



TECHNISCHE HOCHSCHULE NÜRNBERG
GEORG SIMON OHM

Fakultät Informatik

**Entwicklung einer Grafischen
Oberfläche zur Erstellung von
Anwendungen im Internet der
Dinge**

unter Verwendung von Linked Data Schnittstellen

Bachelorarbeit im Studiengang Informatik

vorgelegt von

Thomas Wehr

Matrikelnummer 3114658

Erstgutachter: Prof. Dr. Thomas Fuhr

Zweitgutachter: Prof. Dr. Alexander Kröner

© 2020

Dieses Werk einschließlich seiner Teile ist **urheberrechtlich geschützt**. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Autors unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen sowie die Einspeicherung und Verarbeitung in elektronischen Systemen.

Hinweis: Diese Erklärung ist in alle Exemplare der Abschlussarbeit fest einzubinden. (Keine Spiralbindung)

Prüfungsrechtliche Erklärung der/des Studierenden

Angaben des bzw. der Studierenden:

Name: Wehr

Vorname: Thomas

Matrikel-Nr.: 3114658

Fakultät: Informatik

Studiengang: Informatik

Semester: Wintersemester



2019



Titel der Abschlussarbeit:

Entwicklung einer GUI für IoT-Systeme mit RDF-Graphen

Ich versichere, dass ich die Arbeit selbständig verfasst, nicht anderweitig für Prüfungszwecke vorgelegt, alle benutzten Quellen und Hilfsmittel angegeben sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Ort, Datum, Unterschrift Studierende/Studierender

Erklärung zur Veröffentlichung der vorstehend bezeichneten Abschlussarbeit

Die Entscheidung über die vollständige oder auszugsweise Veröffentlichung der Abschlussarbeit liegt grundsätzlich erst einmal allein in der Zuständigkeit der/des studentischen Verfasserin/Verfassers. Nach dem Urheberrechtsgesetz (UrhG) erwirbt die Verfasserin/der Verfasser einer Abschlussarbeit mit Anfertigung ihrer/seiner Arbeit das alleinige Urheberrecht und grundsätzlich auch die hieraus resultierenden Nutzungsrechte wie z.B. Erstveröffentlichung (§ 12 UrhG), Verbreitung (§ 17 UrhG), Vervielfältigung (§ 16 UrhG), Online-Nutzung usw., also alle Rechte, die die nicht-kommerzielle oder kommerzielle Verwertung betreffen.

Die Hochschule und deren Beschäftigte werden Abschlussarbeiten oder Teile davon nicht ohne Zustimmung der/des studentischen Verfasserin/Verfassers veröffentlichen, insbesondere nicht öffentlich zugänglich in die Bibliothek der Hochschule einstellen.

Hiermit genehmige ich, wenn und soweit keine entgegenstehenden Vereinbarungen mit Dritten getroffen worden sind,
 genehmige ich nicht,

dass die oben genannte Abschlussarbeit durch die Technische Hochschule Nürnberg Georg Simon Ohm, ggf. nach Ablauf einer mittels eines auf der Abschlussarbeit aufgebrachten Sperrvermerks kenntlich gemachten Sperrfrist

von 5 Jahren (0 - 5 Jahren ab Datum der Abgabe der Arbeit),

der Öffentlichkeit zugänglich gemacht wird. Im Falle der Genehmigung erfolgt diese unwiderruflich; hierzu wird der Abschlussarbeit ein Exemplar im digitalisierten PDF-Format auf einem Datenträger beigefügt. Bestimmungen der jeweils geltenden Studien- und Prüfungsordnung über Art und Umfang der im Rahmen der Arbeit abzugebenden Exemplare und Materialien werden hierdurch nicht berührt.

Ort, Datum, Unterschrift Studierende/Studierender

Kurzdarstellung

Das Ziel der Bachelorarbeit ist es, Aufbau und Einsatz von Geräten, aus dem Bereich Internet der Dinge (IoT), zu vereinfachen. Berücksichtigt werden hierbei nur IoT-Geräte, welche Daten zur Kommunikation über das Hypertext Transfer Protocol (HTTP) versenden und mit dem, vom World Wide Web Consortium (W3C) eingeführten, SSN/SOSA Vokabular beschreiben. Umgesetzt wird dies, durch die Implementierung einer grafischen Oberfläche auf der mit simplem Ziehen und Ablegen von Blöcken, das Application Programming Interface (API) eines IoT Systems bedient wird. Als konkretes Anwendungsszenario wird beispielhaft ein vorhandenes System zur Verwaltung eines Postbehälterkreislaufs genutzt. Anwender sollen in der Lage sein, ohne Fachwissen, durch Anordnen von grafischen Elementen Abfragen, in der Abfragesprache SPARQL, zu schreiben. Dabei sollen sie Bedingungen für den Postbehälterkreislauf festlegen können wie beispielsweise „Behälter ist an Poststelle X“, „Behälter hat Standort X verlassen“ oder „Behälterstandort seit Y Stunden unbekannt“. Zusätzlich sollen Anwender, abhängig von den definierten Bedingungen, Aktionen ausführen können. Zum Beispiel könnte die Anwendung eine Nachricht ausgeben, wenn eine Bedingung erfüllt ist. Diese Aufgaben werden abschließend in einer Evaluation von der Zielgruppe durchgeführt. Dabei wird die implementierte Anwendung hinsichtlich der Bedienbarkeit ohne oder mit nur wenig Fachwissen anhand der System Usability Scale, einem standardisiertem Fragebogen, bewertet. Die Anwendung erreicht hier einen überdurchschnittlichen Wert von 81.4.

Inhaltsverzeichnis

1 Einführung	1
1.1 Motivation	1
1.2 Problemstellung	2
1.3 Zielsetzung und Aufbau der Arbeit	3
2 Grundlagen	7
2.1 Semantic Web Technologien	7
2.2 Das Internet der Dinge	8
2.3 Visuelle und blockbasierte Programmiersprachen	10
2.4 Verwandte Arbeiten	11
2.5 Beispiel eines IoT-Systems	13
3 Blockbasierte Konfiguration eines IoT-Systems	17
3.1 Anforderungen	17
3.2 Architektur der Anwendung	19
3.3 Grafische Oberfläche basierend auf Blöcken	20
3.3.1 Implementierte Blöcke	23
3.3.2 Zusammensetzen der Blöcke	28
3.4 Übersetzung und Ausführung der Blöcke	29
4 Evaluierung	35
4.1 Aufbau der Testumgebung	35
4.1.1 Aufgaben	36
4.1.2 Fragebogen	39
4.2 Ergebnisse und Beobachtungen	41
4.3 Möglichkeiten der Blockprogramme im IoT-System des Postbehälterkreislaufs	42
4.4 Diskussion und Bewertung	44
5 Fazit und Ausblick	47
5.1 Zusammenfassung	47

5.2 Ausblick	48
Abbildungsverzeichnis	51
Tabellenverzeichnis	53
Listingverzeichnis	55
Literaturverzeichnis	57

Kapitel 1

Einführung

Im ersten Kapitel der Arbeit wird zunächst die Motivation und Problemstellung dargestellt. Anschließend wird das Testszenario vorgestellt und die Zielsetzung sowie die Forschungsfragen formuliert.

1.1 Motivation

Das Semantic Web spielt im Internet der Dinge (IoT) eine wichtige Rolle, denn im IoT-Bereich werden zunehmend mehr Daten übertragen. Ob Messwerte, berechnete Werte oder andere Beobachtungen, beim Austausch von zusammenhängenden Daten ist eine klare und einheitliche Struktur essentiell. Ebenso wichtig ist die genaue Beschreibung aller Sensoren, Akten und anderer IoT-Geräte. Zur Strukturierung und Beschreibung dieser Daten, bietet das Semantic Web eine Vielzahl von etablierten Ontologien. Die Verwendung eines solchen Schemas erleichtert nicht nur das Verständnis für Dritte, sondern ermöglicht auch eine einfache Wiederverwendung, Anpassung und Erweiterung der Daten (vgl. [Pfis11]). Zum Beispiel kann der gemessene Wert eines Temperatursensors, wenn er im durch das Sensor, Observation, Sample, and Actuator (SOSA) Vokabular (siehe [Sema17]) beschriebenen Format dargestellt wird, von jedem System das diese Ontologie kennt, ohne Anpassung genutzt werden.

Daten-Repräsentationen in diesem Format sind maschineninterpretierbar und für den Menschen leicht lesbar. Um allerdings Abfragen zu schreiben, welche gezielt nur bestimmte Daten aus dieser Darstellung zurückgeben, benötigt man Fachwissen. Üblicherweise wird hierfür die Abfragesprache SPARQL [Seab06] verwendet. Um langes Erlernen von diesem Fachwissen zu umgehen, zum Beispiel falls die Anwendung im

Rapid-Prototyping Umfeld genutzt wird und daher möglichst ohne Einarbeitung nutzbar sein soll, ist eine zusätzliche Schicht zur Abstraktion von SPARQL-Abfragen notwendig. Doch das alleinige Lesen von Daten reicht meist nicht aus. Die übertragenen Werte werden in der Regel vom IoT-System verwendet, um davon abhängige Aktionen auszuführen. Wie beispielsweise eine Benachrichtigung wenn ein bestimmter Sender in Reichweite eines Empfängers kommt. Daher gehört zur o.g. Abstraktionsschicht auch eine Lösung zur Nutzung der abgefragten Daten.

Im Rahmen dieser Arbeit wird demnach versucht, eine grafische Oberfläche zu implementieren, mit deren Hilfe IoT Systeme die Daten als RDF-Graphen übermitteln, konfiguriert werden. Diese Oberfläche soll, durch Abstraktion der Schnittstelle der IoT-Geräte, ohne Vorwissen nutzbar sein.

1.2 Problemstellung

Sensordaten die über Internet- und Webtechnologien abgerufen werden, tendieren zunehmend zu einer Schnittstelle, welche diese Daten in einem der Semantic-Web Standards ausgibt [Jara 14a] [Gyra 16] [Gyra 15]. Da jedoch jeder Hersteller seine eigene Plattform zur Konfiguration mitliefert, existiert eine Vielzahl in sich nicht konsistenter Oberflächen zur Konfiguration von IoT-Systemen mit gleicher Schnittstelle. Um Konsistenz in der Konfiguration zu bewahren oder Systeme unterschiedlicher Hersteller gemeinsam zu verwenden, müsste der Endbenutzer mit den rohen Daten arbeiten. Zum Verständnis dieser Daten benötigt man Kenntnisse des Resource Description Frameworks (RDF) (siehe [Rich 14]) und der verwendeten Vokabulare bzw. Ontologien. Wer eigene Abfragen formulieren möchte, benötigt zudem Übung in der Abfragesprache SPARQL. Falls Aktionen auf Basis der abgefragten Daten ausgeführt werden sollen, muss der Anwender zusätzlich eine Programmiersprache beherrschen. Dies erschwert den Aufbau und die Verwendung von IoT-Systemen.

Durch Abstraktion der Kommunikation mit einem solchem System, könnte das nötige Vorwissen minimiert werden. Hierzu soll jeder der gängigen Lesebefehle und eine Auswahl an Aktionen durch Anordnung von grafischen Elementen definiert werden können. So können alle Systeme, die eine in der Anwendung implementierte Schnittstelle anbieten, durch eine einheitliche Oberfläche bedient werden.

1.3 Zielsetzung und Aufbau der Arbeit

Ziel dieser Arbeit ist es, Aufbau und Einsatz von IoT-Geräten mit SSN/SOSA und HTTP Schnittstelle zu vereinfachen. Umgesetzt wird dies durch die Implementierung einer Abstraktionsschicht für die Kommunikation mit diesen Geräten und einer grafischen Oberfläche. Auf dieser soll mit simplem Ziehen und Ablegen von Puzzleteilen oder Blöcken, möglicherweise ähnlich wie in Googles Blockly¹ oder Microsofts MakeCode², die API eines IoT-Systems bedient werden können.

Dies wird beispielhaft für den Postbehälter-Kreislauf der Fraunhofer-Arbeitsgruppe für Supply-Chain-Services implementiert und getestet. Hier sind bereits verschiedene Bluetooth Sender mit den notwendigen Schnittstellen verbaut. Diese senden in einem festem Intervall Datenpakete. Ein Empfänger am Posteingang von jedem der 3 Fraunhofer Gebäude in Nürnberg sucht alle x Millisekunden nach diesen Bluetoothsignalen. Anhand der Stärke des Bluetooth-Signals wird ermittelt, ob sich die in Containern verbauten Sender in der Nähe eines Empfängers befinden. Dieser Aufbau beinhaltet eine Reihe an Konfigurationen, die über die Anwendung vorgenommen werden sollen:

- Zuordnung Empfänger — Gebäude
- Zuordnung MAC-Adresse — Container
- Aktion bei Eintreffen eines Containers
- Aktion bei Wegfahren eines Containers

Anwender sollen in der Lage sein, ohne Fachwissen und nur durch Anordnen von grafischen Elementen, SPARQL-Abfragen und JavaScript Code zu schreiben, um MAC-Adressen im System zu speichern, Bluetooth Daten auszulesen und ausgehend von abgefragten Daten Aktionen auszuführen. Diese Aufgaben werden in einer Evaluation von der Zielgruppe durchgeführt, um die implementierte Anwendung hinsichtlich der Bedienbarkeit ohne oder mit nur wenig Fachwissen zu bewerten. Ausführlichere Erläuterungen zu den Anforderungen sind in Kapitel 3.1 Anforderungen zu finden.

¹<https://developers.google.com/blockly/>

²<https://makecode.com/docs>

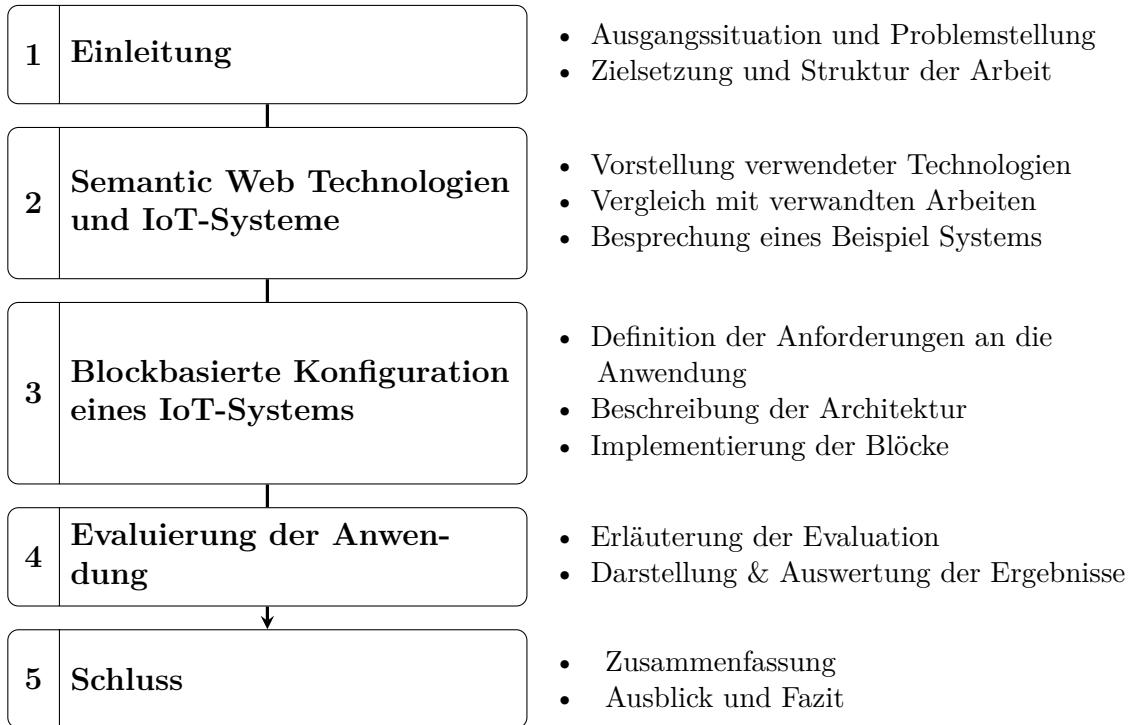


Abbildung 1.1: Aufbau der Arbeit

Hinsichtlich dieser Zielsetzung ergeben sich folgende Forschungsfragen:

F1 Welche existierenden Vorgehensweisen und Algorithmen sind für das gegebene Szenario hilfreich?

F2 Welcher Weg wurde beschritten und wie ist dieser zu bewerten?

F3 Wie ist die Oberfläche, hinsichtlich der Bedienbarkeit ohne Fachwissen, zu bewerten?

F4 Wo liegt Verbesserungspotential?

Die Arbeit ist zur Beantwortung der Forschungsfragen in insgesamt fünf Kapitel aufgeteilt (siehe Abbildung 1.1). Auf den Einleitungsteil folgt in Kapitel 2 zunächst eine Einführung in verwendete Technologien und das Internet der Dinge. Weiterhin werden im zweiten Kapitel verwandte Arbeiten vorgestellt und verglichen. Am Ende des Kapitels wird beispielhaft auf ein bestehendes IoT-System eingegangen.

Kapitel 3 stellt die entwickelte Anwendung vor, indem es deren Anforderungen und Oberfläche beschreibt. Des Weiteren wird die Implementierung des Block-Editors beschrieben.

Die Ergebnisse der Evaluierung der Anwendung werden in Kapitel 4 vorgestellt. Im Rahmen dessen erfolgt zunächst eine Beschreibung des Testaufbaus. Daraufhin werden die Ergebnisse und Beobachtungen aufgezeigt und anschließend diskutiert und bewertet.

Schließlich werden in Kapitel 5 die wesentlichen Erkenntnisse zusammengefasst, die Limitationen der Anwendung dargelegt und eine mögliche Entwicklung in der Zukunft aufgezeigt.

Kapitel 2

Grundlagen

In diesem Abschnitt werden Semantic Web Technologien, insbesondere RDF-Graphen, Linked Data und der Begriff „Internet of Things definiert“. Zudem werden visuelle Programmiersprachen erläutert. Der Abschnitt 2.4 Verwandte Arbeiten beschäftigt sich mit dem momentanen Stand der Technik und erörtert inwiefern verwendete Technologien für die im Rahmen dieser Arbeit entwickelten Anwendung geeignet sind. Das letzte Unterkapitel stellt ein existierendes IoT-System vor, welches von der Anwendung profitieren würde.

2.1 Semantic Web Technologien

In [Bern 01] formuliert Tim Berners-Lee das Ziel des Semantic Webs als Informationen des World Wide Webs mit eindeutigen Bedeutungen zu versehen, um die Arbeit zwischen Mensch und Maschine zu erleichtern. Also diese Informationen menschenlesbar und maschineninterpretierbar machen. Nötig hierfür ist es, einheitliche offene Standards für die Beschreibung von Informationen zu vereinbaren, sodass diese zwischen verschiedenen Anwendungen und Plattformen ausgetauscht und zueinander in Beziehung gesetzt werden können. Eine Zentrale Rolle in der Schaffung dieser Standards spielt das World Wide Web Consortium (W3C)¹.

Neben einigen anderen Standards wurde vom W3C das für diese Arbeit essentielle „Resource Description Framework“ (RDF) [Mill 98] definiert. Dieser offene Standard wird heute oft von IoT-Geräte Herstellern verwendet. Hier wird festgelegt wie Ressourcen beschrieben werden sollen. Eine Ressource kann hier jedes durch einen Uniform Resource Identifier (URI) eindeutig identifizierbare Objekt sein, wie zum Beispiel ein

¹<http://www.w3.org/>

```

1   <http://localhost/ble/>
2     <http://www.w3.org/TR/rdf-schema/type>
3     <http://www.w3.org/ns/sosa/Platform>.
4   <http://localhost/ble/>
5     <https://github.com/aharth/supercool/MacAddress>
6     "a0e6f8371731".

```

Listing 2.1: RDF Beispiel

Ort, ein Bluetooth-Sender oder Empfänger oder ein Container. Beschrieben wird so eine Ressource indem sie durch ein Prädikat, entweder mit einer weiteren Ressource oder mit einem Literal, also einem Wert wie zum Beispiel ein String, Integer oder Datum, verknüpft wird. So lassen sich Aussagen formulieren wie „`http://localhost/ble/` ist vom Typ Plattform“ (Listing 2.1 Zeile 1–3) oder „`http://localhost/ble/` hat die Mac-Adresse `a0e6f8371731`“ (Listing 2.1, Zeile 4–6). Eine solche Aussage ist immer ein Tripel bestehend aus Subjekt, Prädikat und Objekt. Dadurch, dass Ressourcen sowohl Subjekt als auch Objekt sein können, entstehen sehr große Mengen an vernetzten Daten. Diese Daten lassen sich auch als Graphen, mit den Ressourcen und Literalen als Knoten und den Prädikaten als Kanten, darstellen. Deshalb spricht man bei dieser Darstellung in der Regel von RDF-Graphen. Auf RDF-Graphen lassen sich Abfrage-Operationen in verschiedenen Sprachen ausführen. Die beliebteste, auch da vom W3C empfohlene, Sprache zur Erstellung solcher Abfragen ist SPARQL [The 13].

Durch SPARQL-Abfragen auf RDF-Graphen lässt sich gezielt nach bestimmten Informationen suchen. So wäre es zum Beispiel möglich, eine Abfrage zu formulieren, die anhand des RDF-Graphen aus Listing 2.1 heraus findet, von welchem Typ die Ressource mit der Mac-Adresse „`a0e6f8371731`“ ist. Werden RDF-Graphen über Hypertext Transfer Protocol (HTTP) abgerufen, spricht man von Linked Data Schnittstellen [Bern 06]. Die Verwendung von Linked Data Schnittstellen im IoT-Bereich ermöglicht eine einheitliche Datenverarbeitung. Durch konsistente, maschineninterpretabare Darstellung der Informationen können komplexe Abfragen ohne Anpassung auf allen Systemen die RDF-Graphen verwenden, durchgeführt werden. Somit lassen sich deren Daten leicht wiederverwenden, anpassen oder erweitern.

2.2 Das Internet der Dinge

Heutzutage ist die Internetnutzung nicht mehr nur auf Menschen die per Computer oder Smartphone auf das Internet zugreifen beschränkt, sondern erstreckt sich zuneh-

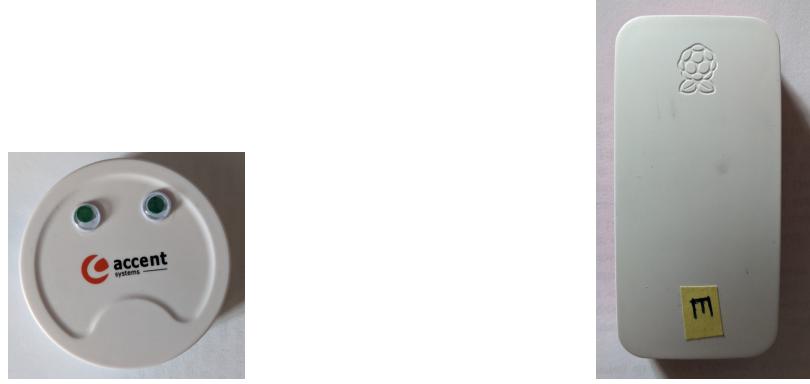


Abbildung 2.1: Beispiel für ein „Ding“, aus dem IoT-System „Postbehälterkreislauf“

mend auf Alltagsgegenstände. Beispielsweise sind Glühlampen, Kühlschränke oder Lieferwagen heute mit dem Internet verbunden. So war im Januar 2020 zum Beispiel die Anzahl der aktiven Internetnutzer bei über 4 Milliarden Menschen [Clem 20]. Gleichzeitig wird erwartet, dass die Anzahl der mit dem Internet verbundenen Geräte in 2020 auf über 30 Milliarden, und bis 2025 auf über 75 Milliarden steigt [Depa 19]. Bei einem solchem Trend ist es nur logisch, dass sich die Kommunikation über das Internet von nur Mensch-zu-Mensch zu Mensch-zu-Maschine oder Maschine-zu-Mensch und auch Maschine-zu-Maschine, beziehungsweise „Ding“ zu „Ding“ erweitert hat. Dinge können also selbstständig Informationen austauschen und verarbeiten und unabhängig Aktionen ausführen. Diese Form der Kommunikation von zwei oder mehreren Geräten über das Internet, ohne Interaktion durch den Menschen, ermöglicht das Internet der Dinge.

Zu den größten Anwendungsdomänen von IoT-Geräten gehören die Industrie, Smart Cities, der Energiesektor, die Medizin, Smart Homes und die Landwirtschaft [Perw 19]. Der Status einer Paketsendung zum Beispiel lässt sich heute, durch Verwendung von Lokalisierungssensoren die über das Internet der Dinge verbunden sind, durchgängig abfragen. Kommt ein, mit einem solchem Sensor ausgestattetes, Paket an einer Landestation an, kann dies ohne Handeln vom Menschen registriert werden und weitere Aktionen vom System autonom durchgeführt werden. Damit dies reibungslos funktioniert, müssen zwei Voraussetzungen erfüllt sein. Einerseits muss das Paket eindeutig identifizierbar sein, denn andere Lokalisierungssensoren oder IoT-Geräte sollen nicht die selbe Aktion auslösen. Das System muss genau erfassen können, um welches Paket es sich handelt. Des Weiteren müssen die verschiedenen Geräte nicht nur Daten untereinander austauschen können, sondern auch in der Lage sein, diese zu verarbeiten.

Datenpakete müssen also eine vordefinierte Darstellung haben, sodass zum Beispiel das Paket seinen Standort nicht in Universal Transverse Mercator (UTM) Koordinaten übermittelt, obwohl das System nur mit Dezimalgraden umgehen kann. Um solche Konflikte zu vermeiden bietet es sich an, die Technologien des Semantic Web, wie Linked Data Schnittstellen, im IoT Umfeld zu verwenden [Jara 14b] [Gyra 15] [Gyra 16]. Deshalb steigt aktuell auch einer der größten Anbieter für IoT-Lösungen, Microsoft Azure, auf die Verwendung von Zusammenhängenden Daten um: [Mann 19] definiert eine auf JavaScript Object Notification-Linked Data (JSON-LD), einer Erweiterung des RDF-Modells [Spor 14], basierende Sprache zur Beschreibung von IoT-Daten. Abbildung 2.1b zeigt beispielhaft einen Bluetooth-Empfänger und Abbildung 2.1a einen Bluetooth-Sender eines Linked Data IoT-Systems am Fraunhofer Institut in Nürnberg. Eine genauere Beschreibung dieses Systems ist im Abschnitt 2.5 Beispiel eines IoT-Systems zu finden.

2.3 Visuelle und blockbasierte Programmiersprachen

Im IoT Umfeld kommt eine Vielzahl an Sensoren zum Einsatz. Dadurch entsteht eine große Menge an Daten. Um diese Datenmenge optimal und uneingeschränkt zu nutzen, reichen einfache Optionsmenüs nicht aus. Zur Umsetzung eigener Ideen, muss der Anwender ausführbaren Code generieren. Die herkömmliche Art dies zu tun, ist die Verwendung einer Hochsprache. Jedoch bringt das Erlernen einer Hochsprache für Einsteiger eine Reihe an Herausforderungen mit sich [Gome 07] [Jenk 02]. Um diese zu umgehen, sind seit den frühen 60er Jahren die verschiedensten visuellen Programmiersprachen (englisch visual programming language, VPL) entstanden [Kell 05]. Alle mit dem Ziel das Generieren von Programmen Einsteiger-freundlicher zu gestalten. [Myer 90] definiert visuelle Programmierung als ein System, dass es dem Nutzer erlaubt, ein Programm in 2- (oder mehr-) dimensionaler Weise zu beschreiben. In VPLs werden Programme also, statt durch Text, mittels grafischen Elementen und deren Anordnung definiert. Durch Verwendung von VPLs muss der Anwender keine Syntax oder Kommandos erlernen und kann mit sehr viel geringerem Vorwissen Programme erstellen [Alta 16]. Daher setzen viele Hersteller von IoT-Systemen auf den Einsatz von visuellen Programmiersprachen (siehe Unterkapitel 2.4). [Ray 17] untersucht visuelle Programmiersprachen im IoT Umfeld und schreibt, IoT scheint durch die reibungslosen Abläufe von VPLs gestärkt zu werden.

Blockbasierte Programmiersprachen sind eine Form der VPLs. Hier wählt der Anwender passende Blöcke aus und steckt diese durch Ziehen und Ablegen wie Puzzleteile aneinander, um Programme zu erstellen. Die Entwicklungsumgebung verhindert hierbei, dass nicht zueinander passende Blöcke verbunden werden. Überwiegend kommen blockbasierte Entwicklungsumgebungen zum Einsatz, um jüngeren Kindern das Programmieren beizubringen. Die bekannteste Anwendung hierfür ist Scratch [Resn 09]. [Joao 19] vergleicht blockbasierte Programmiersprachen. Hierfür unterteilt Joao *et. al.* diese in Kategorien. Am besten für Robotik geeignet seien Micro:bit² und mBlock³, beides Umgebungen die Googles Blockly Bibliothek⁴ verwenden. Dies bestätigt die in Unterkapitel 2.4 gezeigte vorherrschende Verwendung von Blockly im IoT Umfeld.

2.4 Verwandte Arbeiten

Zur Vereinfachung der Erstellung von Abfragen sowie zur Konfiguration von IoT-Systemen mit grafischen Elementen existieren bereits verschiedene Lösungswege. Tools, welche zum Beispiel nur die Erstellung von SPARQL-Abfragen erleichtern sollen, nutzen unterschiedliche Ansätze. SPARKLIS [Ferr 17] beispielsweise bietet ein Tool, das dem Endbenutzer hilft, SPARQL-Abfragen zu schreiben. Umgesetzt wird dies, indem das Tool Abfragen auf Formulierungen abbildet, die der natürlichen Sprache sehr ähnlich sind und mit Vorschlägen durch die Erstellung einer Abfrage leitet. Solche Formulierungen in der natürlichen Sprache sind aber oft zu umfangreich oder unpräzise, um eine wirklich reibungsfreie Benutzererfahrung zu ermöglichen. Deshalb bietet es sich an, die Eingaben etwas zu beschränken. Daher erstellen viele Arbeiten SPARQL-Abfragen mit Hilfe von Graphen-basierten Oberflächen. [Russ 08] zum Beispiel stellt NITELIGHT vor. Hier werden per Drag and Drop Graphen gezeichnet, die dann durch das Tool in Abfragen übersetzt werden können. [Bors 06] beschreibt mit SPARQLViz ein ähnliches Tool, mit dem Unterschied, dass Graphen nicht per Drag and Drop, sondern durch Ausfüllen einer Sequenz an Formularen entstehen. Diese Arbeiten versuchen möglichst viele Ontologien abzubilden. Eine weitere Vereinfachung in der Bedienbarkeit lässt sich durch Beschränkung der Ontologien erreichen. Dies versucht [Fadh 07] mit OntoVQL, einem Tool das nur Abfragen für die OWL-S Ontologie schreibt.

²<https://makecode.microbit.org/>

³<https://www.mblock.cc/>

⁴<https://developers.google.com/blockly/>

Neben dem Graph-basiertem Ansatz verwenden die meisten Arbeiten Blöcke, die sich wie Puzzleteile aneinander setzen lassen, um die Erstellung von Abfragen zu erleichtern. [Ceri17] zum Beispiel, beschreibt SPARQLBlocks. Hier wird das gesamte SPARQL-Vokabular auf Blöcke abgebildet. Wie die meisten Block-basierten Lösungen verwendet SPARQLBlocks Googles Blockly zur Erstellung der Blöcke. Diese Blöcke haben Graph-basierenden Lösungen gegenüber den Vorteil, dass durch die Form der Verbindungen sofort ersichtlich ist, welche Teile aneinander passen und welche nicht. So lassen sich Missverständnisse durch gleich aussehende Knoten und Kanten vermeiden. Der Blockly Ansatz erscheint also intuitiver und leichter verständlich. Doch da eine Abbildung des Gesamten SPARQL-Spektrums nicht benötigt wird, lassen sich diese Blöcke durch Einschränken auf bestimmte Ontologien noch weiter vereinfachen.

Außerdem verwendet die Mehrheit der Arbeiten die sich mit der Konfiguration von IoT-Geräten befassen Blockly. [Ghaz16] und [Wein17] beispielsweise verwenden beide Blockly, um unterschiedliche, einarmige Industrieroboter zu konfigurieren. Beide Anwendungen haben einen hohen Fokus auf Benutzbarkeit ohne Programmiererfahrung. [Ghaz16] entwickelt eine Anwendung, welche als Lehrplattform für Robotik Programmierung dienen soll. Diese ist deshalb besonders Einsteiger-freundlich. Und auch das in [Wein17] entwickelte Tool wurde erfolgreich von Menschen ohne Programmiererfahrung getestet. [Culi15] hat mit Wyliodrin eine Anwendung entwickelt, die automatisch Blöcke für vorher unbekannte IoT Hardware generiert. Was auf dem sich ständig weiterentwickelnden Markt sehr hilfreich sein kann, bringt aber im Fall von einem festgelegtem Modell wie hier, nur Ungenauigkeiten. Deshalb konzentriert sich diese Arbeit auf vordefinierte Blöcke.

In vielen Arbeiten verschmelzen die beiden Aufgaben „Abfragen erstellen“ und „IoT-System konfigurieren“ zu einem. [Toml17] verwendet Datenflussgraphen, deren Knoten mit Blockly spezifiziert sind, um Anwendungsanforderungen von Industriemaschinen zu modellieren. Dadurch kann ein Großteil der Arbeit, bei Installation oder Austausch der beschriebenen Geräte, nicht mehr nur durch Entwickler, sondern auch durch den Endbenutzer ausgeführt werden. Auch in [Quoc12] kommen sowohl Blockly als auch Datenflussgraphen zum Einsatz. Genau wie in [Toml17] werden Abfragen durch Blöcke definiert. Deren Ergebnisse werden dann verwendet, um mit Datenflussgraphen entweder deren Darstellung oder Aktionen festzulegen. Diese Vermischung von zwei Oberflächen ist vor allem dann sinnvoll, wenn der Anwender bereits mit Datenflussgraphen vertraut ist. Sie führt aber auch zu einer steileren Lernkurve. [Bak18]

stellt Smart Block vor, eine grafische Entwicklungsumgebung, die der im Rahmen dieser Arbeit entwickelten Anwendung sehr ähnlich ist. Mit ihrer Hilfe lassen sich Samsung SmartThings⁵ konfigurieren. Der Nutzer hat hier die Möglichkeit, allein durch Verwendung von Blockly-Blöcken, ereignisgesteuerte und bedingungsabhängige Aktionen festzulegen. Allerdings arbeitet SmartThings nicht mit Semantic-Web Standards, sondern konzentriert sich auf die Geräte des Herstellers.

Insgesamt scheint Blockly als geeignete Technologie zur Konfiguration von IoT-Systemen mit RDF-Graphen. Durch die Verwendung von Blöcken wird die Oberfläche robust und, dank deren intuitiver Darstellung als Puzzleteile, einfach in der Bedienung. Indem der Funktionsumfang der Anwendung auf Hardware, welche Linked Data Schnittstellen verwendet eingeschränkt wird, kann dies zusätzlich verstärkt werden. Ein konsistenter Gebrauch von Blöcken durch die gesamten Anwendung hindurch, ermöglicht eine flache Lernkurve. Zusätzlich helfen die Blöcke die Fehleranfälligkeit gering zu halten.

2.5 Beispiel eines IoT-Systems

Im folgenden wird ein am Fraunhofer Institut in Nürnberg existierendes IoT-System beschrieben. Dieses besteht aus einem Raspberry Pi 3 Model B⁶, im folgenden „Basestation“ genannt und fünf Raspberry Pi Zeroe Ws⁷, die „Accesspoints“.

Der Aufbau des Systems ist in Abbildung 2.2 dargestellt. Die Accesspoints sind in verschiedenen Räumen eines Stockwerks verteilt und die Basestation befindet sich im Zentrum. Aufgabe der Accesspoints ist es, in regelmäßigen Abständen nach Bluetooth Geräten in der Nähe zu scannen. Am Ende eines jeden Scans werden alle empfangenen Bluetooth-Daten in einen RDF-Graphen geschrieben. Dieser Graph kann per GET-Request angefragt werden. Demzufolge verwendet das System eine Linked Data Schnittstelle. Durch diese lassen sich an jedem Accesspoint aktuelle Daten aller Bluetooth-Geräte in Reichweite einsehen.

Die Basestation hat zum einen die Aufgabe ein WLAN-Netzwerk zu errichten mit dem sich alle Accesspoints verbinden können. Daher auch die Zentrale Lage der Basestation in Abbildung 2.2. Zum anderen ist es Aufgabe der Basestation, die Informationen der

⁵[https://www.samsung.com/de/support/mobile-devices/
alles ueber smartthings/](https://www.samsung.com/de/support/mobile-devices/alles ueber smartthings/)

⁶<https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>

⁷<https://www.raspberrypi.org/products/raspberry-pi-zero-w/>

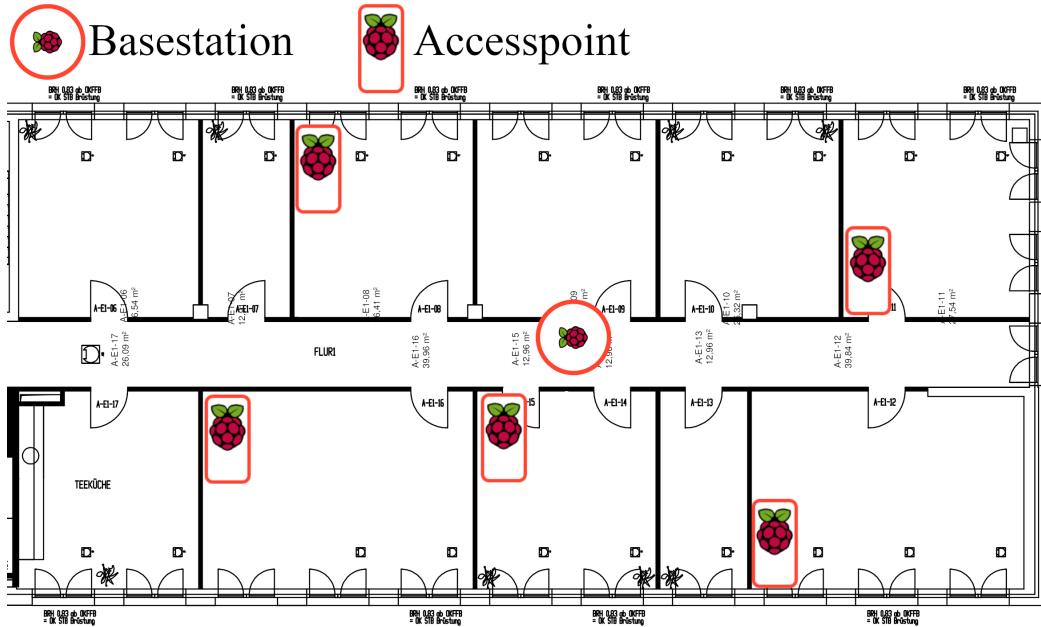


Abbildung 2.2: Aufbau des Beispiel-Systems

Accesspoints zu bündeln. Hierfür sendet sie in einem festgelegtem Intervall, im Bereich von wenigem hundert Millisekunden, Anfragen an alle verbundenen Accesspoints. Die Antwortgraphen werden daraufin zu einem großen Graphen zusammengeführt. Der Ergebnisgraph dieser Operation ist dann per GET-Request auf einer Adresse der Basestation abrufbar. Dadurch muss der Anwender nicht zunächst überprüfen welche Accesspoints aktuell verbunden sind, um dann an jeden einzelne Anfragen zu schicken. Stattdessen kann er, mit einer Anfrage alle Bluetooth-Daten des Stockwerks als RDF-Graph erhalten. Empfangen drei oder mehr Accesspoints Signale von Bluetooth-Geräten, kann die Basestation den Standort des sendenden Bluetooth-Geräts berechnen. Ein auf der Basestation implementierter Algorithmus berechnet dazu anhand der Signalstärken den Abstand von Accesspoint zu Bluetooth-Gerät. Anschließend wird anhand der drei geringsten Entfernung mittels Trilateration, einem Verfahren zur Positionsbestimmung eines Punktes beruhend auf Entfernungsmessungen, der Sender lokalisiert.

Zusätzlich ist die Basestation über Mobilfunk mit dem Internet verbunden. Demzufolge kann der Anwender von überall auf alle angebotenen Schnittstellen des Systems zugreifen. Eine Übersicht aller implementierten Übertragungstechnologien des Systems ist in Abbildung 2.3 zu sehen.

Das Fraunhofer System verwendet RDF-Graphen, um Bluetooth-Daten zu beschreiben. Dies ermöglicht zwar einerseits eine einfache Verarbeitung der Daten für Maschi-

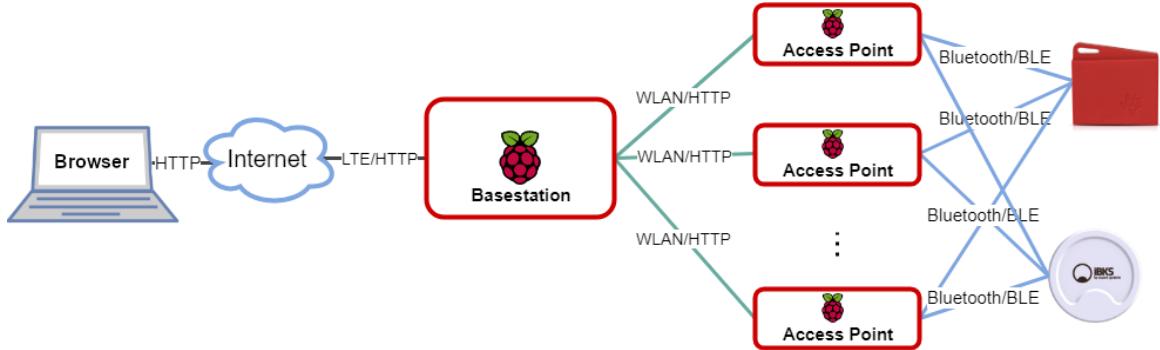


Abbildung 2.3: Übertragungstechnologien des Beispiel Systems

nen, macht es aber andererseits auch unhandlicher für den Endanwender. Dementsprechend macht es Sinn eine, wie die im Rahmen dieser Arbeit entwickelte Anwendung, hier einzusetzen. Mit deren Hilfe könnte der Nutzer zum Beispiel Nachrichten ausgeben, wenn sich ein Bluetooth Gerät in einem bestimmten Raum befindet oder die Daten der RDF-Graphen in einer Tabelle anzeigen lassen.

In diesem Kapitel wurde zunächst der grundlegende Aufbau von RDF-Graphen beschrieben und an einem Beispiel aus dem Anwendungsbereich erläutert. Es wurde erklärt, dass RDF Graphen eine maschineninterpretierbare Darstellung von Informationen ermöglichen. Daher sind diese auch im IoT-Umfeld weit verbreitet. Anschließend wurde der Begriff „Internet der Dinge“ definiert. Er beschreibt ein Internet in dem Maschinen ohne menschliches Eingreifen miteinander kommunizieren und interagieren. Es wurden die größten Anwendungsbereiche aufgezählt und ein dem Testfall aus dieser Arbeit sehr ähnliches Beispiel besprochen. Zudem wurden Visuelle Programmiersprachen beschrieben. Durch Abstraktion der textuellen Ebene einer Programmiersprache in grafische Elemente ermöglichen sie eine anfängerfreundliche Erstellung von ausführbaren Programmen. Als besonders geeignet für das IoT Umfeld wurden die blockbasierten Programmiersprachen beschrieben. Dies ist eine Form von visueller Programmierung. Hier steckten die Anwender Blöcke aneinander, um Code zu generieren. Nachfolgend wurden Arbeiten vorgestellt und verglichen, die sich mit verwandten Themen beschäftigen. In diesem Abschnitt wurde noch einmal besprochen warum blockbasierte Programmiersprachen und insbesondere Blockly als geeignete Technologie erwählt wurde. Zuletzt wurde ein System aus der Fraunhofer Gesellschaft vorgestellt. Dieses System arbeitet mit Linked Data Schnittstellen. Dahingehend wurde erläutert, weshalb ein Einsatz der im Rahmen dieser Arbeit entwickelten Anwendung hier sinnvoll scheint.

Kapitel 3

Blockbasierte Konfiguration eines IoT-Systems

Im folgenden Kapitel wird zunächst auf Anforderungen an die entwickelte Anwendung eingegangen. Dabei werden drei zentrale Aufgaben beschrieben. Anschließend wird die Architektur der Anwendung erläutert. Im Mittelpunkt steht hier die grafische Oberfläche mit dem blockbasierten Editor. Diese wird in den Unterkapiteln 3.3 und 3.4 im Detail besprochen.

3.1 Anforderungen

Die Hauptanforderung an die Anwendung ist eine möglichst einfache Bedienung. Um eine lange Einarbeitung zu vermeiden, soll das nötige Vorwissen so gering wie möglich gehalten werden. Dies soll durch eine grafische Oberfläche realisiert werden. Als Teil dieser wird eine blockbasierte Programmierumgebung implementiert. Getestet

```
1 <http://localhost/ble/54c0e531629/>
2   <http://www.w3.org/TR/rdf-schema/type>
3   <https://github.com/aharth/supercool/beacons#CC2650SensorTag>;
4 <http://localhost/ble/54c0e531629/>
5   <https://github.com/aharth/supercool/MacAddress>
6   "546c0e531629";
7 <http://localhost/ble/54c0e531629/>
8   <https://github.com/aharth/supercool/RSSI>
9   "-71";
10 <http://localhost/ble/54c0e531629/>
11   <https://github.com/aharth/supercool/LocalName>
12   "CC2650 SensorTag" .
```

Listing 3.1: BLE Informationen im RDF Graph

wird die entwickelte Anwendung abschließend mit einem System zur Überwachung des Postbehälterkreislaufs des Instituts für Integrierte Schaltungen der Fraunhofer Gesellschaft in Nürnberg. Hierfür wird ein bereits existierendes IoT-System des Instituts genutzt. Dieses überwacht das Ein- und Austreffen der Container für die interne Hauspost. Dazu sind in jedem von drei Gebäuden Bluetooth-Empfänger und in jedem Postcontainer Bluetooth-Sender verbaut. Die Sender schicken regelmäßig Nachrichten mit Informationen über den Sender. Dies geschieht in einem festgelegtem Intervall von wenigen hundert Millisekunden. Die Empfänger empfangen diese Nachrichten in einem Bereich von ungefähr 30 Metern. Anschließend werden die empfangenen Daten in RDF-Graphen geschrieben. Via HTTP-GET-Anfrage an die Empfänger können diese Graphen abgerufen werden.

Zur Bewertung der Programmierumgebung soll eine Reihe von Aufgaben durch Anordnen von Blöcken umsetzbar sein. Diese werden mit realen Daten des oben beschriebenen IoT-Systems getestet:

1. Benennung von Bluetooth Sendern und Empfängern:

Der Anwender soll in der Lage sein, Mac-Adressen mit Namen zu versehen. Beispielsweise könnte er bestimmen, dass der Sender mit der Mac-Adresse „dc-ba4b9f9b16“ die Bezeichnung „gelber Container“ erhält oder der Empfänger „0c1f14e90d98“ den Namen „Poststelle Gebäude 1“. Somit muss der Nutzer im weiteren Verlauf nicht mehr mit den unhandlichen Mac-Adressen arbeiten, sondern kann leichter einprägsame Namen vergeben.

2. Abfragen aller von den Empfängern empfangenen Bluetooth Daten:

Die Bluetooth-Empfänger empfangen regelmäßig sogenannte „Advertisement Packets“. Diese beinhalten immer die Mac-Adresse des Senders, einen eindeutigen Identifikator des Gerätes und etwas Platz für Hersteller-spezifische Informationen [Jeon 18]. Die Empfänger bieten eine Schnittstelle an, um RDF-Graphen mit diesen Informationen abzurufen. Listing 3.1 zeigt wie ein solcher Graph aussehen kann. Abgebildet sind hier in Zeile 2 bis 4 die Mac-Adresse, die Signalstärke und der Name im lokalen Netzwerk. Diese Informationen sollen dem Nutzer durch die Anwendung anschaulich präsentiert werden

3. Mitteilungen abhängig von diesen Daten anzeigen:

Die abgefragten Daten aus 2. sollen verwendet werden, um Mitteilungen im Browser anzuzeigen. Hierzu soll der Anwender Bedingungen definieren können. Ist eine Bedingung erfüllt wird eine vom Nutzer definierte Browsermitteilung

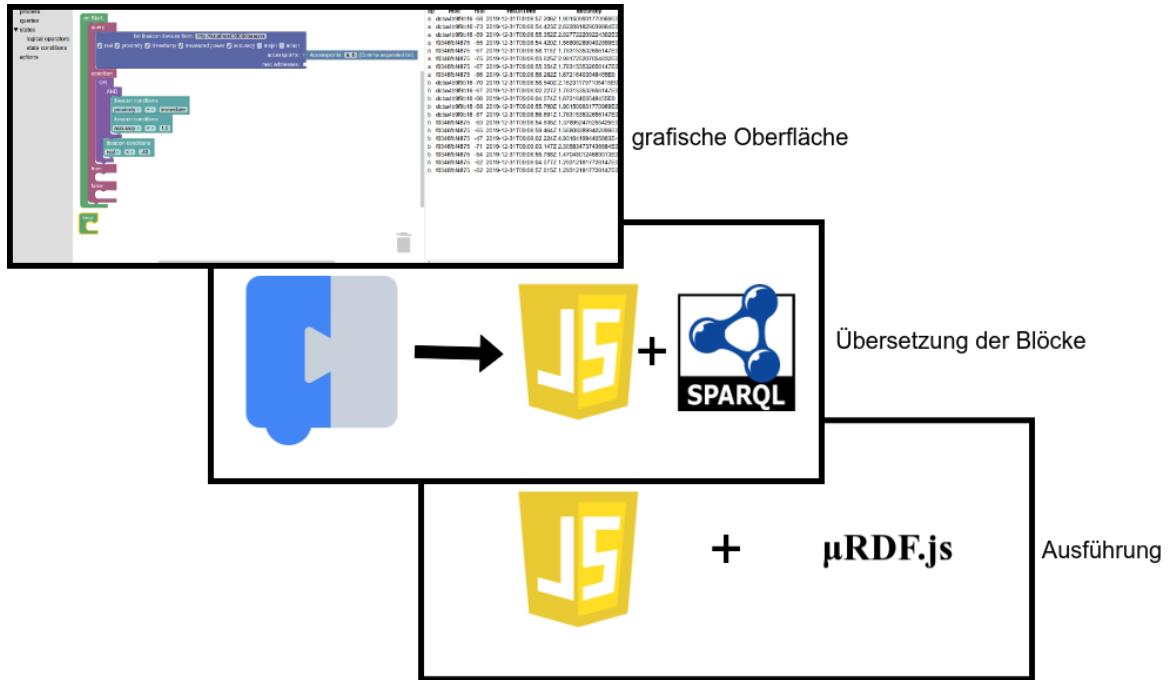


Abbildung 3.1: Schichten der Anwendung

angezeigt. Um Bedingungen zu erstellen, wird immer eine abgefragte Größe mit einem der Vergleichsoperatoren kleiner, gleich oder größer mit einem Wert verbunden. Eine denkbare Bedingung wäre „Signalstärke > -50“. Hiermit lässt sich ermitteln ob ein Sender in unmittelbarer Nähe des Empfängers ist. Mehrere Bedingungen sollen sich mit den logischen Operatoren UND- und ODER verbinden lassen. Dadurch könnte man obige Bedingung, durch UND-Operator, mit einer Bedingung verknüpfen, welche nur einen bestimmten Sender betrachtet, wie beispielsweise „Mac Adresse = ,0c1f14e90d98““. Der Anwender wäre somit in der Lage eine Mitteilung anzuzeigen, wenn ein vorher definierter Empfänger eine Signalstärke größer als ein Wert x zu einem Sender hat. Also zum Beispiel wenn der „gelbe Container“ sich in unmittelbarer Nähe von „Poststelle Gebäude 1“ befindet.

3.2 Architektur der Anwendung

Die im Rahmen dieser Arbeit entwickelte Anwendung muss Aufgaben aus unterschiedlichen Bereichen bewältigen. Bei der Wahl der verwendeten Programmiersprachen und Technologien, sowie dem Aufbau der Anwendung spielen diese eine wichtige Rolle. Im folgenden wird erläutert, welche Methodiken bei der Entwicklung ausgewählt wurden

und weshalb diese eingesetzt wurden. Einerseits muss die Anwendung in der Lage sein, HTTP-Anfragen an die IoT-Systeme zu senden. Dies ist mit einer Vielzahl an Programmiersprachen möglich. Andererseits muss sie im Stande sein, die Antworten zu verarbeiten. Genauer ausgedrückt, benötigt sie einen SPARQL-Prozessor zum Abfragen der empfangenen RDF-Graphen. Das W3C bietet hierfür eine umfangreiche Liste¹ an Möglichkeiten mit unterschiedlichen Stärken. Außerdem soll die Anwendung eine grafische Programmierumgebung bieten. Wie im Abschnitt 2.4 Verwandte Arbeiten erläutert, eignet sich Blockly hierfür am Besten. Die Blockly-Bibliothek ist komplett in JavaScript implementiert. Das Versenden von HTTP-Anfragen, um mit dem System zu kommunizieren, ist in JavaScript auch ohne Weiteres möglich. Also wurde nach einem ebenfalls in JavaScript implementiertem RDF-Prozessor gesucht. [Char 17] implementiert mit µRDF einen SPARQL-Prozessor, welcher speziell für die Verwendung auf Mikrocontrollern implementiert ist. Damit eignet sich µRDF besonders gut für IoT-Geräte. Um das Aufsetzen und Konfigurieren eines Servers zu umgehen, wird die Anwendung mittels NodeJS² ausgeführt. NodeJS bietet eine Plattform, um unter anderem Webserver in JavaScript zu realisieren. Damit muss auch für serverseitige Operationen nicht auf weitere Programmiersprachen zurückgegriffen werden.

Um diese Technologien zu vereinen, ist die Anwendung in drei Schichten eingeteilt (siehe Abbildung 3.1). Die erste Schicht bildet die grafische Oberfläche, also den für den Anwender sichtbaren Teil. Diese bietet einen blockbasierten Editor, sowie eine Tabelle zur Darstellung von Abfrageergebnissen. Zudem werden hier Mitteilungen angezeigt. Die nächste Schicht übersetzt die Elemente der visuellen Programmierumgebung. Sie überführt also die Blockly Blöcke in JavaScript Code und SPARQL Abfragen. Die letzte Schicht führt den generierten Code und die erstellten Abfragen aus. Dies beinhaltet unter anderem die Kommunikation mit den IoT-Systemen per HTTP-Request und das Ausführen von SPARQL-Abfragen an die RDF-Graphen. Die folgenden Abschnitte beschreiben im Detail wie diese drei Schichten aufgebaut sind.

3.3 Grafische Oberfläche basierend auf Blöcken

Der wichtigste Teil für die Bedienbarkeit der Anwendung ist die grafische Oberfläche. Über sie bedient der Anwender das gesamte Programm. Deren Gestaltung ist also essentiell für die gewünschte Einfachheit der Benutzung. [Gali 07] ist eines der

¹<https://www.w3.org/wiki/SparqlImplementations>

²<https://nodejs.org/>

Tabelle 3.1: Bestimmung der Zielgruppe

Frage	Antwort
Wie profitiert die Zielgruppe von der Nutzung der Anwendung?	Erleichterte Nutzung von IoT-Systemen mit Linked Data Schnittstellen
Wie alt ist die Zielgruppe?	Ab ca. 14 Jahren
Wie gut kann die Zielgruppe lesen?	Durchschnittliches Leseniveau ist ausreichend
Wird die Anwendung selbstständig oder als Teil einer Lehrveranstaltung genutzt?	Selbstständige Nutzung
Wie viel Erfahrung haben die Nutzer mit der Technologie?	Wenig bis keine Erfahrung, grundlegendes Verständnis des IoT-Systems
Haben sie bereits ähnliche Anwendungen gesehen?	Nein
Wie schnell müssen sie die Nutzung der Anwendung erlernen?	Die Anwendung soll innerhalb weniger Minuten zu verstehen sein
Wird die Anwendung einmalig oder wiederholt verwendet?	Wiederholt, viele IoT-Systeme werden täglich verwendet

bekanntesten Werke, welches sich mit Prinzipien der Gestaltung von grafischen Oberflächen beschäftigt. Hier wird deutlich, dass sich die Gestaltung stark nach der Art der Oberfläche und der Zielgruppe richtet. Ein Teil des Google Blockly Teams hat daher eine eigene Arbeit mit Blockly-spezifischen Richtlinien veröffentlicht. In [Past 17] definieren Pasternak *et. al.* je acht zentrale Fragen, um Zielgruppe und Umfang der Blockly Anwendung zu bestimmen. Anhand dieser wird in Tabelle 3.1 die Zielgruppe festgelegt. Anschließend werden im darauf folgenden Abschnitt die Fragen bezüglich des Umfangs beantwortet.

Anhand der Fragen und Antworten aus Tabelle 3.1 lassen sich die wichtigsten Aspekte der Zielgruppe wie folgt beschreiben. Sie hat wenig bis keine Erfahrung mit Semantic Web Technologien, versteht aber was das zu bedienende IoT-System tut. Durch die Anwendung ist sie in der Lage, ohne zuvor bereits ähnliche Anwendungen gesehen

Tabelle 3.2: Bestimmung des Anwendungsumfangs

1. Was können die Nutzer mit der Anwendung tun?
2. Was können die Nutzer mit der Anwendung nicht tun?
3. Was sind die Ziele der Nutzers?
4. Was bringt jeder einzelne Block dem Nutzer?
5. Kann die selbe Aktion mit mehr als einem Block durchgeführt werden?
6. Wie lange braucht der Nutzer um benötigte Blöcke oder Kategorien zu finden?
7. Kann der Anwender verstehen was jeder Block tut indem er ihn ansieht?
8. Kann der Anwender verstehen was jeder Block tut indem er ihn ausführt?

zu haben, IoT-Systeme mit Linked Data Schnittstellen zu nutzen. Dies wird von der Zielgruppe innerhalb weniger Minuten erlernt.

Die Tabelle 3.2 zeigt die von Pasternak *et. al.* [Past 17] definierten Fragen zur Bestimmung des Umfangs der Arbeit. Diese werden ausführlich im folgenden Abschnitt beantwortet.

Zunächst wird der Entwickler aufgefordert, sich die Fragen zu stellen, was die Anwender mit der Anwendung tun können (Frage 1) und was nicht (Frage 2). Ziel dieser Anwendung ist es IoT-Systeme zu nutzen. Eine ausführlichere Beschreibung dieses Ziels findet sich in Abschnitt 3.1. Sie ist aber nur in der Lage IoT-Systeme, welche mit RDF-Graphen kommunizieren, zu bedienen (Frage 1). Was der Nutzer mit der Anwendung also nicht tun kann (Frage 2), ist IoT-Systeme, die Daten in anderen Formaten übertragen, zu nutzen.

Als nächstes fragen Pasternak *et. al.* in [Past 17] was die Ziele des Nutzers seien (Frage 3). Nutzer dieser Anwendung wollen Datenabfragen formulieren und Aktionen abhängig von den abgefragten Daten ausführen. Zum einen sollen Anwender in der Lage sein, jedes vom IoT-System angebotene Datum einzeln oder in Kombination abzufragen. Zum anderen sollen mit diesen abgefragten Daten Bedingungen formuliert werden können, um bei Eintreten oder nicht Eintreten dieser Bedingungen Aktionen auszuführen.

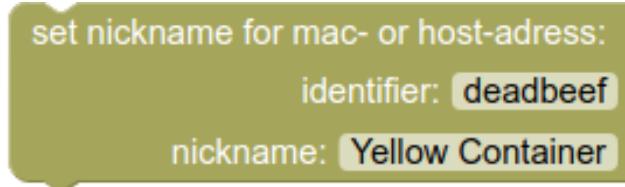


Abbildung 3.2: Der AliasBlock

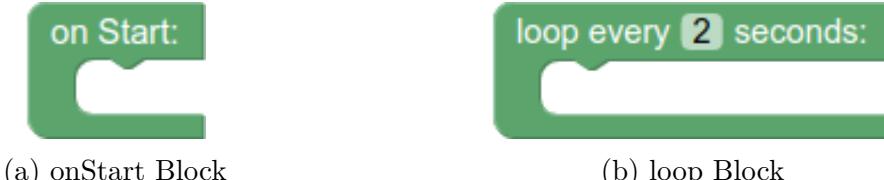
Die Fragen 4 und 5 drehen sich um die Gestaltung der Blöcke. Daher werden im nächsten Abschnitt die fünf Blockkategorien der Anwendung vorgestellt und jeweils am Ende der Kategorie die beiden Fragen dazu diskutiert. Fragen 6 bis 8 richten sich an den Anwender. Diese werden in Kapitel 4 Evaluierung besprochen.

3.3.1 Implementierte Blöcke

Die Blöcke der Anwendung sind in fünf Kategorien eingeteilt: Konfiguration, Kontrollfluss, Abfragen, Bedingungen und Aktionen.

- **Konfiguration:** Erstellung von Konfigurationsaufgaben wie Benennung von Geräten
- **Kontrollfluss:** Bestimmt wann und wie oft eine Abfrage, Konfiguration oder Aktion ausgeführt wird
- **Abfragen:** Erstellung von SPARQL-Abfragen
- **Bedingungen:** Ermöglicht das Formulieren von Bedingungen basierend auf abgefragten Daten
- **Aktionen:** Definition der auszuführenden Aktion bei Erfüllen oder Nichterfüllen der Bedingungen

Zum Zeitpunkt der Veröffentlichung dieser Arbeit ist nur ein **Konfigurationsblock** implementiert: Der „Aliasblock“ (siehe Abbildung 3.2). Mit diesem ist es möglich, Namen an Geräte, beziehungsweise Mac-Adressen, zu vergeben. Somit müssen die Anwender im weiteren Verlauf nicht mehr mit deren unhandlichen Adressen arbeiten, sondern kann Geräte stattdessen über prägnantere, selbst definierte Namen ansprechen. Denkbar wäre zum Beispiel einem Sender, welcher in einem gelben Container Postcontainer eingebaut ist, einfach den Namen „gelber Container“ zu geben. Weitere mögliche Konfigurationsblöcke werden in Unterkapitel 5.2 Ausblick besprochen.



(a) onStart Block

(b) loop Block

Abbildung 3.3: Die Kontrollflussblöcke

Was bringt dieser Block dem Nutzer? Durch den Aliasblock können die Anwender jedem der verwendeten Geräte einen Namen zuweisen. Somit können, bei der Erstellung von Programmen mit dem blockbasierten Editor, aussagekräftigere Namen an Stelle der Mac-Adressen genutzt werden.

Kann diese Aktion mit mehr als einem Block durchgeführt werden? Nein, zur Benennung der IoT-Geräte gibt es keine weiteren Blöcke in der Anwendung.

In der Anwendung sind zwei **Kontrollflussblöcke** implementiert: „onStart“ und „loop“ (siehe Abbildung 3.3). Die onStart und loop Blöcke bestimmen wann und wie oft darin platzierte Blöcke ausgeführt werden. Die Programmierumgebung stellt sicher, dass nur Abfrage-, Aktions oder Konfigurationsblöcke in die onStart und loop Blöcke platziert werden können. Bei Ausführung der Anwendung werden zu Beginn alle Blöcke innerhalb des onStart-Blockes je einmal nacheinander von oben nach unten ausgeführt. Erst wenn die Ausführung dieser Blöcke fertig ist, startet die Ausführung der Blöcke innerhalb des loop Blockes. Diese Blöcke werden bis zum Programmabbruch immer wieder fortlaufend abgearbeitet.

Was bringen diese Blöcke dem Nutzer? Der onStart Block ermöglicht eine einmalige Ausführung von Aufgaben die zum Programmstart erledigt werden müssen. Dies könnten Konfigurationsaufgaben, einmalige Abfragen wie Standorte oder Mac-Adressen oder Aktionen wie das Anzeigen von Daten sein. Durch den loop Block kann der Anwender bestimmte Daten in einem Intervall immer wieder abfragen. Dadurch sind jederzeit aktuelle Daten zur Bedingungsabfrage verfügbar. Der Anwender könnte so zum Beispiel, je nachdem ob ein Sender in der Nähe ist oder nicht, die Mitteilung „Sender ist da“ oder „Sender ist nicht da“ anzeigen lassen.

Kann diese Aktion mit mehr als einem Block durchgeführt werden? Die Kontrollflussblöcke modellieren beide unterschiedliche Abfolgen von Aktionen. Der onStart Block eine einmalige und der loop Block wiederholte Ausführung. Somit sind diese Blöcke nicht austauschbar.

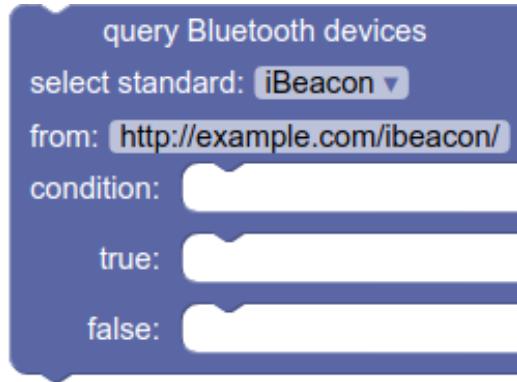


Abbildung 3.4: Abfrageblock für Bluetooth Geräte

Abfrageblöcke bilden die Grundlage der Kommunikation. Jede Anfrage an das IoT-System wird mit einem RDF-Graphen beantwortet. Um den gewünschten Graphen abzufragen und für Bedingungen zu verwenden, können die Anwender mit einem Abfrageblock definieren, welche Daten sie abfragen möchten. In der Anwendung ist ein Block der Daten der beiden Bluetooth Standards Bluetooth Low Energy (BLE)³ und iBeacon⁴ abfragen kann implementiert (siehe Abbildung 3.4). In diesem Abfrageblock für Bluetooth Daten wählt der Anwender über ein Dropdown-Feld zwischen den BLE und iBeacon Standards aus. Zusätzlich bestimmen sie über ein Eingabefeld die Hostadresse und den Pfad zum abzufragenden RDF-Graphen. Der Abfrageblock verarbeitet die abgefragten Daten und überprüft mit Bedingungsblöcken formulierte Gleichungen beziehungsweise Ungleichungen. Anschließend führt er, je nach Erfüllen oder Nichterfüllen dieser Bedingungen, vom Anwender definierte Aktionen aus.

Was bringen diese Blöcke dem Nutzer? Die Abfrageblöcke lassen den Anwender also, ohne Wissen über SPARQL oder andere Abfragesprachen, alle benötigten RDF-Graphen mit Daten zu BLE- oder iBeacon Geräten eines IoT-Systems abfragen. Zusätzlich können die Nutzer durch Abfrageblöcke abhängig von diesen Daten Aktionen ausführen.

Kann diese Aktion mit mehr als einem Block durchgeführt werden? In der Anwendung existieren keine anderen Blöcke, die selbiges tun.

Abfrageblöcke bieten dem Nutzer die Möglichkeit Bedingungen zu definieren. Dies geschieht mit Hilfe der **Bedingungsblöcke**. Hiermit können aus beliebigen abgefragten Daten, einem der Vergleichsoperatoren „=“, „<“ und „>“ und einem Zahlen- oder

³<https://www.bluetooth.com/learn-about-bluetooth/bluetooth-technology/radio-versions/>

⁴<https://developer.apple.com/ibeacon/>



(a) Bedingungsblock für iBeacon Standard (b) Bedingungen durch UND verknüpft

Abbildung 3.5: Bedingungsblöcke

Text-Wert Gleichungen und Ungleichungen formuliert werden. Hierfür muss immer ein zu prüfendes Datum und ein Operator aus einem Dropdown-Feld ausgewählt und ein Wert in das Eingabefeld eingegeben werden. Da die beiden Standards BLE und iBeacon unterschiedliche Daten zurückliefern, gibt es in der Anwendung zwei Bedingungsblöcke. Einen für Daten aus dem BLE Standard und einen für Daten aus dem iBeacon Standard. Die Bedingungsblöcke liefern immer einen booleschen Wert, also WAHR oder FALSCH zurück. Je nachdem ob die durch den Block formulierte Gleichung beziehungsweise Ungleichung gültig ist oder nicht. Abbildung 3.5a zeigt einen solchen Block. Der hier abgebildete Bedingungsblock liefert WAHR zurück falls der Signalstärke Wert (RSSI) größer ist als -40, ansonsten FALSCH. Zusätzlich befinden sich noch UND und ODER Blöcke in der Kategorie Bedingungen. Diese lesen als Eingabe immer zwei Blöcke. Entweder von den zwei oben genannten Bedingungsblöcken oder weitere UND oder ODER Blöcke. Dadurch lassen sich Bedingungen mit beliebig vielen anderen Bedingungen durch die logischen Operatoren UND und ODER verknüpfen. Dies ist zum Beispiel unbedingt erforderlich, wenn nur die Signalstärke eines bestimmten Senders überprüft werden soll. In diesem Fall verknüpft der Anwender zum Beispiel eine Bedingung „Mac-address = dcba4b9f9b“ durch einen UND Block mit der Bedingung „RSSI > -40“ (siehe Abbildung 3.5b).

Was bringen diese Blöcke dem Nutzer? Der Nutzer kann mit den Bedingungsblöcken Bedingungen, basierend auf den Ergebnissen der Abfragen formulieren. Diese lassen sich mit den UND und ODER Blöcken zu neuen Bedingungen verbinden.

Kann diese Aktion mit mehr als einem Block durchgeführt werden? Durch Überschneidungen in den beiden Standards lassen sich manche Bedingungen mit beiden Blöcken abbilden. Beispielsweise wird bei beiden Standards die Mac-Adresse übermittelt. Es lassen sich also die selben Aktionen mit mehreren Blöcken ausführen.

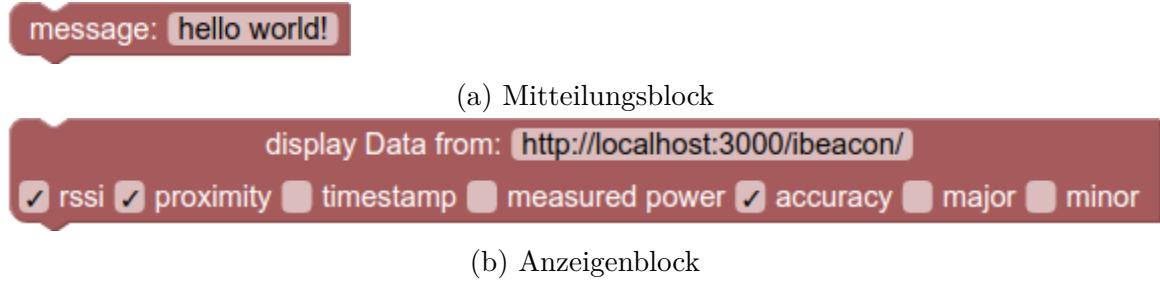


Abbildung 3.6: Aktionsblöcke

Aktionsblöcke ermöglichen es dem Anwender Aktionen durchzuführen. Zum Zeitpunkt der Erstellung dieser Arbeit existieren zwei Aktionsblöcke: der „Mitteilungsblock“ und der „Anzeigenblock“ (siehe Abbildung 3.6). Im Abschnitt 5.2 Ausblick werden mögliche weitere Aktionen besprochen. Mit dem ersten Block kann der Nutzer Mitteilungen definieren, die entweder bei Erfüllen oder Nichterfüllen einer Bedingung angezeigt werden. Der Mitteilungsblock erhält als Eingabe einen Text. Dieser wird bei der Ausführung auf der Oberfläche, in einem Textfeld neben dem Editor, angezeigt. Der Anzeigen-Block (siehe Abbildung 3.6b) schreibt die Ergebnisse der im Block definierten Abfrage in dasselbe Textfeld. Somit erhält der Nutzer eine anschauliche Übersicht gewünschter Daten. Woher diese Daten zu beziehen sind definiert der Anwender im Eingabefeld. Mit den Checkboxen können die Anwender auswählen welche Informationen für sie von Interesse sind. Die in Abbildung 3.6b ausgewählten Kästchen, zeigen beispielsweise Mac-Adresse, „RSSI“ und „proximity“. Als Identifikator wird die Mac Adresse immer mit angezeigt. RSSI ist die Abkürzung für Received Signal Strength Indication, also die Stärke des Bluetooth Signals in dBm. Proximity steht für einen String-Wert, der beschreibt wie nah zum Empfänger das Gerät ist. Dieser kann die vier Werte „immediate“ (wenige Zentimeter), „near“ (1–3 Meter), „far“ (über 3 Meter) und „unkown“ annehmen [Roch 17].

Was bringen diese Blöcke dem Nutzer? Durch den Mitteilungsblock hat der Anwender die Möglichkeit, auf bestimmte Zustände des Systems aufmerksam zu machen. So kann er zum Beispiel, wenn ein Sender in Empfängerreichweite eines Empfängers ist, eine Mitteilung „Sender ist da!“ anzeigen. Der Anzeigenblock dient einerseits der besseren Übersicht über die abfragbaren Daten, andererseits bietet er Hilfestellung bei der Fehlersuche. Falls zum Beispiel eine Bedingung wider Erwarten nicht erfüllt ist, kann der Nutzer mit Hilfe des Anzeigeblocks die tatsächlichen Daten mit den erwarteten vergleichen.

Tabelle 3.3: Blockdefinitionen mit Ein- und Ausgabe

Block	Eingabe	Ausgabe
Aliasblock	Mac-Adresse (<i>String</i>), Name (<i>String</i>)	JavaScript Befehl
onStart	Abfrageblock, Aktionsblock, Konfigurationsblock	Array mit JavaScript Befehlen
loop	Abfrageblock Aktionsblock, Konfigurationsblock	Array mit JavaScript Befehlen
Abfrageblock	Hostname (<i>String</i>), Bluetooth-Standard (<i>Dropdown-Wert</i>), Bedingungsblöcke, Aktionsblöcke	SPARQL SELECT, SPARQL WHERE, JavaScript Befehle
AND / OR Block	Bedingungsblöcke, AND / OR Blöcke	SPARQL FILTER
Bedingungsblock	Feld (<i>Dropdown-Wert</i>), Operator (<i>Dropdown-Wert</i>) Wert (<i>String</i>)	SPARQL FILTER
Aktionsblock	Aktionsparameter(<i>String</i>)	JavaScript Befehl

Kann diese Aktion mit mehr als einem Block durchgeführt werden? Mitteilungsblock sowie der Anzeigeblock sind in der Anwendung einzigartig.

3.3.2 Zusammensetzen der Blöcke

Eine Übersicht wie sich die Blöcke zusammen stecken lassen, also welcher Block welche Ein- und Ausgabetypen hat, findet sich in Tabelle 3.3. Hier ist in der zweiten Spalte abgebildet, welche Wertetypen jeder der jeweiligen Blöcke aus Spalte 1 als Eingabe akzeptiert. Die dritte Spalte stellt den Typ der Ausgabe des Blocks dar. Zusätzlich zur Tabelle 3.3 ist in Abbildung 3.7 ein fertiges Blockly-Programm abgebildet. Dieses fragt bis zum Programmabbruch immer wieder iBeacon Daten von der Adresse „<http://testserver.raspberry.pi>“ ab. Anhand der abgefragten Daten wird eine Bedingung formuliert. Diese liefert den Wert WAHR zurück, wenn der Sender mit der Mac-Adresse dcba4b9f9b entweder mit einer Signalstärke größer als -40 sendet, oder zur gleichen Zeit den Nähe-Wert (engl. proximity) „immediate“, also unmittelbar, und eine Genauigkeit (engl. accuracy) von unter 1.0 übermittelt. Also wenn der Sender sich in unmittelbarer Nähe befindet. Ansonsten gibt er FALSCH zurück. Tritt der in

```

1 Blockly.JavaScript['ibeaconconditions'] = function(block) {
2   var dropdown_ibeaconfield = block.getFieldValue('ibeaconField');
3   var dropdown_operator = block.getFieldValue('operator');
4   var text_value = block.getFieldValue('value').trim();
5
6   var code = '?${dropdown_ibeaconfield} ${operator} \'${text_value}\'';
7   return code;
8 };

```

Listing 3.2: Übersetzung eines Bedingungsblocks

der Bedingung definierte Fall ein, wird eine Mitteilung angezeigt. Diese hat den Text „Sender arrived!“.

3.4 Übersetzung und Ausführung der Blöcke

Beim Klicken auf den Execute-Button werden die Blöcke zunächst in Code übersetzt und daraufhin ausgeführt. Grundsätzlich funktioniert das Überführen der Blöcke in ausführbaren Code so: Ein Block liest zu Beginn seine Eingaben aus und gibt dann vorher definierten Code, welcher die diese Eingaben verwendet, zurück. So wird von innen nach außen Code generiert. Dieser kann dann, wenn alle Blöcke ihre Eingaben ausgelesen und an die äußersten Blöcke, die loop und onStart Blöcke, übergeben haben, ausgeführt werden.

Beispielsweise haben **Bedingungsblöcke** drei Felder für Benutzereingaben. Diese werden zunächst eingelesen und dann verwendet, um Gleichungen beziehungsweise Ungleichungen zu formulieren. Damit hieraus gültiger SPARQL Code entsteht, muss vor das erste Eingabefeld ein Fragezeichen geschrieben und zusätzlich das letzte Eingabefeld, das für die rechte Seite der Gleichung oder Ungleichung, in Anführungszeichen zu gesetzt werden. Der Ergebnisstring hiervon ist ein gültiger SPARQL Filter. In Listing 3.2 ist die Implementierung des Blocks zur Formulierung von iBeacon-Bedingungen zu sehen. Zeile 1–3 lesen die Eingabefelder aus. Anschließend werden die ausgelesenen Werte zu einem SPARQL-konformen Filter aneinandergereiht (Zeile 6). Ein Ergebnis dieser Operation könnte Beispielsweise „?rss > '-40“ lauten. In Zeile 7 wird dieser Filter abschließend zurückgegeben. Der Bedingungsblock für den BLE-Standard funktioniert analog dazu. Einzig die Auswählmöglichkeiten im ersten Dropdown-Feld unterscheiden sich vom oben erklärten iBeacon Bedingungsblock.

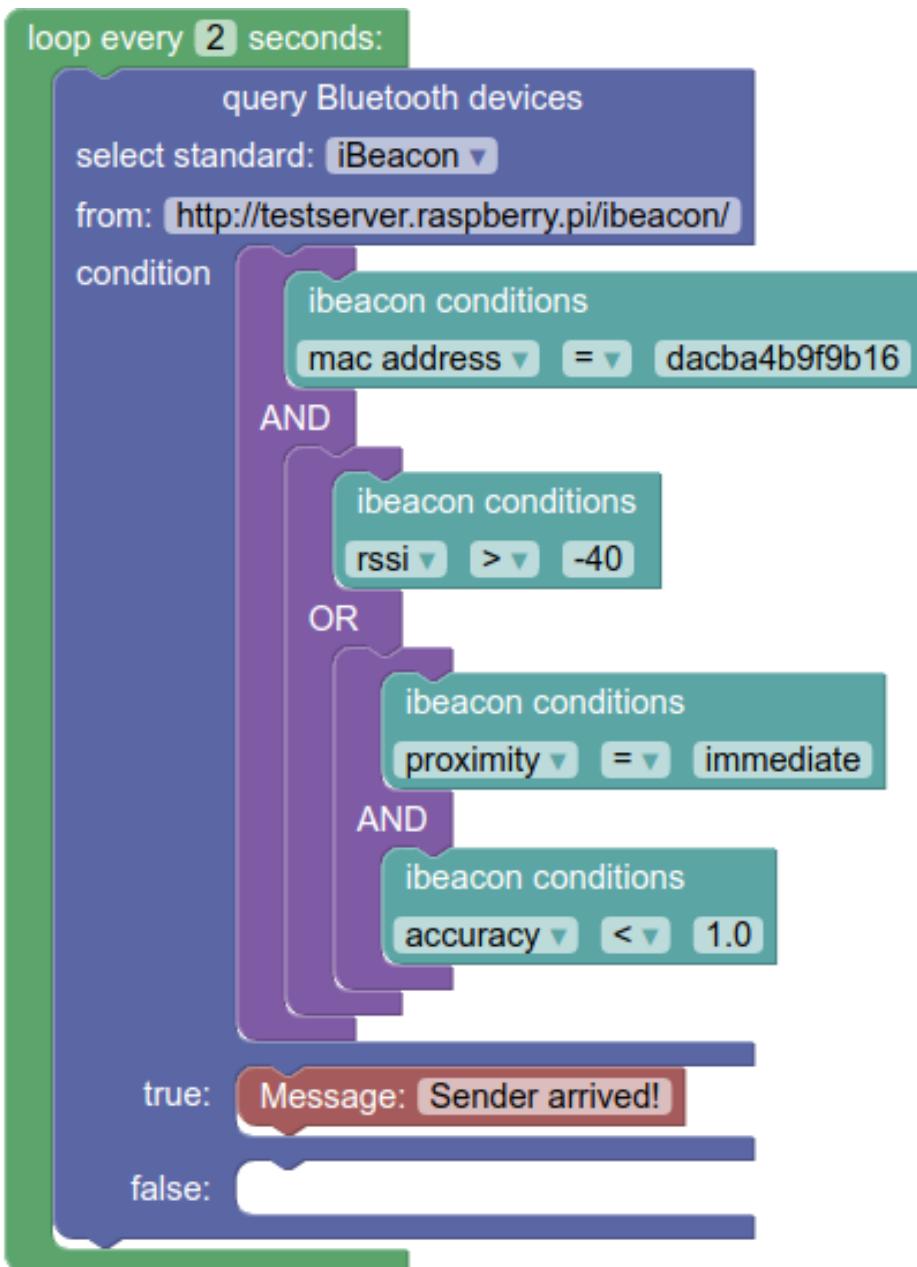


Abbildung 3.7: Beispiel Blockly-Programm

```

1  BASE <http://localhost:3000>
2  PREFIX scp: <https://github.com/aharth/supercool/>
3  PREFIX sosa: <http://www.w3.org/ns/sosa/>
4  PREFIX qudt: <http://qudt.org/1.1/schema/qudt#>
5
6  ASK
7  FROM </ibeacon>
8  WHERE
9  {
10    ?ble sosa:madeBySensor ?apfull .
11    BIND(substr(?apfull , 7,1) AS ?ap) .
12    ?ble sosa:hasResult ?sensor .
13    ?sensor scp:MacAddress ?mac .
14    ?sensor qudt:numericValue ?rss . .
15    ?sensor sosa:resultTime ?resultTime .
16    ?sensor scp:accuracy ?accuracy .
17    ?sensor scp:major ?major .
18    ?sensor scp:major ?minor .
19    ?sensor scp:measuredPower ?measuredPower .
20    ?sensor scp:proximity ?proximity .
21    FILTER((?mac = 'dcba4b9f9b16' && (?rss > '-40' || (?proximity = 'immediate' && ?accuracy < '1.0')))) .
22  }
23

```

Listing 3.3: Beispiel für eine ASK-Abfrage in SPARQL

Die **Aktionsblöcke** geben beide JavaScript Befehle zurück, welche Hilfsfunktionen zur Ausführung der jeweiligen Aktion benutzen. Der Mitteilungsblock beispielsweise hat den Rückgabewert „*insertMessage("\${text_msg}")*; |n“. Bei Ausführung dieses Codes wird also eine Hilfsfunktion „*insertMessage*“ mit dem eingegebenem Text als Parameter aufgerufen.

Etwas aufwändiger ist das Übersetzen der **Abfrageblöcke**. Außer den Eingabefeldern für den Hostnamen und zur Wahl des Bluetooth-Standard, sind die Eingaben der Abfrageblöcke alle Ausgaben anderer Blöcke. Zunächst wird die vom Abfrageblock ausgegebene SPARQL-Abfrage eingelesen. Zusammen mit dem Rückgabewerten der Bedingungsblöcken wird dann eine ASK-Abfrage erstellt. ASK-Abfragen in SPARQL liefern immer einen booleschen Wert zurück. Je nachdem ob die in SPARQL abgefragten Daten vorhanden sind oder nicht, ist dieser entweder WAHR oder FALSCH. In Listing 3.3 ist die aus der Übersetzung der Blöcke resultierende SPARQL-Abfrage des Beispielprogramms aus Listing 3.7 zu sehen. Um den korrekten Wahrheitswert zu erhalten, muss diese Abfrage mit dem RDF-Prozessor µRDF ausgeführt werden. Den Befehl hierfür schreibt der Abfrageblock zusammen mit den Aktionen welche bei Erfüllen und Nichterfüllen ausgeführt werden sollen, in eine Variable zur Rückgabe.

```

1 var code = Blockly.Javascript.workspaceToCode(workspace);
2
3 // writes code into arrays 'onStart' and 'loop'
4 eval(code);
5
6 // eval onStart
7 for(block of onStart){
8   eval(block);
9 }
10
11 // loop through blocks in loop-block
12 interval = setInterval(function() {
13   for(block of loop){
14     eval(block);
15   }
16 }, loopTime * 1000)
17 }
```

Listing 3.4: Ausführung der OnStart Prozesse

Dementsprechend erhalten die **Kontrollflussblöcke**, loop und onStart, eine Variable mit JavaScript-Befehlen als Eingabe. Diese beschreiben, im Fall des Abfrageblocks zum Beispiel, die Ausführung einer Abfrage mittels µRDF[Char17], sowie eine oder mehrere Folgeaktionen. In loop- und onStart-Blöcken können beliebig viele Abfrage-Aktions oder Konfigurationsblöcke platziert werden. Um die dort definierten JavaScript Befehle zu verwalten, werden diese in ein Array gespeichert. Dieses Array ist zugleich der Rückgabewert der Blöcke.

Letztendlich steht der Code für das gesamte durch die Blöcke definierte Programm also in zwei Arrays. Ein Array mit den JavaScript Befehlen des onStart-Blocks und ein zweites für die Befehle des loop-Blocks. Somit muss zur Ausführung des Blockprogramms lediglich nacheinander über die beiden Arrays iteriert werden. Dabei können deren Elemente per *eval* Befehl ausgeführt werden. Listing 3.4 implementiert dies beispielhaft für die beiden Kontrollfluss-Blöcke.

Dieses Kapitel definierte zunächst Anforderungen an die entwickelte Arbeit. Hierbei wurden drei zentrale Aufgaben zur Konfiguration eines IoT-Systems formuliert: Die Benennung von Bluetooth-Geräten, das Abfragen und Darstellen von RDF-Graphen mit Informationen von Bluetooth-Geräten und das Ausführen von Aktionen abhängig von den abgefragten Daten. Anschließend wurde die Anwendung besprochen. Zuerst wurde dabei eine Übersicht über die Architektur aufgezeigt. Diese wurde daraufhin in den Unterkapiteln 3.3.1 und 3.3.2 anhand der von Pasternak *et. al.* [Past17] definierten Fragen erörtert. Der darauf folgende Abschnitt zeigte alle Möglichkeiten die zuvor

definierten Blöcke zu kombinieren auf. Abschließend wurde im letzten Abschnitt der hinter den Blöcken liegende Code erklärt.

Kapitel 4

Evaluierung

In diesem Kapitel sollen zu Beginn die Methoden zur Evaluierung der Anwendung eingehend vorgestellt und diskutiert werden. Daraufhin werden die Ergebnisse der Evaluierung zusammenfassend dargestellt. Zum Schluss dieses Kapitels werden diese Ergebnisse und der Funktionsumfang des entwickelten Programms diskutiert.

4.1 Aufbau der Testumgebung

Um die Bedienbarkeit eines Systems zu ermitteln, existieren verschiedene gängige Vorgehensweisen [Niel94]. Die im Rahmen dieser Arbeit entwickelte Anwendung wurde, zur Messung der subjektiven Bewertung der Bedienbarkeit, mit einem standardisierten Fragebogen evaluiert.

Während einer Veranstaltung des Fraunhofer Instituts für Integrierte Schaltungen, wurden zufällig neun Studentinnen und Studenten aus verschiedenen Studiengängen (siehe Abbildung 4.1) ausgewählt. Alle dieser Probanden hatten zum Zeitpunkt des

Tabelle 4.1: Teilnehmer der Evaluation

Studiengang	Semester	Hochschule
Angew. Mathematik & Physik	Bachelor (Beginn)	TH Nürnberg
Elektromobilität	Master (Ende)	FH Würzburg-Schweinfurt
Informatik	Bachelor	TH Nürnberg
Integrated Life Science	Master (Ende)	FAU Erlangen
Künstliche Intelligenz	Bachelor (Beginn)	TH Ingolstadt
Maschinenbau	Bachelor	FAU Erlangen
Maschinenbau	Bachelor	TH Nürnberg
Medizintechnik	Bachelor	TH Nürnberg
Medizintechnik	Bachelor	OTH Amberg-Weiden

Tests keinerlei Kenntnisse im Bereich SPARQL-Abfragen oder RDF-Graphen. Somit entsprechen alle Teilnehmer der definierten Zielgruppe.

Zunächst wurden Aufgaben gestellt, welche mit Hilfe der Anwendung bewältigt werden mussten. Daraufhin wurde von jedem der Teilnehmer ein Fragebogen zur Bedienbarkeit der Anwendung beantwortet.

Im folgenden werden zuerst die gestellten Aufgaben und deren Absicht beschrieben. Anschließend wird die Wahl des Fragebogens begründet und dessen Fragen vorgestellt.

4.1.1 Aufgaben

Bevor die Teilnehmer die Aufgaben gestellt bekamen, sollten sie mit den Blöcken der Anwendung vertraut werden. Hierzu wurden in der Gruppe die in [Past 17] definierten Fragen „Kann der Anwender verstehen was jeder Block tut indem er ihn ansieht?“ und „Kann der Anwender verstehen was jeder Block tut indem er ihn ausführt?“ besprochen.

In Abbildung 4.1 ist eine Übersicht über die anschließend gestellten Aufgaben zu sehen. Es wird versucht, den Teilnehmer langsam an das Erstellen eines Blockprogramms aus der Realität heranzuführen. In Aufgabe 1 aus der Abbildung 4.1, soll der Proband ein einfaches Programm erstellen, welches eine Mitteilung ausgibt. In den nächsten beiden Aufgabe muss er den loop-Block verwenden, um regelmäßig Daten abzufragen und eine bedingungsabhängige Aktion durchzuführen. Die letzte Aufgabe besteht darin, das Beispielprogramm aus 3.7 zu erstellen. Hierzu wird die Bedingung aus dem Programm der vorherigen Aufgabe erweitert. Die Lösung dieser Aufgabe könnte so verwendet werden, um die Mitteilung „Container ist da!“ auszugeben, wenn ein in einen Postcontainer verbauter Sender in der Nähe eines Empfängers in einer Poststation ist. Es kann also mit dem anschließendem Fragebogen eine Aussage dazu getroffen werden, wie gut ein Anwender ohne Vorkenntnisse die Anwendung in einem realen Szenario bedienen kann.

Aufgabe 1: Die Probanden soll mit Hilfe der Blöcke ein Programm erstellen, welches beim Programmstart in einer Mitteilung den Text „Hello, World!“ ausgibt.

Wozu wird diese Aufgabe gestellt? Zu Beginn des Tests sollen die Probanden sich mit der Oberfläche vertraut machen. Hierzu sollen sie ein möglichst einfaches Programm

1 Hello, World!

Platzieren Sie die Blöcke der Anwendung so, dass **zu Programmstart** die **Mitteilung** "Hello, World!" ausgegeben wird.

2 Schleife

Setzen Sie die Blöcke so zusammen, dass in einer **Schleife** alle 3 Sekunden folgende **Aktion** immer wieder ausgeführt wird:

Zeige die Daten *RSSI, proximity und accuracy* von "<http://testserver.raspberry.pi/ibeacon/>" an.

3 Abfrage

Erweitern Sie das Programm aus Aufgabe 2 mit einem **Abfrageblock** so, dass in der Schleife nach jedem Anzeigen der Daten, folgende Abfrage durchgeführt wird:

1. *iBeacon* Daten von der Adresse "<http://testserver.raspberry.pi/ibeacon/>" abfragen
2. Mit einem **iBeacon Bedingungsblock** überprüfen, ob ein Sender mit der Mac-Adresse "cafebabe" existiert
3. Falls ja, gebe **Mitteilung** "Hello cafebabe!" aus"

4 Logische Operatoren

Erweitern Sie die Bedingung des Programms aus Aufgabe 3 mit **iBeacon Bedingungsblöcken** und **logischen Operatoren** so, dass die Mitteilung nur ausgegeben wird, wenn:

- der iBeacon-Sender mit der Mac-Adresse "cafebabe" existiert **UND**
- entweder
 - dessen Signalstärke (RSSI) größer als -40 ist **ODER**
 - dessen proximity Wert gleich "immediate" **UND**
 - dessen accuracy Wert kleiner als 1.0 ist

Abbildung 4.1: Aufgabenstellung vor der Evaluation

ausführen. Im Vordergrund steht das Erlernen der Bedienung des Editors. So können sie selbstständig Antworten auf folgende grundlegende Fragen erarbeiten: „Wo finde ich die Blockkategorien mit den Blöcken?“, „Wie lassen die Blöcke sich zusammen stecken?“ und „Wie führe ich ein fertiges Programm aus?“.

Aufgabe 2: In der zweiten, in Abbildung 4.1 gestellten, Aufgabe benutzen die Teilnehmer das erste Mal den loop-Block. Hierbei müssen sie zusätzlich zu dem Hereinziehen von Blöcken auch Parameter verändern. Sie müssen eine Intervallzeit von drei Sekunden in das Eingabefeld des loop-Blocks schreiben und die verlangten Checkboxen im Anzeigen-Block auswählen.

Wozu wird diese Frage gestellt? Die Anwender sollen einerseits das Konzept des loop Blockes begreifen. Andererseits erfahren sie, dass durch Eingabefelder verschiedene Parameter der Blöcke verändert werden können. Zusätzlich zu diesen Erkenntnissen sehen sie in der Ausgabe erstmals Beispieldaten des Systems. Somit werden sie besser mit dem IoT-System vertraut.

Aufgabe 3: Das Ziel der Aufgabe 3 aus Abbildung 4.1 ist es, die Lösung aus der zweiten Aufgabe dahingehend zu erweitern, dass nun eine Mitteilung mit dem Text „Hello cafebabe!“ ausgegeben wird, falls nachstehende Bedingung erfüllt ist: In den unter der Adresse „<http://testserver.raspberry.pi/ibeacon/>“ abfragbaren Daten ist die Mac-Adresse „cafebabe“ vorhanden.

Wozu wird diese Aufgabe gestellt? Die Probanden soll erstmals den Abfrageblock und den Bedingungsblock verwenden und verstehen. Sie sind anschließend in der Lage Daten abzufragen, Bedingungen zu formulieren und abhängig von den abgefragten Daten Aktionen auszuführen.

Aufgabe 4: In der letzten Aufgabe aus Abbildung 4.1 sollen die Probanden die Lösung der dritten Aufgabe nochmals erweitern. Die Bedingung muss so verändert werden, dass die Mitteilung „Hello cafebabe!“ nur ausgegeben wird, wenn das iBeacon-Geräte mit der Mac-Adresse „cafebabe“ sich in unmittelbarer Nähe befindet. Also wenn entweder dessen Signalstärke größer als -40 dBm ist, oder wenn er zur selben Zeit den Proximity-Wert „immediate“ und einen Accuracy-Wert kleiner als 1.0 überträgt.

Wozu wird diese Aufgabe gestellt? In dieser Aufgabe wird ein Programmbeispiel, das in der Realität so Verwendung finden könnte, nachgebaut. Dadurch können die Probanden bei der Beantwortung des anschließenden Fragebogens bewerten, wie gut sich die Anwendung in einem realen Szenario bedienen lässt.

Der Teilnehmer wurde anhand der oben genannten Aufgaben schrittweise dazu angeleitet, die Erstellung eines Programms zu erlernen. Dieses gibt eine Mitteilung aus, wenn ein im Programm bestimmter Sender sich in unmittelbarer Nähe befindet. Ein solches Programm könnte auch so im Postbehälterkreislauf des Fraunhofer Instituts in Nürnberg verwendet werden. Der Proband ist also mit Vollendung der vierten Aufgabe aus Abbildung 4.1 in der Lage, solch ein System zu bedienen.

4.1.2 Fragebogen

In der Literatur existiert eine Vielzahl an standardisierten Fragebögen mit der Testanwender die Bedienbarkeit einer Anwendung bewerten können [Assi 16]. Solche standardisierten Fragebögen bieten Objektivität, Reproduzierbarkeit, Quantifizierung, Wirtschaftlichkeit, Kommunikation und wissenschaftliche Verallgemeinerung [Nunn 94]. In [Assi 16] werden einige der am weitesten verbreiteten Fragebögen verglichen. Dabei kommt [Assi 16] zu dem Ergebnis, dass der in [Broo 96] definierte Fragebogen, System Usability Scale (SUS), der am schnellsten auf die richtige Schlussfolgerung konvergierende Fragebogen sei. Zudem liefere er zuverlässige Ergebnisse über alle verwendeten Stichprobengrößen. Auch im Vergleich von [Tull 04] liefert die SUS mit die verlässlichsten Ergebnisse. Aufgrund dieser Ergebnisse wurde der SUS-Fragebogen zur Bewertung der Bedienbarkeit der Anwendung ausgewählt.

Die System Usability Scale (SUS) aus [Broo 96] besteht aus 10 Aussagen (siehe Abbildung 4.2) zur Bedienbarkeit einer Anwendung. Die Aussagen beschreiben unter anderem Komplexität, Einfachheit, Konsistenz, Erlernbarkeit und Notwendigkeit von Vorwissen. Sie decken also alle Ziele zur Bedienbarkeit der Anwendung ab.

Die Aussagen werden von den Probanden durch Ankreuzen einer Skala von 1 für „Stimme gar nicht zu“ bis 5 für „Stimme voll zu“ bewertet. Hierbei werden den Probanden abwechselnd positiv und negativ formulierte Aussagen vorgelegt. Dies verhindert, dass die Teilnehmer den Fragebogen schnell, ohne viel über die Aussage nachzudenken, beantworten.

Insgesamt bietet der SUS einen standardisierten Fragebogen, der alle für diese Arbeit relevanten Themen bezüglich der Bedienbarkeit der Anwendung abdeckt. Außerdem liefert er mit nur 10 Fragen in kurzer Zeit Ergebnisse. Diese sind selbst bei einer geringen Teilnehmeranzahl von 8–12 Personen aussagekräftig [Bang 08].

	Stimme gar nicht zu	Stimme voll zu
1.	1 2 3 4 5	
2.	1 2 3 4 5	
3.	1 2 3 4 5	
4.	1 2 3 4 5	
5.	1 2 3 4 5	
6.	1 2 3 4 5	
7.	1 2 3 4 5	
8.	1 2 3 4 5	
9.	1 2 3 4 5	
10.	1 2 3 4 5	

Abbildung 4.2: System Usability Scale

Tabelle 4.2: Ergebnisse des System Usability Scale
Teilnehmer t1-t9, Fragen f1-f10

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	SUS Score
t1	4	1	5	1	5	3	5	1	5	1	92.5
t2	3	1	5	2	4	1	5	2	3	2	80.0
t3	3	2	4	3.5	2	1	1	2	1	4	46.3
t4	4	1	3	4	2	1	2	2	2	2	57.5
t5	5	1	5	1	5	1	4	1	5	1	97.5
t6	4	1	5	1	5	1	5	1	2	1	90.0
t7	3	1	5	1	5	2	4	1	4	1	87.5
t8	3	1	4	1	5	3	5	1	3	1	82.5
t9	5	1	5	4	4	2	5	1	5	1	87.5
	3.8	1.1	4.6	2.0	4.2	1.8	4.1	1.3	3.5	1.5	81.4

4.2 Ergebnisse und Beobachtungen

Zu Beginn wurde den Teilnehmern die in [Past 17] definierten Fragen 6–8 zum Anwendungsumfang gestellt und in der Gruppe diskutiert. Hierbei war zu beobachten, dass die Teilnehmer jeden der Blöcke innerhalb von wenigen Sekunden finden konnten. Sie waren zudem in der Lage durch Ansehen oder Ausführen zu ermitteln was jeder Block tut.

Nach dieser kurzen Einfindungsphase haben die Probanden oben stehende Aufgaben in 2er-Gruppen gelöst. Hierfür wurde zwischen 9 und 17 Minuten benötigt. Anschließend hat jeder der Teilnehmer den Fragebogen der SUS ausgefüllt, um so eine Bewertung der Bedienbarkeit der Anwendung abzugeben. Die Ergebnisse der SUS werden in diesem Abschnitt erläutert.

Die SUS liefert zunächst 10 Zahlenwerte pro Teilnehmer (siehe Tabelle 4.2), ein Wert von 1 bis 5 für jede Aussage. [Broo 96] beschreibt wie anhand der Antworten der SUS Score ermittelt wird. Dies ist ein Zahlenwert zwischen 0 und 100, wobei 0 das schlecht mögliche Ergebnis darstellt. Der Durchschnitt all dieser berechneten SUS Score Werte, ist der ermittelte SUS Score der Anwendung. In der vorliegenden Evaluierung wurde ein SUS Score von 81.4 erreicht. Warum dieser Wert der Schulnote 2 entspricht, beziehungsweise wie er im Vergleich mit anderen SUS-Tests einzuordnen ist wird im Unterkapitel 4.4 erläutert.

Bei Betrachtung der Werte der einzelnen Aussagen fällt auf, dass die Aussagen „Ich fand das System unnötig komplex.“ und „Ich fand das System sehr umständlich zu nutzen.“ mit einer Abweichung von 1 sehr eindeutige Zustimmung beziehungsweise

Ablehnung erhielten. Die Teilnehmer waren sich demnach einig, dass das System überhaupt nicht umständlich zu nutzen oder unnötig komplex ist. Große Varianz hingegen ist in der Beantwortung der Aussagen 7 „Ich kann mir vorstellen, dass die meisten Menschen den Umgang mit diesem System sehr schnell lernen.“, und 9 „Ich fühlte mich bei der Benutzung des Systems sehr sicher.“ aus Abbildung 4.2 zu beobachten. Die Probanden waren also sehr unterschiedlicher Meinung, ob die meisten Menschen den Umgang mit dem System schnell erlernen könnten und 3 Teilnehmer fühlten sich nicht sehr sicher im Umgang mit dem System.

4.3 Möglichkeiten der Blockprogramme im IoT-System des Postbehälterkreislaufs

Die entwickelte Anwendung ist in der Lage, mittels Blöcken diverse Programme zur Nutzung von IoT-Systemen zu erstellen. Im folgenden wird erläutert in wie weit diese die zuvor definierten Anforderungen erfüllen. Diese werden in der unter diesem Absatz stehenden Aufzählung nochmals zusammengefasst. Zusätzlich wird besprochen wie gut sich die Anwendung zur Verwendung im Postbehälterkreislauf des Fraunhofer Instituts Nürnberg eignet.

1. Benennung von Bluetooth Sendern und Empfängern
2. Abfragen von Bluetooth Daten
3. Aktionen abhängig von diesen Daten ausführen

Abbildung 4.3 zeigt wie ein Blockprogramm, das alle diese Funktionen nutzt, aussehen könnte. Im onStart Block werden zum Programmstart, wie in Anforderung 1 gefordert, Sender und Empfänger mit Namen versehen. Der Sender mit der Macadresse „dcba4b9f9b16“ erhält den Namen „Yellow Container“. Dieser beschreibt den Container in dem der Sender verbaut ist. Zusätzlich dazu werden die Adressen auf denen die „Poststelle A“ und die „Poststelle B“ RDF-Graphen, mit aktuellen Bluetooth-Daten, zur Verfügung stellen mit den Namen „Post A“ und „Post B“ beschrieben.

Die zweite und dritte Anforderung wird jeweils durch Blöcke innerhalb des loop-Blocks erfüllt. Es werden alle zwei Sekunden Bluetooth Daten abgefragt, um, abhängig von diesen Daten, Mitteilungen anzuzeigen. Das abgebildete Blockprogramm liest Informationen der beiden zuvor genannten Poststellen. Zeigen diese an, dass sich der Sender

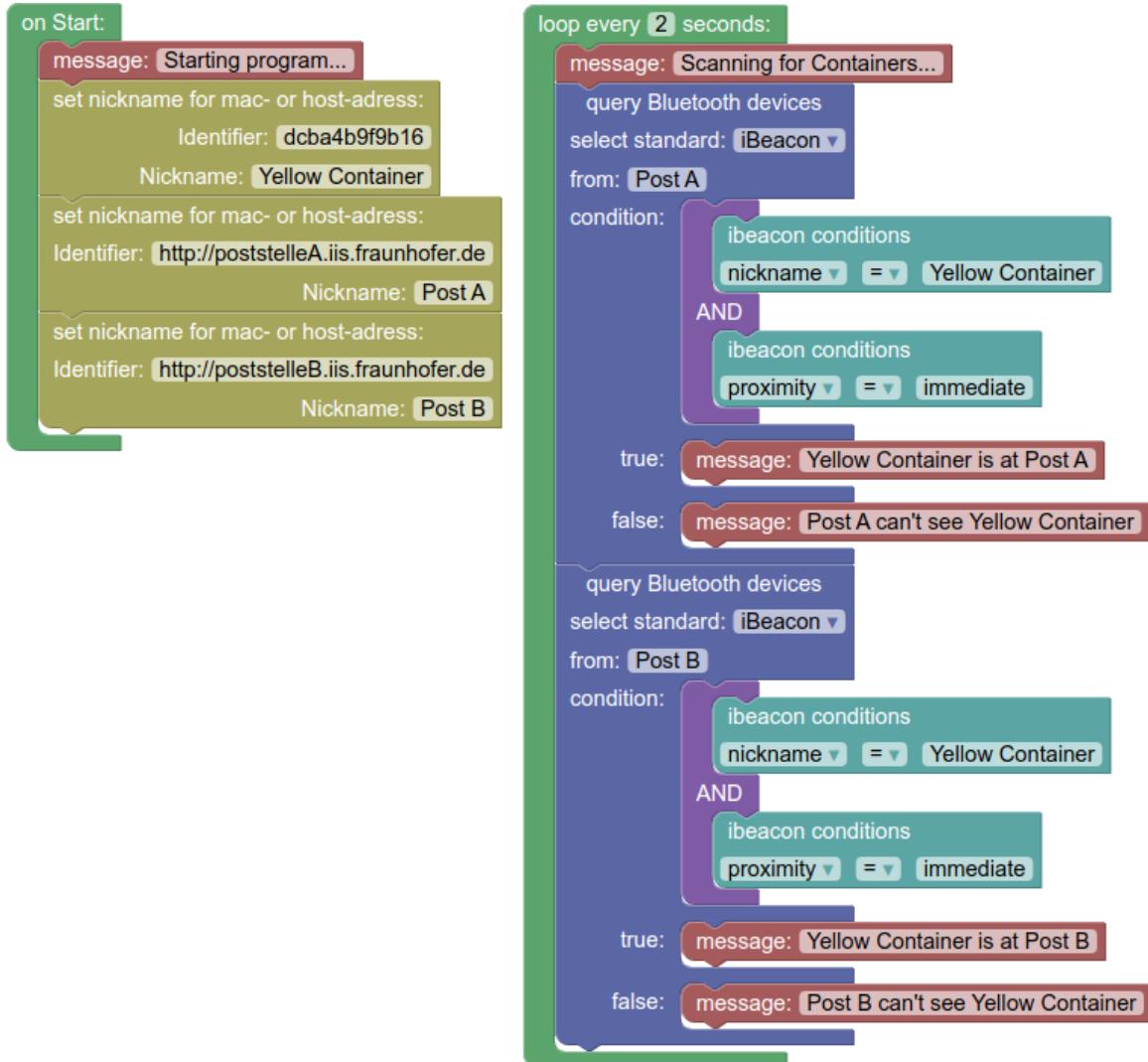


Abbildung 4.3: Blockprogramm für einen Postbehälterkreislauf

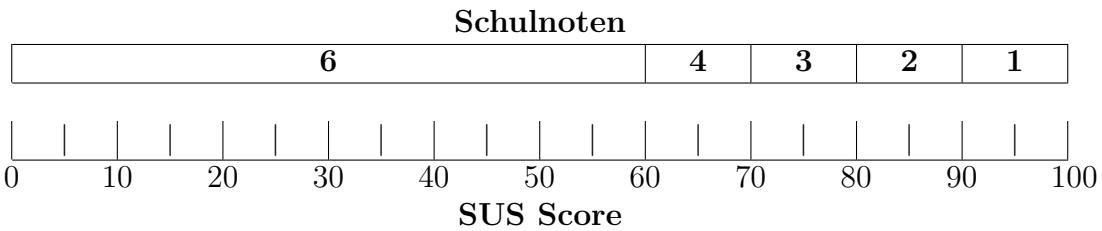


Tabelle 4.3: SUS Score Schulnotenskala

mit dem Namen „Yellow Container“ in der Nähe befindet, wird eine entsprechende Mitteilung angezeigt. Falls nicht, zeigt die Anwendung ebenfalls eine Mitteilung an, welche die Abwesenheit des Containers beschreibt. Dadurch, dass Mitteilungen mit Zeitstempeln versehen sind, lassen sich auch die Zeiten der Abwesenheiten feststellen.

Die im Rahmen dieser Arbeit entwickelte Anwendung ist somit in der Lage, ein Blockprogramm zu erstellen, welches die Status „Behälter ist an Poststelle X“ oder „Behälterstandort ist seit Y Stunden unbekannt“ beschreiben kann. Weitere, noch nicht implementierte, Möglichkeiten die Anwendung zu verbessern werden im folgenden Unterkapitel 4.4 Diskussion und Bewertung besprochen.

4.4 Diskussion und Bewertung

Durch die SUS Fragebögen wurde die Bedienbarkeit durch neun zufällig gewählte Studentinnen und Studenten mit einem SUS Score von 81.4 bewertet. Um den SUS Score einzuordnen wurden in [Bang 08] und [Bang 09] insgesamt über 3500 SUS Ergebnisse betrachtet. Dabei wird ein Durchschnittswert von 68 ermittelt. [Bang 09] vergleicht den SUS Score mit Beschreibungen der getesteten Systeme durch Adjektive wie „gut“, „schlecht“ oder „ausgezeichnet“ und schlägt vor den SUS Score auf einer Schulnoten Skala abzubilden (Siehe Abbildung 4.3). Diese Skala und der SUS Score Durchschnittswert, helfen den für die vorliegende Anwendung ermittelten Wert von 81.4 einzuordnen. Demnach liegt die Bedienbarkeit der Anwendung über dem Durchschnitt und ist mit der Schulnote 2 zu bewerten.

Eine Maßnahme zur Verbesserung der Bedienbarkeit, wären mehr Rückmeldungen seitens der Anwendung zu implementieren. Durch Rückmeldungen ist der Anwender bei Ausführung, besonders im Fehlerfall, besser darüber informiert was die Anwendung

macht. Dies erleichtert es dem Anwender mögliche Fehler in dem von ihm erstelltem Blockprogramm zu finden.

Mit der Anwendung lassen sich verschiedene Blockprogramme erstellen. Die diesbezüglich definierten Anforderungen wurden erfüllt. Um den Einsatz zum Beispiel im Fall des Postbehälterkreislaufs zu verbessern, könnten jedoch noch Verbesserungen durchgeführt werden.

Die Anwendung ist beispielsweise nicht in der Lage, Ereignisse wie „Behälter ist an Standort X angekommen“, oder „Behälter hat Standort X verlassen“ zu beschreiben. Hierfür müsste der aktuelle Zustand mit dem vorherigen verglichen werden. Dies ist in SPARQL umsetzbar, aber nicht mit den implementierten Blöcken abbildbar. Die Bedingungsblöcke der Anwendung können abgefragte Variablen nicht miteinander vergleichen. Deshalb können nicht zwei Zeitstempel miteinander verglichen werden, um eine Reihenfolge von Signalstärkemessungen zu ermitteln.

Durch eine Ereigniserkennung ließe sich das in Abbildung 4.3 gezeigte Blockprogramm zur Nutzung des IoT-Systems eines Postbehälterkreislaufs verbessern. Denn aktuell wird alle 2 Sekunden eine Mitteilung ausgegeben. Wäre die Anwendung in der Lage Ereignisse zu erkennen, könnte man nur bei Ereigniseintritt eine Mitteilung anzeigen.

Kapitel 5

Fazit und Ausblick

Abschließend werden die Knergebnisse in Bezug auf die genannten Forschungsfragen zusammengefasst. Darüber hinaus werden die Grenzen der Arbeit und der zukünftige Forschungsbedarf aufgezeigt.

5.1 Zusammenfassung

In der vorliegenden Arbeit wird eine Anwendung zur Konfiguration von IoT-Systemen mit RDF-Graphen entwickelt. Mittels Befragungen von Anwendern aus der Zielgruppe sowie der Einbeziehung verwandter Arbeiten werden die genannten Forschungsfragen untersucht.

Zum einen stellte sich die blockbasierte Entwicklungsumgebung als die am besten geeignete Möglichkeit heraus, benutzerfreundliche Oberflächen für die geforderten Systeme zu entwickeln. Dies wurde durch Vergleichen existierender Lösungen zu ähnlichen Problemstellungen ermittelt. Andere Lösungen wie beispielsweise Graphenbasierte Editoren sind weniger intuitiv und erfordern mehr Vorwissen. Speziell im IoT-Bereich hob sich die weit verbreitete Blockly-Bibliothek deutlich von anderen Block-Editoren hervor. Somit erschien eine blockbasierte Entwicklungsumgebung auf Basis der Blockly-Bibliothek für das gegebene Szenario am hilfreichsten.

Zum anderen konnte exemplarisch gezeigt werden wie SPARQL und JavaScript Befehle mit dem selben Blockeditor erzeugt werden können. Anders als bei einer Mischform die zum Beispiel Blöcke für die Abfrageerstellung und Graphen für die Bedingungsabhängige Ausführung von Aktionen verwendet, wurden in der Anwendung alle Aufgaben mit einer blockbasierten Umgebung realisiert. Dadurch konnte das Erlernen der Nutzung einfach und die Oberfläche robust gehalten werden. Der Funktionsumfang

erfüllte zwar die Anforderungen, ist aber dadurch, dass nur ein kleiner Teil der Abfragegespräche SPARQL durch die implementierten Blöcke erzeugbar ist, durchaus noch erweiterbar.

In einer abschließenden Evaluation wurde die Bedienbarkeit der Anwendung untersucht. Neun zufällig ausgewählte Studenten bewerteten die Anwendung mit einem SUS Score von 81.4. Abgebildet auf Schulnoten entspricht dies der Note 2. Es konnte festgestellt werden, dass sich die Anwender mehr Rückmeldung vom Programm wünschen.

Zusammenfassend bleibt festzuhalten, dass mit einer blockbasierten Entwicklungsumgebung, welche Technologien von IoT-Systemen abstrahiert, eine gute Bedienbarkeit möglich ist. Die in der vorliegenden Arbeit entwickelte Anwendung könnte, durch mehr Rückmeldungen und einen größeren Funktionsumfang, verbessert werden.

5.2 Ausblick

In diesem Unterkapitel werden abschließend Möglichkeiten diskutiert, um die Bedienbarkeit der Anwendung attraktiver zu gestalten und deren Funktionsumfang weiter auszubauen.

Zum einen ist es für den Anwender nicht immer nachvollziehbar was das Programm zur Ausführungszeit aktuell macht. Klarheit darüber was momentan ausgeführt wird, könnte beispielsweise ein Hervorheben der momentan aktiven Blöcke bringen. Zum Beispiel kann der loop Block bei jedem Aufruf kurz eine Umrandung erhalten. Denkbar wäre auch ein Countdown oder Sanduhr-Widget als Rückmeldung, wann ein loop Block ausgeführt wird. Zusätzlich dazu könnten in Abfrageblöcken, je nach Erfüllen oder Nichterfüllen der darin definierten Bedingungen, die Aktionsblöcke in den WAHR oder FALSCH Eingabefeldern markiert werden.

Zum anderen wäre es hilfreich für die Bedienbarkeit, wenn sich Blockprogramme und Teile von Blockprogrammen zur Wiederverwendung speichern ließen. So müssten häufig verwendete Programmteile nicht jedes Mal neu zusammengesetzt werden.

Zur Erweiterung des Funktionsumfangs wäre es einerseits denkbar mehr Aktionsblöcke einzuführen. Aktionen wie E-Mail oder SMS senden bringen einen großen Mehrgewinn für die Anwendung, da der Anwender durch diese nicht vor Ort sein muss, um benachrichtigt zu werden. Aber auch das implementieren von Aktionsblöcken die das

HTTP nutzen um die APIs von IoT-Systemen zu verwenden wären vorteilhaft. So könnte zum Beispiel eine Heizung ausgeschaltet werden wenn die Temperatur einen vorgegebenen Wert erreicht.

Andererseits kann der Funktionsumfang durch zusätzliche Bedingungsblöcke erweitert werden. Mit der Möglichkeit abgefragte Werte in Variablen zu speichern und diese miteinander zu vergleichen könnten Blockprogramme erstellt werden, welche eine Ereigniserkennung realisieren.

Zudem könnte der Umfang der Anwendung durch Implementieren neuer Konfigurationsblöcke, weiter ausgedehnt werden. Viele IoT-Systeme bieten die Möglichkeit nicht nur Daten per HTTP-GET anzufragen, sondern auch mittels HTTP-PUT Daten auf IoT-Geräte zu schreiben. Die Umsetzung solcher Anfragen über Konfigurationsblöcke, bietet dem Nutzer viele neue Möglichkeiten für die Blockprogrammierung. Zum Beispiel könnten so Intervallzeiten, in denen Bluetooth Empfänger nach Geräten scannen, eingestellt werden. Ebenso könnten Sendeleistung und -Frequenz von Bluetooth-Sendern konfiguriert werden.

Das Erforschen dieser Möglichkeiten wäre durchaus sinnvoll, um die Möglichkeiten der Blockprogramme zu erweitern und deren Bedienbarkeit zu verbessern. Durch mehr Möglichkeiten und einer leichteren Bedienung könnte das Potential von IoT-Geräten noch besser ausgeschöpft werden.

Abbildungsverzeichnis

1.1 Aufbau der Arbeit	4
2.1 Beispiel für ein „Ding“, aus dem IoT-System „Postbehälterkreislauf“	9
2.2 Aufbau des Beispiel-Systems	14
2.3 Übertragungstechnologien des Beispiel Systems	15
3.1 Schichten der Anwendung	19
3.2 Der AliasBlock	23
3.3 Die Kontrollflussblöcke	24
3.4 Abfrageblock für Bluetooth Geräte	25
3.5 Bedingungsblöcke	26
3.6 Aktionsblöcke	27
3.7 Beispiel Blockly-Programm	30
4.1 Aufgabenstellung vor der Evaluation	37
4.2 System Usability Scale	40
4.3 Blockprogramm für einen Postbehälterkreislauf	43

Tabellenverzeichnis

3.1 Bestimmung der Zielgruppe	21
3.2 Bestimmung des Anwendungsumfangs	22
3.3 Blockdefinitionen mit Ein- und Ausgabe	28
4.1 Teilnehmer der Evaluation	35
4.2 Ergebnisse des System Usability Scale Teilnehmer t1-t9, Fragen f1-f10 . .	41
4.3 SUS Score Schulnotenskala	44

Listingverzeichnis

2.1 RDF Beispiel	8
3.1 BLE Informationen im RDF Graph	17
3.2 Übersetzung eines Bedingungsblocks	29
3.3 Beispiel für eine ASK-Abrage in SPARQL	31
3.4 Ausführung der OnStart Prozesse	32

Literaturverzeichnis

- [Alta16] A. Altadmri, M. Kölling, and N. C. Brown. “The cost of syntax and how to avoid it: Text versus frame-based editing”. In: *proceedings of the 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, pp. 748–753, IEEE, 2016.
- [Assi16] A. Assila, H. Ezzedine, *et al.* “Standardized usability questionnaires: Features and quality focus”. *Electronic Journal of Computer Science and Information Technology: eJCIST*, Vol. 6, No. 1, 2016.
- [Bak18] N. Bak, B.-M. Chang, and K. Choi. “Smart Block: A visual programming environment for SmartThings”. In: *proceedings of the 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, pp. 32–37, IEEE, 2018.
- [Bang08] A. Bangor, P. T. Kortum, and J. T. Miller. “An empirical evaluation of the system usability scale”. *Intl. Journal of Human–Computer Interaction*, Vol. 24, No. 6, pp. 574–594, 2008.
- [Bang09] A. Bangor, P. Kortum, and J. Miller. “Determining what individual SUS scores mean: Adding an adjective rating scale”. *Journal of usability studies*, Vol. 4, No. 3, pp. 114–123, 2009.
- [Bern01] T. Berners-Lee, J. Hendler, and O. Lassila. “The semantic web”. *Scientific american*, Vol. 284, No. 5, pp. 34–43, 2001.
- [Bern06] T. Berners-Lee. “Linked data”. 2006.
- [Bors06] J. Borsje and H. Embregts. “Graphical query composition and natural language processing in an RDF visualization interface”. *Erasmus School of Economics and Business Economics, Vol. Bachelor. Erasmus University, Rotterdam*, p. 76, 2006.

- [Broo 96] J. Brooke *et al.* “SUS-A quick and dirty usability scale”. *Usability evaluation in industry*, Vol. 189, No. 194, pp. 4–7, 1996.
- [Ceri 17] M. Ceriani and P. Bottoni. “SPARQLBlocks: Using blocks to design structured linked data queries”. In: *Proceedings of Journal of Visual Languages and Sentient Systems (VLSS)*, 2017.
- [Char 17] V. Charpenay, S. Käbisch, and H. Kosch. “μRDF store: Towards extending the semantic web to embedded devices”. In: *proceedings of the European Semantic Web Conference*, pp. 76–80, Springer, 2017.
- [Clem 20] J. Clement. “Global digital population as of january 2020”. <https://www.statista.com/statistics/617136/digital-population-worldwide/>, 2020. Online, aufgerufen am 04.03.2020.
- [Culi 15] I. Culic, A. Radovici, and L. M. Vasilescu. “Auto-generating google blockly visual programming elements for peripheral hardware”. In: *proceedings of the 2015 14th RoEduNet International Conference-Networking in Education and Research (RoEduNet NER)*, pp. 94–98, IEEE, 2015.
- [Depa 19] S. R. Department. “Internet of Things - number of connected devices worldwide 2015-2025”. <https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/>, 2019. Online, aufgerufen am 04.03.2020.
- [Fadh 07] A. Fadhil and V. Haarslev. “OntoVQL: A graphical query language for owl ontologies”. In: *proceedings of Description Logics*, 2007.
- [Ferr 17] S. Ferré. “Sparklis: an expressive query builder for sparql endpoints with guidance in natural language”. *Semantic Web*, Vol. 8, No. 3, pp. 405–418, 2017.
- [Gali 07] W. O. Galitz. *The essential guide to user interface design: an introduction to GUI design principles and techniques*. John Wiley & Sons, 2007.
- [Ghaz 16] M. Ghazal, F. Haneefa, S. Ali, Y. Al Khalil, and A. Sweleh. “A framework for teaching robotic control using a novel visual programming language”. In: *proceedings of the 2016 IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 1–4, IEEE, 2016.

- [Gome07] A. Gomes and A. J. Mendes. “Learning to program-difficulties and solutions”. In: *proceedings of the International Conference on Engineering Education–ICEE*, 2007.
- [Gyra15] A. Gyrard, M. Serrano, and G. A. Atemezing. “Semantic web methodologies, best practices and ontology engineering applied to internet of things”. In: *proceedings of the 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pp. 412–417, IEEE, 2015.
- [Gyra16] A. Gyrard, P. Patel, S. Datta, and M. Ali. “Semantic web meets internet of things (IoT) and web of things (WoT)”. In: *proceedings of The 15th International Conference on Semantic Web (ISWC). (Oct 2016)*, sn, 2016.
- [Jara14a] A. J. Jara, A. C. Olivieri, Y. Bocchi, M. Jung, W. Kastner, and A. F. Skarmeta. “Semantic web of things: an analysis of the application semantics for the iot moving towards the iot convergence”. *International Journal of Web and Grid Services*, Vol. 10, No. 2-3, pp. 244–272, 2014.
- [Jara14b] A. J. Jara, A. C. Olivieri, Y. Bocchi, M. Jung, W. Kastner, and A. F. Skarmeta. “Semantic web of things: an analysis of the application semantics for the iot moving towards the iot convergence”. *International Journal of Web and Grid Services*, Vol. 10, No. 2-3, pp. 244–272, 2014.
- [Jenk02] T. Jenkins. “On the difficulty of learning to program”. In: *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*, pp. 53–58, Citeseer, 2002.
- [Jeon18] K. E. Jeon, J. She, P. Soonsawad, and P. C. Ng. “BLE beacons for internet of things applications: Survey, challenges, and opportunities”. *IEEE Internet of Things Journal*, Vol. 5, No. 2, pp. 811–828, 2018.
- [Joao19] P. João, D. Nuno, S. F. Fábio, and P. Ana. “A cross-analysis of block-based and visual programming apps with computer science student-teachers”. *Education Sciences*, Vol. 9, No. 3, p. 181, 2019.
- [Kell05] C. Kelleher and R. Pausch. “Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers”. *ACM Computing Surveys (CSUR)*, Vol. 37, No. 2, pp. 83–137, 2005.

- [Mann 19] A. Mannengal. “Digital twin language definition”. <https://github.com/Azure/IoTPlugandPlay/tree/master/DTDL>, 2019. Online, aufgerufen am 04.03.2020.
- [Mill 98] E. Miller. “An introduction to the resource description framework”. *Bulletin of the American Society for Information Science and Technology*, Vol. 25, No. 1, pp. 15–19, 1998.
- [Myer 90] B. A. Myers. “Taxonomies of visual programming and program visualization”. *Journal of Visual Languages & Computing*, Vol. 1, No. 1, pp. 97–123, 1990.
- [Niel 94] J. Nielsen. “Usability inspection methods”. In: *proceedings of the Conference companion on Human factors in computing systems*, pp. 413–414, 1994.
- [Nunn 94] J. C. Nunnally. *Psychometric theory 3E*. Tata McGraw-Hill Education, 1994.
- [Past 17] E. Pasternak, R. Fenichel, and A. N. Marshall. “Tips for creating a block language with blockly”. In: *proceedings of the 2017 IEEE Blocks and Beyond Workshop (B&B)*, pp. 21–24, IEEE, 2017.
- [Perw 19] Y. Perwej, K. Haq, F. Parwej, M. Mumdouh, and M. Hassan. “The internet of things (IoT) and its application domains”. *Int. J. Comput. Appl.*, Vol. 182, No. 49, pp. 36–49, 2019.
- [Pfis 11] D. Pfisterer, K. Romer, D. Bimschas, O. Kleine, R. Mietz, C. Truong, H. Hasemann, A. Kröller, M. Pagel, M. Hauswirth, *et al.* “Spitfire: toward a semantic web of things”. *IEEE Communications Magazine*, Vol. 49, No. 11, pp. 40–48, 2011.
- [Quoc 12] H. N. M. Quoc, M. Serrano, D. Le-Phuoc, and M. Hauswirth. “Super stream collider-linked stream mashups for everyone”. pp. 11–15, 2012.
- [Ray 17] P. P. Ray. “A survey on visual programming languages in internet of things”. *Scientific Programming*, Vol. 2017, pp. 1231430:1–1231430:6, 2017.
- [Resn 09] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, *et al.*

- “Scratch: programming for all”. *Communications of the ACM*, Vol. 52, No. 11, pp. 60–67, 2009.
- [Rich 14] M. L. Richard Cyganiak, David Wood. “RDF 1.1 concepts and abstract syntax”. <https://www.w3.org/TR/rdf11-concepts/>, 2014. Online, aufgerufen am 04.03.2020.
- [Roch 17] Á. Rocha, A. M. Correia, H. Adeli, L. P. Reis, and S. Costanzo. *Recent Advances in Information Systems and Technologies*. Vol. 3, Springer, 2017.
- [Russ 08] A. Russell, P. R. Smart, D. Braines, and N. R. Shadbolt. “NITELIGHT: A graphical tool for semantic query construction”. In: *Proceedings of Semantic Web User Interaction Workshop (SWUI 2008)*, 2008.
- [Seab 06] A. Seaborne and E. Prud’hommeaux. “Sparql query language for RDF”. *W3C recommendation (January 2008)*, 2006.
- [Sema 17] “Semantic sensor network ontology”. <https://www.w3.org/TR/vocab-ssn/>, 2017. Online, aufgerufen am 04.03.2020.
- [Spor 14] M. Sporny, D. Longley, G. Kellogg, M. Lanthaler, and N. Lindström. “JSON-LD 1.0”. *W3C recommendation*, Vol. 16, p. 41, 2014.
- [The 13] The W3C SPARQL Working Group. “SPARQL 1.1 overview”. <https://www.w3.org/TR/sparql11-overview/>, 2013. Online, aufgrufen am 04.03.2020.
- [Toml 17] M. Tomlein and K. Grønbæk. “A visual programming approach based on domain ontologies for configuring industrial IoT installations”. In: *Proceedings of the seventh international conference on the internet of things*, pp. 1–9, 2017.
- [Tull 04] T. S. Tullis and J. N. Stetson. “A comparison of questionnaires for assessing website usability”. In: *proceedings of the Usability professional association conference*, Minneapolis, USA, 2004.
- [Wein 17] D. Weintrop, D. C. Shepherd, P. Francis, and D. Franklin. “Blockly goes to work: Block-based programming for industrial robots”. In: *proceedings of the 2017 IEEE Blocks and Beyond Workshop (B&B)*, pp. 29–36, IEEE, 2017.