet wird.

WIFI_PASSWORD Passwort des entsprechenden WiFi-Netzwerks.

MQTT_SERVER IP-Adresse des Gateway.

MQTT_PORT Port auf dem der *MQTT-Broker* läuft (Standard: 1883).

MQTT_TOPIC_NETWORK Name des Sensornetzwerks als erstes *Subtopic* des *MQTT-Topic*.

MQTT_TOPIC_LOCATION Ort des Sensornetzwerks als zweites *Subtopic* des *MQTT-Topic*.

Die so übergebenen Parameter werden im Programmcode der Node als Präprozessor-Makros definiert. Dadurch kann das selbe Programm mit unterschiedlichen Konfigurationen auf verschiedene Nodes geladen werden, ohne dass jedes Mal der Code angepasst werden muss.

Auf dem Gateway können nun Python-Programme gestartet werden, welche das *FUSION*-Package verwenden. Es wird empfohlen, im Verzeichnis der Programmierschnittstelle den Befehl `jupyter notebook` auszuführen. Dadurch wird *Jupyter*

Notebook gestartet und kann von allen Geräten, die sich im selben WiFi-Netzwerk befinden, über einen Webbrowser aufgerufen werden.

Um die Programmierschnittstelle zu verwenden, erstellt man ein neues Python3-Notebook und importiert benötigte Module aus dem *FUSION*-Package. Dann muss eine Instanz des der Node entsprechenden Moduls erstellt werden. Über die `OnUpdate`-Funktion des Moduls können Callbackfunktionen auf neue Nachrichten der Node registriert werden und über die `get()`-Funktion können Daten des Moduls abgefragt werden. Die `send_message`-Funktion sendet Nachrichten an die Node. Durch Python-Code kann beliebige Programmlogik implementiert werden.

Im Verzeichnis `test_scripts` befinden sich Python-Dateien, die Codebeispiele für die Verwendung der Schnittstelle enthalten.

8.3 Erweiterung

Um eigene Module zu schreiben, muss eine neue Python-Datei im `FUSION`-Verzeichnis erstellt werden. Darin wird eine Klasse erstellt, die von der `FUSION\_MQTT`-Klasse erbt. Im Konstruktor der Klasse werden mithilfe der `add_data_entry`-Funktion neue Datentypen registriert, die das Modul empfangen kann. Eigene Funktionalität kann nach belieben zum Modul hinzugefügt werden. Zuletzt muss die Datei `__init__.py` noch um einen Eintrag zum neu erstellten Modul erweitert werden.

Um eigene Nodes zu implementieren, erstellt man ein Unterverzeichnis in `nodes/`. Darin wird eine `.ino`-Datei erstellt, die den gleichen Namen wie das Verzeichnis hat. In dieser Datei kann die Funktionalität der Node mit C++ implementiert werden. Bei komplexen Projekten empfielt es sich, die Funktionalität in einer Bibliothek zu kapseln. Diese wird im Verzeichnis `nodes/libraries/src/` in Form einer C++-Datei und einer dazugehörigen Header-Datei abgelegt. Externe Bibliotheken können über die *PlatformIO*-Umgebung installiert werden.

9 Evaluierung

Die im Rahmen dieser Arbeit implementierte Programmierschnittstelle und das dazugehörige Sensornetzwerk sollen im Hinblick auf Benutzbarkeit, Funktionalität und Robustheit evaluiert werden. Dabei ist insbesondere der Vergleich zu einer visuellen Programmierumgebung mit ähnlichem Funktionsumfang interessant. Darüber hinaus sollen Schwachstellen und technische Limitierungen des Systems festgestellt werden.

Um die Benutzbarkeit (Usability) eines Systems zu evaluieren, werden im Forschungsgebiet der Mensch-Maschine-Interaktion (HCI) oft Nutzerstudien durchgeführt, bei denen ProbandInnen das System benutzen, um ihnen gestellte Aufgaben (Tasks) zu lösen. Die Usability wird dabei oft mit einem *Mixed-Methods*-Ansatz (van Turnhout et al., 2013), also einer Kombination aus qualitativen und quantitativen Daten festgestellt. So können Kommentare der ProbandInnen während des Tasks Hinweise auf mögliche Usabilityprobleme geben, auf die in einem Interview nach dem Task näher eingegangen werden kann. Laut Nielsen (2000) reicht dabei bereits eine kleine Stichprobe aus, um einen Großteil der Usabilityprobleme eines Systems zu finden. Quantitative Daten zur Usability eines Systems können erhoben werden, indem beispielsweise die Zeit bis zum Abschluss eines Tasks oder die Fehlerrate während der Bearbeitung gemessen wird. Zudem existieren validierte Fragebögen, die von ProbandInnen nach der Benutzung eines Systems ausgefüllt werden, um die Usability zu quantisieren.

Obwohl in der HCI meist die Usability von graphischen Benutzerschnittstellen gemessen wird, können Evaluierungsmethoden wie Nutzerstudien und Heuristiken laut Jiang (o. J.); Grill et al. (2012) auch zur Evaluation der Usability von APIs verwendet werden. Myers & Stylos (2016) befassen sich umfassend mit API-Usabilty, indem sie anhand von Beispielen häufige Usabilityprobleme aufzeigen. Daraus leiten sie Richtlinien zur Evaluation der Usability von APIs ab und passen letztlich Nielsens Heuristiken für *User-Interface-Design* (Nielsen, 1994) an das Design von Programmierschnittstellen an (frei übersetzt nach Myers & Stylos):

Visibility of system status Es sollte für NutzerInnen einfach sein, den Zustand des Systems zu überprüfen. Operationen, die aufgrund des Systemzustands nicht

durchgeführt werden können, sollen angemessene Fehlermeldungen geben.

Match between system and real world Funktionsnamen und die Organisation von Funktionen in Klassen sollen den Erwartungen von NutzerInnen entsprechen. So soll beispielsweise für häufig verwendete Funktionen ein möglichst generischer Name verwendet werden.

User control and freedom NutzerInnen sollen in der Lage sein, Operationen abzubrechen und den Zustand des Systems zurückzusetzen.

Consistency and standards Alle Bestandteile des Designs sollen über das gesamte System hinweg konsistent sein.

Error prevention Das System sollte NutzerInnen so anleiten, dass sie es richtig verwenden.

Recognition rather than recall Da viele NutzerInnen von IDEs die Funktion zur automatischen Codevervollständigung verwenden, um das System zu explorieren, müssen Namen von Funktionen und Klassen klar und verständlich sein, sodass sie wiedererkannt werden können.

Flexibility and efficiency of use NutzerInnen sollen ihre Aufgaben mit einem System effizient erledigen können.

Aesthetic and minialist design Kleinere und weniger komplexe Systeme sind häufig einfacher zu benutzen. Durch eine klare Struktur und sinnvolle Benennungen können aber auch umfangreiche Systeme gut bedienbar gestaltet werden.

Help users recognize, diagnose, and recover from errors Die API sollte auf Fehler mit sinnvollen und hilfreichen Fehlermeldungen reagieren, um gute Usability zu gewährleisten.

Help and documentation (keine nähere Beschreibung durch Myers & Stylos (2016))

9.1 Vergleich FUSION und Node-RED

Da die heuristische Evaluation laut Nielsen & Molich (1990) normalerweise von ExpertInnen durchgeführt wird, die in dieser Arbeit erstellte Programmierschnittstelle aber explizit auf NutzerInnen mit wenig Programmiererfahrung ausgelegt ist, ist

eine heuristische Evaluation im klassischen Sinne in diesem Kontext nicht sinnvoll. Deshalb wird eine Nutzerstudie mit ProbandInnen, die zur Zielgruppe des Systems gehören, durchgeführt. In einem semistrukturierten Interview nach der Studie werden dennoch Aspekte der Heuristiken für API-Usability erfragt. Zudem wird die *System Usability Scale* (SUS) (Brooke, 1996) verwendet, um die Usability der Programmierschnittstelle zu quantisieren. Dieser Fragebogen wurde ausgewählt, weil er bereits mit einer kleinen Anzahl von ProbandInnen ein repräsentatives Ergebnis liefert (Brooke, 1996) und er laut Lewis (2018) auch für die Evaluation von Programmierschnittstellen geeignet ist.

9.1.1 Studiendesign

Ziel der Nutzerstudie war es, die Programmierschnittstelle (im Folgenden *"FUSION"*) mit einer etablierten, visuellen Programmierumgebung im Hinblick auf Effizienz und Usability zu vergleichen. Als Vergleich wurde das bereits in Kapitel 4.2.1 beschriebene System *Node-RED* ausgewählt, da es auf IoT-Anwendungen ausgelegt ist und mit dem in dieser Arbeit erstellten Sensornetzwerk kompatibel ist. Es wurden drei realitätsnahe, voneinander unabhängige Tasks definiert, die einen möglichst großen Teil des Funktionsumfangs der API abdecken und mit beiden Systemen umgesetzt werden können. Um einen möglichen Einfluss durch Befangenheit zu reduzieren, wurden die Tasks von einem unabhängigen Dritten validiert.

Aufgabe 1 Auslesen der Luftfeuchtigkeit, Ausgabe des Ergebnis in die Konsole

Aufgabe 2 Auslesen der Temperatur und Senden einer E-Mail mit dem Ergebnis, sobald ein Button gedrückt wird

Aufgabe 3 An- und Ausschalten einer LED, abhängig von der mit einem Sensor gemessenen Lichtintensität

(Vollständige Tasks: siehe Anhang A[47][48])

Die zum Lösen der Aufgaben benötigten Hardware-Nodes wurden den ProbandInnen zur Verfügung gestellt. Sie waren beschriftet und bereits mit einem funktionierenden Programm bespielt.

[47] Nutzerstudie/Dokumente/tasks_fusion.pdf
[48] Nutzerstudie/Dokumente/tasks_nodered.pdf

Alle TeilnehmerInnen der Studie sollten die Tasks mit beiden Systemen durchführen (*Within-Subjects-Design*). Um Lerneffekten entgegenzuwirken, wurde das System, mit dem begonnen wird, anfangs zufällig und bei dem/der letzten Proband/in manuell festgelegt, sodass letztlich beide Systeme in etwa gleich oft zuerst bearbeitet wurden.

Vor der Studie wurden demographische Daten und die bisherige Programmiererfahrung der ProbandInnen durch einen Fragebogen (siehe Anhang A[49]) erhoben. Dieser baute auf den bereits bei der Anforderungserhebung verwendeten Fragebogen (siehe Kapitel 5.2) auf und ergänzte ihn nur um die Frage, mit welchen Programmiersprachen und -umgebungen die ProbandInnen bereits Erfahrung haben.

Die TeilnehmerInnen wurden vor der Studie durch den Testleiter kurz in das jeweilige System eingeführt, indem er das Starten und Stoppen von Programmen, das Senden und Empfangen von Daten, sowie den allgemeinen Aufbau des Systems erklärte. Zudem erhielten sie eine knappe Anleitung (siehe Anhang A[50][51]), in der die wichtigsten Komponenten beschrieben waren. Die ProbandInnen erhielten neben der Anleitung keine Dokumentation, sondern sollten Fragen direkt an den Testleiter stellen. So konnte gemessen werden, wie oft die Hilfe einer Dokumentation benötigt wurde. Zusätzlich wurden beim Bearbeiten der Tasks mit der in dieser Arbeit erstellten Programmierschnittstelle Codebeispiele bereitgestellt, die simple Anwendungen enthielten (siehe Anhang A[52]). Dies hat den Grund, dass Codebeispiele eine explizite Anforderung an das System darstellen, die sich aus der im Rahmen der Anforderungserhebung durchgeführten Fokusgruppenstudie (siehe Kapitel 5.2) ergeben hat.

Nach der Einführung hatten die ProbandInnen die Gelegenheit, Fragen zum jeweiligen System zu stellen. Danach begannen sie mit der Bearbeitung der Aufgaben. Die Bearbeitungszeit jedes Tasks wurde mit einer Stoppuhr gemessen. Wurde ein Task abgebrochen, so wurde dies festgehalten.

Nachdem alle drei Tasks mit einem System bearbeitet wurden, bewerteten die ProbandInnen die subjektive Usability des eben verwendeten Systems, indem sie den Fragebogen der System Usability Scale ausfüllten. Daraufhin wurden die Pro-

[49] Nutzerstudie/Dokumente/fragebogen.pdf
[50] Nutzerstudie/Dokumente/guide_fusion.pdf
[51] Nutzerstudie/Dokumente/guide_nodered.pdf
[52] Nutzerstudie/Codebeispiele/

bandInnen zu ihrer Meinung über das eben verwendete System befragt und eventuelle während der Bearbeitung der Tasks aufgetretenen Probleme wurde diskutiert.

Direkt im Anschluss wurden die Tasks mit dem zweiten System durchgeführt, worauf wieder ein SUS-Fragebogen und ein kurzes Interview folgte.

9.1.2 Vorstudie

Es wurde eine Vorstudie mit zwei ProbandInnen durchgeführt, um das Studiendesign zu überprüfen und eventuelle technische Probleme am System, sowie schlecht verständliche Formulierungen in Anleitung und Aufgabenstellung, frühzeitig zu erkennen. Auch die Zeit, die zum Lösen der Aufgaben benötigt wird, sollte durch die Vorstudie eingeschätzt werden. Die Vorstudie wurde mit erfahrenen ProgrammiererInnen durchgeführt.

Einige technische Probleme, wie das gelegentliche Abbrechen der Verbindung, sobald mehrere Nodes im Einsatz waren, traten während der Vorstudie auf und wurden noch vor der Hauptstudie behoben.

9.1.3 Hauptstudie

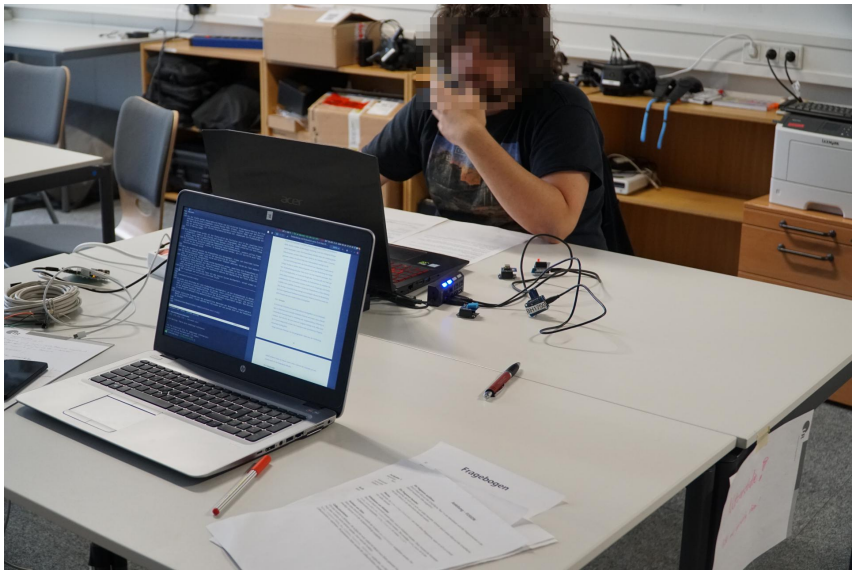

Abbildung 10: Aufbau der Studie

Die Studie wurde mit sieben ProbandInnen (fünf männlich, zwei weiblich, niemand divers) mit einem Alter von 24 bis 54 Jahren (Durchschnitt: 31) durchgeführt. Zwei von ihnen schätzten sich selbst als fortgeschritten ein, vier bezeichneten sich

als AnfängerInnen und eine Person hatte keine Programmierkenntnisse. Die Programmiererfahrung der ProbandInnen lag zwischen null und fünf Jahren (Durchschnitt: 1,9 Jahre). Die Verteilung der Berufe der ProbandInnen kann Tabelle 1 entnommen werden, die den ProbandInnen bekannten Programmiersprachen sind in Tablle 2 aufgelistet.

Beruf	Anzahl
Studierende	3
KFZ-MeisterIn	1
ElektrikerIn/TanzlehrerIn	1
EntwicklungsingenieurIn	1
Training Specialist	1

Tabelle 1: Berufe der ProbandInnen

Programmiersprache	Anzahl
C	4
Java	4
Python	4
JavaScript	2
C#	2
TypeScript	1
MatLab	1
Arduino	1
PHP	1
R	1
SPSS	1

Tabelle 2: Den ProbandInnen bekannte Programmiersprachen und -umgebungen

Da beide verglichenen Systeme über einen Webbrowser bedient werden können und somit keine zusätzliche Installation von Software nötig ist, war es den TeilnehmerInnen der Studie freigestellt, ihren eigenen Rechner zu verwenden. Wenn sie dies nicht wollten, wurde ihnen ein *ASUS*-Laptop (*K53S*) mit dem Betriebssystem Ubuntu 16.10, dem Browser *Mozilla Firefox* und einer Maus zur Verfügung gestellt.

Die Studie wurde im Versuchsraum 4 der *TechBase* in Regensburg durchgeführt. Der genaue Aufbau ist in Abbilung 10 zu sehen.

9.1.4 Ergebnisse der Hauptstudie

Aufgrund der sehr geringen Anzahl an ProbandInnen ist der zentrale Grenzwertsatz (Pólya, 1920) nicht erfüllt. Somit können die quantitativen Ergebnisse der Nutzerstudie nicht verallgemeinert werden. Aus diesem Grund wird bei der Auswer-

tung der Ergebnisse größtenteils auf statistische Tests verzichtet, da diese mit einer so kleinen Stichprobe nicht aussagekräftig wären. Jedoch ist eine Tendenz erkennbar, anhand derer eine ausführlichere Studie geplant und durchgeführt werden kann.

Die Auswertung der SUS-Fragebögen hat ergeben, dass beide Systeme mit durchschnittlichen SUS-Scores unter sechzig gravierende Usabilityprobleme aufweisen (Rauer, 2011). Dabei schneidet *Node-RED* mit einem Durchschnittswert von vierzig deutlich schlechter ab als *FUSION* mit einem Durchschnittswert von 56,4 (siehe Abbildung 11 und Tabelle 3). Nach die Ergebnisse der SUS mit dem Shapiro-Wilk-Test (Shapiro & Wilk, 1965) positiv auf eine Normalverteilung getestet wurden, wurde ein t-Test für unabhängige Stichproben durchgeführt, um den Unterschied in der Usability der beiden Systeme auf Signifikanz zu testen. Der t-Test ergab einen signifikanten Unterschied mit einem p-Wert von 0,03.

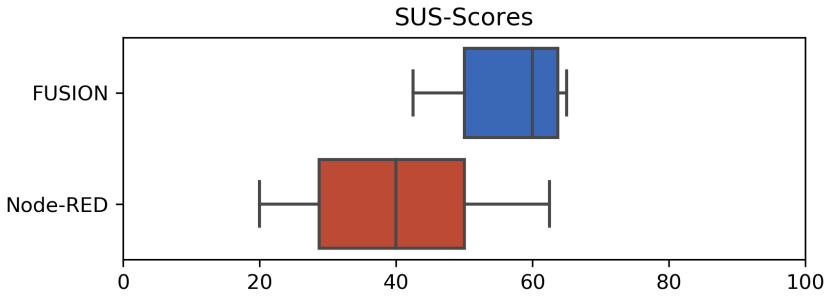

Abbildung 11: Ergebnisse der SUS

System	Minimum	Durchschnitt	Maximum
FUSION	42,5	56,4	65
Node-RED	20	40	62,5

Tabelle 3: SUS-Scores nach System

Die qualitativ durch Interviews und anhand der von den ProbandInnen gestellten Fragen gesammelten Daten werden im Folgenden in den Kontext der von Myers & Stylos (2016) an API-Usability angepassten Nielsen-Heuristiken Nielsen (1994) eingeordnet.

FUSION

Visibility of system status Drei ProbandInnen hatten Probleme mit dem Aufbau von *Jupyter Notebooks*. Sie führten die erste Zelle, welche die nötigen Pakete

importiert, nicht aus und wussten nicht, warum das System beim Ausführen ihres Programms eine Fehlermeldung ausgibt.

Match between system and real world Drei ProbandInnen lobten den Aufbau der Programmierschnittstelle und bezeichneten die gewählten Funktions- und Variablennamen als sinnvoll.

Bei zwei ProbandInnen stimmte deren mentales Modell nicht mit dem Aufbau der Programmierschnittstelle überein.

User control and freedom Ein/e Proband/in war zwar in der Lage, das System bei Fehlverhalten zurückzusetzen, kritisierte aber, dass dies häufig nötig war, um mit dem Bearbeiten der Aufgabe fortfahren zu können.

Consistency and standards (kein Feedback der NutzerInnen zu diesem Punkt)

Error prevention Zwei ProbandInnen hatten Probleme mit der Syntax von Python und der API.

Recognition rather than recall Zwei ProbandInnen gaben an, dass sie die textbasierte Programmierumgebung aufgrund von Gewohnheit bevorzugten.

Flexibility and efficiency of use Zwei ProbandInnen bezeichneten die Verwendung der Programmierschnittstelle als einfach.

Zwei ProbandInnen fanden das Strukturieren des Programms und das Anlegen von Variablen mit *FUSION* einfacher als mit *Node-RED*

Aesthetic and minialist design (kein Feedback der NutzerInnen zu diesem Punkt)

Help users recognize, diagnose, and recover from errors Ein/e Proband/in hielt die ausgegebenen Fehlermeldungen für hilfreich, während ein/e andere/r Proband/in bemängelte, dass das System zu wenig Feedback gibt.

Help and documentation Die bereitgestellten Codebeispiele fanden vier ProbandInnen hilfreich, zwei gaben an, dass die Aufgaben ohne diese für sie nicht lösbar gewesen wären.

Ein/e Proband/in hielt die Anleitung des Systems für zu knapp.

Andere Aspekte Ein/e Proband/in hob den großen Funktionsumfang von *FUSION* als positiv hervor.

Bei zwei ProbandInnen traten technsiche Probleme auf, die sie bemängelten.

Zwei ProbandInnen fanden, dass die Programmierschnittstelle nicht für Einsteiger geeignet sei.

Vier ProbandInnen bevorzugten *FUSION* gegenüber *Node-RED*.

Node-RED

Visibility of system status Zwei ProbandInnen kritisierten fehlendes oder falsches Feedback.

Drei ProbandInnen verstanden den Aufbau der zwischen den Nodes gesendeten Nachrichten nicht.

Match between system and real world Ein/e Proband/in bezeichnete den Aufbau von *Node-RED* als gut nachvollziehbar und intuitiv.

Vier ProbandInnen hatten Probleme, vorhandene Funktionen des Systems zu finden.

Drei ProbandInnen verstanden nicht, wie Variablen in *Node-RED* angelegt, verändert und ausgelesen werden können.

User control and freedom Ein/e Proband/in fragte explizit nach, wie man das Ausführen eines Programms beenden könne.

Consistency and standards (kein Feedback der NutzerInnen zu diesem Punkt)

Error prevention Ein/e Proband/in hob positiv hervor, dass *Node-RED* durch den visuellen Ansatz Syntaxfehler vermeide.

Drei ProbandInnen hatten Probleme damit, Nodes korrekt zu verbinden. So versuchten sie beispielsweise, eine Output-Node als Input zu verwenden, oder die Eingänge zweier Nodes miteinander zu verbinden.

Recognition rather than recall Drei ProbandInnen bezeichneten das System als unübersichtlich.

Flexibility and efficiency of use Vier ProbandInnen kritisierten das Fehlen von Funktionen, die ihrer Meinung nach Teil des Systems sein sollten.

Ein/e Proband/in empfand die Bedienung von *Node-RED* als umständlich.

Aesthetic and minialist design Zwei ProbandInnen fanden *Node-RED* visuell ansprechend.

Zwei ProbandInnen gaben an, dass zu viele Einstellungen die Usability des Systems negativ beeinflussten.

Help users recognize, diagnose, and recover from errors (kein Feedback der NutzerInnen zu diesem Punkt)

Help and documentation Ein/e Proband/in kritisierte die technisch anspruchsvollen Beschreibungstexte der Module.

Andere Aspekte Drei ProbandInnen hielten *Node-RED* für leicht erlernbar.

Zwei ProbandInnen bevorzugten *Node-RED* gegenüber *FUSION*.

Task	System	Minimum	Durchschnitt	Maximum	Abgebrochen
Task 1	FUSION	5:22 min	12:41 min	23:30 min	0
	Node-RED	2:07 min	7:51 min	18:27 min	0
Task 2	FUSION	8:34 min	15:45 min	23:49 min	2
	Node-RED	21:06 min	21:06 min	21:06 min	6
Task 3	FUSION	5:03 min	15:33 min	37:49 min	1
	Node-RED	10:51 min	18:21 min	25:23 min	3

Tabelle 4: Taskzeiten erfolgreich abgeschlossener Aufgaben nach System

Tasks mit dem *FUSION*-System wurden deutlich häufiger erfolgreich abgeschlossen als mit *Node-RED* (siehe Abbruchrate in Tabelle 4). Diese hohe Abbruchrate bestärkt die Vermutung, dass *Node-RED* ernsthafte Usabilityprobleme aufweist.

Task 1 wurde von allen ProbandInnen mit beiden Systemen erfolgreich bearbeitet, dabei wurde die Aufgabe mit *Node-RED* im Durchschnitt schneller gelöst als mit *FUSION*.

Task 2 mit dem *FUSION*-System wurde von zwei ProbandInnen abgebrochen, während nur ein/e Proband/in die Aufgabe mit *Node-RED* lösen konnte. Da deshalb nur eine gemessene Taskzeit für *Node-RED* vorliegt, sind die Ergebnisse für diesen Task nicht vergleichbar.

Task 3 wurde mit *FUSION* im Durchschnitt schneller abgeschlossen als mit *Node-RED*. Auch die Abbruchrate ist bei *FUSION* (ein Abbruch) geringer als bei *Node-RED* (drei Abbrüche).

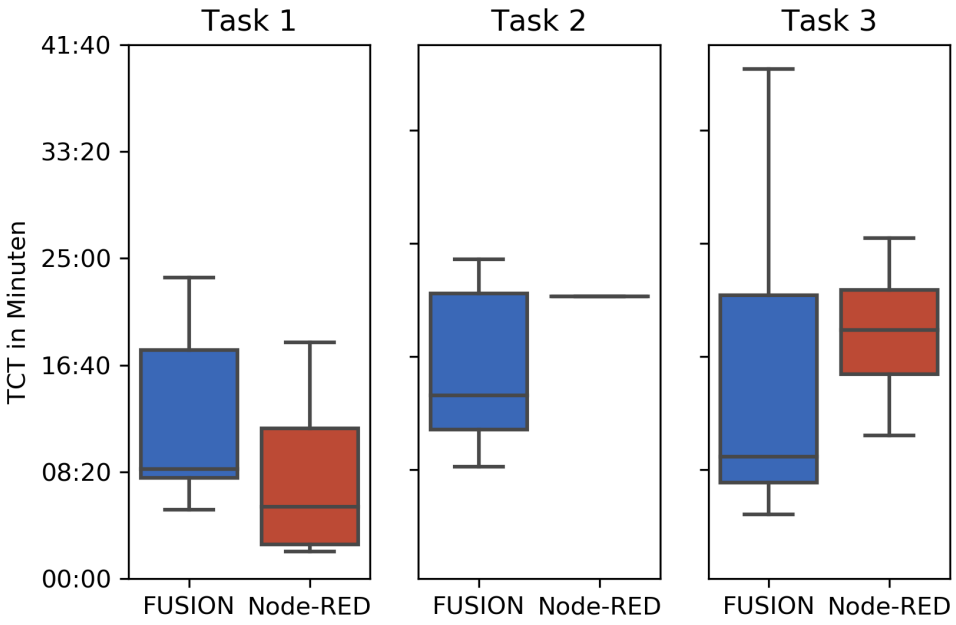

Abbildung 12: Taskzeiten erfolgreich abgeschlossener Aufgaben nach System

In Abbildung 13 ist erkennbar, dass Aufgaben mit dem System, das als zweites verwendet wurde, tendenziell schneller abgeschlossen wurden. Dies lässt auf einen möglichen Lerneffekt schließen, der die Bearbeitungszeit der Aufgaben mit dem zweiten System beeinflusst (Budiu, 2018).

Die Nutzerstudie hat ergeben, dass beide verglichenen Systeme von Usability-problemen betroffen sind. Bei *Node-RED* sind diese so gravierend, dass viele ProbandInnen nicht in der Lage waren, Aufgaben erfolgreich abzuschließen. Trotz geringer oder gar keiner Vorerfahrung im Programmieren wurde die textbasierten Programmierschnittstelle von den ProbandInnen besser verstanden als der Aufbau der visuellen Programmierumgebung, welcher nicht dem mentalen Modell vieler NutzerInnen entsprach. So ist bei *Node-RED* trotz der graphischen Darstellung des Programmablaufs der Systemzustand nicht transparent genug. Darüber hinaus sind einige Funktionen nicht den Erwartungen der NutzerInnen entsprechend benannt, beziehungsweise gar nicht vorhanden.

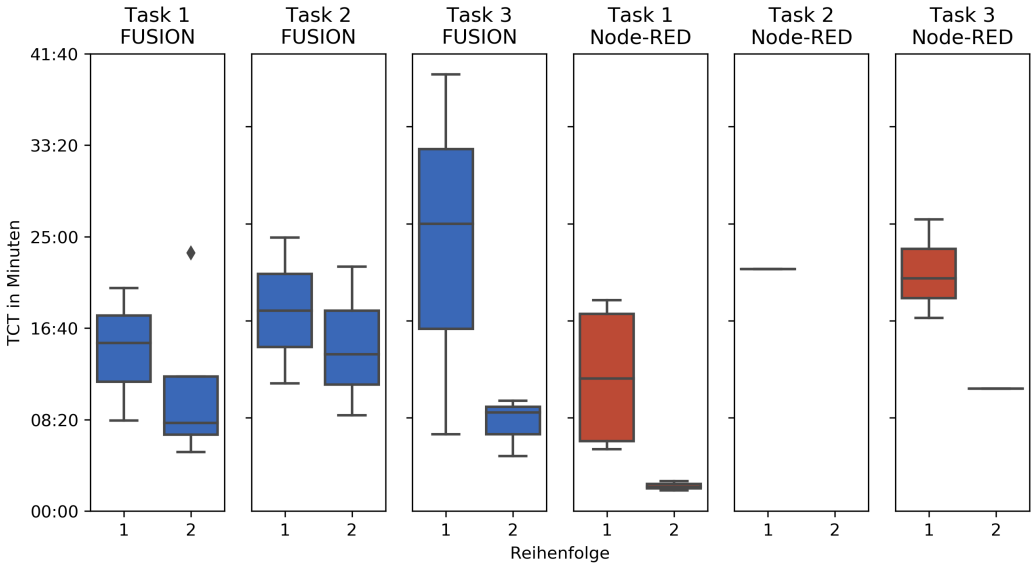

Abbildung 13: Taskzeiten erfolgreich abgeschlossener Aufgaben in Abhängigkeit der Testreihenfolge

Das Bereitstellen von ausführlich kommentierten Codebeispielen half den ProbandInnen, das System zu verstehen und die Aufgaben zu lösen. Dabei war es auch NutzerInnen ohne Programmierkenntnisse möglich, sich die Syntax aus den Beispielen herzuleiten. Insbesondere Personen mit geringer oder keiner Programmiererfahrung gaben an, dass sie diese Art der Hilfestellung gegenüber einer klassischen Dokumentation bevorzugten.

Um Usability und User Experience der in dieser Arbeit entwickelten Programmierschnittstelle zu verbessern, sollte das System mehr und ausführlichere Fehlermeldungen liefern. Darüber hinaus müssen einige technische Mängel, wie der gelgentliche Verbindungsverlust bestimmter Nodes, behoben werden.

9.2 Evaluation durch Experten

Um zu evaluieren, wie einfach sich die Programmierschnittstelle um neue Module erweitern lässt, wurde eine qualitative Nutzerstudie mit einem Probanden mit guten Programmierkenntnissen durchgeführt. Dieser bekam die Aufgabe, einen neuen Sensor (*MAX4466*-Mikrophonverstärker[53]) an das System anzubinden. Dazu gehörte das Installieren der API, das Programmieren der Sensornode und das Erstellen eines neuen Moduls für die Programmierschnittstelle.

Der Proband bekam die Gelegenheit, sich in das System einzulesen und Fragen

[53]https://www.adafruit.com/product/1063

zu stellen. Er wurde aufgefordert, auf positive und negative Aspekte, sowie Unklarheiten und Probleme während der Benutzung hinzuweisen.

Die Aufgabe konnte ohne erwähnenswerte Probleme erfolgreich gelöst werden und das Modul funktionierte einwandfrei.

Anschließend wurde der Proband in einem freien Interview über seine Meinung zum System befragt und um Verbesserungsvorschläge gebeten.

Bei der Installation des Systems gab es keine Probleme. Der Proband gab an, dass er sich schnell im Sourcecode zurechtfand und hob positiv hervor, dass dieser sehr gut strukturiert und kommentiert sei. Variablen, Klassen und Funktionen seien sinnvoll benannt und befänden sich da, wo er sie erwartete. Explizit wurden auch der übersichtliche Aufbau der `.ino`-Dateien und die Möglichkeit, Parameter beim Hochladen des Programms auf die Sensornode zu setzen, gelobt. Allerdings wurde angemerkt, dass eine graphische Benutzeroberfläche das Hochladen von Programmen aufgrund der zahlreichen Parameter des Upload-Skripts deutlich erleichtern würden.

Der Proband schlug vor, ein von ihm als *"Boilerplate-Generator"* bezeichnetes Tool zu integrieren, das die für das Erstellen von Modulen benötigten Verzeichnisse und Dateien automatisch anlegt.

9.3 Technische Evaluation

In einer technischen Evaluation soll die Praxistauglichkeit des in dieser Arbeit erstellten Systems überprüft werden. Wichtige Faktoren bei Sensornetzwerken sind die maximale Akkulaufzeit der Sensorknoten, die Reichweite der drahtlosen Kommunikation und die Robustheit der Module gegenüber Verbindungsverlust und Abstürzen.

Um die Akkulaufzeit zu überprüfen, wurde ein Versuchsaufbau angelegt, bei dem eine Node durch einen an ein *Battery Shield*[54] angeschlossenen, voll geladenen Lithium-Polymer-Akku mit Strom versorgt wurde. Der Akku liefert eine Spannung von 3,7 V und verfügt über eine Kapazität von 720 mAh. Die Node war mit einem *BH1750*[55]-Lichtintensitätssensor verbunden, den sie alle 10 Sekunden auslas und

[54] https://wiki.wemos.cc/products:d1_mini_shields:battery_shield
[55] https://www.mouser.com/ds/2/348/bh1750fvi-e-186247.pdf

das Ergebnis über *MQTT* an das Gateway schickte. Dieser speicherte beim Empfangen von Paketen der Node einen Zeitstempel in eine Log-Datei. Da die Node mit keiner anderen Stromquelle verbunden war, entspricht die Differenz zwischen dem ersten und dem letzten gespeicherten Zeitstempel der Akkulaufzeit.

Der Akku war nach 22 Stunden vollständig entladen, was einen durchschnittlichen Stromverbrauch von rund 33 mA ergibt. Dies deckt sich mit der Messung des Stromverbrauchs der selben Node mithilfe eines *Peaktech 6225A*-Labornetzteils, welches einen konstanten Stromverbrauch von 27 mA mit Peaks von bis zu 60 mA beim Auslesen des Sensors und Senden der Daten anzeigte.

Die Reichweite der drahtlosen Datenübertragung wurde gemessen, indem eine Node (*Wemos D1 mini*), die alle 100 Millisekunden ein Datenpaket von 2 Bytes über *MQTT* an das Gateway (*Raspberry Pi 3b*) schickte, in Schrittgeschwindigkeit vom Gateway weg bewegt wurde. Der Test fand im Freien bei trockenem Wetter statt und keine Hindernisse blockierten das Signal, zudem war keines der beiden Geräte mit einer externen Antenne ausgestattet. Die Übertragung stagnierte bei etwa 100 Metern Abstand und brach nach nach etwa 115 Metern vollständig ab. Die Distanzen wurden über die GPS-Positionierung eines *Google Nexus 5*-Smartphone[56] gemessen.

Der als Gateway verwendete *Raspberry Pi 3b* war während der Implementierung der Programmierschnittstelle ständig im Betrieb und lief über mehr als einen Monat hinweg durchgehend. Um die Robustheit der drahtlosen Verbindung zu testen, wurde bereits in einem frühen Entwicklungsstadium des Systems ein Feldexperiment durchgeführt. Dabei wurden mit fünf Sensorknoten gleichzeitig über einen Zeitraum von drei Tagen hinweg jede Minute Temperatur und Luftfeuchtigkeit in drei Laborräumen gemessen. Zu diesem Zeitpunkt wurde noch das in Kapitel 7.3.1 beschriebene, selbst erstellte Nachrichtenprotokoll verwendet und es kam bei drei der fünf Sensoren zu gelegentlichen Verbindungsverlusten.

Ein weiteres Feldexperiment wurde nach Abschluss der Implementierung, ergo bereits mit einem System, welches das *MQTT*-Protokoll verwendet, durchgeführt. Dabei wurden mit zwei Sensorknoten zwei Tage lang alle fünf Minuten Temperatur, Luftfeuchtigkeit und Lichtintensität im selben Labor wie beim ersten Test gemes-

[56] https://www.gsmarena.com/lg_nexus_5-5705.php

sen (siehe Abbildung 14). Es kam dabei während des Versuchs zu keinem einzigen Verbindungsverlust.

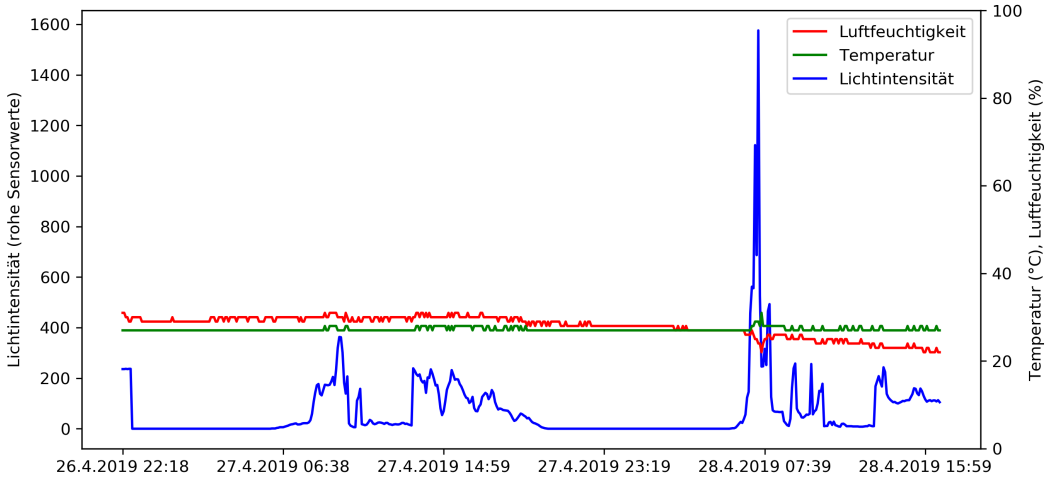

Abbildung 14: Messdaten des Feldexperiments

Aus technischer Sicht ist das im Rahmen dieser Arbeit entwickelte Sensornetzwerk im Hinblick auf Reichweite und Robustheit der Verbindung also durchaus geeignet für den Einsatz in der Praxis. Durch das Anbringen von externen Antennen an Nodes und Gateway kann die Reichweite und Qualität der drahtlosen Datenübertragung weiter verbessert werden. Die Akkulaufzeit der Nodes könnte durch die Implementierung des *"Deep Sleep"*-Modus so weit erhöht werden, dass diese auch über einen längeren Zeitraum hinweg autark funktionieren (Foxworth, 2017).

10 Ergebnisse und Ausblick

In dieser Arbeit wurde, wie in Kapitel 7 beschrieben, eine End-User-Programming-Schnittstelle und ein dazugehöriges Sensornetzwerk auf Basis einschlägiger Literatur (zusammengefasst in Kapitel 4) und einer nutzerzentrierten Anforderungserhebung (siehe Kapitel 5) implementiert.

Eine Evaluation der Programmierschnittstelle aus technischer und nutzerzentrierter Sicht wurde durchgeführt, deren Ergebnisse sind in Kapitel 9 zu finden.

10.1 Designempfehlungen für EUP-Schnittstellen

Aus den im Verlauf dieser Arbeit gesammelten Erfahrungen, sowie den Ergebnissen der Evaluation des Systems, ergaben sich zahlreiche Erkenntnisse über den Aufbau von Programmierschnittstellen für Endnutzer. Im Folgenden werden auf diese Erkenntnisse aufbauende Empfehlungen für das Design von End-User-Programming-Schnittstellen, präsentiert. Dabei wird sowohl auf technische Aspekte, als auch auf die Usability des Systems eingegangen.

Erwartungen erfüllen Die Nutzerstudie hat bestätigt, dass AnwenderInnen oft eine subjektive Vorstellung davon haben, wie ein System funktioniert und welche Werkzeuge zum Lösen eines Problems benötigt werden. Stimmt der Aufbau des Systems nicht mit diesem mentalen Modell überein, so kann es dazu kommen, dass Funktionen nicht gefunden werden. Somit empfiehlt es sich, das Hintergrundwissen der Zielgruppe bei der Benennung von Variablen, Klassen und Funktionen zu berücksichtigen. Außerdem ist es wichtig, jene Funktionalität, die von NutzerInnen erwartet wird, vollständig zu implementieren.

Transparenz Den ProbandInnen der Nutzerstudie war bei der Verwendung von *Node-RED* häufig nicht klar, wie die im System verlaufenden Nachrichten aufgebaut sind und welchen Weg diese nahmen. Daher ist es wichig, insbesondere bei komplexen Bestandteilen eines Systems, den momentanen Systemzustand, sowie den Verlauf von Daten im System, zu kommunizieren. Durch aussagekräftige und nachvollziehbare Fehlermeldungen und Warnun-

gen sollten NutzerInnen angeleitet werden, wo Fehler aufgetreten sind und wie diese behoben werden können.

Modularer Aufbau Durch den modularen Aufbau einer Programmierschnittstelle können technisch anspruchsvolle Komponenten gekapselt werden, sodass weniger erfahrene NutzerInnen mit diesen nicht in Kontakt kommen, während fortgeschrittene NutzerInnen der Zugang dazu trotzdem nicht verwehrt wird. Neben der dadurch gewonnenen Übersichtlichkeit hat ein modularer Aufbau auch den Vorteil einer hohen Flexibilität. So können beispielsweise ohne großen Aufwand Komponenten des Systems ausgetauscht oder hinzugefügt werden.

Codebeispiele Die ProbandInnen der Nutzerstudie nutzten die bereitgestellten Codebeispiele intensiv und entwickelten durch das Lesen von funktionierendem Code ein Verständnis für das System. Einige gaben im Interview nach der Studie an, dass sie diese Art der Hilfestellung gegenüber einer klassischen Dokumentation bevorzugen. Somit ist es bei Programmierschnittstellen für NutzerInnen mit wenig Programmiererfahrung empfehlenswert, ausführlich kommentierte Codebeispiele für häufige Anwendungsfälle mitzuliefern.

Aufbau auf etablierte Systeme Aus technischer Sicht ist es sinnvoll, so weit wie möglich auf bestehende Systeme und Schnittstellen aufzubauen. Neben dem geringeren Implementierungsaufwand bietet dies den Vorteil, dass NutzerInnen, die mit diesen Systemen bereits vertraut sind, der Einstieg erleichtert wird. Darüber hinaus trägt dies auch zur Modularität des Systems bei. Durch die Verwendung geläufiger Kommunikationsschnittstellen kann die Programmierschnittstelle auch leichter an andere Systeme angebunden werden, wodurch der Funktionsumfang erhöht wird.

Insgesamt ergänzen diese Ergebnisse die von Myers & Stylos (2016) ausgearbeiteten Heuristiken zur API-Usabilty, an welchen man sich beim Design einer Programmierschnittstelle allgemein orientieren sollte.

10.2 Limitierungen und Ausblick

Wie die Evaluation der Programmierschnittstelle ergab, weist diese noch ernsthafte Usabilityprobleme auf, welche behoben werden sollten, bevor eine weitere, größer angelegte Nutzerstudie durchgeführt wird, die auf statistisch Aussagekräftige Ergebnisse abzielt.

Eine graphische Oberfläche, die das Hochladen von Programmen auf die Nodes erleichtert, sowie der bei der Evaluation des Systems gewünschte *Boilerplate-Generator*, sollten hinzugefügt werden. Denkbar wäre auch ein Wizard, der NutzerInnen bei der Installation des Systems anleitet und über den globale Einstellungen vorgenommen werden können, ohne Konfigurationsdateien manuell zu bearbeiten.

Neben der Integration von Sicherheitsfeatures, die explizit kein Bestandteil dieser Arbeit war, und der Optimierung der Nodes hinsichtlich ihres Stromverbrauchs, sollte das Sensornetzwerk in einer Feldstudie unter realistischen Bedingungen auf seine Verwendbarkeit in der Praxis überprüft werden.

Durch die Verwendung eines Chips mit mehr Rechenleistung als Gateway könnten Sensordaten durch hardwarebeschleunigte Algorithmen, beispielsweise für Sensorfusion, Bildverarbeitung oder Machine Learning, verarbeitet werden. Dies würde auch die Integration von Sensorik mit großer Datenbandbreite, wie zum Beispiel Kameras oder Spektographen Langer et al. (2016) ermöglichen, wodurch das System ein größeres Einsatzgebiet abdecken würde.

Während die Sensordaten über Python-Programme bereits jetzt einfach ausgelesen, verarbeitet und weitergeleitet werden können, wäre die Möglichkeit, über eine standardisierte Schnittstelle, auf diese Werte zugreifen zu können, wünschenswert. So könnte beispielsweise eine REST[57]-Schnittstelle angebunden werden, über die mit HTTP-Requests Daten ausgelesen und Aktoren angesteuert werden können. Zusätzlich wäre es mit dieser Art des Zugriffs denkbar, über den Request Daten bereits in einem bestimmten Format (zum Beispiel Einheit, Größenordnung), versehen mit bestimmten Metadaten (zum Beispiel den Zeitpunkt der Messung), oder eine Kombination aus verschiedenen Daten anzufordern. Ein ähnliches Konzept wurde von Wimmer (2018) für Dateisysteme vorgeschlagen.

[57] https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest

Literatur

Arduino Day 2019: Thank you 659 times! (2019, März). Zugriff am 2019-04-09 auf https://blog.arduino.cc/2019/03/21/arduino-day-2019-thank-you-659-times/

Ashton, K. (2009, Juni). *That 'Internet of Things' Thing - 2009-06-22 - Page 1 - RFID Journal.* Zugriff am 2019-04-09 auf https://www.rfidjournal.com/articles/view?4986,

Baldwin, C. Y. & Clark, K. B. (2000). *Design Rules: The power of modularity*. MIT Press. (Google-Books-ID: oaBOuo4mId8C)

Bari, N., Mani, G. & Berkovich, S. (2013, Juli). Internet of Things as a Methodological Concept. In *2013 Fourth International Conference on Computing for Geospatial Research and Application* (S. 48–55). doi: 10.1109/COMGEO.2013.8

Blackwell, A. F. (2006). Psychological Issues in End-User Programming. In *End User Development* (S. 9–30). Springer, Dordrecht. Zugriff am 2018-07-18 auf https://link.springer.com/chapter/10.1007/1-4020-5386-X_2 doi: 10.1007/1-4020-5386-X_2

Brooke, J. (1996). SUS - A quick and dirty usability scale. , 8.

Brooks, F. (1986). No Silver Bullet – Essence and Accident in Software Engineering. , 16.

Brown, E. (2016, September). *21 Open Source Projects for IoT.* Zugriff am 2019-04-15 auf https://www.linux.com/news/21-open-source-projects-iot

Budiu, R. (2018, März). *Between-Subjects vs. Within-Subjects Study Design.* Zugriff am 2019-05-01 auf https://www.nngroup.com/articles/between-within-subjects/

Chong, C.-Y. & Kumar, S. (2003, August). Sensor networks: Evolution, opportunities, and challenges. *Proceedings of the IEEE, 91* (8), 1247–1256. Zugriff am 2019-04-15 auf http://ieeexplore.ieee.org/document/1219475/ doi: 10.1109/JPROC.2003.814918

Coleman, A. (2017, April). *Europe's Emerging IoT Scene And Six Startups To Watch.* Zugriff am 2019-04-09 auf https://www.forbes.com/sites/alisoncoleman/2017/04/18/europes-emerging-iot-scene-and-six-startups-to-watch/

Costabile, M. F., Fogjli, D., Fresta, G., Mussio, P. & Piccinno, A. (2003). Building Environments for End-user Development and Tailoring. In *Proceedings of the 2003 IEEE Symposium on Human Centric Computing Languages and Environments* (S. 31–38). Washington, DC, USA: IEEE Computer Society. Zugriff am 2019-04-16 auf http://dl.acm.org/citation.cfm?id=1153917.1153963

Costabile, M. F., Fogli, D., Mussio, P. & Piccinno, A. (2006). End-User Development: The Software Shaping Workshop Approach. In *End User Development* (S. 183–205). Springer, Dordrecht. Zugriff am 2018-07-18 auf https://link.springer.com/chapter/10.1007/1-4020-5386-X_9 doi: 10.1007/1-4020-5386-X_9

D1 mini [WEMOS Electronics]. (2018, Dezember). Zugriff am 2019-02-06 auf https://wiki.wemos.cc/products:d1:d1_mini#technical_specs

Day, M., Turner, G. & Drozdiak, N. (2019, April). Amazon Workers Are Listening to What You Tell Alexa. *Bloomberg*. Zugriff am 2019-04-14 auf https://www.bloomberg.com/news/articles/2019-04-10/is-anyone-listening-to-you-on-alexa-a-global-team-reviews-audio

Dertouzos, M. L. (1998). *What Will Be: How the New World of Information Will Change Our Lives*. DIANE Publishing Company.

Diamond, H. & Hinman, W. S. (1940, August). An automatic weather station. *Journal of Research of the National Bureau of Standards*, *25* (2), 133. Zugriff am 2019-04-15 auf https://nvlpubs.nist.gov/nistpubs/jres/25/jresv25n2p133_A1b.pdf doi: 10.6028/jres.025.036

Dougherty, D. (2012, Juli). The Maker Movement. *Innovations: Technology, Governance, Globalization*, *7* (3), 11–14. Zugriff am 2019-04-25 auf http://www.mitpressjournals.org/doi/10.1162/INOV_a_00135 doi: 10.1162/INOV_a_00135

Fangohr, H. (2004). A Comparison of C, MATLAB, and Python as Teaching Languages in Engineering. In T. Kanade et al. (Hrsg.), *Computational Science - ICCS 2004* (Bd. 3039, S. 1210–1217). Berlin, Heidelberg: Springer Berlin Heidelberg. Zugriff am 2019-04-29 auf http://link.springer.com/10.1007/978-3-540-25944-2_157 doi: 10.1007/978-3-540-25944-2_157

Foxworth, T. (2017, März). *Making the ESP8266 Low-Powered With Deep Sleep.* Zugriff am 2019-04-11 auf https://www.losant.com/blog/making-the-esp8266-low-powered-with-deep-sleep

Giezeman, W. (o. J.). *The Things Network – An Interview with Wienke Giezeman on IoT Business | Farnell element14.* Zugriff am 2019-04-09 auf https://de.farnell.com/things-network-interview-iot-business

Greenfield, A. (2010). *Everyware: The Dawning Age of Ubiquitous Computing*. New Riders. (Google-Books-ID: noMNgMcZvL0C)

Grill, T., Polacek, O. & Tscheligi, M. (2012). Methods Towards API Usability: A Structural Analysis of Usability Problem Categories. In *Proceedings of the 4th International Conference on Human-Centered Software Engineering* (S. 164–180). Berlin, Heidelberg: Springer-Verlag. Zugriff am 2019-04-25 auf http://dx.doi.org/10.1007/978-3-642-34347-6_10 doi: 10.1007/978-3-642-34347-6_10

Hart, J. K. & Martinez, K. (2006, Oktober). Environmental Sensor Networks: A revolution in the earth system science? *Earth-Science Reviews*, *78* (3-4), 177–191. Zugriff am 2019-04-14 auf https://linkinghub.elsevier.com/retrieve/pii/S0012825206000511 doi: 10.1016/j.earscirev.2006.05.001

Hartmann, F. & Mietzner, D. (2017). The Maker Movement - Current Understanding and Effects on Production. , 29.

Henderson, A. & Kyng, M. (1992). Design at Work. In J. Greenbaum & M. Kyng (Hrsg.), (S. 219–240). Hillsdale, NJ, USA: L. Erlbaum Associates Inc. Zugriff am 2019-04-16 auf http://dl.acm.org/citation.cfm?id=125470.125427

Hussain, S., Schaffner, S. & Moseychuck, D. (2009, Mai). Applications of Wireless Sensor Networks and RFID in a Smart Home Environment. In *2009 Seventh Annual Communication Networks and Services Research Conference* (S. 153–157). doi: 10.1109/CNSR.2009.32

ISO 9241-210. (2010, März). Zugriff am 2019-04-18 auf http://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/05/20/52075.html

Jiang, L. (o. J.). How Do People Evaluate the Usability of APIs. , 5.

Ko, A. J., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., … Wiedenbeck, S. (2011, April). The State of the Art in End-user Software Engineering. *ACM Comput. Surv.*, *43* (3), 21:1–21:44. Zugriff am 2018-05-15 auf http://doi.acm.org/10.1145/1922649.1922658 doi: 10.1145/1922649.1922658

Krajewski, A. (2017, Juni). *User Centred IoT-Design.* Zugriff am 2019-04-09 auf https://medium.com/the-state-of-responsible-internet-of-things-iot/andreakrajewski-aff52af1e065

Krueger, R. A. (2014). *Focus Groups: A Practical Guide for Applied Research*. SAGE Publications. (Google-Books-ID: tXpZDwAAQBAJ)

Kruger, J. & Dunning, D. (1999). Unskilled and Unaware of It: How Difficulties in Recognizing One's Own Incompetence Lead to Inflated Self-Assessments. , 17.

Kushner, D. (2011, Oktober). *The Making of Arduino.* Zugriff am 2019-04-09 auf https://spectrum.ieee.org/geek-life/hands-on/the-making-of-arduino

Lampkin, V. (2012, März). *What is MQTT and how does it work with WebSphere MQ? (Application Integration Middleware Support Blog)* [CT915]. Zugriff am 2019-04-05 auf https://www.ibm.com/developerworks/community/blogs/aimsupport/entry/what_is_mqtt_and_how_does_it_work_with_websphere_mq

Langer, F., Hohenleutner, M., Schmid, C. P., Poellmann, C., Nagler, P., Korn, T., … Huber, R. (2016, Mai). Lightwave-driven quasiparticle collisions on a subcycle timescale. *Nature*, *533* (7602), 225–229. Zugriff am 2019-04-18 auf http://www.nature.com/articles/nature17958 doi: 10.1038/nature17958

Langtangen, H. P. (Hrsg.). (2008). *Python Scripting for Computational Science*. Berlin, Heidelberg: Springer Berlin Heidelberg. Zugriff am 2019-04-29 auf https://doi.org/10.1007/978-3-540-73916-6_1 doi: 10.1007/978-3-540-73916-6_1

Lewis, J. R. (2018, Dezember). Measuring Perceived Usability: The CSUQ, SUS, and UMUX. *International Journal of Human–Computer Interaction*, *34* (12), 1148–1156. Zugriff am 2019-04-24 auf https://www.tandfonline.com/doi/full/10.1080/10447318.2017.1418805 doi: 10.1080/10447318.2017.1418805

Lieberman, H., Paternò, F., Klann, M. & Wulf, V. (2006, Januar). End-User Development: An Emerging Paradigm. In *End User Dev.* (Bd. 9, S. 1–8). doi: 10.1007/1-4020-5386-X_1

List of Arduino boards and compatible systems. (2019, April). Zugriff am 2019-04-09 auf https://en.wikipedia.org/w/index.php?title=List_of_Arduino_boards_and_compatible_systems&oldid=891229699 (Page Version ID: 891229699)

Lämmel, R. & Peyton Jones, S. (2003, März). Scrap Your Boilerplate: A Practical Design Pattern for Generic Programming. In (Bd. 38, S. 26–37). doi: 10.1145/604174.604179

MacLean, A., Carter, K., Lövstrand, L. & Moran, T. (1990). User-tailorable Systems: Pressing the Issues with Buttons. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (S. 175–182). New York, NY, USA: ACM. Zugriff am 2019-04-16 auf http://doi.acm.org/10.1145/97243.97271 doi: 10.1145/97243.97271

Mann, S. (o. J.). *Wearable Computing*. Zugriff am 2019-04-11 auf https://www.interaction-design.org/literature/book/the-encyclopedia-of-human-computer-interaction-2nd-ed/wearable-computing

Mattern, F. & Flörkemeier, C. (2010). From the Internet of Computers to the Internet of Things. In K. Sachs, I. Petrov & P. Guerrero (Hrsg.), *From Active Data Management to Event-Based Systems and More* (Bd. 6462, S. 242–259). Berlin, Heidelberg: Springer Berlin Heidelberg. Zugriff am 2019-04-09 auf http://link.springer.com/10.1007/978-3-642-17226-7_15 doi: 10.1007/978-3-642-17226-7_15

Mayhew, D. J. (1992). *Principles and Guidelines in Software User Interface Design*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Meyer, B. (1988). Object-Oriented Software Construction. , 1406.

Miller, L. (1981, Februar). Natural language programming: Styles, strategies, and contrasts. *IBM Systems Journal*, *20*, 184–215. doi: 10.1147/sj.202.0184

The mind behind Linux | Linus Torvalds. (2016, Mai). Zugriff am 2019-04-17 auf https://www.youtube.com/watch?v=o8NPllzkFhE

Moore, G. (1965). *Cramming more components onto integrated circuits.* Zugriff am 2019-04-25 auf https://drive.google.com/file/d/0By83v5TWkGjvQkpBcXJKT1I1TTA/view?usp=sharing&usp=embed_facebook

Myers, B. A. & Stylos, J. (2016, Mai). Improving API usability. *Communications of the ACM, 59* (6), 62–69. Zugriff am 2018-12-04 auf http://dl.acm.org/citation.cfm?doid=2942427.2896587 doi: 10.1145/2896587

Mørch, A. (1995, August). *Three Levels of End-User Tailoring: Customization, Integration, and Extension.* Zugriff am 2019-04-16 auf https://www.researchgate.net/profile/Anders_Morch/publication/228090876_Three_levels_of_end-user_tailoring_Customization_integration_and_extension/links/02e7e51f831231aa8c000000/Three-levels-of-end-user-tailoring-Customization-integration-and-extension.pdf

Nardi, B. A. (1993). *A Small Matter of Programming: Perspectives on End User Computing* (1st Aufl.). Cambridge, MA, USA: MIT Press.

Nielsen, J. (1994, April). *10 Heuristics for User Interface Design: Article by Jakob Nielsen.* Zugriff am 2019-04-29 auf https://www.nngroup.com/articles/ten-usability-heuristics/

Nielsen, J. (2000, März). *Why You Only Need to Test with 5 Users.* Zugriff am 2019-04-30 auf https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/

Nielsen, J. & Molich, R. (1990). Heuristic Evaluation of User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (S. 249–256). New York, NY, USA: ACM. Zugriff am 2019-04-29 auf http://doi.acm.org/10.1145/97243.97281 doi: 10.1145/97243.97281

Norman, D. A. & Draper, S. W. (1986). *User Centered System Design; New Perspectives on Human-Computer Interaction*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc.

OASIS. (2014, November). *Foundational IoT Messaging Protocol, MQTT, Becomes International OASIS Standard | OASIS.* Zugriff am 2019-04-08 auf https://www.oasis-open.org/news/pr/foundational-iot-messaging-protocol-mqtt-becomes-international-oasis-standard

Pane, J. F. & Myers, B. A. (2006). More Natural Programming Languages and Environments. In *End User Development* (S. 31–50). Springer, Dordrecht. Zugriff am 2018-07-18 auf https://link.springer.com/chapter/10.1007/1-4020-5386-X_3 doi: 10.1007/1-4020-5386-X_3

Paternò, F. (1999). *Model-Based Design and Evaluation of Interactive Applications*. Springer Science & Business Media. (Google-Books-ID: zCyXcL3YcjwC)

Pipek, V. & Kahler, H. (2006). Supporting Collaborative Tailoring. In *End User Development* (S. 315–345). Springer, Dordrecht. Zugriff am 2018-07-18 auf https://link.springer.com/chapter/10.1007/1-4020-5386-X_15 doi: 10.1007/1-4020-5386-X_15

Porter, J., Arzberger, P., Braun, H.-W., Bryant, P., Gage, S., Hansen, T., … Williams, T. (2005, Juli). Wireless Sensor Networks for Ecology. *BioScience*, *55* (7), 561–572. Zugriff am 2019-04-08 auf https://academic.oup.com/bioscience/article/55/7/561/306750 doi: 10.1641/0006-3568(2005)055[0561:WSNFE]2.0.CO;2

Price, R. (2016, April). *Google's parent company is deliberately disabling some of its customers' old smart-home devices.* Zugriff am 2019-04-21 auf https://www.businessinsider.de/googles-nest-closing-smart-home-company-revolv-bricking-devices-2016-4

Pólya, G. (1920, September). Über den zentralen Grenzwertsatz der Wahrscheinlichkeitsrechnung und das Momentenproblem. *Mathematische Zeitschrift*, *8* (3), 171–181. Zugriff am 2019-04-30 auf https://doi.org/10.1007/BF01206525 doi: 10.1007/BF01206525

Rauer, M. (2011, April). *Quantitative Usablility-Analysen mit der System Usability Scale (SUS).* Zugriff am 2019-04-29 auf https://blog.seibert-media.net/blog/2011/04/11/usablility-analysen-system-usability-scale-sus/

Raymond, E. S. (2003, September). *The Art of Unix Programming.* Zugriff am 2019-04-10 auf http://www.catb.org/~esr/writings/taoup/html/

Seidel, D. (2013, August). *MQTT – Eine Einführung.* Zugriff am 2019-04-03 auf http://dennisseidel.de/mqtt-eine-einfuhrung/

Shapiro, S. S. & Wilk, M. B. (1965, Dezember). An Analysis of Variance Test for Normality (Complete Samples). , 22.

Smith, D. C., Cypher, A. & Schmucker, K. (1996, September). Making Programming Easier for Children. *interactions*, *3* (5), 58–67. Zugriff am 2019-04-16 auf http://doi.acm.org/10.1145/234757.234764 doi: 10.1145/234757.234764

Stiemerling, O. (2000, Januar). *Component-based tailorability.* Zugriff am 2019-04-16 auf https://www.researchgate.net/publication/268322294_Component-based_tailorability

Tech Talk: Linus Torvalds on git - YouTube. (2007, Mai). Zugriff am 2019-04-17 auf https://www.youtube.com/watch?v=4XpnKHJAok8

Unreal Engine 4.0 Update Notes. (2014). Zugriff am 2019-04-11 auf https://docs.unrealengine.com/en-us/Support/Builds/ReleaseNotes/4_0/Updates

van Turnhout, K., Craenmehr, S., Holwerda, R., Menijn, M., Zwart, J.-P. & Bakker, R. (2013). Tradeoffs in Design Research: Development Oriented Triangulation. In *Proceedings of the 27th International BCS Human Computer Interaction Conference* (S. 56:1–56:6). Swinton, UK, UK: British Computer Society. Zugriff am 2019-04-30 auf http://dl.acm.org/citation.cfm?id=2578048.2578115

Was ist Industrie 4.0? (o. J.). Zugriff am 2019-04-25 auf https://www.plattform-i40.de/PI40/Navigation/DE/Industrie40/WasIndustrie40/was-ist-industrie-40.html

Weiser, M. (1991). *The Computer for the 21st Century.* Zugriff am 2019-04-15 auf https://www.lri.fr/~mbl/Stanford/CS477/papers/Weiser-SciAm.pdf

Whitley, K. N. & Blackwell, A. F. (1997). Visual Programming: The Outlook from Academia and Industry. In *Papers Presented at the Seventh Workshop on Empirical Studies of Programmers* (S. 180–208). New York, NY, USA: ACM. Zugriff am 2019-04-16 auf http://doi.acm.org/10.1145/266399.266415 doi: 10.1145/266399.266415

Wimmer, R. (2018). Files As Directories: Some Thoughts on Accessing Structured Data Within Files. In *Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming* (S. 166–170). New York, NY, USA: ACM. Zugriff am 2019-04-19 auf http://doi.acm.org/10.1145/3191697.3214323 doi: 10.1145/3191697.3214323

Wulf, V. & Golombek, B. (2001, Januar). Direct activation: A concept to encourage tailoring activities. *Behaviour & Information Technology*, *20* (4), 249–263. Zugriff am 2019-04-16 auf https://doi.org/10.1080/01449290110048016 doi: 10.1080/01449290110048016

Anhang

Anhang A: CD

```
Anhang - Masterarbeit Andreas Schmid/
├── Anforderungserhebung
│   ├── einverständniserklärung_fokusgruppe_01.pdf
│   └── fragebogen_fokusgruppe_01.pdf
├── Code
│   └── FUSION_EUP-release_alpha
│       ├── additional_files
│       ├── FUSION
│       ├── fusion.conf
│       ├── LICENSE
│       ├── nodes
│       │   ├── bh1750
│       │   ├── button
│       │   ├── dht11
│       │   ├── led
│       │   ├── libraries
│       │   ├── minimal
│       │   ├── mqtt
│       │   ├── pin
│       │   ├── pio
│       │   └── pir
│       ├── pio_compile.sh
│       ├── README.md
│       ├── setup.sh
│       └── test_scripts
├── Nutzerstudie
│   ├── Auswertung
│   ├── Codebeispiele
│   │   ├── Button.ipynb
│   │   ├── E-Mail.ipynb
│   │   ├── LED.ipynb
│   │   ├── Light Intensity (BH1750).ipynb
│   │   └── Temperature & Humidity (DHT11).ipynb
│   └── Dokumente
│       ├── fragebogen.pdf
│       ├── guide_fusion.pdf
│       ├── guide_nodered.pdf
│       ├── tasks_fusion.pdf
│       └── tasks_nodered.pdf
└── Verschriftlichung
    └── Masterarbeit Andreas Schmid.pdf
```