FORSCHUNGSINITIATIVE Zukunft BAU

F 3007

Dirk Timmermann, Vlado Altmann, Jan Skodzik,
Tim Wegner, Arne Wall, Hannes Raddatz

Offene Schnittstellen im Smart Home unter Verwendung semantischer Plug and Play Technologien

Bundesinstitut
für Bau-, Stadt- und
Raumforschung
im Bundesamt für Bauwesen
und Raumordnung

Fraunhofer IRB Verlag

F 3007

2017

ISBN 978-3-8167-9918-4

Fraunhofer IRB Verlag
Fraunhofer-Informationszentrum Raum und Bau

Postfach 80 04 69
70504 Stuttgart

Nobelstraße 12
70569 Stuttgart

Telefon 07 11 9 70 - 25 00
Telefax 07 11 9 70 - 25 08

E-Mail irb@irb.fraunhofer.de

www.baufachinformation.de

www.irb.fraunhofer.de/bauforschung

FORSCHUNGSINITIATIVE Zukunft BAU

Offene Schnittstellen im Smart Home unter Verwendung semantischer Plug&Play-Technologien

Endbericht

Der Forschungsbericht wurde mit Mitteln der Forschungsinitiative Zukunft Bau des Bundesinstitutes für Bau-, Stadt- und Raumforschung gefördert.
(Aktenzeichen: II 3-F20-13-3-001/SWD-10.08.18.7-13.12)
Die Verantwortung für den Inhalt des Berichtes liegt beim Autor.

Bearbeiter: Dr.-Ing. Vlado Altmann, Dr.-Ing. Jan Skodzik, Dr.-Ing. Tim Wegner, M. Sc. Arne Wall, M. Sc. Hannes Raddatz

Projektleiter: Prof. Dr. Dirk Timmermann

Projektbeginn: 01.10.2013

Projektlaufzeit: 33 Monate

22.06.2016

Universität Rostock Traditio et Innovatio

Kurzfassung des Berichts

Das Ziel des Projektes ist der Einsatz von semantischen Plug&Play-Technologien wie Web Services bzw. Devices Profile for Web Services (DPWS) im Smart Home. Im Rahmen des Projektes wurde zunächst der Stand der Technik gründlich analysiert. Dabei wurden die Kommunikationsgrundlagen untersucht, u.a. bezüglich Web Services, die für die Durchführung des Projektes erforderlich sind. Anschließend erfolgte eine Analyse der in der klassischen Gebäudeautomation zum Einsatz kommenden Technologien und der aktuellen Technologien im Smart Home.

Damit bei einem Einsatz von neuen Technologien die alte Technik nicht vollständig verdrängt werden muss, wurde als erster Aspekt des Projektes die Integration der Legacy-Technologien in Web-Service-Umgebungen untersucht. Ein kompletter Umstieg auf eine neue Technologie ist mit höheren Kosten verbunden, da alle bestehenden Anlagen/Installationen in diesem Fall vollständig modernisiert werden müssen. Eine kostengünstige Lösung ist dagegen eine schrittweise Modernisierung. Dabei werden nur einige Geräte ersetzt oder neue Geräte installiert. Die bestehenden Geräte sollen weiterhin genutzt und in die neue Installation integriert werden. Um dieses Ziel zu erreichen, müssen alte Geräte mittels eines Gateways mit DPWS-Geräten kommunizieren können.

Dies wird am Beispiel eines ausgewählten Protokolls gezeigt. Ausgehend von der Analyse existierender Protokolle wird als geeignetes Protokoll BACnet ausgewählt. BACnet bietet im Vergleich zu anderen Protokollen die größte Funktionsvielfalt. Darüber hinaus existieren bereits mehrere Gateway-Umsetzungen von BACnet auf andere Protokolle wie z.B. KNX, LON, ZigBee. Dadurch wird es ermöglicht, andere Protokolle in das DPWS-Gateway zu integrieren.

Zu Beginn wurden die Standards DPWS und BACnet unabhängig voneinander analysiert. Es wird zum einen betrachtet, welche Möglichkeiten der jeweilige Standard bietet, um Informationen im Netzwerk darzustellen und wie die Kommunikation zwischen den Geräten abläuft. Zusätzlich wird BACnet/WS, eine Form der Darstellung von Gebäudedaten mit Web Services, betrachtet. Im Anschluss findet ein Vergleich von BACnet und DPWS hinsichtlich der gewünschten Gateway-Funktionalitäten statt. Um Geräteinformationen protokollunabhängig im Gateway speichern zu können, wird eine Struktur entwickelt, in der die einzelnen Komponenten der Geräte auf Entities abgebildet werden.

Für die Implementierung des BACnet-DPWS-Gateways wurde das OSGi-Framework Eclipse Equinox als Basisframework verwendet. Das Gateway besteht aus einem zentralen Gateway-Bundle und jeweils einem Gate-Bundle für die Standards BACnet und DPWS. Die Bundles verwenden definierte Schnittstellen, wobei die Schnittstellen der Gates identisch und protokollunabhängig sind.

Durch die vom Kommunikationsprotokoll unabhängigen Gate-Schnittstellen und die protokollneutrale Gateway-interne Darstellung von Geräten ist eine Grundlage für Gates anderer Protokolle in der Gebäudeautomation gelegt. Ein Gate muss nur dazu in der Lage sein, eine Struktur von Entities auf eine dem Standard entsprechende Darstellungsform zu übertragen sowie eine Anordnung von Entities für Geräte des Standards zu erstellen. Somit kann aus dem Gateway für die Standards BACnet und DPWS ein Gateway für die Gebäudeautomation entstehen.

Als nächster Aspekt des Projekts wurde Smart Metering im Smart Home untersucht. Smart Metering stellt eine der tragenden Säulen im Smart Home dar, da die Nutzer nicht nur Geräte steuern, sondern auch Informationen über das Haus wie z.B. Tagesverbrauch von Strom, Wasser oder Gas erhalten möchten. Solche Informationen werden in den meisten Fällen zwar von den Zählern erfasst, können jedoch aufgrund einer fehlenden Kommunikationseinheit nicht weitergegeben werden. Dieses Problem soll von intelligenten Zählern, den Smart Metern, gelöst werden. Die Einführung der Smart Meter in Deutschland hat sich jedoch stark verlangsamt. Es stellt sich damit die Frage, wie eine Realisierung des Smart Home in naher Zukunft ohne flächendeckenden Einsatzes der Smart Meter möglich ist. Eine Möglichkeit wäre, existierende Zähler für Strom, Gas und Wasser mit einer kostengünstigen Kommunikationseinheit zu ergänzen. Die herkömmlichen Zähler sind jedoch in der Regel mechanisch, sodass eine Erweiterung der Funktionalität schwierig ist. Zu den Aufgaben der Kommunikationseinheit würde folglich auch das Ablesen des Zählers gehören. Um die Lösung möglichst kostengünstig zu gestalten, soll die Kommunikationseinheit in der Lage sein, beliebige Zählerstände ablesen zu können und den abgelesenen Wert über eine Kommunikationsschnittstelle zu übertragen. Durch die vielfältige Beschaffenheit der Zähler ist es einheitlich nur möglich, an den Zählerstand direkt über die Anzeige zu gelangen. Die Anzeige muss somit optisch erfasst und der Zählerstand ausgewertet werden. Hierdurch könnten beliebige Zähler kostengünstig zu Smart Metern aufgerüstet werden. Eine solche Lösung kann darüber hinaus schnell umgesetzt werden und dabei die Einführung des Smart Home deutlich beschleunigen.

Eines der Ziele dieses Projektes war es daher, ein kostengünstiges System zu entwickeln, das den Zählerstand eines analogen Stromzählers in kurzen Intervallen ausliest und diese Daten über einen Web Service im lokalen Netzwerk verfügbar macht. Hierfür wird in diesem Bericht zunächst ein kurzer Überblick über den Stand der Technik der Bildverarbeitung und Mustererkennung gegeben. Anschließend wird ein Konzept vorgestellt, mit dem alte Ferraris-Zähler zu Smart Metern aufgerüstet werden können. Da es sich dabei um ein sehr komplexes System handelt, wird es in mehrere Module unterteilt. Diese sind Bilderfassung, Vorverarbeitung, Detektion der „Regions of Interest“, Segmentierung, Suche nach Zeichenketten, Ziffernerkennung und Bereitstellung der Daten über einen Web Service. Als

Plattform wird der Raspberry Pi gewählt, da dieser trotz geringer Anschaffungs- und Verbrauchskosten eine relativ hohe Rechenleistung erreicht. Die Bilder werden mit einer ebenfalls kostengünstigen Kamera aufgenommen, die direkt an den Raspberry Pi angeschlossen wird. Damit ist ein abgeschlossenes System entstanden, das die gestellten Aufgaben vollständig übernehmen kann. Die Verbrauchsdaten können von beliebigen anderen eingebetteten Systemen, PCs oder Smartphones abgerufen werden. Das System wurde darüber hinaus in verschiedenen Ausführungen implementiert, die sich in Erkennungsrate und Laufzeit für eine Zählerstanderkennung unterscheiden. Das beste System erreicht dabei eine gute Erkennungsrate von 98%. Die Zeit für die Verarbeitung eines Bildes des Zählerstands inklusive Bildaufnahme beträgt maximal 1,7 Sekunden. Hiermit ist es möglich, dem Nutzer eine vollfunktionierende und kostengünstige Erweiterung für alte Zähler anzubieten, die es ermöglicht, aktuelle Daten über den Stromverbrauch, grundsätzlich aber auch über Wasser- und Gasverbrauch zur Verfügung zu stellen. Damit kann die Zeit bis zur Einführung und zum Einbau der Smart Meter überbrückt und die Vernetzung von Smart Home und Smart Metering schon jetzt vorangetrieben werden.

Ein weiterer Aspekt ist die nutzerfreundliche Konfiguration von Smart Home Geräten durch den Anwender. Dazu wurde ein Konzept entwickelt, welches die dezentrale Konfiguration solcher Geräte ermöglicht. Der Benutzer kann Geräte miteinander verknüpfen und Regeln für deren Interaktion mittels eines Endgerätes wie Smartphone oder Computer aufstellen. Dabei findet die Kommunikation allein zwischen den Geräten und dem Nutzergerät statt. Dies ist ein Vorteil der dezentralen Konfiguration im Gegensatz zur Kommunikation über einen zentralen Hub, welche häufig in proprietären Lösungen angewendet wird.

Das Fehlen eines Datenmodells in DPWS stellt eine Hürde für die herstellerunabhängige Kommunikation zwischen DPWS-Geräten dar. Gleiche Geräte unterschiedlicher Hersteller können verschiedene Daten für die Konfiguration benötigen. Um diese Hürde zu überwinden, wurde eine Datenbank im Internet eingerichtet. Entwickler von DPWS-Geräten können so genannte WSDL-Dateien, welche eine Beschreibung der individuellen Implementierung enthalten, hochladen und die Dateien anderer Hersteller einsehen. Somit wurde eine Plattform geschaffen, die den Austausch von Gerätedaten und die Etablierung von Standards für Konfigurationsdaten von spezifischen Gerätetypen ermöglicht.

Abschließend wurde eine Demonstrationsumgebung aufgebaut, in der die in diesem Projekt entwickelten Konzepte und Mechanismen umgesetzt und veranschaulicht werden. Der Fokus liegt dabei im Besonderen auf der dezentralen Konfiguration von DPWS-Geräten, der Einbindung von Legacy-Technologien und der kamera-basierten Zählerstanderfassung.

Die im Projektantrag genannten Arbeitsphasen

- Untersuchung des Standes der Technik
- Definition von Use Case und Anwendungsszenarios
- Ableitung allgemeiner Anforderungen und Spezifikation des Kommunikationsprotokolls
- Ausarbeitung einer offenen Kommunikationslösung für das Smart Home
- Spezifikation der Infrastruktur / Architektur des Gesamtsystems
- Zusammenfassung der Ergebnisse
- Umsetzung einer Demonstration für das Anwendungsszenario

wurden damit vollständig abgeschlossen. Das Projekt liegt damit exakt im Zeitplan.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

6LoWPAN ..... IPv6 over Low power Wireless Personal Area Network
ANSI ..... American National Standards Institute
ARCNET ..... Attached Resource Computer Network
ARM ..... Advanced RISC Machines
ASCII ..... American Standard Code for Information Interchange
ASHRAE ..... American Society of Heating, Refrigeration, and Air-Conditioning Engineers
BACnet ..... Building Automation and Control Networks
BACnet/WS ..... BACnet Web Services Interface
BACnet4J ..... BACnet for Java
BIBB ..... BACnet Interoperability Building Blocks
BP ..... Backpropagation
BSI ..... Bundesamt für Sicherheit in der Informationstechnik
CEA ..... Consumer Electronics Association
CNN ..... Convolutional Neuronal Network
CoAP ..... Constrained Application Protocol
COV ..... Change of Value
CR ..... Carriage Return
DIN ..... Deutsches Institut für Normung
EEP ..... EnOcean Equipment Profile
EHS ..... European Home Systems
EIA ..... Electronic Industries Alliance
EIB ..... Europäischer Installationsbus
EN ..... Europäische Norm
ETS ..... Engineering Tool Software
FFD ..... Full Function Devices
HDMI ..... High Definition Multimedia Interface
HTTP ..... Hypertext Transfer Protocol
HTTPS ..... Hypertext Transfer Protocol Secure
ICDAR ..... International Conference on Document Analysis and Recognition
IDE ..... Integrated Development Environment
IEC ..... International Electrotechnical Commission
IOB ..... Interoperabilitätsbereich
IoT ..... Internet of Things
IR ..... Infrarot
ISO ..... International Organization for Standardization
ITU ..... International Telecommunication Union
ITU-T .... International Telecommunication Union Telecommunication Standardization Sector
JMEDS ..... Java Multi Edition DPWS Stack
LF ..... Line Feed
LON ..... Local Operation Network
LWL ..... Lichtwellenleiter
M2M ..... Machine-to-Machine
MIME ..... Multipurpose Internet Mail Extensions
MLP ..... Multilayer Perceptron
MSTP ..... Master-Slave/Token-Passing
OASIS ..... Organization for the Advancement of Structured Information Standards
OS ..... Operating System
OSGi ..... Open Services Gateway initiative

OSI ..... Open Systems Interconnection
PAN ..... Personal Area Network
PICS ..... Protocol Implementation Conformance Statement
PL ..... Powerline
QName ..... Qualified Name
RAM ..... Random Access Memory
REST ..... Representational State Transfer
RF ..... Radio Frequency
RFD ..... Reduced Function Device
RGB ..... Rot, Grün, Blau
RISC ..... Reduced Instruction Set Computer
ROI ..... Regions of Interest
ROM ..... Read Only Memory
SCPT ..... Standard Configuration Property Type
SFPT ..... Standard Functional Profile Type
SNVT ..... Standard Network Variable Type
SO ..... Shared Object
SOA ..... Service-Oriented Architecture
SOAP ..... Simple Object Access Protocol
SPoF ..... Single Point of Failure
SRD ..... Short Range Devices
SSD ..... Solid State Drive
TLS ..... Transport Layer Security
TP ..... Twisted Pair
UDDI ..... Universal Description, Discovery and Integration
UPnP ..... Universal Plug and Play
URI ..... Uniform Resource Identifier
URL ..... Uniform Resource Locator
USB ..... Universal Serial Bus
UUID ..... Universally Unique Identifier
VM ..... Virtuelle Maschine
WS ..... Web Service
WS.I ..... Web Services Interoperability Organization
WSDL ..... Web Services Description Language
WWW ..... World Wide Web
XML ..... Extensible Markup Language
XSD ..... XML Schema Definition
ZCL ..... ZigBee Cluster Library
ZDO ..... ZigBee Device Object

1. Einführung

Die derzeitige Situation im Smart Home-Bereich ist dadurch gekennzeichnet, dass überwiegend proprietäre Technologien eingesetzt werden. Diese Tatsache führt zu einer geringen Interoperabilität der Geräte von verschiedenen Herstellern. Darüber hinaus bedürfen diese Technologien für die Installation und Inbetriebnahme eines umfangreichen Fachwissens. Das erschwert in großem Maße die Verbreitung der intelligenten Umgebungen, da das Installationspersonal oft nicht über das notwendige Wissen verfügt. Durch den Einsatz semantischer Plug&Play-Technologien entfällt die Installationshürde. Darüber hinaus soll der Benutzer in die Lage versetzt werden, sein Smart Home-System selbstständig mit neuen Geräten zu erweitern. Embedded Web Services stellen eine domänenübergreifende Internettechnologie zur Vernetzung von (eingebetteten) Systemen dar, und zwar vom miniaturisierten drahtlosen Sensor bis hin zum kabelgebundenen Großgerät. Ein Vertreter der Web Service-Technologie für eingebettete Systeme ist „Devices Profile for Web Services“ (DPWS). DPWS ermöglicht eine mediumunabhängige IP-Kommunikation, die auf der bewährten Web Service-basierten Architektur beruht und dabei auf etablierten Sicherheitsstandards fußt.

Die Verwendung von Web Services benötigt keine spezielle Hardware und ermöglicht damit deutlich kostengünstigere Geräte. Die dynamische und flexible Struktur der Web Services erlaubt die Entwicklung spezieller Profile, die auf ein gewisses Einsatzgebiet oder eine Geräteart abgestimmt sind. Damit können die Web Services als ein übergreifender und harmonisierender Standard eingesetzt werden. Die Web Services ermöglichen die Bildung einer geräte- und technologieunabhängigen Vernetzung/Infrastruktur in heterogenen Geräteumgebungen und sollen diesbezüglich auch auf die Eignung als Kommunikationsstandard im Smart Home untersucht werden. Zudem bieten die Web Services die Möglichkeit einer barrierefreien Vernetzung durch die Nutzung bestehender und weit verbreiteter Internetstandards.

Das Ziel dieses Forschungsvorhabens ist es, eine geeignete Plug&Play-Lösung für das Smart Home herauszuarbeiten. Dabei werden die existierenden Plug&Play-Technologien wie DPWS auf die Möglichkeit eines Einsatzes in Smart Home untersucht. Die Einbindung bereits existierender Systeme wird evaluiert.

2. Stand der Technik

In diesem Kapitel wird auf den Stand der Technik eingegangen, der aktuelle Technologien im Bereich der Gebäudeautomation und des Smart Home widerspiegelt.

2.1. ISO/OSI-Referenzmodell

Die International Organization for Standardization (ISO) hat mit dem Open Systems Interconnection (OSI)-Referenzmodell ein Schichtenmodell entwickelt, welches die Netzwerkkommunikation in 7 Schichten aufteilt. Das Modell beschreibt auf allgemeingültige Weise, wie die Rechner vernetzt werden sollen. Jedes Protokoll kann demnach einer bestimmten Schicht im Referenzmodell zugeordnet werden [1]. Da Rechnerkommunikation oft über das globale Netzwerk Internet abläuft und dabei nur 5 Schichten des ISO/OSI-Referenzmodells umfasst, wurde ein reduziertes Referenzmodell – das Internet-Modell – eingeführt [2]. Die beiden Referenzmodelle sind in Abbildung 1 gegenübergestellt.

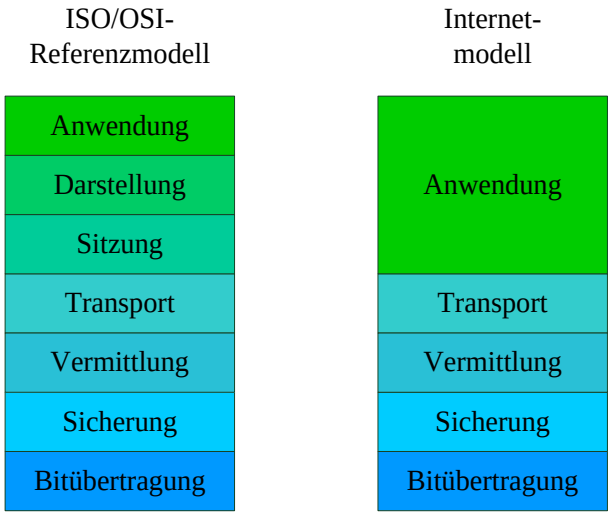


Abbildung 1: ISO/OSI-Referenzmodell und Internet-Modell im Vergleich

Die *Bitübertragungsschicht* (Schicht 1) steuert die Übertragung des Bit-Streams über das physikalische Medium. Sie beschäftigt sich mit den mechanischen und elektrischen Spezifikationen des Interfaces und des Übertragungsmediums. Die Bitübertragungsschicht definiert darüber hinaus die Repräsentation der Bits auf dem Übertragungsmedium (Codierung), die Datenrate, die Synchronisation des Senders und des Empfängers, die physikalische Topologie und den Übertragungsmodus (Simplex, Halbduplex, Vollduplex). Simplex ermöglicht die Übertragung auf einem Kanal nur in eine Richtung. Halbduplex ermöglicht die Übertragung in beide Richtungen, jedoch nicht gleichzeitig. Bei Vollduplex ist eine parallele Übertragung in beide Richtungen möglich.

Die *Sicherungsschicht* (Schicht 2) wandelt eine reine Bitübertragung in eine zuverlässige Verbindung um. Dabei wird der Bit-Stream in die besser handhabbaren Dateneinheiten

(Frames) unterteilt. Des Weiteren bietet die Sicherungsschicht eine Adressierung der Geräte an. Damit können die Sender und Empfänger identifiziert werden. Eine weitere Aufgabe dieser Schicht ist die Fehlerkontrolle. Dadurch können fehlerhafte oder verlorene Frames detektiert und gegebenenfalls neu übertragen werden. Zu den weiteren wichtigen Aufgaben der Sicherungsschicht gehört die Zugriffskontrolle. Wenn mehrere Sender gleichzeitig Daten übertragen wollen, bestimmt die Zugriffskontrolle, welcher Sender das Medium für sich beanspruchen kann. Darüber hinaus kann die Sicherungsschicht eine Flusssteuerung anbieten. Dadurch kann die Rate, mit der der Sender die Daten erzeugt, auf die maximal mögliche Rate, mit der der Empfänger Daten empfangen kann, angepasst werden. Dadurch lässt sich eine Überlastung des Empfängers verhindern.

Die *Vermittlungsschicht* (Schicht 3) ist für die Zustellung der Pakete von der Quelle zur Senke zuständig. Die Zustellung kann dabei über mehrere Netze hinweg stattfinden. Hierfür bietet die Vermittlungsschicht eine logische Adressierung an. Um das Ziel erreichen zu können, gehört das Routing zu den Aufgaben dieser Schicht. Das Routing bestimmt dabei den nächsten Hop (Zwischenstation) auf dem Pfad von der Quelle zur Senke. Geräte, die mehrere Netze verbinden, werden Router genannt.

Die *Transportschicht* (Schicht 4) ist für die Zustellung der Nachrichten zwischen den Prozessen zuständig. Ein Prozess ist ein Programm, welches auf einem Host ausgeführt wird. Zu den Aufgaben der Transportschicht gehört die Adressierung der Prozesse. Auf einem Host können gleichzeitig mehrere Programme ausgeführt werden, die über das Netzwerk mit verschiedenen Netzwerkteilnehmern kommunizieren. Die Adressierung der Prozesse ermöglicht eine Zuordnung der Pakete zu einem bestimmten Programm. Die Transportschicht unterstützt zwei Kommunikationsarten. Bei einer verbindungslosen Kommunikation wird jedes Paket als eine unabhängige Dateneinheit betrachtet. Bei einer verbindungsorientierten Kommunikation wird zunächst eine sichere Verbindung zwischen dem Sender und dem Empfänger aufgebaut. Anschließend findet der Datenaustausch statt. Nach dem Datenaustausch wird die Verbindung abgebaut. Die Transportschicht ermöglicht, die Daten auf mehrere Pakete im Sender aufzuteilen (segmentieren) und im Empfänger wieder zusammenzusetzen (reassemblieren). Für eine korrekte Datenwiederherstellung spielt die Reihenfolge der Pakete eine Rolle. Hierfür werden die Pakete mit einer Sequenznummer gekennzeichnet. Ähnlich der Sicherungsschicht bietet die Transportschicht eine Fehlererkennung und eine Flusssteuerung. Diese beziehen sich jedoch auf die Interprozesskommunikation und nicht auf die einzelnen physikalischen Verbindungen [2].

Die *Sitzungsschicht* (Schicht 5) dient der Einrichtung, Aufrechterhaltung und Synchronisation einer Interaktion zwischen zwei kommunizierenden Netzwerkteilnehmern. Die *Darstellungsschicht* (Schicht 6) beschäftigt sich mit der Syntax und Semantik der Informationen, die zwischen zwei Netzwerkteilnehmern ausgetauscht werden. Dazu zählen

Datenkonvertierung, Verschlüsselung und Kompression. Die *Anwendungsschicht* (Schicht 7) dient der Übertragung beliebiger Daten einer Anwendung wie z.B. Email.

Damit jede Netzwerkschicht ihren Aufgaben nachkommen kann, müssen die erforderlichen Steuerinformationen in den Datenframe hinzugefügt werden. Die Steuerinformationen einer Netzwerkschicht werden als sogenannte Header den Daten der jeweils übergeordneten Schicht vorangestellt.

2.2. Software-Architektur-Stile

Durch eine konstant steigende Verbreitung der Netzwerktechnologien in nahezu allen Bereichen des Lebens sind im Laufe der Entwicklung viele neue Protokolle entstanden. Die Komplexität der Protokolle und die zu lösenden Aufgaben haben ebenfalls zugenommen. Um Lösungen für zukünftige netzwerktechnische Anwendungen schneller entwickeln zu können, war es notwendig, Technologien bereitzustellen, die die Konzipierung und Implementierung der Anwendungsprotokolle vereinfachen und automatisieren. Zwei Architektur-Stile wurden als Lösung für das Problem vorgeschlagen. Diese sind die Ressourcen-orientierte Architektur, auch unter der Bezeichnung Representational State Transfer (REST)-Architektur bekannt, und die Service-orientierte Architektur (SOA) [1].

Die SOA-basierte Herangehensweise zur Lösung des Problems bietet die Möglichkeit, Protokolle zu generieren, die auf jede beliebige Anwendung angepasst werden können. Die Kernelemente von SOA sind ein Framework für die Protokollspezifikation, Software-Toolkits für eine automatische Generierung einer Protokollimplementierung aus der Spezifikation sowie modulare Teilspezifikationen, die in Protokollen benutzt werden können.

Die REST zugrundeliegende Idee ist, alle Informationen und Funktionen als Ressourcen zu behandeln. Die Ressourcen werden durch Uniform Resource Identifier (URI) bzw. Uniform Resource Locator (URL) gekennzeichnet und über das Hypertext Transfer Protocol angesprochen. Die REST-Architektur spiegelt die Web-Architektur wider. Im Folgenden wird auf die einzelnen Architektur-Stile eingegangen.

2.2.1. Ressourcen-orientierte Architektur

Die REST-Architektur wurde zum ersten Mal in der Dissertation von Fielding beschrieben [3]. Die Architektur basiert auf dem Client-Server-Modell. Die Kommunikation zwischen dem Client und dem Server ist zustandslos (stateless). Das bedeutet, dass der Sitzungszustand komplett vom Client überwacht werden muss. Jede Anfrage vom Client an den Server muss alle Information enthalten, die für die Verarbeitung dieser Anfrage notwendig sind. Die Antworten vom Server können zwischengespeichert (gecached) werden. Solange die zwischengespeicherten Daten gültig sind, können sie als Antworten für erneute Anfragen benutzt werden, ohne eine neue Anfrage an den Server zu schicken. Die REST-Architektur

benutzt ein uniformes Interface, welches von der Anwendung unabhängig ist. Jede beliebige Information wird als eine Ressource aufgefasst [4].

Jede beliebige Ressource kann mittels URI adressiert werden. Eine URI identifiziert vier Parameter: Das Protokoll für den Zugriff auf die Ressource, den Host, den Port und den Pfad zur Ressource auf dem Host. Die URI wird im Klartext angegeben und besitzt das folgende Muster: *Protocol://Host:Port/Path* [2]. Es können auch weitere optionale Parameter wie Query und Fragment folgen. Eine vollständige Beschreibung des URI-Aufbaus kann der Spezifikation RFC 3986 entnommen werden [5].

Jede Ressource hat eine Repräsentation, die eine Abfolge von Bytes in einem bestimmten Format darstellt. Das uniforme Interface definiert vier Methoden, die auf eine Ressource angewendet werden können. Diese sind in Tabelle 1 aufgelistet [6].

Methode	Beschreibung
GET	Abruf der Repräsentation einer Ressource
POST	Versand von Informationen an eine Ressource
PUT	Neue Ressource erstellen
DELETE	Eine Ressource löschen

Tabelle 1: Methoden des uniformen Interfaces der REST-Architektur

In den folgenden Abschnitten wird auf die Vertreter der REST-Architektur eingegangen.

2.2.2. Hypertext Transfer Protocol (HTTP)

Das Hypertext Transfer Protocol (HTTP) ist ein Protokoll der Anwendungsschicht des ISO/OSI-Referenzmodells. HTTP ist ein einfaches Request-Response-Protokoll. In der Transportschicht wird für eine sichere Übertragung TCP eingesetzt [7]. HTTP ist ein textbasiertes Protokoll. Die Request- und Response-Header werden im American Standard Code for Information Interchange (ASCII) codiert. HTTP wird vor allem für den Zugriff auf World Wide Web (WWW)-Inhalte eingesetzt. Darüber hinaus wird HTTP als ein Transportprotokoll für andere Anwendungen wie z.B. Machine-to-Machine (M2M)-Kommunikation eingesetzt und bildet eine Grundlage für REST.

Eine HTTP-Request-Nachricht wird vom Client an den Server geschickt und besteht aus einer Request-Zeile, den Headern und einem optionalen Body (Nachrichten-Payload). Eine HTTP-Response-Nachricht wird vom Server an den Client geschickt und beinhaltet eine Statuszeile, die Header und einen optionalen Body. Die Request-Zeile beinhaltet eine Methode, die auf eine Ressource angewendet werden soll. Neben den Standardmethoden, die in Tabelle 1 aufgelistet sind, bietet HTTP vier weitere Methoden an: HEAD, TRACE, CONNECT und OPTION. Die Beschreibung der zusätzlichen Methoden kann der HTTP-Spezifikation RFC

2616 entnommen werden [8]. Der Methode folgt in der Request-Zeile ein URL, der die angefragte Ressource eindeutig identifiziert. Auf diese Ressource soll die angegebene Methode angewendet werden. Die Request-Zeile wird mit einer HTTP-Versionsnummer abgeschlossen [2]. Die Statuszeile enthält einen Statuscode, der angibt, ob das Request erfolgreich war oder nicht, und wenn nicht, warum. Zusätzlich zum Code wird der Status auch in Textform angegeben.

Nach der Request- bzw. Status-Zeile werden die Header übertragen. Diese können Request- oder Response-spezifisch oder für beide Nachrichtentypen vorgesehen sein. Die Header werden ebenfalls in Klartext übertragen. Die Aufgabe der Header ist, zusätzliche Informationen zwischen dem Client und dem Server auszutauschen. Alle Header werden nach demselben Muster aufgebaut: *HeaderName: HeaderValue*. Ein Header wird mit den aufeinanderfolgenden ASCII-Zeichen Carriage Return (CR) und Line Feed (LF) abgeschlossen, die einen Zeilenumbruch darstellen. Das Ende des Header-Teils wird mit dem zweifachen CRLF (einer leeren Zeile) signalisiert.

Nach dem Header-Teil folgt der Body. Dieser stellt die HTTP-Payload dar. Die Codierung und die Art des Body wird durch den Header *Content-Type* festgelegt. Der Body kann einer der vielen Multipurpose Internet Mail Extensions (MIME)-Typen sein.

2.2.3. Constrained Application Protocol (CoAP)

Constrained Application Protocol (CoAP) wurde für den Einsatz in eingebetteten Systemen mit geringem Energieverbrauch wie z.B. Sensornetzwerke konzipiert. Die Entwicklung von CoAP wurde durch HTTP inspiriert, weist aber dennoch signifikante Unterschiede auf. CoAP ist ebenfalls ein Anwendungsschichtprotokoll und ermöglicht es, die REST-Architektur auf Geräten mit eingeschränkten Ressourcen zu nutzen und diese so in das „Internet of Things“ (IoT) einzubinden [9]. CoAP wurde im Juni 2014 als RFC 7252 standardisiert [10].

Im Gegensatz zu HTTP ist CoAP ein binäres Protokoll. Diese Protokollart wurde für eine M2M-Kommunikation konzipiert und ermöglicht kompaktere Nachrichten, da der Inhalt nicht für Menschen lesbar (engl.: human-readable) sein muss. Demzufolge sind die CoAP-Request- und CoAP-Response-Nachrichten deutlich kleiner als die von HTTP. Ein weiterer signifikanter Unterschied ist die Nutzung von UDP zur Übertragung der CoAP-Nachrichten. CoAP verwendet die zugrundeliegenden REST-Methoden wie in Tabelle 1 dargestellt. Zur Ressourcenadressierung finden ebenfalls URIs Verwendung, die jedoch nicht komplett in Klartext übertragen werden. Da die Nachrichtenübertragung mittels UDP keine Nachrichtenzustellung garantiert, bringt CoAP einen eigenen Mechanismus für die zuverlässige Übertragung mit. Die CoAP-Nachrichten können „Confirmable (CON)“ (dt.: bestätigt) oder „Non Confirmable (NON)“ (dt.: nicht bestätigt) sein. CON-Nachrichten müssen von dem CoAP-Server bestätigt werden. Die Bestätigung kann sowohl zusammen mit

einer Response-Nachricht als auch in einer separaten Bestätigungsnachricht (ACK) übertragen werden. Die tatsächliche Response-Nachricht kann dann später verschickt werden. Diese sogenannte „Delayed Response“ unterscheidet CoAP ebenfalls von HTTP. Falls der Client keine Bestätigung vom Server erhält, wird das Request wiederholt. Wenn eine Nachricht empfangen wurde, jedoch aus bestimmten Gründen nicht verarbeitet werden kann, wird dies dem Sender mit einer Reset-Nachricht (RST) signalisiert.

Ein weiterer Unterschied von CoAP zu HTTP ist die Unterstützung von Multicast. Da HTTP auf TCP basiert, ist per se kein Multicast möglich. Multicast stellt besonders in Automatisierungsszenarien eine wichtige Eigenschaft dar, da hierbei mehrere Netzwerkteilnehmer gleichzeitig informiert oder abgefragt werden können. Eine Anwendung, die darauf basiert, ist der Resource-Discovery-Mechanismus. Durch diesen Mechanismus kann ein Client mittels Multicast Ressourcen im Netzwerk finden, die bestimmten Kriterien entsprechen. CoAP unterstützt u.a. semantische Angaben wie Ressourcentyp und Interface-Typ. Das Discovery wird bei CoAP in eine separate Spezifikation namens Constrained RESTful Environments (CoRE) Link Format ausgelagert, die in RFC 6690 festgehalten wird [11].

Des Weiteren bietet CoAP einen Subscription/Notification-Mechanismus (CoAP Observe). Dieser ist ebenfalls für M2M-Anwendungen vorgesehen. Durch CoAP Observe kann der Client z.B. automatische Wertänderungen vom Server erhalten, ohne ein Request abschicken zu müssen. Voraussetzung dafür ist, dass der Client die Wertänderungen abonniert hat [12].

2.2.4. Service-orientierte Architektur (SOA)

SOA ist ein Designparadigma für die Interaktion zwischen heterogenen Komponenten unter den Gesichtspunkten der Selbstverwaltung und Interoperabilität. Jede Komponente kann bestimmte Funktionen anbieten, an denen andere Komponenten interessiert sein können. Die selbstbeschreibenden Interfaces zu diesen Funktionen werden Services genannt [13] [14]. Die Services basieren auf offenen Standards, die für Interoperabilität sorgen. SOA ermöglicht Flexibilität, Wiederverwendbarkeit, Anpassbarkeit und Integration der Services in heterogene Umgebungen. Die Services sind lose gekoppelt und können dynamisch gefunden und genutzt werden. Die ursprüngliche SOA-Architektur definiert ein Service Registry, das ein zentrales Verzeichnis für alle verfügbaren Services in einer Umgebung darstellt. Eine Implementierung eines solchen Service Registry stellte z.B. Universal Description, Discovery and Integration (UDDI) dar [15]. Es wurden später jedoch weitere Mechanismen entwickelt, die auch ein Service Discovery ohne Service Registry ermöglichen.

Eine SOA definiert drei Rollen: Service Provider, Client und ein optionales Service Registry. Der Service Provider stellt einen Service zur Verfügung, der von einem Client in Anspruch

genommen werden kann. Die Services können sich bei dem Service Registry anmelden und damit den Clients ermöglichen, danach im Service Registry zu suchen.

Jeder Service bietet eine eigene Beschreibung an, die die Eigenschaften und Funktionalitäten des Service beschreibt. Sie enthält alle Informationen, die für die Interaktion mit dem Service notwendig sind. Diese Beschreibung ist standardisiert und wird mit Extensible Markup Language (XML) dargestellt. Dadurch ist sie sowohl für Maschinen als auch für Menschen lesbar.

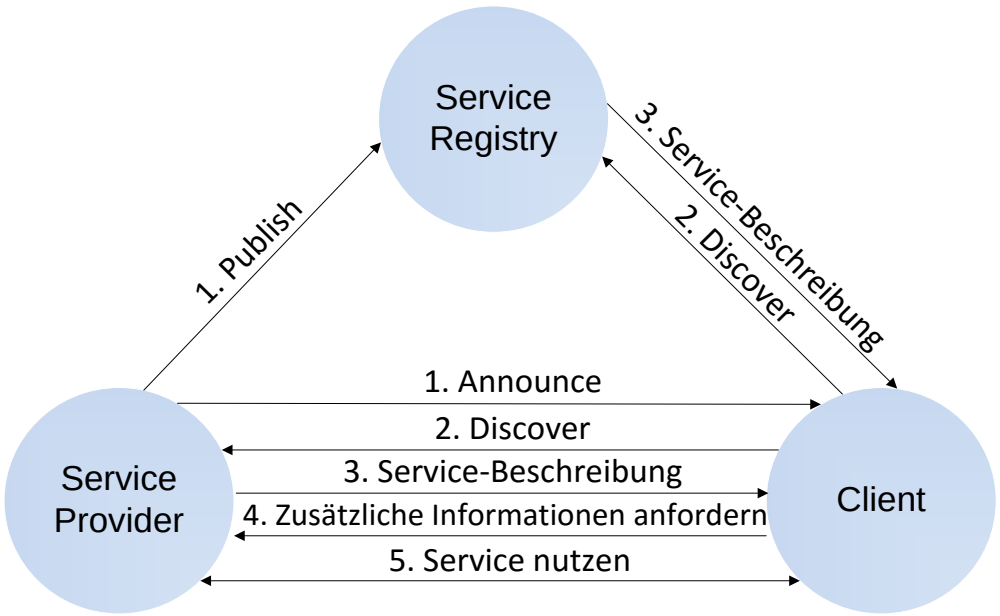


Abbildung 2: Rollen in SOA und ihre Interaktionen

Eine klassische SOA-Implementierung umfasst neben den drei Rollen fünf mögliche Interaktionen zwischen ihnen, die in Abbildung 2 illustriert sind [16]. In Schritt 1 stellt sich der Service Provider im Netzwerk vor. Dies geschieht entweder durch die Veröffentlichung (engl.: Publish) des Service im Service Registry oder durch eine Ankündigung (engl.: Announcement) im Netzwerk. In Schritt 2 schickt der Client eine Discovery-Anfrage an das Service Registry oder an das Netzwerk. Alternativ kann der Client auch auf Ankündigungen warten. In Schritt 3 erhält der Client eine Service-Beschreibung. Diese kann je nach durchgeführter Discovery-Art entweder vom Service Registry oder direkt vom Service Provider geschickt werden. In Schritt 4 fordert der Client eine detaillierte Beschreibung des Service an, die eine Interface-Beschreibung enthält. In Schritt 5 kann der Client den Service in Anspruch nehmen. Wenn der Service Provider das Netzwerk verlässt, muss er sich beim Service Registry und beim Client abmelden [14].

2.2.5. Web Services

Web Services (WS) stellen die SOA-Implementierung mit der höchsten Marktdurchdringung dar. WS stellen einen Satz aus modularen Protokollbausteinen zur Verfügung, die auf verschiedene Weise kombiniert werden können, um bestimmte Anforderungen einer Anwendung zu erfüllen. Die WS-Standards werden auch als WS-* bezeichnet. Kombinationen von WS-Standards werden Profile genannt [17]. WS stellen Standards zur Verfügung, die Service Discovery, Service-Beschreibung, Security, Policy und andere Funktionalitäten ermöglichen. WS definieren lediglich eine Schnittstelle zu einer Anwendung. Die Anwendung kann eine beliebige Implementierung besitzen und kann zur Laufzeit ausgetauscht werden. WS benutzen XML für die Daten-Codierung und Simple Object Access Protocol (SOAP) für den Datenaustausch. Diese wichtigen WS-Bestandteile werden in den folgenden Abschnitten erklärt. Anschließend wird ein spezielles WS-Profil für eingebettete Systeme namens Devices Profile for Web Services (DPWS) vorgestellt.

2.2.6. Extensible Markup Language (XML)

XML ist eine Sprache für die Beschreibung eines strukturierten Inhaltes [18]. Der Inhalt wird als Text repräsentiert. Eine vollständige XML-Beschreibung eines Inhaltes wird als XML-Dokument bezeichnet. Die sogenannten Tags (Markups) werden verwendet, um Informationen über den Inhalt bereitzustellen [1]. Die Inhalte werden zwischen den Start- und End-Tags eingeschlossen: *<tag>value</tag>*. Die XML-Syntax bietet eine verschachtelte Struktur aus Tag/Wert-Paaren, die eine Baumstruktur bilden. Solche Tag/Wert-Paare werden Elemente genannt. Zusätzliche Metainformationen zu einem bestimmten Wert können innerhalb eines Start-Tags als Schlüsselwort-Werte-Paare angegeben werden: *<tag key="keyvalue">value</tag>*. Solche Schlüsselwort-Werte-Paare werden als Attribute bezeichnet [19]. XML-Dokumente können durch die Textrepräsentation sowohl von Maschinen als auch von Menschen gelesen werden. Module, die XML-Dokumente interpretieren, werden XML-Parser genannt.

Die Struktur eines XML-Dokuments kann mit einem Schema beschrieben werden. Eine vollständige Beschreibung des Aufbaus eines XML-Dokuments wird als XML Schema oder XML Schema Definition (XSD) bezeichnet [20]. XSD selbst stellt ebenfalls ein XML-Dokument dar. Obwohl alle Werte in XML in Textform und damit als Strings dargestellt werden, bietet XML Schema Datentypen wie Integer, Float usw. an. Dadurch können atomare und komplexe Datentypen in XML abgebildet werden. XSD definiert damit nicht nur die Syntax, sondern auch ein abstraktes Datenmodell. Ein XML-Dokument, das der XSD entspricht, stellt eine Datensammlung dar, die mit dem entsprechenden abstrakten Datenmodell konform ist [1].

Durch den modularen Aufbau kann ein bestimmtes Schema ein Teil eines anderen Schemas sein. Es kann dadurch zu Namenskollisionen kommen, die für die Tags (Elementnamen) verwendet werden. Um dieses Problem zu lösen, werden Namespaces eingesetzt. Jeder Namespace wird durch einen eindeutigen URI identifiziert. Die Elemente werden dann einem Namespace zugeordnet. Gleiche Elementnamen sind dadurch erlaubt, solange diese unterschiedlichen Namespaces gehören. Eine Kombination aus dem Elementnamen und dem Namespace wird als Qualified Name (QName) bezeichnet. Damit ein URI nicht ständig einem Elementnamen vorangestellt wird, werden Namespace-Präfixe verwendet. Diese stellen eine kurze Schreibform für einen vollständigen URI dar und erlauben eine kompaktere Schreibweise. Weitere Informationen zu XML und XSD können den jeweiligen Spezifikationen [18] und [20] entnommen werden.

2.2.7. Simple Object Access Protocol (SOAP)

Simple Object Access Protocol (SOAP) ist ein leichtgewichtiges XML-basiertes Protokoll, das für den Datenaustausch von Web Services benutzt wird [21] [22]. SOAP ist ein Protokoll der Anwendungsschicht. SOAP selbst kann nach definierten Bindings über andere Protokolle der Anwendungsschicht wie z.B. HTTP [23] oder auch der Transportschicht wie z.B. UDP übertragen werden [24].

SOAP-Nachrichten werden mit XML codiert. Eine SOAP-Nachricht ist ein XML-Dokument, das aus einem obligatorischen SOAP Envelope, einem optionalen SOAP Header und einem obligatorischen SOAP Body besteht [25]. Die Struktur von SOAP kann wie folgt dargestellt werden:

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
    <env:Header>
    </env:Header>
    <env:Body>
    </env:Body>
</env:Envelope>
```

Der SOAP Header soll Metadaten enthalten, die für die Zustellung und Verarbeitung des Nachrichteninhalts notwendig sind. Die Payload der Nachricht wird im SOAP Body übertragen. Der Inhalt von Header und Body ist in SOAP nicht spezifiziert und wird durch ein WS-Profil bestimmt [14].

2.2.8. Devices Profile for Web Services (DPWS)

Devices Profile for Web Services (DPWS) ist ein WS-Profil für eingebettete Systeme bzw. Geräte mit eingeschränkten Ressourcen. DPWS wurde im Jahr 2004 als ein möglicher Nachfolger für Universal Plug and Play (UPnP) vorgestellt [26]. DPWS 1.0 wurde im Jahr

2006 verabschiedet [27]. DPWS 1.1 ist im Jahr 2009 als OASIS[1]-Standard erschienen [28]. DPWS besteht aus Protokollen und WS-Standards, die notwendig sind, um eine sichere Kommunikation zwischen Geräten zu ermöglichen. Darüber hinaus kann DPWS mit anderen WS-Standards kombiniert und erweitert werden [29]. Der Aufbau von DPWS ist in Abbildung 3 dargestellt. Bevor auf die einzelnen DPWS-Bestandteile eingegangen wird, soll zunächst der Kommunikationsablauf erklärt werden.

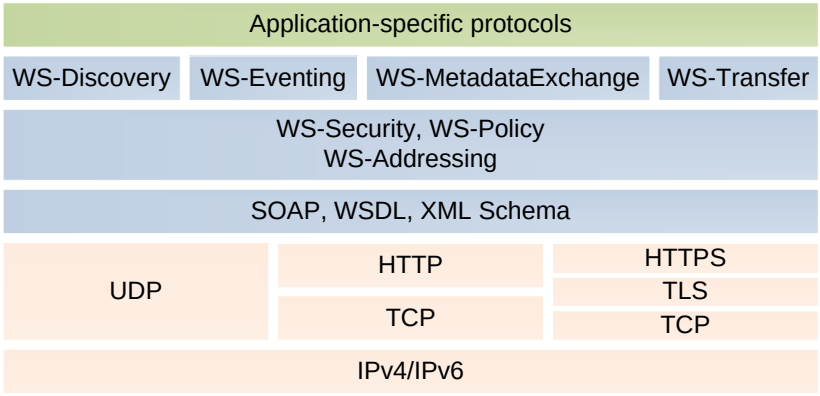


Abbildung 3: DPWS Stack [14]

Ein DPWS-Gerät (DPWS Device) wird als Hosting Service bezeichnet. Der Hosting Service enthält Informationen über das Gerät wie z.B. Gerätebezeichnung, Hersteller und andere. Darüber hinaus gibt er die vom Gerät angebotenen Services an, die als Hosted Services bezeichnet werden. Der Aufruf eines Service findet in mehreren Schritten statt.

Als erstes sucht der Client nach einem DPWS-Gerät (Discovery). Hierfür verschickt er eine *Probe*-Nachricht mit dem gewünschten Gerätenamen. Ein physikalisches Gerät kann unter Umständen mehrere logische Geräte beinhalten, z.B. Drucker und Scanner. Discovery mit einem leeren Gerätenamen führt eine Suche nach allen Geräten durch. Wenn es im Netzwerk kein zentrales Service Registry (Discovery Proxy) gibt, wird die Probe-Nachricht per Multicast mittels SOAP-over-UDP Binding verschickt. Alternativ kann der Client auf die Ankündigungen von Services durch die *Hello*-Nachrichten lauschen. DPWS-Geräte, die den Suchkriterien entsprechen, melden sich bei dem Client mit einer *ProbeMatch*-Nachricht. Diese Nachricht wird als Unicast per UDP geschickt. Die *ProbeMatch*-Nachricht enthält u.a. die sogenannte Endpoint Reference. Diese stellt Kontaktinformationen des DPWS-Gerätes dar. Falls eine *ProbeMatch*-Nachricht nur eine eindeutige ID des Gerätes – Universally Unique Identifier (UUID [30]) – und keine Informationen zum direkten Verbindungsaufbau enthält, werden diese per *Resolve*-Nachricht vom Client abgefragt. Die Resolve-Nachricht wird ebenfalls über Multicast mittels SOAP-over-UDP Binding verschickt. Das DPWS-Gerät mit der entsprechenden UUID antwortet in diesem Fall mit einer *ResolveMatch*-Nachricht, die die noch fehlenden Kontaktinformationen liefert. Diese Nachricht wird ähnlich *ProbeMatch*-

[1] Organization for the Advancement of Structured Information Standards

Nachricht als Unicast per UDP geschickt. Alle weiteren Nachrichten basieren auf einer zuverlässigen Übertragung mittels SOAP-over-HTTP Binding.

Im nächsten Schritt kann der Client die Metadaten des Gerätes anfordern (*Get Metadata*). Diese liefern, wie oben erwähnt, die Beschreibung des Gerätes (Hosting Service) sowie Endpoint References der verfügbaren (Hosted) Services. Um die Beschreibung eines Service zu erhalten, kann der Client weiterhin eine *Get Metadata*-Nachricht an einen (Hosted) Service schicken. Die Antwort darauf enthält eine Service-Beschreibung, die mit Web Services Description Language (WSDL) beschrieben ist. Diese Beschreibung enthält alle Informationen, die notwendig sind, um mit dem Service zu interagieren. Nun kann der Client einen Service-Aufruf durchführen (engl.: Service Invocation). Als Antwort darauf erhält der Client eine Invocation Response. Falls alle Service-Informationen dem Client im Voraus bekannt sind, kann der Client alle vorherigen Schritte überspringen und direkt einen Service-Aufruf schicken.

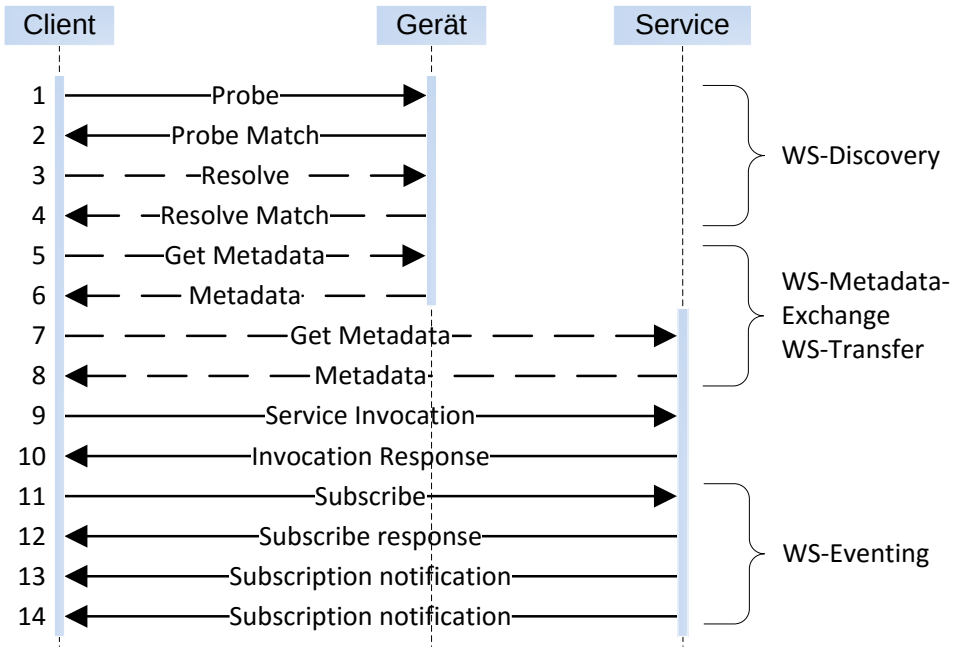


Abbildung 4: DPWS-Nachrichtenaustausch [26]

Um bestimmte Daten oder Werte aktuell zu halten, muss der Service in regelmäßigen Abständen abgefragt werden. Auch wenn keine neuen Daten vorliegen, muss der Service eine Antwort schicken. Um die Kommunikation bei solchen Anwendungsfällen effizienter zu gestalten, können Wertänderungen bei einem Service abonniert werden. Hierfür schickt der Client eine *Subscribe*-Nachricht an den Service. Der Service antwortet mit der *SubscribeResponse*-Nachricht und teilt dem Client mit, ob eine bestimmte Wertänderung erfolgreich abonniert wurde. Die Änderungen von abonnierten Werten werden mit einer *Notification*-Nachricht geliefert. Die beschriebenen Abläufe sind für einen besseren Überblick in Abbildung 4 illustriert.

Im Folgenden wird auf die einzelnen DPWS-Bestandteile eingegangen.

2.2.9. WS-Addressing

WS-Addressing spezifiziert Mechanismen, die es ermöglichen, Web Services auf eine transportprotokollunabhängige Weise zu adressieren [31]. Dadurch können SOAP-Nachrichten durch mehrere Hops und heterogene Transportprotokolle übertragen werden. WS-Addressing stellt das Konzept von Endpoint Reference vor. Für jedes Gerät existiert eine global eindeutige UUID, die ein DPWS-Gerät kennzeichnet. Diese wird im Adressenteil (*Address*) von Endpoint Reference eines DPWS-Gerätes angegeben. Die Services besitzen eine eindeutige HTTP-Transportadresse, die im Adressteil von ihren Endpoint Reference angegeben wird. Des Weiteren ermöglicht WS-Addressing einen zuverlässigen Nachrichtenaustausch mittels *Message ID*. Die *Message ID* erlaubt es, die Antworten den Anfragen zuzuordnen sowie Nachrichtenduplikate zu erkennen. Darüber hinaus kann mittels WS-Addressing die Absicht einer Nachricht (*Action*) mitgeteilt werden. *Action* kann beispielsweise der Aufruf einer Funktion eines Service sein. WS-Addressing bietet zwar noch weitere Funktionalitäten wie z.B. eine Umleitung von Antworten oder Fehlermeldungen an einen anderen Empfänger, jedoch wurden diese in DPWS nicht übernommen.

2.2.10. WS-Policy

WS-Policy wird benutzt, um dynamische Bestandteile einer Kommunikation auszuhandeln [32]. WS-Policy stellt ein Framework zur Verfügung, mit deren Hilfe Einschränkungen, Anforderungen, Fähigkeiten und Charakteristiken der Kommunikationspartner zum Ausdruck gebracht werden können [14]. Eine Policy ist eine Auswahl an Alternativen. Der Kommunikationspartner kann die für ihn am besten geeignete Policy-Alternative aus einem Policy-Satz auswählen. Eine Policy-Alternative besteht aus einer Reihe von „Assertions“ (Behauptungen, Anforderungen), die vom Kommunikationspartner erfüllt werden müssen. Die Policy Assertions können auch optional sein. WS-Policy spezifiziert jedoch nicht, wie die Policy auf einen bestimmten WS-Standard angewandt werden muss. Diese Aufgabe wird von WS-PolicyAttachment übernommen [33].

2.2.11. WS-Security

Die Verschlüsselung, Authentifizierung und Integrität können bei DPWS bereits durch Transport Layer Security (TLS) bzw. Hypertext Transfer Protocol Secure (HTTPS) übernommen werden. WS-Security ermöglicht ebenfalls die Verschlüsselung, die Authentifizierung und die Integrität auf WS-Ebene [34]. WS-Security ist ein Sicherheits-Framework, das einen großen zusätzlichen Overhead mitbringt. Da HTTPS bereits wichtige Sicherheitsmechanismen bietet, werden nur einige Aspekte von WS-Security verwendet. DPWS hat die Mechanismen von WS-Security nur für Discovery-Nachrichten übernommen,

da diese HTTPS nicht nutzen können. Hierbei können nur Signaturen für die vom DPWS-Gerät geschickten Nachrichten während des Discovery-Prozesses verwendet werden, um die Echtheit des Gerätes zu verifizieren. Da DPWS auf Geräte mit eingeschränkten Ressourcen ausgelegt ist, sind die Sicherheits-Features optional. Dennoch kann DPWS bei Bedarf einen großen Sicherheitsumfang bieten.

2.2.12. WS-MetadataExchange

Web Services nutzen Metadaten, um dem Client notwendige Informationen mitzuteilen, die für die Interaktion mit dem Service notwendig sind. WS-MetadataExchange definiert, wie die Metadaten als eine Ressource dargestellt und mittels WS-Transfer angefordert werden können. Des Weiteren spezifiziert WS-MetadataExchange wie die Metadaten in eine Endpoint Reference eingebettet werden können und wie eine Anfrage und eine Antwort für die Übertragung der Metadaten aussehen soll [35].

WS-MetadataExchange definiert zwei mögliche Anfragearten für die Anforderung der Metadaten. Wenn die Endpoint Reference für die bestimmten Metadaten bekannt ist, z.B. durch das Discovery, kann der Client die Metadaten mittels WS-Transfer *Get*-Operation anfordern (vgl. Abbildung 4). Wenn die Endpoint Reference für die bestimmten Metadaten unbekannt ist, kann der Client eine *GetMetadata*-Anfrage an den WS Endpoint schicken, um alle Metadaten anzufordern.

Die Gerätebeschreibung enthält wichtige Informationen über das Device sowie die angebotenen Services. Die Metadaten werden in drei Abschnitte unterteilt: *ThisModel*, *ThisDevice* und *Relationship*. *ThisModel* stellt Metadaten über den Gerätetyp zur Verfügung wie z.B. Gerätehersteller, Modelname usw. *ThisDevice* enthält Informationen über ein konkretes Gerät wie z.B. Firmware-Version oder Seriennummer. *Relationship* gibt Auskunft über die Endpoint References der Services und deren Interface-Typen.

2.2.13. WS-Transfer

WS-Transfer wird für die Interaktion mit Ressourcen unter Benutzung von Web Services eingesetzt [36]. Die Interaktion erfolgt mittels Operationen *Get*, *Put*, *Delete* und *Create*, die ähnlich dem REST-Ansatz für die Standardoperationen wie Ressource lesen, aktualisieren, löschen und erstellen vorgesehen sind. DPWS verwenden WS-Transfer für den Zugriff auf Metadaten des Gerätes oder des Service, die in Form von Ressourcen dargestellt werden. Hierbei muss jedoch nur der *Get*-Operator unterstützt werden, mit dessen Hilfe die Metadaten angefordert werden können.

2.2.14. Web Services Description Language (WSDL)

WSDL ist ein XML-Dokument, das den Aufbau eines Service als eine Menge von *Endpoints* und deren Interaktionen (Operationen) mittels Nachrichten (Messages) beschreibt [37]. Operationen und Nachrichten werden zunächst abstrakt dargestellt. Anschließend werden sie einem konkreten Netzwerkprotokoll und Nachrichtenformat zugewiesen und bilden damit einen Endpoint. Die konkreten Endpoints werden zu einem abstrakten Endpoint zusammengefasst, der einen Service repräsentiert. WSDL ist erweiterbar, um eine Beschreibung von beliebigen Endpoints und deren Nachrichten unabhängig vom Netzwerkprotokoll und Nachrichtenformat zu ermöglichen.

Die Services, die durch eine Menge von Endpoints dargestellt werden, werden in WSDL als *Ports* bezeichnet. Eine abstrakte Darstellung von Endpoints und Nachrichten ermöglicht eine mehrfache Nutzung dieser Definitionen. Nachrichten sind eine abstrakte Darstellung der Daten, die übertragen werden, und *Port Types* sind eine abstrakte Menge von Operationen. Eine konkrete Spezifikation eines Protokolls bzw. eines Datenformats für einen bestimmten Port Type bildet ein Binding. Ein Port entsteht durch die Zuweisung einer Netzwerkadresse zu einem Binding. Eine Menge von Ports bildet den Service. Folglich benutzt ein WSDL-Dokument folgende Elemente zur Definition eines Services, wie in Tabelle 2 gezeigt ist.

Element	**Beschreibung**
Types	Ein Container für die Definition von Datentypen
Message	Eine abstrakte Darstellung der zu übertragenden Daten
Operation	Eine abstrakte Darstellung einer Aktion, die vom Gerät unterstützt wird
Port Type	Ein abstrakter Satz an Operationen, die von einem oder mehreren Endpoints unterstützt werden
Binding	Eine konkrete Protokoll- und Datenformatspezifikation für einen bestimmten Port Type
Port	Ein Endpoint, definiert durch eine Kombination aus einem Binding und einer Netzwerkadresse
Service	Eine Sammlung von Endpoints

Tabelle 2: Elemente eines WSDL-Dokuments

Ein WSDL-Dokument kann in einen konkreten und einen abstrakten Teil aufgeteilt werden. Der Aufbau eines WSDL-Dokuments ist in Abbildung 5 dargestellt.

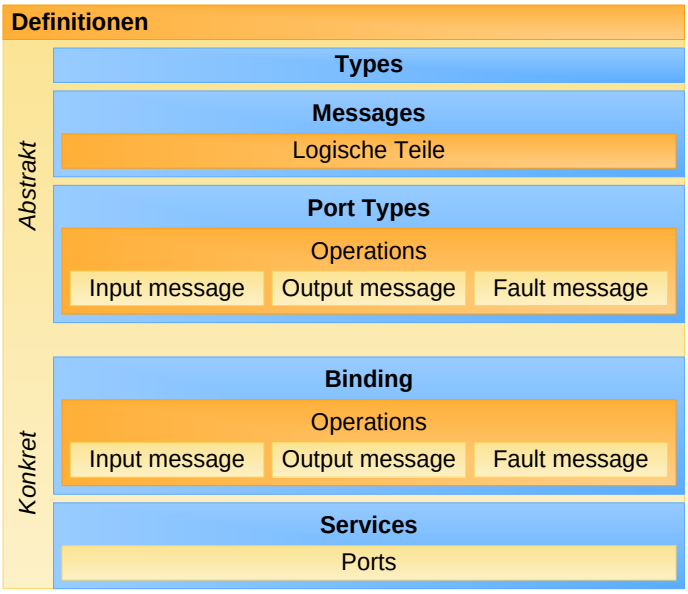


Abbildung 5: Aufbau eines WSDL-Dokuments [16]

Abstrakter Teil: Der abstrakte Teil eines WSDL-Dokuments ermöglicht die Wiederverwendbarkeit durch die Beschreibung der Services unabhängig von technischen Details wie Transportprotokoll oder Nachrichtenformat. Zuerst müssen die Datentypen, die für den Nachrichtenaustausch notwendig sind, definiert werden. Um die höchste Interoperabilität zu gewährleisten, nutzt WSDL XML-Schema für die Beschreibung der Datentypen. Messages (Nachrichten) sind abstrakte Definitionen der Daten, die von und zu WS übermittelt werden. Die Messages bestehen aus einem oder mehreren logischen Teilen. Die Port Types können als abstrakte Interfaces zu beschriebenen Services angesehen werden. Ein Port Type ist ein Satz an abstrakten Operationen. Eine Operation ist ein Satz an Messages, die einem bestimmten Datenaustauschmuster folgen. WSDL unterstützt die vier folgenden Datenaustauschmuster:

- One-way: Der Client schickt eine Nachricht an den Service
- Request-Response: Der Client schickt eine Nachricht an den Service und erwartet eine Antwort
- Solicit-Response: Der Service schickt eine Nachricht an den Client und erwartet eine Antwort
- Notification: Der Service schickt eine Nachricht an den Client

Das Datenaustauschmuster wird in der WSDL-Beschreibung implizit durch die Erscheinungsreihenfolge der Input- und Output-Messages angegeben. Request-Response und Solicit-Response-Operationen können zusätzlich eine Fault-Message für den Fall definieren, dass die Antwort nicht geschickt werden kann.

Konkreter Teil: Der konkrete Teil enthält die technischen Details zu WS. Ein Binding weist einen Port Type, seine Operationen und Messages einem bestimmten Protokoll und Datenformat zu. Mehrere Bindings können dabei mit einem Port Type erstellt werden. Alle definierten Port Types müssen sich im konkreten Teil widerspiegeln. Die konkreten Fault-Messages sind nicht definiert, da diese durch das Protokoll gegeben sind. Der Service stellt eine Sammlung von Ports dar. Ports sind die Netzwerkadressen für die Service Endpoints [14].

2.2.15. WS-Eventing

WS-Eventing stellt einen Publish/Subscribe-Mechanismus bereit, mit dessen Hilfe der Client automatisch über die Wertänderungen vom Service Provider benachrichtigt werden kann [38]. WS-Eventing definiert vier Rollen: den Subscriber, die Event-Quelle, die Event-Senke und den Subscription Manager. Der *Subscriber* kann bei einem Web Service Provider (*Event-Quelle*) ein Event (Wertänderung) abonnieren. Hierfür schickt dieser eine Subscription-Nachricht an die Event-Quelle und teilt ihr eine *Event-Senke* mit. Wenn das Event auftritt, benachrichtigt die Event-Quelle die Event-Senke über die Änderungen mittels einer Notification-Nachricht (vgl. Abbildung 4). In verteilten Publish/Subscribe-Systemen kann die Event-Quelle die Managementaufgaben an einen *Subscription Manager* übergeben. Die Subscriptions können zeitlich beschränkt sein, sodass der Subscriber diese bei Bedarf mit der Renew-Nachricht erneuern kann. Der Subscriber kann sich darüber hinaus über die GetStatus-Nachricht nach dem Status der Subscription erkundigen. Falls die Events nicht mehr an die Event-Senke geschickt werden sollen, kann der Subscriber die Subscription vorzeitig mit einer Unsubscribe-Nachricht beenden.

2.2.16. WS-Discovery

WS-Discovery definiert ein Protokoll für die Suche nach den Services [39]. Der Client kann dabei nach einem oder mehreren Service Providern suchen. Die Suche erfolgt nach Service-Typen. Als Services werden bei WS-Discovery die Hosting Services (DPWS-Geräte) verstanden. Das Protokoll beschreibt zwei Operationsmodi, den Ad-hoc-Mode und den Managed Mode. Im Ad-hoc-Mode sendet der Client eine *Probe*-Nachricht an eine Multicast-Gruppe. Services, die die Suchanfrage erfüllen, senden eine Antwort (*ProbeMatch*-Nachricht) direkt an den Client zurück. Wenn die *ProbeMatch*-Nachricht keine Transport-Adresse enthält, schickt der Client eine *Resolve*-Nachricht an die Multicast-Gruppe mit der Endpoint Reference des gesuchten Service. Der Service mit der entsprechenden Endpoint Reference beantwortet diese Anfrage mit einer *ResolveMatch*-Nachricht und teilt dem Client seine Transportadresse mit (vgl. Abbildung 4). Der Service kann auch über mehrere Transportadressen verfügen, z.B. durch mehrere IP-Adressen.

Damit der Client das Netzwerk nicht regelmäßig nach den Services durchsucht, melden sich diese im Netzwerk automatisch mittels einer *Hello*-Nachricht an. Die *Hello*-Nachricht wird ebenfalls an die Multicast-Gruppe geschickt. Auf diese Weise kann der Client neue Services im Netzwerk ohne wiederholtes Suchen detektieren. Um einige Skalierbarkeitsaspekte zu verbessern sowie die Suche über die Grenzen eines Netzwerks zu ermöglichen bietet WS-Discovery den Managed Mode an. In diesem Modus senden die Services ihre Ankündigungen (*Hello*-Nachrichten) per Unicast an den Discovery Proxy und die Clients senden ihre Suchanfragen *Probe* und *Resolve* ebenfalls per Unicast an den Discovery Proxy. Discovery Proxy kündigt sich im Netzwerk mittels einer Multicast-Hello-Nachricht an. Die Adresse des Discovery Proxy kann auch im Client vorkonfiguriert werden. Der Discovery Proxy stellt allerdings einen Single Point of Failure (SPoF) dar. In einem Netzwerk, das auf Managed Mode ausgelegt ist, führt der Ausfall des Discovery Proxy zu der Nichtverfügbarkeit des Service Discovery.

Die Services, die das Netzwerk verlassen, melden sich mit einer *Bye*-Nachricht ab. Hierdurch werden Clients informiert, dass bestimmte Service Provider nicht mehr erreichbar sind.

2.3. Gebäudeautomation

Gebäudeautomation ist ein wachsendes Gebiet, das sich mit der Vernetzung der Geräte im Gebäude beschäftigt. Der Bedarf an Vernetzung der Geräte im Gebäude entstand aus dem Wunsch nach mehr Komfort und Energieeffizienz. Durch die Vernetzung lassen sich die Funktionen der Gebäude sowie einzelne Geräte aus der Entfernung steuern. Ein Gebäudeautomationssystem kann auch vollautomatisch auf bestimmte Ereignisse reagieren und Funktionen auslösen. Das einfachste Beispiel ist die automatische Beleuchtungsanpassung. Befindet sich ein Mensch im Raum, wird seine Präsenz durch spezielle Sensoren wie Präsenzmelder erfasst. Dann kann die Beleuchtung an die voreingestellte Helligkeit mithilfe von zusätzlichen Helligkeitssensoren angepasst werden. Ist der Raum dagegen leer, kann die Beleuchtung ausgeschaltet werden. Diese Funktion stellt gleichzeitig sowohl eine Komfort- als auch eine Energieeinsparfunktion dar.

Um Geräte zu vernetzen, ist ein geeignetes Kommunikationsprotokoll notwendig. Gegenwärtig ist Gebäudeautomation vor allem durch eine Vielzahl proprietärer Protokolle charakterisiert. Diese Protokolle sind in der Regel nicht interoperabel und erschweren damit die Integration verschiedener Hersteller in eine Installation. In diesem Kapitel wird auf ausgewählte Protokolle der Gebäudeautomation eingegangen, die derzeit am Markt vorzufinden sind.

Zur Einordnung der Technologien soll ein 3-Schichten-Modell für Automatisierungstechnik herangezogen werden [40]. Dieses ist in Abbildung 6 dargestellt.

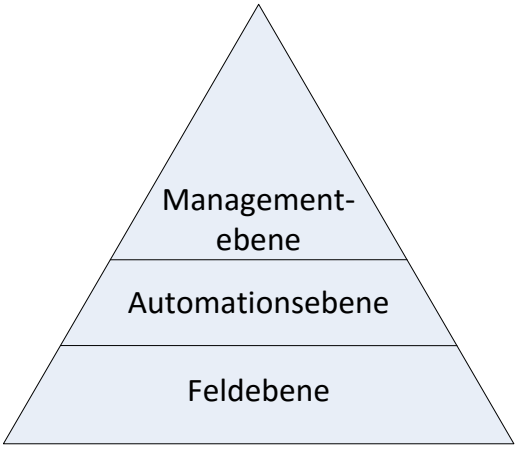


Abbildung 6: Ebenen der Gebäudeautomation

Die Feldebene ist die unterste Ebene in der Automatisierungshierarchie. In der Feldebene erfolgt eine direkte Kommunikation mit Sensoren und Aktoren. Die Kommunikation findet in der Regel über sogenannte Feldbusse statt. Die Automationsebene übernimmt die Steuerung und Regelung von Sensoren und Aktoren sowie das Sammeln der Messwerte. In der Managementebene erfolgen die Überwachung der Automatisierungsanlage sowie die Visualisierung der Daten. Im Folgenden werden ausgewählte Protokolle der klassischen Gebäudeautomation untersucht.

2.3.1. Klassische Gebäudeautomation

2.3.2. BACnet

BACnet steht für Building Automation and Control Networks. Das Protokoll wurde von der American Society of Heating, Refrigeration, and Air-Conditioning Engineers (ASHRAE) entwickelt und im Jahr 1995 als American National Standards Institute (ANSI)/ASHRAE-Norm 135 standardisiert [41]. Seit 2003 ist BACnet ebenfalls bei dem Deutschen Institut für Normung (DIN) als eine Europäische Norm (EN) und eine ISO Norm 16484-5 standardisiert [40]. Ziel von BACnet war, eine Interoperabilität zwischen verschiedenen Herstellern zu erreichen, da zu diesem Zeitpunkt jeder Hersteller ein eigenes proprietäres Protokoll eingesetzt hat.

BACnet kann über verschiedene Medien kommunizieren. RS-232 und RS-485 ermöglichen die serielle Kommunikation. Die Benutzung derselben unteren Kommunikationsschichten wie bei Local Operation Network (LonTalk) ist ebenfalls möglich. Weiterhin stellt Attached Resource Computer Network (ARCNET) eine echtzeitfähige Bustechnologie mit Token Passing dar und war als Konkurrenz zu Ethernet (IEEE 802.3) konzipiert. Mit einer raschen Entwicklung des Ethernets hat diese jedoch an Bedeutung verloren und wird nur vereinzelt in der Industrie eingesetzt [42]. Im Jahr 1999 wurde eine Erweiterung des Standards als BACnet/IP verabschiedet [43]. Dadurch wurde BACnet mit der Kommunikationsmöglichkeit

über das IP-Protokoll ergänzt. Im Jahr 2011 wurden für BACnet verfügbare Medien mit der Funktechnologie ZigBee ergänzt [44].

Im Rahmen dieser Arbeit wird nur auf die Anwendungsschicht von BACnet eingegangen. Informationen über andere Schichten können der Norm DIN EN ISO 16484-5 entnommen werden [45]. Die Anwendungsschicht definiert ein Datenmodell für die im Gerät verfügbaren Informationen. Diese Informationen werden als Objekte repräsentiert. Die Properties (Eigenschaften) eines Objektes beschreiben bestimmte Aspekte der Software, der Hardware oder des Betriebs eines Gerätes. Alle Objekte eines Gerätes stellen das sichtbare Kommunikationsinterface eines BACnet-Gerätes. Der Zugriff auf die Objekte erfolgt mittels Funktionen, die Services genannt werden.

BACnet definiert 54 Objekttypen [46]. Darüber hinaus können die Hersteller eigene proprietäre Objekttypen definieren. Diese können jedoch in der Regel von anderen Herstellern nicht interpretiert werden. Die 18 Basistypen sind nachfolgend aufgelistet [47]:

- Analog Input
- Analog Output
- Analog Value
- Binary Input
- Binary Output
- Binary Value
- Multi-state Input
- Multi-state Output
- File
- Loop
- Device
- Event Enrollment
- Program
- Schedule
- Calendar
- Command
- Group
- Notification Class

Die mit *Binary* bezeichneten Objekttypen dienen der Steuerung oder der Zustandserfassung einer Komponente des Gerätes mit nur zwei möglichen Zuständen wie z.B. ein Schalter oder ein Relay. Die *Multi-state*-Objekttypen ermöglichen dagegen mehrere Zustände. Alle anderen Werte (Integer, Float) und Steuerungskomponenten, z.B. eines Sensors, können grundsätzlich mit den als *Analog* bezeichneten Objekten erfasst werden. Die *Calendar*- und *Schedule*-Objekte ermöglichen das Einstellen von zeitlichen Parametern. *Command*-Objekt ermöglicht die Ausführung eines Befehls oder einer Prozedur, wobei mehrere Werte in mehreren Objekten auf mehreren Geräten gleichzeitig oder nacheinander gesetzt werden können. Mit dem Objekt *Event Enrollment* können zusätzliche Events wie z.B. Alarmsignale definiert werden. Objekte wie Analog Input, Analog Output usw. verfügen bereits über die Eventing-Funktionalitäten und benötigen damit das *Event Enrollment-Objekt* nicht. Das *Group*-Objekt ermöglicht den Zugriff auf mehrere Objekte während einer Anfrage. Das Objekt *Loop* wird zur Abbildung der Regelalgorithmen verwendet. *Notification Class* steht in Zusammenhang mit *Event Enrollment* und ermöglicht das Management von Event-Subscriptions. Das *Program*-Objekt ermöglicht das Laden, die Ausführung, und das Beenden eines Programms im Gerät. Mit dem *File*-Objekt können Dateien gelesen und geschrieben werden. Eine besondere Rolle hat das *Device*-Objekt. Es enthält Metainformationen über das Gerät wie z.B.

Hersteller, Firmware sowie eine Liste von Objekten und Services, die vom Gerät unterstützt werden. Das Device-Objekt steht damit in Zusammenhang mit dem Discovery-Mechanismus der BACnet-Geräte. Jedes Objekt kann in einem Gerät beliebig oft vorkommen, außer dem Device-Objekt.

Die Objekte bestehen aus Properties. Je nach Objekt können sich diese unterscheiden. Einige Properties sind für einen bestimmten Objekttyp obligatorisch, andere optional. BACnet ermöglicht weiterhin die proprietären Properties zu den Standardobjekten hinzuzufügen. Diese können jedoch in der Regel von anderen Herstellern nicht interpretiert werden. Nachfolgend sind beispielhaft die obligatorischen Properties eines Analog Input-Objektes dargestellt. Die Namen der Properties sollen hier selbsterklärend sein:

- Object_Identifier
- Object_Name
- Object_Type
- Present_Value
- Status_Flags
- Event_State
- Out_Of_Service
- Units

Der Zugriff auf Objekte bzw. Properties erfolgt, wie oben erwähnt, über Funktionen, die in BACnet-Kontext Services genannt werden. In Tabelle 3 sind einige ausgewählte Services aufgelistet. Die Services lassen sich in Service-Typen je nach Verwendung einteilen: Objektzugriff-Services, Alarm- und Ereignis-Services, Dateizugriff-Services, Device- und Netzwerkmanagement-Services sowie Virtual-Terminal-Services.

Service	**Erklärung**
ReadProperty/WriteProperty	Eine Property lesen/schreiben
ReadPropertyConditional	Auslesen mehrerer Properties, die bestimmte Bedingungen erfüllen
ReadPropertyMultiple/WritePropertyMultiple	Mehrere Properties lesen/schreiben
Who-Has	Discovery nach einem bestimmten Objekt
I-Have	Antwort auf Who-Has
Who-Is	Discovery nach einem bestimmten Gerät
I-Am	Antwort auf Who-Is
SubscribeCOV	Wertänderungen abonnieren (Change of Value - COV)
ConfirmedCOVNotification/ UnconfirmedCOVNotification	Bestätigte/unbestätigte Benachrichtigung über eine abonnierte Wertänderung

Tabelle 3: Ausgewählte BACnet-Services

Wie aus Tabelle 3 ersichtlich, bietet BACnet nicht nur einen Request-Response- sondern auch einen Notification-Mechanismus an. Der Client kann dabei automatisch über die

Wertänderungen benachrichtigt werden, wenn er diese abonniert hat. Darüber hinaus bietet BACnet auch Discovery-Mechanismen an, wobei die Suche sowohl nach Geräten (Who-Is) als auch nach bestimmten Objekten (Who-Has) erfolgen kann. Die beiden Discovery-Nachrichten werden mittels Broadcast geschickt und können bei Bedarf von jedem Gerät beantwortet werden. Ein BACnet-Gerät muss die Discovery-Funktionalität nicht anbieten. Die BACnet-Services können als Nachrichten verstanden werden. Diese werden sowohl vom Service Provider als auch vom Client angeboten.

Für die Interoperabilität zwischen dem Service Provider und dem Client werden BACnet Interoperability Building Blocks (BIBB) eingesetzt. Diese beschreiben, welche Funktionalitäten der Service Provider und der Client unterstützen müssen, um eine bestimmte Operation ausführen zu können. Korrespondierende BIBBs bei dem Client und dem Service Provider sind Voraussetzung für eine funktionierende Kommunikation. Die BIBBs werden in 5 Interoperabilitätsbereiche (IOB) aufgeteilt [46]:

- Gemeinsame Datennutzung (Data Sharing, DS)
- Alarm- und Ereignis-Verarbeitung (Alarm and Event Management, AE)
- Zeitplan (Scheduling, SCHED)
- Trendaufzeichnung (Trending, T)
- Device- und Netzwerkmanagement (Device and Netzwork Management, DM)

Der BIBB-Name gibt an, zu welchem IOB er gehört, um welche Funktion es sich handelt und ob es sich um die Sicht des Clients (A) oder des Service Providers (B) handelt. Möchte beispielsweise ein Client die Operation ReadProperty an einem Sensor ausführen, muss dieser den BIBB DS-RP-A und der Sensor DS-RP-B entsprechend unterstützen. Eine vollständige Liste von BIBBs kann der BACnet-Spezifikation entnommen werden. Um die BACnet-Geräte nach ihrer Leistungsfähigkeit bewerten zu können, wurden Konformitätsklassen eingeführt. Jede Klasse beschreibt die jeweiligen Mindestanforderungen an ein Gerät. In Tabelle 4 sind die definierten Klassen aufgelistet. Die höheren Konformitätsklassen schließen die Eigenschaften der niederen Klassen bereits mit ein.

Um das Gerät einer Klasse zuzuordnen, werden von Herstellern Protocol Implementation Conformance Statement (PICS) verfasst. PICS enthält Informationen darüber, welche BIBBs unterstützt werden, welche Services als Client und Service Provider verfügbar sind, welche Standard-Objekttypen und proprietäre Objekttypen eingesetzt werden. Der oben beschriebene Aufbau der BACnet-Anwendungsschicht ist in Abbildung 7 dargestellt.

BACnet kann grundsätzlich auf allen Ebenen der Gebäudeautomation eingesetzt werden (vgl. Abbildung 6). Es findet in der Praxis jedoch vor allem Verwendung in der Automations- und Managementebene [40].

Kl.	Bezeichnung	Erklärung
1	Smart Sensor	Ein Gerät kann lediglich Objekt-Properties melden (ReadProperty)
2	Smart Actuator	Ein Gerät kann eine oder mehrere Objekt-Properties schreiben
3	Application Specific Controller	Ein Gerät kann mehrere Objekt-Properties lesen und schreiben, es kann sich selbstständig im Netzwerk anmelden, identifizieren und Auskunft über seine Objekte geben.
4	Advanced Application Controller	Das Gerät beinhaltet Client-Services, es kann ReadProperty und WriteProperty initiieren
5	Building Controller	Eine programmierbare Station, die für verschiedene Steuerungs- und Regelungsaufgaben eingesetzt werden kann
6	Operator Workstation	Benutzerschnittstelle zum BACnet-System

Tabelle 4: BACnet-Konformitätsklassen

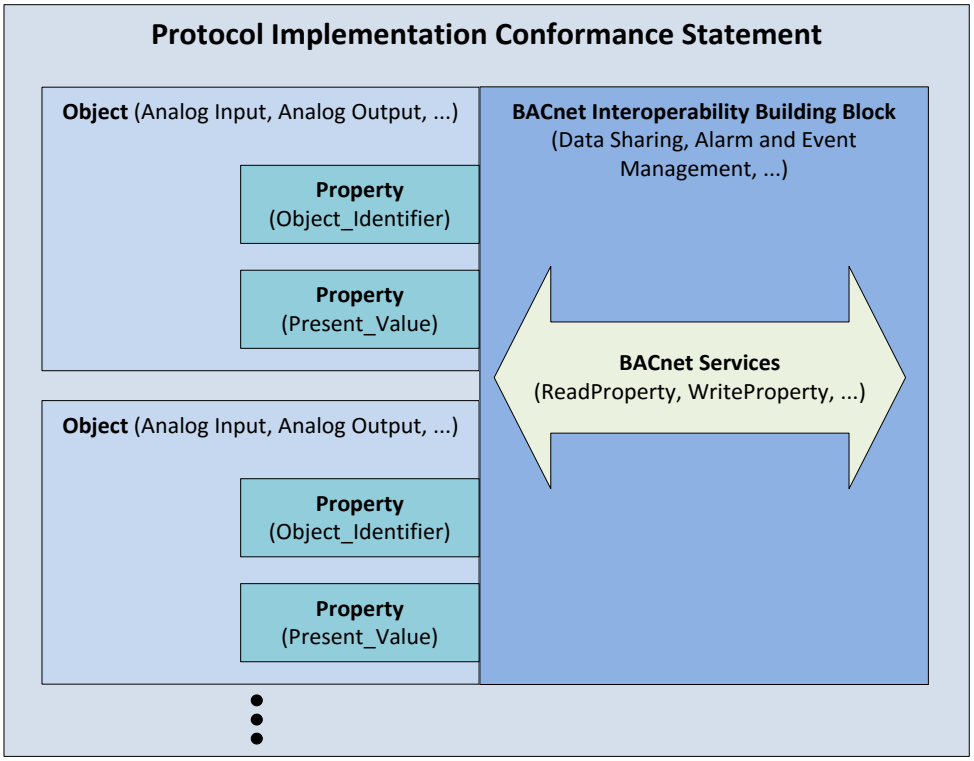


Abbildung 7: Struktur der BACnet-Anwendungsschicht

BACnet/WS

2006 erweiterte die ASHRAE den BACnet-Standard im Addendum 135-2004c um Annex N [48], welcher die Spezifizierung des *BACnet Web Services Interface* (BACnet/WS) beinhaltet. Das Ziel dieser Erweiterung war die Bereitstellung einer Reihe von Web-Diensten, die eine

protokollneutrale Schnittstelle zwischen Gebäudeautomationssystemen und dem Internet darstellen können. Dafür sind Funktionen zum Lesen und Schreiben der gemeinsamen Elemente aller Gebäudeautomationssysteme wie Werte, Zustände oder Zeiten definiert sowie Methoden zum Identifizieren der Datenquelle. Annex N wurde für einen einfachen Datenaustausch entworfen. Für anspruchsvollere Systeme ist dies nicht ausreichend. Deshalb entwickelt die ASHRAE mit Addendum 135-2012am eine neue Erweiterung [49]. Dieses Addendum befindet sich derzeit im Status der öffentlichen Begutachtung. Annex N verwendet für die WS das *Basic Profile 1.0* der *Web Services Interoperability Organization* (WS.I). Das Basic Profil benutzt SOAP über HTTP zum Informationsaustausch. Im neuen Addendum 135-2012am wird das *Representational-State-Transfer*-Protokoll (REST) anstelle von SOAP eingesetzt.

Wie Gebäudedaten bestimmt, übertragen und gespeichert werden, beschreibt das Datenmodell von BACnet/WS. Da es sich um ein abstraktes Modell handelt, lassen sich Informationen von allen Quellen damit repräsentieren. Der neue Standard (135-2012am) unterscheidet dabei zwischen Daten (*Data*) und Metadaten (*Metadata*). Daten enthalten immer einen Wert (*Value*). Metadaten enthalten Charakteristiken, Beschränkungen oder Beziehungen, mit denen Daten beschrieben werden. Die Instanzen beider Formen heißen *Data-Item* und *Metadata-Item*. Untergruppen der Data-Items sind *Points*, *Objects*, *Properties* und *Interfaces* (vgl. Abbildung 8). Konkrete Metadata-Items sind zum Beispiel *name*, *type* oder *description*.

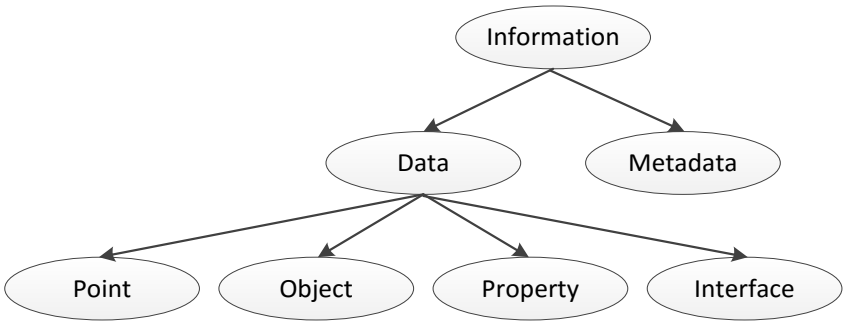


Abbildung 8: BACnet/WS Datenmodell

Points sind für Protokolle vorgesehen, die dem Konzept nachgehen, Gebäudedaten als Punkte darzustellen. Sie enthalten einen konkreten Wert. Diese Punkte sind in der Regel in verschiedenen Protokollen unterschiedlich strukturiert. Damit ein Kommunikationspartner eines anderen Protokolls die Daten dennoch interpretieren kann, sind in BACnet/WS sogenannte *normalized Points* festgelegt. Für diese Points sind obligatorische Metadaten festgelegt. Objects können von objektorientierten Protokollen wie BACnet genutzt werden. Ein Object-Item entspricht einem BACnet-Objekt. Die Kindelemente der Object-Items sind standardmäßig Property Items, die die Objekteigenschaften von BACnet repräsentieren. Die Abbildung eines BACnet-Objekts mit dessen Eigenschaften stellt Listing 1 dar. Interfaces

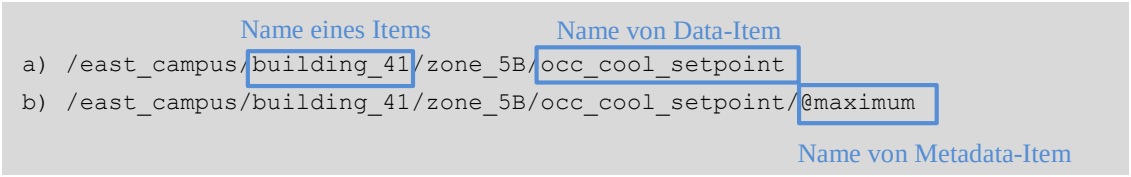
sind ein logisches Konzept, um Daten beliebiger Komplexität zu modellieren. Sie können Properties, Points oder andere Interfaces enthalten.

```
<Object name="A Great Object">
   <Enumerated name="object-type" value="analog-input"/>
   <ObjectIdentifier name="ObjectIdentifier" value="analog-input,2"/>
   <Real name="present-value" value="75.5"/>
   ...
</Object>
```

Listing 1: Ein BACnet-Objekt in BACnet/WS [49]

Alle Data-Items lassen sich nach unterschiedlichen Kriterien in einer hierarchischen Struktur anordnen. Nach welchen Kriterien die Hierarchie gebildet wird, ist im Standard nicht spezifiziert. Möglich wäre zum Beispiel eine Hierarchie nach dem Standort (Gebäude, Etage, Raum) oder eine Einteilung nach den Gewerken (Klima, Licht, Zutrittskontrolle). Eine Hierarchie beginnt mit einem Wurzelelement (*Root*), welches Kindelemente besitzt. Alle weiteren Elemente in der Struktur haben ein Elternelement und können Kindelemente besitzen. Alle Data-Items haben einen Namen, der in der Regel installationsabhängig ist, und sind von einem standardisierten oder proprietären Typ. Zu den standardisierten Typen gehören einzelne primitive Typen (z. B. Boolean, Integer, String), zusammenfassende Typen (z. B. Object, Interface, Struct, Array, List, Collection) und BACnet-spezifische Typen (z. B. ObjectIdentifier).

Für Adressierung von Data-Items oder Metadata-Items werden URIs verwendet. Die Teile der URI entsprechen den Namen der Items (vgl. Listing 2). Namen von Metadata-Items unterscheiden sich durch ein vorangestelltes ‚@' von den Namen der Data-Items.



Listing 2: a) URI eines Data-Items, b) URI eines Metadata-Items [49]

Services nutzen den URI für den Zugriff auf ein Item. Über die REST-Operatoren GET, PUT oder POST und den entsprechenden URI erfolgt der Aufruf einer Information. Mit GET lässt sich der Wert eines Data- oder Metadata-Items lesen. Die Antwort erfolgt standardmäßig im XML-Format. Mit PUT kann ein Wert verändert werden. Der Wert wird ebenfalls im XML-Format übertragen. Beide Vorgänge sind in Abbildung 9 dargestellt. POST wird in BACnet/WS für die Meldungen bei Abonnements verwendet. Ein Server verwendet diesen Operator, um zum Beispiel den Abonnenten eine Wertänderung mitzuteilen [49].

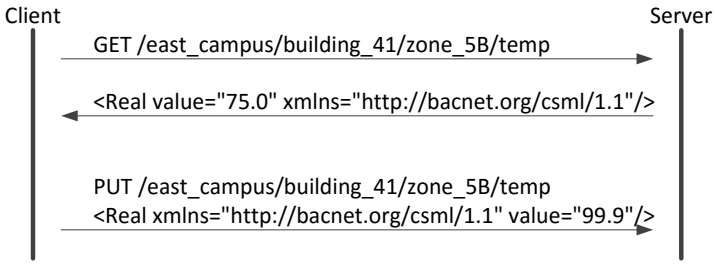


Abbildung 9: GET und PUT in BACnet/WS [49]

2.3.3. KNX

KNX ist ein Feldbus zur Vernetzung von Geräten in der Gebäudeautomation. KNX entstand als ein Zusammenschluss des Europäischen Installationsbusses (EIB), des BatiBUS und der European Home Systems (EHS). Die erste Spezifikation von KNX wurde von der KNX Association im Jahr 2002 veröffentlicht [50]. KNX ist dabei zum EIB rückwärtskompatibel. 2003 wurde KNX zum europäischen Standard EN 50090 [51], 2006 zum internationalen Standard ISO/IEC 14543-3.

KNX ist mehr als ein reines Kommunikationsprotokoll. Neben der Definition, wie die Daten transportiert werden, beschreibt der Standard ebenfalls, wie ein KNX-System verwaltet und wie die Geräte von verschiedenen Herstellern implementiert werden müssen, um ein ordnungsgemäßes Zusammenwirken zu ermöglichen [52].

Der KNX-Standard sieht für die Kommunikation der Geräte folgende Medien vor: KNX Twisted Pair (KNX TP), KNX Powerline (KNX PL), KNX Radio Frequency (KNX RF) sowie Ethernet (KNX IP) [53]. Die Planung einer KNX-Installation erfolgt in vier Schritten. Zuerst muss ein Übertragungsmedium gewählt werden. Danach müssen die erforderlichen Geräte wie Sensoren und Aktoren ausgewählt werden. Im nächsten Schritt sollen den Geräten eindeutige physikalische Adressen zugewiesen werden, wodurch auch die Topologie eindeutig festgelegt wird. Die KNX-Topologie besteht aus Linien und Bereichen. Die Teilnehmer werden bestimmten Linien zugeordnet. Mehrere Linien werden an eine Hauptlinie angeschlossen und stellen einen Bereich dar. Die Bereiche können weiterhin über eine Bereichslinie miteinander verbunden werden. Die Geräteadressen werden in Form von *Bereich.Linie.Teilnehmer* angegeben. Neben den physikalischen Adressen erhalten die Geräte auch logische bzw. Gruppenadressen. Abschließend müssen die Geräte mit der speziellen Software „Engineering Tool Software (ETS)“ programmiert werden.

Die Kommunikation in KNX basiert auf dem Publish/Subscribe-Modell [54]. Die Sender benutzen dabei Gruppen-Adressen, um die Nachrichten mittels Multicast zuzustellen. Ein

Gerät kann dabei Mitglied in mehreren Gruppen sein. Die Unicast-Nachrichten werden nur für Konfigurations- und Management-Zwecke genutzt.

Da KNX unterschiedliche Medien unterstützt, lässt sich der Protokoll-Stack in medienabhängige und medienunabhängige Schichten aufteilen. Im Rahmen dieser Arbeit wird nur die Anwendungsschicht näher erläutert. Informationen über andere Schichten können dem KNX-Standard EN 50090 entnommen werden.

Zum Datenaustausch verwendet KNX Kommunikationsobjekte (engl. Datapoints), die in Funktionsblöcke zusammengefasst werden. Diese sind das zentrale Konzept im KNX-Datenmodell. Die KNX-Kommunikationsobjekte können sich auf den Zustand eines Aktors, auf den Wert eines Sensors oder auch einen Parameter beziehen, der das Verhalten eines Gerätes steuert [52]. Die Kommunikationsobjekte können sowohl aus einem atomaren Datentyp als auch aus mehreren atomaren Datentypen (strukturierter Datentyp) bestehen [55]. Folgende typische atomare Datentypen werden von KNX unterstützt: Boolean, Character, 8 Bit Unsigned Integer, 8 Bit Signed Integer, 16 Bit Float usw. Ein Kommunikationsobjekt besitzt weiterhin eine Reihe von Attributen wie z.B. Nummer, Name, Funktion, Gruppenzugehörigkeit, Flags. Die Flags bestimmen, ob ein Kommunikationsobjekt gelesen oder geschrieben werden kann oder ob Wertänderungen gemeldet werden müssen. Die Metainformationen der Kommunikationsobjekte sind jedoch im regulären Betrieb nicht für andere Geräte sichtbar, sondern sie werden in der Konfigurationsphase gesetzt bzw. ausgelesen. Beispielhafte KNX-Kommunikationsobjekte sind ein Schalter (1 Bit Ein/Aus) oder ein relativer Dimmer (*Integer* Prozent). Jedes Kommunikationsobjekt wird einer Gruppe zugeordnet und ist über die Gruppenadresse adressierbar.

Die Kommunikationsobjekte werden zu Funktionsblöcken zusammengefasst und stellen in der Regel die Geräte selbst dar. Ein verbreiteter Funktionsblock ist ein elementarer Dimmer, der aus einem Schalter, einem relativen Dimmer und einem absoluten Dimmer besteht. Alle anderen Kommunikationsobjekte sind optional.

Das Verknüpfen von Kommunikationsobjekten wird mittels Bindings realisiert [52]. Der Zugriff auf die Kommunikationsobjekte erfolgt mittels Services. Die KNX-Services sind mit den Nachrichtentypen vergleichbar. Die Services lassen sich in zwei Klassen aufteilen. Die erste Klasse dient dem Datenaustausch zwischen den Geräten. Die wichtigsten und am meisten benutzten sind Services für die Gruppenkommunikation. Diese sind A_GroupValue_Read und A_GroupValue_Write. Die zweite Serviceklasse beinhaltet Services zur Konfiguration und Wartung wie z.B. das Setzen von Gruppenadressen, das Übertragen einer spezifischen Anwendung usw. Sowohl A_GroupValue_Read als A_GroupValue_Write Services werden als Multicast verschickt. A_GroupValue_Write Service wird benutzt, um neue Daten an eine Gruppe zu verteilen. A_GroupValue_Write ist

ein unbestätigter Service und garantiert somit nicht die Zustellung der Nachricht. Jeder Knoten, der die angegebene Gruppenadresse besitzt, kann die Nachricht verarbeiten. Da keine andere Datenübertragungsmöglichkeit im laufenden Betrieb außer Multicast besteht, sind Sicherheitsmechanismen bei KNX schwer umsetzbar. Der A_GroupValue_Read Service repräsentiert dagegen eine Anfrage. Diese wird von allen Knoten, die der Gruppe angehören, mit A_GroupValue_Response beantwortet. Die Antwort wird ebenfalls per Multicast verschickt und ist äquivalent zur A_GroupValue_Write-Nachricht. Das bedeutet jedoch, dass die Knoten, die keine Anfrage gestellt haben, die Antwort trotzdem bekommen. Der oben beschriebene Aufbau der KNX-Anwendungsschicht ist in Abbildung 10 dargestellt.

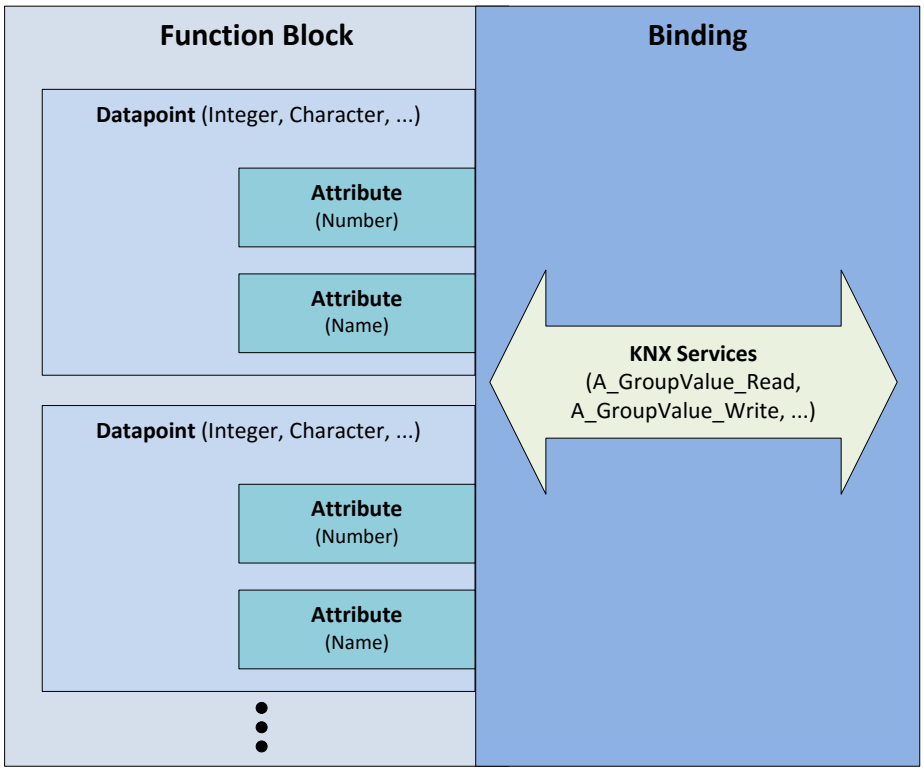


Abbildung 10: Struktur der KNX-Anwendungsschicht

Mit dem KNX-Standard können prinzipiell sowohl kleine als auch große Installationen umgesetzt werden. KNX-Produkte sind jedoch recht teuer. Eine Investition in die KNX-Installation lohnt sich im Allgemeinen nur, wenn mehrere Gewerke mit einander verknüpft werden sollen [40]. Die Installation von KNX-Geräten ist grundsätzlich nur durch Elektrohandwerksbetriebe oder Ingenieurbüros möglich.

KNX wird primär in der Feldebene der Gebäudeautomation eingesetzt (vgl. Abbildung 6). Einzig kann die IP-basierte Variante von KNX in der Automationsebene Verwendung finden [40].

2.3.4. LON

LON steht für Local Operation Network und ist ein im Jahr 1990 von der Firma Echelon entwickeltes Bussystem. LON oder auch LonWorks wurde zunächst im Jahr 1999 als ANSI / Electronic Industries Alliance (EIA)-709 und ANSI / Consumer Electronics Association (CEA)-709 standardisiert [56]. LON wurde erst 2005 zum europäischen Standard EN 14908 und 2008 zum internationalen Standard ISO / International Electrotechnical Commission (IEC) 14908 [57] [58]. LON steht in direkter Konkurrenz zu KNX.

LON besteht aus dem Kommunikationsprotokoll LonTalk, dem Neuron Chip, dem LonWorks Transceiver und den LonWorks Tools [40]. Der Neuron Chip stellt das Herzstück der LON-Technologie dar. Er beinhaltet 3 Prozessoren, Read Only Memory (ROM) und Random Access Memory (RAM). Jeder Prozessor ist für eine spezielle Aufgabe bestimmt. Der MAC-Prozessor steuert den Zugriff auf das Medium sowie den Empfang und Versand von Nachrichten. Der Netzwerkprozessor verarbeitet die oberen Schichten des LonTalk-Protokolls. Der Anwendungsprozessor führt eine spezifische Anwendung aus, die die Aufgaben eines Gerätes definiert [56]. Die Programmierung für den Neuron Chip erfolgt in einer speziellen Sprache Neuron C, die einen ANSI C Dialekt darstellt. Neben dem für LON üblichen Neuron Chip gibt es vereinzelte Lösungen, die auf einem Advanced „Reduced Instruction Set Computer (RISC)“ Machines (ARM)-Prozessor basieren, wie z.B. LC3020 Chip der Firma LOYTEC.

Für den Anschluss des Neuron Chips an das Übertragungsmedium wird ein separates Bauteil, der LonWorks Transceiver, verwendet. Für die Übertragung des LonTalk-Protokolls stehen Transceiver für folgende Medien zur Verfügung: TP, RS-485, PL, RF, Lichtwellenleiter (LWL), Infrarot (IR), Ethernet und andere [59]. Damit LON-Geräte im Netzwerk erkannt werden, haben sie eine weltweit eindeutige 48 Bit Identifikationsnummer, die Neuron ID genannt wird.

Für die Konfiguration der Geräte werden spezielle LonWorks Tools verwendet. Während der Konfiguration werden den Geräten logische Adressen zugewiesen. Diese bestehen aus einer Domain, einem Subnetz und einer Node ID. Die Domain stellt das gesamte LON-Netzwerk dar, das mit einer Domain ID identifiziert wird. Das LON-Netzwerk besteht physikalisch aus mehreren Kanälen, die durch Router und Repeater verbunden sind. Logisch werden sie in Subnetze unterteilt. LON implementiert das ISO/OSI-Netzwerkmodell und besitzt sowohl medienabhängige als auch medienunabhängige Schichten. Im Rahmen dieser Arbeit wird jedoch hauptsächlich auf die Anwendungsschicht von LON eingegangen.

Die Netzwerkschicht von LON ermöglicht Unicast-, Multicast- und Broadcast-Nachrichten. Das LonTalk-Protokoll stellt für den Datenaustausch zwischen den Geräten im Betrieb vier Services zur Verfügung, die in Tabelle 5 dargestellt sind. Neben den Services für den

Datenaustausch existiert noch eine Reihe von weiteren Services für das Netzwerkmanagement und die Diagnostik.

Für den Austausch von Daten werden sogenannte Netzwerkvariablen verwendet [56]. Sie sind grundlegende Kommunikationsobjekte, die logische Datenpunkte der Anwendung darstellen. Die Netzwerkvariablen besitzen eine Reihe von Properties, die Metadaten eines Kommunikationsobjektes darstellen. Die Eingangsnetzwerkvariablen können Werte vom Netzwerk empfangen und die Ausgangsnetzwerkvariablen die Werte ins Netzwerk schicken. Die Verknüpfungen zwischen den Eingangs- und Ausgangsnetzwerkvariablen werden mittels Bindings realisiert, die während der Konfiguration auf den Geräten durch das Setzen spezieller Tabelleneinträge eingerichtet werden. Die Werte werden automatisch gemeldet, wenn sich der Wert geändert hat, oder auch in eingestellten zeitlichen Abständen. Wenn sich beispielsweise die Werte der Ausgangsnetzwerkvariablen ändern, werden die Tabelleneinträge benutzt, um die Nachrichten an die Knoten zu übermitteln, die die entsprechenden verknüpften Eingangsvariablen besitzen. Damit werden die Werte der Eingangsvariablen auf den verknüpften Knoten aktualisiert. Die Bindings können 1-zu-1, 1-zu-n und n-zu-1 sein. 1-zu-n-Bindings werden benutzt, wenn mehrere Eingangsnetzwerkvariablen den Wert einer Ausgangsnetzwerkvariable brauchen. Ein n-zu-1 Binding ist problematischer, da der Wert einer Eingangsnetzwerkvariable unter Umständen von mehreren Quellen gleichzeitig überschrieben wird. Die Grundvoraussetzung für ein Binding ist die Kompatibilität der Ausgangsvariablen eines Sensors mit den Eingangsvariablen eines Aktors [40].

Service	**Erklärung**
Acknowledged	Die Nachricht wird an einen oder mehrere Knoten geschickt und eine Empfangsbestätigung wird erwartet.
Request/Response	Die Nachricht wird an einen oder mehrere Knoten geschickt und eine Antwort wird erwartet.
Repeated	Die Nachricht wird mehrmals an einen oder mehrere Knoten geschickt
Unacknowledged	Die Nachricht wird einmal an einen oder mehrere Knoten geschickt

Tabelle 5: LonTalk-Services für den Datenaustausch

Um spezielle Automationsfunktionen und die entsprechenden Netzwerkvariablen zu unterteilen, werden sie ähnlich wie bei KNX in die Funktionsblöcke zusammengefasst. Ein Funktionsblock definiert das Interface eines Gerätes, das aus den Eingangs- und Ausgangsnetzwerkvariablen sowie Konfigurations-Properties besteht. Die Konfigurations-Properties können dabei nur von den LonWorks Tools gesetzt werden. Um die Interoperabilität zwischen verschiedenen Herstellern zu erreichen, werden standardisierte

Templates für die Implementierung der Funktionsblöcke, sogenannte Standard Functional Profile Types (SFPT), von LonMark definiert. Diese definieren die obligatorischen und optionalen Netzwerkvariablen und Konfigurations-Properties. Für die Interoperabilität der Netzwerkvariablen und Konfigurations-Properties werden Standard Network Variable Types (SNVT) und Standard Configuration Property Types (SCPT) von LonMark spezifiziert [60]. Diese Variablen können von atomaren oder auch komplexen Datentypen sein. Zu den atomaren Datentypen gehören u.a. Short, Long und Quad Integer, Float, Enumeration usw. Die komplexen Datentypen sind Struct oder Union, gebildet aus atomaren Datentypen. Eine beispielhafte komplexe Netzwerkvariable ist SNVT_environment, die als ein Struct u.a. aus SNVT_environment.supplyVoltage, SNVT_environment.supplyCurrent und SNVT_environment.power besteht. Die Hersteller können weiterhin eigene proprietäre Netzwerkvariablen und Konfigurations-Properties verwenden, die jedoch von anderen Herstellern in der Regel nicht interpretiert werden können. Der oben beschriebene Aufbau der LON-Anwendungsschicht ist in Abbildung 11 dargestellt.

Ähnlich wie bei KNX ist die Installation von LON-Geräten grundsätzlich nur durch Elektrohandwerksbetriebe oder Ingenieurbüros möglich. Als eine klassische Bustechnologie wird LON in der Feldebene der Gebäudeautomation eingesetzt [40].

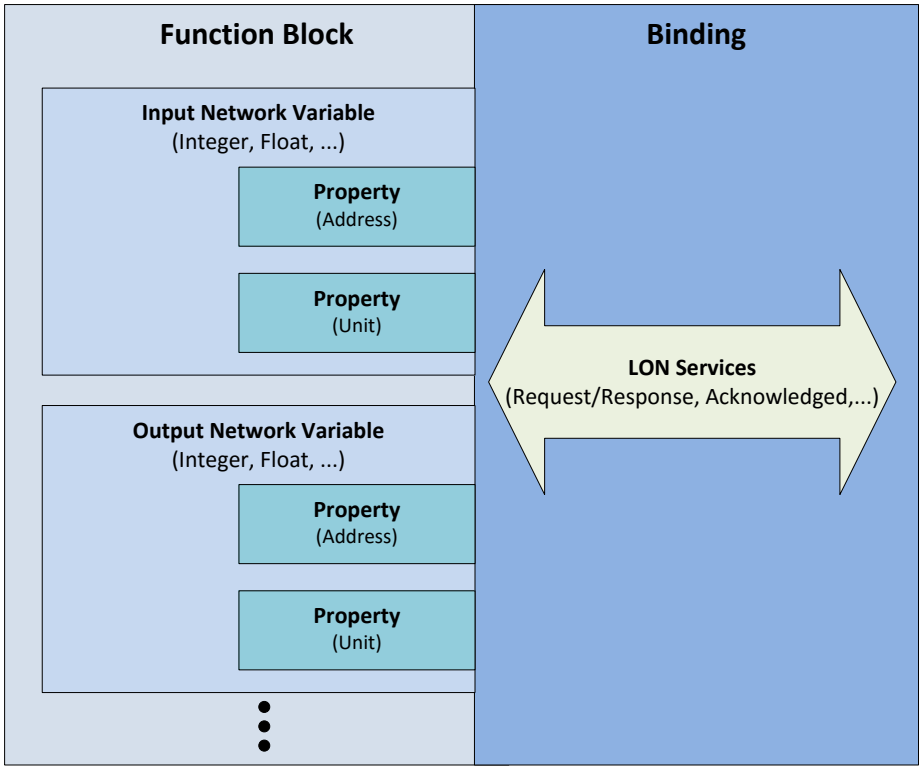


Abbildung 11: Struktur der LON-Anwendungsschicht

2.3.5. Neue Technologien der Gebäudeautomation und des Smart Home

2.3.6. Z-Wave

Z-Wave ist eine proprietäre Technologie für die Gerätekommunikation. Zum Zeitpunkt des Schreibens wurden nur die physikalische und Sicherungsschicht von Z-Wave von International Telecommunication Union Telecommunication Standardization Sector (ITU-T) im Jahr 2012 als G.9959 Norm standardisiert. Z-Wave wurde von der Firma Sigma Designs (ehem. Zensys) entwickelt. Diese ist gleichzeitig ein Hersteller der Z-Wave-Chips. Andere Hersteller, die die Technologie nutzen wollen, sind Mitglieder der Z-Wave Alliance. Ziel von Z-Wave ist, eine einfache und zuverlässige Methode zur Steuerung von Geräten zu Hause zu entwickeln [61]. Das Anwendungsgebiet von Z-Wave soll in erster Linie das Smart Home sein.

Z-Wave ist eine reine Funktechnologie. Für die Signalübertragung wird das Short Range Devices (SRD)-Band 868,42 MHz benutzt [62]. Die Z-Wave-Geräte bilden ein Mesh-Netzwerk, d.h., die Kommunikation läuft zwischen zwei Knoten über andere Netzwerkknoten als Vermittler ab. Die vermittelnden Knoten können dabei gewöhnliche Geräte sein, die ebenfalls Sensoren oder Aktoren sind. Eine Nachricht kann jedoch maximal vier Mal weitergeleitet werden [61].

In einem Z-Wave-Netzwerk gibt es zwei Gerätetypen: Controller und Slaves [63]. Die Controller können die Routen im Mesh-Netzwerk berechnen. Die Slaves sind Sensoren und Aktoren. Die berechneten Routen können an die Slaves weitergeleitet werden, um Verbindungswege zu erstellen. Das Netzwerk wird um einen Controller herum aufgebaut. Es können mehrere Controller im Netzwerk vorhanden sein, jedoch kann nur einer davon der Primary Controller sein. Dieser entscheidet u.a., welche Geräte dem Netzwerk beitreten dürfen. Die Controller können Kommandos initiieren und diese an andere Knoten schicken. Die Slaves enthalten grundsätzlich keine Routing-Tabelle. Die sogenannten Routing Slaves oder Enhanced Routing Slaves können vorkonfigurierte Routen vom Controller bekommen. Batteriebetriebene Geräte, die nicht permanent auf den Netzwerkverkehr lauschen, werden nicht als Vermittlerknoten berücksichtigt. Die Slaves können nur auf Kommandos antworten und diese ausführen [64]. Sie können nicht selbstständig Informationen an andere Slaves oder Controller senden, es sei denn, sie wurden dazu mit einem Kommando aufgefordert. Um Änderungen der Werte oder des Zustandes der Slaves festzustellen, muss der Controller die Slaves regelmäßig abfragen. Eine Ausnahme bilden die Routing Slaves, die Nachrichten wie z.B. Alarmnachrichten unaufgefordert schicken können.

Um einem Netzwerk beizutreten, benötig das Gerät mindestens einen „Initiator“. Ein Initiator kann ein physikalisch vorhandener Knopf, eine spezielle Betätigung des Knopfes (z.B. gedrückt halten für einige Sekunden), eine Kombination von Knöpfen (z.B. zwei Knöpfe

gleichzeitig drücken) oder ein Punkt in einem Auswahlmenü sein. Die Slave-Initiatoren werden benutzt, um dem Netzwerk beizutreten bzw. es zu verlassen und die Knoten mit einander zu verknüpfen bzw. zu trennen. Die Controller verfügen darüber hinaus über Operations-Initiatoren, die für die Steuerung der verknüpften Geräte benutzt werden. Geräte, die über keine Knöpfe verfügen, können nur mittels Controller verknüpft werden.

Mehrere Z-Wave-Netzwerke werden voneinander mittels Home ID getrennt. Die Home ID ist ein 32 Bit Identifier und wird fest in die Controller einprogrammiert. Um einzelne Geräte zu adressieren, wird eine Node ID verwendet. Diese ist 8 Bit breit und wird vom Primary Controller zugewiesen. Ein Z-Wave-Netzwerk kann jedoch maximal 232 Knoten enthalten. Z-Wave unterstützt alle drei Standard-Adressierungsarten Unicast, Multicast und Broadcast. Die Unicast-Nachrichten können sowohl bestätigt als auch unbestätigt sein. Die Multicast-Nachrichten werden dabei nicht wie üblich an eine Gruppenadresse geschickt, sondern sie enthalten eine Liste von Zieladressen, die die Nachricht bekommen sollen.

Um gerätespezifische Informationen zu bekommen, werden Node Information Frames geschickt. Dieser Frame ist ein Teil des Z-Wave-Protokolls und beschreibt die Fähigkeiten eines Gerätes wie z.B. Gerätetyp (Controller oder Slave), welche Funktionen unterstützt werden, ob das Gerät Nachrichten weiterleiten kann und andere protokollspezifische Parameter [61].

Die Anwendungsschicht von Z-Wave besteht aus drei Teilen: Kommandoklasse, Kommando und Kommandoparameter. Kommandoklassen sind eine Gruppe von Kommandos und Antworten, die eine bestimmte Funktion eines Gerätes abbilden. Die Kommandoklassen werden in zwei Typen unterteilt: Z-Wave-Protokollkommandos und anwendungsspezifische Kommandos. Die Z-Wave-Protokollkommandos werden hauptsächlich für das Setzen der Home ID und Node ID auf den Geräten verwendet. Z-Wave definiert eine Menge von anwendungsspezifischen Kommandoklassen wie z.B. Multilevel_Sensor oder Thermostat_Setpoint. Für die Kommunikation müssen einige Kommandoklassen von den Geräten unterstützt werden, die für eine bestimmte Geräteklasse wie z.B. Thermostat vorgeschrieben sind. Welche Kommandoklassen unterstützt werden, wird von den Geräten während des Netzwerkbeitritts dem Controller mittels Node Information Frame gemeldet. Die meisten Kommandos beschränken sich auf SET-, GET- und REPORT-Methoden. Die SET-Methode ermöglich eine Modifikation eines Datums auf dem Gerät. GET ist eine Abfrage eines Datums und REPORT ist eine Antwort auf diese Abfrage [61]. Die Kommandoklasse SwitchMultilevel würde z.B. auf eine GET-Anfrage Get() den aktuellen Zustand des Schalters liefern. Mit der SET-Anfrage Set(level, duration) kann der neue Zustand des Schalters gesetzt und optional ein Zeitintervall für den neuen Zustand angegeben werden [65]. Die Parameter, die zusammen mit den Kommandos übertragen werden, haben in der Regel atomare Datentypen wie Integer, Float, String. Komplexe Datentypen werden bei einigen wenigen

Kommandoklassen eingesetzt. Um Interoperabilität zu gewährleisten, müssen alle Kommandos, die zu einer Klasse gehören, auf dem Gerät implementiert werden. Der oben beschriebene Aufbau der Z-Wave-Anwendungsschicht ist in Abbildung 12 dargestellt.

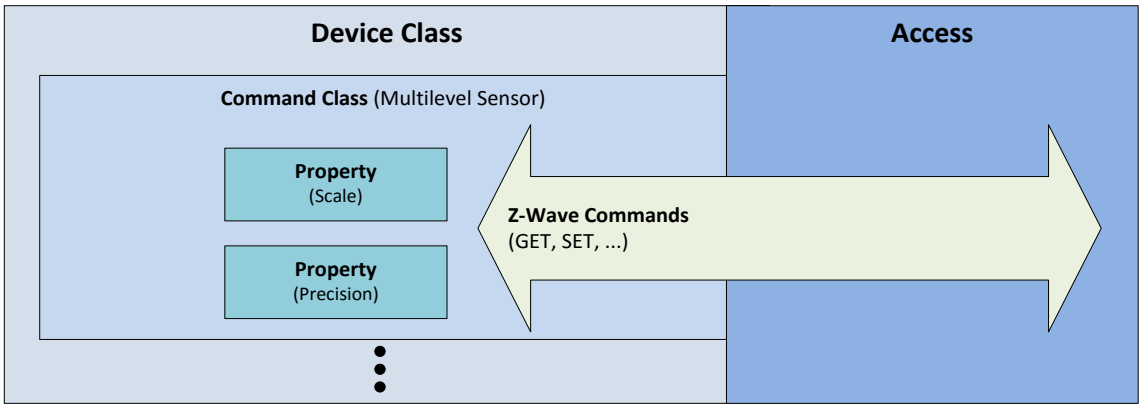


Abbildung 12: Struktur der Z-Wave-Anwendungsschicht

Der Vorteil der Z-Wave-Technologie ist eine leichte Konfigurierbarkeit der Geräte ohne ein tiefgreifendes Fachwissen. Der Primary Controller vom Z-Wave-Netzwerk stellt jedoch einen SPoF dar. Fällt dieser aus, wird das gesamte Netzwerk außer Betrieb gesetzt. Alle Geräte müssen dabei zurückgesetzt und mit dem neuen Primary Controller neu konfiguriert werden [63].

2.3.7. EnOcean

Die EnOcean-Technologie wurde von der Firma EnOcean GmbH entwickelt, die im Jahr 2001 als ein Spin-Off der Siemens AG entstand [66]. Das Hauptaugenmerk von EnOcean liegt auf den batterielosen energieautarken Geräten. Die Energie wird dabei aus der Umgebung mittels „Energy Harvesting“ geschöpft. Im Jahr 2012 wurde EnOcean zum internationalen Standard ISO/IEC 14543-3-10. Der Standard deckt dabei die OSI-Schichten 1-3 ab. Die Anwendungsschicht ist derzeit noch nicht standardisiert. EnOcean spezifiziert ein drahtloses Protokoll für Geräte mit niedrigem Energieverbrauch für das Wohnumfeld. Das Protokoll ist so entwickelt, dass der Energieverbrauch von Geräten extrem niedrig gehalten werden kann [67]. Andere Hersteller, die EnOcean-Produkte herstellen wollen, haben sich in der EnOcean Alliance zusammengeschlossen.

EnOcean ist eine reine Funktechnologie. Für die Signalübertragung wird in Europa das SRD-Band 868 MHz verwendet. Die Energiequellen können mechanischer (elektrodynamischer Energiewandler), solarer (Solarzellen) oder thermischer (Peltier-Elemente) Herkunft sein [66]. Für die Umgebungen oder Anwendungen, die kein Energy Harvesting ermöglichen, können ebenfalls batteriebetriebene Geräte eingesetzt werden. Im Gegensatz zu Z-Wave bildet EnOcean kein Mesh-Netzwerk, d.h., zwei kommunizierende Geräte müssen in

unmittelbarer Reichweite sein. Falls das nicht möglich ist, können EnOcean Repeater Abhilfe schaffen. Diese können das Signal verstärken und damit längere Distanzen überbrücken [68].

Die Datenübertragung findet in der Regel unidirektional statt. Grund dafür vor allem, dass die Geräte nur für kurze Zeit eine Energieversorgung haben. Im Gegensatz zu anderen Funkprotokollen ist ein Mechanismus zur Kollisionsvermeidung (CSMA/CA) optional und wird von den energieautarken Geräten nicht unterstützt. Um die Zuverlässigkeit der Zustellung der Datenpakete zu erhöhen, werden diese bis zu drei Mal hintereinander gesendet [69]. Eine bidirektionale Kommunikation ist mittels Smart Acknowledge möglich [70]. Diese Funktion ist jedoch optional, da diese bei Sensoren einen zusätzlichen Energieverbrauch impliziert. Darüber hinaus ist ein zusätzlicher netzbetriebener Repeater oder Controller notwendig, der die Rolle eines Postmasters für den Sensor übernimmt.

Für die Interoperabilität zwischen verschiedenen Herstellern werden EnOcean Equipment Profiles (EEP) definiert. Ein EEP besteht aus drei Teilen: Radio-Telegramm-Typ, Basisfunktionalität und Gerätetyp. Der Typ des Telegramms beschreibt, welche Anwendungsdaten übertragen werden wie z.B. „Variable Length Data“, „Manufacturer Specific Communication“ oder „Remote Management“ [71]. Die Basisfunktionalität beschreibt eine Geräteklasse wie z.B. „Schaltwippe mit 2 Tasten“ oder „mechanische Klinke“. Der Gerätetyp beschreibt ein konkretes Gerät aus der Klasse wie z.B. Lichtsteuerung oder Fensterklinke respektive. Die übertragenen Nutzdaten sind stets von einem atomaren Datentyp wie Integer oder Enumeration. Es können jedoch mehrere Werte in einem Telegramm geschickt werden wie z.B. Helligkeits- und Temperaturwerte.

Das Koppeln der EnOcean Geräte wird als „Teach-in“ (Lernprozess) bezeichnet. In der Teach-in-Prozedur wird festgelegt, welcher Empfänger (Controller) auf welchen Sender (Sensor/Schalter) hören soll. Für diesen Zweck besitzt jeder Sender eine einzigartige Sender-ID. Anhand der Sender-ID erkennt der Empfänger, ob das Gerät bereits bekannt (gelernt) oder unbekannt ist. Nachrichten von unbekannten Sendern werden verworfen. Der Lernprozess läuft grundsätzlich in zwei Schritten ab. Zuerst wird der Empfänger in den Lernmodus versetzt. Das kann durch die Betätigung einer speziellen Taste auf dem Gerät erfolgen. Der Sender soll nun sein Telegramm einmalig schicken. Das kann ebenfalls durch das Drücken einer Taste erfolgen. Im Lernmodus werden die empfangenen Telegramme vom Empfänger als autorisiert betrachtet und die Sender-ID wird gespeichert. Weitere Schritte können abhängig vom Gerätetyp folgen. Damit keine anderen Geräte fälschlicherweise gelernt werden, wird die Empfangsstärke des Empfängers reduziert. Die Geräte müssen dabei sehr dicht aneinander gebracht werden. Die Controller können alternativ mittels Remote Management-Nachrichten konfiguriert werden. Diese Art der Konfiguration muss jedoch von Controllern nicht zwingend unterstützt werden [71]. Die Nachrichten vom Sender werden immer als Broadcast geschickt. Der Sender hat somit keine Informationen, welche Empfänger

die Nachricht erhalten sollen. Jeder Empfänger entscheidet beim Empfang eines Telegramms anhand der Sender-ID, ob er diese Nachricht auswerten soll oder nicht. Der oben beschriebene Aufbau der EnOcean-Anwendungsschicht ist in Abbildung 13 dargestellt.

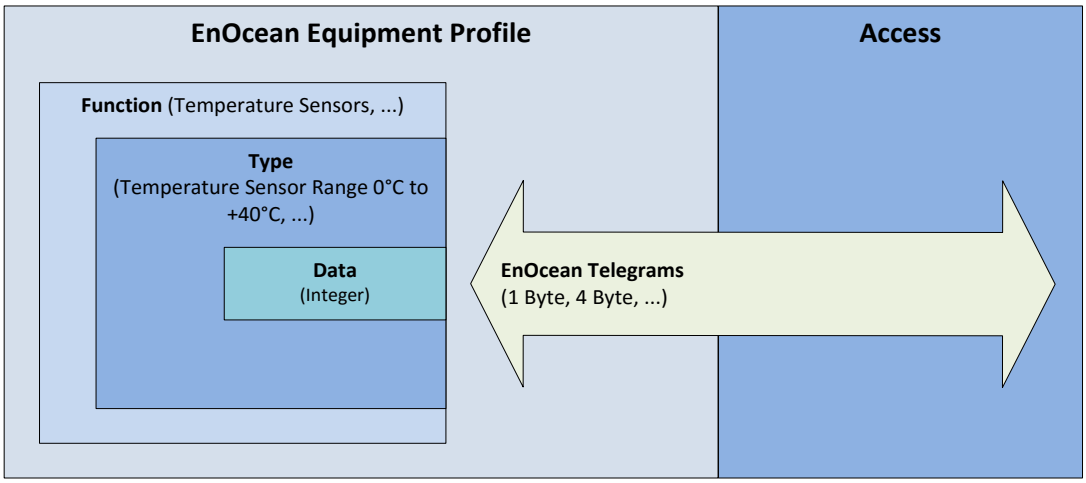


Abbildung 13: Struktur der EnOcean-Anwendungsschicht

EnOcean-Geräte stellen eine relativ einfache Funktionalität bereit. Für komplexere Installationen werden deswegen Gateways zur Anbindung der EnOcean-Geräte in die klassischen Protokolle der Gebäudeautomatisierung wie BACnet [72], KNX [73] und LON [74] angeboten. Die EnOcean-Technologie eignet sich prinzipiell sowohl für Gebäudeautomation als auch Smart Home. Die Gerätepreise sind jedoch u.a. technologiebedingt relativ teuer.

2.3.8. ZigBee

ZigBee ist eine drahtlose Technologie für die Verbindung von kostengünstigen Geräten mit geringem Stromverbrauch. ZigBee basiert auf dem Standard IEEE 802.15.4, der die physikalische und Sicherungsschicht beschreibt. Die ZigBee-Technologie wird von ZigBee Alliance entwickelt, die 2002 gegründet wurde [75]. Im Gegensatz zu den meisten Protokollen der Gebäudeautomation ist die Spezifikation von ZigBee öffentlich verfügbar, was es zu einem offenen Standard macht. ZigBee wird als direkter Konkurrent zu Z-Wave angesehen.

ZigBee ist eine reine Funktechnologie. Für die Signalübertragung können sowohl 2,4 GHz als auch 868 MHz verwendet werden. Die physikalische Datenübertragung sowie der Medienzugriff werden in IEEE 802.15.4 standardisiert. Die ZigBee-Spezifikation beschreibt nur die oberen Netzwerkschichten (ab der Vermittlungsschicht). Es werden mehrere Netzwerktopologien wie Stern, Baum und Mesh unterstützt [76]. In der Sterntopologie wird das Netzwerk durch ein einziges Gerät, den sogenannten Coordinator, gesteuert. Alle anderen Geräte kommunizieren direkt mit dem Coordinator. In der Baum- und Mesh-Topologie wird

das Netzwerk von einem Coordinator initialisiert und gestartet, es kann aber durch zusätzliche Router erweitert werden. In der Baumtopologie wird eine hierarchische Routing-Strategie zwischen den Routern eingesetzt. In der Mesh-Topologie kann das Routing über beliebige Geräte ablaufen. Jedes ZigBee-Netzwerk wird durch eine Personal Area Network (PAN) ID identifiziert, die vom Coordinator gewählt wird. Des Weiteren vergibt der Coordinator den beitretenden Geräten eindeutige Netzwerkadressen.

ZigBee unterscheidet drei logische Gerätearten: Coordinator, Router und Endgerät [77]. Der Coordinator ist in der Lage, ein Netzwerk aufzubauen. Der Router ist ein Gerät, das die Nachrichten an andere Geräte weiterleiten kann. Sowohl Router als auch Coordinator sind Full Function Devices (FFD). Die FFDs können dabei ebenfalls Sensoren und Aktoren sein. Das Endgerät oder auch Reduced Function Device (RFD) ist das einfachste ZigBee-Gerät ohne Vermittlungsfähigkeit.

Für die Interoperabilität zwischen verschiedenen Herstellern werden Anwendungsprofile (Application Profiles) wie z.B. „Home Automation“ oder „Smart Energy“ definiert. Die Profile können sowohl öffentlich als auch herstellerspezifisch sein. Ein Profil umfasst einen Satz von Netzwerkparametern, Gerätetypen sowie eine Reihe von Nachrichten und Attributen [78]. Jedem Profil wird eine Profile ID zugeordnet. Die Geräte werden als Anwendungsobjekte (Application Objects) modelliert. Jedes Anwendungsobjekt ist einem Anwendungsprofil zugeordnet.

Die Daten oder Funktionen, die von einem Gerät angeboten werden oder der Steuerung des Gerätes dienen, werden als Attribute bezeichnet. Jedes Attribut wird mit einer Attribute ID gekennzeichnet. Ein Attribut wird durch einen Attributdatentyp beschrieben. Die Attribute haben atomare Datentypen wie Integer, Float, String usw. Eine Gruppe von Attributen und zugehörigen Zugriffsbefehlen bildet einen Cluster. Der Cluster arbeitet nach dem Prinzip des Client/Server-Modells. Somit besteht ein Cluster aus der Client- und der Server-Seite [79]. Der Server-Cluster wird als Input Cluster und der Client-Cluster als Output Cluster bezeichnet. Jedes Anwendungsobjekt kann einen oder mehrere Cluster beinhalten. Ein Cluster stellt damit das Interface eines Gerätes oder einen Teil davon dar [80]. Jeder Cluster wird mit einer Cluster ID gekennzeichnet. ZigBee Alliance hat zu Interoperabilitätszwecken eine ZigBee Cluster Library (ZCL) entwickelt, die alle anerkannten Cluster wie z.B. „On/Off“ oder „Level Control“ bereitstellt [81].

Die Cluster-Befehle werden benutzt, um Attribute zu lesen (Read Attributes), zu schreiben (Write Attributes) und Wertänderungen zu melden (Configure Report und Report Attributes). Jeder Befehl wird mit der Command ID gekennzeichnet. Der Schreib- und Lesebefehl kann gleichzeitig auf mehrere Attribute angewendet werden.

Eine besondere Rolle hat das ZigBee Device Object (ZDO). Das ZDO dient der Steuerung und dem Management eines Gerätes wie die Bestimmung der Rolle des Gerätes im Netzwerk, Geräte- und Service-Discovery und andere [79]. Die Objekte auf einem Gerät werden mittels Endpoint IDs adressiert. Der Zugriff auf ein Attribut erfolgt unter Angabe der Netzwerkadresse, der Profile ID, der Endpoint ID, der Cluster ID und der Attribute ID.

Die logische Kopplung der Geräte miteinander wird als Binding bezeichnet. Dabei wird eine Beziehung zwischen zwei Endpoints hergestellt. Das Binding ist unidirektional. Es sind 1-zu-1, 1-zu-n und n-zu-1-Bindings erlaubt. ZigBee unterstützt somit Unicast-, Multicast- und Broadcast-Nachrichten. Eine Voraussetzung für das Binding ist die Unterstützung des entsprechenden Input bzw. Output Clusters. Um das Binding auszuführen, melden sich beide Geräte bei dem Coordinator mit dem Binding-Request. Stimmen die Binding-Parameter überein, verschickt der Coordinator ein Binding-Response mit entsprechenden Parametern des Gegenübers [79]. Diese werden in die lokale Binding-Tabelle des Senders aufgenommen. Das Binding wird darum auch als Source Binding bezeichnet. Anschließend können die Geräte direkt miteinander kommunizieren. Alternativ kann das Binding mit einem Inbetriebnahme-Tool realisiert werden, das es ermöglicht, direkt Einträge in der Binding-Tabelle des Senders zu erstellen [82]. Der oben beschriebene Aufbau der ZigBee-Anwendungsschicht ist in Abbildung 14 dargestellt.

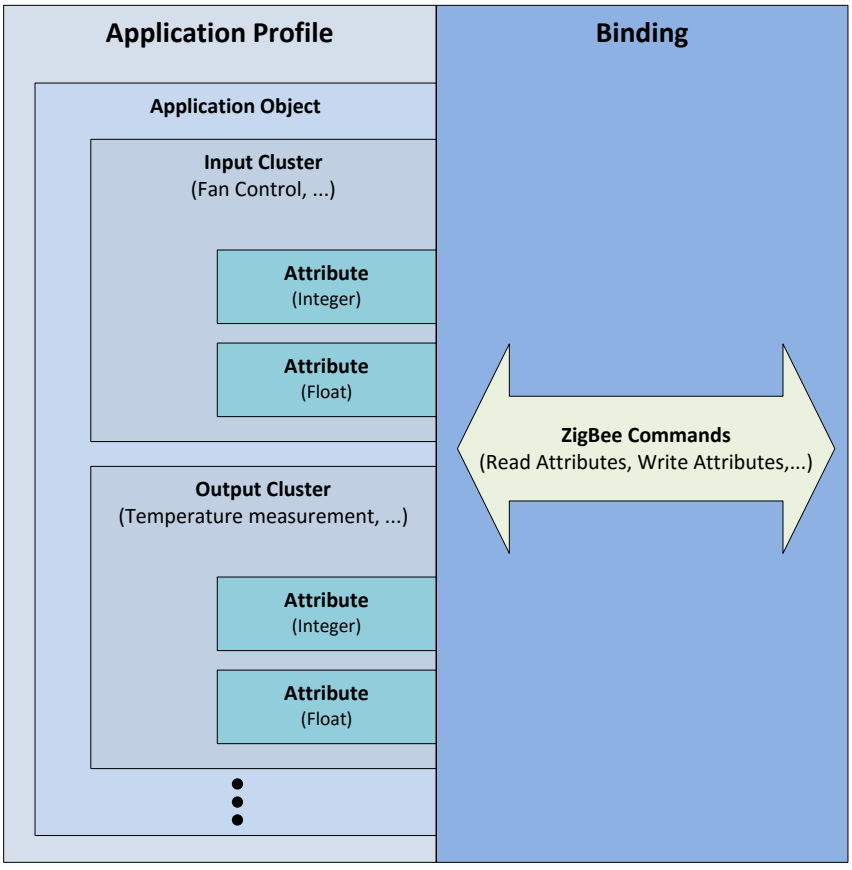


Abbildung 14: Struktur der ZigBee-Anwendungsschicht

Mit ZigBee IP wurde eine Stack-Erweiterung vorgestellt, die es ermöglicht, ZigBee-Nachrichten über IPv6 zu verschicken [83]. Hierfür wird der IPv6 over Low power Wireless Personal Area Network (6LoWPAN) Adaptation Layer verwendet, der ebenfalls auf IEEE 802.15.4 basiert. Damit eröffnen sich für ZigBee weitere Möglichkeiten; u.a. eine direkte Kommunikation mit anderen IP-fähigen Geräten in anderen Medien wie z.B. Ethernet.

Im Vergleich zu Z-Wave bietet ZigBee eine größere Palette an Möglichkeiten. Dadurch können komplexere Anwendungen umgesetzt werden. Den Herstellern von ZigBee-Produkten sind weniger Grenzen gesetzt. Allerdings sind in der Praxis viele ZigBee-Geräte dadurch nicht miteinander kompatibel. Darüber hinaus ist in der Praxis die Konfiguration der ZigBee-Geräte weniger benutzerfreundlich [84].

2.3.9. Weitere Technologien

Es existieren weitere proprietäre Technologien wie z.B. digitalSTROM von digitalSTROM AG [85] oder HomeMatic von eQ3 [86]. Geräte dieser Technologien sind zwar auf dem Markt erhältlich, werden jedoch von einer einzigen Firma hergestellt und sind nicht standardisiert. Der Nachteil solcher Technologien liegt darin, dass der Kunde sich an einen einzigen Hersteller bindet, der die Gerätepreise allein bestimmt. Verschwindet der Hersteller vom Markt, muss im Worst Case die komplette Anlage ersetzt werden, was mit sehr hohen Kosten verbunden ist.

3. Integration der Legacy-Technologien in Web Services

Der Markt erfordert immer flexiblere, automatisierte und erweiterbare Lösungen für das Smart Home. Für zukünftige Smart Home-Systeme ist es unabdingbar, über Plug&Play-Funktionalitäten zu verfügen. Der Grund hierfür ist die Notwendigkeit einer einfachen Installation der Geräte ohne tiefgreifendes Fachwissen. Somit können die Smart Home-Umgebungen unkompliziert und zu geringen Kosten in bestehenden Immobilien bzw. Neubauten eingerichtet werden. Die Technologie soll darüber hinaus offene Schnittstellen für Hersteller bereitstellen, um die Interoperabilität zu gewährleisten bzw. diese bei der Geräteentwicklung zu unterstützen. Solche Eigenschaften bieten wie in Abschnitt 2.2 beschrieben die Web Service-Technologien wie DPWS.

Ein kompletter Umstieg auf eine neue Technologie ist mit höheren Kosten verbunden, da alle bestehenden Anlagen/Installationen in diesem Fall vollständig modernisiert werden müssen. Eine kostengünstige Lösung ist dagegen eine schrittweise Modernisierung. Dabei werden nur einige Geräte ersetzt oder neue Geräte installiert. Die bestehenden Geräte sollen weiterhin genutzt und in die neue Installation integriert werden. Um dieses Ziel zu erreichen, müssen alte Geräte mittels eines Gateways mit DPWS-Geräten kommunizieren können. Im Rahmen des Projektes soll dies am Beispiel eines ausgewählten Protokolls gezeigt werden. Ausgehend aus der Analyse existierender Protokolle (vgl. Abschnitt 2.3) wird als geeignetes Protokoll BACnet ausgewählt. BACnet bietet im Vergleich zu anderen Protokollen die meiste Funktionsvielfalt. Darüber hinaus existieren bereits mehrere Gateway-Umsetzungen von BACnet zu anderen Protokollen wie z.B. KNX [87], LON [88], ZigBee [89] und andere. Dadurch wird es auch möglich sein, andere Protokolle in das DPWS-Gateway zu integrieren.

3.1. Vergleich von BACnet und DPWS und daraus resultierende Gateway-Anforderungen

Um die funktionalen Anforderungen des BACnet-DPWS-Gateways erfüllen zu können, müssen beide Standards im Hinblick auf die Funktionen verglichen und Äquivalenzen gefunden werden. Funktionalitäten, die das Gateway in beide Richtungen bieten muss, sind:

- Suchen und Finden von Netzwerkteilnehmern
- Lesen und Schreiben von Daten
- Abonnieren und Melden von Wertänderungen

Sowohl der DPWS-Standard als auch der BACnet-Standard beschreiben Schnittstellen für Geräte. Eine Aufgabe des BACnet-DPWS-Gateways muss es sein, eine DPWS-Schnittstelle als BACnet-Schnittstelle darzustellen und umgekehrt. Stellt man die Strukturen beider Standards gegenüber, so zeigt sich, dass beide hierarchisch aufgebaut sind (siehe Abbildung

15). Die Struktur eines BACnet-Gerätes setzt sich aus Objekten zusammen, welche wiederum Eigenschaften (Properties) besitzen. Die Struktur von DPWS-Geräten ist deutlich vielschichtiger angelegt. DPWS-Geräte bieten Services an. Ein Service bekommt für jeden Port über das Binding und den Port-Typ Operationen zugeordnet. Operationen können Nachrichten als Eingang (Input) oder Ausgang (Output) definieren. Die einzelnen Parts der Nachrichten enthalten dann durch die Zuweisung eines Typs Elemente. Durch den Vergleich beider Strukturen wird deutlich, dass eine Eins-zu-Eins-Zuweisung der einzelnen Komponenten nicht möglich ist. Für das Gateway muss ein Weg gefunden werden, die Strukturen ineinander zu überführen. Dabei ist vor allem wichtig, dass ein DPWS-Gerät eine BACnet-konforme Geräteschnittstelle erhält und umgekehrt. Für die Identifikation eines Gerätes sind in beiden Standards eindeutige Gerätekennungen definiert. Bei DPWS ist dies die Endpunkt-Referenz beziehungsweise der IRI darin und in BACnet ist es der ObjectIdentifier des Device-Objekts beziehungsweise die im BACnet-Netzwerk einmalige Instanznummer. Das heißt, dass einem BACnet-Gerät eine gültige IRI zugewiesen werden muss. Für das DPWS-Gerät muss das Gateway eine im Netzwerk noch nicht vergebene Instanznummer für das Device-Objekt finden und dem Gerät zuordnen. Die Zuordnung einer IRI zu einem BACnet-Gerät und einer Instanznummer zu einem DPWS-Gerät muss im Gateway gespeichert sein, um Nachrichten entsprechend weiterleiten zu können. Bei der Darstellung eines Gerätes im jeweils anderen Standard muss zudem der Inhalt der Elemente in DPWS dem Inhalt der Objekteigenschaften in BACnet entsprechen. Diese Komponenten beinhalten auf beiden Seiten Werte.

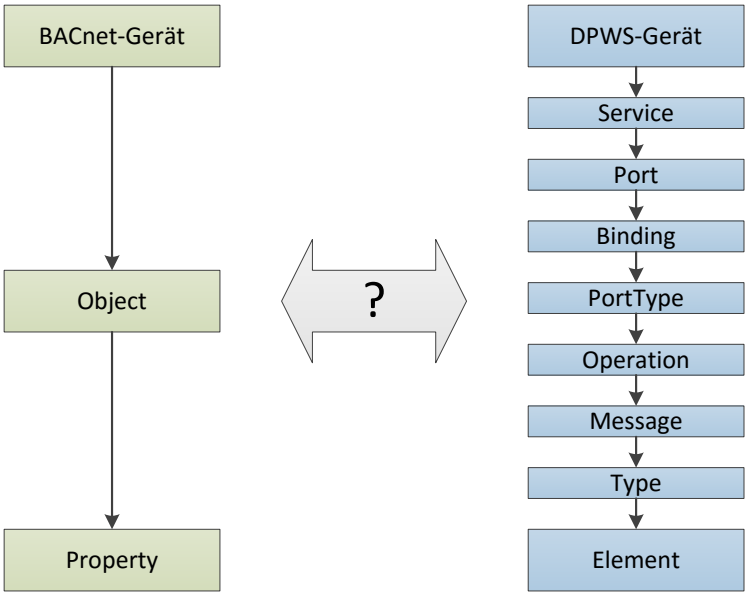


Abbildung 15: Gegenüberstellung von BACnet und DPWS

Auch die Nachrichten müssen durch das Gateway der Funktion entsprechend übersetzt werden. Zur allgemeinen Suche nach Geräten verschickt ein DPWS-Gerät eine Probe-Nachricht, auf die alle verfügbaren Geräte mit einer Probe-Match-Nachricht antworten

müssen. Einer Probe-Nachricht entspricht auf der BACnet-Seite der WhoIs-Dienst. Empfängt das Gateway eine Probe-Nachricht, ist es dessen Aufgabe, eine WhoIs-Nachricht per Broadcast in das BACnet-Netzwerk zu senden. Eine daraufhin eintreffende IAm-Nachricht hat die gleiche Funktion wie eine ProbeMatch-Nachricht auf der DPWS-Seite. Ist zuvor keine Probe-Nachricht beim Gateway eingegangen, entspricht eine IAm-Nachricht einer Hello-Nachricht. Das Gateway muss in der Lage sein, diese Unterscheidung vorzunehmen. Eine WhoIs-Nachricht, welche das Gateway erreicht, entspricht einer Probe-Nachricht und die dazugehörigen ProbeMatch-Nachrichten entsprechen IAm-Nachrichten. Auch Hello-Nachrichten haben in DPWS die gleiche Funktion wie IAm-Nachrichten in BACnet. Wenn ein bestimmtes Gerät gesucht werden soll, so kann dafür von beiden Seiten nicht die Gerätekennung verwendet werden, weil ein BACnet-Gerät keine eigene IRI besitzt und ein DPWS-Gerät keinen ObjectIdentifier beziehungsweise ein Device-Objekt. Über das Gateway ist aber der Geräte-Name ein möglicher Suchparameter. Damit ist es mit dem Gateway für einen DPWS-Client möglich, ein BACnet-Gerät zu finden und umgekehrt.

Zum Lesen, Schreiben und Abonnieren von Daten entsprechen die jeweiligen BACnet-Dienste DPWS-Operationen. Der ReadProperty-Dienst zum Lesen des Inhalts einer Objekteigenschaft gleicht einer Operation, deren Input-Nachricht im Gegensatz zur Output-Nachricht keine Elemente enthält. Die Elemente der Output-Nachricht enthalten die gelesenen Werte. Eine Operation mit einer Input-Nachricht, die Elemente als Eingangsparameter für die Operation enthält, bildet das Äquivalent zum WriteProperty-Dienst. Werden die BACnet-Dienste den DPWS-Operationen und umgekehrt entsprechend zugeordnet, können über das Gateway auch Daten zwischen Geräten beider Standards ausgetauscht werden.

Für die dritte Funktionalität, das Abonnieren und Melden von Wertänderungen, gibt es ebenfalls äquivalente Nachrichtentypen in beiden Standards. Mit EventSubscription auf DPWS-Seite und SubscribeCOV in BACnet kann sich ein Client registrieren, um Meldungen zu erhalten. Die Meldung erfolgt dann mit einer EventNotification- beziehungsweise COVNotification-Nachricht. Alle Parallelen der beiden Standards sind noch einmal in Abbildung 16 dargestellt.

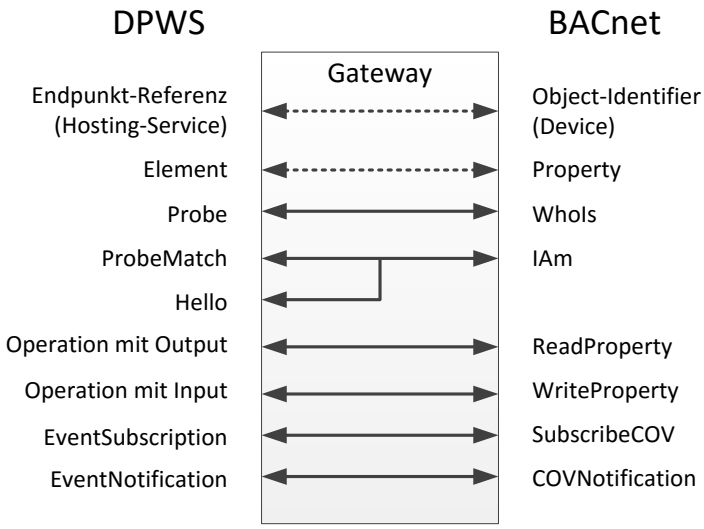


Abbildung 16: Parallelen zwischen DPWS und BACnet

3.2. Entwurf eines Konzepts für ein BACnet-DPWS-Gateway

Nach dem Vergleich beider Standards hinsichtlich ihrer Geräteschnittstellen und der Funktionen zum Informationsaustausch, muss ein Weg gefunden werden, die Gerätedarstellungen ineinander zu überführen. Dabei stehen zwei Fragen im Zentrum:

- Wie sieht die DPWS-Schnittstelle für ein BACnet-Gerät aus?
- Wie sieht die BACnet-Schnittstelle für ein DPWS-Gerät aus?

Das BACnet-DPWS-Gateway muss für jedes BACnet-Gerät, das sich im Netzwerk befindet, eine Schnittstellenbeschreibung in Form eines WSDL-Dokuments erzeugen und bereitstellen. Dafür gilt es, eine geeignete Zuweisung der einzelnen BACnet-Komponenten (Device, Object, Property) zu den WSDL-Komponenten (Device, Service, PortType etc.) zu finden. Fest steht, dass einem BACnet-Device durch das Gateway eine DPWS-Schnittstelle zugewiesen werden muss und umgekehrt. Weiterhin muss der Wert einer Objekteigenschaft dem Inhalt eines Elements entsprechen.

Eine Variante, die BACnet-Komponenten den WSDL-Komponenten zuzuordnen, ist, für jedes Objekt einen Service bereitzustellen. Dieser Service kann mit dem Objekttyp und/oder dem Objekt-Namen bezeichnet werden. Jede Objekteigenschaft wird auf einen Port-Typ abgebildet. Jeder Port-Typ enthält dann zum Lesen, Schreiben und Abonnieren der Objekteigenschaft jeweils eine Operation. In Abbildung 17 ist diese Variante dargestellt.

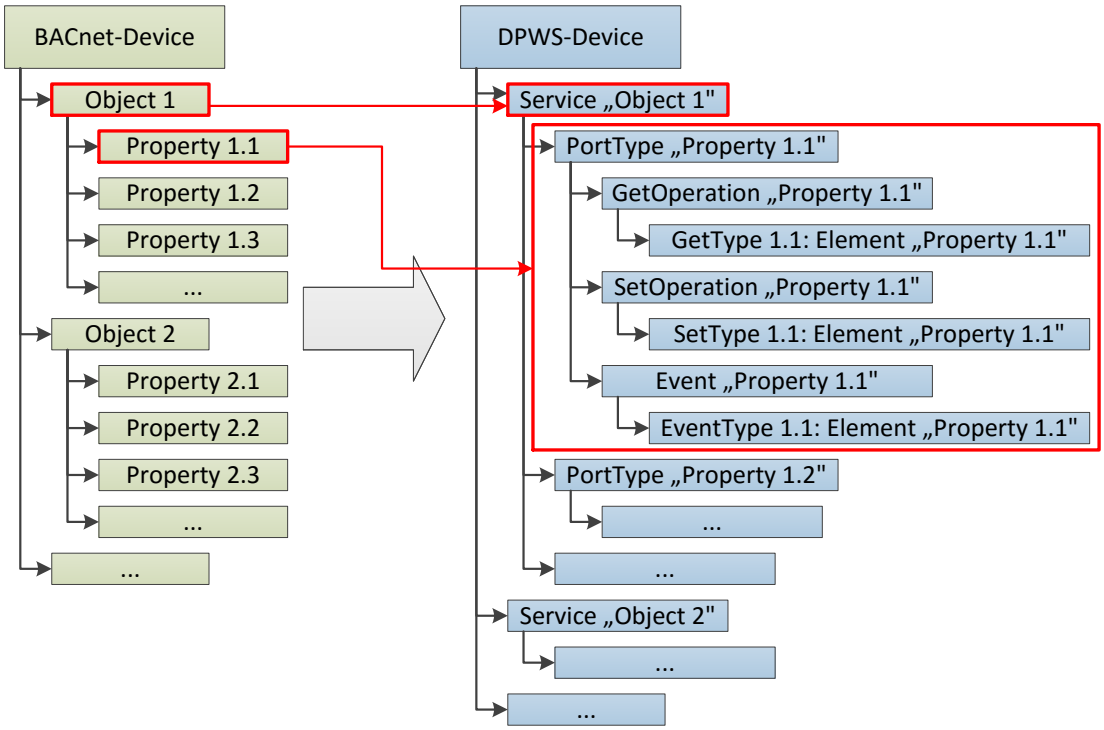


Abbildung 17: Erste Variante der BACnet-DPWS-Zuordnung

In diesem Fall hätte die DPWS-Schnittstelle so viele Services, wie das BACnet-Gerät Objekte besitzt. Jeder Service hätte eine noch größere Anzahl an Port-Typen und jeder Port-Typ hätte immer genau die drei Operationen zum Lesen, Schreiben und Abonnieren eines Wertes. Eine so flache Hierarchie ist nicht erwünscht, da mit jeder Operation immer nur eine Objekteigenschaft gelesen oder geschrieben werden kann, wodurch unter Umständen ein großer Overhead an Nachrichten im Netzwerk entsteht. Hat beispielsweise ein BACnet-Gerät 3 Objekte und jedes Objekt besitzt 10 Eigenschaften, wären 60 Nachrichten (Anfrage und Antwort) notwendig, um alle Werte eines Gerätes zu lesen.

Die zweite Variante sieht nur einen einzigen Service in der DPWS-Schnittstelle für ein BACnet-Gerät vor. Für jedes Objekt gibt es einen Port-Typ, der für jede Objekteigenschaft die entsprechenden Operationen bereithält. In Abbildung 18 ist zu sehen, dass die Hierarchie damit nicht mehr ganz so flach wie in der ersten Variante ist, da in jedem Fall nur ein Service angeboten wird.

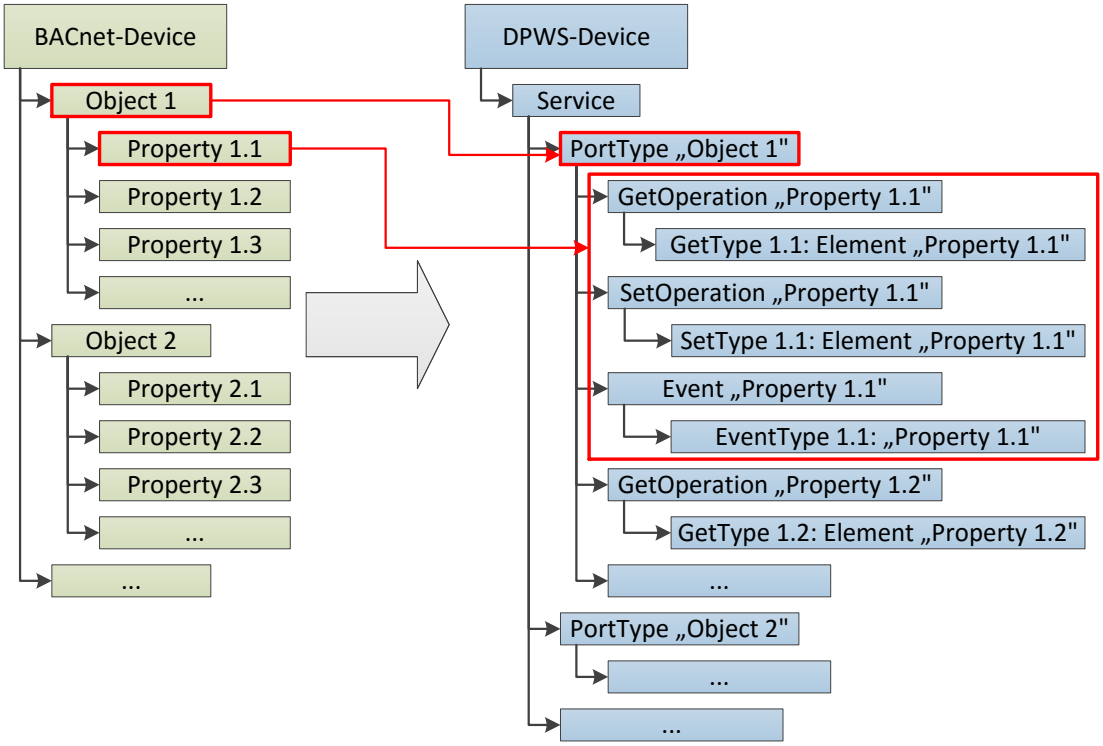


Abbildung 18: Zweite Variante der BACnet-DPWS-Zuordnung

Mit dieser Variante gibt es zwar nur noch einen Service, die Anzahl der Nachrichten ändert sich im Vergleich zu Variante 1 jedoch nicht. Auch in Variante 2 kann eine Operation nur auf eine Objekteigenschaft angewendet werden.

Die dritte Variante bietet ebenfalls pro BACnet-Gerät nur einen Service an. Die Operationen werden in dieser Variante aber immer auf ein ganzes Objekt und alle seine Objekteigenschaften angewendet. Die Objekteigenschaften sind in diesem Fall Elemente in einer Nachricht. Abbildung 19 stellt diese Variante dar.

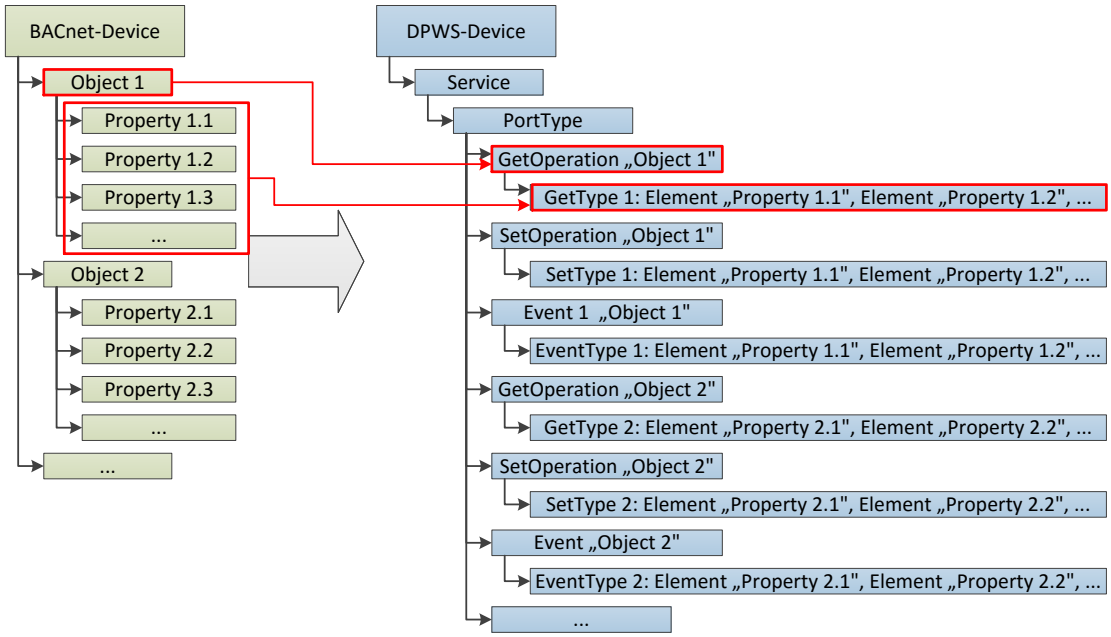


Abbildung 19: Dritte Variante der BACnet-DPWS-Zuordnung

Die Get-Operation entspricht dem ReadProperty-Dienst und dient dem Abrufen der Werte. Da der BACnet-Standard vorschreibt, dass alle Objekte den ReadProperty-Dienst unterstützen müssen, gibt es diese Operation für jedes Objekt und alle seine Eigenschaften. Die zu dieser Operation gehörende Nachricht enthält demnach für jede Objekteigenschaft ein Element. Die Set-Operation ist mit dem WriteProperty-Dienst gleichzusetzen. Die Nachricht der Set-Operation enthält nur Elemente für die Objekteigenschaften, die auch schreibbar sind. Das Gateway muss daher herausfinden, welche Objekteigenschaft verändert werden können. Ist gar keine Eigenschaft veränderbar, wird auch die Set-Operation nicht angeboten. Genauso verhält es sich für das Event, das dem SubscribeCOV-Dienst in BACnet entspricht. Es müssen also nicht immer alle Operationen für alle Objekte vorhanden sein.

In dieser Variante ist die Anzahl der auszutauschenden Nachrichten beim Lesen aller Werte eines Gerätes nur noch von der Anzahl der Objekte und nicht mehr von der Anzahl der Objekteigenschaften abhängig. Für ein BACnet-Gerät mit 3 Objekten mit je 10 Eigenschaften bedeutet das 6 Nachrichten im Vergleich zu 60 in den ersten beiden Varianten. Wegen der entscheidenden Vorteile wird Variante 3 für die Realisierung des BACnet-DPWS-Gateways ausgewählt. Der Port-Typ wird in dieser Variante keiner BACnet-Komponente direkt zugeordnet. Im BACnet-DPWS-Gateway wird der Port-Typ verwendet, um die Objekte in Gruppen einzuordnen. Diese Gruppen können physikalische Größen darstellen oder angeben, ob es sich um ein Binär-Objekt (Datentyp der PresentValue-Eigenschaft ist binär) oder ein MultiState-Objekt (die PresentValue-Eigenschaft enthält einen Zustand) handelt. Objekteigenschaften von einem primitiven Datentyp wie Boolean, Integer oder Float, werden in der DPWS-Schnittstelle als Elemente von diesem Datentyp repräsentiert. Andere Datentypen, die zum Teil BACnet-spezifisch sind, werden als CharacterString dargestellt. Ein

Beispiel für einen als CharacterString dargestellten BACnet-Datentyp ist zum Beispiel der ObjectIdentifier oder die BACnetStatusFlags.

Bei der Darstellung einer BACnet-Schnittstelle für ein DPWS-Gerät kann nicht von einer festen Datendarstellung in einem DPWS-Gerät ausgegangen werden, da durch den DPWS-Standard keine Darstellungsform vorgeschrieben wird. Elemente können beliebige Namen besitzen und sowohl einzeln als auch in einer komplexen Struktur angeordnet sein. Es wird hier davon ausgegangen, dass komplexe Typen nur primitive Typen und keine weiteren komplexen Typen enthalten. Die BACnet-Schnittstelle für ein DPWS-Gerät ist für einfache und komplexe Typen in Abbildung 20 dargestellt.

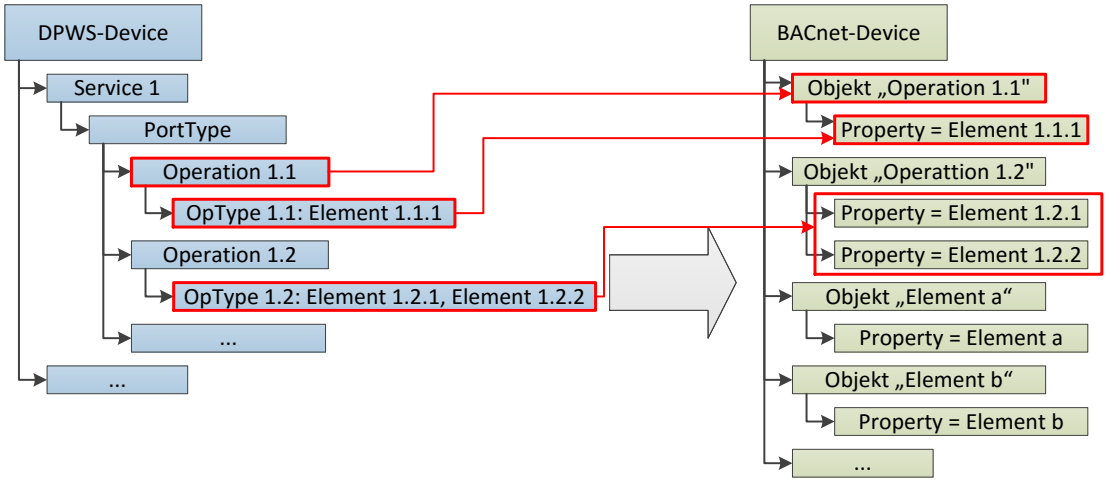


Abbildung 20: Die DPWS-BACnet-Zuordnung mit bekannten Element-Namen

Grundsätzlich gilt, dass jede Operation, die in einem DPWS-Gerät vorhanden ist, auf mindestens ein BACnet-Objekt abgebildet wird. Der Objekttyp richtet sich nach dem Datentyp der Elemente in den Operationen. So wird zum Beispiel ein Float-Element als AnalogValue-Objekt dargestellt, ein Boolean-Element als BinaryValue-Objekt und ein String-Element als CharacterStringValue-Objekt. Bei komplexen Typen besteht die Möglichkeit, die Elemente bestimmten Eigenschaften eines Objekts zuzuordnen. Dafür müssen aber vorgegebene Namen für die Elemente verwendet werden. Enthält eine Operation zum Beispiel ein Element „value“ vom Datentyp Float und ein Element „unit“ vom Datentyp CharacterString, dann kann das Gateway beide Elemente den Eigenschaften PresentValue und Unit in einem Objekt AnalogValue zuordnen. Dies ist in Abbildung 20 bei Operation 1.2 und Objekt 1.2 dargestellt. Sind wie in Abbildung 21 die Elementnamen dem Gateway unbekannt, können sie keiner Objekteigenschaften zugeordnet werden. In dem Fall werden für beide Elemente separate Objekte vom (Objekt a, Objekt b). Die Elemente werden dann durch die Objekteigenschaft PresentValue dargestellt.

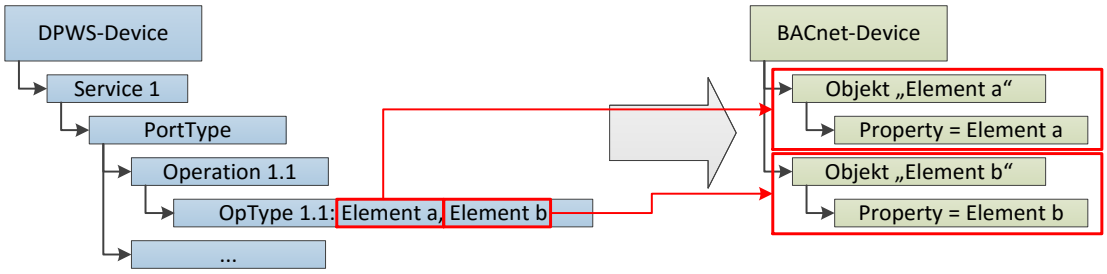


Abbildung 21: Die DPWS-BACnet-Zuordnung mit unbekannten Element-Namen

Bei den durch ein Objekt unterstützen BACnet-Diensten muss der Kompromiss eingegangen werden, dass es nur gelesen (ReadProperty), nur geschrieben (WriteProperty) oder nur abonniert (SubscribeCOV) werden kann. Es kann nicht davon ausgegangen werden, dass es in DPWS für jede Operation, die einen oder mehrere Werte verändern kann, auch eine Operation gibt, mit der genau die gleichen Werte gelesen werden können. Wären beide Operationen für den gleichen Wert vorhanden, müssten diese durch das Gateway auch als solche erkannt werden. Da Operationen beliebig bezeichnet werden können, ist dies aber nicht uneingeschränkt möglich. Ob eine Operation als ein schreibbares, lesbares oder abonnierbares Objekt dargestellt wird, hängt von den Nachrichten der Operation ab. Besitzt eine Operation eine Input-Nachricht mit Elementen, wird diese als schreibbares Objekt dargestellt. Da der BACnet-Standard festlegt, dass alle Objekte lesbar sein müssen, wird bei einer ReadProperty-Anfrage an ein solches Objekt eine Fehlermeldung gesendet. In dieser Fehlermeldung wird dem User mitgeteilt, dass der Wert nicht initialisiert ist. Eine mögliche Output-Nachricht derselben Operation wird vom Gateway verworfen. Besitzt eine Operation eine Input-Nachricht, die keine Elemente enthält, wird diese Operation als nur lesbares Objekt dargestellt. Der WriteProperty-Dienst wird von diesem Objekt nicht unterstützt. Event-Quellen werden als nur abonnierbare Objekte dargestellt, die den SubscribeCOV-Dienst unterstützen. Auch hier erfolgt bei einer ReadProperty-Anfrage eine Fehlermeldung wegen fehlender Initialisierung des Wertes.

3.3. Darstellung der Geräte innerhalb des Gateways

Bei einer direkten Umwandlung der BACnet-Darstellung in die DPWS-Darstellung können Informationen verloren gehen, die im jeweils anderen Standard nicht benötigt werden, aber dem Gateway trotzdem bekannt sein müssen, um Anfragen weiterleiten zu können. Daher wird die Überführung der Schnittstellen ineinander nicht direkt vorgenommen, sondern zuvor in eine Gateway-interne Darstellungsform gebracht. Diese Darstellungsform ist protokollunabhängig und so aufgebaut, dass sowohl die BACnet-Darstellung als auch die DPWS-Darstellung in diese überführt werden können. Zudem enthält die Gateway-interne Darstellung alle Informationen, um daraus die Geräteschnittstellen für beide Standards zu erstellen.

Die Geräte beider Standards werden innerhalb des Gateways als eine hierarchische Anordnung von *Entities* dargestellt. Dabei werden drei Formen der Entities unterschieden:

- *Root-Entity*
- *Sub-Entity*
- *Parameter-Entity*

Am Anfang jeder Anordnung steht das Root-Entity. Das Root-Entity repräsentiert das Gerät selbst. Jedes Root-Entity enthält ein oder mehrere Sub-Entities. Sub-Entities können wieder ein oder mehrere Sub-Entities enthalten. Neben weiteren Sub-Entities kann ein Sub-Entity auch ein oder mehrere Parameter-Entities besitzen. Ein Parameter-Entity repräsentiert die Teile aus den verschiedenen Standards, die einen Wert enthalten. So entstehen je nach Protokoll unterschiedlich tiefe hierarchische Anordnungen, aus denen aber für beide Standards konforme Darstellungen gebildet werden können. Die Gesamtstruktur kann im Gateway gespeichert werden und enthält alle benötigten Informationen zu einem Gerät. Abbildung 22 und Abbildung 23 veranschaulichen, wie die Geräte beider Standards im Gateway gespeichert werden.

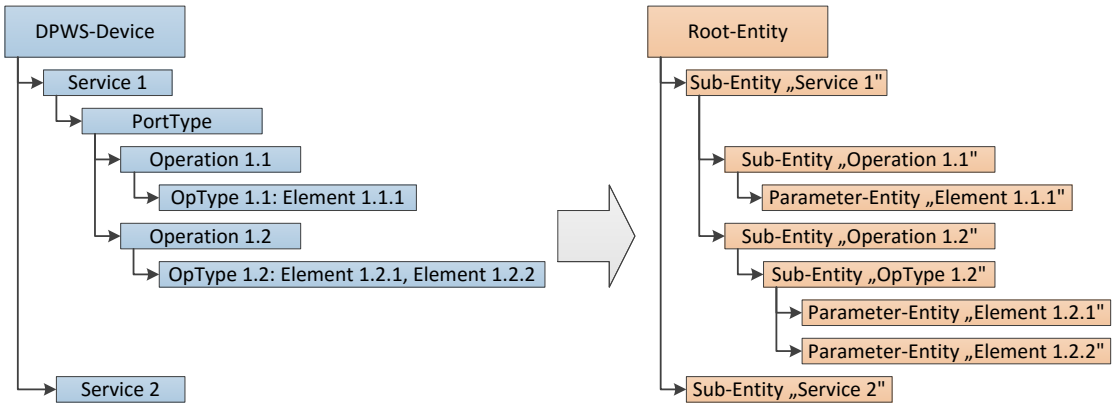


Abbildung 22: Interne Darstellung eines DPWS-Gerätes

Ein DPWS-Gerät wird auf ein Root-Entity abgebildet. Dieses Root-Entity enthält für jeden Service jeweils ein Sub-Entity. Die Sub-Entities zu den Services enthalten unmittelbar die Sub-Entities zu den Operationen. Der Port-Typ wird für die Gateway-interne Gerätdarstellung nicht verwendet, da zwei Port-Typen dieselbe Operation enthalten können. Dann wäre eine Operation doppelt in der Entity-Struktur vertreten. Der Port-Typ könnte aber verwendet werden, um die Sub-Entities, die den Operationen entsprechen, Gruppen zuzuordnen. Da die Port-Typ-Bezeichnung aber frei gewählt werden kann, kann ohne Einschränkung der Bezeichnungen keine Gruppierung vorgenommen werden. Enthält eine Operation in ihrer Nachricht ein Element von einem primitiven Typ, wird dafür direkt ein Parameter-Entity angelegt (Sub-Entity „Operation 1.1“). Ist der Element-Typ komplex, enthält erst ein weiteres

Sub-Entity in der Hierarchie die Parameter-Entities zu den Elementen (Sub-Entity „Operation 1.2“).

Die Entity-Darstellung eines BACnet-Gerätes ist eine direkte Zuweisung von Objekten zu Sub-Entities und Objekteigenschaften zu Parameter-Entities.

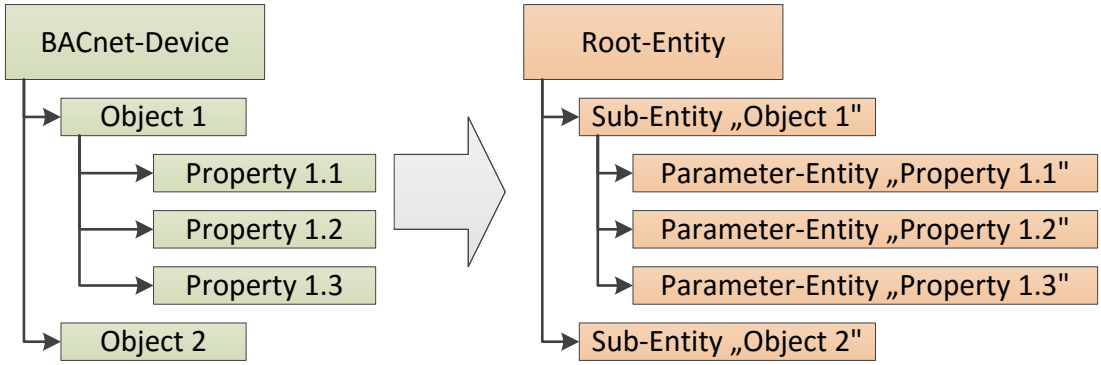


Abbildung 23: Interne Darstellung eines BACnet-Gerätes

3.4. Realisierung des BACnet-DPWS-Gateways

3.5. Verwendete Frameworks

3.5.1. OSGi

Die OSGi-Plattform wurde durch die OSGi Alliance (ehemals *Open Services Gateway initiative*) entwickelt. Sie basiert auf Java und bietet eine hardwareunabhängige Laufzeitumgebung für Softwarekomponenten. Die Grundlage der Plattform ist das OSGi-Framework. Dieses Framework organisiert die dynamische Integration der Softwarekomponenten in das Gesamtsystem und schafft die Grundlage für eine Interaktion der Komponenten miteinander. Das Framework besteht aus drei Schichten (siehe Abbildung 24) und einer optionalen Security-Schicht. Die drei nicht-optionalen Schichten sind [90]:

- Modulschicht
- Life-Cycle-Schicht
- Dienstschicht

Die Modulschicht ist die unterste Schicht des Frameworks. In ihr ist das grundlegende Konzept der OSGi-*Bundles* festgelegt. Ein Bundle ist eine Softwarekomponente, die eine geschlossene Einheit von Klassen und Ressourcen innerhalb des Frameworks darstellt. Jedes Bundle kann eigenständig im Framework hinzugefügt oder entfernt werden. Es kann erforderlich sein, dass ein Bundle Ressourcen eines anderen Bundles benötigt. In diesem Falle muss einerseits das Bundle, welches die Ressource besitzt, die Ressource explizit exportieren, andererseits muss das Bundle, welches die Ressource benötigt, diese explizit importieren. Ist

eine benötigte Ressource nicht vorhanden, kann das Bundle nicht ausgeführt werden. Welche Ressourcen ein Bundle exportiert oder importiert, entnimmt das Framework der Manifest-Datei des Bundles. Die Manifest-Datei, oder auch das Bundle-Manifest, ist neben dem funktionalen Inhalt (Attribute und Methoden von Klassen) ein weiterer Bestandteil des Bundles. Darin sind Informationen zum Bundle selbst enthalten, wie zum Beispiel Name, Version oder eben importierte und exportierte Ressourcen. Die Interaktion der Bundles untereinander erfolgt in der Dienstschicht mittels Services. Das Konzept entspricht einer SOA. Beim Start eines Bundles registriert das Bundle seine bereitgestellten Dienste bei der Service-Registry. Ein OSGi-Service ist ein beliebiges Java-Objekt. Die Registrierung erfolgt über den Klassen- oder Interface-Namen des Objekts. Andere Bundles können den Service bei der Registry erfragen und anschließend nutzen [90]. Die Rollenverteilung innerhalb des Frameworks ist in Abbildung 24 dargestellt.

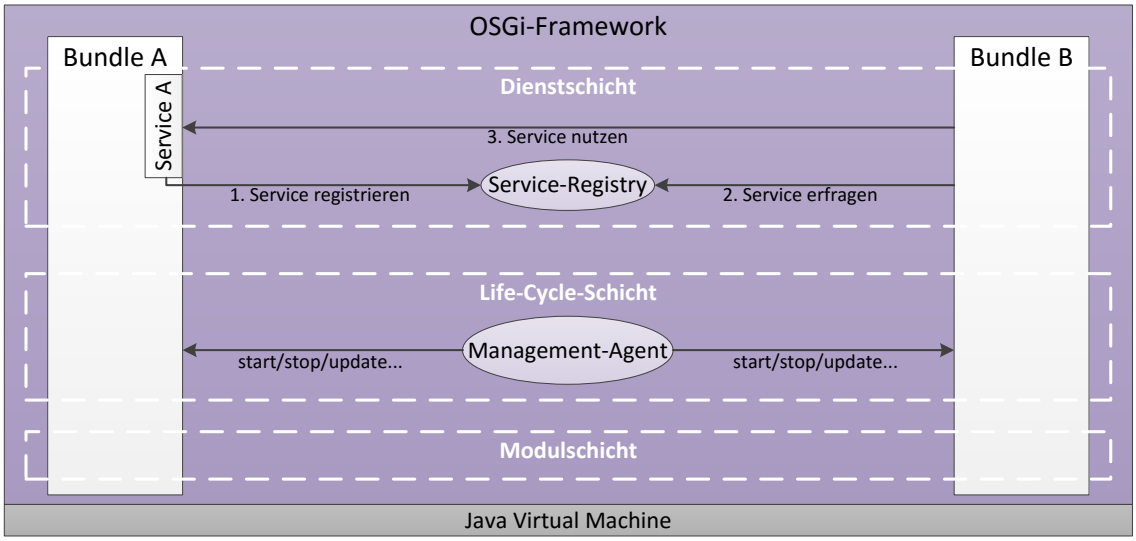


Abbildung 24: Rollen im OSGi-Framework

Bundles können innerhalb des Frameworks unterschiedliche Zustände einnehmen, welche in der Life-Cycle-Schicht festgelegt sind. Die Zustände eines Bundles können durch den *Management-Agent* verändert werden. Der Management-Agent ist Teil des Frameworks und verwaltet die Bundles. Ein Bundle kann verschiedene Zustände einnehmen, wie sie in Abbildung 25 dargestellt sind.

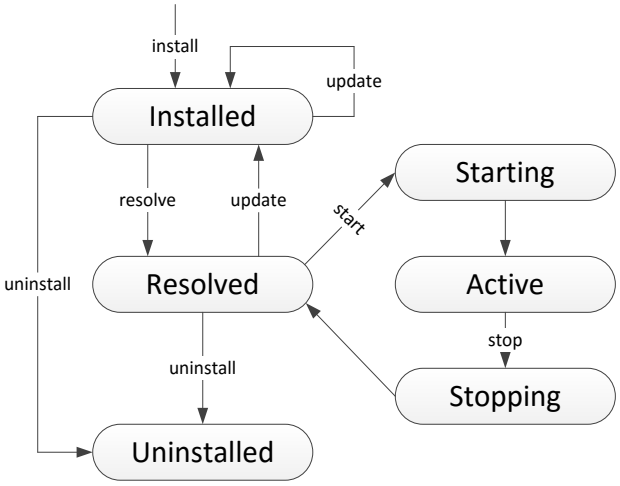


Abbildung 25: Lebenszyklus eines OSGi-Bundles [90]

Nach dem Hinzufügen eines Bundles zum Framework (*install*) befindet es sich im Zustand *Installed*. Wenn alle Ressourcen, die das Bundle benötigt, im Framework vorhanden sind, geht es in den Zustand *Resolved* über. In diesem Zustand ist das Bundle bereit gestartet zu werden. Für den Start eines Bundles wird die start()-Methode des *Bundle-Activators* aufgerufen. Der Bundle-Activator ist ein Objekt einer Klasse innerhalb des Bundles (z. B. Activator.java), die das Interface org.osgi.framework.BundleActivator implementiert. Welche Klasse das Bundle-Activator-Interface implementiert, ist in der Manifest-Datei des Bundles angegeben. Beim Installieren des Bundles wird der Bundle-Activator erzeugt. Dafür muss auf die Konstruktor-Methode öffentlich zugegriffen werden können. Das implementierte BundleActivator-Interface gibt die Methoden start() und stop() vor. In der start()-Methode können alle Variablen-Deklarationen und Funktionsaufrufe vorgenommen werden, die das Bundle zu Beginn benötigt. Mit dem Aufruf der Methode start() wechselt das Bundle in den *Starting*-Zustand, in dem die Methode abgearbeitet wird. Tritt dabei kein Fehler auf, wechselt das Bundle in den Zustand *Active*. In diesem Zustand führt das Bundle seine Funktionalität aus und kann mit anderen Bundles interagieren. Die OSGi-Plattform ermöglicht es, zur Framework-Laufzeit ein Bundle zu aktualisieren oder aus dem Framework zu entfernen. Dazu muss ein aktives Bundle mit der stop()-Methode des Bundle-Activator-Interfaces angehalten und damit in den *Resolved*-Zustand gesetzt werden. Dann kann das Bundle deinstalliert werden oder durch eine neue Version ersetzt werden [90] [91].

OSGi zeichnet sich vor allem dadurch aus, dass Services zur Laufzeit eingebunden und entfernt werden können. Die OSGi-Plattform bietet Bundles die Möglichkeit, auf ein hinzugefügtes oder entferntes Bundle zu reagieren. Dafür können Bundles einen sogenannten *Service-Tracker* besitzen. Service-Tracker reagieren darauf, wenn ein Service, also ein Objekt einer bestimmten Klasse, im OSGi-Framework hinzugefügt oder entfernt wird. Der Service-Tracker reagiert immer auf eine festgelegte Klasse oder ein Interface, das durch eine Klasse

implementiert ist. Das Bundle, welches Service-Tracker enthält, kann dann zur Laufzeit auf einen neu hinzugefügten Service reagieren [90].

3.5.2. JMEDS

Als DPWS-Framework liegt dem Gateway der *Java Multi Edition DPWS Stack*[2] (JMEDS) zugrunde. JMEDS ist eine Entwicklung von der Technischen Universität Dortmund und dem Unternehmen MATERNA Information & Communications. Der Stack bietet eine umfangreiche Unterstützung bei der Implementierung von DPWS-Clients und DPWS-Servern. Unterteilt ist der Stack in einen *Communication-Layer*, einen *Dispatching-Layer* und einen *Application-Layer*. Der Communication-Layer ist für einen Server und einen Client identisch. Er stellt technologiespezifische Kommunikations-Manager bereit. Es wird sowohl die Übertragung von SOAP-Nachrichten über HTTP als auch über UDP unterstützt. Den Inhalt der SOAP-Nachrichten erhält der Communication-Layer vom Dispatching-Layer. Der Dispatching-Layer des Clients besteht aus einem *Dispatcher*, einer *Device- und Service-Registry* und einem *Search Manager*. Der Dispatcher leitet ausgehende Nachrichten des Application-Layers an den jeweiligen Kommunikations-Manager weiter. Die Device- und Service-Registry enthält bereits bekannte Geräte und Services. Die Suche nach Geräten und Services ist Aufgabe des Search Managers. Dieser benutzt dazu den WS-Discovery-Standard im Ad hoc Mode. Der Dispatching-Layer des Servers enthält keinen Search Manager, dafür aber einen *Subscription Manager*. Dieser verwaltet die Abonnenten von Events. Die oberste Schicht von JMEDS stellt der Application-Layer dar. Darin können Client und Server implementiert werden, wofür der Stack zahlreiche Klassen zur Verfügung stellt. Für die Identifikation der Geräte verwendet JMEDS URIs. Die Schichten und Komponenten sind in Abbildung 26 dargestellt [92].

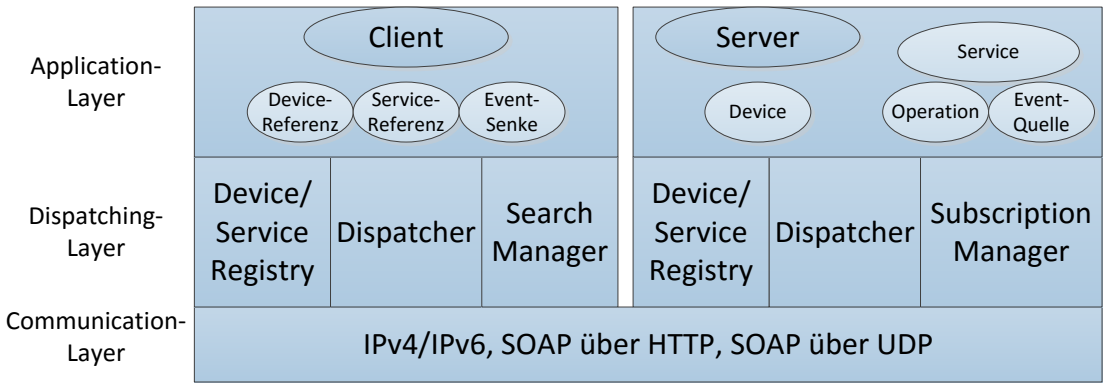


Abbildung 26: Die JMEDS-Architektur [92]

[2] Download unter http://sourceforge.net/projects/ws4d-javame/

3.5.3. BACnet4J

Zum Implementieren von BACnet-Geräten in Java gibt es den Stack *BACnet for Java*[3] (BACnet4J). Für die Kommunikation unterstützt der Stack Master-Slave/Token-Passing (MSTP) und BACnet/IP, wobei vor allem Letzteres für das Gateway im Rahmen des Projektes verwendet wurde. Ein Gerät wird über den Stack einem Netzwerk zugeordnet und wird bei der Verwendung von BACnet/IP an einen UDP-Port gebunden. BACnet4J stellt zum Implementieren von BACnet-Geräten BACnet-Objekttypen und deren Eigenschaften zur Verfügung. Um auf eintreffende IAm-Nachrichten und Meldungen reagieren zu können, kann eine Klasse, die das DeviceEventListener-Interface implementiert, dem Gerät hinzugefügt werden. Dieses Interface enthält Methoden, die beim Eintreffen einer entsprechenden Nachricht aufgerufen werden. Das Verarbeiten von Anfragen und das Melden von Wertänderungen werden vom BACnet4J-Stack übernommen. Die für das Gateway notwendigen Modifizierungen des Stacks, werden in Abschnitt 3.6.6 miterläutert.

3.6. Implementierung des BACnet-DPWS-Gateways

Das BACnet-DPWS-Gateway setzt sich im Wesentlichen aus drei OSGi-Bundles zusammen (siehe Abbildung 27). Das zentrale Bundle ist das *Gateway*-Bundle (org.unirostock.bagateway). Die Bezeichnung *bagateway* steht für *Building-Automation-Gateway*. In dem Gateway-Bundle werden Informationen zu Geräten beider Standards gespeichert und Daten weitergeleitet. Außerdem enthält es Klassen, die von den äußeren Bundles genutzt werden können beziehungsweise müssen. Die äußeren Bundles heißen *Gates*, da diese das Tor zum BACnet-Netzwerk (org.unirostock.bagateway.bacnet) beziehungsweise zum DPWS-Netzwerk (org.unirostock.bagateway.dpws) bilden. In ihnen sind die jeweiligen Stacks (BACnet4J, JMEDS) enthalten.

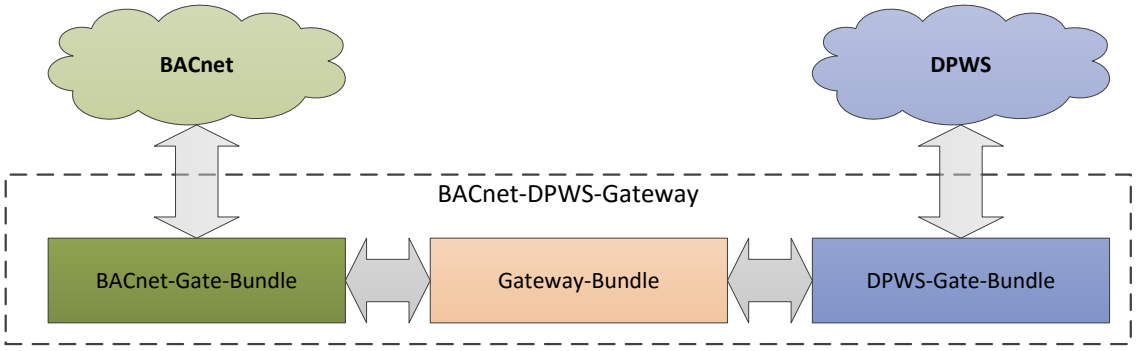


Abbildung 27: Struktur des BACnet-DPWS-Gateways

Da sowohl der JMEDS-Stack als auch der BACnet4J-Stack Klassen von externen Bibliotheken verwendet, sind diese Bibliotheken in einem zusätzlichen Bundle enthalten. Die benötigten Klassen werden von diesem Zusatz-Bundle exportiert und stehen damit im

[3] Download unter https://github.com/empeeoh/BACnet4J

Framework zur Verfügung. Ein Service wird von diesem Bundle nicht angeboten. Zudem wird von allen Bundles das Bundle org.apache.commons.logging_(Versionsnummer).jar für Aufzeichungen (Logging) benötigt.

In den folgenden Ausführungen bezeichnet der Begriff Gateway das Gesamtsystemen, bestehend aus BACnet-Gate-Bundle, Gateway-Bundle und DPWS-Gate-Bundle. Für die äußeren Bundles wird auch der Begriff Gate verwendet.

3.6.1. Das Gateway-Bundle

Das Gateway-Bundle besteht aus dem Java-Paket org.unirostock.bagateway. Darin sind neben dem Bundle-Activator (Activator.java) und der Implementierung des funktionalen Teils des Gateway-Bundles (GatewayImpl.java) drei weitere Teilpakete enthalten. Die Teilpakete werden exportiert und stehen so im OSGi-Framework für die Gate-Bundles zur Verfügung. Das Bundle hat den in Abbildung 28 dargestellten Inhalt.

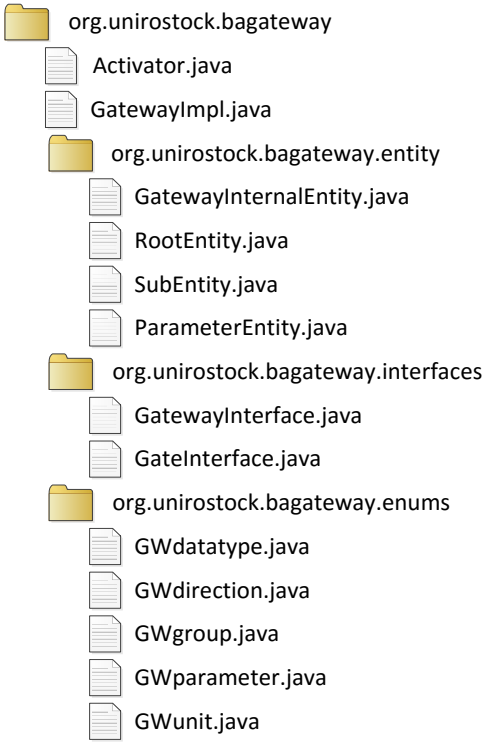


Abbildung 28: Inhalt des Gateway-Bundles

3.6.2. Umsetzung der Entities

Wie Geräte im Gateway dargestellt werden, ist bereits in Abschnitt 3.3 beschrieben. Die Klassen für die drei Entity-Typen werden von der gemeinsamen Klasse GatewayInternalEntity abgeleitet (vgl. Abbildung 29).

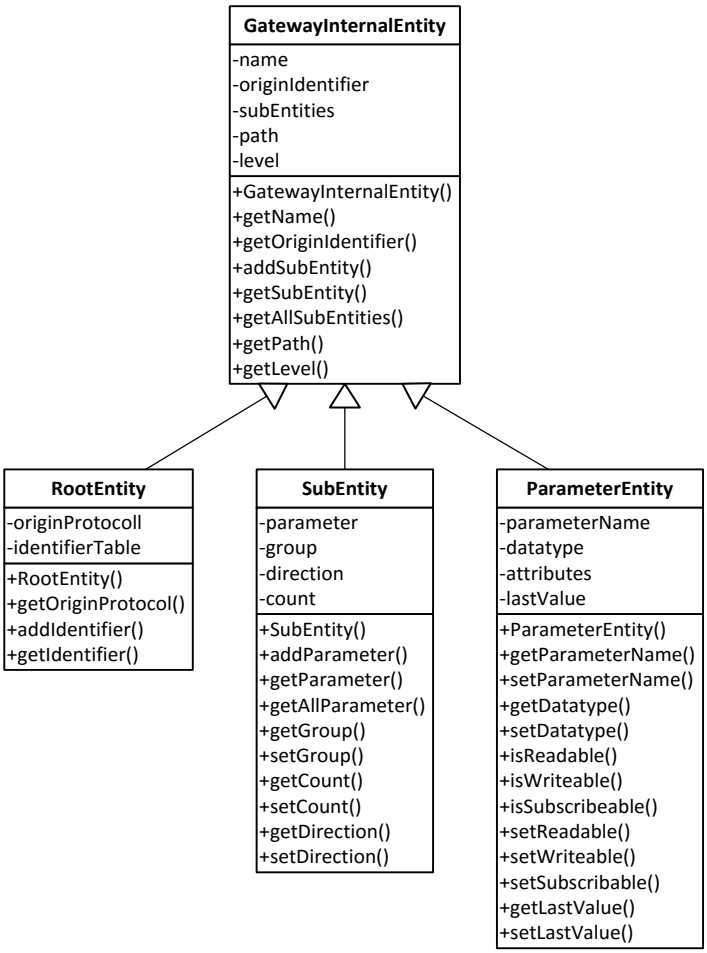


Abbildung 29: Die Entity-Klassen

Das Gateway-Internal-Entity selbst wird in der internen Gerätestruktur nicht verwendet, sondern enthält nur Attribute und Methoden, die alle Entity-Typen gemeinsam haben. Jedes Entity besitzt einen Namen in Form einer Zeichenkette (CharacterString), der im *name*-Attribut steht. Der Name wird beim Erzeugen eines Entity-Objekts gesetzt und steht von da an fest. Auslesen lässt sich das name-Attribut mit der Methode *getName()*. Neben dem Namen wird auch der *originIdentifier* beim Erzeugen eines Entities festgelegt. Danach kann dieser nicht mehr geändert werden. Der originIdentifier enthält den protokollabhängigen Bezeichner. Das Gate für den Standard, von dem das Entity stammt, kann diesen Bezeichner interpretieren und zur Identifizierung des Entities nutzen. So kann das Gate das Entity einer Komponente seines Standards zuordnen. Gates für einen anderen Standard können den Bezeichner als Ganzes verwenden. Für diese Gates ist der originIdentifier eine Blackbox. Er kann zwar zur Identifikation des Entities genutzt werden, eine Interpretation durch das Gate ist aber nicht möglich. Als Beispiel soll hier der ObjectIdentifier von BACnet dienen. Der ObjectIdentifier, der sich aus BACnet-Objekttyp und Instanznummer zusammensetzt, wird einem Entity als originIdentifier zugewiesen. Das BACnet-Gate kann damit dem originIdentifier den Objekttyp und die Instanznummer entnehmen. Das DPWS-Gate kann den originIdentifier als

Gesamtkonstrukt verwenden. Die Bedeutung dahinter ist dem DPWS-Gate aber unbekannt. Über die Methode *getOriginIdentifier()* erhält man den Inhalt des originIdentifier-Attributs.

Die Sub-Entities, die ein Entity besitzen kann, sind in der Liste *subEntities* zusammengefasst. Damit befinden sich die Sub-Entities auf der nächsten Ebene der Hierarchie. Zum Hinzufügen eines neuen Sub-Entity gibt es die Methode *addSubEntity()*. Um ein einzelnes Sub-Entity zu bekommen, gibt es die Methode *getSubEntity()*. Mit *getAllSubentities()* erhält man die vollständige Liste. Durch die Schachtelung mehrerer Entities entstehen in der Hierarchie Pfade. Ausgehend vom Root-Entity lassen sich über diese Pfade die Entities in der Hierarchie finden. Der Pfad eines Entities steht im *path*-Attribut. Das ist eine Liste, deren Einträge die originIdentifier-Attribute von allen Entities sind, die in der Hierarchie über diesem Entity stehen. Das letzte Element in der Liste ist der eigene originIdentifier. Mit der Methode *getPath()* kann dieses Attribut abgefragt werden. Jedes Entity steht auf einer bestimmten Ebene in der Hierarchie. Die Ebene des Entities wird durch den Integer-Wert in dem *level*-Attribut dargestellt. Mit *getLevel()* kann man die Hierarchie-Ebene des Entities abfragen. Root-Entities stehen auf Ebene 0 in der Hierarchie. Abbildung 30 zeigt eine Beispielanordnung von Entities mit den gerade vorgestellten Attributen.

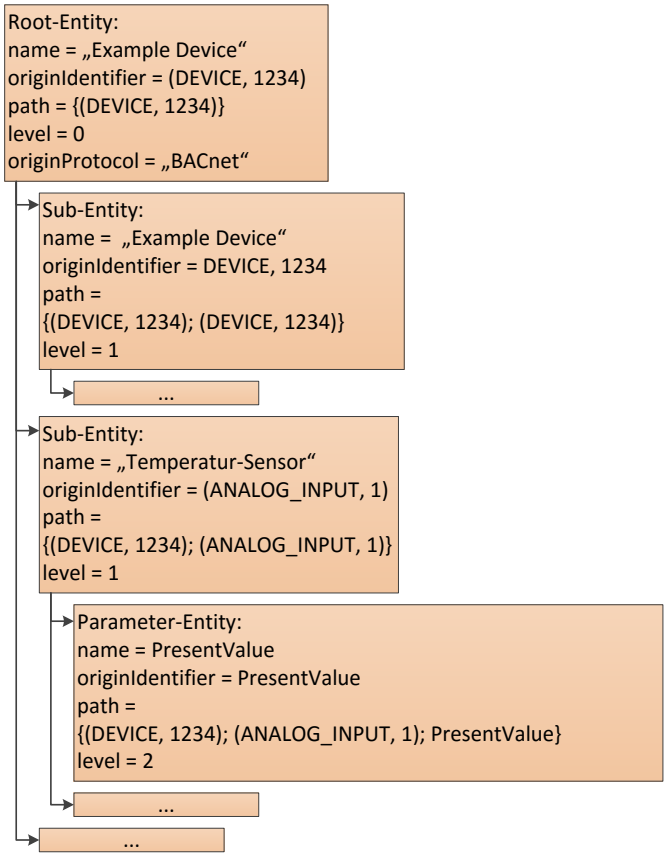


Abbildung 30: Beispielanordnung von Entities mit Attributen I

Das Root-Entity besitzt noch zwei ihm eigene Attribute und die dazugehörigen Methoden. Das *originProtocol*-Attribut gibt an, welches Kommunikationsprotokoll das Gerät verwendet. Damit kann es einem Gate eindeutig zugeordnet werden. Das Attribut wird beim Erzeugen des Objekts festgelegt und kann mit der Methode *getOriginProtocol()* abgefragt werden. Ein Gate für ein bestimmtes Protokoll, das nicht das originProtocol ist, bildet das Gerät auf eine für seinen Standard konforme Darstellung ab. Dabei wird dem Gerät eine Gerätekennung dieses Standards zugewiesen. Damit erhält zum Beispiel jedes BACnet-Gerät einen URI. Diese Kennung wird in dem *identifierTable*-Attribut abgespeichert. Dieses Attribut ist eine Map, die alle Gerätekennungen enthält, die dem Gerät von den Gates zugewiesen wurden. Als Schlüssel eines Eintrags wird der Name des entsprechenden Protokolls verwendet („BACnet“, „DPWS“). Mit der Methode *addIdentifier()*, der beim Aufruf der Name des Protokolls und die Gerätekennung übergeben wird, kann dem Gerät eine Gerätekennung zugewiesen werden. Mit der Methode *getIdentifier()* und der Protokollbezeichnung als Parameter kann ein Gate die zuvor zugewiesene Gerätekennung abfragen.

Jedes Root-Entity sollte mindestens ein Sub-Entity besitzen, da nur ein Sub-Entity eine Liste von Parameter-Entities enthalten kann. Parameter-Entities bilden den Abschluss eines Pfades in der Hierarchie und repräsentieren die Teile aus einem Protokoll, die einen Wert enthalten. Die Sub-Entities besitzen eine *parameter*-Liste, in der die Parameter-Entities zusammengefasst sind. Die Methoden *addParameter()*, *getParameter()* und *getAllParameters()* sind äquivalent zu den Methoden addSubEntity(), getSubEntity() und getAllSubEntities(). Um die Sub-Entities von Objekten wie in Abschnitt 3.2 beschrieben Gruppen zuordnen zu können, gibt es das *group*-Attribut. Das group-Attribut ist ein Objekt der Gateway-eigenen Enumeration *GWgroup*. Es bietet die Möglichkeit das Sub-Entity und die eventuell enthaltenen Parameter-Entities einer Gruppe zuzuordnen, um anderen Gates eine bessere Interpretation der Daten zu ermöglichen. Mit den Methoden *setGroup()* und *getGroup()* kann das Attribut gesetzt beziehungsweise gelesen werden. Wird zum Beispiel ein Temperatursensor durch das Sub-Entity dargestellt, kann es der Gruppe „TEMPERATURE“ zugeordnet werden. Ein Schalter, der die zwei Zustände Ein und Aus besitzt, würde zur Gruppe „BINARY“ gehören. Welche Gruppen das Gateway bisher anbietet, sind in Anhang A.1 angegeben. Ebenfalls der besseren Interpretation der Daten dient das *direction*-Attribut. Es gibt Auskunft darüber, ob es sich bei dem Sub-Entity um einen Eingang (Input), Ausgang (Output) oder einen Wert (Value) handelt. Eingänge können zum Beispiel Sensoren sein, Ausgänge können Aktoren sein. Werte sind zum Beispiel Sub-Entities, die einen Sollwert enthalten. Das direction-Attribut ist ein Objekt der Enumeration *GWdirection*. Welche Werte GWdirection anbietet steht in Anhang A.1. Mit den Methoden *setDirection()* und *getDirection()* lässt sich das direction-Attribut setzen beziehungsweise abfragen. Enthält ein Entity mehrere Sub-Entities mit gleichem Inhalt des direction-Attributes, können diese mit dem count-Attribut nummeriert werden. Besitzt ein Gerät (Root-Entity) beispielsweise zwei

Sensoren als Eingänge, hätte das Root-Entity ein Sub-Entity mit direction = „INPUT“ und count = 1 und ein weiteres Sub-Entity mit direction = „INPUT“ und count = 2. Dadurch lassen sich beide Eingänge voneinander unterscheiden. Mit den Methoden *setCount()* und *getCount()* lässt sich das count-Attribut setzen beziehungsweise abfragen. Ein Beispiel zu diesen Attributen zeigt Abbildung 31.

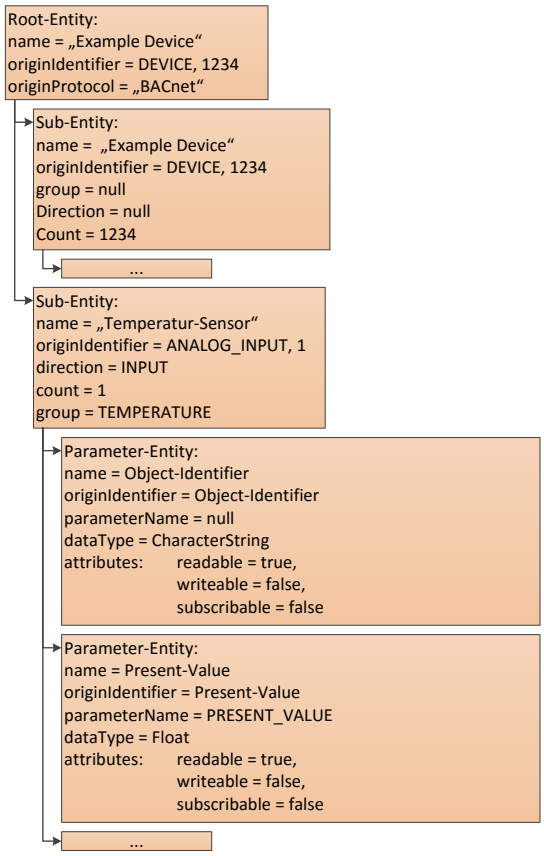


Abbildung 31: Beispielanordnung von Entities mit Attributen II

Auch die dritte Form der Entities, das Parameter-Entity, hat ihm eigene Attribute, die in Abbildung 31 dargestellt sind. Das Attribut *parameterName* enthält einen Wert der Enumeration *GWparameter* (Anhang A.1). Da die Bezeichnung von Parametern damit im Gateway vereinheitlicht wird, ist eine bessere Interpretation der Daten für die Gates möglich. ParameterName wird zum Beispiel genutzt, um Elemente aus einem DPWS-Gerät einer BACnet-Objekteigenschaft zu zuordnen. Mit den Methoden *setParameterName()* und *getParameterName()* lässt sich das parameterName-Attribut setzen beziehungsweise abfragen. Das *datatype*-Attribut ist besonders wichtig für richtige Dateninterpretation. Nur damit kann dem Wert hinter einem Parameter-Entity ein Datentyp zugeordnet werden. Das datatype-Attribut enthält einen Wert der Enumeration *GWdatatype*, welche einige Datentypen vorgibt (Anhang A.1). Mit den Methoden *setDatatype()* und *getDatatype()* lässt sich das datatype-Attribut setzen beziehungsweise abfragen. Um anzugeben, ob ein Wert lesbar,

schreibbar und/oder abonnierbar ist, gibt es das *attribute*-Attribut. Mit den Methoden *setReadable(), setWriteable()* und *setSubscribable()* können diese Eigenschaften im Parameter-Entity gesetzt werden. Die Methoden *isReadable(), isWriteable()* und *isSubscribable()* dienen dem Abfragen der Eigenschaften. Wenn ein Wert eines Parameter-Entities gelesen oder geschrieben wird, gibt es die Möglichkeit diesen im Attribut *lastValue* zu speichern. Mit der Methode *setLastValue()* lässt sich der Wert setzen. Mit der Methode *getLastValue()* kann der Wert abgefragt werden.

Die Gates müssen in der Lage sein, ein Gerät von ihrem Protokoll in diese Gateway-interne Struktur zu überführen. Außerdem ist es erforderlich, dass die Gates aus einer Anordnung von Entities eine Geräteschnittstelle erstellen, die zu ihrem Standard konform ist. Es sollte dabei von einer Hierarchie beliebiger Tiefe ausgegangen werden. Es kann vorkommen, dass nicht immer alle Attribute einen Wert enthalten. Dies kann vor allem bei Attributen des Sub-Entity und Parameter-Entity der Fall sein, die die Interpretation der Daten erleichtern sollen (z. B. group, datatype, parameterName). Für diesen Fall sollte ein Gate entweder ein anderes Attribut oder einen selbst festgelegten Default-Wert verwenden. Die gesamte Struktur eines Gerätes aus Entities wird im Gateway-Bundle gespeichert und verwaltet.

3.6.3. Die Bundle-Schnittstellen

Im Gateway-Bundle sind im Teilpaket org.unirostock.bagateway.interfaces Schnittstellen definiert, die der Interaktion des Gateway-Bundles mit den Gate-Bundles dienen. Mit GatewayInterface gibt es eine Schnittstellenbeschreibung für das Gateway-Bundle und mit GateInterface eine Schnittstellenbeschreibung für die Gates. Welche Methoden die Schnittstellen enthalten, ist in Abbildung 32 dargestellt.

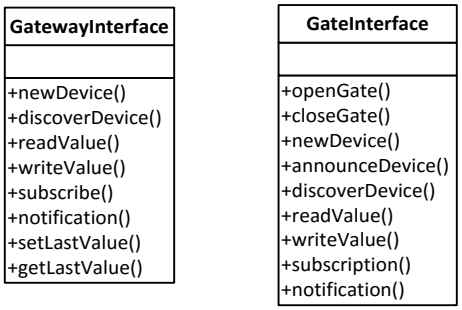


Abbildung 32: Die Bundle-Schnittstellen

Diese Schnittstellen sind die im OSGi-Framework bereitgestellten Services. Das Gateway-Bundle, das das Gateway-Interface implementiert, stellt den GatewayInterface-Service zur Verfügung und die Gate-Bundles, die das Gate-Interface implementieren, stellen jeweils einen GateInterface-Service bereit. Damit wird sichergestellt, dass das Gateway erkennt, welche Methoden von den Gates bereitgestellt werden. Die Gates haben Kenntnis darüber, welche

Methoden das Gateway anbietet. Abbildung 33 zeigt, wo die Schnittstellen im Gateway angeordnet sind. Zum Verständnis der Abbildung sei erwähnt, dass das Gateway-Bundle nur ein Gateway-Interface besitzt, auf das aber beide Gates zugreifen. Wie die Methoden des Gate-Interfaces implementiert werden, ist protokollabhängig. Die Gate-Interfaces selbst sind aber unabhängig vom Kommunikationsprotokoll. Die protokollunabhängigen Bundle-Schnittstellen, die protokollneutrale interne Gerätdarstellung in Form von Entities und die Möglichkeit OSGi-Bundles dynamisch kombinieren zu können, bilden die Grundlage, um das Gateway später um Gates weiterer Kommunikationsstandards zu erweitern.

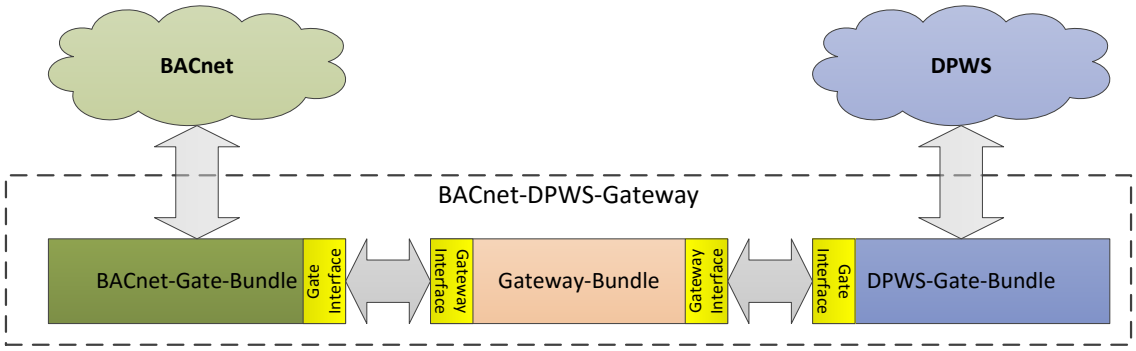


Abbildung 33: Platzierung der Interfaces im Gateway

Meldet sich ein Gerät im Netzwerk eines Gates an, erstellt das Gate aus den Geräteinformationen eine Entity-Struktur und übergibt diese mit der Methode *newDevice()* des Gateway-Interfaces dem Gateway-Bundle. Das Gateway-Bundle hat zum einen die Aufgabe die Struktur und damit alle Informationen zu dem Gerät zu speichern und zum anderen die Struktur an das andere Gate weiterzuleiten. Ist das Gerät für das Gateway neu, verwendet das Gateway-Bundle zum Weiterleiten die newDevice()-Methode des Gate-Interfaces. Das Gate erstellt aus der Entity-Struktur eine seinem Standard entsprechende Schnittstellendarstellung und macht das Gerät in seinem Netzwerk bekannt. Ist das Gerät dem Gateway bekannt, das heißt, es ist bereits im Gateway-Bundle gespeichert, ist es nicht erforderlich, dass ein Gate die Entity-Struktur erneut umwandelt. In diesem Fall ruft das Gateway-Bundle die Methode *announceDevice()* auf. Daraufhin wird nur eine neue Bekanntmachung des schon existierenden Gerätes in das Netzwerk gesendet. Erhält ein Gate aus seinem Netzwerk eine Suchanfrage, leitet es diese Anfrage mit *discoverDevice()* an das Gateway-Bundle weiter. Als Übergabeparameter ist das Protokoll, von dem die Anfrage kommt, anzugeben. Daran erkennt das Gateway, an welches Gate es die Anfrage nicht weiterleiten soll. Die Aufgabe des Gateway-Bundles ist es, die Anfrage nur an das andere Gate zu übergeben. Als zusätzlicher Parameter kann ein Name in Form einer Zeichenkette als Suchparameter übergeben werden. Mit den gerade vorgestellten Methoden ist es möglich, Geräte über Standardgrenzen hinweg zu suchen und bekannt zu machen.

Der Austausch von Nutzdaten zwischen den Bundles erfolgt mit den folgenden Methoden der Interfaces. Wenn bei einem Gate eine Lese-Anfrage aus seinem Netzwerk eintrifft, gibt es diese über die *readValue()*-Methode an das Gateway-Bundle weiter. Beim Aufruf der Methode werden zwei Parameter übergeben. Der erste Parameter ist das path-Attribut des Sub-Entities, welches die zu lesenden Parameter-Entities enthält. Der zweite Übergabeparameter der Methode ist eine Map, in der die originIdentifier-Attribute der zu lesenden Parameter stehen. Die originIdentifier bilden den Schlüssel für die Mapeinträge. Das Gateway-Bundle sucht über das erste Element des übergebenen path-Attributs aus den gespeicherten Geräten das zum Sub-Entity gehörende Root-Entity heraus. Dieses übergibt das Gateway-Bundle zusammen mit dem path-Attribut und der Map mit den Parametern an das Gate, welches mit dem Ziel-Gerät kommunizieren kann. Dafür ruft das Gateway-Bundle die readValue()-Methode des Gate-Interfaces auf. Dieses Gate hat nun die Aufgabe, die Werte zu den Parametern vom Zielgerät zu erfragen und in die Map einzutragen. Anschließend gibt es die Map über das Gateway-Bundle zurück und das Gate, welches die Lese-Anfrage erhalten hat. Dann erfolgt das Senden der Antwort auf die ursprüngliche Lese-Anfrage. Abbildung 34 stellt diesen Ablauf dar.

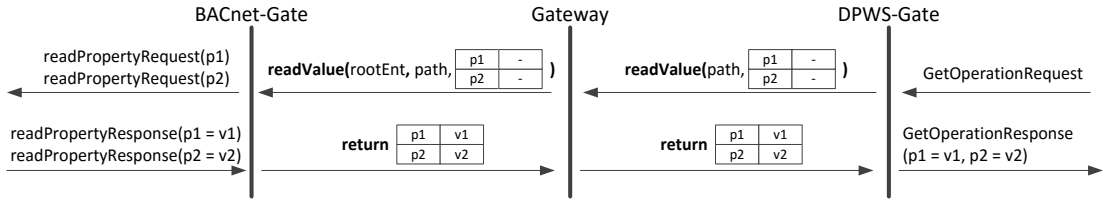


Abbildung 34: Verarbeitung einer Lese-Anfrage im Gateway

Auf diese Weise kann auch mehr als ein Wert mit einer Anfrage gelesen werden. Die Werte werden innerhalb des Gateway als Zeichenkette dargestellt. Es ist Aufgabe der Gates, diese in das für sie erforderliche Datenformat umzuwandeln. Die Verarbeitung einer Schreib-Anfrage mit den *writeValue()*-Methoden der Interfaces verläuft im Gateway ähnlich zu den readValue()-Methoden. Der Unterschied ist, dass vom Gate, welches die Anfrage erhält, die Map mit den Parametern und den zu schreibenden Werten übergeben wird. Außerdem hat die writeValue()-Methode keinen Rückgabewert. Erhält ein Gate eine Anfrage aus seinem Netzwerk zum Abonnieren von Ereignissen oder Wertänderungen eines Gerätes aus dem anderen Netzwerk, übergibt es die Anfrage mit der *subscription()*-Methode an das Gateway-Bundle. Beim Aufruf der Methode wird der Pfad des Sub-Entities übergeben sowie eine Zeitangabe in Millisekunden, wie lange das Abonnement andauern soll. Sowohl der BACnet-Standard als auch der DPWS-Standard bieten eine zeitliche Begrenzung der Abonnements. Das Gateway-Bundle leitet die Anfrage an das entsprechende Gate weiter. Trifft eine Meldung zu einem Abonnement in einem Gate ein, wird die Information zusammen mit dem Pfad des Sub-Entities mit der Methode *notification()* an das Gateway-Bundle übergeben. Die gemeldeten Parameter sowie deren Werte werden auch hier in einer Map übergeben. Das

Gateway-Bundle leitet die Meldung an das andere Gate weiter, welches daraus eine Meldung erstellt. Mit den Methoden readValue(), writeValue(), subscription() und notification() ist es dem Gateway möglich, Daten zwischen den verschiedenen Standards zu übertragen.

Das GatewayInterface stellt noch zwei Methoden bereit, um die zuletzt gelesenen und geschriebenen Werte in den Parameter-Entities zu speichern und auszulesen. Zum Speichern eines Wertes wird der Methode *setLastValue()* das path-Attribut des Parameter-Entities übergeben und der zu schreibende Wert als Zeichenkette. Wird der Methode *getLastValue()* der Pfad des gewünschten Parameter-Entities übergeben, wird der Wert zurückgeliefert. Der zuletzt bekannte Wert wird in folgendem Szenario benötigt. Angenommen die Input-Nachricht einer Operation enthält einen komplexen Typ, in dem mehrere Elemente zwingend vorkommen müssen. Das BACnet-Gate bildet diese Operation auf ein Objekt ab, in dem die Elemente verschiedenen Objekteigenschaften zugeordnet werden. Wenn nun aus dem BACnet-Netzwerk versucht wird, den writeProperty-Dienst auf eine dieser Eigenschaften anzuwenden, hat das Gate keine Möglichkeit dem User zu sagen, dass auch eine andere Eigenschaft geschrieben werden muss. Das BACnet-Gate leitet die Schreib-Anfrage über das Gateway-Bundle an das DPWS-Gate weiter. Das DPWS-Gate erhält den Wert und weist diesen dem gewünschten Element zu. Die anderen Elemente, die einen Wert haben müssen, werden mit dem zuletzt bekannten Wert belegt. Daher ist es erforderlich bei jeder Lese- und Schreib-Anfrage die Werte in den Parameter-Entities im Gateway-Bundle über die Methode setLastValue() zu speichern. Mit der Methode getLastValue() kann das DPWS-Gate sich die Werte holen und die Input-Nachricht der Operation damit vervollständigen.

Das Gate-Interface bietet zusätzlich die Methoden *openGate()* und *closeGate()* an. Die Methode openGate() veranlasst das Gate, sich in einen Zustand zu versetzen, in dem es fähig ist, Informationen auszutauschen und sich in seinem Netzwerk wenn möglich anzumelden. Die Methode closeGate() veranlasst das Gate den Bereitschaftszustand zu verlassen und sich wenn möglich aus seinem Netzwerk abzumelden.

Im Bundle-Activator wird beim Starten des Gateway-Bundles ein Service-Tracker für Klassen angelegt, die das Gate-Interface implementieren. Wird ein Gate dem OSGi-Framework hinzugefügt, kann das Gateway sofort darauf reagieren. So können beispielsweise mit der discoverDevice()-Methode die verfügbaren Geräte des neuen Protokolls herausgefunden werden.

3.6.4. Die Funktionalität des Gateway-Bundles

Die Funktionalität des Gateway-Bundles befindet sich in der Klasse GatewayImpl (siehe Abbildung 35). In der start()-Methode des Bundle-Activators wird ein Objekt dieser Klasse erzeugt, die das GatewayInterface implementiert. In GatewayImpl gibt es für das BACnet-Gate und das DPWS-Gate jeweils ein Attribut vom Typ GateInterface (bacnetGate,

dpwsGate), über das die Methoden beider Gates genutzt werden können. Voraussetzung dafür ist, dass die Gates im Framework installiert sind und gestartet wurden. Nur dann werden über den Service-Tracker die Attribute mit den Gate-Objekten belegt. Des Weiteren enthält die Klasse die Map entityList. In dieser Map werden alle Geräte in Form der Entity-Hierarchie, mit dem Root-Entity als oberstes Element, gespeichert. Als Schlüssel eines Mapeintrags wird der originIdentifier des Root-Entities verwendet. Die Methoden von GatewayImpl sind die des GatewayInterface und wurden in Abschnitt 3.6.3 beschrieben. Die Hauptaufgabe der Methoden ist die Vermittlung von Anfragen und Nachrichten zwischen den Gates.

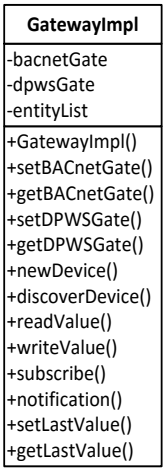


Abbildung 35: Die Klasse GatewayImpl

3.6.5. Das DPWS-Gate-Bundle

Das DPWS-Gate-Bundle ermöglicht dem Gateway die Kommunikation mit DPWS-Geräten im Netzwerk. Das Bundle setzt das Vorhandensein des Gateway-Bundles und des Bundles mit den externen Bibliotheken im Framework voraus. Ist eines dieser Bundles nicht installiert, können die Abhängigkeitsbeziehungen des DPWS-Gate-Bundles aus der Manifest-Datei nicht aufgelöst werden und das Bundle kann nicht in den Zustand Resolved gesetzt werden. Es werden vom DPWS-Gate-Bundle keine Ressourcen exportiert. Das Gate implementiert die Methoden des GateInterface in der Klasse DPWSGateImpl. Beim Start erfährt der Service-Tracker von dem neu bereitgestellten GateInterface-Service und kann ihn über den Bundle-Namen dem Attribut dpwsGate in GatewayImpl zuordnen. Das DPWS-Gate-Bundle enthält die in Abbildung 36 dargestellten Pakete und Klassen.

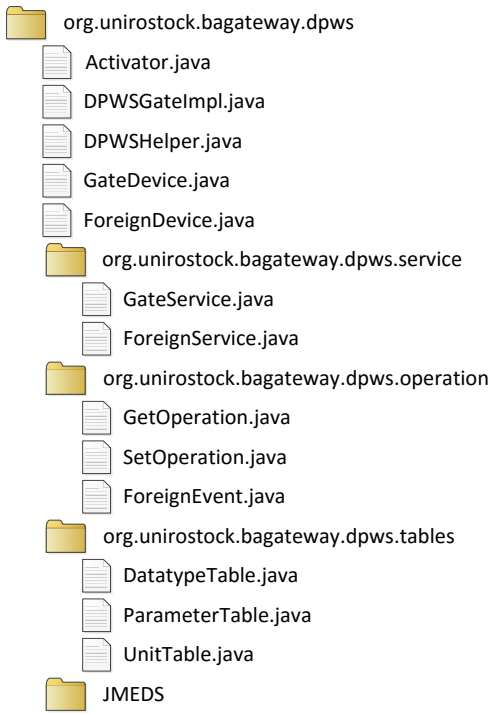


Abbildung 36: Inhalt des DPWS-Gate-Bundles

Als DPWS-Framework kommt JMEDS (Version 2 Beta 10) zum Einsatz. Von der von JMEDS bereitgestellten Klasse DefaultDevice werden für das Gate die Klassen GateDevice und ForeignDevice abgeleitet. Ein Objekt der Klasse GateDevice repräsentiert das Gateway im DPWS-Netzwerk. Das GateDevice stellt den GateService als Web Service bereit, worin Operationen definiert werden können, die das Gateway selbst betreffen. Objekte der Klasse ForeignDevice werden erzeugt, wenn das Gate vom Gateway-Bundle eine neue Entity-Struktur mit newDevice() übergeben bekommt. Ein ForeignDevice-Objekt stellt demnach die DPWS-Schnittstelle eines protokollfremden Geräts dar. Ein Foreign-Device bietet als Web Service den Foreign-Service an, welcher unterschiedliche Port-Typen enthalten kann. In den Port-Typen können GetOperation, SetOperation oder ForeignEvent als Lese-Operation, Schreib-Operation oder zum Abonnement enthalten sein. Dies wird später in diesem Abschnitt erläutert. Des Weiteren enthält das DPWS-Gate in der Klasse DPWSGateImpl einen Client (GateClient), welcher von der JMEDS-Klasse DefaultClient abgeleitet ist. Über diesen Client kann das Gate auf Hello-Nachrichten und Event-Meldungen reagieren. Zudem werden die Lese-, Schreib- und Abonnement-Anfragen aus dem Gateway über den Client an DPWS-Geräte gesendet. Nachrichten, auf die der Gate-Client nicht reagiert, werden vom *DPWSMessageListener* verarbeitet. Zu diesen Nachrichten gehören Probe- und Probe-Match-Nachrichten. Abbildung 37 zeigt den Aufbau des DPWS-Gates.

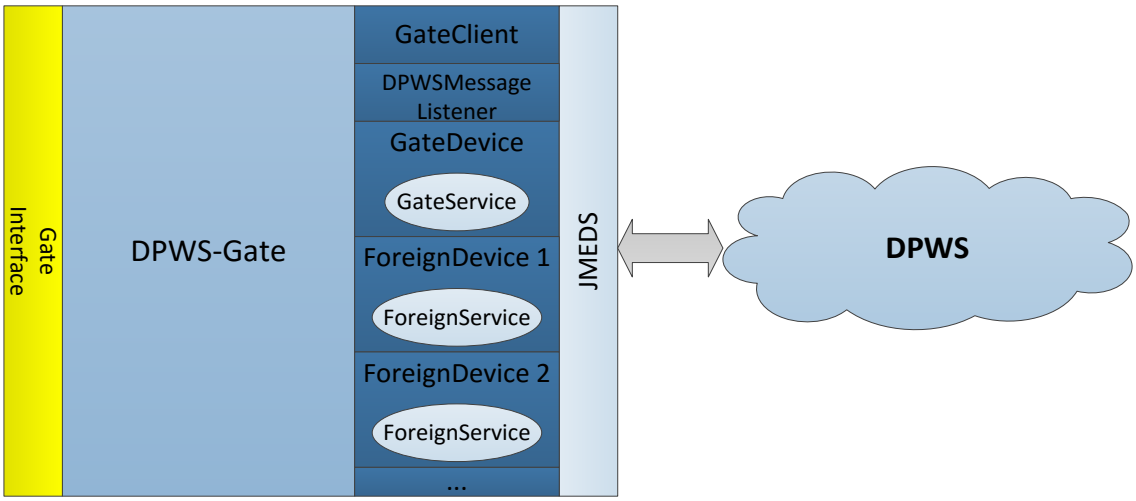


Abbildung 37: Aufbau des DPWS-Gates

Im Konstruktor des GatewayImpl wird als erstes das JMEDS-Framework gestartet und das Gate-Device mit dem Gate-Service angelegt. Operationen sind für diesen Service nicht implementiert. Es ist aber möglich, Operationen zur Konfiguration des Gates beziehungsweise des Gateways darin bereitzustellen. Neben dem Gate-Device werden Instanzen des Gate-Clients und des DPWS-Message-Listeners angelegt. An dieser Stelle sind alle Komponenten, die das DPWS-Gate benötigt, angelegt. Es kann aber noch keine Kommunikation mit dem DPWS-Netzwerk stattfinden. Erst mit dem Aufruf der openGate()-Methode durch das Gateway-Bundle werden das Gate-Device, der Gate-Client und der Message-Listener gestartet. Mit dem Start des Gate-Devices wird zugleich eine Hello-Nachricht gesendet, die das Gateway im DPWS-Netzwerk bekannt macht. Jetzt ist es dem Gate möglich DPWS-Nachrichten zu empfangen und zu senden.

Wenn der Gate-Client eine Hello-Nachricht empfängt, wird als erstes geprüft, ob die Nachricht von dem Gate-Device oder einem Foreign-Device gesendet worden ist. In diesen Fällen wird nicht weiter auf die Nachricht reagiert. Kommt die Hello-Nachricht von einem anderen DPWS-Gerät, erstellt das Gate aus den erhaltenen Informationen eine Entity-Struktur, wie sie in Abbildung 38 dargestellt ist.

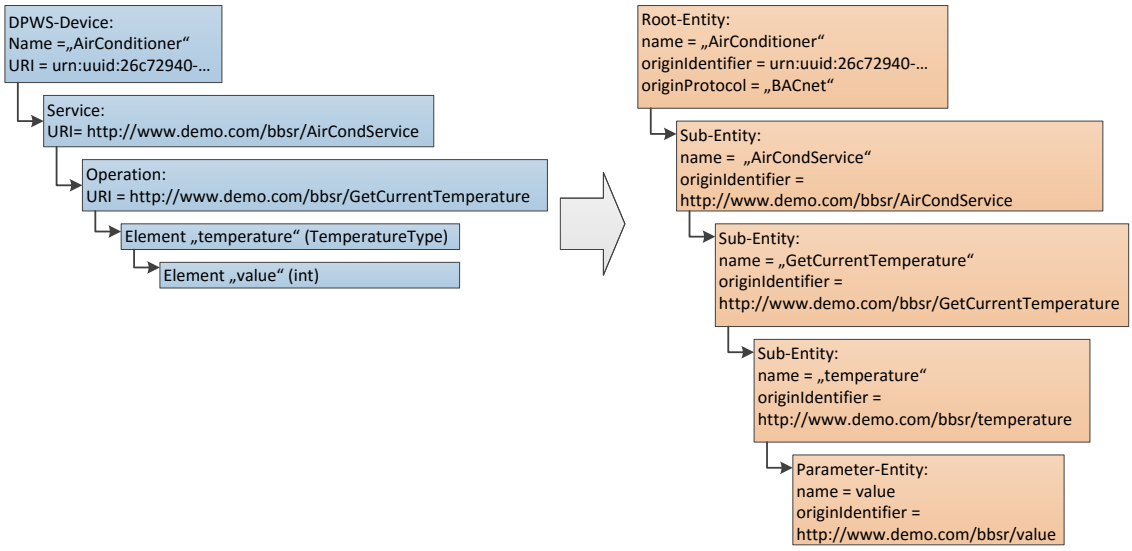


Abbildung 38: Beispiel einer Entity-Struktur aus einem DPWS-Gerät

Dies beginnt mit dem Anlegen eines Root-Entity für das DPWS-Gerät. Als originIdentifier wird der URI aus der Endpunkt-Referenz des Gerätes genommen. Als Name dient der Inhalt des Attributs *Friendly-Name*. Dem Root-Entity wird anschließend für den ersten Service, den das Gerät anbietet, ein Sub-Entity hinzugefügt. Der originIdentifier für dieses Entity ist der URI des Services. Im nächsten Schritt erhält dieses Sub-Entity weitere Sub-Entities für jede Operation, die der Service anbietet. Jede Operation wird dahingehend geprüft, ob sie eine Input-Nachricht mit oder ohne Elementen besitzt. Enthält die Nachricht ein Element von einem primitiven Typ, erhält das Sub-Entity über die Methode setWriteable(true) die Eigenschaft, dass es schreibbar ist und es wird ein Parameter-Entity für das Element hinzugefügt. Bei einem komplexen Typ wird zunächst ein weiteres Sub-Entity angelegt, welches dann schreibbar gemacht wird und die Parameter-Entities enthält. Enthält die Operation eine Input-Nachricht ohne Elemente, wird für die Output-Nachricht ein lesbares Sub-Entity angelegt. Mit Elementen von primitiven und komplexen Typen sowie weiteren Operationen wird dabei identisch verfahren. Wenn Sub-Entities für alle Operationen angelegt wurden, werden für alle Ereignis-Quellen Sub-Entities angelegt. Das Erzeugen der Entities für die Elemente findet hier ebenfalls statt. Allerdings werden die Entities der Event-Quellen mit setSubscribable(true) nur abonnierbar gemacht. Beim Anlegen der Parameter-Entities spielt der Name des Elements eine entscheidende Rolle. Die Map dpws2gw aus der Klasse ParameterTable ordnet bestimmten Namen Parameterbezeichnungen aus der GWparameter-Enumeration zu (siehe Anhang A.1). Kann ein Element-Name einer Parameterbezeichnung zugeordnet werden, wird das parametername-Attribut in dem Parameter-Entity mit dieser Bezeichnung belegt. Genauso wird mit den Datentypen vorgegangen. In der Klasse DatatypeTable werden XML-Schema-Datentypen auf Bezeichnungen der GWdatatype-Enumeration abgebildet (siehe Anhang A.1). Wurden alle Sub-Entities für alle Operationen und Event-Quellen sowie deren Elemente erzeugt, übergibt das Gate die gesamte Struktur mit

der Methode newDevice() an das Gateway-Bundle. Erhält der DPWS-Message-Listener eine Probe-Match-Nachricht, wird damit genauso verfahren.

Probe-Nachrichten werden durch den DPWS-Message-Listener verarbeitet. Beim Eintreffen einer Probe-Nachricht wird diese zusammen mit der Systemzeit, zu der die Nachricht empfangen wurde, gespeichert. Dann wird die Suchanfrage mit der discoverDevice()-Methode an das Gateway-Bundle übergeben. Bekommt das DPWS-Gate vom Gateway-Bundle über die Funktion newDevice() eine Entity-Struktur eines neuen Gerätes eines anderen Protokolls übergeben, erstellt das Gate daraus eine DPWS-Schnittstellenbeschreibung. Das WSDL-Dokument wird von JMEDS automatisch erzeugt.

Aus dem übergebenen Root-Entity wird ein Objekt der Klasse ForeignDevice erzeugt. Diese Klasse ist von der Klasse DefaultDevice aus JMEDS abgeleitet. In dem URI des Foreign-Devices wird der Namespace des Gateways („http://unirostock.org/gateway“) um die Protokollbezeichnung und den Gerätenamen aus dem Root-Entity erweitert (z. B. „http://unirostock.org/gateway/BACnet/BeispielGerät“). Dem Foreign-Device wird dann ein Objekt der Klasse ForeignService hinzugefügt. Die Bezeichnung des Services setzt sich aus dem Gerätenamen und „Service“ zusammen (z. B. BeispielGerätService). Diesem Service wird für jedes Sub-Entity, das eine Liste von Parameter-Entities besitzt, eine Operation zugeordnet. Dafür stehen die Klassen GetOperation, SetOperation und ForeignEvent zur Verfügung (siehe Abbildung 39).

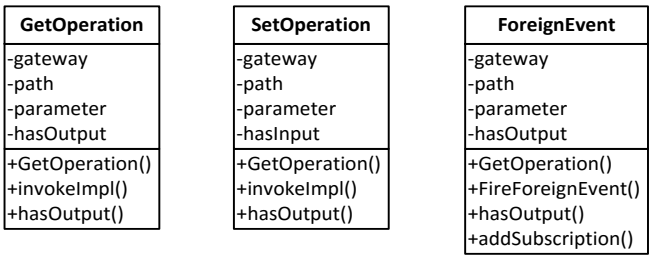


Abbildung 39: Die Operation- und Event-Klassen

Diese Klassen sind von der JMEDS-Klasse Operation beziehungsweise DefaultEventSource abgeleitet. Beim Erzeugen einer Operation wird für jedes Parameter-Entity mit den entsprechenden Methoden geprüft, ob es lesbar (isReadable()), schreibbar (isWriteable()) oder abonnierbar (isSubscribeable()) ist. Für jede Operation wird ein komplexer Datentyp generiert, der die Parameter-Entities als Elemente enthält. Für die Zusammenfassung der Operationen in Port-Typen gibt es zwei Möglichkeiten. Wurde das Sub-Entity, welches die Parameter-Enties enthält, einer Gruppe zugeordnet, das heißt, hat das group-Attribut einen Wert, wird dieser Wert als Port-Typ-Bezeichnung verwendet. Hat das group-Attribut keinen Wert, wird der Inhalt des originIdentifier-Attributs als Port-Typ verwendet. Auch für die Bezeichnung der Operationen kann der originIdentifier verwendet werden. Zuvor wird aber

geprüft, ob die Attribute direction und count des Sub-Entities einen Wert haben. In diesem Fall werden diese für die Bezeichnung der Operationen verwendet. Beim Erzeugen der Operationen wird „Set“, „Get“ oder „Subscribe“ vorangestellt (z. B. SubscribeInput1).

Sind für alle Sub-Entities, die Parameter-Entities enthalten, Operationen angelegt, wird das Foreign-Device in einer Liste mit weiteren Foreign-Devices gespeichert. Im Anschluss daran wird das Gerät gestartet, wobei eine Hello-Nachricht gesendet wird. Danach wird die Liste der gespeicherten Probe-Nachrichten durchgegangen und die gespeicherte Systemzeit mit der aktuellen Systemzeit verglichen. Ist die Differenz größer als 5 Sekunden wird der Eintrag verworfen. Dadurch wird unterbunden, dass auf zu alte Probe-Nachrichten reagiert wird. Ist die Differenz kleiner, wird im Foreign-Device die Methode *sendProbeMatch()* aufgerufen. Darin sendet das Foreign-Device eine Probe-Match-Nachricht als Reaktion auf die zuvor eingegangene Probe-Nachricht. Als Abschluss der newDevice()-Methode wird dem Root-Entity der URI des dazugehörigen Foreign-Devices mit der Methode addIdentifier() zugewiesen. Damit wird die ganze Entity-Struktur an das Gateway-Bundle zurückgegeben, wo sie wieder gespeichert wird.

Wird eine Operation eines Foreign-Devices von einem DPWS-Gerät ausgelöst, ruft JMEDS die *invokeImpl()*-Methode auf. Die Operation besitzt den Pfad (path) des Sub-Entities, welches die Parameter-Entities enthält, und eine Liste mit den Parameter-Entities, auf die die Operation angewendet werden kann. In der Funktion invokeImpl() werden die originIdentifier der Parameter-Entities in eine Map gepackt und zusammen mit dem Pfad an das Gateway-Bundle (gateway) übergeben. Bei der SetOperation enthält die Map zusätzlich die Werte, die geschrieben werden sollen. Bei der GetOperation kommt die Map mit den gelesenen Werten zurück und sie können in der Antwort-Nachricht dem Client übermittelt werden.

Für die Klasse ForeignEvent war eine Veränderung von JMEDS notwendig. Die Operation *addSubscription()* von der JMEDS-Klasse DefaultEventSource kann im Normalfall nicht überschrieben werden. Eine Event-Subscription-Nachricht verarbeitet JMEDS standardmäßig selbstständig. Da diese Nachricht aber an das Gateway-Bundle weitergeleitet werden muss, war es nötig diese Methode zu überschreiben und in die Klasse ForeignEvent aufzunehmen.

In addSubscription() wird eine eingetroffene Abonnement-Anfrage mit Pfad und Parameter-Map an das Gateway-Bundle übergeben. Wird im DPWS-Gate durch das Gateway-Bundle die notification()-Methode aufgerufen und die gemeldeten Werte übergeben, wird über die Methode *fireForeignEvent()* eine Meldung mit diesen Werten an alle Abonnenten gesendet. Ein Parameter-Typ wird bei den Operationen und dem Event gesondert behandelt. Stellt das Parameter-Entity eine Einheit dar (GWparameter.unit), wird der Wert ebenfalls auf eine Gateway-interne Darstellung abgebildet. In der GWunit-Enumeration (siehe Anhang A.1) sind dafür mehrere Einheiten definiert. In der Klasse UnitTable werden Zeichenketten auf

eine dieser Bezeichnungen abgebildet (z. B. „C“ auf GWunit.Celsius). Das dient dazu, dass andere Gates eine Einheit auch als diese interpretieren können.

An diesem Punkt ist das DPWS-Gate in der Lage, ein DPWS-Gerät auf die Gateway-interne Darstellungsform mit Entities zu bringen und aus einer Entity-Struktur eine DPWS-Schnittstelle zu generieren. Außerdem kann auf das Auslösen einer Operation sowie das Abonnieren eines Ereignisses reagiert werden. Lese-, Schreib- und Abonnement-Anfragen, die von anderen Gates kommen, werden vom Ablauf her ähnlich abgearbeitet. Wird eine der Methoden readValue(), writeValue() oder subscription() des DPWS-Gateways vom Gateway-Bundle aufgerufen, werden die übergebenen Parameter direkt an die gleichnamigen Methoden des Gate-Clients übergeben. Die Parameter sind das Root-Entity, der Pfad des Sub-Entities und die Map mit den gewünschten Parameter-Entities. Der Gate Client holt sich anhand des originIdentifiers des Root-Entities die Endpunkt-Referenz des DPWS-Gerätes vom Device- und Service-Manager von JMEDS. Der originIdentifier enthält nämlich in diesem Fall den URI des DPWS-Gerätes. Der Service und die Operation beziehungsweise das Event können anschließend über deren Bezeichner, die im Pfad enthalten sind, ermittelt werden.

Abbildung 44 zeigt am Beispiel eines Air-Conditioners, was beim Aufruf der Methode readValue() passiert. Links sind das Root-Entity, der Pfad und die Map mit den zu lesenden Parametern dargestellt, die der Methode übergeben werden. Rechts sind die Komponenten des Air-Conditioners abgebildet. Die Pfeile zeigen welche Attribute die Informationen enthalten, um das DPWS-Gerät, den Service darin und die Operation zu finden. Diese Operation wird dann ausgelöst. Aus der Antwort-Nachricht kann dann der Wert für das Element „value“, welches im komplexen Element „temperature“ enthalten ist, entnommen werden. In diesem Fall ist die Übergabe des Root-Entities nicht notwendig. Der Uri ist auch im Pfad enthalten. Da die Methode readValue() vom GateInterface vorgegeben ist und dieses protokollunabhängig ist, wird das Root-Entity in dieser Methode mit übergeben. Es besteht die Möglichkeit, dass ein anderes Protokoll weitere Informationen aus der Entity-Struktur benötigt (z. B. Name), die nur darin enthalten sind.

3.6.6. Das BACnet-Gate-Bundle

Für die Kommunikation des Gateways im BACnet-Netzwerk dient das BACnet-Gate-Bundle. Auch für dieses Bundle müssen das Gateway-Bundle und das Bundle mit den externen Bibliotheken im OSGi-Framework installiert sein, da die Abhängigkeitsbeziehungen ansonsten vom Management-Agent nicht aufgelöst werden können. Wie auch beim DPWS-Gate-Bundle werden von diesem Bundle keine Ressourcen exportiert. Das Gate-Interface, welches für die Interaktion mit dem Gateway-Bundle notwendig ist, wird in der Klasse BACnetGateImpl implementiert. Wird das BACnet-Gate-Bundle gestartet, kann der Service-Tracker des Gateway-Bundles das neue Gate anhand des Bundle-Namens dem bacnetGate-

Attribut in GatewayImpl zuweisen. Pakete und Klassen, die das BACnet-Gate-Bundle enthält, sind in Abbildung 40 dargestellt.

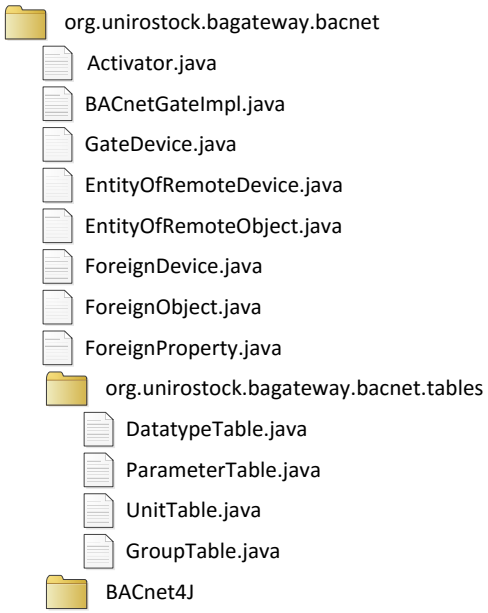


Abbildung 40: Inhalt des BACnet-Gate-Bundles

Das BACnet-Gate verwendet den BACnet4J-Stack in der Version 2.0. Der Stack stellt zum Implementieren von BACnet-Geräten die Klasse LocalDevice bereit. Von dieser Klasse sind für das Gate die Klassen GateDevice und ForeignDevice abgeleitet. Das Foreign-Device kann Objekte der Klasse ForeignObject enthalten. ForeignObject ist von der Klasse BACnetObject des Stacks abgeleitet. Der Klasse ForeignProperty liegt keine Klasse des BACnet4J-Stacks zugrunde. Diese Klassen und ihre Funktion werden später in diesem Abschnitt genauer erläutert. Das Gate-Device repräsentiert auch hier das Gateway selbst. Es besitzt außer dem Device-Objekt keine weiteren BACnet-Objekte. Das Gate-Device belegt den Standard-BACnet-UDP-Port 0xBAC0 im System. Das hat zur Folge, dass keine weitere Anwendung in diesem System über diesen Port Nachrichten senden oder empfangen kann. Die weiteren durch das Gate angelegten BACnet-Schnittstellen (Foreign-Devices) senden und empfangen über die nachfolgenden Ports 0xBAC1, 0xBAC2 und so weiter. Eine Vielzahl von BACnet-Nachrichten wird von BACnet4J selbstständig verarbeitet. Für einige wenige Dienste ist in der Klasse BACnetListener das DeviceEventListener-Interface implementiert. Der BACnetListener enthält Methoden, die zum Beispiel beim Eintreffen einer IAm-Nachricht oder einer Ereignis-Meldung aufgerufen werden. Der BACnetListener ist Teil des Gate-Devices und empfängt somit die Nachrichten über den UDP-Port 0xBAC0. Abbildung 41 zeigt, wie das BACnet-Gate aufgebaut ist.

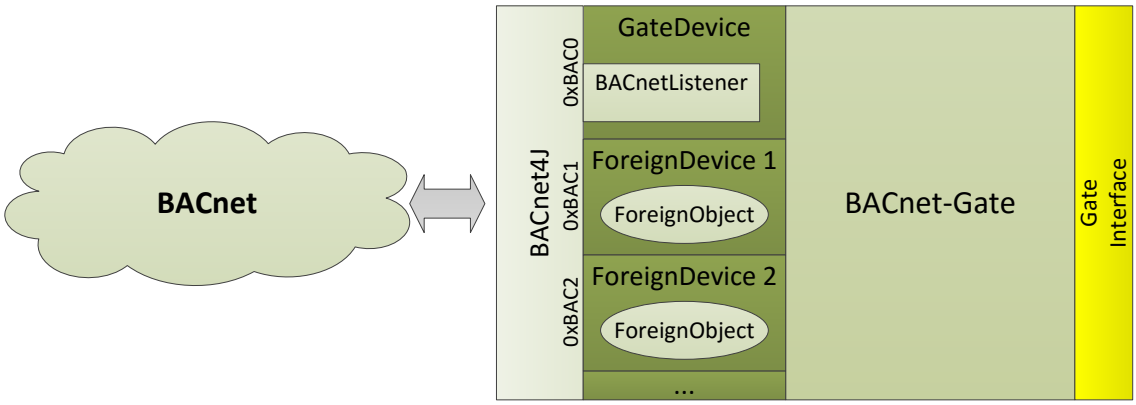


Abbildung 41: Aufbau des BACnet-Gates

Wenn ein Objekt der Klasse BACnetGateImpl beim Start des Bundles erzeugt wird, wird das Gate-Device erzeugt und der UDP-Port 0xBAC0 zugewiesen. Die Instanznummer des Gate-Device ist 4194303. Dies ist die größtmögliche Instanznummer, die vergeben werden kann. Die Instanznummer wird der Liste *usedDeviceIDs* hinzugefügt. In diese Liste werden alle Instanznummern eingetragen, von denen bekannt ist, dass sie im Netzwerk vergeben sind. Das bedeutet, sowohl die Instanznummern von Geräten aus dem Netzwerk, als auch von den Foreign-Devices sind darin enthalten. Für neue Foreign-Devices wird von der größtmöglichen Instanznummer abwärts gezählt und mit dieser Liste überprüft, ob diese schon vergeben ist. Wird eine freie Instanznummer gefunden, kann sie dem Gerät zugewiesen werden. Wenn das Gate-Device erzeugt worden ist, kann es noch nicht im BACnet-Netzwerk kommunizieren. Erst mit dem Aufruf der Methode openGate() wird es im Framework initialisiert und der BACnet-Listener wird für das Gate-Device erzeugt. Zudem wird in dieser Methode durch das Versenden einer IAm-Nachricht per Local-Broadcast (nur in diesem Subnetz) das Gateway bekannt gemacht.

Empfängt das BACnet-Gate eine IAm-Nachricht von einem BACnet-Gerät, so ruft BACnet4J die Methode *iAmReceived()* im BACnet-Listener des Gate-Devices auf. Darin wird als erstes über die Instanznummer des Absenders geprüft, ob die IAm-Nachricht von einem Gerät im Gate oder einem anderen BACnet-Gerät stammt. Ist der Absender ein anderes BACnet-Gerät, wird als erstes dessen Instanznummer der Liste usedDeviceIDs hinzugefügt, wenn diese nicht schon enthalten ist. Da die IAm-Nachricht nicht mehr Information als die Instanznummer des BACnet-Gerätes enthält, muss das BACnet-Gate sich die notwendigen Informationen zum Erstellen der Entity-Struktur nacheinander vom Gerät holen. Um ein Root-Entity anlegen zu können, muss zuerst der Gerätename ausgelesen werden. Für die Sub-Entities muss in Erfahrung gebracht werden, welche BACnet-Objekte das Gerät besitzt. Dazu wird die Objekteigenschaft *objectList* des Device-Objekts mit dem ReadProperty-Dienst gelesen. In der dazugehörigen Antwort sind alle ObjectIdentifier (Objekttyp und Instanznummer) der BACnet-Objekte enthalten. Als nächstes muss von jedem Objekt die objectName-Eigenschaft

gelesen werden. Dann können dem Root-Entity die Sub-Entities hinzugefügt werden. Diesen Sub-Entities werden für jede Objekteigenschaft Parameter-Entities hinzugefügt. BACnet4J enthält für viele Objekttypen ein Array mit den Objekteigenschaften, die das Objekt enthalten kann. Da einige dieser Eigenschaften aber optional sind, muss das Gate herausfinden, welche Eigenschaften tatsächlich vorhanden sind. Dafür wird für jede Eigenschaft eine ReadProperty-Anfrage an das Objekt gestellt. Kommt ein Wert zurück, kann für diese Eigenschaft ein Parameter-Entity erzeugt werden, welches auch lesbar ist. Im nächsten Schritt wird geprüft, ob die Eigenschaft auch schreibbar ist. Dazu sendet das Gate eine WriteProperty-Anfrage mit dem gerade gelesenen Wert. Kommt als Antwort eine Bestätigung, dass der Wert geschrieben wurde, wird das Parameter-Entity auch schreibbar gemacht. Ob eine Eigenschaft auch abonnierbar ist, hängt von zwei Faktoren ab. Zum einen ist das Abonnieren von Wertänderungen durch den BACnet-Standard auf bestimmte Eigenschaften begrenzt. Für die Analog-, Binary- und MultiState-Objekttypen ist das für die Eigenschaften *presentValue* und *status* der Fall. Zum anderen ist es davon abhängig, ob das BACnet-Gerät den COVNotification-Dienst unterstützt. Dies kann in Erfahrung gebracht werden, indem vom Device-Objekt des BACnet-Gerätes die Objekteigenschaft *ProtocolServicesSupported* ausgelesen wird. Diese Eigenschaft enthält die Information, welche Dienste vom BACnet-Gerät unterstützt werden und welche nicht. Wenn beide Faktoren zutreffend sind, wird das Parameter-Entity zusätzlich abonnierbar gemacht. Damit wäre die Entity-Struktur vollständig. Es fehlen jedoch noch die Entity-Attribute count, direction, datatype, parametername und group. Das count-Attribut wird mit der Instanznummer des Objekts belegt und das direction-Attribut ist davon abhängig, ob es sich um einen Eingang (z. B. Analog-Input), einen Ausgang (z. B. Analog-Output) oder einen Wert (z. B. Analog-Value) handelt. Für die Datentypen gibt es auch im BACnet-Gate eine Map in der Klasse DatatypeTable. Diese bildet die BACnet-Datentypen auf die GWdatatype-Datentypen ab und umgekehrt. Für das parametername-Attribut gibt es eine Map in der Klasse ParameterTable. Für das group-Attribut enthält die Klasse GroupTable eine Map. Objekte von einem Binary- oder MultiState-Objekttyp werden darin den Gruppen „BINARY“ oder „MULTISTATE“ entsprechend zugeordnet. Für die Analog-Objekttypen ist eine andere Eigenschaft ausschlaggebend. Da die Objekteigenschaft *unit* für diese Typen eine obligatorische Eigenschaft ist, kann diese in jedem Fall gelesen werden. Eine Map in der Klasse GroupTable bildet eine BACnet-Einheit auf eine in der GWgroup-Enumeration enthaltenen physikalischen Größe ab. Abbildung 42 stellt an einem Beispiel dar, wie die Entity-Attribute aus einem Objekt dem Sub- und Parameter-Entity zugeordnet werden. Damit sind in der Entity-Struktur auch die Informationen zur besseren Interpretation der Daten enthalten. Diese Struktur wird dann an das Gateway-Bundle übergeben.

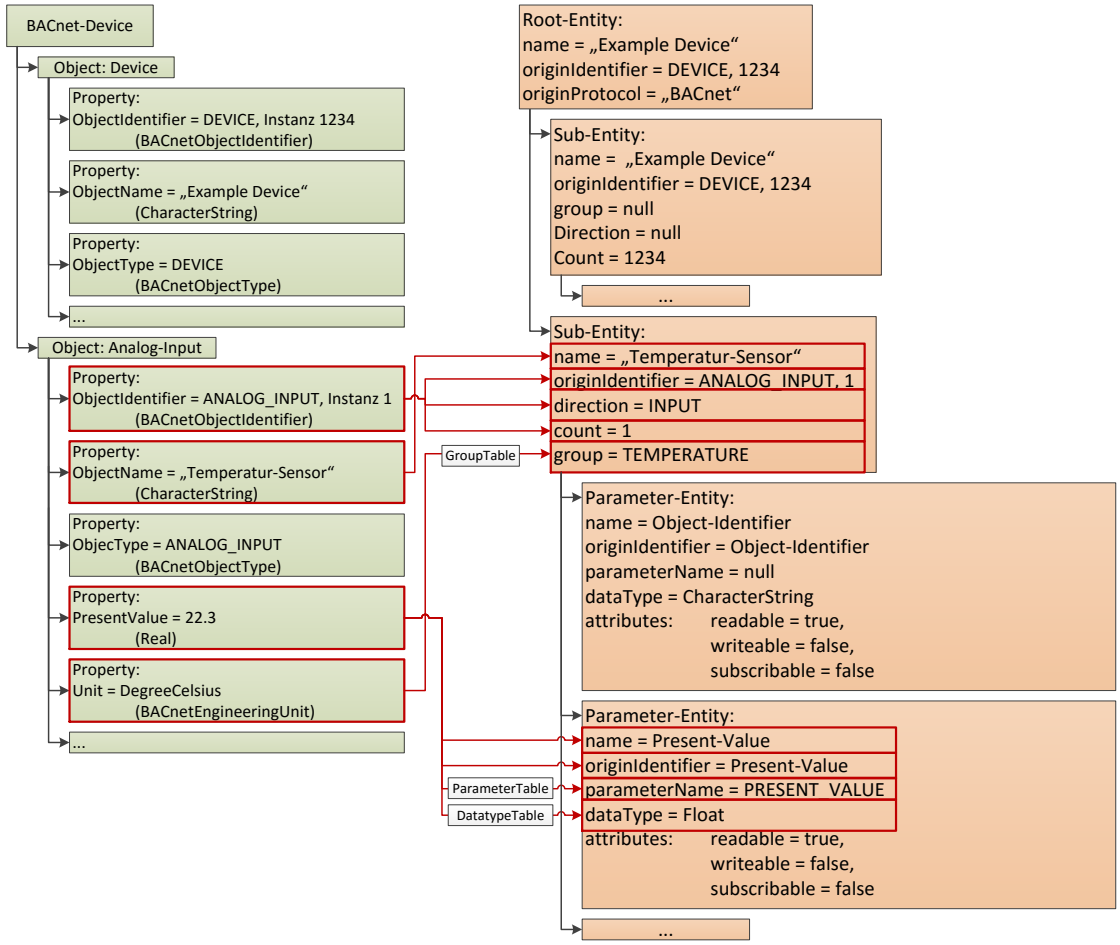


Abbildung 42: Belegung der Entity-Attribute aus den Objekteigenschaften.

Damit das BACnet-Gate eine WhoIs-Nachricht an das Gateway-Bundle weiterleiten kann, sind Veränderungen im BACnet4J-Stack vorgenommen worden. Das DeviceEventListener-Interface wurde um die Methode *whoIsReceived()* und die Klasse *DeviceEventHandler* um *fireWhoIsReceived()* erweitert. In der Methode fireWhoIsReceived() wird whoIsReceived() aufgerufen. Der Aufruf von fireWhoIsReceived() erfolgt in der handle()-Methode der Klasse *WhoIsRequest*.

Im BACnet-Listener wird mit dieser Erweiterung von BACnet4J die Methode whoIsReceived() beim Eintreffen einer WhoIs-Nachricht aufgerufen. Darin wird die Methode discoverDevice() im Gateway-Bundle aufgerufen. Wird dem BACnet-Gate über newDevice() eine neue Entity-Struktur übergeben, wird als erstes geprüft, ob noch eine Instanznummer für ein weiteres BACnet-Gerät in diesem Netzwerk frei ist. Wenn das der Fall ist, wird ein neues Foreign-Device mit einer freien Instanznummer erstellt. Die gewählte Instanznummer wird in die Liste usedDeviceIds eingetragen. Dann wird das Foreign-Device dem nächsten freien UDP-Port (nach 0xBAC0) zugewiesen. Beim Erstellen einer BACnet-Darstellung aus der

Entity-Struktur kommen die Klassen ForeignDevice, ForeignObject und ForeignProperty zum Einsatz (siehe Abbildung 43).

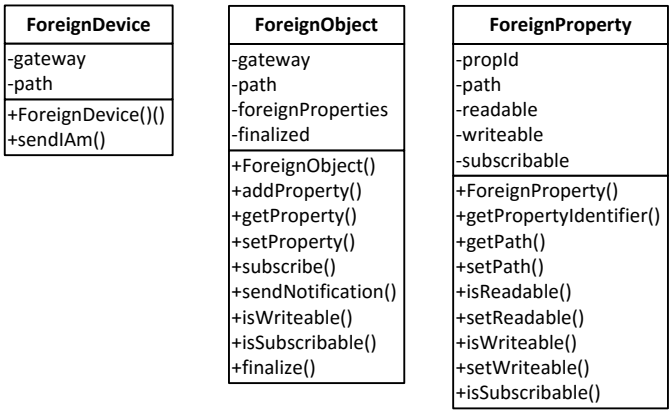


Abbildung 43: Die Foreign-Klassen

Der Name des Root-Entities wird als Geräte-Name eingetragen. Dann werden in der Entity-Struktur die Sub-Entities gesucht, die Parameter-Entities enthalten. Können Parameter-Entities Objekteigenschaften zugeordnet werden, wird für diese ein gemeinsames Foreign-Objekt angelegt. Für Parameter-Entities, die keiner Objekteigenschaft zugeordnet werden können, wird ein extra Objekt angelegt. Der Objekttyp hängt von dem Datentyp des Parameter-Entities, das der PresentValue-Eigenschaft zugeordnet wird, ab. Welcher Datentyp welchem Objekttyp zugewiesen wird, enthält Anhang A.1. Der Objekt-Name ist aus den Namen der Entities zusammengesetzt, die zu dem Parameter-Entity führen. BACnet4J erlaubt es nicht, dass zwei Objekte in einem Gerät denselben Objekt-Namen besitzen. Daher kann für die Bezeichnung weder der Name des Sub-Entities (Konflikt, wenn dessen Parameter-Entities auf mehrere Objekte abgebildet werden) noch der Name des Parameter-Entities selbst (Konflikt, wenn dieser Name in einem anderen Sub-Entity enthalten ist) verwendet werden. Der Pfad ist für jedes Parameter-Entity eindeutig. Im nicht modifizierten BACnet4J-Stack wäre das Anlegen eines BACnet-Gerätes an diesem Punkt abgeschlossen. Die Klasse ForeignObject wird gegenüber der Klasse BACnetObject dahingehend erweitert, dass es eine Liste mit Objekten der Klasse ForeignProperty enthält. In dieser Liste sind alle Objekteigenschaften zusammengefasst, die über das Gateway gelesen, geschrieben oder abonniert werden können. Außerdem sind darin die Angaben enthalten, ob eine Objekteigenschaft lesbar, schreibbar oder abonnierbar ist. Sind alle Parameter-Entities Objekteigenschaften zugeordnet, ist die Erzeugung der BACnet-Schnittstelle für das Gerät abgeschlossen. Das Foreign-Device sendet nun eine IAm-Nachricht von seinem UDP-Port per Local-Broadcast auf den BACnet-Standard-UDP-Port der anderen BACnet-Geräte im Netzwerk. Abschließend wird der ObjectIdentifier des Device-Objekts (Device +

Instanznummer) über addIdentifier() im Root-Entity eingetragen, bevor die Entity-Struktur an das Gateway-Bundle zurückgegeben wird.

Wie zu Beginn dieses Abschnitts erwähnt, verarbeitet BACnet4J viele Dienst-Anfragen eigenständig. Dazu gehören auch die Dienste ReadProperty, WriteProperty und SubscibeCOV. Zum Lesen und Schreiben werden im Foreign-Object die Methoden *getProperty()* und *setProperty()* der Klasse BACnetObject überschrieben. Damit kann das Gate auf Dienst-Anfragen dieser Art anders reagieren, als in BACnet4J vorgesehen. Wird mit dem Dienst auf eine Eigenschaft zugegriffen, die in der Liste foreignProperties enthalten ist, wird die Anfrage mit der entsprechenden Methode des Gateway-Bundles weitergeleitet. Bezieht sich der Dienst auf eine Eigenschaft, die nicht in der Liste steht und damit auch auf kein Parameter-Entity zugreift (z. B. ObjectIdentifier, Status), werden die gleichen Schritte abgearbeitet wie in der Methode in BACnetOperation. Bei den Diensten ReadPropertyMultiple und WritePropertyMultiple arbeitet der Stack die Eigenschaften nacheinander ab. Das bedeutet, dass für jede Eigenschaft die Methode setProperty beziehungsweise getProperty einzeln aufgerufen wird. Wenn ein Parameter-Entity nicht lesbar ist, wird bei einer ReadProperty-Anfrage an die dazugehörige Objekteigenschaft der Fehlercode für einen nicht initialisierten Wert als Antwort gesendet. Bei einer Schreibanfrage an eine Eigenschaft, die nicht geschrieben werden kann, sendet BACnet4J eigenständig die Fehlermeldung, dass der Dienst für diese Eigenschaft nicht unterstützt wird.

Für Ereignis-Abonnements ist eine weitere Veränderung von BACnet4J notwendig. BACnet4J sieht vor, dass entweder ein Objekt den SubscribeCOV-Dienst unterstützt oder nicht. Es ist nicht von einzelnen Objekteigenschaften abhängig. Im BACnet-Gate unterstützen alle Objekte grundsätzlich den SubscribeCOV-Dienst. Ob der Dienst aber tatsächlich angewendet werden kann, ist abhängig von der Objekteigenschaft. Daher wird in der handle()-Methode der Klasse SubscribeCOVRequest mit der Methode *isSubscribable()* geprüft, ob der SubscribeCOV-Dienst auf die Objekteigenschaft angewendet werden kann. Wenn die Eigenschaft abonnierbar ist, wird mit der Methode *subscribe()* des Foreign-Objects die Weiterleitung zum Gateway-Bundle veranlasst. Wenn nicht, sendet das Objekt die Fehlermeldung, dass der Dienst nicht unterstützt wird. Kommt eine Ereignis-Meldung von dem Gerät, das durch das Foreign-Device repräsentiert wird, erfolgt das Senden einer COVNotification über die Methode *sendNotification()*. Bei allen Funktionen werden auch im BACnet-Gate Werte, die eine Einheit darstellen, gesondert behandelt. Für die Zuweisung von BACnet-Einheiten zu den Einheiten der GWunit-Enumeration enthält die Klasse UnitTable eine Map.

An diesem Punkt ist das BACnet-Gate fähig, ein BACnet-Gerät in eine Entity-Struktur zu überführen sowie aus einer Entity-Struktur eine BACnet-Schnittstelle mit Objekten und Objekteigenschaften zu erstellen. Auf die Dienste ReadPropety(Mulitple),

WriteProperty(Multiple) und SubscribeCOV an die BACnet-Schnittstelle kann das Gate reagieren und das Gateway-Bundle weiterleiten. Kommt vom Gateway-Bundle über die Methoden readValue(), writeValue() oder subscribe() wiederum eine Anfrage, muss das BACnet-Gate diese an das entsprechende BACnet-Gerät weiterleiten. Allen drei Methoden werden das Root-Entity und der Pfad des Sub-Entities übergeben. Die Methoden readValue() und writeValue() erhalten zusätzlich die Map mit den Parameter-Entities. Aus dem Root-Entity kann über das originIdentifier-Attribut der Object-Identifier des Device-Objekts vom Zielgerät bestimmt werden. Dem Pfad wird der Object-Identidifier des Ziel-Objekts entnommen. Damit kann für jede gewünschte Eigenschaft aus der Map eine ReadProperty- beziehungsweise WriteProperty-Anfrage an das BACnet-Gerät gesendet werden. Das BACnet-Gate nutzt zum Lesen und Schreiben von Werten den einfachen ReadProperty- und Write-Property-Dienst, da nicht jedes Gerät den Zugriff auf mehrere Objekteigenschaften mit den Diensten ReadPropertyMultiple und WritePropertyMulitple unterstützt. In der readValue()-Methode werden empfangene Werte den Parameter-Entities zugeordnet und die Map kann an das Gateway-Bundle zurückgegeben werden. Die Methode subscribe() benötigt keine Angabe von Eigenschaften, die abonniert werden sollen, da der SubscribeCOV-Dienst immer auf das ganze BACnet-Objekt angewendet wird. Kommt vom Gateway-Bundle eine Ereignis-Meldung über die notification()-Methode, wird im dazugehörigen Foreign-Object die Methode sendNotification aufgerufen. Damit kann auch das BACnet-Gate alle gestellten funktionalen Anforderungen erfüllen.

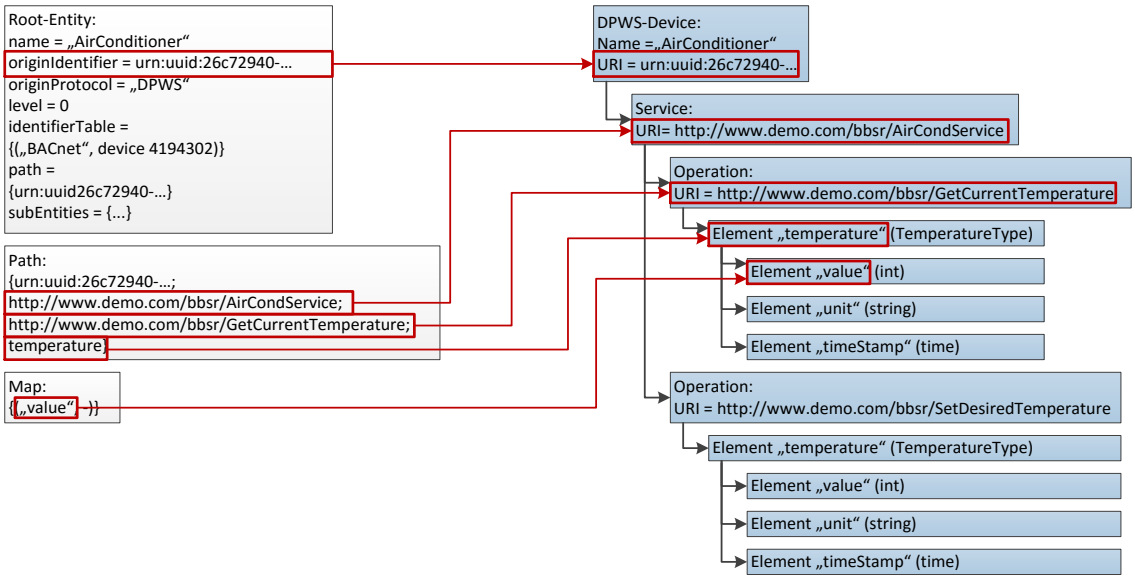


Abbildung 44: Aufruf der readValue()-Methode im DPWS-Gate-Bundle

Bei einer Lese-Operation und einem Event können diese dann ausgelöst beziehungsweise abonniert werden. Bei einer Schreib-Operation werden zunächst alle obligatorischen

Elemente mit dem letzten bekannten Wert belegt. Im Anschluss daran werden einige Werte mit denen aus der Parameter-Map überschrieben. Dann kann auch die Schreib-Operation ausgelöst werden. Damit erfüllt das DPWS-Gate alle funktionalen Anforderungen. Dabei arbeitet es unabhängig davon, von welchem Protokoll das Gate ist, mit dem über das Gateway-Bundle kommuniziert wird.

3.7. Zusammenfassung des Kapitels

Ein Teilaspekt der vorliegenden Arbeit ist es, ein Konzept für ein Gateway zu entwickeln, über das Geräte aus den Standards DPWS und BACnet Informationen austauschen können. Die dazu erforderlichen Funktionalitäten sind das Suchen und Finden von Geräten des jeweils anderen Standards sowie das Lesen und Schreiben von Daten und Abonnieren und Melden von Wertänderungen, die das jeweilige Gerät bereitstellt. Dann wird ein Prototyp des Gateways implementiert und auf seine Funktionsfähigkeit getestet. Da das BACnet-Kommunikationsprotokoll auf die Gebäudeautomation beschränkt ist, liegen auch die Einsatzmöglichkeiten des BACnet-DPWS-Gateways in diesem Bereich.

Zu Beginn werden die Standards unabhängig voneinander analysiert. Es wird zum einen betrachtet, welche Möglichkeiten der jeweilige Standard bietet, um Informationen im Netzwerk darzustellen, und wie die Kommunikation zwischen den Geräten abläuft. Zusätzlich wird BACnet/WS eine Form der Darstellung von Gebäudedaten mit Web Services betrachtet. Im Anschluss findet ein Vergleich von BACnet und DPWS hinsichtlich der gewünschten Gateway-Funktionalitäten statt. Um Geräteinformationen protokollunabhängig im Gateway speichern zu können, wird eine Struktur entwickelt, in der die einzelnen Komponenten der Geräte auf Entities abgebildet werden.

Da die Datendarstellung in BACnet vom Standard stark spezifiziert ist, kann von einer festen Struktur der BACnet-Geräte ausgegangen werden. Diese Struktur kann durch das BACnet-DPWS-Gateway gut auf eine DPWS-Schnittstelle abgebildet werden. DPWS spezifiziert keine feste Darstellung der Daten, wodurch auch von keiner festen Struktur ausgegangen werden kann, um für DPWS-Geräte eine passende BACnet-Darstellung zu erzeugen. Das Gateway bietet jedoch mit den GW-Enumerations und den in den Gates implementierten Maps (GroupTable, DatatypeTable etc.) eine Möglichkeit, die Informationen dennoch richtig zu interpretiert. Dafür müssen aber im DPWS-Gerät Einschränkungen bei der Bezeichnung der Elemente berücksichtigt werden.

Für die Implementierung des BACnet-DPWS-Gateways wird das OSGi-Framework Eclipse Equinox als Basisframework verwendet. Das Gateway besteht aus einem zentralen Gateway-Bundle und jeweils ein Gate-Bundle für die Standards BACnet und DPWS. Zusätzlich gibt es ein Bundle, das die Bibliotheken enthält, die von den Frameworks der Gates benötigt werden. Das DPWS-Gate verwendet als Framework JMEDS. Das BACnet-Gate baut auf den

BACnet4J-Stack auf. Die Bundles verwenden definierte Schnittstellen, wobei die Schnittstellen der Gates identisch und protokollunabhängig sind.

Durch die vom Kommunikationsprotokoll unabhängigen Gate-Schnittstellen und die protokollneutrale Gateway-interne Darstellung von Geräten ist eine Grundlage für Gates anderer Protokolle in der Gebäudeautomation gelegt. Ein Gate muss dazu in der Lage sein, eine Struktur von Entities auf eine dem Standard entsprechende Darstellungsform zu übertragen sowie eine Anordnung von Entities für Geräte des Standards zu erstellen. Dazu müssen die Methoden des GateInterface entsprechend vom Gate implementiert werden. Somit kann aus dem Gateway für die Standards BACnet und DPWS ein Gateway für die Gebäudeautomation entstehen.

4. Smart Metering im Smart Home

Smart Metering stellt eine der tragenden Säulen im Smart Home. Da die Nutzer nicht nur Geräte steuern, sondern auch Informationen über das Haus wie z.B. Tagesverbrauch von Strom, Wasser oder Gas erhalten möchten. Solche Informationen werden in den meisten Fällen zwar von den Zählern erfasst, können jedoch aufgrund einer fehlenden Kommunikationseinheit nicht weitergegeben werden. Dieses Problem soll von intelligenten Zählern, den Smart Metern, gelöst werden. Die Einführung der Smart Metern in Deutschland hat sich jedoch stark verlangsamt. Ein Grund hierfür ist die mangelnde Sicherheit aktueller am Markt erhältlichen Smart Metering-Systeme. Abhilfe soll eine vom Bundesamt für Sicherheit in der Informationstechnik (BSI) vorgestellte Richtlinie für eine Kommunikationseinheit eines intelligenten Messsystems für Stoff- und Energiemengen schaffen [93]. Eine zentrale Komponente der Richtlinie ist ein Smart Meter Gateway, das die Kommunikation zwischen den Zählern und den Messstellenbetreibern absichern. Da die Richtlinie sich weiterhin in der Entwicklung befindet, hält sie die Hersteller davon ab, kompatible Smart Meter bzw. Smart Meter Gateway herzustellen und zu zertifizieren. Darüber hinaus hat die Studie von Ernst & Young über die Kosten-Nutzen-Analyse der Smart Meter ergeben, dass sich ein flächendeckendes Rollout von Smart Metern nur im Fall einer großen Renovierung oder Neubau rentiert [94], da der Einbau der Smart Meter mit hohen Kosten verbunden ist. Es ist daher nicht zu erwarten, dass Hausbesitzer freiwillig Smart Meter einbauen werden, solange sich die Kosten nicht amortisieren.

Es stellt sich damit die Frage, wie eine Realisierung des Smart Home in naher Zukunft ohne eines flächendeckendes Einsatzes der Smart Meter möglich ist. Eine Möglichkeit wäre die existierenden Zähler für Strom, Gas, Wasser mit einer kostengünstigen Kommunikationseinheit zu ergänzen. Die herkömmlichen Zähler sind jedoch in der Regel mechanisch, sodass eine Erweiterung der Funktionalität schwierig ist. Zu den Aufgaben der Kommunikationseinheit würde dann die Ablesung des Zählers gehören. Um die Lösung möglichst kostengünstig zu gestalten, soll die Kommunikationseinheit in der Lage sein beliebige Zählerstände ablesen zu können und den abgelesenen Wert über eine Kommunikationsschnittstelle zu übertragen. Durch die vielfältige Beschaffenheit der Zähler ist es einheitlich nur möglich an den Zählerstand direkt über die Anzeige zu kommen. Die Anzeige muss somit optisch erfasst und der Zählerstand ausgewertet werden. Hierdurch könnten beliebige Zähler kostengünstig zu Smart Metern aufgerüstet werden. Eine solche Lösung kann darüber hinaus schnell umgesetzt werden und dabei die Einführung des Smart Home deutlich beschleunigen.

Eines der Ziele dieses Projektes ist es daher, ein kostengünstiges System zu entwickeln, das den Zählerstand eines analogen Stromzählers in kurzen Intervallen ausliest und diese Daten leicht zur Verfügung stellt. Die Daten werden als Web Service im lokalen Netzwerk

verfügbar gemacht. Als Plattform wurde der Raspberry Pi gewählt, da dieser trotz geringer Anschaffungs- und Verbrauchskosten eine relativ hohe Rechenleistung erreicht. Gerade die verfügbare Rechenleistung war wichtig für die Wahl der Plattform, weil Algorithmen der Bildverarbeitung und Mustererkennung im Allgemeinen sehr rechenintensiv sind. Abbildung 45 zeigt die Grundidee des Systems. Mit einer Kamera soll ein Bild vom Stromzähler aufgenommen, aus diesem der Zählerstand extrahiert und die erhaltenen Daten über einen Web Service im Netzwerk verfügbar gemacht werden. Aufgrund der Interoperabilität von Web Services können diese Daten dann von verschiedenen Geräten wie Desktops, Notebooks oder Smartphones abgerufen werden.

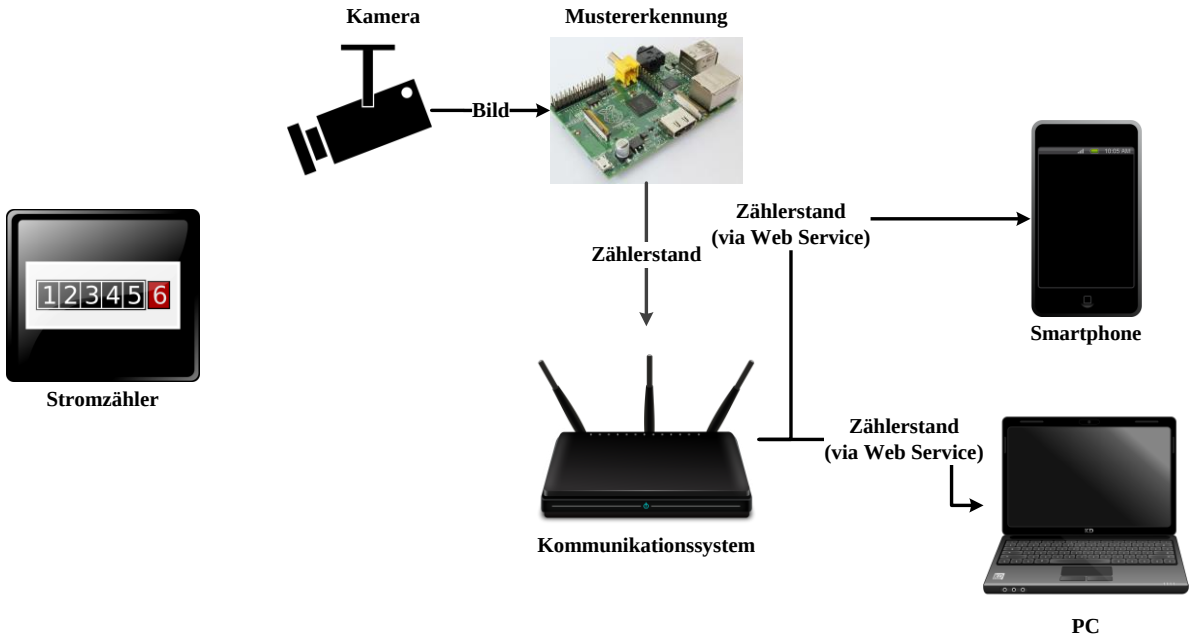


Abbildung 45: Grundidee des Systems

In Wirtschaft und Forschung gibt es aktuell nur wenige Systeme, die versuchen, alte Stromzähler mit neuen Systemen zu erweitern. Häufig sollen bestehende Zähler komplett durch neue Einheiten ersetzt werden. In diesem Projekt wird ein System entwickelt, welches die bestehenden Stromzähler erweitert, um so die Umrüstungskosten zu senken.

Ein aktuelles Problem in der optischen Mustererkennung ist die geringe Erkennungsrate bei niedrigen Auflösungen, komplexen Hintergründen und schlechter Beleuchtung. Dieses Problem tritt unabhängig vom Einsatzgebiet wie Schrifterkennung, Erkennung von Kraftfahrzeugkennzeichen und automatisches Ablesen von Stromzählern auf [4, 5].

4.1. Grundlagen

In diesem Kapitel werden die Grundlagen dargestellt. Hierzu wird auf die Gebiete der Bildverarbeitung, der Mustererkennung und speziell der künstlichen neuronalen Netzen eingegangen.

4.1.1. Bildverarbeitung

Im folgenden Abschnitt soll auf jene Verfahren der digitalen Bildverarbeitung eingegangen werden, welche im späteren Verlauf der Arbeit Verwendung finden. Zunächst wird die Kantendetektion beschrieben. Kantendetektionsverfahren werden in dieser Arbeit unter anderem zur Extraktion der Bereiche genutzt, in denen sich der Zählerstand befinden könnte. In der Mustererkennung werden diese Bereiche Regions of Interest (ROIs) genannt. Danach wird auf die Hough-Transformation eingegangen. Die Hough-Transformation erlaubt die Detektion von Geraden auf verrauschten Bildern. Auch dieses Verfahren kommt bei der ROI-Extraktion zum Einsatz. In der Folge werden Segmentierungsverfahren beschrieben. Die Segmentierung bzw. Schwellwertbildung wird in dieser Arbeit verwendet, um ein Bild in den Hintergrund und die Vordergrundobjekte zu unterteilen. Anschließend werden die Operationen der Erosion und Dilatation erläutert. Mit Hilfe dieser Operationen können nach der Segmentierung kleinere Objekte effizient herausgefiltert werden.

4.1.2. Kantendetektion

Die Kantendetektion ist eine häufige Aufgabe in der Bildverarbeitung. Bei einem Kantendetektor handelt es sich immer um einen Nachbarschaftsoperator, welcher auf einen Pixel und dessen direkte Nachbarschaft angewendet wird. Der Kantendetektor ist dabei in der Grundidee ein Filter, welches konstante Regionen im Bild unterdrückt und Veränderungen hervorhebt.

Da in der digitalen Bildverarbeitung immer mit räumlich abgetasteten Bildern gearbeitet wird, muss die Ableitung, auf der die Kantendetektion beruht, immer durch eine Differenzbildung approximiert werden. Dabei kommt es zu Fehlern. Beispielsweise werden Kanten nicht in allen Richtungen gleich gut erkannt, da lediglich in *x*- und *y*-Richtig abgleitet wird [6]. Einer der wichtigsten Kantendetektoren ist der Sobel-Operator. Bei der Sobel-Kantendetektion wird das Bild *X* mit zwei Masken *Dx* und *Dy* gefaltet, wie die Formeln (1) **und** (2) **beschreiben. Formel** (4) **zeigt die Maske *Dx*, mit welcher die Ableitung in *x*-Richtung erzeugt wird. Formel** (5) **zeigt die Maske *Dy*, zur Erzeugung der Ableitung in *y*-Richtung. Das Kantenbild *E* ergibt sich durch Addition der Absolutbeträge der Ableitungen in *x*- und *y*-Richtung nach Formel** (3)**.**

Abbildung 46 zeigt das Eingangsbild *X*, Abbildung 47 das Ergebnis der Kantendetektion *E*.

$$Ex = Dx * X \tag{1}$$

$$Ey = Dy * X \tag{2}$$

$$E = |Ex| + |Ey| \tag{3}$$

$$Dx = \begin{matrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{matrix} \tag{4}$$

$$Dy = \begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix} \tag{5}$$

Abbildung 46: Originalbild

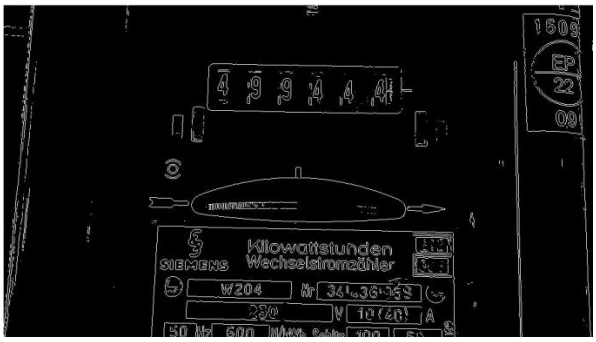


Abbildung 47: Bild nach der Kantendetektion

4.1.3. Hough-Transformation

Die Hough-Transformation kann zur Erkennung von Geraden eingesetzt werden. Auch auf verrauschten Bildern erreicht sie dabei sehr gute Ergebnisse. Die Grundidee liegt darin, dass eine Gerade im Bild in einen Punkt im Hough-Raum transformiert wird.

Nach der Hesseschen Normalform lässt sich eine Gerade eindeutig beschreiben, durch ihren Abstand *d* vom Koordinatenursprung und den Normalenvektor $\overrightarrow{n_0}$, der senkrecht auf der Geraden steht. Es gehören alle Punkte zur Geraden, deren Ortsvektor $\vec{x}$ die Gleichung (6) erfüllt. Abbildung 48 zeigt eine Gerade und den dazugehörigen Ortsvektor n_0. Abbildung 49 zeigt, dass sich die Komponenten x_0 und y_0 des Ortsvektors aus dem Winkel ρ berechnen lassen. Aus den trigonometrischen Beziehungen am rechtwinkligen Dreieck und mit $|\overrightarrow{n_0}| = 1$ ergeben sich die Komponenten x_0 und y_0 nach den Gleichungen (7) und (8). Alle möglichen Geraden, die durch einen beliebigen Punkt *(x,y)* gelegt werden können, lassen sich also durch die Gleichung (9) darstellen. Jeder Punkt der Kurve entspricht dabei genau einer Geraden. Eine Gerade im *x,y*- Raum entspricht somit einem Punkt im ρ, d-Raum und für jeden Punkt im *x,y*- Raum lässt sich eine Kurve im ρ, d-Raum (Hough-Raum) erzeugen. Auf dieser Kurve liegen dann alle Geraden, die durch den Punkt im *x,y*- Raum gelegt werden können.

Zunächst wird eine Kantendetektion auf das Bild angewendet, damit nur noch Pixel, die auf einer Kante liegen, im Bild einen Wert ungleich Null haben. Für jeden dieser Punkte wird eine Kurve nach Gleichung (9) im Hough-Raum erzeugt. Die wahrscheinlichsten Geraden im Originalbild entsprechen dann den lokalen Maxima im Hough-Raum [6].

$$\vec{x} \cdot \overrightarrow{n_0} = d \tag{6}$$

$$x_0 = \cos(\rho) \tag{7}$$

$$y_0 = \sin(\rho) \tag{8}$$

$$d = \vec{x} \cdot \overrightarrow{n_0} = \begin{pmatrix} x \\ y \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = x \cdot \cos(\rho) + y \cdot \sin(\rho) \tag{9}$$

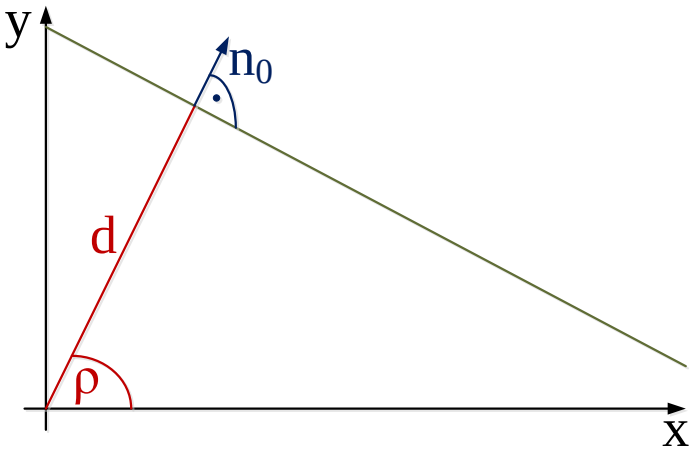


Abbildung 48: Gerade im x,y-Raum

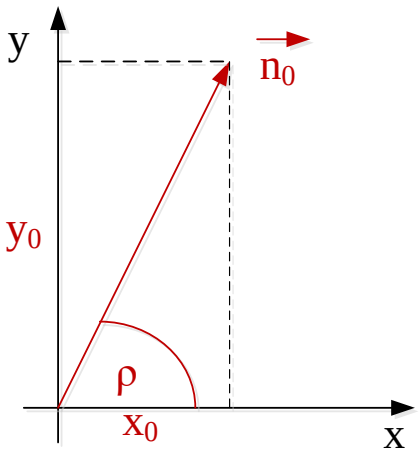


Abbildung 49: Berechnung der Komponenten von n_0

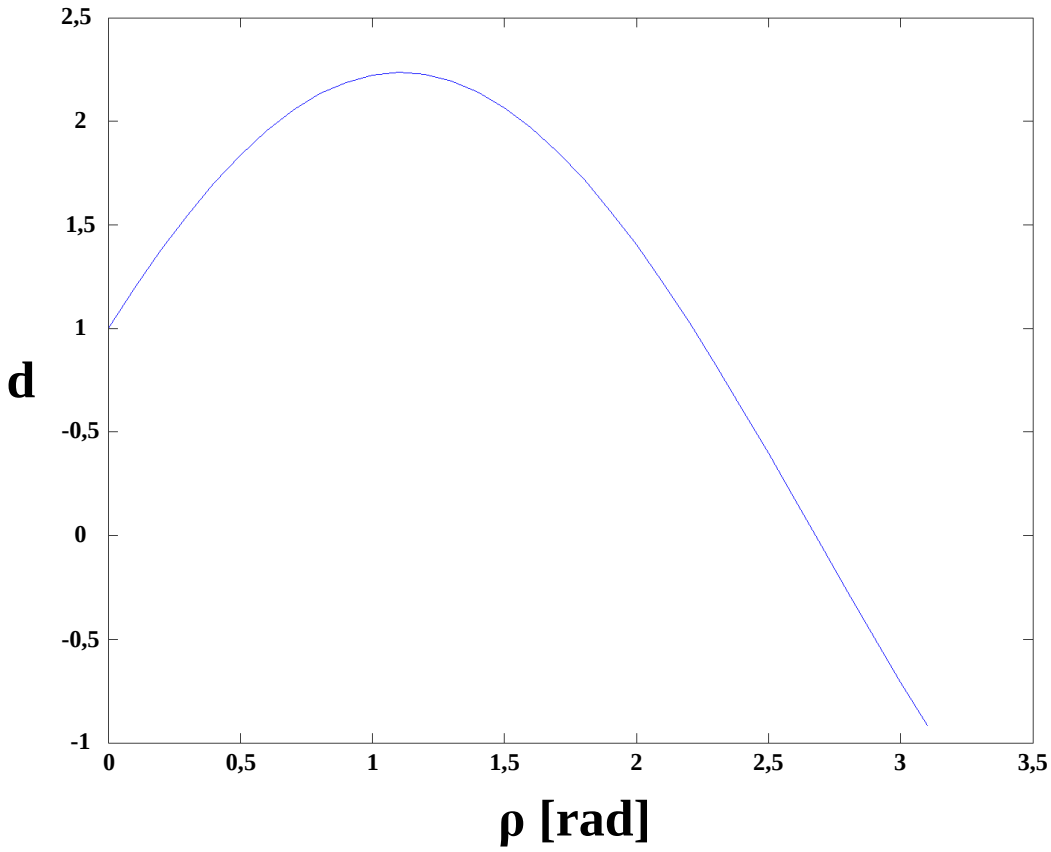


Abbildung 50: Kurve in Hough-Raum für den Punkt x=1,y=2

4.1.4. Segmentierung und Schwellwertbildung

Die **Segmentierung** beschäftigt sich im Allgemeinen mit der Bestimmung der Zugehörigkeit eines Pixels zu einem Objekt. Hierbei kann zwischen pixelbasierter oder regionenorientierter Segmentierung unterschieden werden. Bei den pixelbasierten Methoden werden nur die Grauwerte einzelner Pixel betrachtet, während die regionenorientierten Verfahren die Grauwerte von zusammenhängenden Regionen in Betracht ziehen. Segmentierung und **Schwellwertbildung** sind eng miteinander verknüpft. Bei der Schwellwertbildung wird ein Schwellwert auf ein Grauwertbild angewendet. Alle Pixel, deren Grauwert über dem Schwellwert liegt, werden mit einem Wert belegt (z.B. 1) und alle anderen Pixel mit einem anderen Wert (z.B. 0). Eine simple Methode zur Segmentierung eines Bildes besteht darin, das Bild zunächst in ein Grauwertbild umzuwandeln und es dann mit Hilfe eines Schwellwertes in die Vordergrundobjekte und den Hintergrund zu segmentieren. Eine Segmentierung auf Grundlage der Grauwerte ist immer dann sinnvoll, wenn sich die Verteilung der Grauwerte im Bild als bimodale Merkmalsverteilung darstellt. Eine bimodale Verteilung ist eine Verteilung, die zwei gut getrennte Maxima aufweist. Abbildung 51 zeigt eine bimodale Verteilung. Für die Grauwerte im Bild bedeutet dies, dass die Menge der Vordergrundpixel und die Menge der Hintergrundpixel nur eine geringe Varianz bezüglich der Grauwerte aufweisen, die beiden Mittelwerte der zwei Mengen aber weit voneinander entfernt liegen. Die Wahl des Schwellwertes ist für die Weiterverarbeitung des Bildes von großer Bedeutung. Ein schlecht gewählter Schwellwert führt dazu, dass Objekte zu groß bzw. zu klein segmentiert werden und es ggf. zu einer fehlerhaften Bestimmung der geometrischen Merkmale des Objektes kommt. Bei einer bimodalen Merkmalsverteilung und symmetrischen Objektkanten entspricht der optimale Schwellwert genau dem mittleren Grauwert aus Objekt- und Hintergrund-Pixeln. Allerdings versagt dieses Vorgehen bei einem inhomogenen Hintergrund und Objekten mit unterschiedlichen Grauwerten, weil dann nicht mehr von einer bimodalen Verteilung ausgegangen werden kann. In der Praxis hat sich gezeigt, dass es u.U. unmöglich ist, einen geeigneten globalen Schwellwert zu finden. Daher wird zunächst versucht, die ungleichmäßige Beleuchtung auszugleichen, um danach eine verbesserte Segmentierung erreichen zu können. Bei Videodaten kann die ungleichmäßige Beleuchtung unter anderem mittels Division durch Referenzbilder ausgeglichen werden.

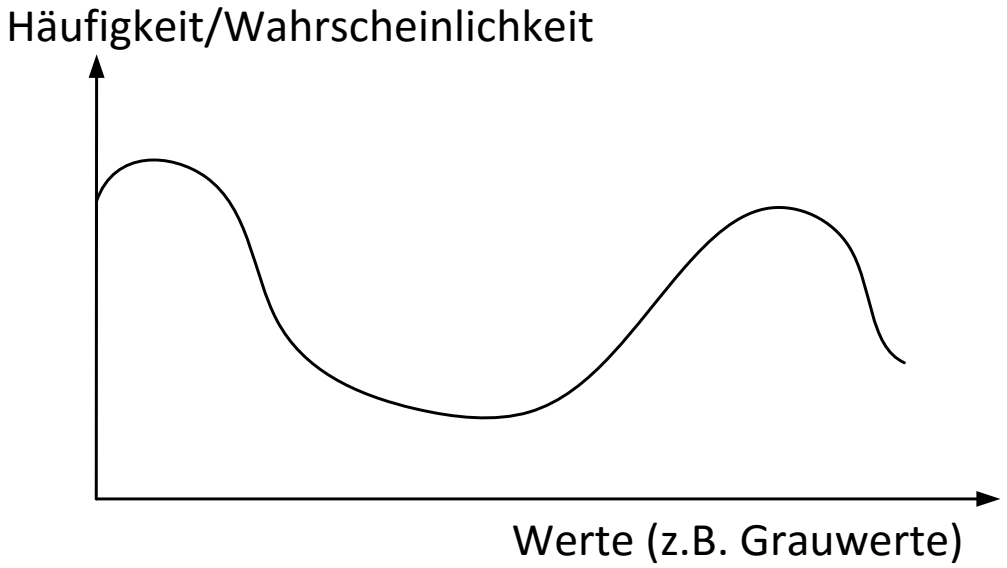


Abbildung 51: Bimodale Wahrscheinlichkeitsverteilung

Wenn zur Objekterkennung ein Grauwertbild in ein Binärbild (Werte 0 und 1) umgewandelt werden soll, muss der ursprüngliche Bereich (z.B. von 0 bis 255) auf diese Werte abgebildet werden. Dabei kann eine 0 anzeigen, dass es sich um einen Hintergrundpixel handelt und eine 1, dass der Pixel zu einem Vordergrundobjekt gehört. Die Schwellwertbildung beschäftigt sich dabei mit der Frage, wie der optimale Schwellwert T gewählt werden muss, sodass sich Vordergrund und Hintergrund möglichst gut voneinander trennen lassen. Die Anwendung eines Schwellwertes auf ein Bild kann wie in Formel (10) erfolgen, wobei BW_i der i-te Pixel des Binärbildes, G_i der i-te Pixel des Grauwertbildes und T der Schwellwert ist.

$$BW_i = \begin{cases} 1, falls\ G_i > T \\ 0, falls\ G_i \leq T \end{cases} \quad (10)$$

Dabei kann T als **globaler Schwellwert** berechnet werden. In diesem Fall gilt ein Schwellwert für das gesamte Bild. Hierbei kann es jedoch zu den angesprochenen Problemen kommen, wenn Teile des Bildes durch ungleichmäßige Beleuchtung heller sind als andere. Wenn es nicht möglich ist, die Beleuchtungsunterschiede auszugleichen, ist es oft sinnvoll einen **lokalen Schwellwert** zu verwenden, der nur auf einen bestimmten Ausschnitt des Bildes angewendet wird und der daher robuster gegenüber Helligkeitsunterschieden ist. Allerdings ist ein lokaler Schwellwert deutlich rechenintensiver, da dieser für jeden Teilbereich des Bildes neu berechnet werden muss. Des Weiteren kommt es beim lokalen Schwellwert zu ungewolltem Rauschen, weil das Verfahren nicht zwischen homogenen und inhomogenen Regionen im Originalbild unterscheiden kann. Beispielsweise versucht das Verfahren auch eine Region, die komplett zum Hintergrund gehört, in Hintergrund und Vordergrund zu unterteilen. In Abschnitt 4.1.5 wird ein Verfahren vorgestellt, wie dieses Rauschen zumindest teilweise unterdrückt werden kann. Außerdem kann es beim lokalen Schwellwert zu Sprüngen an den Grenzen der Berechnungsregionen kommen.

4.1.5. Erosion und Dilatation

Bei der Erosion und Dilatation handelt es sich um morphologische Operatoren. Sie verändern die Form von Objekten. Werden die Operatoren genau in dieser Reihenfolge angewendet, können damit sehr effizient kleine Objekte und das Segmentierungsrauschen, wie es bei einem lokalen Schwellwert auftritt, herausgefiltert werden. Beide Operatoren lassen sich auf Binärbilder anwenden. Das heißt auf solche, deren Pixel nur die Werte Null oder Eins haben können. Eine Maske (Matrix) wird über das gesamte Bild geschoben und alle Pixelwerte unter der Maske werden mit einem logischen UND (Erosion) bzw. mit einem logischen ODER (Dilatation) verknüpft. Der Wert des zentral unter der Maske liegenden Pixels wird mit dem Ergebnis der logischen Operation ersetzt. Als Resultat werden Objekte entweder erodiert oder gestreckt. Da bei der Erosion alle Objekte, die kleiner sind als die Maske, herausgefiltert werden, eignet sich das Verfahren ideal zur Unterdrückung von Segmentierungsrauschen. Eine anschließende Dilatation stellt die übriggebliebenen Objekte in ihrer ursprünglichen Größe wieder her. Dabei ist zu beachten, dass die Operationen in dieser Reihenfolge zwar nur geringe Auswirkungen auf die Lage der Objekte haben, die Kantenformen können jedoch erheblich verändert werden. Die Größe der Faltungsmaske kann in Abhängigkeit vom Problem gewählt werden [6].

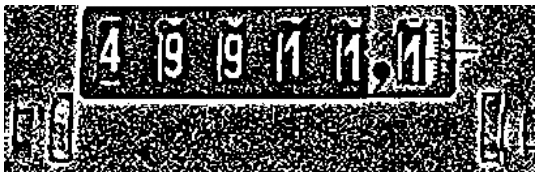


Abbildung 52: Eingangsbild

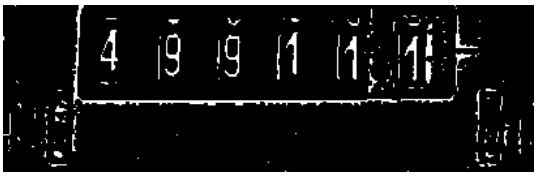


Abbildung 53: Bild nach Erosion

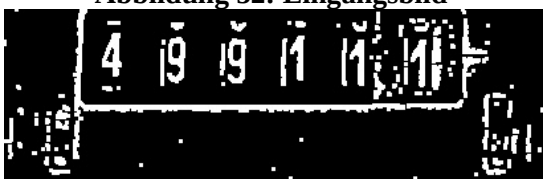


Abbildung 54: Bild nach Dilatation

4.1.6. Mustererkennung

In diesem Abschnitt werden einige grundlegende Sachverhalte der Mustererkennung dargestellt. In Abschnitt 4.1.9 wird sehr detailliert auf künstliche neuronale Netze, einen Bereich der Mustererkennung, eingegangen. Aufgabe der Mustererkennung ist es im Allgemeinen, ein Klassifikationsproblem zu lösen. Bei einem Klassifikationsproblem sollen Datensätze, z.B. Bilder oder Messdaten, in bestimmte Klassen eingeordnet bzw. klassifiziert werden. Der Klassifikator ist das Kernelement des Mustererkennungssystems. Dieser trifft die Entscheidung, welcher Datensatz in welche Klasse eingeordnet wird.

4.1.7. Prozessschritte

Es hat sich gezeigt, dass es wichtig ist, die rechnerische Komplexität eines Klassifikationssystems zu beachten. Die Komplexität der Probleme lässt sich an einem einfachen Beispiel verdeutlichen. Angenommen, ein Klassifikator in einem einfachen Klassifikationsproblem soll Binär-Bilder (Werte 0 und 1) der Größe 20×20 klassifizieren. Wird hierbei der triviale Ansatz gewählt und die Klassifikationsergebnisse zu allen Bildern in einer Lookup-Tabelle gespeichert, dann hätte diese Lookup-Tabelle $2^{20\times20} \approx 2{,}5 \cdot 10^{120}$ Einträge. Bei Grauwerten zwischen 0 und 255 oder gar RGB-Werten (Rot, Grün, Blau) als Eingangswerten, wäre die Komplexität des Problems noch ungleich höher.

Weil auch einfache Klassifikationsprobleme wie oben beschrieben eine hohe Komplexität aufweisen, gibt es zahlreiche Ansätze, um die Komplexität des Klassifikationsproblems und den Rechenaufwand zu reduzieren. Ein wichtiger Ansatz ist hierbei die Dimensionsreduktion, wobei zum Beispiel aus einer großen Anzahl von Rohwerten pro Trainingsmuster eine kleine Anzahl von Merkmalen berechnet werden kann. Die Dimension des Klassifikationsproblems wurde dann von der Anzahl der Rohwerte pro Trainingsmuster, auf die Anzahl der Merkmale pro Trainingsmuster reduziert. Dabei gilt es die Merkmale möglichst so zu wählen, dass die Klassifikation trotz des Informationsverlustes genauso bzw. nahezu so effektiv ist wie auf den Rohdaten. Ein weiterer Ansatz ist es, mehrere Klassifikatoren in Reihe zu schalten. Jeder Klassifikator kann dabei ein etwas einfacheres Klassifikationsproblem lösen. Während sich der Rechenaufwand für die Klassifikation mit einer Reihenschaltung von Klassifikatoren sehr gut reduzieren lässt, kann es jedoch zu einer Verschlechterung der Erkennungsrate des Gesamtsystems kommen, da die Erkennungsrate des Gesamtsystems niemals höher sein kann als die Erkennungsrate des schlechtesten Unter-Klassifikators. Aufgrund dieser Überlegungen wird die Mustererkennung oft in Prozessschritte unterteilt. Die drei wichtigsten Prozessschritte sind: Vorverarbeitung, Merkmalsextraktion und eigentliche Klassifikation.

Die **Vorverarbeitung** hat das grundlegende Ziel, die Daten für die Klassifikation aufzubereiten. Im Bereich der optischen Mustererkennung, in dem sich diese Arbeit bewegt, kann es sich beispielsweise um Bildverarbeitungsalgorithmen wie Farbraumtransformationen, Kompensation von Helligkeitsunterschieden oder die Rauschunterdrückung handeln.

Nachdem die Daten aufbereitet wurden, können aus ihnen die **Merkmale** berechnet werden. Welche Merkmale für eine Klassifikation sinnvoll sind, ist dabei immer problemabhängig. Es kann allerdings anhand der Merkmalsverteilung abgelesen werden, ob sich ein Merkmal gut oder schlecht dazu eignet, zwei Klassen voneinander zu trennen. Liegt das Merkmal beispielsweise in einer bimodalen Verteilung vor (siehe Abbildung 51), dann eignet es sich wahrscheinlich sehr gut, um zwei Klassen voneinander zu unterscheiden.

Zum Schluss werden die Merkmale auf den **Klassifikator** gegeben. Dieser trifft dann auf Grundlage der Merkmale die Entscheidung, zu welcher Klasse das Eingangsmuster gehört.

Als Klassifikator können beispielsweise neuronale Netze, Fuzzy Logik, Support Vektor Machines, Entscheidungsbäume oder selbstorganisierte Karten eingesetzt werden.

4.1.8. Lernen und Evaluierung

Es gibt zwar zahlreiche unterschiedliche Lernverfahren für statistische Klassifikatoren, allerdings basieren sehr viele auf dem Prinzip des Gradientenabstiegs. Die Lernverfahren lassen sich in drei Kategorien unterteilen:

- Beim **überwachten Lernen** gibt ein externer „Lehrer“ zu jedem Trainingsmuster die korrekte Ausgabe oder Kostenfunktion vor.
- Beim **bestärkenden Lernen** gibt der „Lehrer“ nur an, ob das Ergebnis vom Klassifikator korrekt oder inkorrekt ist. Er liefert jedoch keine Information über Betrag oder Richtung des Fehlers. Typischerweise wird vom Klassifikator zunächst die Klassenzugehörigkeit berechnet und dann die bekannte Klassenzugehörigkeit verwendet, um den Klassifikator anzupassen.
- Beim **unüberwachten Lernen** oder Clustering bekommt der Klassifikator gar keine Information über die Korrektheit des Ergebnisses. Das System unterteilt die Daten von selbst in Klassen. Meist ist dabei jedoch die Anzahl der Klassen vorgegeben.

Ziel eines Klassifikators ist es, die Aposteriori-Wahrscheinlichkeit $P(w|x)$, dass ein Testmuster x zur Klasse w gehört, aus einer Menge von Trainingsdaten möglichst genau zu rekonstruieren. In der Praxis hat sich gezeigt, dass die effektivsten Klassifikatoren mit realen Testdaten trainiert werden und nicht alleine auf Grundlage von Annahmen arbeiten [7]. Im Allgemeinen lässt sich festhalten, dass eine größere Zahl von Trainingsdaten auch zu einer effektiveren und besseren Klassifikation führt.

Ein wichtiges Problem im Bereich der Mustererkennung ist das **Overfitting**. Dabei klassifiziert ein übermäßig komplexer Klassifikator die Testmuster perfekt. Allerdings ist die Klassifikation auf neuen Daten deutlich schlechter. In diesem Fall hat der Klassifikator die Trainingsdaten auswendig gelernt, was sich zu Lasten der Generalisierungseigenschaften auswirkt (siehe Abbildung 55). Obwohl dieses Problem bekannt ist, kann es in der Praxis schwierig sein, Overfitting zu erkennen und zu verhindern. Ein allgemeiner Ansatz ist es, die Rohdaten in Trainings- und Validierungsdaten zu unterteilen. Während der Klassifikator mit den Trainingsdaten trainiert wird, werden die Validierungsdaten nicht zum Training genutzt und dienen dazu, die Generalisierungsfähigkeit des Klassifikators zu beobachten. Es kann keine allgemeine Aussage getroffen werden, welcher Anteil der Daten zur Validierung genutzt werden sollte. Allerdings lässt sich sagen, dass ein höherer Anteil von Trainingsdaten zu einem besseren Klassifikator führt, während ein höherer Anteil an Validierungsdaten eine bessere Abschätzung der Erkennungsrate erlaubt [8].

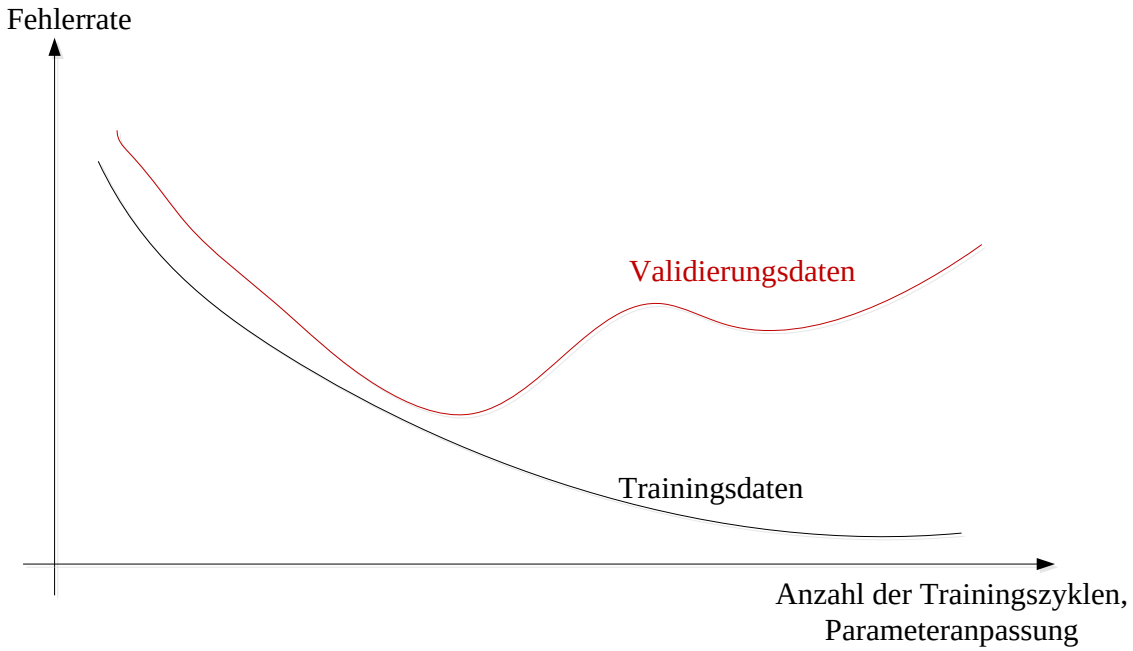


Abbildung 55: Overfitting eines Klassifikators

4.1.9. Künstliche neuronale Netze

Künstliche neuronale Netze orientieren sich an der Neuronen-Struktur im menschlichen Gehirn, um komplexe Algorithmen und statistische Zusammenhänge abzubilden. Seitdem neuronale Netze in den 40er Jahren zum ersten Mal als technisches System vorgeschlagen wurden, gab es sehr viele verschiedene Vorschläge über die Topologie von neuronalen Netzen. In dieser Arbeit wird vorwiegend das Multilayer Perceptron (MLP) verwendet. Das MLP ist das am häufigsten eingesetzte künstliche neuronale Netz, weil es sich sehr gut in verschiedensten Problemstellungen einsetzen lässt [9, 10].

4.1.10. Künstliche Neuronen

Im Laufe der Jahre wurden sehr viele unterschiedliche Modelle von Neuronen vorgestellt. Meist sind diese stark idealisiert. Die Grundidee ist häufig, ein möglichst einfaches Neuron als Grundelement zu verwenden. Die Stärke des neuronalen Netzes wird erst durch das gesamte Netz bzw. durch die Gewichte zwischen den Neuronen erreicht.

Zunächst besitzt ein Neuron n Eingänge. Diese sind meist Ausgänge anderer Neuronen. Die Eingänge lassen sich als Vektor $X = (x_1,\ldots,x_n)$ darstellen. Jeder der n Eingänge ist über eine gewichtete Verbindung mit dem aktuellen Neuron verbunden. Die Gewichte sind durch den Gewichtsvektor $W = (w_1,\ldots,w_n)$ repräsentiert. Die Aktivierungsfunktion $f_a(X,W)$ eines Neurons berechnet aus den gewichteten Eingängen die Aktivierung des Neurons. Aus der Aktivierung wird dann die Ausgabe o über die Ausgabefunktion $o = f_o(f_a(X,W))$ bestimmt. Ein einzelnes Neuron lässt sich wie in Abbildung 56 darstellen [9].

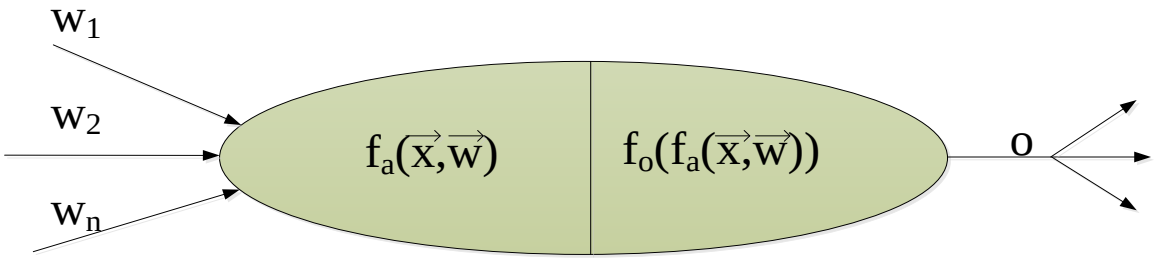


Abbildung 56: Darstellung eines künstlichen Neurons

Beim biologischen Vorbild der künstlichen Neuronen muss zur Auslösung des Aktionspotentials ein bestimmter Schwellwert überschritten werden. Als Ausgabefunktion würde daher grundsätzlich eine binäre Schwellwertfunktion (siehe Abbildung 57) genügen. Um später einen Lernalgorithmus auf das neuronale Netz anwenden zu können, eignen sich jedoch vor allem Ausgabefunktionen, die stetig differenzierbar und beschränkt sind [9]. Am häufigsten werden s-förmige (sigmoide) Funktionen verwendet wie sie in Formel (11) und (12) sowie in Abbildung 58 zu finden sind.

$$f_o(f_a) = \frac{1}{1 + e^{-x}} \tag{11}$$

$$f_o(f_a) = \tanh(x) \tag{12}$$

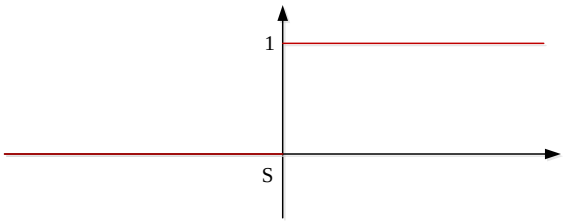


Abbildung 57: Binäre Schwellwertfunktion

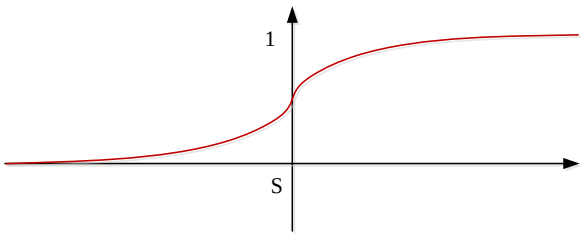


Abbildung 58: Sigmoide Schwellwertfunktion

Wird der Schwellwert als innere Eigenschaft des Neurons definiert, dann kann er entweder nicht trainiert oder muss gesondert vom Lernalgorithmus betrachtet werden. Daher bietet es sich an, den Schwellwert als einen weiteren Eingang zu realisieren, dessen Gewicht wie alle anderen Gewichte trainiert werden kann, was zu einem einheitlichen Lernalgorithmus führt [9].

4.1.11. Struktur und Vernetzung der Neuronen

Neuronale Netze bestehen aus zwei Grundelementen: Neuronen und Verbindungen zwischen diesen Neuronen. Jedes Neuron kann im Allgemeinen beliebig viele Eingänge besitzen und erzeugt aus diesen einen Ausgangswert. Über beliebig viele Verbindungen kann das Neuron den Ausgangswert zu anderen Neuronen weiterleiten. Die Neuronen sind meist in Schichten

oder Ebenen angeordnet. Dabei gibt es eine Eingangsschicht, welche mit Daten von außen beschrieben wird und eine Ausgangsschicht, welche die Ergebnisdaten des neuronalen Netzes nach außen weitergibt. Neben der Eingangs- und Ausgangsschicht kann ein neuronales Netz eine beliebige Anzahl von Verbindungsschichten (Hidden-Layer) enthalten. Eine grobe Einteilung von neuronalen Netzen lässt sich erreichen, wenn man diese nach der Art der Verbindungen zwischen den Neuronen klassifiziert.

Einerseits gibt es **Feedforward-Netze** ohne Rückkopplungen, nur mit Verbindungen von einer niedrigeren zu einer höheren Schicht. Ein Feedforward-Netz wird auch als **Multilayer Perceptron** bezeichnet [10]. Es gibt drei Ausführungen von Feedforward-Netzen [9]:

- Ebenenweise verbundene Feedforward-Netze: Es bestehen nur Verbindungen zwischen einer Ebene und der nächst höheren Nachbarebene.
- Allgemeine Feedforward-Netze: Ebenen dürfen übersprungen werden. Verbindungen zwischen nicht benachbarten werden Shortcuts genannt.
- Total verbundene Feedforward-Netze: Jedes Neuron einer Ebene ist mit jedem Neuron der nächst höheren Nachbarebene verbunden.

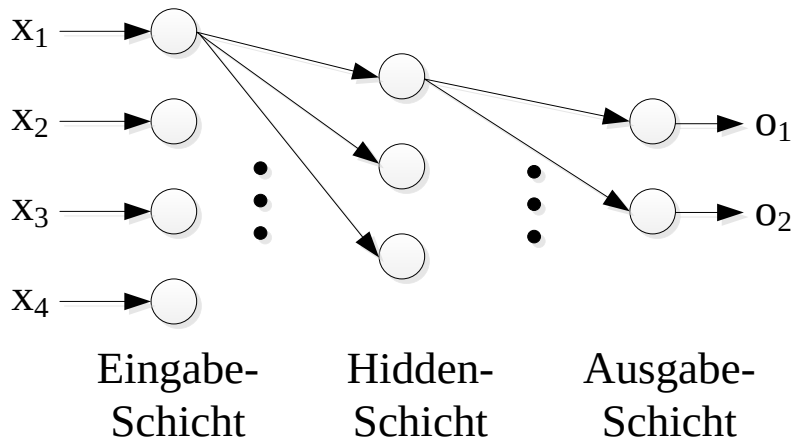


Abbildung 59: Total verbundenes Feedforward-Netz mit 3 Ebenen

Abbildung 59 zeigt ein typisches Multilayer Perceptron. Andererseits gibt es **Netze mit Rückkopplungen** (rekurrente Netze). Sie lassen sich wie folgt unterteilen [9] :

- Direct Feedback-Netze: Es ist erlaubt den Ausgang eines Neurons wieder auf dem Eingang desselben Neurons zurückzuführen. Als Resultat nehmen die Neuronen oft die Grenzzustände der Ausgabefunktion an, da sie sich selbst verstärken bzw. hemmen.

- Indirect Feedback-Netz: Es gibt nur Verbindung von höheren Ebenen zu niedrigeren Ebenen, was dazu führt, dass bestimmte Bereiche oder Eingangsneuronen besonders beachtet werden.
- Lateral Feedback-Netze: Es gibt Rückkopplungen innerhalb einer Schicht. Damit ist es möglich, dass nur ein einzelnes Neuron aus einer Gruppe aktiv wird. Das stärkste Neuron hemmt dann alle anderen Neuronen („winner takes all“).
- Vollständig verbundene Netze: Es gibt Verbindungen von jedem Neuron zu allen anderen Neuronen, allerdings ist kein Neuron mit sich selbst verbunden. Diese Netze werden auch als Hopfield-Netze bezeichnet.

Neben der Anzahl und Art der Verbindungen zwischen den Neuronen eines Netzes, müssen auch Designentscheidungen über die Anzahl der Ebenen des Netzes und die Anzahl der künstlichen Neuronen pro Ebene getroffen werden. Während die Anzahl der Eingangsneuronen durch die Anzahl der Merkmale und die Anzahl der Ausgangsneuronen durch die Anzahl der Klassen festgelegt wird, kann die Anzahl der Neuronen pro Hidden-Schicht (Hidden-Neuronen) frei gewählt werden. Dabei hat die Anzahl der Hidden-Neuronen direkten Einfluss auf die Anzahl der Gewichte im Netz und bestimmt somit die Stärke des Netzes. Bei einfachen Problemen mit gut separierten Mustern genügen wenige Hidden-Neuronen, während bei schwierigen Problemen und schwer separierbaren Mustern viele Hidden-Neuronen gewählt werden sollten. Wenn für ein gegebenes Problem zu viele freie Parameter gewählt wurden, führt dies zu einer schlechten Generalisierung (vgl. Overfitting, Abschnitt 4.1.8). Wurden jedoch zu wenige freie Parameter gewählt kann die Trainingsmenge nicht mehr gelernt werden [8].

Es lässt sich zeigen, dass jede kontinuierliche Funktion zwischen Eingang und Ausgang des neuronales Netzes durch ein dreilagiges neuronales Netz abgebildet werden kann. Allerdings können mehr als drei Ebenen zu besseren Generalisierungseigenschaften bezüglich Translation und Rotation der Testmuster führen [8, 9]. Das in Abschnitt 4.1.12 vorgestellte Verfahren der Backpropagation lässt sich grundsätzlich auf Netze mit beliebig vielen Ebenen anwenden. Bei steigender Anzahl der Ebenen, steigt jedoch auch die Anzahl der Dimensionen der Fehleroberfläche, was ein optimales Training erschwert.

4.1.12. Lernen/Trainieren von neuronalen Netzen

Es gibt viele verschiedene Vorschläge, wie neuronale Netze trainiert bzw. angelernt werden. Die aktuell am häufigsten angewandte Lernregel ist die **Backpropagation** (BP) [9]. Allerdings gibt es auch hier wieder zahlreiche Anpassungs- und Verbesserungsvorschläge. Die Backpropagation ist eine überwachte Lernregel. Sie basiert auf dem Prinzip des Gradientenabstiegs. Dabei werden die Gewichte $\vec{w}$ in mehreren Iterationen derart angepasst,

dass eine vorher definierte Fehlermetrik $F(\vec{w})$ ein Minimum erreicht. In jeder Iteration werden die Gewichte dabei in Richtung des negativen Gradienten der Fehlermetrik nach Formel (13) verändert.

$$\vec{w}^{neu} = \vec{w}^{alt} - \eta \nabla F(\vec{w}) \tag{13}$$

Im Folgenden soll gezeigt werden, wie sich $\nabla F(\vec{w})$ berechnen lässt. Die Elemente des Nabla-Operators sind die partiellen Ableitungen, wie Formel (14) zeigt. Folglich lässt sich aus Formel (13), welche die Veränderung des gesamten Vektors $\vec{w}$ anzeigt, Formel (15) für jedes $w_{h,i,j}$ erzeugen. Dabei steht der Index *h* für die Ebene des Gewichtes, der Index *i* für das Zielneuron und der Index *j* für das Ausgangsneuron des Gewichtes. In der Folge wird die Berechnung der Gewichtsanpassung für ein beliebiges Gewicht $w_{h,i,j}$ dargestellt. In Abbildung 60 findet sich hierzu eine Übersicht über die Notation von Ebenen, Neuronen und Gewichten. Zu Vereinfachung wurde nur ein Neuron pro Ebene dargestellt.

$$\nabla = \left(\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, \ldots \frac{\partial}{\partial w_{n-1}}, \frac{\partial}{\partial w_n}\right) \tag{14}$$

$$w_{h,i,j}^{neu} = w_{h,i,j}^{alt} - \eta \cdot \frac{\partial F(\vec{w})}{\partial w_{h,i,j}} \tag{15}$$

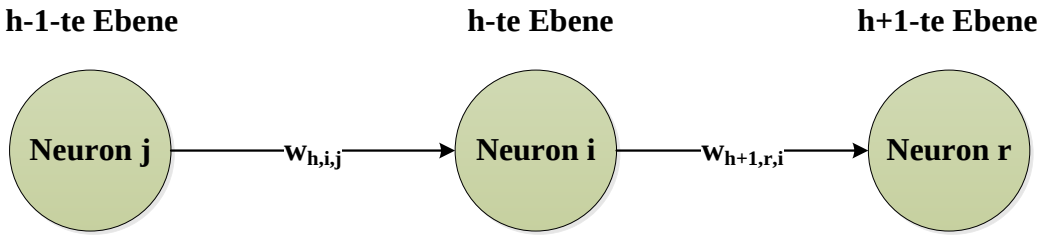


Abbildung 60: Notation bei Ebenen, Neuronen und Gewichten

Ausgehend von Formel (15) lässt sich die Gewichtsanpassung pro Gewicht berechnen. Um die Rechnung etwas zu vereinfachen, soll zur Gewichtsanpassung nur ein einziges Testmuster genutzt werden. Ausgehend von der Gewichtsanpassung für ein Testmuster, lässt sich das Ergebnis der Rechnung sehr einfach für eine beliebige Anzahl von Testmustern anpassen. Da der Fehler *F*, der sich am Ausgang des neuronalen Netzes ergibt, von der Aktivität $A_{h,i}$ des *i*-ten Neurons der *h*-ten Ebene abhängt, kann die Ableitung des Fehlers nach dem Gewicht $w_{h,i,j}$ wie in Formel (16) umgeschrieben werden. Des Weiteren lässt sich für die Ableitung des Fehlers nach der Aktivität $A_{h,i}$ als $\delta_{h,i}$ notieren, wie Formel (17) zeigt. Dieses $\delta_{h,i}$ lässt sich als Fehler am *i*-ten Neuron der *h*-ten Ebene interpretieren. Der zweite Faktor der Formel (16) entspricht der Ableitung der Aktivität $A_{h,i}$ nach dem Gewicht $w_{h,i,j}$. Beim Berechnen

dieser Ableitung ergibt sich der Ausgang $o_{h-1,j}$ des *j*-ten Neurons der *h-1*-ten Ebene, wie Formel (18) zeigt. Dabei ist $\#(U_{h-1})$ die Anzahl der Neuronen in der *h-1*-ten Ebene.

$$\frac{\partial F(\vec{w})}{\partial w_{h,i,j}} = \frac{\partial F(\vec{w})}{\partial A_{h,i}} \cdot \frac{\partial A_{h,i}}{\partial w_{h,i,j}} = \delta_{h,i} \cdot o_{h-1,j} \tag{16}$$

$$\delta_{h,i} = \frac{\partial F(\vec{w})}{\partial A_{h,i}} \tag{17}$$

$$\frac{\partial A_{h,i}}{\partial w_{h,i,j}} = \frac{\partial}{\partial w_{h,i,j}} \sum_{r=0}^{\#(U_{h-1})} o_{h-1,r} \cdot w_{h,i,r} = o_{h-1,j} \tag{18}$$

Nun lässt sich das Ergebnis aus Formel (16) in Formel (15) einsetzen, was für die Anpassung des Gewichtes $w_{h,i,j}$ zu Formel (19) führt. Die Gewichtsänderung des Gewichtes $w_{h,i,j}$ zwischen dem *i*-ten Neuron der *h*-ten Ebene und dem *j*-ten Neuron der *h-1*-ten Ebene ergibt sich aus der Lernrate η, dem Ausgabewert $o_{h-1,j}$ (des *j*-ten Neurons der *h-1*-ten Ebene) und dem Fehler $\delta_{h,i}$ (am *i*-ten Neuron der *h*-ten Ebene).

$$w_{h,i,j}^{neu} = w_{h,i,j}^{alt} - \eta \cdot \delta_{h,i} \cdot o_{h-1,j} \tag{19}$$

Mit Gleichung (19) könnte die Berechnung der Gewichtsänderung als abgeschlossen betrachtet werden. Bei der Lernrate η handelt es sich um eine vor dem Training festgelegte Konstante, bei dem Gewicht $w_{h,i,j}^{alt}$ um einen gegebenen Parameter des neuronalen Netzes und bei dem Ausgabewert $o_{h-1,j}$ um einen Wert, der sich aus dem Parametern des neuronalen Netzes und dem Testmuster ergibt. Allerdings wurde noch nicht gezeigt, wie sich $\delta_{h,i}$ berechnet, was im Folgenden betrachtet werden soll. Dabei müssen zwei Fälle unterschieden werden, abhängig von der Ebene in der $\delta_{h,i}$ berechnet werden soll. Im ersten Fall wird die Ausgabeschicht betrachtet und es gilt *h = H-1*, wobei H die Anzahl der Ebenen im neuronalen Netz ist. In diesem Fall ist die Ausgabefunktion f_a die Indentitätsfunktion und es gelten daher die Formeln (20) und (21). Der Fehler $\delta_{h,i}$ lässt sich für die Ausgangssicht somit nach Formel (22) berechnen. Dabei wurde als Fehlermetrik am Ausgang der quadratische Fehler verwendet. Außerdem ist y_i der Soll-Ausgangswert am *i*-ten Ausgangsneuron und $o_{h,i}$ der Ist-Ausgangswert am *i*-ten Ausgangsneuron.

Fall1:

(h=H-1)

$$o_{h,i} = f_a(A_{h,i}) = A_{h,i} \tag{20}$$

$$\delta_{h,i} = \frac{\partial F(\vec{w})}{\partial A_{h,i}} = \frac{\partial F(\vec{w})}{\partial o_{h,i}} \tag{21}$$

$$\delta_{h,i} = \frac{\partial}{\partial o_{h,i}} \sum_{r=0}^{\#(U_{H-1})} (y_r - o_{h,r})^2 = -2(y_i - o_{h,i}) \tag{22}$$

Für alle anderen Ebenen des neuronalen Netzes mit $h \neq H\text{-}1$ gilt Formel (20) jedoch nicht, weil die Ausgabefunktion f_a nicht die Identitätsfunktion, sondern eine sigmoide Funktion ist. In diesem Fall lässt sich Formel (23) nicht weiter vereinfachen und der Fehler $\delta_{h,i}$ ergibt sich nach Formel (24).

Fall 2:

($h \neq H\text{-}1$)

$$o_{h,i} = f_a(A_{h,i}) \tag{23}$$

$$\delta_{h,i} = \frac{\partial F(\vec{w})}{\partial A_{h,i}} = \frac{\partial F(\vec{w})}{\partial o_{h,i}} \cdot \frac{\partial o_{h,i}}{\partial A_{h,i}} = \frac{\partial F(\vec{w})}{\partial o_{h,i}} \cdot f_a{}'(A_{h,i}) \tag{24}$$

Der Faktor $f_a{}'(A_{h,i})$ in Formel (24) ist die Ableitung der Ausgabefunktion f_a nach der Aktivität. Während die Wahl von f_a vor dem Training getroffen wird, ergibt sich die Aktivität aus den Parametern des neuronalen Netzes und dem Eingangsmuster. Daher bleibt nur noch der erste Faktor in Formel (24) zu berechnen. Hierbei sei noch einmal auf Abbildung 60 verwiesen, denn die Aktivität $A_{h+1,r}$ des *r*-ten Neurons der *h+1*-ten Ebene entsteht unter anderem durch $o_{h,i}$, den Ausgangswert am *i*-ten Neuron der *h*-ten Ebene. Somit trägt $o_{h,i}$ auch zum Fehler $\delta_{h+1,r}$ von $A_{h+1,r}$ bei. Der Faktor, mit dem $o_{h,i}$ zu $\delta_{h+1,r}$ beiträgt, lässt sich durch die Ableitung in Formel (26) bestimmen. Wie zu erwarten war, beträgt der Faktor $w_{h+1,r,i}$ exakt das Gewicht zwischen den beiden Neuronen.

$$\frac{\partial F(\vec{w})}{\partial o_{h,i}} = \sum_{r=0}^{\#(U_{H+1})} \frac{\partial F(\vec{w})}{\partial A_{h+1,r}} \cdot \frac{\partial A_{h+1,r}}{\partial o_{h,i}} = \sum_{r=0}^{\#(U_{H+1})} \delta_{h+1,r} \cdot w_{h+1,r,i} \tag{25}$$

$$\frac{\partial A_{h+1,r}}{\partial o_{h,i}} = \frac{\partial}{\partial o_{h,i}} \sum_{i2=0}^{\#(U_h)} o_{h,i1} \cdot w_{h+1,r,i2} = w_{h+1,r,i} \tag{26}$$

Wird das Ergebnis aus Formel (25) in Formel (24) eingesetzt, so ergibt sich Formel (27). Der Fehler $\delta_{h,i}$ am *i*-ten Neuron der *h*-ten Ebene, ergibt sich also aus der Ableitung der Ausgabefunktion f_a nach der Aktivierung $A_{h,i}$ und der gewichteten Summe der Fehler $\delta_{h+1,r}$ der *r*-ten Neuronen in der *h+1*-ten Ebene. Aufgrund dieses Zusammenhangs wurde das gesamte Verfahren Backpropagation genannt, denn der Fehler wird bei der Ausgabeschicht beginnend, von der Schicht *h+1* zur vorherigen Schicht *h* zurückgeleitet.

$$\delta_{h,i} = \frac{\partial F(\vec{w})}{\partial o_{h,i}} \cdot \frac{\partial o_{h,i}}{\partial A_{h,i}} = f_a{}'(A_{h,i}) \cdot \sum_{r=0}^{\#(U_{H+1})} \delta_{h+1,r} \cdot w_{h+1,r,i} \tag{27}$$

In Abhängigkeit vom Index der Schicht kann also die Formel (27) oder die Formel (22) für $\delta_{h,i}$ in Formel (19) eingesetzt und die Gewichtsanpassung berechnet werden. Der Vollständigkeit halber soll nochmal erwähnt werden, dass bisher nur die Gewichtsanpassung für ein Testmuster berechnet wurde. Wie bereits angedeutet, lässt sich die Rechnung aber leicht für eine beliebige Anzahl von Testmustern erweitern, wie Formel (28) verdeutlicht. Es wird zunächst für jedes Testmuster die Gewichtsanpassung berechnet und später der Mittelwert gebildet, nachdem dem neuronalen Netz alle Testmuster präsentiert wurden. Dieser Mittelwert aller Gewichtsanpassungen entspricht dann einer Gewichtsanpassung, um welche das Gewicht $w_{h,i,j}$ tatsächlich verändert wird. Dabei ist P die Anzahl der Testmuster, $\delta_{h,i}^{p}$ der Fehler am *i*-ten Neuron der *h*-ten Ebene beim *p*-ten Testmuster und $o_{h-1,j}^{p}$ der Ausgang des *j*-ten Neurons der *h*-1-ten Ebene beim *p*-ten Testmuster. Weitere ausführliche Darstellungen finden sich in [11] sowie in [12].

$$w_{h,i,j}^{neu} = w_{h,i,j}^{alt} - \eta \cdot \frac{1}{P} \cdot \sum_{p=1}^{P} \delta_{h,i}^{p} \cdot o_{h-1,j}^{p} \tag{28}$$

In jeder Iteration der Backpropagation wird jedes Testmuster einmal präsentiert. Pro Testmuster gibt es also drei Phasen: den Forward-Pass, die Bestimmung des Fehlers und den Backward-Pass. Beim Forward-Pass wird das Trainingsmuster auf das neuronale Netz gegeben und die entsprechenden Werte in der Ausgangsschicht bestimmt. Dann wird der Fehler über eine Metrik berechnet und danach beim Backward-Pass der am Ausgang entstandene Fehler an alle anderen Neuronen weitergegeben, also rückpropagiert. Nachdem alle Testmuster dem neuronalen Netz präsentiert wurden, werden die Gewichte angepasst. Als Metrik zur Fehlerbestimmung wird meist der in Formel (29) dargestellte quadratische Fehler gewählt. Dabei ist $F_i(\vec{w})$ der Fehler am i-ten Ausgangsneuron, o_i der Ist-Wert der i-ten Ausgangszelle und t_i der Soll-Wert der i-ten Ausgangszelle. Alternativ kann allerdings auch die Kreuzentropie als Fehlermetrik Verwendung finden [8].

$$F_i(\vec{w}) = (o_i - t_i)^2 \tag{29}$$

Beim Verfahren der Backpropagation, kann es zu den üblichen Problemen eines Gradientenabstiegs-Verfahrens kommen. Ein Problem bilden zum Beispiel lokale Minima (vgl. Abbildung 61). Der Gradientenabstieg kann im Allgemeinen immer an lokalen Minima hängenbleiben. Lokale Minima bekommen eine wachsende Bedeutung bei neuronalen Netzen mit vielen Gewichten, da die Dimension der Fehleroberfläche mit der Anzahl der Gewichte im Netz steigt und diese auch immer zerklüfteter wird. Zerklüftet bedeutet hier, dass die Fehleroberfläche mit wachsender Dimension immer größer wird und daher mehr lokale Minima aufweist und sich der Gradient durch den Einfluss von immer mehr Parametern immer sprunghafter ändert, wodurch das Training erschwert wird. Ein spezielles Problem von neuronalen Netzen ist, dass die Gewichte immer zufällig initialisiert werden müssen.

Insbesondere dürfen die Gewichte niemals mit Null initialisiert werden, da sonst kein Training zustande kommt [8]. Aber auch das Initialisieren aller Gewichte auf denselben, von Null verschiedenen Wert, ist wenig ratsam, da sonst alle Gewichte zwischen zwei Schichten immer gleich sind [9]. Ein weiteres Problem sind flache Plateaus der Fehlerebene (vgl. Abbildung 62). Die Gewichtsänderung hängt vom Betrag des Gradienten ab und daher stagniert die Backpropagation auf flachen Plateaus oder endet im schlechtesten Fall sogar auf diesen. Es handelt sich um ein grundsätzliches Problem, dass flache Plateaus sich nicht von Minima unterscheiden lassen. Des Weiteren kann es zur Oszillation kommen. In diesem Fall springt das BP-Verfahren nach einer bestimmten Anzahl von Iterationen wieder zurück zum Ausgangspunkt und das Verfahren erreicht keine Verbesserung, obwohl der Betrag des Gradienten niemals Null wird. Außerdem kann es zum Verlassen guter Minima kommen, wenn das Minimum in einem steilem Tal liegt und die relativ gute Region mit einem großen Sprung verlassen wird [9]. In Abschnitt 4.1.14 werden einige Anpassungen vorgestellt, mit denen die oben genannten Probleme zumindest teilweise kompensiert werden.

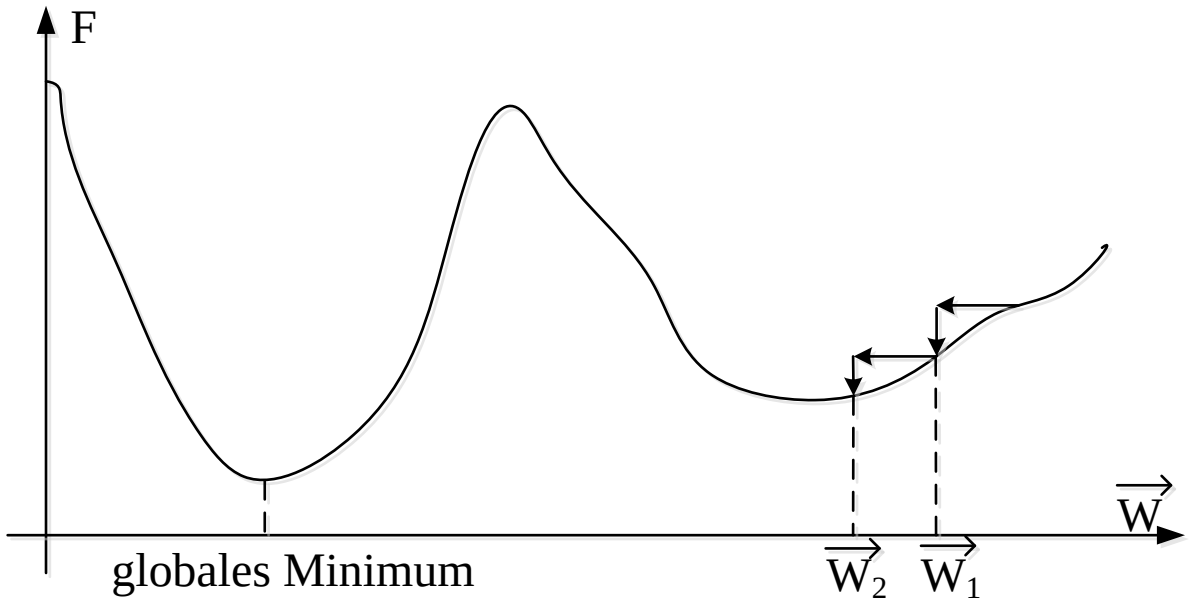


Abbildung 61: Gradientenabstieg endet in lokalem Minimum

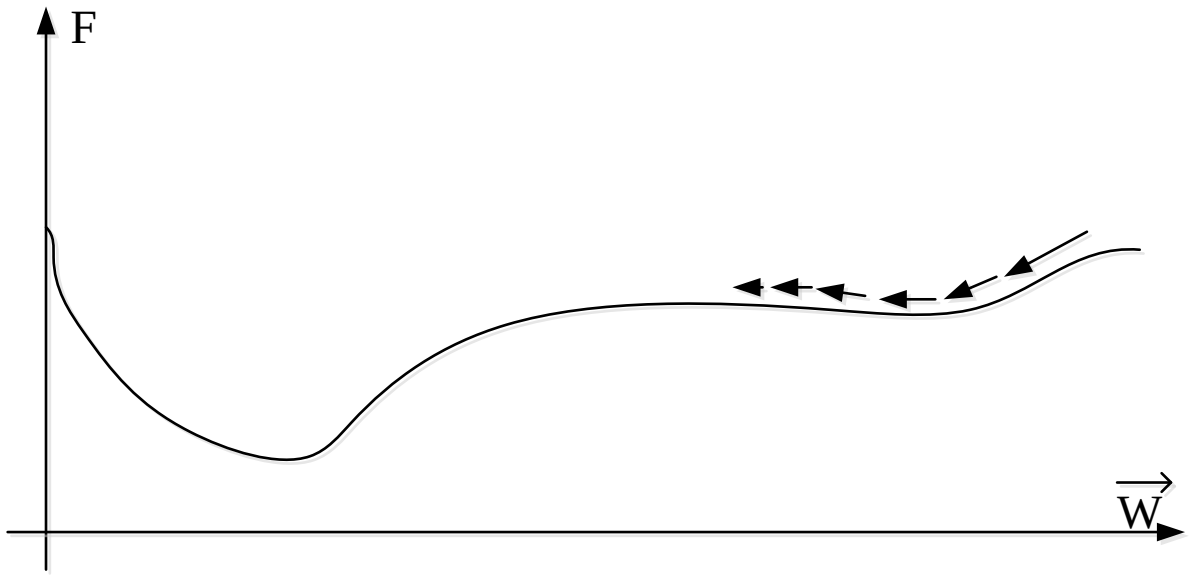


Abbildung 62: Gradientenabstieg endet auf flachem Plateau

4.1.13. Trainingsdaten

In der Mustererkennung hat sich gezeigt, dass die besten Klassifikatoren mit realen Daten trainiert und nicht lediglich auf Grundlage von Annahmen der Entwickler konzipiert wurden [8]. Wichtig ist dabei, dass die Trainingsdaten möglichst exakt den Daten entsprechen, mit denen das neuronale Netz auch später im praktischen Einsatz arbeiten muss. Des Weiteren gilt im Allgemeinen, dass sich durch mehr Trainingsdaten auch bessere Erkennungsraten erzielen lassen [8].

Bei der Vorverarbeitung der Trainingsdaten ist zu beachten, dass die Eingangswerte für jedes Merkmal mittelwertfrei sind und alle Merkmale dieselbe Varianz aufweisen. Beides ist wichtig, damit nicht bestimmte Merkmale präferiert werden, sondern alle Merkmale in den Erkennungsprozess einbezogen und somit möglichst viel Information in das neuronale Netz integriert werden kann. Wenn die Testbilder direkt auf das neuronale Netz gegeben werden, müssen beispielsweise für jeden Pixel über alle Trainingsbilder der Mittelwert und die Varianz berechnet werden. Mit diesen Werten werden die Pixelwerte dann skaliert, bevor sie auf das neuronale Netz gegeben werden [8].

$$X_{neu} = \frac{X_{alt} - Mittelwert}{Varianz} \tag{30}$$

Weiterhin sollten die Werte der Ausgangsneuronen +1 oder -1 betragen, um die Klassenzugehörigkeit jedes Testmusters anzuzeigen. Das Neuron, zu dessen Klasse das Testmuster gehört soll den Wert +1 haben, die Neuronen aller anderen Klassen sollten den

Wert -1 anzeigen [8]. Wenn nicht genügend reale Trainingsdaten zur Verfügung stehen, besteht die Möglichkeit, Trainingsdaten auf verschiedenen Wegen künstlich zu generieren. Einerseits ist es möglich idealisierte Daten mit Rauschen zu überlagern, andererseits können bestehende Muster rotiert oder bei der Schrifterkennung z.B. die Schriftdicke variiert werden. Solche Anpassungen sind sinnvoll wenn bekannt ist, dass später auch derart verformte Buchstaben oder Bilder erkannt werden sollen.

4.1.14. Verbesserungen von neuronalen Netzen

Wie bereits in Abschnitt 4.1.12 erwähnt wurde, gibt es viele Vorschläge für Anpassungen und Verbesserungen von neuronalen Netzen, und insbesondere für das Lernverfahren der Backpropagation. In diesem Abschnitt sollen die wichtigsten Anpassungen vorgestellt werden.

Die wichtigste Modifikation ist die Einführung eines **Momentum**-Terms in der BP-Lernregel. Die Grundidee ist hierbei, dem Verfahren des Gradientenabstiegs ein Trägheitsmoment zu verleihen. Damit soll sich das Verfahren auch auf flachen Plateaus noch hinreichend gut verbessern. Die Gewichtsveränderung pro Iterationsschritt wird wie in Formel (31) modifiziert. Dabei ist F das Fehlermaß, $\alpha \in [0,1[$ der Momentum-Term und η die Lernrate.

$$w_i(t+1) = w_i(t) + \Delta w_i(t) \tag{31}$$

$$\Delta w_i(t) = -(1-\alpha) \cdot \eta \cdot \frac{\partial F(t)}{\partial w_i(t)} + \alpha \cdot \Delta w_i(t-1) \tag{32}$$

Es wird die Annahme getroffen, dass auf einem Plateau der Fehleroberfläche viele Gewichtsänderungen das gleiche Vorzeichen haben, während es in einer Senke viele Gewichtsänderungen mit unterschiedlichem Vorzeichen gibt. Auf einem Plateau würde der Term $\alpha * \Delta w_i(t-1)$ folglich dazu führen, dass der Fehler wie von einem Trägheitsmoment weitergetragen wird, während der Term in einer Senke verschwindet, weil sich die Summanden des Terms gegenseitig aufheben. Ein Problem der Momentum-Modifikation ist, dass das Trägheitsmoment nach oben hin beschränkt ist. Auf Plateaus ist dies aber nicht erwünscht. Weiterhin kann es dazu kommen, dass das Trägheitsmoment nach Betrag größer ist als die Gewichtsanpassung nach der ursprünglichen BP-Lernregel. Wenn dann beide Terme nicht das gleiche Vorzeichen haben, wird w_i in Richtung des Gradienten verschoben. Der Fehler wird also größer. Aus diesem Grund ist die BP mit Momentum-Modifikation kein reines Gradientenabstiegsverfahren mehr. Außerdem ist der Betrag des Momentum-Terms α problemabhängig und muss experimentell bestimmt werden. Die Momentum-Modifikation kann auch als Tiefpassfilterung interpretiert werden [8].

Weiterhin kann durch **Weight Decay** eine Verbesserung erzielt werden. Weight Decay beruht auf der Annahme, dass nach Betrag große Gewichte eher schlecht sind, da sie eine steile und zerklüftete Fehleroberfläche erzeugen. Dies erhöht die Häufigkeit von Oszillationen und führt

zur Zunahme von unkontrollierten Sprüngen. Kleine Gewichte hingegen führen zu besseren Generalisierungseigenschaften des Netzes. Außerdem wird die Initialisierung der Gewichte weniger wichtig. Die Größe der Gewichte wird durch einen neuen Term in der Fehlerfunktion berücksichtigt, wie in Formel (33) dargestellt ist.

$$F_{neu} = F + \frac{d}{2}\sum_i w_i^2 \tag{33}$$

Dabei wird für *d* meist ein Wert zwischen 0,005 und 0,03 gewählt. Wenn ein zu großes *d* gewählt wird, dann kann es dazu kommen, dass alle Gewichte klein gehalten werden [9].

4.2. Konzept

Wie in Abschnitt 4.1.6 beschrieben, bestehen Mustererkennungssysteme häufig aus mehreren Stufen. Dabei dienen die einzelnen Stufen dazu, das Klassifikationsproblem bezüglich des Rechenaufwandes möglichst effizient lösen zu können. Abbildung 63 zeigt den Programmablaufplan der Software. Das vorgestellte System wurde in mehrere Stufen unterteilt:

- Bilderfassung: Zunächst wird ein Bild vom Stromzähler eingelesen. Dabei wird entweder ein neues Bild mit der Kamera aufgenommen oder eine vorhandene Bilddatei von der Festplatte geladen.
- Vorverarbeitung: Wichtigste Aufgabe der Vorverarbeitung ist es, den mittleren Bereich des Bildes auszuschneiden und somit die Rechenzeit des Algorithmus zu verkürzen.
- Detektion der Regions of Interest: In dieser Stufe werden die ROIs im Bild detektiert. Bei den ROIs handelt es sich um Bereiche, in denen sich der Zählerstand des Stromzählers befinden könnte. Die Detektion der ROIs ist wichtig, um die späteren Phasen der Segmentierung, die Suche nach Zeichenketten und die Ziffernerkennung effizienter gestalten zu können. Es wurden verschiedene Algorithmen zur Detektion der ROIs implementiert und miteinander verglichen.
- Segmentierung: Die ROIs werden einzeln weiterverarbeitet, wobei sie zunächst segmentiert werden. Zur Segmentierung kommt ein Schwellwert zum Einsatz. Die Segmentierung kann dabei genauer erfolgen, weil der Schwellwert nicht auf das gesamte Bild, sondern lediglich auf das ROI angewendet wird.
- Suche nach Zeichenketten: Die gefundenen Objekte werden in einem Algorithmus daraufhin überprüft, ob sie in einer Reihe angeordnet sind und es sich somit um eine Zeichenkette handeln könnte. Der Algorithmus verwirft alle Objekte, die bestimmte a-priori-Annahmen nicht erfüllen. Zu den Annahmen gehört u.a., dass die Objekte über

die gleiche Größe verfügen und einen konstanten Abstand zueinander haben. Die Zeichenkette wird nur weiterverarbeitet, wenn sie aus mehr als 5 Zeichen besteht.

- Ziffernerkennung: Auf die erkannten Objekte wird, falls sie zu einer Zeichenkette gehören, einzeln die Ziffernerkennung angewendet und der erhaltene Zählerstand wird gespeichert.

Mehrere Systemkomponenten können als Klassifikatoren angesehen werden: Die Detektion der ROIs, die Suche nach Zeichenketten und die Ziffernerkennung selbst. Es wird die Tatsache ausgenutzt, dass sich ein komplexes Klassifikationssystem auf mehrere, in Reihe geschaltete, einfachere Unterklassifikatoren abbilden lässt. Es ist zwar möglich die Bilder direkt auf einen sehr komplexen Klassifikator zu geben, allerdings wären dann sowohl das Trainieren dieses Klassifikators, als auch die Klassifikation selbst sehr rechenintensiv. Ein Nachteil von mehreren in Reihe geschalteten Klassifikatoren ist, dass sich die Erkennungsrate des Gesamtsystems aus dem Produkt der Erkennungsraten der einzelnen Klassifikatoren ergibt. Die Erkennungsrate des Gesamtsystems kann daher niemals besser sein als die Erkennungsrate des schlechtesten Unterklassifikators.

4.2.1. Bildererfassung

Die Bilderfassung kann in zwei unterschiedlichen Modi erfolgen. Einerseits ist es möglich Bilder von der Kamera des Raspberry Pi einzulesen. Zum Ansteuern dieser Kamera wurde eine Softwarebibliothek der Universität Cordoba genutzt [28]. Andererseits ist es möglich Bilder aus einer Validierungsmenge von der Festplatte einzulesen. Auch die Validierungsbilder wurden mit der Kamera des Raspberry Pi und derselben Softwarebibliothek aufgenommen, um nach Möglichkeit die gleichen Rahmenbedingungen bei Validierung und Normalbetrieb der Software zu erreichen. Aktuell besteht die Validierungsmenge aus 180 Bildern, welche von neun verschiedenen Messpunkten aus aufgenommen wurden. Abbildung 64 zeigt den genauen Abstand der Messpunkte zum Stromzähler. Es wird grundsätzlich davon ausgegangen, dass das Smart Metering System später zentral vor dem Zähler positioniert wird. Dennoch wurde das System unter verschiedenen Winkeln validiert, um eine Robustheit gegenüber kleinen Abweichungen vom Idealzustand (zentrale Position) nachweisen zu können. Von jedem der Messpunkte aus wurden zehn Bilder mit unterschiedlichen Ziffernstellungen aufgenommen. Die drei ersten Ziffern des Zählerstandes sind bei allen Bildern konstant. Nur die letzten drei Ziffern des Zählerstandes werden von Bild zu Bild verändert und nehmen dabei alle Werte zwischen 0 und 9 an. Die gesamte Bildaufnahme wurde einmal bei hellem Tageslicht und einmal bei eingeschalteter Raumbeleuchtung durchgeführt, um die Erkennungsrate bei verschiedenen Beleuchtungssituationen abschätzen zu können. Zwischen Validierungsmodus und normalem Betrieb kann beim Kompilieren der Software mittels Define-Angaben gewählt werden.

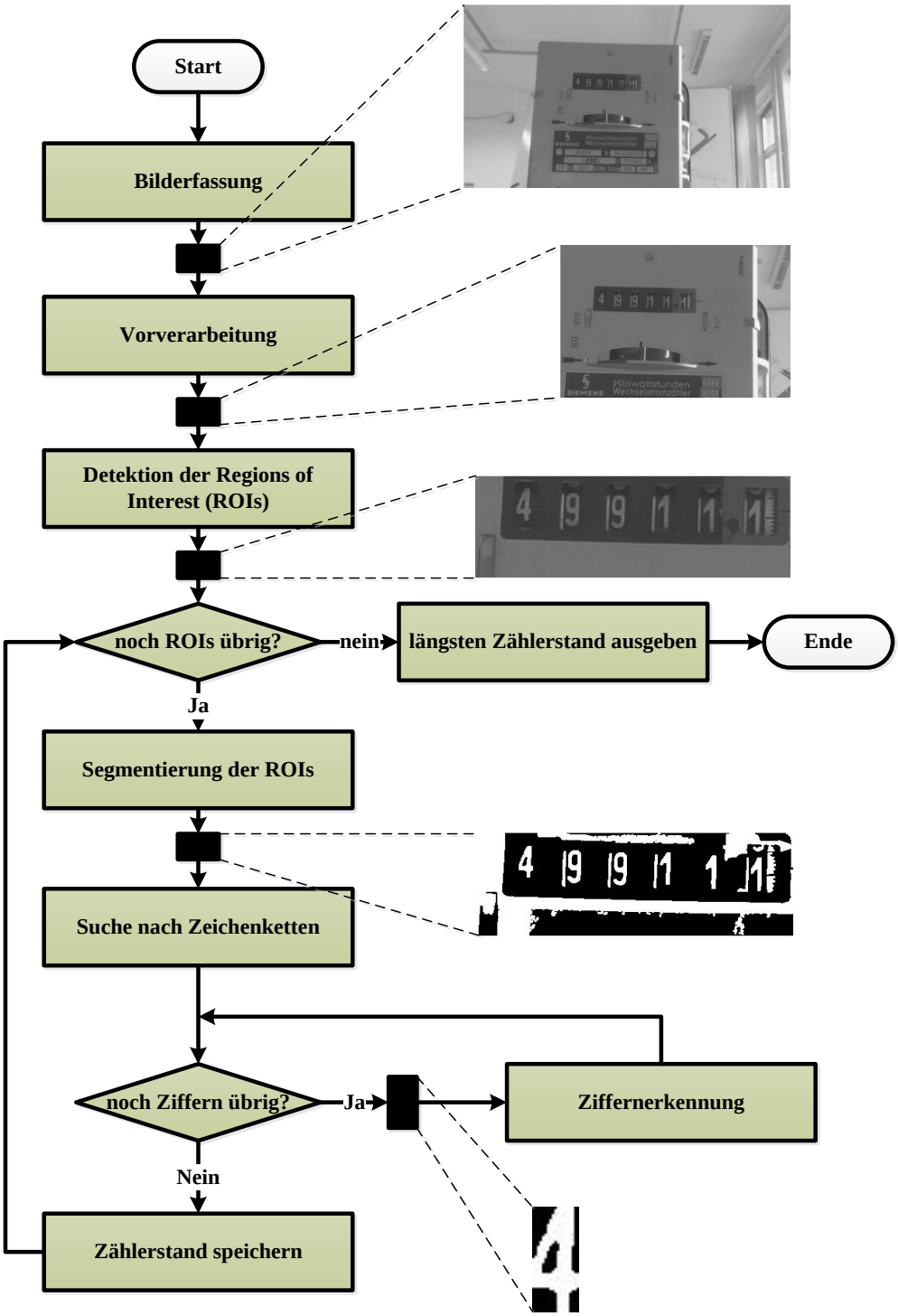


Abbildung 63: Programmablaufplan

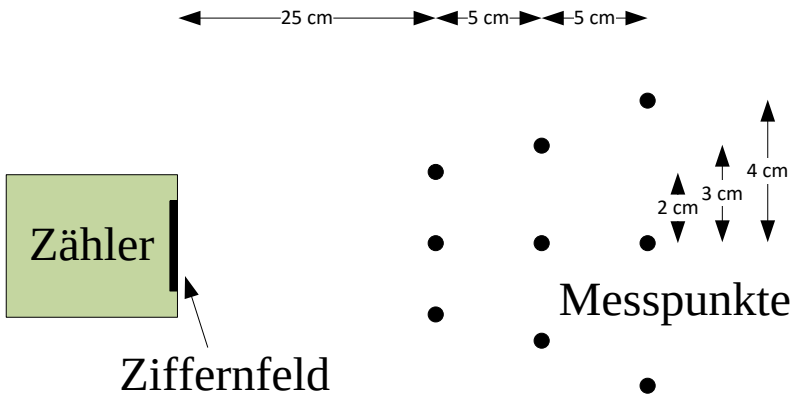


Abbildung 64: Versuchsaufbau

4.2.2. Vorverarbeitung

Um die Verarbeitungsgeschwindigkeit des Systems zu verbessern, wird das Eingangsbild mit einer Größe von 1920 × 1080 nicht komplett verarbeitet, sondern lediglich ein Ausschnitt der Größe 900 × 600. In der Vorverarbeitung werden daher zunächst die mittleren 900 × 600 Pixel des Bildes ausgeschnitten. Bezüglich des Rechenaufwandes wäre es zwar effektiver, direkt ein Bild der Größe 900 × 600 von der Kamera aufzunehmen, dies wird durch den Kamera-Treiber aber nur bedingt unterstützt. Bei zu kleinen Auflösungen wird vom Treiber nicht etwa ein kleinerer Teil des Bildes ausgegeben, sondern das Bild lediglich von einer höheren Auflösung auf eine geringere Auflösung herunter skaliert. Auf diesen herunter skalierten Bildern sind Objekte dann deutlich unschärfer, als auf solchen mit hoher nativer Auflösung. Daher wird in der Vorverarbeitung des Smart Metering-Systems der mittlere Teil des Full-HD Eingangsbildes ausgeschnitten. Abbildung 65 zeigt das Eingangsbild mit Full-HD-Auflösung, Abbildung 66 den ausgeschnittenen Mittelbereich des Eingangsbildes der Größe 900 × 600, welcher vom System weiterverarbeitet wird.

Des Weiteren arbeitet das System mit Grauwert-Bildern. Einerseits können diese aufgrund der geringeren Datenmengen schneller verarbeitet werden, andererseits wird in [5] festgestellt, dass Erkennungssysteme, welche die Farbinformation nutzten, i.A. eine schlechtere Erkennungsrate aufweisen als solche, die ohne Farbinformation arbeiten. Dies ist unter anderem dadurch bedingt, dass Farben und vor allem Farbtöne extrem beleuchtungsabhängig sind. Aufgrund der Erkenntnisse aus der Forschung und den genannten Effizienzgründen wurde beim Entwurf des Designs entschieden, dass das System auf Grauwertbildern arbeiten soll. Eine weitere Aufgabe der Vorverarbeitung ist daher ggf. die Konvertierung einer eingelesenen Bilddatei vom RGB-Farbraum in Graustufen. Wird ein neues Bild von der Kamera aufgenommen, dann handelt es sich bereits um ein Graustufenbild. Auch die

Aufnahme von Grauwertbildern kostet weniger Rechenzeit als die Aufnahme von Bildern mit RGB-Werten.

Abbildung 65: Originalbild

Abbildung 66: Ausschnitt des Originalbildes, der verarbeitet wird

4.2.3. Detektion der Regions of Interest

Wie bereits erwähnt, ist es häufig sinnvoll bei Klassifikationsproblemen mehrere Klassifikatoren in Reihe zu schalten, um das Klassifikationsproblem zu vereinfachen. Daher wird vor der Segmentierung und Ziffernerkennung zunächst nach den ROIs im Bild gesucht. ROIs sind Bereiche, in denen sich das Ziffernfeld des Zählers befinden könnte. In [5] wird für die Kennzeichenerkennung festgestellt, dass die besten Klassifikationssysteme zunächst ROIs suchen, die dann später weiterverarbeitet werden. Auch in der Literatur, die sich explizit mit dem Ablesen analoger Stromzähler befasst, werden häufig ROI-Detektoren eingesetzt.

In [5] wird eine Vielzahl von Verfahren zur ROI-Detektion vorgestellt. Dabei wird hervorgehoben, dass kantenbasierte Verfahren besonders gute Ergebnisse bezüglich der Erkennungsrate und der Rechenzeit erreichen. In diesem System wurde daher ein kantenbasiertes Verfahren implementiert, welches in Abschnitt 4.2.5 genauer beschrieben wird. Dieses Verfahren erreicht auch bei schlechter Beleuchtung und geringem Kontrast sehr gute Ergebnisse. Außerdem wurde ein zweites, alternatives Verfahren umgesetzt, welches auf der Hough-Transformation beruht (vgl. Abschnitt 0).

In [20] wird zur ROI-Detektion eine Form des Projektionsverfahrens genutzt. Dabei werden nach einer Schwellwertbildung die Anzahl der schwarzen Pixel pro Zeile und Spalte gezählt. Die Grenzen des Zählerfeldes werden in den Bereichen gesucht, in denen sich diese Werte besonders stark verändern. Dieses Verfahren kann aber Nachteile haben, wenn der Stromzähler über eine umfangreiche Beschriftung verfügt, das Ziffernfeld hell ist, oder durch eine schlechte Beleuchtung viele dunkle Bereiche auf dem Bild zu erkennen sind. Daher wurde das Verfahren aus [20] nicht übernommen.

4.2.4. ROI-Detektion auf Grundlage der Hough-Transformation

Ein in dieser Arbeit implementierter Ansatz, um die ROIs im Bild finden zu können, basiert auf der Hough-Transformation. Die Grundannahme besteht darin, dass das Ziffernfeld des Zählers rechteckig ist und sich relativ gut vom Hintergrund absetzt. Insbesondere wird es im oberen und unteren Bereich von relativ langen horizontalen Kanten begrenzt. Die Hough-Transformation eignet sich sehr gut dazu, Kanten auch auf stark verrauschten Bildern effizient zu detektieren (vgl. Abschnitt 0).

Zunächst wird ein Sobel-Kantendetektor auf das Bild angewendet, um aus dem Grauwertbild ein Binärbild zu erzeugen. In dem Binärbild besagt dann eine Eins, dass der entsprechende Pixel zu einer Kante gehört. Eine Null bedeutet, dass keine Kante vorliegt. Abbildung 67 zeigt das Bild nach der Kantendetektion.

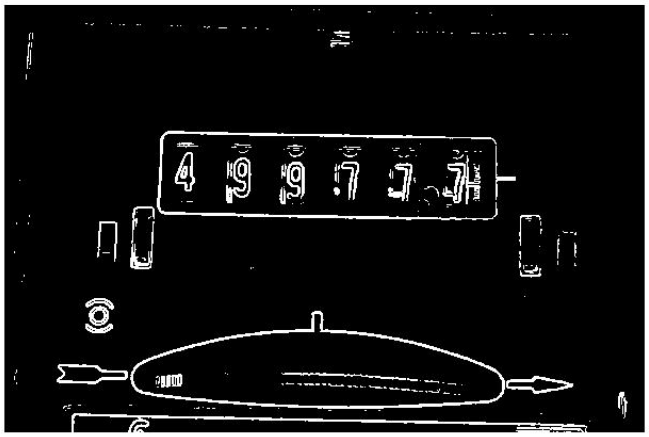


Abbildung 67: Originalbild nach der Kantendetektion mittels Sobel-Operator

Nach der Kantendetektion wird die Hough-Transformation auf das Bild angewendet. Die Hough-Transformation erzeugt dann, wie in Abschnitt 0 beschrieben, ein Bild in der Hough-Ebene, wobei die lokalen Maxima den wahrscheinlichsten Geraden im Bild entsprechen. Abbildung 68 zeigt die Hough-Transformation des Eingangsbildes.

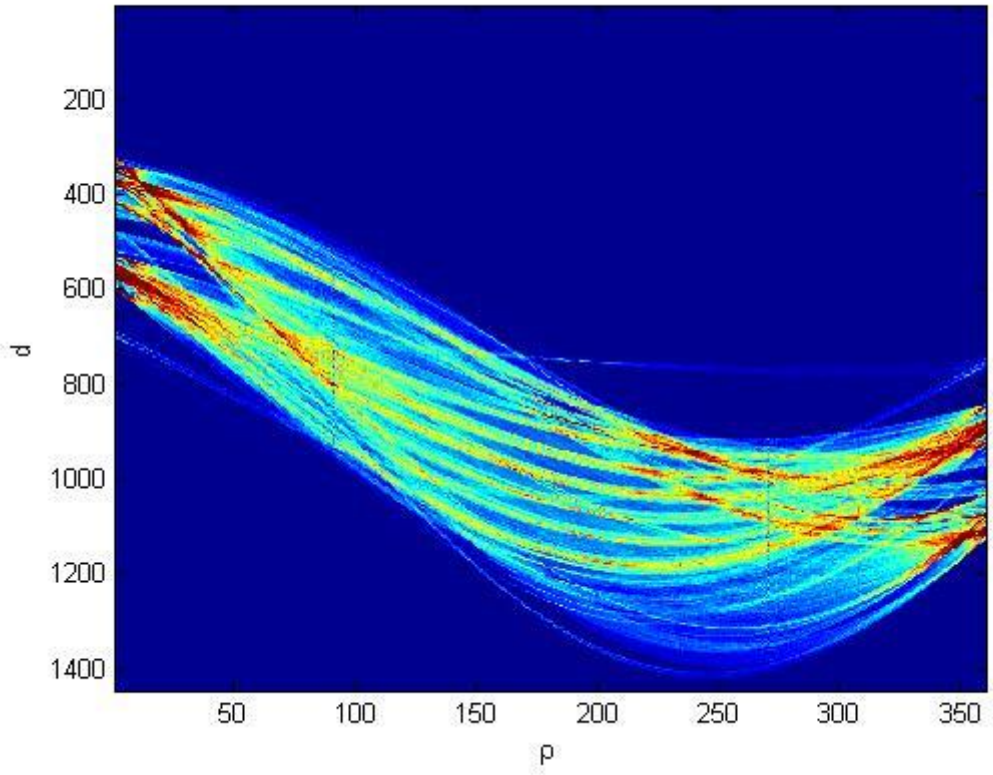


Abbildung 68: Hough-Transformation des Eingangsbildes

Im nächsten Schritt müssen die lokalen Maxima gefunden werden. Die Koordinaten in der Hough-Ebene entsprechen dabei dem Abstand d der Geraden zum Koordinatenursprung und dem Winkel ρ, welcher zwischen diesem und der x-Achse aufgespannt wird. Vor der Suche nach den lokalen Maxima, wird der mittlere Bereich des Hough-Bildes gelöscht. Dieser Bereich soll nicht untersucht werden, weil die hier gefundenen Geraden im *x,y*-Raum senkrecht stehen und einen Winkel von $\rho < 45°$ aufweisen. Wie eingangs bereits erwähnt, sucht der Algorithmus nur nach der langen oberen und unteren Kante des Ziffernfeldes. Die korrespondierenden Geraden müssen also horizontal liegen. Daher werden alle senkrechten Geraden im Hough-Raum unterdrückt. Damit der Algorithmus nicht zu viele Geraden erkennt, wird ein Schwellwert definiert. Dieser Schwellwert liegt bei $0{,}1 \cdot max$. Wobei *max* den Maximalwert im Hough-Raum bezeichnet, welcher der wahrscheinlichsten Geraden im *x,y*-Raum entspricht. Der Wert von *0,1* hat sich in praktischen Versuchen als geeigneter Wert erwiesen, auch auf kontrastarmen Bildern. Im Folgenden wird nach Maxima im Hough-Raum gesucht. Da vorwiegend die lokalen Maxima von Bedeutung sind, wird nach jedem gefunden Maximum der Wert an der Stelle des Maximums und alle Werte in der Nachbarschaft mit Nullen überschrieben. Der Bereich, in dem die Werte auf null gesetzt werden, hat die Größe $n \cdot m$, wobei *n* und *m* mit den Formeln (34) und (35) berechnet werden. Dabei ist *n* die Höhe der Maske und *m* die Breite der Maske. Der Faktor *2/50 = 1/25* zwischen dem Hough-Raum

und der Maske hat sich in praktischen Versuchen als guter Wert erwiesen. Die Berechnungsformel wurde so gewählt, dass *m* und *n* immer ungerade ganze Zahlen sind.

$$n = 2 \cdot \lceil Höhe/50 \rceil + 1 \tag{34}$$

$$m = 2 \cdot \lceil Breite/50 \rceil + 1 \tag{35}$$

Es sind nur lokale Maxima bei der Suche von Bedeutung, weil es bei der Hough-Transformation auf verrauschten Bildern dazu kommen kann, dass statt einer sehr wahrscheinlichen Geraden eine Bündelung von mehreren weniger wahrscheinlichen, aber sehr ähnlichen Geraden gefunden wird. Im schlechtesten Fall kann es dazu kommen, dass statt mehrerer Geraden im Bild nur verschiedene Ausführungen einer Geraden gefunden werden. Daher ist es ein gängiger Ansatz, nur nach lokalen Maxima zu suchen sowie größere Bereiche von hohen Werten zu einem Maximum zusammenzufassen.

Mit den lokalen Maxima wurden die wahrscheinlichsten Geraden im Bild gefunden. Nun müssen allerdings noch die exakten Kanten im Bild gefunden werden. Die Lage von unendlich langen Geraden in einem Bild ist für die Bildverarbeitung nur von geringer Bedeutung. Viel entscheidender ist die Lage der Start- und Endpunkte der Kanten im Bild. Diese müssen in einem weiteren Schritt gefunden werden. Bevor im Kantenbild allerdings nach Kanten gesucht werden kann, wird das Bild zunächst mit einer rechteckigen Maske gefaltet, um die vorhandenen Kanten zu verbreitern. Die Verbreiterung der Kanten im Kantenbild ist wichtig, weil die Kanten im Originalbild nur einen Pixel breit sind. Aufgrund von Ungenauigkeiten bei der endlichen Auflösung des Hough-Raums und des Kantenbildes kann es ansonsten zur fehlerhaften Detektion der Start- und Endpunkte kommen. Die Faltungsmaske hat dabei eine Höhe von m und eine Breite von $2 \cdot m$, wobei *m* mit den Formeln (36) und (37) berechnet wurden. Die Maske wird also adaptiv berechnet, ist aber immer doppelt so breit wie hoch. Diese Anpassung wurde wieder aufgrund der Annahme vorgenommen, dass die gesuchten Kanten horizontal im *x*,*y*-Raum liegen. Durch die Form der Faltungsmaske werden also horizontale Kanten im *x*,*y*-Raum unterstützt. Der Wert von *400*, durch welchen die Breite in Formel (36) dividiert wird, hat sich in praktischen Versuchen als gut erwiesen.

$$n = \lceil Breite/400 \rceil \tag{36}$$

$$m = 2 \cdot n + 1 \tag{37}$$

Im ersten Schritt des Algorithmus werden auf Grundlage der gefundenen Geraden zunächst Start- und End-Punkte der Kanten im Bild gesucht. Hierfür wird eine Wertetabelle mit allen *x*- und *y*-Werten im Bild erzeugt. Die Geradengleichung wird in der Form *y=f(x)* aufgestellt, da nur horizontale Kanten im Bild von Interesse sind. Zur Suche nach Start- und Endpunkten wird nun das Bild entlang der Punkte in der Wertetabelle untersucht. Wenn der Pixelwert von

einem Punkt zum nächsten von 0 auf 1 wechselt, dann wurde der Startpunkt einer Kante gefunden. Wechselt der Pixelwert dagegen von 1 auf 0, dann handelt es sich um den Endpunkt einer Kante. Die so gefundenen Punkte werden in zwei Arrays geschrieben. Nachdem das gesamte Bild entlang der Punkte in der Wertetabelle untersucht wurde, wird zu jedem Endpunkt der am nächsten liegende Startpunkt gesucht und beide Punkte werden als Eckpunkte einer Kante gespeichert.

Am Ende der Suche wird nach parallelen Kanten gesucht und getestet, ob diese ein Viereck bilden könnten. Dazu müssen die Kanten folgende Bedingungen erfüllen:

- Die Kanten müssen ein Viereck ergeben, dass ein Seitenverhältnis von maximal 1:2 und mindestens 1:10 hat.
- Die Kanten müssen in etwa übereinander liegen.

Zunächst wird aus den Kanten die kurze Seite *c* des potenziellen Vierecks nach Formel (38) berechnet. Dabei sind d_1 und d_2 die Abstände der beiden zugehörigen Geraden zum Koordinatenursprung. Wenn die beiden Kanten parallel sind, lässt sich so ihr Abstand berechnen.

$$c = |d_1 - d_2| \tag{38}$$

Danach wird der reale Abstand zwischen den Start- und Endpunkten der Kanten berechnet. Abbildung 69 verdeutlicht die Suche nach parallelen Kanten. Die Abstände zwischen den Start- und Endpunkten (rote Linien) werden mit *c* verglichen. Von diesen Abständen dürfen zwei maximal um 5% von dem berechneten *c* abweichen, wenn die beiden Geraden wirklich parallel sind. Erfüllen zwei Geraden diese Bedingungen, dann werden sie als gefundenes ROI gespeichert.

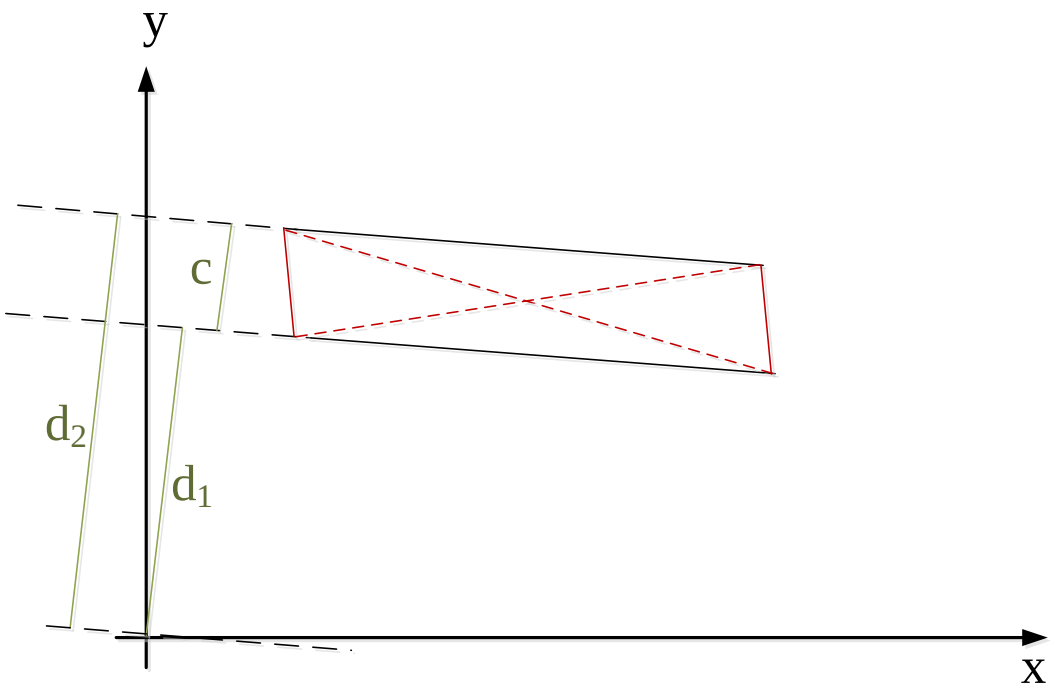


Abbildung 69: Parallele Kanten

Nachdem alle potenziellen ROIs im Bild gefunden wurden, werden diese der Größe nach sortiert.

4.2.5. ROI-Detektion auf Grundlage von vertikalen Kanten

Ein alternativer Ansatz, um ROIs zu detektieren, befasst sich mit den vertikalen Kanten im Bild. Das Verfahren beruht auf der Annahme, dass sich im Bereich des Zählerstandes relativ viele vertikale Kanten befinden und im Rest des Bildes relativ wenige. Daher wird zunächst ein besonderer Sobel-Operator (vgl. Abschnitt 4.1.2) auf das Bild angewandt. Der angewendete Sobel-Operator berechnet nur die Ableitung in x-Richtung, d.h., betrachtet nur vertikale Kanten. Zur Berechnung der Ableitung in x-Richtung wird das Bild mit einer Matrix, wie in Gleichung (39) dargestellt, gefaltet und anschließend der Betrag gebildet, da nur der Betrag und nicht das Vorzeichen der Ableitung in x-Richtung relevant ist. Abbildung 70 zeigt das Eingangsbild vor der Kantendetektion und Abbildung 71 das Ergebnis der Faltung des Eingangsbildes mit der genannten Matrix.

$$Dx = \begin{matrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{matrix} \tag{39}$$

Abbildung 70: Eingangsbild

Abbildung 71: Detektion vertikaler Kanten

Anschließend wird das Kantenbild durch die Faltung mit einer Maske geglättet, um auf Grundlage der Kanten möglichst die Regionen mit vielen Kanten zu finden. Eine Faltung entspricht immer einer Mittelung, daher ist der Wert jedes Pixels nach der Faltungsoperation proportional zur Anzahl der Kanten in der Umgebung des Pixels. Die Größe der Faltungsmaske beträgt immer ein Zwanzigstel der Auflösung des Bildes. Bei der Standardauflösung von 900 × 600 hat die Maske also eine Größe von 45 × 30. Nachdem nun jeder Pixelwert eine Aussage darüber trifft, wie viele Kanten in der Umgebung des Pixels liegen, wird ein Schwellwert nach der Niback Formel (vgl. Abschnitt 4.2.6, [5]) auf das Bild angewandt. Abbildung 72 zeigt das Ergebnis der Schwellwertbildung.

Abbildung 72: Ergebnis der Schwellwertbildung

Nach der Schwellwertbildung werden um die entstehenden Objekte Bounding-Boxes erzeugt. Eine Bounding-Box bezeichnet das kleinste Rechteck, welches um ein Objekt im Bild gelegt werden kann. Im Rahmen dieser Arbeit werden die Bounding-Boxes auf zwei verschiedene Arten erzeugt. Eine Connected-Component-Analyse sucht in einem Binärbild (Werte 0 und 1 bzw. Hintergrund und Vordergrund) nach Objekten, indem es benachbarte Vordergrundpixel zu Objekten zusammenfügt. Es kann bei der Definition der Nachbarschaft zwischen zwei Varianten unterschieden werden. Entweder werden zu jedem Pixel die angrenzenden 4 Pixel als Nachbarn interpretiert oder die angrenzenden 8 Pixel. Abbildung 73 und Abbildung 74

zeigen beide Varianten. In dieser Arbeit werden die angrenzenden 8 Pixel als Nachbarn angesehen. Diese Variante ist zwar rechenaufwendiger, allerdings werden schlecht segmentierte Objekte besser erkannt.

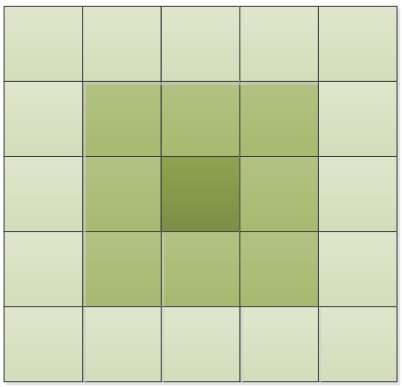


Abbildung 73: 8 Pixel werden als Nachbarn interpretiert

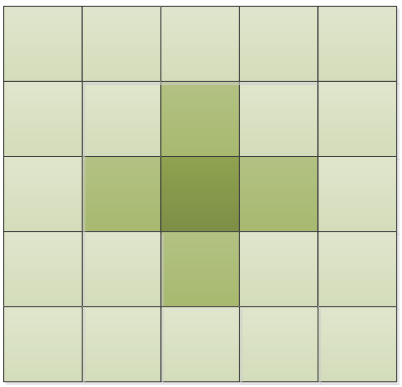


Abbildung 74: 4 Pixel werden als Nachbarn interpretiert

Die erhaltenen Bounding-Boxes werden dann in beiden Fällen als ROIs interpretiert. Es wird implizit angenommen, dass die ROIs bzw. Ziffernfelder eine rechteckige Form haben. Die ROIs dürfen sich gegenseitig überschneiden, um eine maximale Fehlertoleranz zu erzielen. Diese Designentscheidung wurde getroffen, da es tolerierbar ist, wenn der als ROI erkannte Bereich des Bildes größer ist als der wirkliche Zählerstand. Wenn aber bestimmte Bereiche des Zählerstandes bei der ROI-Detektion abgeschnitten werden, kann der Zählerstand in den folgenden Schritten nicht mehr richtig erkannt werden. Hier zeigt sich der bereits beschriebene Sachverhalt, dass die Erkennungsrate des Gesamtsystems niemals besser sein kann als die Erkennungsrate einer Klassifikationsstufe. Abbildung 75 zeigt das Ergebnis der ROI Detektion mit den Bounding-Boxes um die erkannten ROIs.

Abbildung 75: Erkannte ROIs

4.2.6. Segmentierung

Alle weiteren Verarbeitungsschritte werden für jedes ROI einzeln durchgeführt. Im Rahmen dieser Arbeit wird angenommen, dass die Ziffern des Zählerstandes sich deutlich vom Hintergrund abheben und die Grauwerte somit ein gutes Merkmal zur Segmentierung des Bildes sind. Es wird eine bimodale Verteilung wie in Abbildung 51 angenommen. Diese Annahme kann getroffen werden, weil das Ziffernfeld von Stromzählern immer aus weißen Ziffern auf schwarzem Hintergrund oder aus schwarzen Ziffern auf weißem Hintergrund besteht. In jedem der beiden Fälle heben sich die Zahlen bei ausreichender Beleuchtung gut vom Hintergrund ab. Es wurden zwei verschiedene Schwellwerte implementiert.

Beim ersten Ansatz wurde ein **globaler Schwellwert** nach der Niback Formel [5] aus Gleichung (40) berechnet. Dabei ist m der Mittelwert, σ die Standardabweichung, k ein beliebiger Faktor und T der resultierende Schwellwert. In dieser Arbeit wurde auf Grundlage von Experimenten für k der Wert *0,2* gewählt, weil sich mit diesem eine gute Erkennungsrate erzielen lässt.

$$T = m + k \cdot \sigma \tag{40}$$

Ein globaler Schwellwert kann an dieser Stelle sinnvoll sein, weil bereits eine ROI-Detektion auf das Bild angewandt wurde und somit der globale Schwellwert nicht mehr auf das gesamte Bild angewendet wird, sondern nur noch auf das aktuelle ROI. Es ist davon auszugehen, dass die Helligkeitsunterschiede innerhalb des ROIs den globalen Schwellwert weniger verfälschen, als dies bei einer Schwellwertbildung auf dem gesamten Bild der Fall wäre.

Der zweite Ansatz verwendet einen **lokalen Schwellwert**. Dabei wird für jedes Pixel im Bild ein separater Schwellwert berechnet. Hierfür wird eine 15 × 15 Pixel große Maske über das Bild geschoben. Der Schwellwert ergibt sich dann als gewichtete Summe aller Pixel unter der Maske, wie in Formel (41) dargestellt. Hierbei ist c eine Konstante, p_i der i-te Pixel unter der Maske und k_i der i-te Koeffizient der Maske. Die Koeffizienten der Maske ergeben sich aus den Werten einer 2-dimensionalen Gaußfunktion. Um die Schwellwertbildung möglichst adaptiv zu gestalten, wird die Konstante c nach Formel (42) berechnet. Damit ist der Schwellwert abhängig von der Standardabweichung σ, wobei σ im Bereich des Bildes berechnet wird, der unter der Maske liegt. Der Faktor von 0,07 wurde experimentell bestimmt und hat sich als optimal bezüglich der Erkennungsrate herausgestellt.

$$T = c + \sum k_i \cdot p_i \tag{41}$$

$$c = 0{,}07 \cdot \sigma \tag{42}$$

Bei der Segmentierung können deutlich mehr Objekte als die Zahlen des Zählerstandes gefunden werden. Von allen gefundenen Objekten werden nur die größten Objekte weiterverarbeitet und alle anderen verworfen. Es ist davon auszugehen, dass nur wenige Objekte im Bild größer sind als die Zahlen des Zählerstandes. Insbesondere wird angenommen, dass die durch die lokale Schwellwertbildung entstandenen Artefakte deutlich kleiner sind als die Zahlen des Zählerstandes. Wie viele Objekte nach der Segmentierung weiterverarbeitet werden, kann mit einer Konfigurationskonstante vor dem Kompilieren des Programms entschieden werden. Bei praktischen Versuchen haben sich 15 Objekte als ein guter Wert herausgestellt.

Abbildung 76: Eingangsbild

Abbildung 77: Ergebnis der lokalen Schwellwertbildung mit Artefakten

Abbildung 78: Ergebnis der Erosion und Dilatation, mit teilweise unterdrückten Artefakten

Abbildung 79: Ergebnis der Segmentierung

4.2.7. Suche nach Zeichenketten

Nach der Detektion der ROIs und der Segmentierung werden oft nicht nur die Zahlen des Zählerstandes als Objekte erkannt, sondern auch ungewollte Objekte. Die Gründe hierfür sind einerseits eine ungenaue Detektion der ROIs, sodass z.B. Teile der Beschriftung des Stromzählers miterkannt werden. Andererseits kann es auch zu einer komplett falschen ROI-Detektion kommen, sodass Objekte als ROI erkannt werden, die gar keinen Zählerstand enthalten. Außerdem können sich selbst bei einem perfekt erkannten Zählerfeld neben den Ziffern noch andere Objekte im Bereich des Zählerstandes befinden. Als Beispiele wären Lichtreflektionen, Markierungen innerhalb des Zählerfeldes sowie Artefakte zu nennen, die durch die lokale Schwellwertbildung entstanden sind.

Bei der Entwicklung des Systems und den ersten praktischen Tests hat sich gezeigt, dass durch den Einsatz von Apriori-Wissen sehr sicher zwischen den Zahlen des Zählerstandes und sonstigen Objekten unterschieden werden kann. Hierbei wurde folgendes Apriori-Wissen über die Beschaffenheit des Zählerstandes genutzt:

- Die Zahlen des Zählerstandes sind von links nach rechts im Ziffernfeld angeordnet.
- Die Zahlen des Zählerstandes haben aufgrund der maschinellen Fertigung des Zählers einen sehr genauen Abstand in *x*-Richtung voneinander.

- Die *y*-Werte der Zahlen des Zählerstandes weichen nur relativ gering voneinander ab.
- Die Zahlen des Zählerstandes haben alle nahezu die gleiche Breite
- Die Zahlen des Zählerstandes haben sehr genau die gleiche Höhe.
- Alle oben genannten Annahmen über x-Abstand, y-Abstand, Höhe und Breite müssen auf mindestens 5 Objekten gelten. Diese können dann als Zahlen eines Zählerstandes interpretiert werden.

Zunächst werden die Objekte in ein Array geschrieben und nach aufsteigendem x-Wert sortiert. Grundlage für diese Überlegung ist, dass Bilder als Matrizen interpretiert und, wie in Abbildung 80 dargestellt, mit Indizes bzw. *x*- und *y*-Werten versehen werden. Der Pixel *(0,0)* liegt immer in der oberen linken Ecke des Bildes. Von diesem ausgehend werden alle weiteren Pixel mit positiv ansteigenden *x*- und *y*-Werten versehen. Das Objekt mit dem niedrigsten x-Wert trägt nach der Sortierung den Index 0, das Objekt mit dem zweitniedrigsten x-Wert den Index 1 usw. bis zum letzten Objekt. Um die Wirkung dieser Sortierung zu veranschaulichen soll das Objekt mit dem Index *k* betrachtet werden, wobei *k* beliebig gewählt werden kann. Für jedes andere Objekt mit dem Index $l \neq k$ in Array gilt dann:

- Fall 1 - $l > k$: Das Objekt befinden sich im Bild weiter rechts als das Objekt *k*, da sein x-Wert größer ist als der des *k*-ten Objektes.
- Fall 1 - $l < k$: Das Objekt befinden sich im Bild weiter links als das Objekt *k*, da sein x-Wert kleiner ist als der des *k*-ten Objektes.

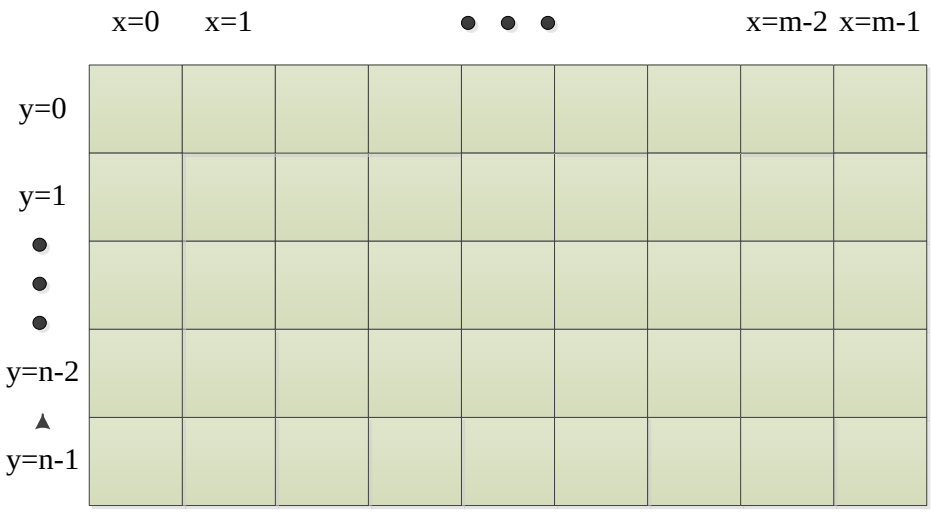


Abbildung 80: x- und y-Werte der Pixel in einem Bild

Alle Objekte im Array werden auf die oben genannten Bedingungen bezüglich *x*- und *y*-Abstand, Höhe und Breite getestet. Dazu werden drei Zähler verwendet, deren Bedeutung im Folgenden erklärt werden soll.

- *baseCtr*: Alle Objekte, auf welche der *baseCtr* zeigt, werden als am weitesten links stehendes bzw. erstes Objekt der Zeichenkette interpretiert. Unter allen Objekten der gesuchten Zeichenkette hat dieses Objekt den geringsten x-Wert. Die Höhe, Breite und der y-Wert dieses Objektes dienen als Referenz für alle weiteren Objekte in der Zahlenreihe.
- *startCtr*: Dieser Zähler enthält den Index des Objektes, welches als zweites Objekt der zu findenden Zeichenkette interpretiert wird. Dieses Objekt hat den zweitniedrigsten x-Wert aller Objekte der Zeichenkette und steht somit an zweiter Stelle von links im Bild. Zwischen den Objekten, auf welche *baseCtr* und *startCtr* zeigen, lässt sich der Abstand in x-Richtung berechnen.
- *destCtr*: Der dritte Zähler zeigt der Reihe nach auf alle zu testenden Objekte. Die Objekte, auf welche der *destCtr* zeigt, werden daraufhin überprüft, ob sie zur Zeichenkette gehören oder nicht. Dazu müssen sie die o.g. Bedingungen bezüglich des x- und y-Abstands, der Länge und der Breite erfüllen. Alle Objekte, auf welche der *destCtr* zeigt, stehen rechts von den *baseCtr*-Objekten und *startCtr*-Objekten im Bild und haben daher höhere Indizes als die *baseCtr*-Objekte und *startCtr*-Objekte.

Abbildung 81 zeigt für den sehr einfachen aber anschaulichen Fall von 4 Objekten, auf welche Objekte die drei Zähler in jeder Iteration zeigen.

	Objekt 0	Objekt 1	Objekt 2	Objekt 3
0.Iteration	baseCtr	startCtr	destCtr	
1.Iteration	baseCtr	startCtr		destCtr
2.Iteration	baseCtr		startCtr	destCtr
3.Iteration		baseCtr	startCtr	destCtr

Abbildung 81: Belegung der Zähler bei 4 Objekten

In der 0. Iteration zeig der *baseCtr* auf das Objekt 0 und der *startCtr* auf das Objekt 1. Der *destCtr* zeigt zunächst auf das Objekt 2. Von Objekt 2 werden nun der x- und y-Wert, sowie die Breite und die Höhe darauf getestet, ob das Objekt mit den Objekten 0 und 1 eine Reihe bilden könnte.

In der 1. Iteration bleiben *baseCtr* und *startCtr* unverändert, der *destCtr* zeigt aber auf das Objekt 3. Die Position des *destCtr* ist dabei unabhängig vom Ergebnis der 0. Iteration. Allerdings ist der Test, der auf Objekt 3 angewendet wird von dem Ergebnis der 0. Iteration abhängig. Es können zwei Fälle auftreten:

- Fall 1 - Das Objekt 2 bildet eine Reihe mit den Objekten 0 und 1: In diesem Fall wird Objekt 3 darauf getestet, ob es mit den Objekten 0,1 und 2 eine Reihe bildet.
- Fall 2 - Das Objekt 2 bildet **keine** Reihe mit den Objekten 0 und 1: In diesem Fall wird Objekt 3 darauf getestet, ob es mit den Objekten 0 und 1 eine Reihe bildet.

In der 2. Iteration wird nun der *startCtr* inkrementiert, weil der *destCtr* bereits alle verbleibenden Objekte daraufhin getestet hat, ob sie mit den Objekten 0 und 1 eine Reihe bilden. Der *startCtr* zeigt nun auf das Objekt 2. Der *destCtr* wird auf den Wert *startCtr+1* gesetzt und zeigt auf das Objekt 3. In der Folge werden alle weiteren Objekte daraufhin getestet, ob sie mit den Objekten 0 und 2 eine Reihe bilden. Daher muss mit der Inkrementierung des *startCtr* nun auch der x-Abstand neu berechnet werden. Nun wird das Objekt 3 daraufhin getestet, ob es mit den Objekten 0 und 2 eine Reihe bildet.

In der 3. Iteration wird der *baseCtr* inkrementiert, weil alle Objekte daraufhin getestet wurden, ob sie eine Reihe bilden, in der das Objekt 0 das erste Objekt ist. Der *baseCtr* zeigt also auf das Objekt 1, der *startCtr* wird auf den Wert *baseCtr+1* und der *destCtr* auf den Wert *startCtr+1* gesetzt. Nun wird also das Objekt 3 daraufhin getestet, ob es mit den Objekten 1 und 2 eine Reihe bildet.

Nach der 3. Iteration würde der Algorithmus nun auf 4 Objekten enden, weil alle Objekte daraufhin getestet wurden, ob sie eine Reihe ergeben.

Abbildung 82 zeigt noch einmal ausführlich den Ablaufplan des Algorithmus zur Suche nach Zeichenketten. Dabei steht die Variable *FoundObjs* für die Anzahl der Objekte der Zeichenkette, die bereits gefunden wurden. Auf diese *FoundObjs* Objekte treffen die vier oben genannten Bedingungen zu. Wenn alle Objekte untersucht wurden und *FoundObjs* größer oder gleich 5 ist, dann wird die Suche beendet. Alle Objekte, auf welche die vier oben genannten Bedingungen zutreffen, werden als Zahlen des Zählerstandes gespeichert. Alle anderen Objekte werden gelöscht.

Es ist zu beachten, dass der vorgeschlagene Algorithmus relativ rechenintensiv ist. Insbesondere skaliert der Algorithmus aufgrund seiner iterativen Form sehr schlecht für eine große Zahl von zu untersuchenden Objekten, weil die Anzahl der Iterationen proportional zur Anzahl der zu untersuchenden Objekte ist. Daher wird nur eine feste Anzahl von Objekten nach der Segmentierung weiterverarbeitet. Wie bereits erwähnt, wurde diese Anzahl auf 15 festgelegt, damit der Algorithmus möglichst effektiv arbeiten kann (vgl. Abschnitt 4.2.6).

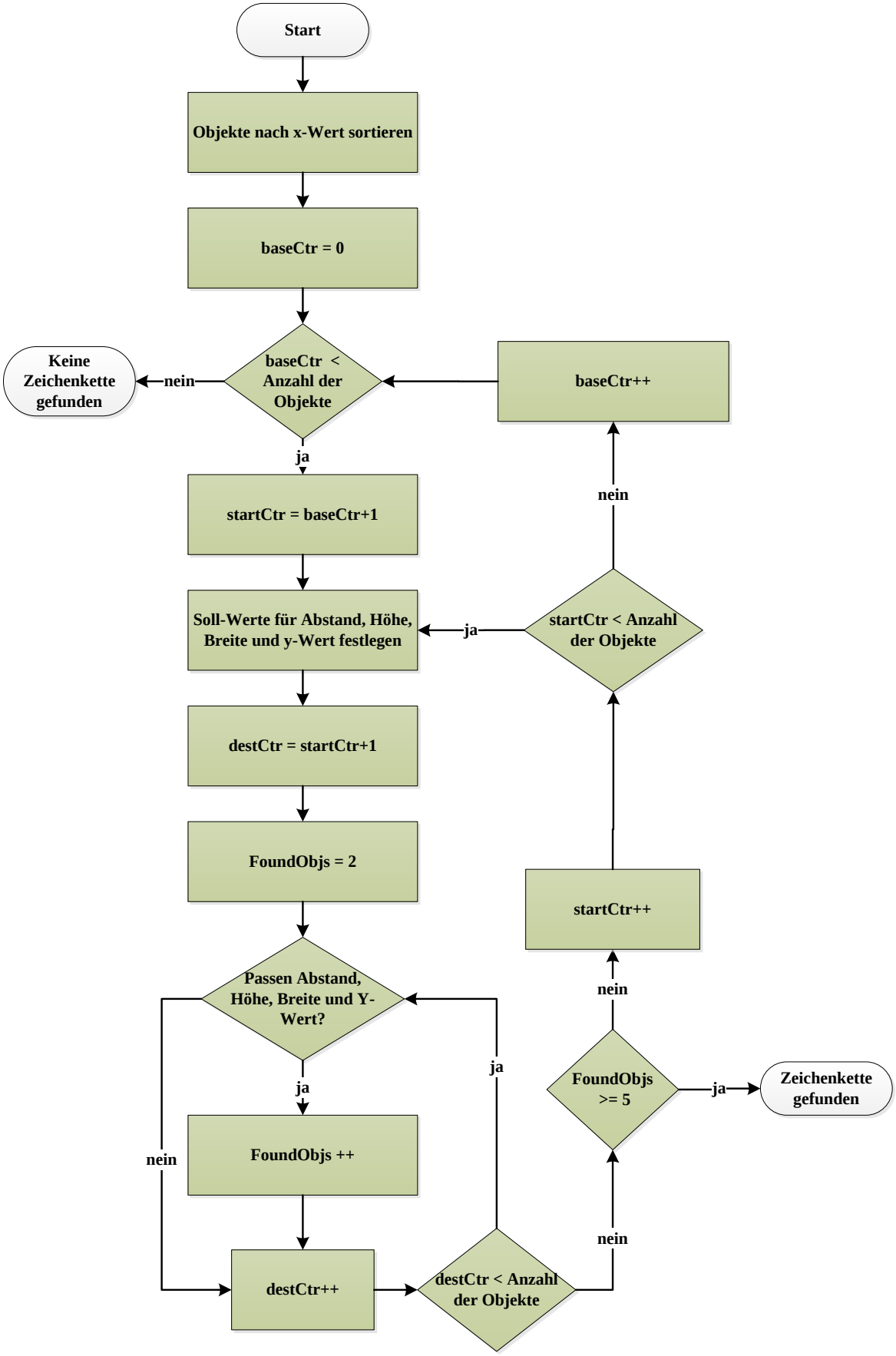


Abbildung 82: Ablauf des Algorithmus zur Suche nach Zeichenketten

Durch die anfängliche Sortierung der Objekte nach aufsteigendem x-Wert wird Rechenzeit eingespart. Insbesondere kann nach der Sortierung das Wissen genutzt werden, dass nur die Objekte, deren Index höher ist als der *baseCtr*, zur Zahlenkette gehören können.

Die Anzahl der Iterationen und somit die Komplexität des Algorithmus wurden von $O(n^6)$ auf $O(n^3)$ verringert. Die Komplexität der Suche nach 6 Objekten, die eine Zeichenkette bilden ergibt sich ohne Sortierung nach Formel (43) als $O(n^6)$. Zunächst müssen Paare von Objekten gefunden werden, welche die ersten beiden Ziffern einer Zeichenkette bilden. Bei *n* Objekten existieren genau $n \cdot (n-1)$ solcher Paare. Für jedes dieser Paare müssen für jede weitere Ziffer alle anderen Objekte untersucht werden. Jedes dieser Objekte wird daraufhin überprüft, ob es zur Zeichenkette gehört. Die Zahl der zu untersuchenden Objekte sinkt zwar pro Ziffer um 1, weil ja bereits einige der *n* Objekte zur Kette gehören, dennoch ergibt sich insgesamt Formel (43) und als Gesamtanzahl der Iterationen $O(n^6)$.

$$n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdot (n-4) \cdot (n-5) = O(n^6) \qquad (43)$$

Im Folgenden soll nun der implementierte Algorithmus (vgl. Abbildung 82) betrachtet werden. In diesem werden die Objekte zunächst nach aufsteigendem x-Wert sortiert. Danach müssen wieder Paare von Objekten gesucht werden. Nun existieren $n \cdot (n-1)/2$ solcher Paare. Im Vergleich zur normalen Form des Algorithmus müssen nur halb so viele Paare untersucht werden, weil die Reihenfolge der Objekte bereits bekannt ist. Der Index des ersten Objektes dieses Paares ist im *baseCtr* gespeichert, der Index des zweiten Objektes im s*tartCtr*. Für jedes dieser Paare müssen aber nur noch die $(n-k)$ Objekte daraufhin überprüft werden, ob sie mit dem ausgewählten Objektpaar eine Reihe bilden. Diese Objekte werden der Reihe nach vom d*estCtr* adressiert. Der Index *k* steht für den Index des zweiten Objektes, welcher im s*tartCtr* gespeichert wurde. Weil die Objekte nach aufsteigendem x-Wert gespeichert wurden, können nur noch die nachfolgenden $(n-k)$ Objekte zur Kette gehören. Bei jedem der $(n-k)$ untersuchten Objekte, wird geprüft, ob es zur Zeichenkette gehört. Für jedes Objekt, welches als zur Zeichenkette gehörend erkannt wird, wird der Wert von *FoundObjs* um 1 erhöht. Wurden alle $(n-k)$ Objekte untersucht, wird geprüft, ob der Wert von *FoundObjs* größer als 5 ist. Trifft dies zu, so wurde eine Kette von 5 oder mehr Zeichen gefunden, anderenfalls muss weitergesucht werden. In jedem der beiden Fälle werden für jedes der $n \cdot (n-1)/2$ Paare nur $(n-k)$ Tests durchgeführt. Die Anzahl der Iterationen ergibt sich also wie nach Formel (44). Eine Abschätzung der Iterationen bzw. der Komplexität des Algorithmus lässt sich auch beim Betrachten der verwendeten Zähler treffen. Mit dem b*aseCtr*, s*tartCtr* und d*estCtr* werden 3 Zähler verwendet, die jeweils auf ungefähr *n* Objekte zeigen. Somit ergibt sich eine Komplexität von $O(n)$ pro Zähler und eine Komplexität von $O(n^3)$ für den gesamten Algorithmus.

$$n \cdot (n-1) \cdot (n-k)/2 = O(n^3) \qquad (44)$$

Während der Algorithmus bei vielen Problemen sehr sicher arbeitet, sind vor allem falsch erkannte ROIs mit maschinell erstellten Schriftobjekten ein Problem. So gelten die oben genannten Bedingungen beispielsweise auch für Seriennummern oder Beschriftungen auf dem Zähler.

Im Allgemeinen hat sich jedoch gezeigt, dass der beschriebene Algorithmus sehr gut zwischen Zahlen eines Zählerstandes und anderen Objekten unterscheiden kann. Zwar ist davon auszugehen, dass das Smart Metering-Modul in etwa senkrecht vor dem Zähler platziert ist, dennoch wurde bei der Validierung des Systems belegt, dass der Algorithmus auch auf leicht perspektivischen Bildern solide arbeitet und somit eine ausreichende Toleranz gegenüber kleinen Abweichungen aufweist.

4.2.8. Ziffernerkennung

Zur Ziffern- bzw. Zeichenerkennung finden sich in der Literatur viele Ansätze mit guten Erkennungseigenschaften. Allerdings erreichen künstliche neuronale Netze aktuell die höchsten Erkennungsraten [4, 17, 18]. Künstliche neuronale Netze haben jedoch zwei Nachteile. Einerseits ist das von den neuronalen Netzen gespeicherte „Wissen“ in den Gewichten gespeichert und kann somit nicht auf Korrektheit überprüft werden. Wie das neuronale Netz auf neuen, bisher unbekannten Daten arbeitet, kann daher nicht vorhergesagt werden. Somit sind neuronale Netze immer mit einer gewissen Unsicherheit behaftet. Weiterhin haben besonders gut klassifizierende neuronale Netze oft sehr viele Neuronen und Gewichte. Aus diesem Grund sind sie ggf. während der Klassifizierung, mindestens aber während des Trainings sehr rechenintensiv. Des Weiteren benötigen sehr große Netze auch eine sehr große Anzahl von Trainingsmustern. Bei der Erkennung von gedruckten Ziffern, wie sie im vorgestellten System gefordert ist, handelt es sich um einen in der aktuellen Forschung wenig betrachteten Ansatz. Daher ist im Besonderen das Finden von geeigneten Testmustern ein Problem. Für die Erkennung von gedruckten Ziffern sind keine Testbibliotheken vorhanden, während in den Bereichen der Texterkennung in natürlicher Umgebung oder zur Erkennung von handgeschriebenen Ziffern und Buchstaben eine Vielzahl solcher Bibliotheken vorhanden sind.

Obwohl neuronale Netze relativ rechenintensiv sind und ein korrektes Training u.U. eine Herausforderung darstellt, kommt im vorgestellten System ein neuronales Netz zum Einsatz. Vor allem, weil gerade bezüglich der Stromabrechnung die Genauigkeit der Zählerstanderfassung eine höhere Priorität hat, als eine besonders schnelle Erkennung des Zählerstandes. In dieser Arbeit wird daher ein relativ komplexer Klassifikator eingesetzt, um eine möglichst gute Erkennungsrate auf Kosten der Erkennungsgeschwindigkeit zu erreichen.

4.2.9. Wahl der Architektur

Wie in Abschnitt 4.1.11 dargestellt, finden sich in der Literatur viele verschiedene Ansätze für Topologie und Vernetzung von neuronalen Netzen. Daher stellt sich die Frage, welche Architektur verwendet werden soll. In dieser Arbeit wurde ein Feedforward-Netz eingesetzt, d.h. ein mehrlagiges neuronales Netz ohne Rückkopplungen, wobei jedes Neuron einer Lage mit jedem Neuron der nächsten Lage verbunden ist. Ein solches Netz wird auch als Multilayer Perceptron (MLP) bezeichnet und zeichnet sich dadurch aus, dass es aufgrund seiner Topologie universal einsetzbar ist [8, 10]. Daher ist das MLP auch das am häufigsten eingesetzte künstliche neuronale Netz.

Alternative Klassifikatoren zum MLP wären zum Beispiel Konvolutionsnetze (CNNs – engl: Convolutional Neuronal Networks). Es gibt einige Ausführungen von CNNs, welche im Bereich der Schrifterkennung bessere Erkennungsraten erreichen als das MLP [17]. In [18] konnte allerdings gezeigt werden, dass mit einem MLP grundsätzlich genauso gute Erkennungsraten erzielt werden können wie mit CNNs. Außerdem sind MLPs im Allgemeinen weniger rechenintensiv als CNNs [9]. Vor Allem weil ein eingebettetes System als Zielplattform eingesetzt wird, muss die Rechenintensivität in Betracht gezogen werden. Zu guter Letzt unterstützen viele Softwarebibliotheken zwar MLPs, nicht aber die neueren CNNs. Auch die in dieser Arbeit verwendete Bibliothek OpenCV und das Programm Matlab unterstützen nativ nur das MLP. Aus diesen Gründen konzentriert sich diese Arbeit darauf, ein MLP als Klassifikator zu verwenden.

Nachdem das MLP als Architektur gewählt wurde, muss nun die Topologie des MLP bestimmt werden, insbesondere die Anzahl der Hidden-Neuronen und die Anzahl der Ebenen. Wie schon in Abschnitt 4.1.11 beschrieben, wirken sich mehrere Hidden-Ebenen positiv auf die Translations- und Rotationsinvarianz des Klassifikators aus, haben aber gleichzeitig den Nachteil, dass sich eine komplexere Fehleroberfläche ergibt und somit das Training erschwert wird [8]. In einigen praktischen Versuchen hat sich für das vorgestellte System keine Verbesserung der Erkennungsrate für mehr als eine Ebene von Hidden-Neuronen ergeben. Zu erklären ist dies einerseits durch die relativ kleine Anzahl der Testmuster. Denn je mehr Ebenen das Netz besitzt, desto mehr Gewichte besitzt es auch. Allgemein entspricht die Anzahl der Gewichte der Klassifikationsstärke des neuronalen Netzes. Je größer das Netz wird, umso mehr freie Parameter (Gewichte) hat es und desto mehr Stützstellen (Testmuster) werden auch benötigt, um das Netz zu trainieren. Ein zweiter Grund, warum mehr als eine Hidden-Ebene die Klassifikation nicht wesentlich verbessert ist, dass die Testmuster vor der Klassifikation skaliert werden, d.h., nur eine geringe Translationsvarianz aufweisen. Weiterhin ist auch die Rotationsvarianz der Testmuster eher gering und tendiert gegen Null. Eine Verbesserung der Translations- und Rotationsinvarianz hätte also nur einen geringen Einfluss auf die Klassifikation, weil sowohl Rotation, als auch Translation der Ziffern im praktischen Einsatz des Systems nur eine untergeordnete Rolle spielen.

Des Weiteren muss die optimale Anzahl der Hidden-Neuronen in Abhängigkeit vom Klassifikationsproblem gewählt werden. Dazu wurden verschiedene neuronale Netze erstellt, trainiert und validiert, um die optimale Anzahl von Hidden-Neuronen zu bestimmten. Abbildung 83 zeigt das Ergebnis der Untersuchung. Zur Untersuchung des Sachverhalts wurden zunächst neuronale Netze erstellt, die eine konstante Anzahl von Eingangs- und Ausgangsneuronen besitzen. Die Anzahl von Hidden-Neuronen wurde zwischen 2 und 40 variiert. Sodass 39 Klassen von neuronalen Netzen entstanden sind, wobei sich die Klassen untereinander nur durch die Anzahl der Hidden-Neuronen unterscheiden. Um ein besseres Ergebnis zu erzielen, wurden von jeder Klasse 20 Netze erzeugt. Insgesamt wurden also 760 Netze getestet. Nachdem die Netze trainiert wurden, wurde die Erkennungsrate auf Validierungsbildern gemessen. Zur Berechnung der Erkennungsrate wurden *15%* der Muster verwendet, während *85%* der Muster zum Training eingesetzt wurden. Bei der Gesamtzahl von 907 verwendeten Mustern entspricht dies also *770* Trainingsmustern und *137* Validierungsmustern. Die Anzahl der Validierungsmuster ist damit ausreichend groß, um die Erkennungsrate grob zu bestimmen. Später wurden zur genaueren Ermittlung der Erkennungsrate die Bilder der Validierungsmenge verwendet. Diese enthält bei *180* Validierungsbildern und *3* variablen Ziffern pro Bild $3 \cdot 180 = 540$ Validierungsmuster. Die Erkennungsraten pro Klasse wurden gemittelt. Wie Abbildung 83 zeigt, steigt die mittlere Erkennungsrate der Klassen bis etwa 15 Hidden-Neuronen an, bleibt danach aber auf einem konstant hohen Wert. Theoretisch müsste die Erkennungsrate nach dem optimalen Punkt zwar wieder absinken (vgl. Abbildung 55), in der Praxis wird das Training bei einer festen Erkennungsrate jedoch häufig abgebrochen, um das Overfitting des Klassifikators zu vermeiden. Die Erkennungsraten der Klassen mit mehr als 25 Hidden Neuronen wurden der Übersichtlichkeit wegen nicht dargestellt. Diese wiesen keine weitere Verbesserung auf.

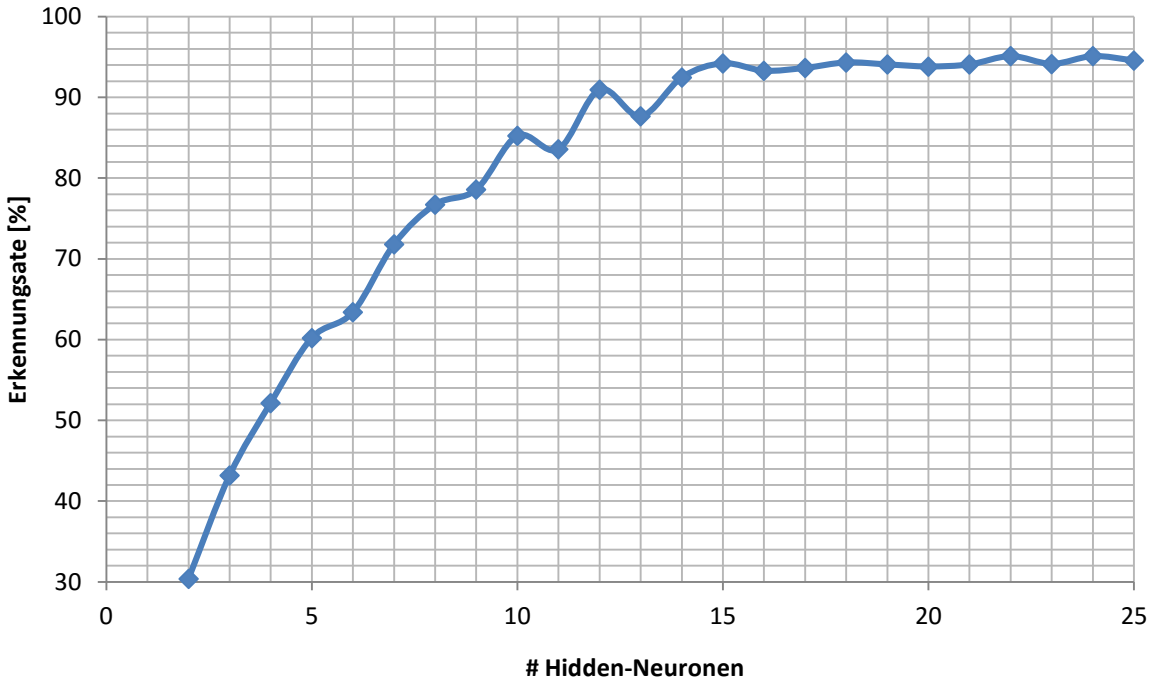


Abbildung 83: Experimentelle Bestimmung der Anzahl der Hidden Neuronen

4.2.10. Wahl der Trainingsparameter

Nachdem das MLP als Architektur und dessen Topologie gewählt wurde, gilt es, das MLP weiter zu optimieren. Insbesondere müssen die Trainingsparameter Momentum und die Lernrate bestimmt werden (vgl. Absatz 4.1.14). Da auch diese beiden Parameter problemspezifisch sind, wurden auch sie experimentell bestimmt. Nach dem selben Ansatz, der bereits zur Bestimmung der Anzahl der Hidden-Neuronen geführt hat, wurde auch hier wieder eine große Anzahl von neuronalen Netzen erstellt und trainiert. Die Topologie war bei allen Netzen identisch und so hatten alle Netze *312* Eingangsneuronen, *15* Hidden-Neuronen und *10* Ausgangsneuronen. Es wurden lediglich vor dem Training die beiden Trainingsparameter Momentum und Lernrate verändert. Während die Lernrate von *0,1* bis *0,7* in Schritten der Größe *0,01* verändert wurde, wurde das Momentum von *0* bis *0,9* in Schritten der Größe *0,01* angepasst. Erneut wurden von jeder so entstandenen Klasse genau *20* Netze erzeugt und die Erkennungsrate über alle Netze einer Klasse gemittelt. Die Klassen unterscheiden sich dabei untereinander nur durch das Momentum und die Lernrate. Zur Berechnung der Erkennungsrate wurden wieder *15%* der Muster verwendet während, *85%* der Muster zum Training eingesetzt wurden. Bei der verwendeten Gesamtzahl von 907 Mustern entspricht dies erneut einem Verhältnis von *770* Trainingsbildern und *137* Validierungsbildern. Insgesamt wurden *108 000* Netze erstellt und validiert, um einen möglichst großen Bereich untersuchen zu können und dabei gleichzeitig eine relativ genaue Auflösung bezüglich des Momentums und der Lernrate zu erzielen. Abbildung 84 zeigt das Ergebnis der Untersuchung. Es lässt sich feststellen, dass insbesondere niedrige Werte der Lernrate und niedrige bis mittlere Werte des Momentums sehr gute Erkennungsraten erzielen.

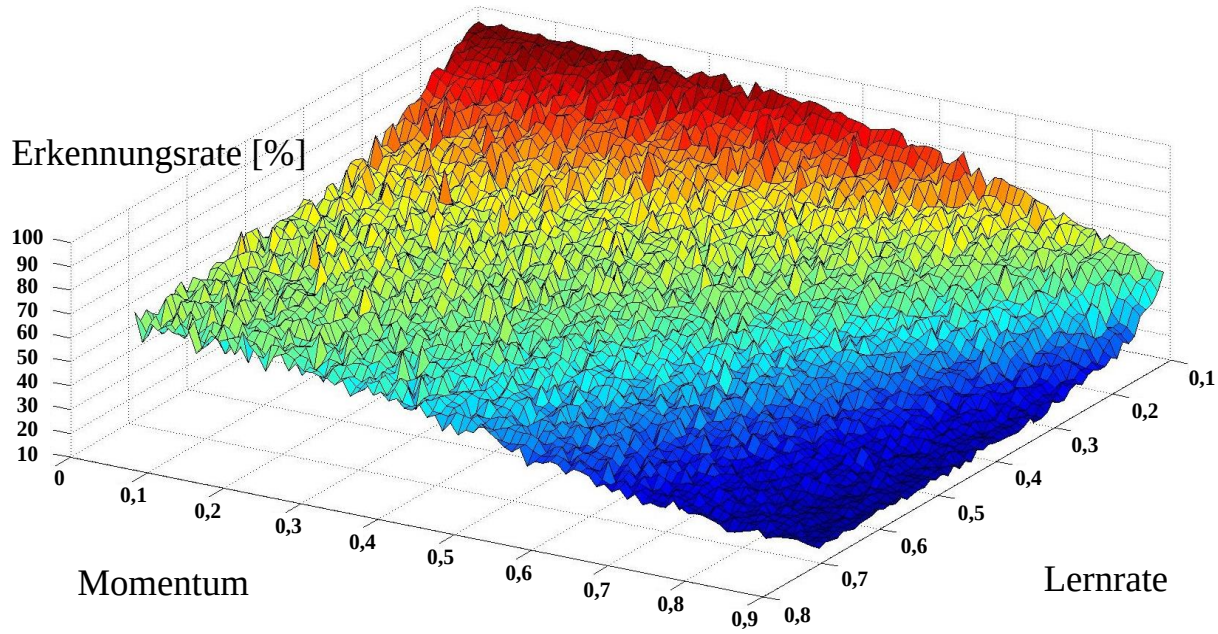


Abbildung 84: Wahl der Trainingsparameter Momentum und Lernrate

4.2.11. Trainingsmuster

Wie bereits in Abschnitt 4.1.13 erwähnt, führt eine größere Anzahl von Trainingsmustern auch zu einem stärkeren Klassifikator. Daher sind in vielen Forschungsgebieten auch umfangreiche Testbibliotheken vorhanden. So gibt es auf dem sehr umfangreichen Forschungsgebiet der optischen Zeichenerkennung große Bibliotheken von Testbildern, mit denen die Klassifikatoren trainiert werden können. Im Bereich der automatischen Zählerstanderfassung existieren solche Testbibliotheken jedoch nicht.

Um möglichst viele Trainingsmuster zu erhalten, wurde die Trainingsmenge aus verschiedenen Quellen zusammengesetzt. Einerseits wurden die Trainingsbilder aus der der International Conference on Document Analysis and Recognition (ICDAR)-2003-Trainingsmenge verwendet. Es wurden allerdings nur die Zeichen aus der Datenmenge verwendet, die Zahlen sind. Außerdem wurden Ziffern per Hand aus Bildern von Stromzählern und Druckbuchstaben von Word-Schriftarten ausgeschnitten und mit Rauschen überlagert. Insgesamt wurden so 907 Trainingsbilder erzeugt. Die gesamte Trainingsmenge wird in Trainingsbilder und Validierungsbilder unterteilt. Während die Trainingsbilder zum Trainieren des Netzes genutzt wurden, fanden die Validierungsbilder ausschließlich Verwendung, um die Erkennungsrate des neuronalen Netzes zu bestimmen.

Bevor die Testmuster auf das neuronale Netz gegeben werden, müssen sie auf eine einheitliche Größe skaliert werden. Daher werden die Muster vor der Klassifikation auf eine Größe von 26×12 skaliert. Außerdem hat sich gezeigt, dass das neuronale Netz besser mit

Grauwerten arbeiten kann als mit binären Bildern. Daher wird die Klassifikation auf Grundlage von Grauwerten durchgeführt. Des Weiteren ist es, wie bereits in Abschnitt 4.1.13 beschrieben, wichtig, dass auch der Wertebereich der Grauwerte skaliert wird, bevor das Muster klassifiziert werden kann. Eine besonders gute Klassifikation wird erreicht, wenn das Eingangsmuster mittelwertfrei ist und die Standardabweichung eins beträgt [8]. Die Grauwerte der Testmuster werden also unter Anwendung von Formel (45) skaliert.

$$X_{neu} = \frac{X_{alt} - Mittelwert}{Varianz} \tag{45}$$

Für das Training werden auch die Zielwerte der Ausgangsneuronen benötigt. Hierbei ist der Soll-Wert des Neurons, welches die Klasse des Musters anzeigt +1, alle anderen Neuronen haben den Soll-Wert -1.

Es wird die gleiche Funktion für das Vorverarbeiten der Testmuster verwendet, die auch später im praktischen Betrieb zum Einsatz kommt. Alle oben beschriebenen Verarbeitungsschritte werden auch im praktischen Betrieb mit echten Bildern durchgeführt.

4.2.12. Bereitstellung der Daten als Web Service

Bisher wurde nur das Mustererkennungssystem beschrieben, welches aus den Bildern des Stromzählers den Zählerstand extrahiert. Zur Vervollständigung des Smart Metering-Systems müssen die gewonnenen Daten allerdings auch abrufbar gemacht werden. Um dem Smart Metering-System die maximale Bandbreite von Einsatzmöglichkeiten zu ermöglichen, werden die Daten als Web Service im Netzwerk bereitgestellt. Dabei wurde das Devices Profile for Web Services (DPWS) verwendet (vgl. Abschnitt 2.2.8). Ein großer Vorteil von DPWS ist, dass es auch auf eingebetteten Systemen eingesetzt werden kann. Die Daten sollen auf zwei Arten vom Web Service (WS) angeboten werden:

- Einmalig: Es soll möglich sein, den aktuellen Zählerstand abzufragen. In diesem Fall wird einmalig eine Anfrage gestellt, welche vom WS direkt beantwortet wird, indem der WS den Zählerstand ausgibt.
- Fortlaufend: Es soll möglich sein, den aktuellen Zählerstand kontinuierlich zu erhalten. Auch in diesem Fall soll nur einmalig eine Anfrage gestellt werden. Der WS gibt den Zählerstand immer dann aus, wenn dieser sich ändert.

4.3. Implementierung

Im Folgenden wird auf die Implementierung des Systems eingegangen. Das System wurde zunächst als Prototyp in Matlab implementiert. Danach wurde die Funktionalität der Matlab-Codegenerierung genutzt, um aus den Matlab-Dateien des Prototyps Quellcode in der Programmiersprache C++ zu erzeugen. Es hat sich herausgestellt, dass die Berechnungszeit des Matlab-Systems auf einem Desktop-Prozessor mit x86-Architektur zwar relativ gut ist,

der generierte C++ Code auf dem Raspberry Pi jedoch eine relativ lange Berechnungszeit zum Erkennen des Zählerstandes benötigt. Tabelle 6 zeigt die ungefähren Werte für die Matlab-Systeme auf unterschiedlichen Plattformen. Es sei angemerkt, dass als Desktop-System ein zum aktuellen Zeitpunkt sehr starkes System verwendet wurde mit:

- Intel Core i7-3770 CPU mit 3,4 GHz
- 32 GB RAM
- 256 GB Solid State Drive (SSD)
- Windows 8.1 64-Bit Betriebssystem (Operating System, OS)

Dabei war es schwierig nachzuvollziehen, welche Teile des Programms auf dem Raspberry Pi besonders langsam arbeiten, insbesondere weil Optimierungen zwar im Matlab-Code, aber nur eingeschränkt im generierten Quellcode, möglich sind. Aus diesem Grund wurde das laufende System mit Hilfe der C++ Bibliothek OpenCV implementiert. Im Besonderen konnte das Programm somit auf C++-Ebene optimiert werden, um eine schnellere Berechnungszeit bei gleicher Erkennungsrate erzielen zu können.

CPU-Architektur, OS (IDE)	**Methode zur ROI-Extraktion**	**Programmlaufzeit für eine Zählerstanderkennung [s]**
x86, Windows 8.1 (Matlab)	Hough-Transformation	≈ 4
x86, Windows 8.1 (Matlab)	Vertikale Kanten	≈ 7
ARM, Raspbian	Hough-Transformation	≈ 52
ARM, Raspbian	Vertikale Kanten	≈ 625

Tabelle 6: Laufzeitabschätzung der Matlab-Systeme

4.3.1. Übersicht über den Workflow

Zunächst soll eine Übersicht, über die Bearbeitungsstufen vom Quellcode hin zum ausführbaren Programm gegeben werden. Im Zuge dessen werden auch die verwendeten Tools benannt und ggf. erklärt. Der Workflow ist zwischen der Matlab-Implementierung und der OpenCV-Implementierung in nativem C++ unterschiedlich.

Zunächst soll auf den Matlab-Workflow eingegangen werden. Abbildung 85 zeigt die einzelnen Prozessschritte. Das System wurde zunächst mit Matlab unter Windows 8 implementiert und somit die Matlab-Dateien erstellt. Mit der Matlab-Codegenerierung wurden aus den Matlab-Dateien C++-Dateien erzeugt. Da mit der Matlab-Codegenerierung nur C++-Funktionen, nicht aber ausführbare Programme erzeugt werden können, muss nun ein Wrapper (Hüllprogramm) um die erzeugten Funktionen implementiert werden. Der Wrapper sorgt dafür, dass das Programm Bilddateien aus dem Speicher oder Bilder von der Kamera des Raspberry Pi einlesen kann. Für das Einlesen von Bilddateien und das

Zuschneiden dieser Bilder werden OpenCV-Funktionen, zum Ansprechen der Kamera des Raspberry Pi die Softwarebibliothek Raspicam verwendet [28]. Daher müssen sowohl die OpenCV-Header-Dateien als auch die Header-Dateien der Raspicam Bibliothek mit eingebunden werden. Des Weiteren werden einige Matlab-spezifische Header- und Source-Dateien benötigt, in denen allgemeine Funktionen und Datentypen spezifiziert werden, die in den von Matlab erzeugten C++-Dateien Verwendung finden. Als Entwicklungsumgebung (Integrated Development Environment, IDE) wurde Eclipse verwendet, um Implementierung und Debugging des Systems möglichst effizient zu gestalten. Es wäre auch möglich, den Wrapper um die von Matlab erzeugten C++-Files direkt auf dem Raspberry Pi unter Raspbian zu implementieren. Allerdings sind unter Raspbian nur Texteditoren verwendbar. Für den Betrieb einer IDE reichen die Hardwareressourcen des Raspberry Pi nicht aus. Daher wurde die DIE unter dem Linux-Betriebssystem Ubuntu 14.04 verwendet, weil dieses in vielen Situationen ein ähnliches Verhalten aufweist, wie das später auf dem Raspberry Pi verwendete Raspbian. Nach der abschließenden Anpassung der Quelldateien unter Ubuntu, wurden die Dateien auf dem Raspberry Pi mit CMake zu einem Projekt zusammengeführt, dabei wurde sowohl OpenCV als auch Raspicam als Bibliothek (Shared Object, SO) angebunden. Zum Schluss wurde mit dem make-Programm eine ausführbare Datei erzeugt.

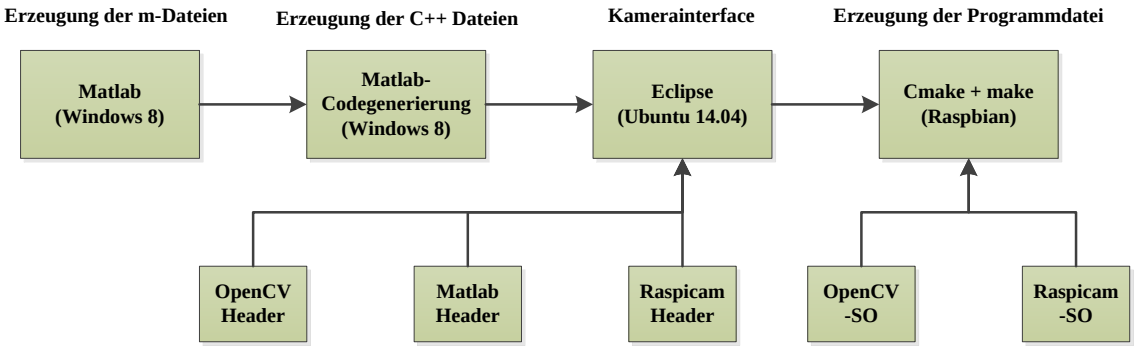


Abbildung 85: Workflow für Matlab Projekt

In der OpenCV-Implementierung des Systems konnte der Workflow etwas abgekürzt werden. Abbildung 86 zeigt den Workflow in der schematischen Übersicht. Zunächst wurde das System mit der Eclipse-IDE unter Ubuntu implementiert. Dabei wurden wieder die Header-Dateien der Bibliotheken OpenCV und Raspicam hinzugefügt. Danach wurden die Quelldateien wieder auf dem Raspberry Pi unter Raspbian mit CMake zu einem Projekt zusammengefügt und mit dem make-Programm eine ausführbare Datei erzeugt. Der kürzere Workflow der OpenCV-Implementierung hatte auch beim Entwurf des Systems entscheidende Vorteile. Änderungen im Programm konnten durch den kürzeren Workflow deutlich schneller getestet werden.

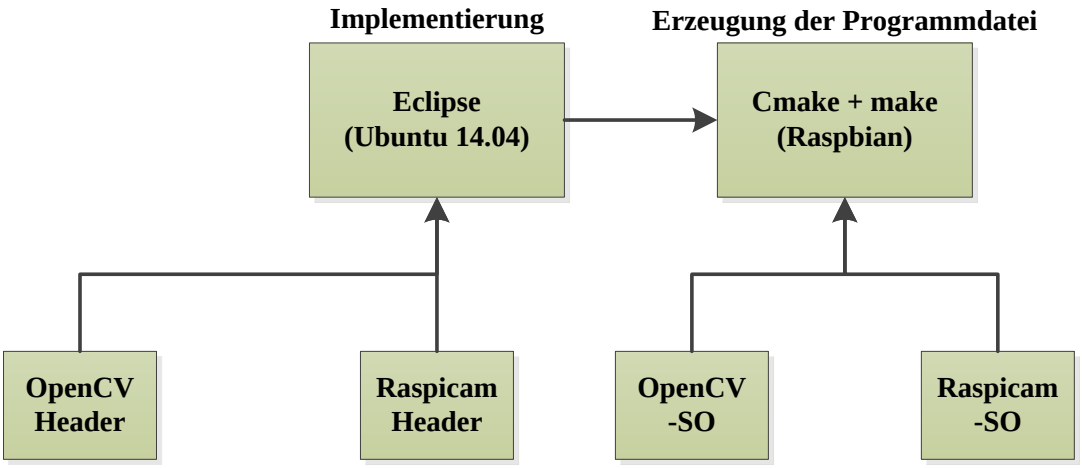


Abbildung 86: Workflow für OpenCV Projekt

4.3.2. Raspberry Pi als Hardwareplattform

In diesem Abschnitt wird der Raspberry Pi als Hardwareplattform vorgestellt. Der Raspberry Pi ist ein Mini-PC von der Größe einer Kreditkarte. Mehrere Universal Serial Bus (USB)-Ports und ein High Definition Multimedia Interface (HDMI)-Anschluss erlauben es, den Raspberry Pi mit Tastatur, Maus und Bildschirm zu verwenden. Im Rahmen dieser Arbeit wurde der Raspberry Pi jedoch vorwiegend über den Ethernet-Anschluss mit einer Kommandozeile über das Secure Shell-Netzwerkprotokoll betrieben. Von dem Raspberry Pi gibt es die drei Modelle A, B und B+. In dieser Arbeit wurde das Model B des Raspberry Pi verwendet.

Als Betriebssystem wurde Raspbian gewählt. Dabei handelt es sich um eine optimierte Version des Linux-Betriebssystems Debian. Insbesondere wurde das Betriebssystem Raspbian an den Prozessor des Raspberry Pi mit der ARM-Architektur und an die relativ geringe Rechenleistung des Raspberry Pi angepasst. Unter anderem wird beim Start des Betriebssystems nicht automatisch der rechenintensive X-Server gestartet. So kann das System nur mit einem Terminal als Benutzerschnittstelle verwendet werden, was sich positiv auf den Verbrauch von Rechenressourcen auswirkt. Abbildung 87 zeigt das verwendete Modell des Raspberry Pi, Tabelle 7 und Tabelle 8 zeigen ausgewählte Daten aus der Hardwarespezifikation des verwendeten Raspberry Pi Models und des Kameramoduls [29].

Abbildung 87: Raspberry Pi Model B

Prozessor-Typ	BCM2835	**HDMI-Port**	1
CPU-Architektur	ARM	**USB-Ports**	2
CPU-Takt	700 MHz	**LCD-Display-Schnittstelle**	1
SDRAM	512MB	**Kamera-Schnittstelle**	1
MicroUSB-Ports	1	**SD-Karten-Slot**	1
3,5mm Audio-Buchse	1	**Abmessungen**	85,6 x 53,98 x 17 mm
Ethernet Port	1	**Gewicht**	9 g

Tabelle 7: Daten aus der Hardwarespezifikation des Raspberry Pi Model B

Auflösung	2592 x 1944 Pixel	**Abmessungen**	25 x 20 x 9 mm
(fester) Fokus	1m bis unendlich	**Gewicht**	3 g

Tabelle 8: Daten des Kameramoduls des Raspberry Pi

4.3.3. Matlab-System

In den nachfolgenden Ausführungen wird genauer auf das implementierte Matlab-System eingegangen.

Matlab bietet eine große Anzahl von Funktionen zur Bilderverarbeitung, zur Mustererkennung im Allgemeinen, aber auch speziell zu neuronalen Netzen. Viele der im System benötigten Funktionen (vgl. Abschnitt 0) werden von Matlab unterstützt. Relativ wenige Funktionen müssen für einen Prototypen neu implementiert werden, der nur in Matlab

lauffähig ist. Um jedoch ein Programm zu entwickeln, dass auch auf dem Raspberry Pi lauffähig ist, müssen deutlich mehr Funktionen neu implementiert werden. Obwohl Matlab eigentlich eine große Anzahl von Funktionen unterstützt, ist es nur für einen relativ kleinen Teil dieser Funktionen möglich, mit Hilfe der Codegenerierung Quellcode (C/C++) zu erzeugen, der auch auf dem Raspberry Pi lauffähig ist. Es wurden einige Funktionen, die nicht von der Codegenerierung unterstützt werden, neu implementiert. Die Namen dieser Funktionen sind dabei an die Namen der originalen Matlab-Funktionen angelehnt. Die Parameter der Funktionen sind meist identisch mit den Parametern der originalen Matlab-Funktionen, um die Austauschbarkeit beider Varianten zu gewährleisten. Da Matlab, wie jede Programmiersprache, keine zwei Funktionen mit demselben Namen erlaubt, wurden die Funktionsnamen der neuen Funktionen in der Form „my“<Name der Originalfunktion> gewählt. Abbildung 88 zeigt den Ablaufplan des Programms in Matlab. Das gesamte System ist in der Matlab-Funktion *MeterReadingLooped()* zusammengefasst. Im Folgenden werden die wichtigsten Funktionen des Systems erklärt.

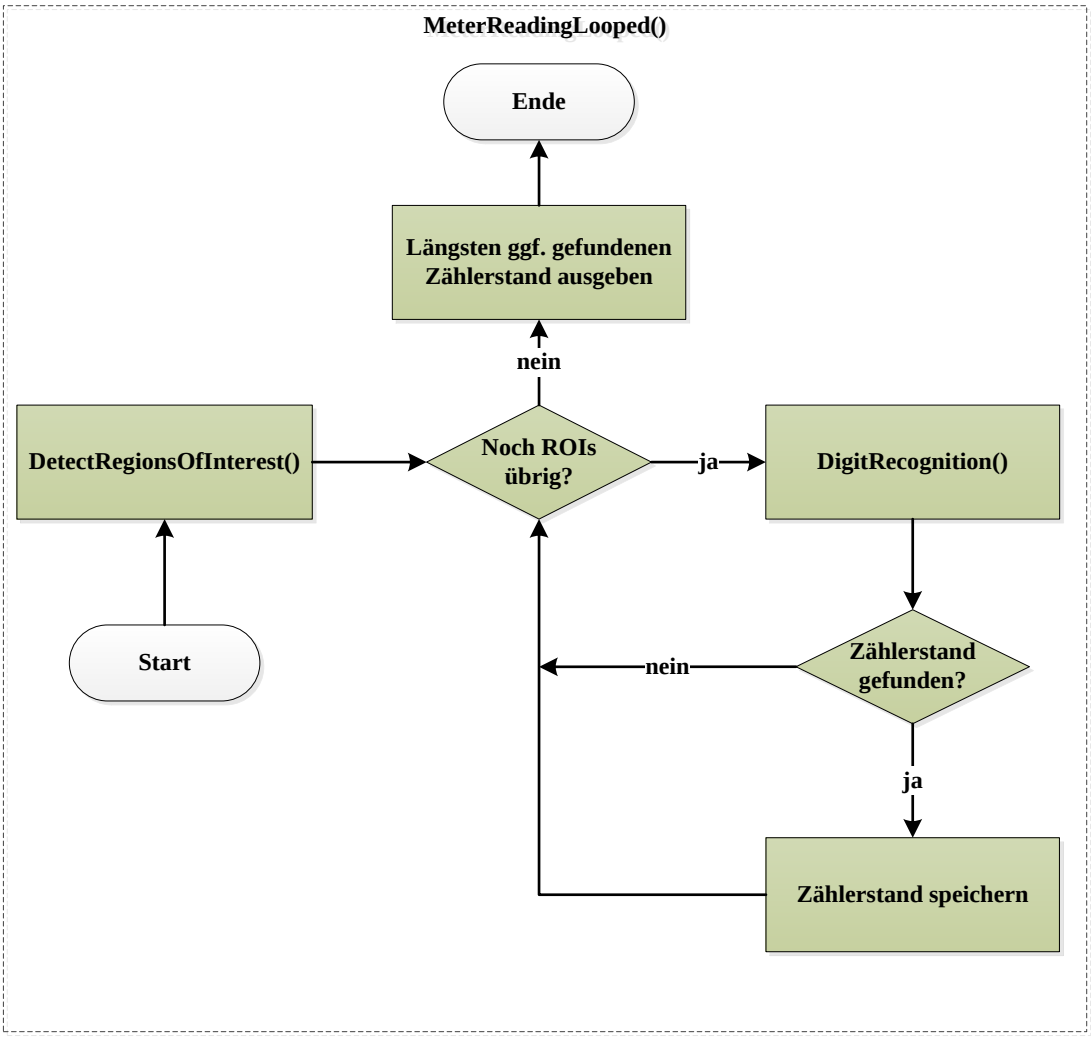


Abbildung 88: Ablaufplan des Programms in Matlab

- *DetectRegionsOfInterest()*: Diese Funktion ist eine Hüllfunktion, um die Funktionen zur Detektion der ROIs aufzurufen. Der Funktion kann die Methode der ROI-Detektion mit übergeben werden, wobei aktuell 2 Varianten unterstützt werden: Die ROI-Detektion auf Grundalge der Hough-Transformation (vgl. Abschnitt 4.2.4) und die ROI-Detektion auf Grundlage der vertikalen Kanten im Bild (vgl. Abschnitt 4.2.5).
- *DigitRecognition()*: Diese Funktion bekommt ein ROI übergeben und gibt einen ggf. erkannten Zählerstand wieder zurück. Die Funktion ruft zunächst die Funktion *ForegroundDetection()* auf, die das Eingangsbild segmentiert und die größten *N* Objekte zurückgibt. Danach werden die segmentierten Objekte einzeln auf das neuronale Netz gegeben und dann die Suche nach Zeichenketten (vgl. Abschnitt 4.2.7) auf die Objekte angewendet. Wenn sich die gefundenen Objekte als Zeichen einer Zeichenkette erweisen, dann werden sie als Zählerstand ausgegeben. Abbildung 89 zeigt den Ablauf der Funktion Digit Recognition in Matlab.

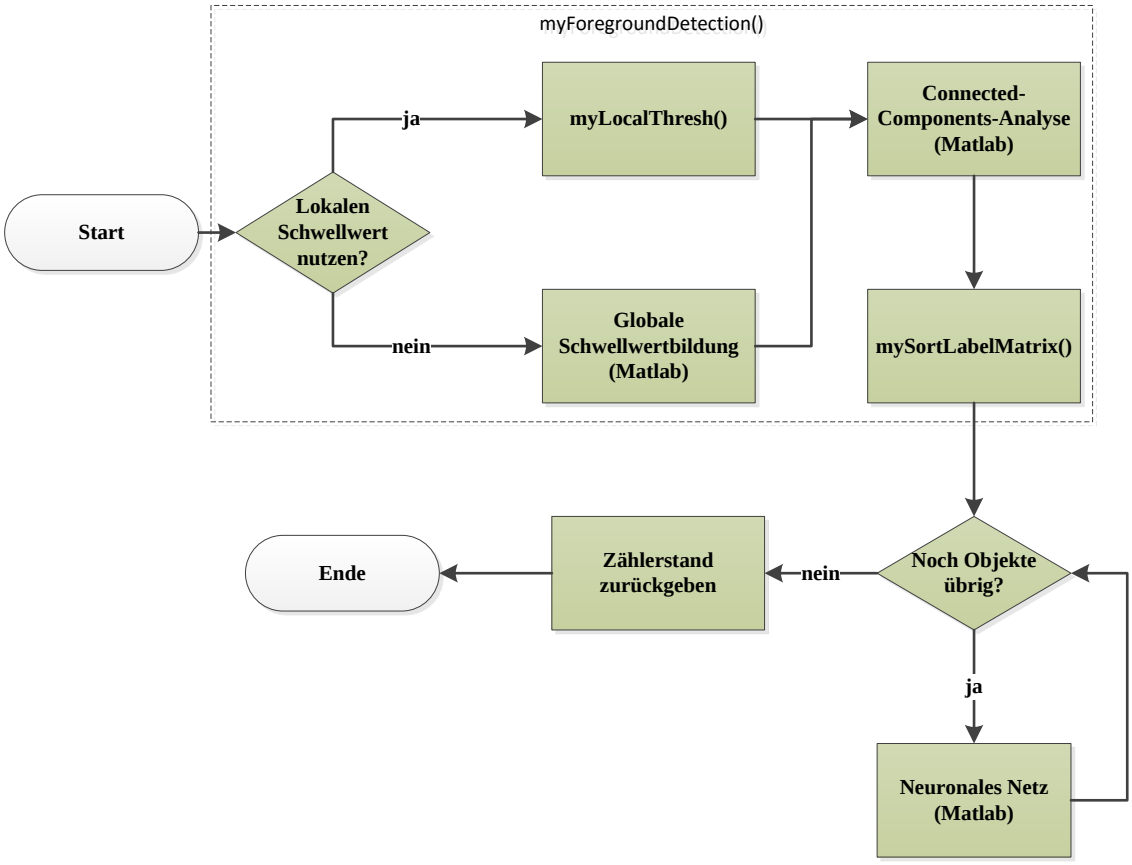


Abbildung 89: Ablauf der Funktion DigitRecogintion() in Matlab

- *ForegroundDetection():* Diese Funktion bekommt ein Bild übergeben und gibt die segmentierten Vordergrundobjekte zurück. Hierzu wird das Bild ggf. zunächst von dem RGB-Farbraum in Graustufen umgerechnet. Danach wird eine Schwellwertbildung vorgenommen, um ein Binärbild zu erzeugen. Dabei kann zwischen lokalen und globalem Schwellwert gewählt werden. Im Falle eines lokalen

Schwellwertes, werden nach der Schwellwertbildung die Funktionen der Erosion und Dilatation auf das Bild angewandt, um ggf. auftretendes Segmentierungsrauschen zu unterdrücken (vgl. Abschnitt 4.1.5). Als Default-Einstellung wurde ein globaler Schwellwert gewählt, insbesondere, weil die lokale Schwellwertbildung deutlich mehr Rechenzeit benötigt. Tabelle 9 zeigt diesbezüglich die Ergebnisse einer Versuchsreihe im Matlab-Programm unter Windows 8. Zur lokalen Schwellwertbildung wird die Funktion *myLocalThresh()* verwendet, die globale Schwellwertbildung wird mit Hilfe einer Matlab-Funktion berechnet. Nach der Schwellwertbildung werden dann mittels der Connected-Components-Analyse Objekte im Binärbild gesucht und den gefundenen Objekten werden Labels gegeben. Das bedeutet, dass alle Hintergrundpixel den Wert bzw. das Label 0 haben. Die Pixel des ersten Objektes haben den Wert 1 usw. bis zum letzten Objekt. Die gefundenen Objekte werden in der Folge der Größe nach sortiert und die größten 15 Objekte werden zurückgegen. Bei Versuchen auf der Validierungsmenge hat sich ein Wert von 15 als guter Wert für eine hohe Erkennungsrate auf dem Validierungsdatensatz herausgestellt. Zur Sortierung und Auswahl der größten Objekte wird die Funktion *mySortLabelMatrix* verwendet.

Methode	**Programmlaufzeit [s]**
Hough + lokaler Schwellwert	5,1
Hough + globaler Schwellwert	0,73
Edge + lokaler Schwellwert	6,9
Edge + globaler Schwellwert	1,5

Tabelle 9: Laufzeiten des Matlab-Programms unter Windows 8 in Abhängigkeit vom der Schwellwertmethode

- *myLocalThresh():* Diese Funktion bekommt ein Grauwertbild übergeben und führt eine lokale Schwellwertbildung durch. Matlab selbst bietet nur Funktionen zur Berechnung eines globalen Schwellwertes an. Zur lokalen Schwellwertbildung wird ein Fenster über das Bild geschoben und aus allen Pixeln unter dem Fenster ein Schwellwert berechnet. Der errechnete Schwellwert wird dann auf den zentralen Pixel unter dem Fenster angewendet. Jeder der lokalen Schwellwerte kann entweder nach der Niback-Formel (40) oder nach der Methode von Otsu berechnet werden. Es hat sich gezeigt, dass die Methode nach Otsu etwas schneller ist, als die Berechnung auf Grundlage der Niback-Formel.
- *mySortLabelMatrix():* Diese Funktion bekommt ein durch Segmentierung entstandenes Binärbild übergeben, auf welchem den einzelnen Objekten bereits Labels zugeordnet wurden. Den Objekten werden dann, in Abhängigkeit von ihrer Größe, neue Labels zugeordnet. Nur eine bestimmte Anzahl von Objekten wird

weiterverarbeitet, diese Anzahl wurde auf 15 Objekte festgelegt. Zunächst berechnet die Funktion das Histogramm des Eingangsbildes. Implizit werden die Labels im Eingangsbild als Grauwerte interpretiert. Da jedes Objekt im Bild genau ein Label hat, entspricht die Häufigkeit, mit der die Labels im Histogramm auftreten, der Anzahl der Pixel, die zu dem jeweiligen Objekt gehören. Der erste Eintrag im Histogramm wird gelöscht, weil dieser dem Label 0 im Eingangsbild entspricht, welches zum Hintergrund gehört. Danach werden die 15 größten Objekte im Bild gesucht und diesen wird ein neues Label zugewiesen.

Aus dem in Abbildung 88 gezeigten Matlab-Programm wurde dann, wie in Abschnitt 4.3.1 beschrieben, mittels Codegenerierung C++-Quellcode erzeugt. Um den erzeugen Quellcode herum wurde ein Wrapper-Programm implementiert. Das Wrapper-Programm bietet die Funktionalität, Bilder entweder von dem Kamera-Modul des Raspberry Pi oder vom Speicher (SD-Karte) einzulesen. Das Wrapper-Programm wurde in der Programmiersprache C++ und mit Hilfe der Softwarebibliothek OpenCV implementiert. Abbildung 90 zeigt den Ablaufplan für einen Durchlauf des Wrapper-Programms. Zur Verbesserung der Übersichtlichkeit wurde nur ein Durchlauf des Programms dargestellt. Praktisch läuft das Programm entweder in einer Endlosschleife (Normalbetrieb) oder liest die Bilder der Validierungsmenge ein (Validierungsmodus). Der Normalbetrieb arbeitet in einer Endlosschleife, um eine kontinuierliche Erkennung des Zählerstandes zu gewährleisten. Zwischen Validierungsmodus und Normalbetrieb kann beim Kompilieren der Software per Define-Anweisung gewählt werden. Im Folgenden werden die aufgerufenen Funktionen kurz erklärt.

- *captureImage()*: Diese Funktion wird aufgerufen, um ein Bild einzulesen. Dabei wird, in Abhängigkeit vom Betriebsmodus entweder die Funktion *getImage()* oder die Funktion *imread()* aufgerufen. Bei *getImage()* handelt es sich um eine selbstimplementierte Funktion, welche die Raspicam-Softwarebibliothek nutzt und ein Bild von der Kamera des Raspberry Pi einliest. Die Funktion *imread()* ist eine OpenCV Funktion, die eine Bilddatei aus dem Speicher lädt.
- *MeterReadingLooped_initialize():* Diese Funktion muss laut der Dokumentation der Matlab-Codegenerierung aufgerufen werden, bevor alle anderen von Matlab erzeugten Funktionen aufgerufen werden dürfen.
- *MeterReadingLooped():* Diese Funktion wurde mit Hilfe der Matlab-Codegenerierung aus dem gleichnamigen Matlab-Programm erzeugt, welches in Abbildung 88 dargestellt ist. Bei *MeterReadingLooped()* handelt es sich um die eigentliche Erkennungsfunktion, die ein Bild übergeben bekommt und den Zählerstand zurückgibt.

- *MeterReadingLooped_terminate():* Diese Funktion muss laut der Dokumentation der Matlab-Codegenerierung aufgerufen werden, nachdem alle anderen von Matlab erzeugten Funktionen aufgerufen wurden.

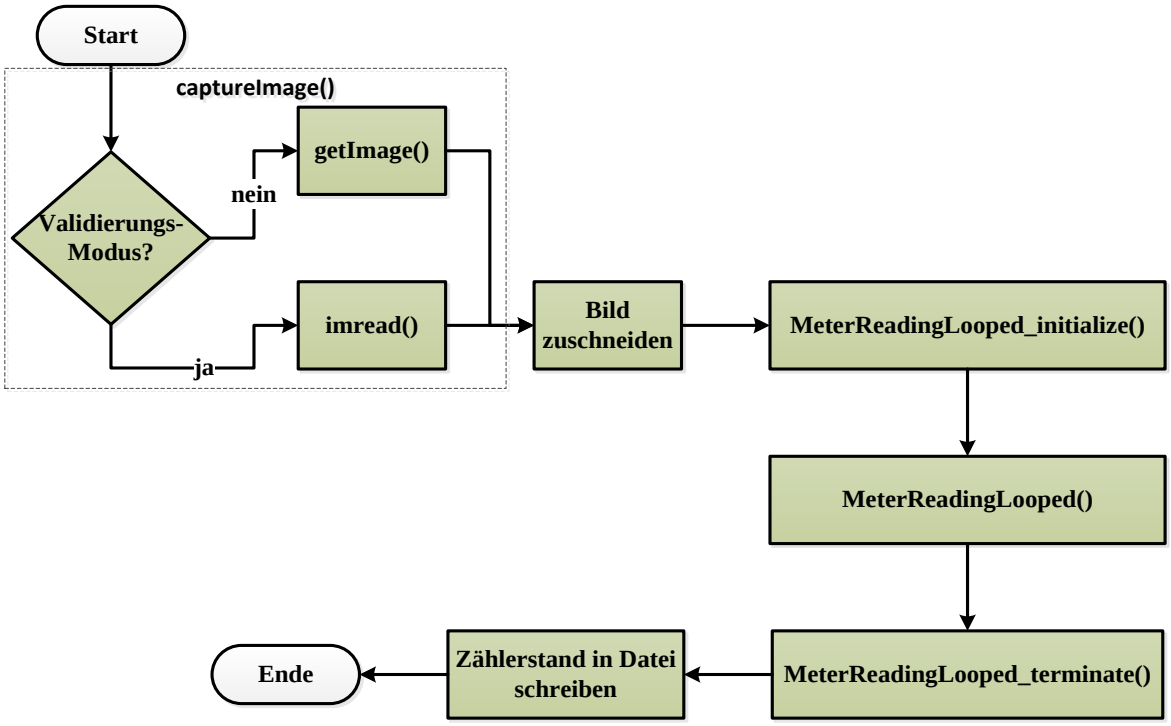


Abbildung 90: Ein Durchlauf des Wrapper-Programms für das Matlab--System

4.3.4. OpenCV-System

Aufgrund der relativ langen Berechnungszeit, welche das Matlab-System auf dem Raspberry Pi benötigt (vgl. Tabelle 6), wurde das Smart Metering-System in der C++ Softwarebibliothek OpenCV implementiert. OpenCV wurde für Einsatzszenarien in der Bildverarbeitung entwickelt, bietet aber auch einige Funktionen zur Mustererkennung. Insbesondere für die Verwendung neuronaler Netze, bietet OpenCV jedoch nicht den Funktionsumfang von Matlab mit seinen vielseitigen Toolboxen. Abbildung 91 zeigt den Ablauf eines Programmdurchlaufs mit OpenCV. Praktisch läuft das Programm, genau wie das Matlab-Programm, entweder in einer Endlosschleife (Normalbetrieb) oder liest in einer Schleife der Reihe nach alle Bilder der Validierungsmenge ein (Validierungsmodus). Zwischen Validierungsmodus und Normalbetrieb kann auch beim OpenCV-System vor dem Kompilieren der Software per Define-Anweisung gewählt werden. Im Folgenden werden die in Abbildung 91 genannten Funktionen und Methoden erklärt.

- *captureImage()*: Ist identisch mit der *captureImage()*-Funktion im Wrapper des Matlab-Programms und liest entweder eine Bilddatei aus dem Speicher oder ein Bild von der Kamera.

- *preprocessing()*: Implementiert die Vorverarbeitung, d.h. das Zuschneiden des Eingangsbildes und ggf. die Transformation vom RGB-Farbraum in ein Graustufenbild (vgl. Abschnitt 4.2.2).
- *detect()*: Diese Methode ist Bestandteil der Klasse *RegionOfInterestDetectorEdge*, welche die ROI-Detektion auf Grundlage der vertikalen Kanten im Bild implementiert. Die *detect()*-Methode bekommt ein Grauwertbild übergeben und gibt ein Array von Bounding Boxes um die gefundenen ROIs zurück. Eine Bounding Box ist ein kleines Viereck, welches sich um ein Objekt legen lässt.
- *segment()*: Diese Methode ist Bestandteil der Klasse *CharacterSegmentationClass* und führt die Segmentierung eines ROI durch. Die *segment()*-Methode bekommt das ROI als Grauwertbild übergeben und gibt ein Array mit gefundenen Bild-Objekten zurück. Die verwendeten Bild-Objekte haben dabei drei Member-Variablen:
    - Die Bounding Box um das Bild-Objekt,
    - einen booleschen Wert, der anzeigt, ob das Objekt zu einer Zeichenkette gehört,
    - den Wert des Objektes, den das Objekt später bei der Ziffernerkennung vom neuronalen Netz zugewiesen bekommt, im Idealfall entspricht dieser Wert der Ziffer, die das Objekt darstellt.
- *findChain()*: Diese Methode ist Bestandteil der Klasse *ChainSearch*. Die *findChain()*-Methode implementiert die Suche nach Zeichenketten (vgl. Abschnitt 4.2.7). Sie bekommt ein Array mit gefundenen Objekten übergeben und gibt dasselbe Array wieder zurück, wobei die Methode alle Bild-Objekte löscht, die nicht zu einer Zeichenkette gehören.
- *digitRecognition()*: Die Funktion führt die Ziffernerkennung durch. Sie bekommt ein Array mit Objekten übergeben und gibt alle gültigen Bild-Objekte auf das neuronale Netz. Die gültigen Bild-Objekte sind dabei jene, die nicht von *findChain()* gelöscht wurden. Das neuronale Netz setzt den Ziffern-Wert der Objekte.

Zum Schluss erfolgt die Ausgabe des Zählerstandes auf der Konsole und in eine Textdatei, sodass diese vom Web Service-Programm ausgelesen werden kann.

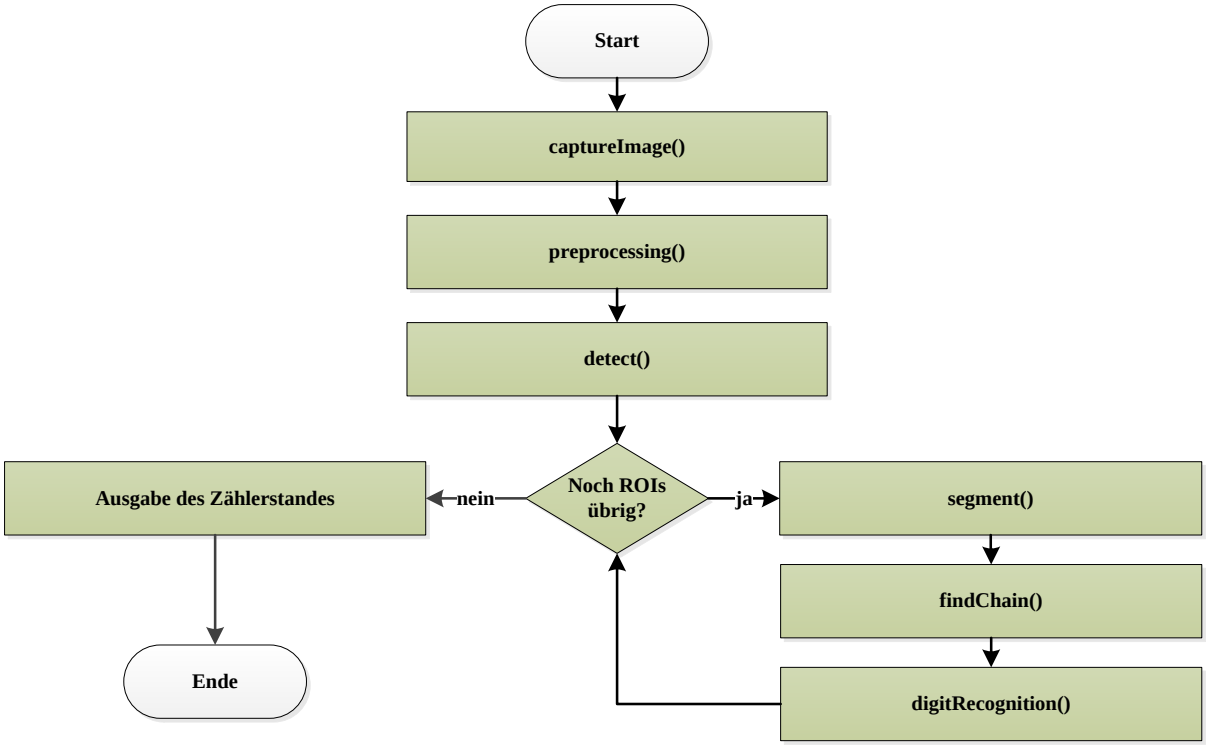


Abbildung 91: Ablaufplan eines Durchlaufs des OpenCV-Programms

In Abschnitt 4.2.10 wurde auf die Suche nach den optimalen Trainingsparametern eingegangen. Die Erkennungsrate des neuronalen Netzes hängt aber auch von der Initialisierung der Gewichte ab. Daher muss, nachdem geeignete Trainingsparameter gefunden wurden, ein konkretes neuronales Netz ausgewählt werden. Zu diesem Zweck wurden mit einem C++-Programm 660 neuronale Netze erzeugt und trainiert. In Abbildung 84 zeigt sich, dass besonders hohe Erkennungsraten bei geringen Lernraten unter 0,2 und mittleren Momentum-Termen unter 0,5 auftreten. Daher wurden die neuronalen Netze mit folgenden Parametern erzeugt:

- Momentum: Als Startwert für das Momentum wurde der Wert 0 gewählt, der Endwert lag bei 0,5. Die Schrittweite lag bei 0,05.
- Lernrate: Als Startwert für die Lernrate wurde der Wert 0,1 gewählt, der Endwert lag bei 0,2. Die Schrittweite lag erneut bei 0,05.
- Mit jeder Kombination aus Momentum und Lernrate wurden 20 neuronale Netze trainiert.

Jedes der 660 erzeugten Netze wurde zusammen mit dem gesamten OpenCV-System auf dem Validierungsdatensatz von 180 Bildern validiert. Anschließend wurde das Netz mit der besten Erkennungsrate ausgewählt. Das ausgewählte Netz wurde mit einem Momentum von 0,2 und einer Lernrate von 0,1 trainiert.

4.3.5. Anbindung und Funktion des Web Service

Bisher wurde auf die Implementierung des Mustererkennungssystems eingegangen, im Folgenden wird die Bereitstellung der Daten und die Kommunikation zwischen dem Web Service und dem Mustererkennungssystem dargestellt.

Der Web Service wurde mit dem DPWS-Stack JMEDS in der Programmiersprache Java implementiert. JMEDS bietet vor allem zwei Vorteile:

- Geringer Ressourcen-Verbrauch: Da es sich bei JMEDS um eine Implementierung von DPWS handelt, ist JMEDS auch auf eingebetteten Systemen mit eingeschränkten Ressourcen einsetzbar.
- Portabilität: Da JMEDS in der Programmiersprache JAVA implementiert wurde, läuft das gesamte Programm in einer Java-Laufzeitumgebung und muss nicht mehr explizit für den Raspberry Pi kompiliert werden.

Der implementierte Web Service hat dabei folgende Struktur: Als Device (Hosting Service) dient das MeterReadingDevice, welches nur einen (Hosted) Service anbietet, den MeterReadingService. Dieser bietet wiederum 3 aufrufbare Operationen an: InitializeMeterReading, GetMeterReading, MeterReadingSubscription.

Die Operation **GetMeterReading** gibt den aktuellen Zählerstand zurück. Dabei wird der letzte, korrekt ausgelesene Zählerstand ausgegeben.

Die Operation **MeterReadingSubscription** ist die wichtigste Operation des Systems. Sie ermöglicht es einen Service zu abonnieren, welcher den Abonnenten von jeder Änderung des Zählerstands in Kenntnis setzt und dabei immer den aktuellen Zählerstand übermittelt.

InitializeMeterReading ist eine Operation, die aktuell keinen Einfluss auf die Funktionsweise des Systems hat. Sie ist dazu gedacht, den Zählerstand erstmals an das Smart Metering-System zu übertragen, um dieses ggf. zu initialisieren. Allerdings wird diese Funktion aktuell nicht für das System benötigt. Für etwaige Systemerweiterungen kann die Operation jedoch genutzt werden.

Wie bereits erwähnt, handelt es sich bei dem System zur Mustererkennung und dem Web Service zur Bereitstellung der Daten um zwei unterschiedliche Programme. Die **Kommunikation** zwischen diesen beiden Programmen wurde über das Dateisystem realisiert. Dabei kann das Mustererkennungssystem in eine Datei nur Daten schreiben und der Web Service kann aus dieser Datei nur Daten lesen. Abbildung 92 zeigt die Struktur des Web Services.

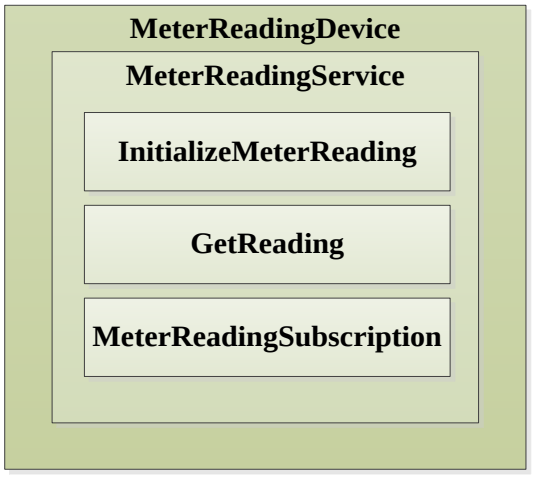


Abbildung 92: Struktur des Web Services

4.4. Ergebnisse und Auswertung

In diesem Kapitel werden die Ergebnisse der Arbeit vorgestellt und bewertet. Des Weiteren werden die zwei implementierten Ansätze zur ROI-Detektion miteinander verglichen.

4.4.1. Experimentelle Ergebnisse

In diesem Abschnitt werden die experimentell ermittelten Ergebnisse bezüglich der Erkennungsrate und der Laufzeit vorgestellt. Der Wert der Erkennungsrate bezieht sich auf die Erkennungsrate pro Ziffer des Zählerstandes. Der Wert der Laufzeit bezieht sich auf die benötigte Zeit für einen Programmdurchlauf, d.h. die Zeit vom Einlesen eines Bildes bis zur Ausgabe des Zählerstandes. Die Erkennungsrate ist bei der automatischen Zählerwerterfassung der wichtigste Parameter. Die Laufzeit ist weniger relevant, doch je geringer die Laufzeit ist, desto häufiger kann der Zählerstand bestimmt werden und desto höher ist die zeitliche Auflösung, wenn der Verbrauch über die Zeit dargestellt werden soll. Weiterhin kann mit einer kurzen Laufzeit besser über verschiedene Erkennungsergebnisse gemittelt werden, damit sinkt die Anfälligkeit gegenüber Fehlern bzw. Ausreißern. Die Laufzeitmessung erfolgte neben der Messung der Erkennungsrate. Die Laufzeit entspricht dem Mittelwert der Erkennungszeiten aller Bilder im Validierungsdatensatz. Die Standardabweichung der Laufzeit ist bei allen Systemen und auf allen Plattformen in Klammern hinter der Laufzeit aufgeführt. Da bei der Berechnung der Laufzeit nicht die Zeit für die Bildaufnahme von der Kamera des Raspberry Pi beachtet wird, muss diese gesondert betrachtet werden. Tabelle 10 zeigt die Ergebnisse der Messungen. Die Zeit pro Bildaufnahme wurde mit und ohne den Web Service-Prozess im Hintergrund gemessen. Des Weiteren wurde einmal mit normaler CPU-Taktfrequenz von 700 MHZ und einmal mit auf 950 MHz übertakteter CPU gemessen. Für alle Messungen wurde die Funktion *getImage()* verwendet, die auch im Normalbetrieb der Software zum Einsatz kommt. Die Messung wurde über die

Aufnahme von 100 Bildern gemittelt, wobei sich eine zu vernachlässigende Standardabweichung von unter einem Prozent ergab. Es lässt sich allgemein festhalten, dass die Zeit pro Bildaufnahme erstens relativ gering ist und zweitens keine Abhängigkeit von der Übertaktung der CPU oder dem Laufen des WS-Prozesses im Hintergrund aufweist. Beides lässt sich einerseits dadurch erklären, dass bei der Bildaufnahme vorwiegend das Kameramodul des Raspberry Pi arbeitet und weniger die CPU zum Einsatz kommt und andererseits dadurch, dass der zur Implementierung des WS verwendete JMEDS-Stack einen relativ geringen Ressourcenverbrauch aufweist.

CPU übertaktet	**WS-Prozess im Hintergrund**	**Zeit pro Bildaufnahme [s]**
Ja	Ja	0,2171
Ja	Nein	0,2187
Nein	Ja	0,2191
Nein	Nein	0,2176

Tabelle 10: Messung der benötigten Zeit für die Bildaufnahme

Die Erkennungsraten und die Laufzeiten der verschiedenen Systeme wurden mit Hilfe des umfassenden Validierungsdatensatzes bestimmt, welcher aus insgesamt 180 Bildern besteht. Von den 180 Bildern wurden 90 bei Kunstlicht aufgenommen und 90 bei natürlichem Licht. Der Messaufbau wird in Abbildung 64 dargestellt. Von jedem der 9 Messpunkte wurden jeweils 10 Bilder aufgenommen. Jedes Bild zeigt einen unterschiedlichen Zählerstand. Während die ersten 3 Ziffern bei allen 180 Bildern gleich sind (499), sind die letzten 3 Ziffern in jedem Bild unterschiedlich (000, 111, …, 888, 999). Die Validierung wurde auf zwei verschiedenen Plattformen durchgeführt, einmal in einer virtuellen Maschine (VM) unter dem Linux-Betriebssystem Ubuntu 14.04 und einmal auf dem Raspberry Pi unter dem Linux-Betriebssystem Raspbian. Die Laufzeitmessung auf dem Raspberry Pi wurde bei auf 950 MHz übertakteter CPU durchgeführt. Tabelle 11 zeigt die Ergebnisse der Validierung. Im Folgenden wird auf die Ergebnisse der einzelnen Systeme genauer eingegangen.

Beim **Matlab-Hough-System** handelt es sich um das Matlab-System, welches die ROI-Detektion auf Grundlage der Hough-Transformation durchführt. Das gemessene System wurde mit Hilfe der Matlab Codegenerierung in C++ übersetzt und mit einem Wrapper-Programm versehen, damit es außerhalb von Matlab, unter Ubuntu und Raspbian, lauffähig ist. Es fällt eine recht geringe Erkennungsrate im Vergleich zu den anderen beiden Systemen auf. Des Weiteren benötigt das System eine sehr lange Laufzeit, insbesondere auf dem Raspberry Pi. Die schlechte Laufzeit ist einerseits dadurch zu erklären, dass für die Codegenerierung viele Funktionen neu implementiert werden mussten. Handgeschriebene Funktionen in Matlab sind allgemein etwas langsamer als die internen Matlab-Funktionen.

Andererseits sind einige Optimierungen in C++ möglich, aber nicht in Matlab. Vor allem gilt dies für den gesamten Pointer-Bereich.

System, OS, Plattform	**Laufzeit (σ) [s]**	**Erkennungsrate [%]**
Matlab-Hough, Ubuntu, VM	26.269 (0.286)	67,593
Matlab-Edge, Ubuntu, VM	5,384 (1,973)	87.037
OpenCV-Edge, Ubuntu, VM	0,050 (0,003)	98.148
Matlab-Hough, Raspbian, Pi	618,553 (1,135)	67,593
Matlab-Edge, Raspbian, Pi	152.576 (53.326)	87.037
OpenCV-Edge, Raspbian, Pi	1,458 (0,072)	98.148

Tabelle 11: Ergebnisse für Laufzeit und Erkennungsrate der verschiedenen Systeme auf unterschiedlichen Plattformen

Beim **Matlab-Edge-System** handelt es sich um das Matlab-System, welches die ROI-Detektion auf Grundlage der vertikalen Kanten im Bild durchführt. Es zeigt sich, dass das System eine mittlere Erkennungsrate hat. Der Grund hierfür besteht darin, dass die Funktion *std2()* (zur Berechnung der Standardabweichung von Matrizen) nicht von der Matlab-Codegenerierung unterstützt wird. Somit kann die Niback-Formel nicht bzw. nur ohne Einfluss der Standardabweichung umgesetzt werden. Dies führt zu einer schlechteren ROI-Detektion und Segmentierung als beim OpenCV-Edge-System. Mit einer Implementierung von *std2()* sollte das System eine ähnlich gute Erkennungsrate erzielen wie das OpenCV-Edge-System. Die gemessene Laufzeit ist einerseits deutlich länger als die des OpenCV-Edge-Systems, andererseits etwas kürzer als des Matlab-Hough-Systems. Während bereits beim Matlab-Hough-System begründet wurde, weswegen die Matlab-Systeme langsamer sind als das OpenCV-System, lässt sich die Differenz zum Matlab-Hough-System dadurch erklären, dass es sich bei der Hough-Transformation um eine sehr aufwändige Funktion handelt. Insbesondere ist der Rechenaufwand abhängig von der Anzahl der Kanten-Pixel im Bild.

Beim **OpenCV-Edge-System** handelt es sich um das System, welches direkt in C++ mit Hilfe von OpenCV implementiert wurde. Das System weist eine sehr gute Erkennungsrate auf. Des Weiteren lässt sich feststellen, dass das System eine deutlich kürzere Laufzeit aufweist als die mit Hilfe der Matlab-Codegenerierung erzeugten Systeme. Diese kürzere Laufzeit war zu erwarten, weil C++ deutlich bessere Optimierungsmöglichkeiten bietet als Matlab. Des Weiteren wurden viele OpenCV-Funktionen verwendet. Da es sich bei OpenCV um eine Bilderverarbeitungsbibliothek handelt, wurden viele OpenCV-Funktionen bei der Entwicklung der Softwarebibliothek auf die Laufzeit optimiert. Die kürzere Laufzeit kommt insbesondere auf dem Raspberry Pi zum Tragen und erlaubt ggf. auch eine weitere Erhöhung

der Komplexität des Algorithmus zur Steigerung der Erkennungsrate. Durch Anpassung der Matlab-Systeme ließen sich die Laufzeiten dieser Implementierungen ggf. noch verbessern, es ist aber nicht davon auszugehen, dass diese dann im Bereich der OpenCV-Implementierung lägen.

4.4.2. Vergleich der ROI-Detektoren

Die Erkennungsraten der Gesamtsysteme wurden zwar bereits in Abschnitt 4.4.1 miteinander verglichen, allerdings sind diese Erkennungsraten durch den Einfluss der anderen Systemkomponenten verfälscht. Da die ROI-Detektoren zusammen mit dem Gesamtsystem auf dem Raspberry Pi nur schwer miteinander verglichen werden können, werden nur die ROI-Detektoren noch einmal in Matlab direkt miteinander verglichen.

Dazu wurde ein Matlab-Programm geschrieben, welches ein Bild einliest und die Vorverarbeitung auf das Bild anwendet. Anstatt die gefundene ROIs weiterzuverarbeiten, gibt das Programm die ROIs einzeln aus. Danach wurde die Anzahl der richtig detektieren ROIs gezählt und daraus die ROI-Erkennungsrate berechnet. Als Validierungsbilder wurden die ersten 90 Bilder des Validierungsdatensatzes verwendet. Diese wurden von allen 9 Messpunkten (vgl. Abbildung 64) bei natürlicher Beleuchtung aufgenommen.

ROI-Detektor	**5 Zahlen richtig [%]**	**6 Zahlen richtig [%]**
Hough	90	87,7
Edge	100	100

Tabelle 12: Vergleich der ROI-Detektoren

Es sind zwei unterschiedliche ROI-Detektoren miteinander verglichen worden.

- Der Hough-Detektor extrahiert die ROIs mit der in Abschnitt 4.2.4 vorgestellten Methode auf Grundlage der Hough-Transformation. Dieser ROI-Detektor nutzt nur für die Hough-Transformation selbst eine Matlab-Funktion, dagegen werden zum Suchen nach lokalen Maxima und Kanten im Bild eigene, neu implementierte Funktionen verwendet. Der ROI-Detektor unterstützt die Codegenerierung und kann auf dem Raspberry Pi eingesetzt werden. Der Detektor kommt auch bei dem in Abschnitt 4.4.1 ausgemessenen Matlab-Hough-System zum Einsatz.
- Der Edge-Detektor extrahiert die ROIs mit Hilfe der in Abschnitt 4.2.5 vorgestellten Methode auf Grundlage der vertikalen Kanten im Bild. Alle Funktionen dieses ROI-Detektors werden von der Codegenerierung unterstützt. Der Detektor wurde auch in dem in Abschnitt 4.4.1 ausgemessenen System Matlab-Edge eingesetzt.

Anhand der in Tabelle 12 gezeigten Ergebnisse lässt sich festhalten, dass der ROI-Detektor auf Grundlage der Hough-Transformation eine geringere Erkennungsrate aufweist. Der Detektor

auf Grundlage der vertikalen Kanten zeigt auf der Validierungsmenge eine perfekte Erkennung. Von diesem Detektor wurden alle ROIs aus den Bildern extrahiert.

Obwohl die Erkennungsrate bei der ROI-Detektion auf Grundlage der Hough-Transformation deutlich geringer ist, konnte diese die Lage der ROIs sehr genau bestimmen. Auf Abbildung 93 und Abbildung 94 werden die Ergebnisse der ROI-Extraktion beider Detektoren für dasselbe Bild gezeigt. Während die Hough-Detektion sehr genau die Lage des Ziffernfeldes bestimmen konnte, wird bei der Edge-Detektion zwar das Ziffernfeld gefunden, die gefundene Region ist aber deutlich größer als der eigentliche Zählerstand. Je ungenauer der ROI-Detektor das Ziffernfeld erkennt, desto schwieriger wird die anschließende Segmentierung des ROI bzw. die Extraktion der einzelnen Ziffern.

Es lässt sich feststellen, dass die Hough-Detektion Probleme bei schwachen Kanten zeigen, insbesondere wird häufig bei der ROI-Detektion die letzte Ziffer abgeschnitten. Diese wird beim Testzähler von einem roten Viereck umgeben, welches sich nur relativ schwach vom Hintergrund abhebt.

Abbildung 93: ROI-Detektion Hough

Abbildung 94: ROI-Detektion Edge

4.4.3. Einschränkungen

Obwohl der Validierungsdatensatz mit 180 Bildern ausreichend groß gewählt wurde, ist es wichtig, die Rahmenbedingungen für das System genau zu definieren.

Einerseits kann das System aktuell nur bei Zählern eingesetzt werden, deren Ziffernfeld aus weißen Zahlen besteht, da das eingesetzte neuronale Netz nur mit weißen Ziffern trainiert wurde. In der Literatur wurde gezeigt, dass neuronale Netze auch auf unterschiedlichen Ziffernfarben arbeiten können [18]. Bei praktischen Versuchen hat sich jedoch gezeigt, dass die verwendete Anzahl von Testmustern zu gering für die Erkennung von weißen Ziffern auf schwarzem Hintergrund und schwarzen Ziffern auf weißem Hintergrund mit ein und

demselben Klassifikator ist. Dieses Problem lässt sich jedoch sehr leicht beheben, indem der Nutzer dem System die Ziffernfarbe einmal mitteilt und die Zahlen dann ggf. invertiert werden. Das neuronale Netz wurde unter anderem mit solchen invertierten Ziffern trainiert.

Andererseits hat sich bei der Validierung des Systems gezeigt, dass für eine akzeptable Erkennungsrate eine ausreichend gute Beleuchtung des Zählers benötigt wird. Für den praktischen Einsatz muss der Zähler also während der Bildaufnahme ausreichend beleuchtet werden.

Weiterhin wurde der in dieser Arbeit vorgestellte Prototyp nur auf einem Zählermodell validiert. Wie bei jedem Mustererkennungssystem gilt die gemessene Erkennungsrate nur auf den Validierungsdaten. Vor dem Einsatz auf weiteren Zählermodellen anderer Hersteller, müssen zunächst Validierungsbilder von diesen Zählermodellen getestet werden. Zur Anpassung an andere Zählermodelle müssen ggf. einige der gewählten Parameter und Faktoren angepasst werden. Es ist allerdings davon auszugehen, dass es sich bei der Anpassung lediglich um eine Feinjustierung des bestehenden Systems handelt, da keine zählerspezifischen Apriori-Annahmen getroffen wurden. Insbesondere sind keine Änderungen des grundsätzlichen Systems zu erwarten, da es sich bei den, bei der Entwicklung eingesetzten Elementen, wie der ROI-Detektion auf Basis vertikaler Kanten und der Ziffernerkennung mittels neuronaler Netze, um Elemente handelt, die in vielen ähnlichen Bereichen (KZF-Zeichenerkennung, Schrifterkennung) ebenfalls sehr gute Ergebnisse erzielt haben.

4.5. Zusammenfassung des Kapitels

In diesem Kapitel wurde ein Smart-Metering-System vorgestellt, welches es ermöglicht, alte Stromzähler mit analoger Anzeige zu modernen Smart-Metern zu erweitern. Das vorgestellte System liest den Zählerstand kontinuierlich aus und stellt die gemessenen Daten im Netzwerk als Web Service zur Verfügung.

Das System wurde in verschiedenen Ausführungen implementiert, die sich in Erkennungsrate und Laufzeit für eine Zählerstanderkennung unterscheiden. Das beste System erreicht dabei eine gute Erkennungsrate von 98% bei einer sehr kurzen Laufzeit von 1,7 Sekunden. Als Plattform kam das Raspberry Pi Model B als kostengünstiges eingebettetes System zum Einsatz.

Um die Erkennungsrate weiter zu steigern, wäre es möglich, mehrere Bilder vom Stromzähler auszuwerten und über die Erkennungsergebnisse den Mittelwert bzw. Median für jede Ziffer zu bilden. Selbst bei einer Mittelung über zehn Bilder, wäre es trotzdem noch möglich, den Verbrauch über der Zeit mit einer zeitlichen Auflösung von 17 Sekunden darzustellen.

Aufgrund der eingesetzten Algorithmen zur Detektion einzelner Bereiche, in denen sich der Zählerstand befindet, lässt sich das System auch derart erweitern, dass mit einem Erkennungssystem gleichzeitig mehrere Zähler ausgelesen werden könnten.

Ein weiterer Ansatz zur ROI-Detektion könnte sich mit der zeitlichen Mittelung mehrerer Bilder des Stromzählers befassen. Grundlage dabei ist die Annahme, dass zeitliche Änderungen nur im Bereich des Zählerstandes zu finden sind, während der Rest des aufgenommen Bildes statisch ist. Methoden zur Vordergrund-Detektion auf Grundlage zeitlicher Filter werden in der Videoverarbeitung eingesetzt und erzielen dort sehr gute Ergebnisse [30].

Für einen Einsatz auf anderen Zählermodellen wäre eine Erweiterung der Validierungsmenge um Bilder von allen gängigen Modellen möglich. Außerdem kann das System auch auf Zählermodellen mit digitaler Anzeige angepasst werden, denn auch viele moderne Zählermodelle mit digitaler Anzeige haben kein Kommunikationsmodul zur Übertragung des Zählerstandes.

5. Dezentrale Gerätekonfiguration

Geräte sollen in einem Smart Home miteinander kommunizieren, wodurch sich weitere Funktionalitäten ergeben. Die Kopplung von Geräten muss nutzerfreundlich und ohne das Fachwissen eines Technikers möglich sein. Der Endnutzer soll in der Lage sein auf einfache Art und Weise mit DPWS-Geräten unterschiedlicher Hersteller eigene Smart Home-Funktionalitäten einzurichten. Um dieses Ziel zu realisieren, wurde in diesem Kapitel ein Konzept zur dezentralen Gerätekonfiguration entwickelt. Außerdem beschreibt dieses Kapitel die Realisierung des Konzeptes durch die in DPWS standardisierten Funktionalitäten. Es wird kein neues Geräteprofil zur Gerätekonfiguration benötigt.

5.1. Motivation

In einem Anwendungsfall für Smart Home sollen zwei Geräte, die je einen Web Service anbieten, miteinander verbunden werden. Die beiden Geräte kommunizieren über eine Verbindung, die vom Benutzer initiiert wird. Ein Nutzer soll die Möglichkeit haben Regeln für die Steuerung von Geräten zu definieren. Dies soll an einem Beispiel erläutert werden: Das Ziel ist eine Klimaanlage in Abhängigkeit von der Innentemperatur zu steuern. Dazu existiert ein Temperatursensor, der Messdaten bereitstellt. Die Klimaanlage wird vom Nutzer mithilfe einer Regel angewiesen, einen Temperatursollwert zu halten. Beim Überschreiten der Temperatur schaltet sich die Klimaanlage ein, beim Unterschreiten der Temperatur aus. Um die Geräte miteinander zu verbinden, bietet sich eine Methode an, bei der Tasten genutzt werden, um Geräte zu koppeln. Zuerst wird der Knopf an der Klimaanlage gedrückt. Daraufhin muss der Nutzer innerhalb einer definierten Zeit von beispielsweise einer Minute einen Knopf auf dem gewünschten Temperatursensor betätigen. Über diesen Mechanismus kann eine direkte Kommunikationsverbindung beider Geräte hergestellt werden. Ein Nachteil ist, dass jedes Gerät über einen Knopf verfügen muss. Deshalb ist diese Methode nicht für alle Gerätearten geeignet. Außerdem ist es nicht möglich auf einfache Art und Weise Regeln für die Ausführung von Funktionen zu definieren. Andere kommerzielle Smart Home Lösungen setzen auf zentrale Hubs, die die Geräte miteinander verknüpfen. Ein Hub ist dabei ein wichtiger zentraler Netzwerkknoten, der einen Single Point of Failure darstellt. Fällt dieser Knoten aus, so sind keine Smart Home Funktionalitäten mehr verfügbar, was zu einer schlechten Nutzerakzeptanz führt. Außerdem finden sich bei diesen Lösungen proprietäre Protokolle, die eine Erweiterung durch Geräte anderer Hersteller erschweren. Die Konfiguration der Hubs erfolgt über eine Smartphone App. Im entwickelten Konzept soll eine Smartphone App zur Konfiguration zweier Geräte genutzt werden. Jedes Smartphone ist in der Lage als Vermittler zwischen zwei Geräten zu agieren, da die offene Schnittstelle DPWS verwendet wird. Auf Basis von Gerätebeschreibungen, die jedes Gerät über sich bereitstellt, können Geräte miteinander gekoppelt werden, die in der Lage sind miteinander zu interagieren.

5.2. Konzept

Um zwei beliebige DPWS-Geräte miteinander zu verbinden, dient ein Smartphone als Vermittler. Dadurch wird kein zentraler Knoten zur Geräteverwaltung benötigt. Das Smartphone konfiguriert die Geräte über einen Konfigurations-Service, sodass daraufhin die Kommunikation zwischen den eingerichteten Geräten stattfindet. Es ist nur möglich zwei Geräte aneinander zu koppeln, die denselben Port Type unterstützen. Eine Klimaanlage würde beispielsweise einen Temperaturwert als Port Type verarbeiten können. Geräte, die diesen Port Type bereitstellen, wie ein Temperatursensor, lassen sich an die Klimaanlage koppeln. Während der Konfiguration kann nicht nur eine Verbindung zwischen zwei Geräten initiiert werden, sondern es lassen sich auch Regeln für Smart Home-Funktionen definieren. Abbildung 95 beschreibt die Konfiguration einer Klimaanlage und eines Temperatursensors durch ein Smartphone. Nachdem die Geräte miteinander gekoppelt wurden, ruft die Klimaanlage zyklisch die Temperaturwerte des Sensors ab, um nach einer durch den Nutzer definierten Regel zu arbeiten. Sobald die Temperatur im Innenraum zu hoch ist (Sollwert wird durch den Nutzer festgelegt), schaltet sich die Klimaanlage ein.

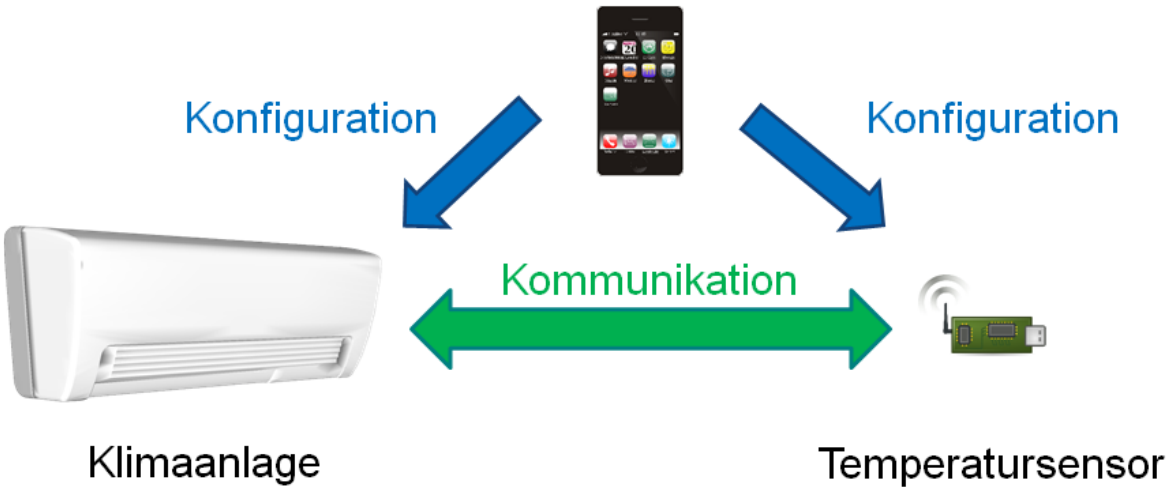


Abbildung 95: Konzept der Konfiguration am Beispiel einer Klimaanlage und eines Temperatursensors.

Um diesen Mechanismus zu realisieren, wurde ein Konfigurationsmechanismus auf Basis eines Konfigurations-Services entwickelt. Dieser soll im nachfolgenden Abschnitt näher erläutert werden.

5.3. Realisierung

Das Smartphone als Vermittler zwischen den zu verbindenden Geräten führt lediglich einen DPWS-Client aus, um mit den DPWS-Geräten zu interagieren. Mindestens eines der Geräte muss einen „DPWS Config Service“ bereitstellen, welcher folgende Operationen beinhaltet:

Operation	**Bedeutung**
GetSupportedInterfaces	Unterstützte Interfaces ermitteln
ConnectService	Service anbinden
GetConnectedServices	Angebundene Services abfragen
DisconnectService	Verbindung trennen
GetRuleParameters	Regelparameter abfragen
GetRules	Bereits definierte Regeln abfragen
SetRule	Regeln einstellen
DeleteRule	Regel löschen

Tabelle 13: Operationen des Konfigurations-Services

Die Operation „GetSupportedInterfaces“ dient der Smartphone App, um die von einem Gerät unterstützten Interfaces zu ermitteln. Dabei handelt es sich um den Port Type eines Daten-Services eines anderen Gerätes. Durch die „ConnectService“-Operation kann ein Daten-Service an ein Gerät mit einem Konfigurations-Service angebunden werden. Dabei erhält die „ConnectService“-Operation eine Referenz zu dem entfernten Gerät. Über die „GetConnectedServices“-Operation lassen sich die verbundenen Geräte abrufen. Durch einen Aufruf der „DisconnectService“-Operation können Geräte wieder voneinander getrennt werden. Sollen Regeln für das Zusammenspiel von zwei DPWS-Geräten erstellt werden, so lassen sich die erwarteten Regelparameter über die „GetRuleParameter“-Operation abrufen. Alle bereits erstellten Regeln sind über „GetRules“ abrufbar. Soll eine neue Regel dem Gerät hinzugefügt werden, kann dies über die „SetRule“-Operation realisiert werden. Bestehende Regeln lassen sich durch „DeleteRule“ aufheben.

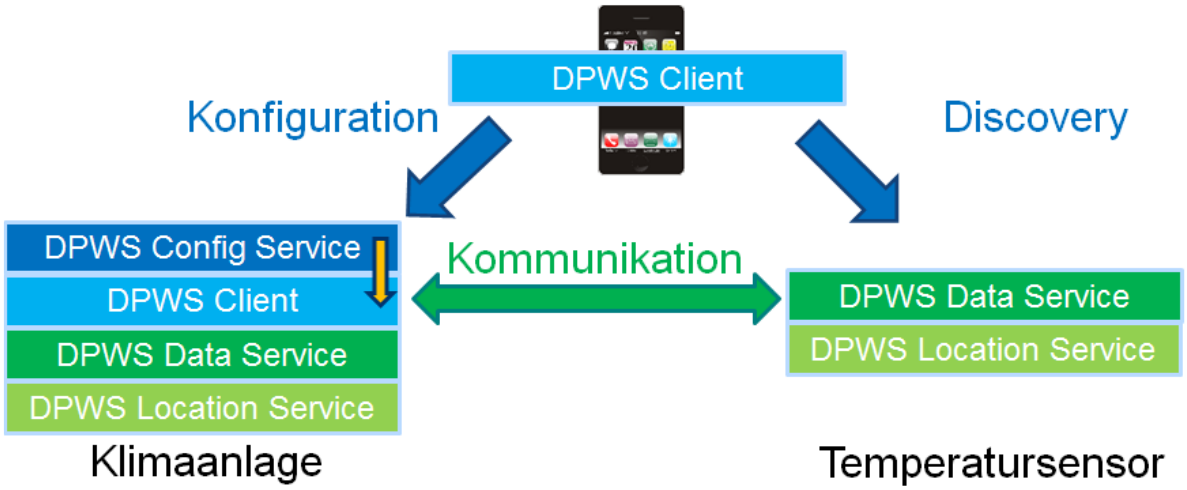


Abbildung 96: Realisierung der Gerätekonfiguration

Die beschriebene Realisierung soll am Beispiel der Vernetzung einer Klimaanlage mit einem Temperatursensor erläutert werden: Dabei stellt die Klimaanlage den Konfigurations-Service „DPWS Config Service“ zur Verfügung (Abbildung 96). Nachdem der DPWS-Client, welcher auf einem Smartphone ausgeführt wird, die Klimaanlage gefunden hat, können über die „GetSupportedInterfaces“-Operation die erwarteten Schnittstellen der Klimaanlage abgerufen werden. Dabei handelt es sich um das Interface eines Temperatursensors, welcher die aktuelle Zimmertemperatur bereitstellt. Mit dieser Information kann die Smartphone-App alle verfügbaren DPWS-Geräte visualisieren, die dieses Interface unterstützen. Da es mehrere Geräte mit demselben Interface geben kann (mehrere Temperatursensoren), wird ein Location Service definiert, der Informationen über den Standort des Gerätes bereitstellt. Die Standortinformation ist nicht in der WSDL eines Gerätes enthalten. Nach Auswahl des gewünschten Sensors wird über die „ConnectService“-Operation der ausgewählte Temperatursensor mit der Klimaanlage gekoppelt (Abbildung 97). Daraufhin wird von der Klimaanlage ein DPWS-Client gestartet, der die Temperaturwerte vom Sensor abruft. Anschließend kann der Nutzer mithilfe der Smartphone-App Regeln erstellen. Es kann ein Sollwert für die Innentemperatur vorgegeben werden. Sobald dieser überschritten wird, startet die Klimaanlage. Das Starten der Kühlung wird ebenfalls durch einen Webservice, welcher von der Klimaanlage angeboten wird, realisiert. Der lokal ausgeführte DPWS-Client steuert dabei die Klimaanlage. Alternativ lassen sich auch komplexere Anwendungen erstellen. So kann zusätzlich ein Außentemperatursensor an die Klimaanlage gekoppelt werden. Sobald die Innentemperatur höher als der Sollwert und die Außentemperatur kleiner als der Sollwert ist, kann anstelle der Klimaanlage ein Fenster automatisch geöffnet werden, um die Innentemperatur zu senken (Abbildung 98, siehe Abschnitt 7.2.2). Damit ist eine energieeffiziente Temperatursteuerung möglich.

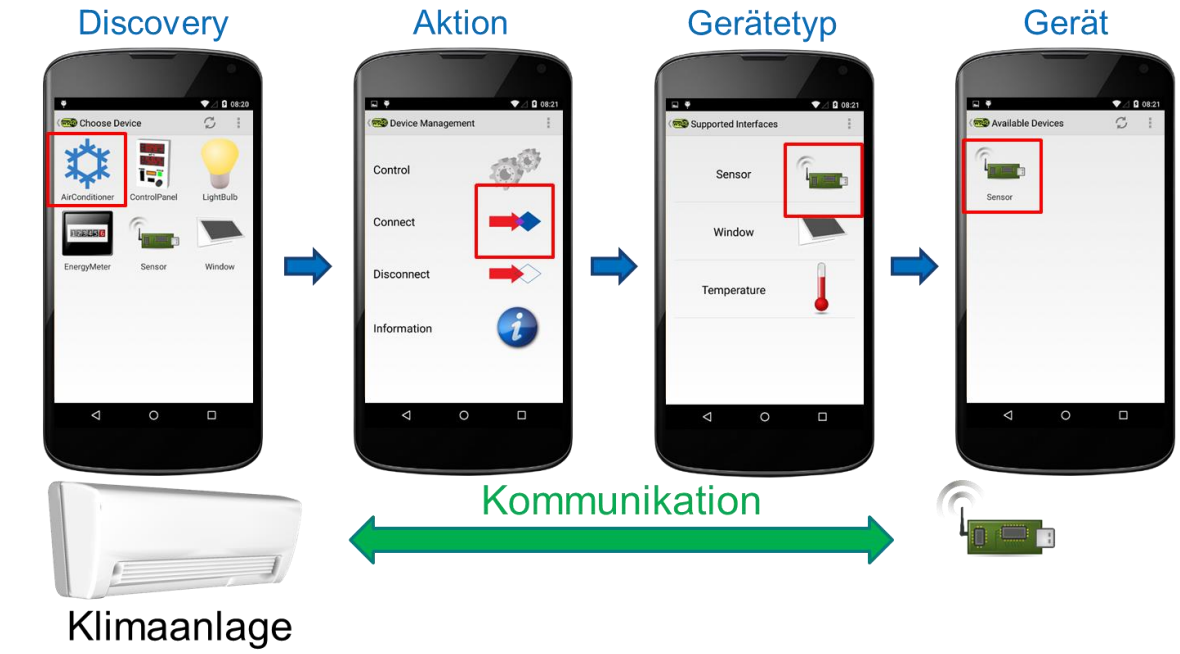


Abbildung 97: Ablauf der Konfiguration von Klimaanlage und Temperatursensor

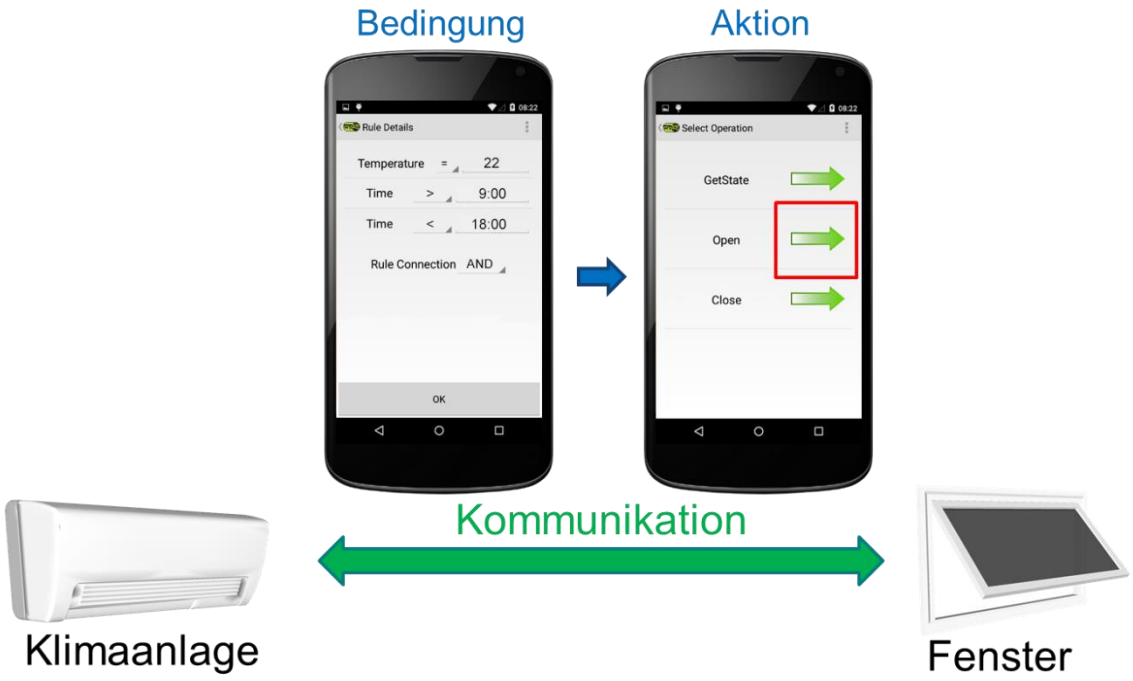


Abbildung 98: Definition von Bedingung und Aktion einer Regel

5.4. Implementierung

Das Konzept zur dezentralen Gerätekonfiguration definiert einen Konfigurations-Service und beschreibt eine Smartphone Applikation. Nachfolgend wird der Konfigurations-Service, welcher durch ein Interface definiert ist, erläutert. Weiterhin beinhaltet dieser Abschnitt eine Beschreibung der umgesetzten Android-App.

5.4.1. ConfigurationServiceInterface

Um den Konfigurations-Service allgemeingültig zu beschreiben, bietet die Programmiersprache Java die Möglichkeit Interfaces zu nutzen. Ein Interface definiert welche Attribute und Methoden eine Klasse besitzen muss. Die Methoden, die in dem ConfigurationServiceInterface (Anhang B) definiert sind, finden sich in Tabelle 13. Des Weiteren werden komplexe Datentypen zur abstrakten Beschreibung eingeführt: Dazu zählt das DPWSInterface, welches die Schnittstelle zu einem anderen Gerät beschreibt. Diese Schnittstelle ist durch einen Namen, einen Namensraum, einer Rolle und die Information ob Regeln definiert werden können beschrieben. Weiterhin findet sich ein Typ zur Beschreibung eines Endpunktes (Endpoint). Dieser ist durch zwei URIs und zwei DPWSInterfaces gekennzeichnet. Ein DPWSService wiederum ist durch einen Endpoint, eine Rolle und eine Identifikationsnummer definiert. Die Rolle wird durch einen Typ beschrieben, der die Zustände „Service“ oder „Client“ annehmen kann. Diese Zustände beschreiben, in welcher Art und Weise der Service agiert. Außerdem schreibt das ConfigurationServiceInterface die Struktur einer Regel vor. Eine Regel kann aus mehreren Parametern, die logisch miteinander verknüpft sind, bestehen. Weiterhin wird mit der Regel eine Aktion definiert, die ausgeführt wird, sobald die Regel zutrifft. Die Methode, welche die von einem Gerät unterstützten Interfaces liefert, ist in dem ConfigurationServiceInterface enthalten. Die zurückgegebene Liste bestehend aus den unterstützten Interfaces ist nicht mit den Interfaces zur Klassenbeschreibung zu verwechseln. Die unterstützten Interfaces beschreiben die Schnittstellen (Port Type) zu anderen Geräten.

5.4.2. DPWS Configurator

Der DPWS Configurator kann DPWS-Geräte im lokalen Netzwerk finden und steuern. Dabei kann insbesondere auf den Konfigurations-Service zugegriffen werden. Alle gefundenen Geräte lassen sich innerhalb einer Activity (Benutzeroberfläche und Bestandteil einer Android App) anzeigen. Nach Auswahl eines Gerätes öffnet sich eine Activity um das Gerät entweder zu steuern, zu verbinden, eine bestehende Verbindung zu trennen oder Geräteinformationen abzurufen. Die Gerätesteuerung greift auf die herkömmlichen Services eines Gerätes zu. Die Optionen „Connect“ (dt.: verbinden) und „Disconnect“ (dt.: Verbinden trennen) interagieren mit dem Konfigurations-Service. Nach Auswahl der Option „Connect“ werden die von einem Gerät unterstützten Kommunikationsinterfaces abgerufen und visualisiert.

Nach Auswahl des gewünschten Interface werden Geräte angezeigt, die dieses Interface anbieten. Nach Wählen des gewünschten Gerätes wurde eine Verbindung hergestellt. Für jede Verbindung können mehrere Regeln definiert werden. Der DPWS Configurator stellt für diesen Zweck die grafische Oberfläche zur Verfügung. Eine detaillierte Veranschaulichung des DPWS Configurators findet sich in Kapitel 7.2 „Demoszenarien“.

5.5. Zusammenfassung des Kapitels

Während des Projektes wurde ein Konzept entwickelt, das eine dezentrale und nutzerfreundliche Gerätekonfiguration ermöglicht. Bei dem Konzept muss das DPWS-Profil nicht verändert werden. Es genügt das Offerieren eines Konfigurations-Service, um verschiedene Smart Home Geräte miteinander zu koppeln. Für jede Verbindung lassen sich Regeln erstellen. Dabei können alle Parameter und die gewünschten Aktionen vom Nutzer frei definiert werden. Die Steuerung der Gerätekonfiguration erfolgt in dezentraler Form durch ein beliebiges Smartphone mit der Konfigurations-App. Der Prototyp einer solchen Applikation wurde durch die Android-App DPWS Configurator realisiert.

6. WSDL-Datenbank

Die WSDL-Datenbank dient zum herstellerübergreifenden Austausch von Gerätebeschreibungen, den WSDLs. Die WSDL-Dateien können mit einer Kurzbeschreibung versehen und über eine Webseite in einer Datenbank veröffentlicht werden. Nutzer dieser Datenbank können gezielt nach Geräten suchen, um eigene Geräte unter Berücksichtigung der WSDLs zu entwickeln. Das Problem des fehlenden Datenmodells in DPWS lässt sich somit vereinfachen. Dieses Problem soll im nachfolgenden Abschnitt erläutert werden. Anschließend wird der Aufbau der WSDL-Datenbank erläutert. Diesbezüglich wird die Nutzung der Webseite beschrieben.

6.1. Motivation

Es ist möglich, dass zwei unabhängig arbeitende Unternehmen ein Produkt entwickeln, das dieselben Funktionalitäten aufweist. Als Beispiel soll eine Lampe dienen, welche sich über das Netzwerk DPWS-basiert steuern lässt. Der Nutzer soll in der Lage sein die Lampe ein- und auszuschalten sowie das Licht zu dimmen. Die Funktionalität soll über einen Webservice von der Lampe im Netzwerk angeboten werden. Dabei ergeben sich unterschiedliche Möglichkeiten das Gerät mit der definierten Funktionalität zu implementieren. Die jeweilige Servicebeschreibung mittels WSDL weicht von der anderen ab (Abbildung 99).

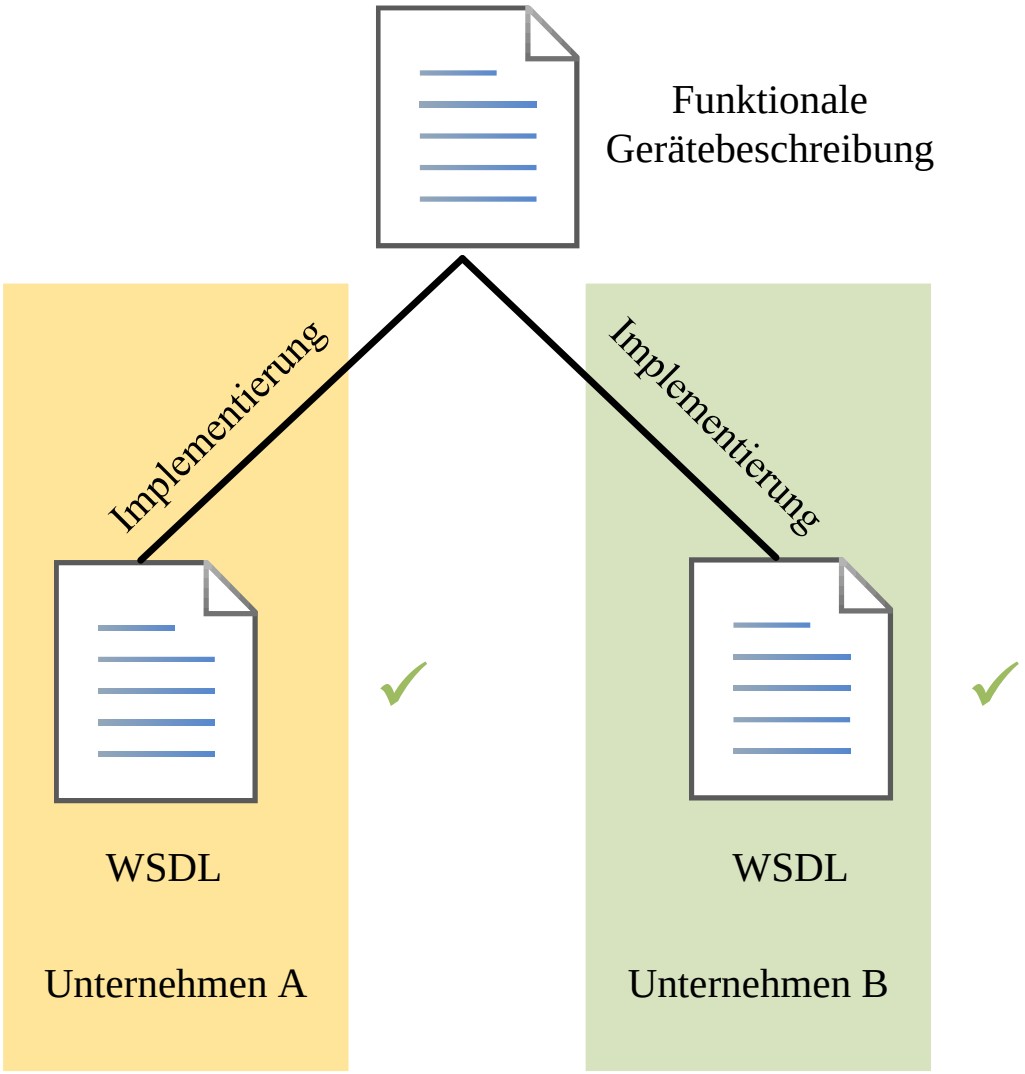


Abbildung 99: Verschiedene Implementierungen desselben Gerätes

Die Anforderung des Ein- und Ausschaltens der Lampe lässt sich durch eine Operation erzielen, die als Eingabewert die Zeichenkette „An“ bzw. „Aus“ beinhaltet. Eine weitere Operation stellt die Dimm-Funktionalität bereit. Die Helligkeit kann durch eine prozentuale Angabe vom Datentyp Integer eingestellt werden. Alternativ können beide Funktionalitäten auch durch eine einzelne Operation realisiert werden. Dabei dient diese zum setzten der Helligkeit. Das Einschalten der Lampe mit voller Helligkeitsstufe erfolgt durch das Senden des Wertes „100%“ an die Lampe. Umgekehrt besteht die Möglichkeit durch Setzen der Helligkeit auf „0%“ die Lampe auszuschalten. Als Datentyp kann eine Fließkommazahl im Bereich Null bis Eins genutzt werden. Dieses Beispiel verdeutlicht die Problematik des fehlenden Datenmodells. Ein weiterer Aspekt ist die fehlende Semantik aus Maschinen-Sicht. Die Bedeutung der WSDL und der damit verbundenen Art und Weise wie mit einem Gerät zu interagieren ist, um ein gewünschtes Verhalten zu erzielen, ist für einen Computer nicht erkennbar.

Bei geeigneter Operationsbeschreibung wäre ein Mensch in der Lage die Operationen eines Gerätes richtig aufzurufen.

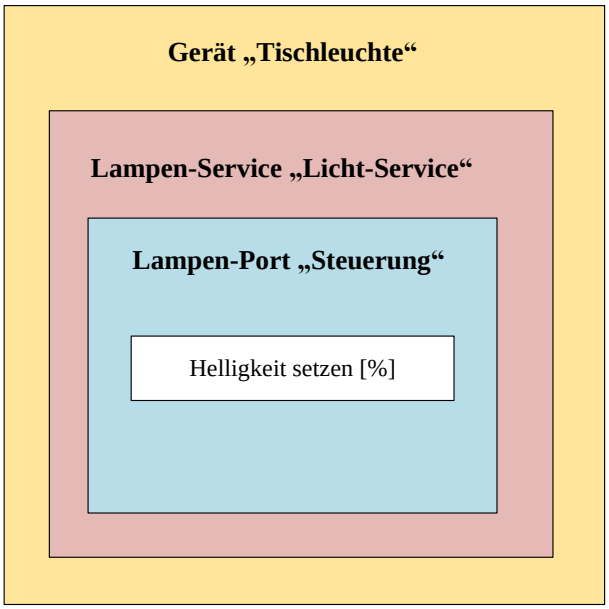


Abbildung 100: Gerätebeschreibung Lampe A

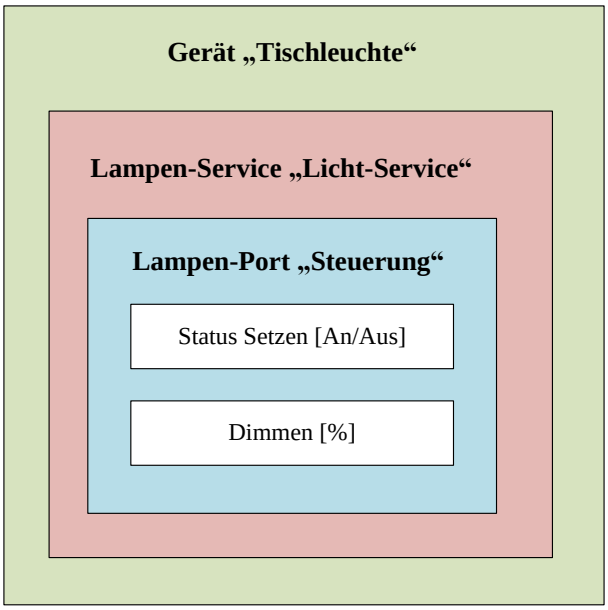


Abbildung 101: Gerätebeschreibung Lampe B

Möchte ein Drittanbieter eine Gerätefernsteuerung mittels eines DPWS-Clients bauen, so muss ihm der Port Typ mit den jeweiligen Operationen zur Entwurfszeit bekannt sein, um mit dem fernzusteuernden Gerät zu interagieren. Um diesem Problem zu begegnen, wurde im Rahmen dieses Projektes eine öffentliche WSDL-Datenbank entwickelt.

6.2. Funktionen der WSDL-Datenbank

Die WSDL-Datenbank (http://ws4d.org/wsdl/, Abbildung 102) ist über ein Webinterface, welches das Suchen, Herunterladen und Veröffentlichen von Gerätebeschreibungen ermöglicht, erreichbar. In den nachfolgenden Abschnitten werden diese Funktionen näher erläutert:

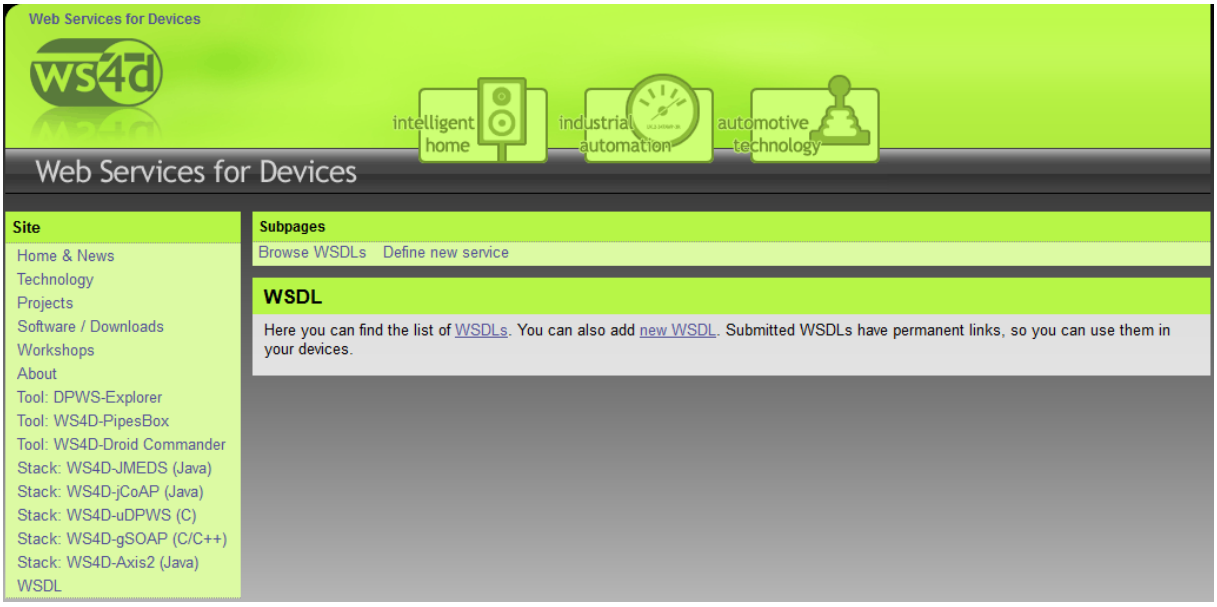


Abbildung 102: Startseite WSDL-Datenbank

6.2.1. Suchen und Herunterladen von WSDLs

Abbildung 103: Herunterladen der WSDL

Die WSDL-Datenbank, welche unter http://ws4d.e-technik.uni-rostock.de/wsdl/browse-wsdls/ verfügbar ist (Abbildung 103), beinhaltet DPWS Servicebeschreibungen in Form von WSDLs. Der Port Type dient als Interface zum Steuern von Geräten. Basierend auf dem Port Type ist es möglich entsprechende Clients für den Web Service zu implementieren.

Neben der Bezeichnung für den Port Type finden sie die Versionsnummer, den Autor, Stichworte, eine kurze Beschreibung, das Upload-Datum sowie einen Link zum Download der WSDL.

6.2.2. Veröffentlichen von WSDLs

Nutzer sind in der Lage über die Webseite http://ws4d.e-technik.uni-rostock.de/wsdl/define-new-service/ eigene WSDL-Dateien zu veröffentlichen (Abbildung 104). Die Webseite beinhaltet ein Formular über das die Gerätebeschreibung hochgeladen werden kann. Der Nutzer muss dazu den Port Type seines Gerätes, die Versionsnummer, seinen Namen und E-Mail-Adresse und Schlagworte eintragen, die die Suche nach dem Gerät innerhalb der Datenbank erleichtern. Optional besteht die Möglichkeit eine Kurzbeschreibung hinzuzufügen. Nachdem der Dateipfad in das Hochladeformular eingetragen wurde, können die Informationen abgesendet werden. Anschließend befindet sich ein neuer, öffentlicher Eintrag in der WSDL-Datenbank. Basierend darauf können andere Entwickler diese WSDL-Dateien verwenden, um eigene Geräte zu entwickeln. Dabei kann es sich um Client-Anwendungen handeln, die andere Geräte fernsteuern oder um eigene Services, die eine bereits veröffentlichte Gerätebeschreibung nutzen.

Abbildung 104: Veröffentlichen einer Gerätebeschreibung (WSDL)

7. Demonstrator

Alle in diesem Projekt entwickelten Mechanismen sollen in Form eines Demonstrators getestet und veranschaulicht werden. Dazu wurden die Remotekonfiguration von DPWS-Geräten, die Einbindung von Legacy-Technologien und die kamera-basierte Zählerstanderfassung prototypisch umgesetzt. Die Mechanismen sollen anhand von Demoszenarien vorgeführt werden.

7.1. Umsetzung

Im Demonstrationsaufbau (Abbildung 105) befinden sich verschiedene Smart Home-Geräte, die in einem WLAN miteinander verbunden sind. Dabei stellt ein WLAN-Router den Access Point für das Netzwerk dar. Zu den Smart Home Geräten gehören ein Temperatur- und ein Helligkeitssensor (als „Sensor“ zusammengefasst), eine Klimaanlage, eine Lampe, ein Fenster und ein Smart Meter, deren Ansteuerung durch DPWS realisiert ist. Des Weiteren findet sich ein BACnet Gerät, das über ein BACnet-DPWS-Gateway in das Netzwerk eingebunden ist. In der Demonstration handelt es sich um einen weiteren allgemeinen Datenpunkt, dessen Service über BACnet bereitgestellt wird und über das Gateway via DPWS erreichbar ist. Die Kommunikation zwischen den Haushaltsgeräten ist in Abbildung 105 grün gekennzeichnet. Alle Geräte lassen sich durch den DPWS-Configurator, welcher auf einem Android Smartphone ausgeführt wird, ansteuern und verwalten. Die Konfiguration der Endgeräte ist in Abbildung 105 blau dargestellt. Weiterhin lassen sich die Geräte neben dem DPWS-Configurator mit einem Knopf an jedem Gerät miteinander koppeln. Dieser Kopplungsmechanismus wurde im Rahmen des BBSR Projektes „Webservices for Devices als Integrationsplattform für intelligente Dienste der Gebäudetechnik“ entwickelt.

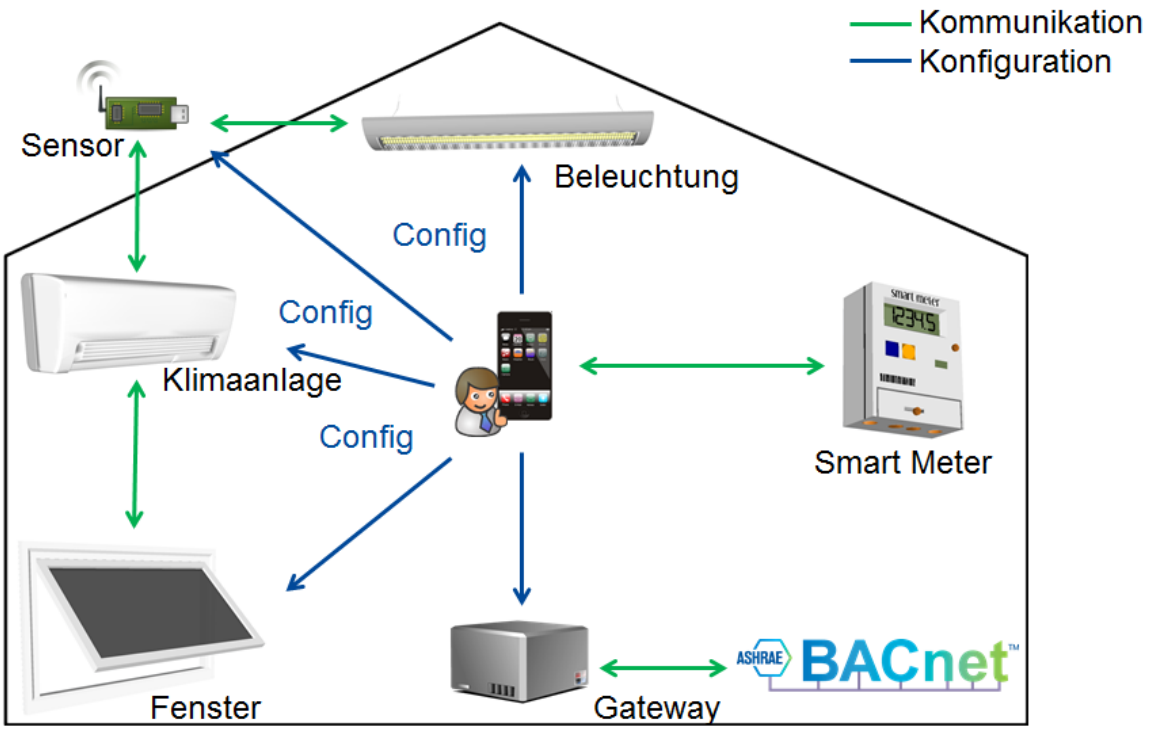


Abbildung 105: Smart Home Demonstrationsgeräte

Alle Geräte werden mit dem Einplatinencomputer Raspberry Pi Version 2 (Abbildung 106, Tabelle 14) realisiert. Auf diesen Computern wird als Betriebssystem Raspbian (Linux) ausgeführt, welches auf Debian basiert. Um das Board drahtlos mit dem Versuchsnetzwerk zu verbinden, wurde ein WLAN-Adapter via USB an den Raspberry Pi angeschlossen. Nach dem Bootvorgang verbinden sich alle Geräte automatisch mit dem Versuchsnetzwerk. Die Netzwerkkonfiguration der Endgeräte erfolgt durch einen auf dem WLAN-Zugangspunkt ausgeführten DHCP-Server.

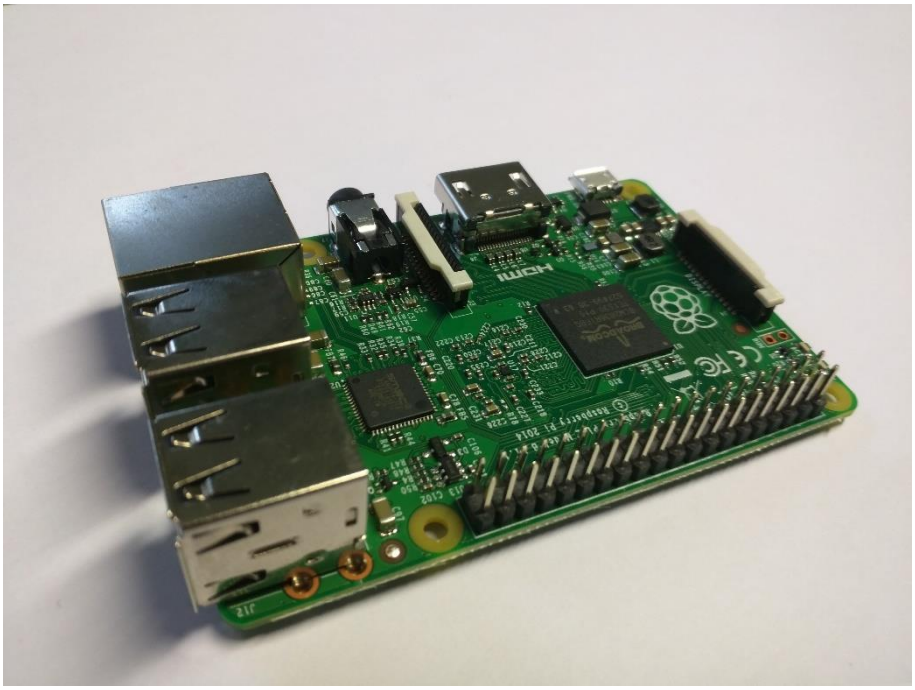


Abbildung 106: Raspberry Pi 2, Model B

Größe	85,60 mm × 53,98 mm × 17 mm
SoC	Broadcom BCM2836
CPU	ARM Cortex-A7 (900 MHz)
GPU	Broadcom VideoCore IV
RAM	1024 MB
Videoausgabe	FBAS, HDMI
Netzwerk	10/100-MBit-Ethernet-Controller
Schnittstellen	USB, 26 GPIO-Pins, SPI, I²C, UART
ROM	SD (SDHC und SDXC)
Leistungsaufnahme	max. 4W (800mA)
Stromversorgung	5 V

Tabelle 14: Spezifikationen Raspberry Pi 2, Model B

Um Demogeräte wie die Glühlampe und die Klimaanlage zu realisieren, wird je ein Raspberry Pi durch eine Schaltung erweitert. Diese Schaltung ermöglicht das Steuern von 230V Netzgeräten durch Relais.

7.2. Demoszenarien

Zu den Demoszenarien gehören die Remotekonfiguration von Helligkeitssensor und Lampe. Des Weiteren sollen drei Geräte, ein Temperatursensor, eine Klimaanlage und ein Fenster, aneinandergekoppelt werden. Außerdem wird die kamera-basierte Stromzählererfassung demonstriert. Weiterhin findet sich eine Demonstration zur Einbindung eines BACnet-Gerätes in ein DPWS-basiertes Netzwerk.

7.2.1. Remotekonfiguration Helligkeitssensor-Lampe

In diesem Demoszenario soll eine Lampe mit einem Helligkeitssensor gekoppelt werden (Abbildung 107). Dabei bietet der Helligkeitssensor die Messwerte über DPWS an. Die Lampe lässt sich sowohl direkt über einen Web Service steuern als auch über einen weiteren Web Service konfigurieren. Als Vermittler zwischen beiden Geräten fungiert ein Android Smartphone, das den „DPWS Configurator“ als App ausführt.

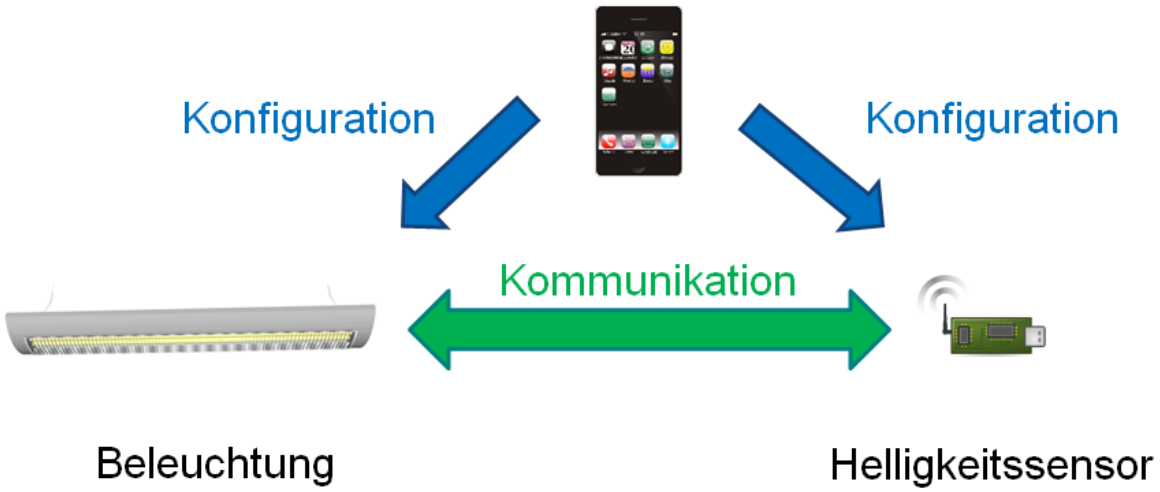


Abbildung 107: Konfiguration von Beleuchtung und Helligkeitssensor

Durch den Konfigurations-Service lassen sich Helligkeitssensor und Beleuchtung miteinander koppeln und Regeln definieren. Die nachfolgenden Abbildungen zeigen den Ablauf beim Kopplungsvorgang zwischen dem Helligkeitssensor und der Beleuchtung.

Nach dem Start des DPWS Configurators, öffnet sich der Startbildschirm (Abbildung 108) über den die Suche nach DPWS-Geräten per WS-Discovery ausgeführt wird. Die gefundenen Geräte werden anschließend in der Android Activity „Choose Device“ mit Symbolen aufgelistet (Abbildung 109). Neben dem DPWS-Temperatursensor und der DPWS-Lampe ist ein Computer im lokalen Netzwerk gefunden worden, da dieser unter Windows 7 einen UPnP-Dienst ausführt und somit vom DPWS Configurator entdeckt wird.

Abbildung 108: Startseite

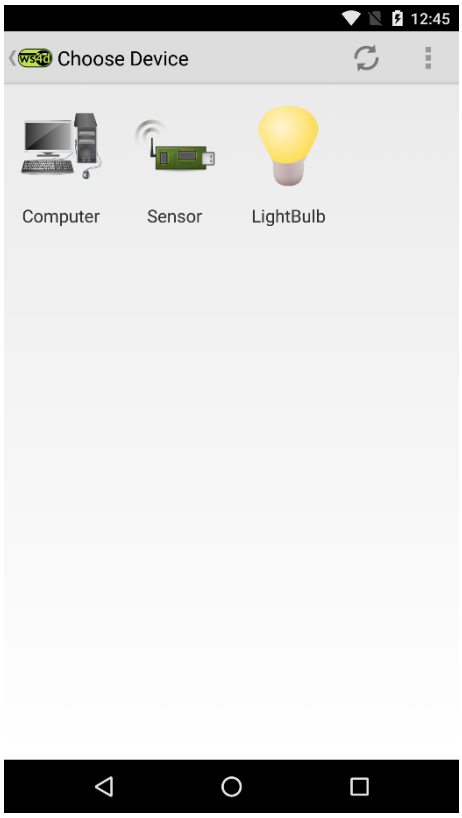


Abbildung 109: Geräteauswahl

Der Nutzer kann zur Verbindung von Sensor und Lampe eines der beiden Geräte auswählen. Daraufhin öffnet sich die „Device Management“-Activity (Abbildung 110) über die ein Gerät gesteuert, mit einem anderen Gerät verbunden und von einem anderen Gerät getrennt werden kann. Außerdem lassen sich über diese Activity Informationen über das Gerät abrufen. Um die Lampe mit einem anderen Gerät zu verbinden, wählt der Nutzer die Option „Connect“. Daraufhin werden die unterstützten Interfaces in der Activity „Supported Interfaces“ (Abbildung 111) aufgelistet. Die Lampe unterstützt das Interface „Brightness“ (dt.: Helligkeit) über das die Helligkeitswerte eines Sensors empfangen werden können.

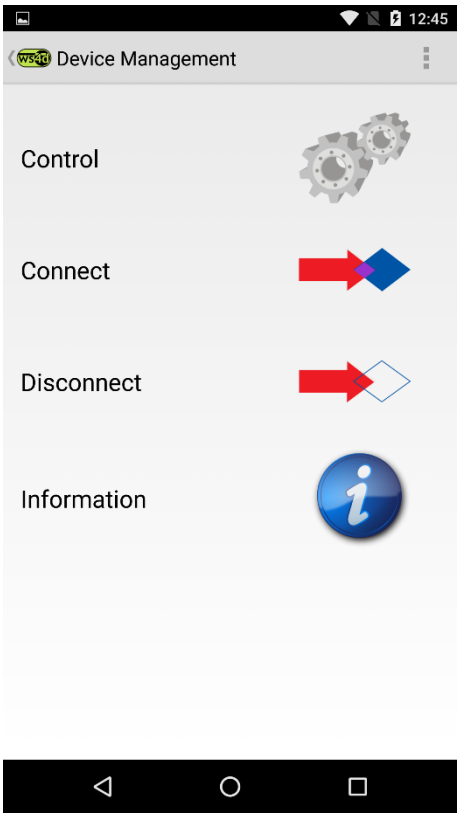


Abbildung 110: Geräteverwaltung

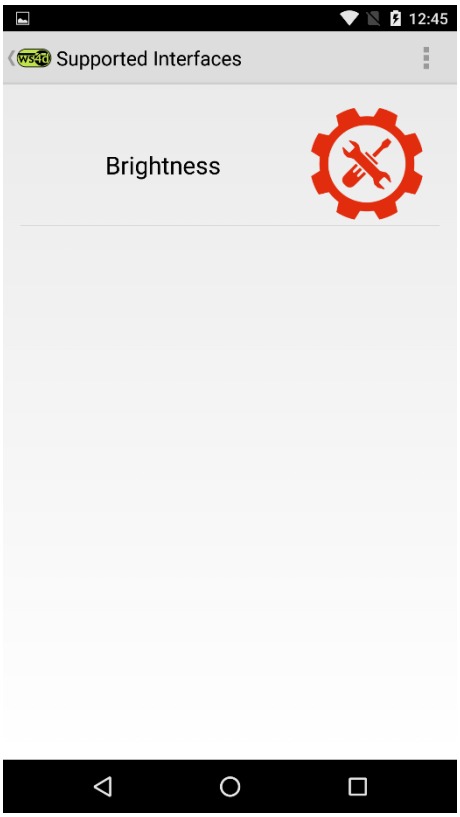


Abbildung 111: Unterstützte Interfaces

Anschließend wird im lokalen Netzwerk nach Geräten gesucht, die dieses Interface unterstützen. In der Activity namens „Available Devices“ (Abbildung 112) werden alle verfügbaren Geräte symbolisch dargestellt. Nach Auswahl des gewünschten Sensors wurde eine Verbindung zwischen der Lampe und dem Sensor hergestellt. Alternativ kann auch zuerst der Sensor ausgewählt werden, um anschließend nach Geräten zu suchen, die einen Helligkeitswert über das Brightness-Interface entgegennehmen. Dabei wird die Lampe unter den verfügbaren Geräten aufgelistet.

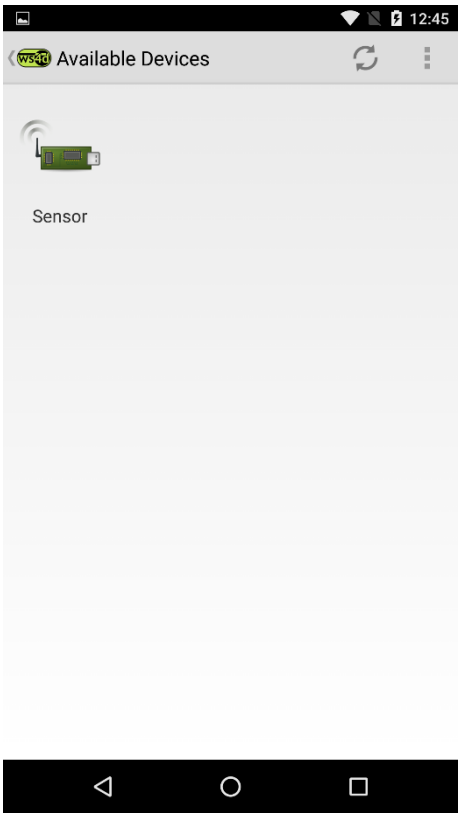


Abbildung 112: Verfügbare Geräte

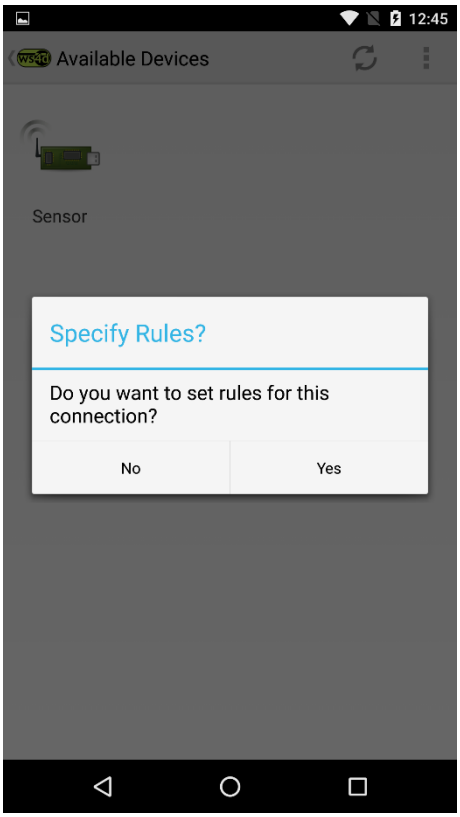


Abbildung 113: Regel festlegen

Nachdem die Verbindung hergestellt wurde, steht es dem Nutzer frei Regeln für diese Verbindung zu erstellen (Abbildung 113). Dabei können pro Verbindung mehrere Regeln definiert werden (Abbildung 114). In diesem Beispiel wird der Helligkeitswert mit der Einheit Lux mit einem vom Nutzer festgelegten Wert von 80 Lux verglichen (Abbildung 115). Diese Teilbedingung kann mit anderen Teilbedingungen logisch verknüpft werden. Somit sind z. B. zeitabhängige Regeln möglich. In dem gewählten Beispiel löst die Regel aus, wenn die Helligkeit den Wert von 80 Lux innerhalb eines Zeitintervalls unterschreitet.

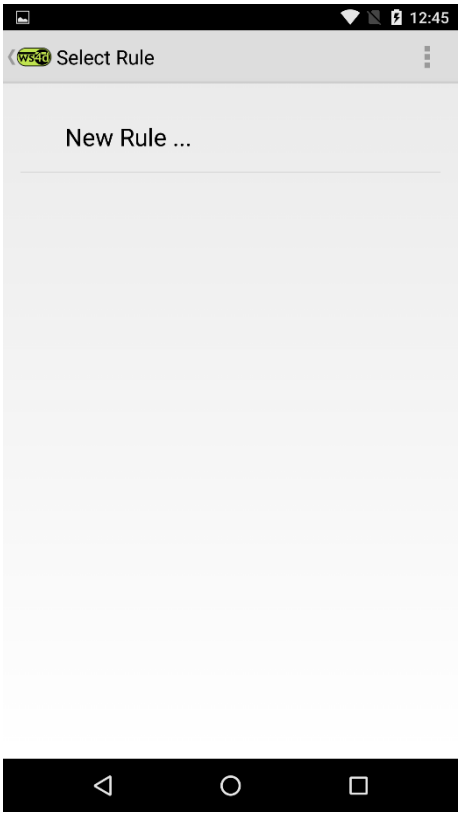


Abbildung 114: Regel auswählen

Abbildung 115: Regel Parameter

Wenn eine Regel zutrifft, dann soll eine Aktion ausgeführt werden. Dazu wählt der Nutzer in der „Select Operation“ Activity (Abbildung 116) die gewünschte Funktionalität aus. In diesem Fall, bei Unterschreiten der Helligkeit von 80 Lux abends und nachts, soll die Lampe eingeschaltet werden. Nach Auswahl der entsprechenden Operation „On“ kann der Nutzer Parameter, in der „Operation Details“ Activity (Abbildung 117), an die Operation übergeben. Da diese Operation keine Parameter benötigt um die Lampe einzuschalten kann die Regel mit OK bestätigt werden. Abschließend kann ein Name für die Regel frei vergeben werden, um die Regel zu beschreiben (Abbildung 118). Nach Abschluss dieses Vorganges wird die Regel über den Configuration Service an die Lampe übertragen und ausgeführt. Dabei ruft ein DPWS-Client, welcher von der Lampe ausgeführt wird, zyklisch den aktuellen Helligkeitswert vom DPWS-Helligkeitssensor ab. Die Periodendauer dieser Abfrage lässt sich über das Konzept des Configuration Service an das Gerät übertragen. Alternativ sind Szenarien denkbar in denen im unterstützten Interface eine Eventing Operation definiert ist, die das Endgerät abonniert, um asynchron über Änderungen, von z. B. Messwerten, benachrichtigt zu werden.

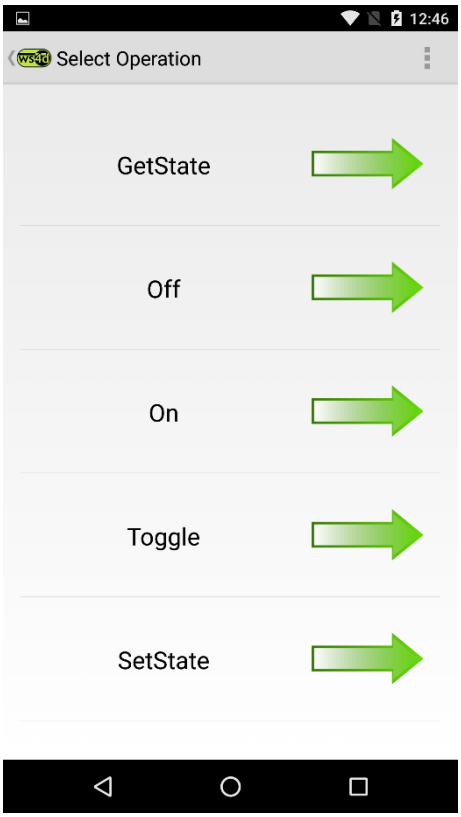


Abbildung 116: Aktion auswählen

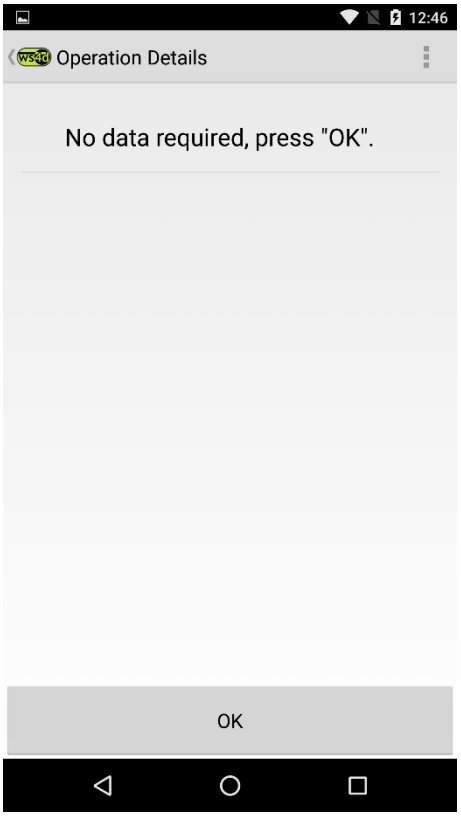


Abbildung 117: Aktions Parameter

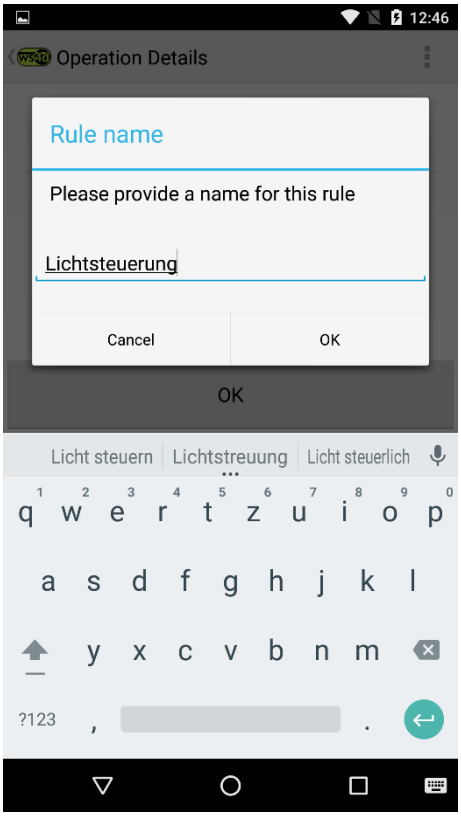


Abbildung 118: Regel benennen

7.2.2. Remotekonfiguration Temperatursensor-Klimaanlage-Fenster

Es ist ebenso möglich drei DPWS-Geräte miteinander zu koppeln. So kann ein Temperatursensor den aktuellen Außentemperaturwert liefern. In Abhängigkeit dieses Temperaturwertes und der gewünschten Innentemperatur kann die Kühlung auf energieeffiziente Weise realisiert werden. Ist die Außentemperatur kleiner als die gewünschte Innentemperatur, so öffnet sich das Fenster, ohne dass die Klimaanlage aktiv kühlen muss. Die gesamte Gerätekonfiguration erfolgt ebenfalls dezentral durch eine Smartphone Anwendung.

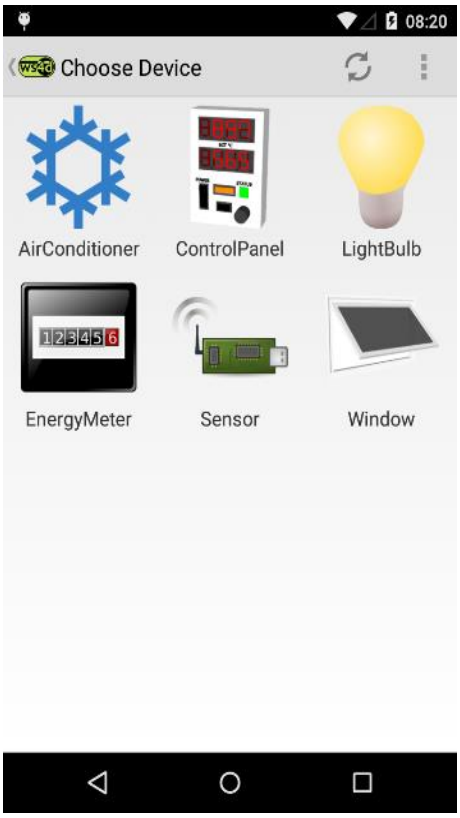


Abbildung 119: Geräteauswahl

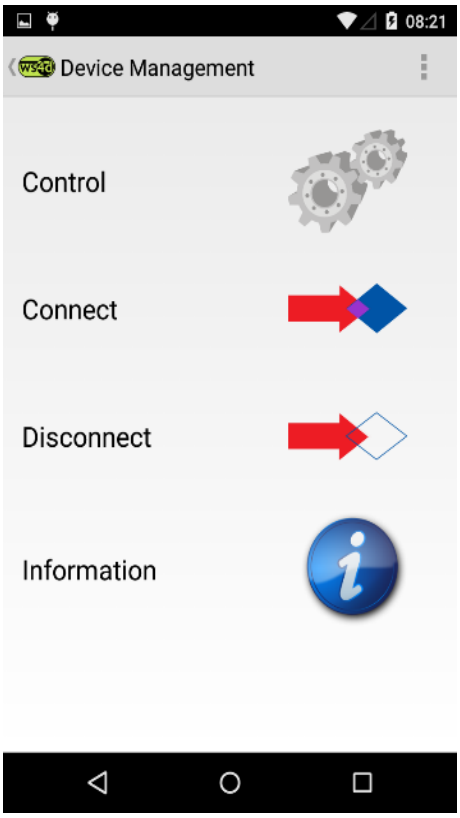


Abbildung 120: Geräteverwaltung

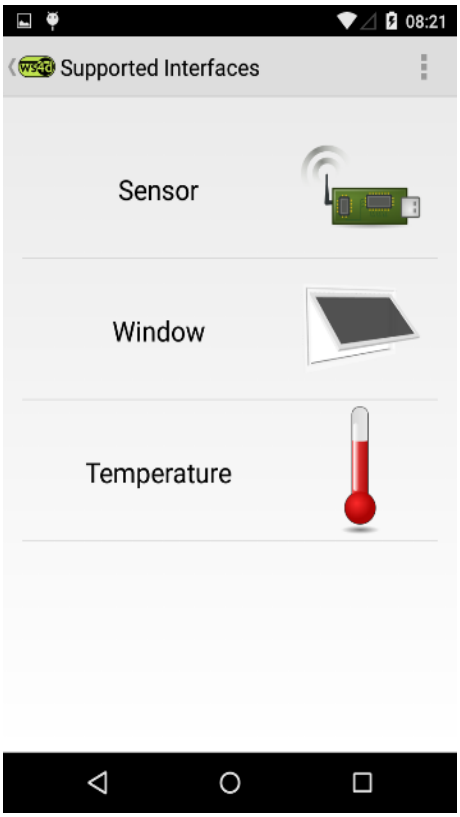


Abbildung 121: Unterstützte Interface

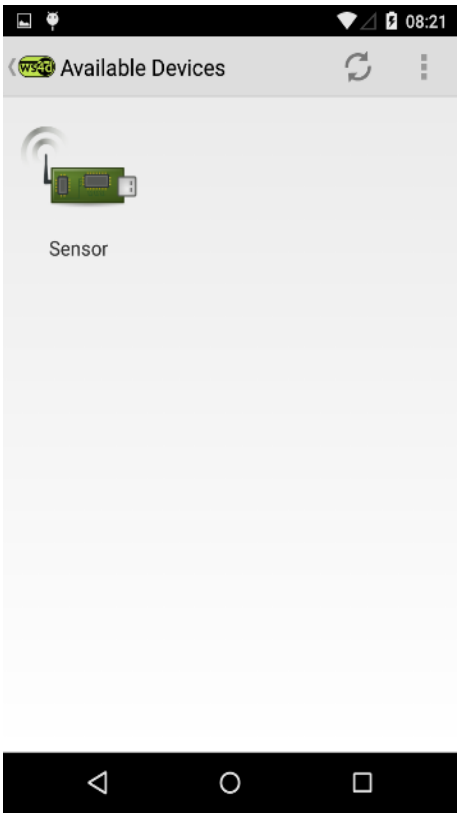


Abbildung 122: Verfügbare Geräte

Abbildung 123: Regel definieren

Abbildung 124: Aktion auswählen

In Abbildung 119 werden alle Smart Home Geräte angezeigt, woraufhin der Nutzer die Klimaanlage auswählt. Das Gerät kann mit anderen Geräten durch Wahl der Option „Connect“ (dt.: verbinden) verbunden werden (Abbildung 120). Anschließend werden die unterstützten Interfaces angezeigt (Abbildung 121). Da die Klimaanlage mit einem Temperatursensor gekoppelt werden soll, wird das Temperature (deutsch: Temperatur) – Interface ausgewählt (Abbildung 121). Daraufhin zeigt der DPWS Configurator alle verfügbaren Temperatursensoren im lokalen Netzwerk (Abbildung 122). Der gewünschte Temperatursensor kann nun mit der Klimaanlage verbunden werden. Anschließend besteht die Möglichkeit Regeln für die Klimaanlage zu definieren (Abbildung 123). So lässt sich das Fenster öffnen (Abbildung 124), wenn die Außentemperatur kleiner ist als der Sollwert der Innentemperatur. Somit muss die Klimaanlage den Raum nicht aktiv kühlen, was eine Reduktion des Energiebedarfs bewirkt.

7.2.3. Kamera-basierte Stromzählererfassung

Bei der kamera-basierten Stromzählererfassung kommt die prototypische Implementierung der Bildverarbeitung aus Kapitel 4 zum Einsatz. Die Kamera zur Bilderfassung und der Rechner zur Bildverarbeitung sind in einem Gehäuse integriert (Abbildung 125). Dabei wird die Bildverarbeitung als Software auf dem „Raspberry Pi 2 Model B“ ausgeführt. Der ermittelte Zählstand kann durch einen Web Service abgerufen werden. Als Client für diesen Webservice wird der DPWS Configurator genutzt. Dieser zeigt den aktuellen Wert des Stromzählers an. Die Demo zeigt, wie ein handelsüblicher Stromzähler (Abbildung 126) durch eine einfach zu installierende Kamera samt Bildverarbeitung zu einem Smart Meter umgebaut werden kann. Kostenintensive Neuinstallationen des gesamten Zählers entfallen durch die Installation einer Kamera.

Abbildung 125: Kamera zur Zählererfassung

Abbildung 126: Stromzähler

7.2.4. Einbindung von Legacy-Technologien

Bei diesem Demoszenario soll ein BACnet Gerät durch einen DPWS-Client angesteuert werden. Das BACnet Gerät stellt ein exemplarisches Gerät dar, das Daten unterschiedlicher Typen bereitstellt. Dazu zählen analoge, binäre und multistate Daten. Diese Datentypen können auch als Eingangsparameter für das BACnet Gerät dienen. Die Demo zeigt diese grundlegenden Datentypen, da mit ihnen verschiedenste Anwendungen realisiert werden können. Das DPWS-BACnet-Gateway wird als OSGi-Anwendung auf einem Desktop-PC ausgeführt. Im Praxiseinsatz lässt sich die Software des Gateways auf einem eigebetteten System ausführen.

8. Zusammenfassung

Ziele dieses Forschungsprojektes sind die benutzerfreundliche Installation und Verwendung von Smart Home-Geräten durch semantische Plug&Play-Technologien. Web Services, welche eine domänenübergreifende Internettechnologie darstellen, bieten sich als Möglichkeit an diese Ziele zu erreichen. Die Untersuchungen haben gezeigt, dass es ebenso möglich ist, bereits installierte Legacy-Technologien weiterhin zu nutzen. Die Legacy-Technologie BACnet wurde in diesen Untersuchungen exemplarisch in ein DPWS-basiertes System von Smart Home-Geräten integriert. Dazu wurde ein BACnet-DPWS-Gateway implementiert, um zwischen den beiden Technologien zu übersetzen. Außerdem lassen sich mithilfe der erarbeiteten Konzepte andere Technologien durch das Gateway einbinden.

Es konnte gezeigt werden, dass sich herkömmliche, bereits installierte Stromzähler, zu einem Smart Meter umrüsten lassen. Dabei wird der Stromzähler durch eine Kamera abgelesen, die Bilddaten werden verarbeitet und der Zählerstand kann auf Basis von DPWS für andere Geräte zur Verfügung gestellt werden. Die Kosten für die Hardware können als sehr gering abgeschätzt werden. Die Installation beschränkt sich im Wesentlichen auf die Befestigung der Kamera an dem Stromzähler.

Durch das BACnet-DPWS-Gateway und die kamera-basierte Stromzählererfassung können nicht-DPWS Geräte eingebunden werden. Außerdem wurde die Konfiguration von DPWS Geräten betrachtet. Dabei können DPWS-basierte Smart Home-Geräte dezentral und nutzerfreundlich durch eine Smartphone Anwendung miteinander verbunden werden, um Mehrwertanwendungen zu realisieren.

Da DPWS kein Datenmodell besitzt, können Probleme bei der herstellerübergreifenden Gerätevernetzung auftreten. Um diesem Problem zu begegnen, bietet sich eine zentrale, öffentliche Datenbank am, um Gerätebeschreibungen (WSDLs) auszutauschen. Dazu wurde eine Webseite samt Datenbank zum Publizieren von WSDLs implementiert.

Alle in diesem Forschungsprojekt erarbeiteten Konzepte wurden prototypisch implementiert. Ihre Funktionalität konnte anhand eines Demonstrators bewiesen werden. DPWS als Internettechnologie kann innerhalb von Smart Home zur Plug&Play-Gerätevernetzung eingesetzt werden. Verschiedene Hersteller können Geräte entwickeln die über die offene Schnittstelle DPWS kommunizieren. Somit lassen sich Smart Home-Anwendungen herstellerübergreifend realisieren.

A. Anhang

A.1. Gateway-eigene Enumerations

A.1.1. GWgroup

GWgroup	BACnetEngineeringUnits
ACCELERATION	metersPerSecondPerSecond
AREA	squareCentimeters, squareFeet, squareInches, squareMeters
CAPACITANCE	farads
CONDUCTANCE	siemens
CURRENT	amperes, milliamperes
ENERGY	joules, kilojoules, megajoules
FORCE	newton
FREQUENCY	cyclesPerHour, cyclesPerMinute, hertz, kilohertz, megahertz, perHour, perMinute, perSecond
HUMIDITY	gramsOfWaterPerKilogramDryAir, percentRelativeHumidity
ILLUMINATION	luxes
INDUCTANCE	henrys
LENGTH	meters, millimeters, centimeters, inches, feet
LUMINOUS_FLUX	lumens
LUMINOUS_INTENSITY	candelas
MASS	Kilograms, poundsMass, tons
POWER	Watts, milliwatts, kilowatts, megawatts, horsepower
PRESSURE	pascals, hectopascals, kilopascals, bars, millibars
RESISTANCE	ohms, milliohms, kilohms, megohms
TEMPERATURE	degreesCelsius, degreesKelvin, degreesFahrenheit
TIME	Years, months, weeks, days, hours, minutes, seconds, hundredthsSeconds, milliseconds
VELOCITY	millimetersPerSecond, millimetersPerMinute, metersPerSecond, metersPerMinute, metersPerHour, kilometersPerHour, feetPerSecond, feetPerMinute, milesPerHour
VOLTAGE	volts, millivolts, kilovolts, megavolts
VOLUME	cubicFeet, cubicMeters, imperialGallons, liters, usGallons
BINARY	-
MULTISTATE	-

Tabelle 15: Die Enumeration GWgroup und zugeordnete BACnet-Einheiten

A.1.2. GWdirection

GWdirectio	BACnet-Objekttyp
INPUT	Analog-Input, Binary-Input, MultiState-Input
OUTPUT	Analog-Output, Binary-Output, MultiState-Output
VALUE	Analog-Value, Binary-Value, MultiState-Value

Tabelle 16: Die Enumeration GWdirection und zugeordnete BACnet-Objekttypen

A.1.3. GWparmeter

GWparameter	BACnet-Objekteigenschaft	DPWS-Elementname
INFO	Description	-
INFO_ACTIVE	ActiveText	-
INFO_INACTIVE	InactiveText	-
NAME	ObjectName	-
NOTIFICATION_DELAY	TimeDelay	-
NOTIFICATION_INCREMENT	COVIncrement	-
NOTIFICATION_LIMIT_MAX	HighLimit	-
NOTIFICATION_LIMIT_MIN	LowLimit	-
NOTIFICATION_LIMITS_ON	LimitEnable	-
PRESENT_VALUE	PresentValue	value
PRESVAL_UPDATEINTERVAL	UpdateInterval	-
PRESVAL_LIMIT_MAX	MaxPresValue	maxvalue
PRESVAL_LIMIT_MIN	MinPresValue	minvalue
PRESVAL_STATES_NUM	NumberOfStates	-
PRESVAL_STATES_INFO	StateText	-
PRESVAL_POLARITY	Polarity	-
PRESVAL_RESOLUTION	Resolution	-
PRESVAL_UNIT	Units	unit
STATUS	StatusFlags	status

Tabelle 17: Die Enumeration GWparameter und zugeordnete BACnet-Objekteigenschaften und DPWS-Elementnamen

A.1.4. GWdatatype

GWdatatype	BACnet-Datentyp	XML-Schema-Datentypen
Boolean	Boolean, BinaryPV	Boolean
Byte	OctetString	Byte, Unsigned Byte,
CharacterString	CharacterString	String, Token
Double	Double	Double
Float	Real	Float
Integer	Signed	Int, Integer, NonPositiveInteger, NegativeInteger

Unit	EngineeringUnits	-
Unsigned Integer	Unsigned	UnsignedInt, NonNegativeInteger, PositiveInteger

Tabelle 18: Die Enumeration GWdatatype und zugeordnete BACnet-Datentypen und XML-Schema-Datnetypen

GWdatatype	BACnet-Objekttyp
CharacterString (und default)	Characterstring Value
Byte	OctetString Value
Boolean	Binary Value
Double	Large Analog Value
Float	Analog Value
Integer	Integer Value
Unit	(wird der Objekteigenschaft Unit zugeordnet)
Unsigned Integer	Positive Integer Value

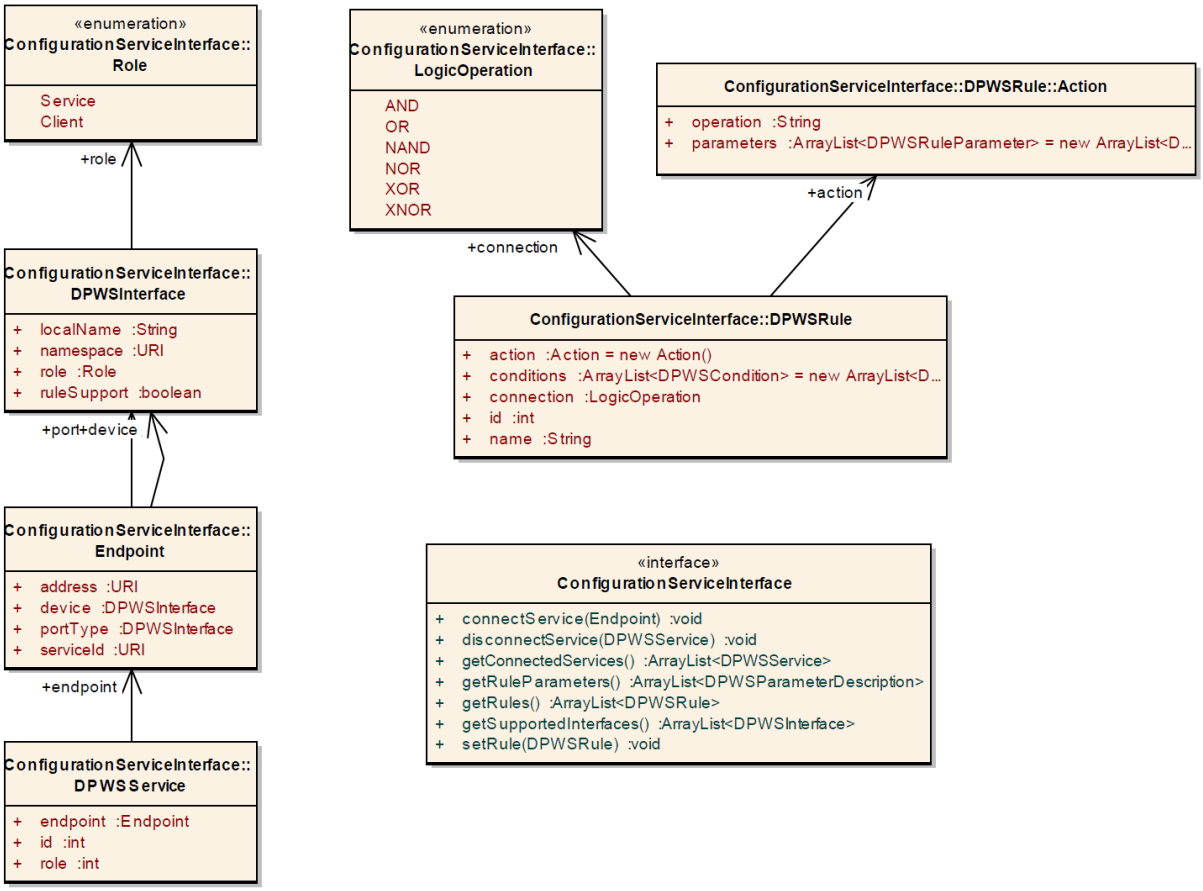
Tabelle 19: Objekttypen zu den Elementen der Enumeration GWdatatype

A.1.5. GWunit

GWunit	BACnetEngeneeringUnit	DPWS-Einheit
Percent	percent	%
Celsius	degreesCelsius	C
Fahrenheit	degreesFahrenheit	F
Kelvin	degreesKelvin	K

Tabelle 20: Die Enumeration GWunit und zugeordnete BACnet-Einheiten und Bezeichnung in DPWS

A.2. Konfigurations-Service Interface



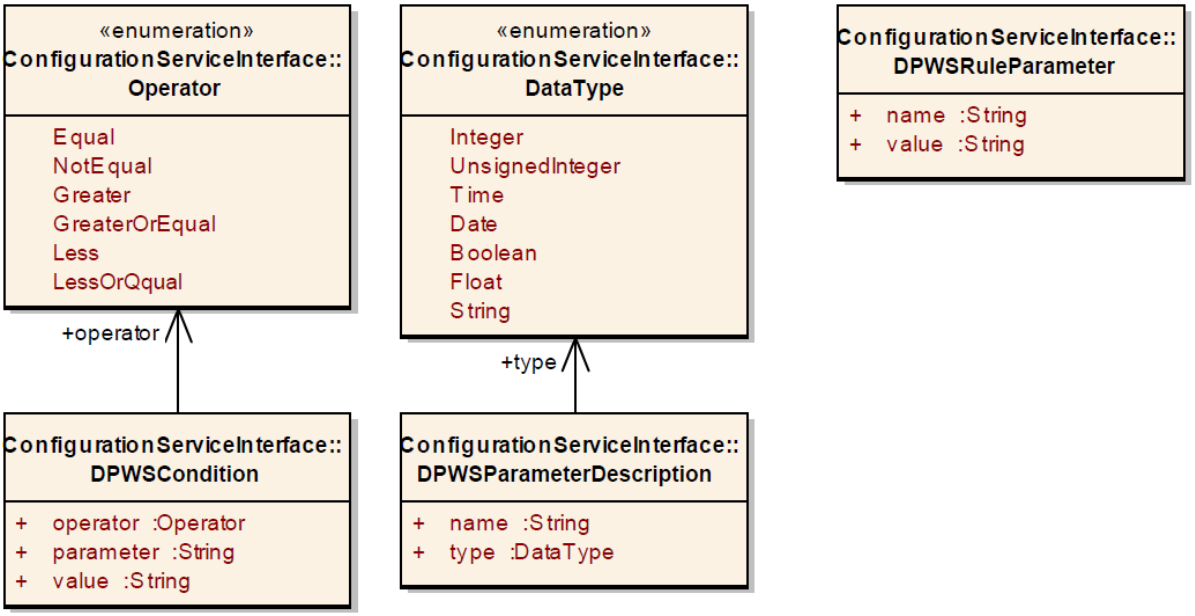


Abbildung 127: Interface des Konfigurations-Service

Literaturverzeichnis

[1] L. L. Peterson und B. S. Davie, Computer Networks: a systems approach, Morgan Kaufmann Publishers, 2012.

[2] B. A. Forouzan, Data Communications and Networking, McGraw-Hill, 2007.

[3] R. Fielding, „Architectural Styles and the Design of Network-based Software Architectures," University of California, 2000.

[4] L. Richardson und S. Ruby, RESTful Web Services, O'Reilly Media, 2007.

[5] T. Berners-Lee, R. Fielding und L. Masinter, Uniform Resource Identifier (URI): Generic Syntax, Internet Standard RFC 3986, 2005.

[6] T. Bayer und D. M. Sohn, REST Web Services, yeebase media GmbH, 2007.

[7] A. S. Tanenbaum und D. J. Wetherall, Computer Networks, Prentice Hall, 2011.

[8] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach und T. Berners-Lee, Hypertext Transfer Protocol - HTTP/1.1, Internet Standard RFC 2616, 1999.

[9] L. Richardson, M. Amundsen und S. Ruby, RESTful Web APIs, O'Reilly, 2013.

[10] Z. Shelby, K. Hartke und C. Bormann, „The Constrained Application Protocol (CoAP)," Internet Standard RFC 7252, 2014.

[11] Z. Shelby, Constrained RESTful Environments (CoRE) Link Format, Internet Standard RFC 6690, 2012.

[12] F. G. Guido Moritz, „IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) and Constrained Application Protocol (CoAP)," in *The Industrial Communication Technology Handbook 2nd Edition*, CRC Press, 2014, pp. 38.1-38.13.

[13] T. Erl, Service Oriented Architecture: Concepts, Technology, and Design, Pearson

Education Inc., 2005.

[14] H. Bohn und F. Golatowski, „Web Services for Embedded Devices," in *Embedded Systems Handbook, Second Edition,* CRC Press, 2009, pp. 19.1-19.31.

[15] T. Bellwood, S. Capell, L. Clement, J. Colgrave, M. J. Dovey und andere, „UDDI Version 3.0.2," *UDDI Spec Technical Committee Draft,* 2004.

[16] I. Melzer, Service-Orientierte Architekturen mit Web Services, Spektrum Akademischer Verlag GmbH, 2010.

[17] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris und D. Orchard, „Web Services Architecture," *W3C Working Group Note,* 2004.

[18] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler und F. Yergeau, „Extensible Markup Language (XML) 1.0 (Fifth Edition)," *W3C Recommendation,* 2008.

[19] H. Vonhoegen, Einstieg in XML, Galileo Press, 2011.

[20] D. C. Fallside und P. Walmsley, „XML Schema Part 0: Primer Second Edition," W3C Recommendation, 2004.

[21] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H. F. Nielsen und andere, „SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)," *W3C Recommendation,* 2007.

[22] N. Mitra und Y. Lafon, „SOAP Version 1.2 Part 0: Primer (Second Edition)," *W3C Recommendation,* 2007.

[23] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau und H. F. Nielsen, „SOAP Version 1.2 Part 2: Adjuncts (Second Edition)," *W3C Recommendation,* 2007.

[24] R. Jeyaraman, „SOAP-over-UDP Version 1.1," *Public Review Draft,* 2009.

[25] J. Dong, Network Protocols Handbook, Javvin Press, 2007.

[26] J. Schlimmer, A Technical Introduction to the Devices Profile for Web Services, MSDN,

2004.

[27] S. Chan, D. Conti, C. Kaler, T. Kuehnel, A. Regnier und andere, Devices Profile for Web Services, Microsoft Corporation, 2006.

[28] T. Nixon, A. Regnier, D. Driscoll und A. Mensch, Devices Profile for Web Services Version 1.1, OASIS, 2009.

[29] G. Moritz, E. Zeeb, S. Prüter, F. Golatowski, D. Timmermann und R. Stoll, „Device Profile for Web Services and the REST," *The 8th IEEE International Conference on Industrial Informatics (INDIN),* pp. 584-591 , 2010.

[30] P. Leach, M. Mealling und R. Salz, „A Universally Unique IDentifier (UUID) URN Namespace," Internet Standard RFC 4122, 2005.

[31] M. Gudgin, M. Hadley und T. Rogers, „Web Services Addressing 1.0 - Core," *W3C Recommendation,* 2006.

[32] A. S. Vedamuthu, D. Orchard, F. Hirsch, M. Hondo, P. Yendluri und andere, „Web Services Policy 1.5 - Framework," *W3C Recommendation,* 2007.

[33] A. S. Vedamuthu, D. Orchard, F. Hirsch, M. Hondo, P. Yendluri und andere, „Web Services Policy 1.5 - Attachment," *W3C Recommendation,* 2007.

[34] A. Nadalin, C. Kaler, R. Monzillo und P. Hallam-Baker, „Web Services Security: SOAP Message Security 1.1 (WS-Security)," *OASIS Standard Specification,* 2006.

[35] K. Ballinger, B. Bissett, D. Box, F. Curbera, D. Ferguson und andere, „Web Services Metadata Exchange 1.1 (WS-MetadataExchange)," *W3C Member Submission,* 2008.

[36] J. Alexander, D. Box, L. F. Cabrera, D. Chappell und G. D. andere, „Web Services Transfer (WS-Transfer)," *W3C Member Submission,* 2006.

[37] E. Christensen, F. Curbera, G. Meredith und S. Weerawarana, „Web Services Description Language (WSDL) 1.1," *W3C Note,* 2001.

[38] D. Box, L. F. Cabrera, C. Critchley, F. Curbera, D. Ferguson und andere, „Web Services

Eventing (WS-Eventing),“ *W3C Member Submission,* 2006.

[39] V. Modi und D. Kemp, „Web Services Dynamic Discovery (WS-Discovery) Version 1.1,“ *OASIS Standard,* 2009.

[40] C. Hübner, H. Merz und T. Hansemann, Gebäudeautomation: Kommunikationssysteme mit EIB/KNX, LON und BACnet, Carl Hanser Verlag GmbH & Co. KG, 2009.

[41] S. T. Bushby, „BACnet - A standard communication infrastructure for intelligent buildings,“ *Automation in Construction,* pp. 529-540, 1997.

[42] „ARCNET - the deterministic, robust fieldbus,“ e.V., ARCNET User Group, [Online]. Available: http://www.arcnet.de/. [Zugriff am 14 März 2014].

[43] C. Müller, „BACnet als Standardbussystem in der Gebäudeautomation,“ Honeywell AG, 2002.

[44] N. Nardone und K. Schader, „BACnet Wireless Options Expand to Include ZigBee,“ *BACnet International,* 2011.

[45] International Organization for Standardization, Building automation and control systems - Part 5: Data communication protocol, DIN EN ISO 16484-5:2012 Norm, 2012.

[46] H. R. Kranz, BACnet Gebäudeautomation 1.12, Cci Dialog Gmbh, 2012.

[47] B. Swan, „The Language of BACnet-Objects, Properties and Services,“ Alerton Technologies, Inc., 1996.

[48] ASHRAE, „ANSI/ASHRAE Addendum c to ANSI/ASHRAE Standard 135-2004,“ 29 09 2006. [Online]. Available: http://www.bacnet.org/Addenda/Add-2004-135c.pdf. [Zugriff am 2014 06 12].

[49] ASHRAE, „Proposed Addendum am to Standard 135-2012, BACnet®,“ 04 2014. [Online]. Available: http://www.bacnet.org/Addenda/Add-135-2012am-PPR1-draft-26_chair_approved.pdf. [Zugriff am 2014 08 05].

[50] KNX Association, „Standardization,“ [Online]. Available: http://www.knx.org/knx-

en/knx/technology/standardisation/index.php. [Zugriff am 17 März 2014].

[51] Europäisches Komitee für Normung, „Elektrische Systemtechnik für Heim und Gebäude (ESHG) - Teil 1: Aufbau der Norm," DIN EN 50090-1:2011 Norm, 2011.

[52] W. Kastner, F. Praus, G. Neugschwandtner und W. Granzer, „KNX," in *The Industrial Electronics Handbook: Industrial Communications Systems, Second Edition*, CRC Press, 2011, pp. 42.1-42.13.

[53] KNX Association, „Grundlagenwissen zum KNX Standard," 2013.

[54] F. Praus, „A versatile networked embedded platform for KNX/EIB," Dimplomarbeit, 2005.

[55] KNX-Association, „Interworking," Home and Building Management Systems.

[56] U. Ryssel, H. Dibowski, H. Frank und K. Kabitzsch, „LonWorks," in *The Industrial Electronics Handbook: Industrial Communications Systems*, CRC Press, 2011, pp. 41.1-41.13.

[57] „LonWorks Technology Achieves ISO/IEC Standardization," LonMark International, 2008. [Online]. Available: http://www.lonmark.org/news_events/press/2008/1208_iso_standard. [Zugriff am 18 März 2014].

[58] International Organization for Standardization, „Information technology - Control network protocol - Part 1: Protocol stack," ISO/IEC EN 14908-1:2012 Norm, 2012.

[59] C. Brönniman, „Technische Grundlagen zur LonWorks Technologie," LonMark Schweiz, 2010.

[60] Lonmark International, „LonMark Resource Files Version 14," 2013. [Online]. Available: http://types.lonmark.org. [Zugriff am 19 März 2014].

[61] M. Galeev, „Catching the Z-Wave," *Electronic Engineering Times India,* 2006.

[62] International Telecommunication Union, „Short range narrow-band digital

radiocommunication transceivers – PHY and MAC layer specifications,“ in *Series G: Transmission Systems and Media, Digital Systems and Networks*, ITU-T Recommendation G.9959, 2012.

[63] J. Franck, „Z-Wave Node Type Overview and Network Installation Guide,“ Zensys A/S, 2008.

[64] N. T. Johansen, „Z-Wave Protocol Overview,“ Zensys A/S, 2006.

[65] Z-Wave.Me Team,, „Z-Way Developers Documentation,“ 2013.

[66] T. Köthke, „Self-powered Radio Technology for Building Automation Systems,“ *Hannover Messe,* 2011.

[67] International Organization for Standardization, „Wireless Short-Packet (WSP) protocol optimized for energy harvesting - Architecture and lower layer protocols,“ in *Information technology - Home Electronic Systems*, ISO/IEC 14543-3-10 Standard, 2012.

[68] A. Anders, „Reichweitenplannng für EnOcean Funksysteme,“ EnOcean GmbH, 2009.

[69] EnOcean GmbH, „EnOcean Radio Protocol 2,“ *Specification V1.0,* 2013.

[70] EnOcean GmbH, „Smart Acknowledge,“ *System Specification,* 2013.

[71] EnOcean Alliance, „EnOcean Equipment Profiles (EEP),“ *Technical Task Group Interoperability,* 2013.

[72] M. Dugré, F. Freyer und A. Anders, „BACnet and EnOcean enable Energy Efficient Buildings,“ 2009.

[73] T. Weinzierl und A. Anders, „KNX and EnOcean,“ 2009.

[74] LonMark International & EnOcean Alliance, „LonMark® International and EnOcean Alliance team-up for optimal network topologies in building automation,“ 2008.

[75] Z. Alliance, „Our goal is to provide standards to help you control your world,“ [Online]. Available: http://www.zigbee.org/About/AboutAlliance/TheAlliance.aspx. [Zugriff am

28 März 2014].

[76] S. Mahlknecht, T. Dang, M. Manic und S. A. Madani, „ZigBee," in *The Industrial Electronics Handbook - Industrial Communications Systems,* CRC Press, 2011, pp. 50.1-50.10.

[77] ZigBee Alliance, „ZigBee Specification," 2008.

[78] S. Ashton, „ZigBee Technology Overview," ZigBee Alliance, 2009.

[79] A. Elahi und A. Gschwender, ZigBee Wireless Sensor and Control Network, Prentice Hall, 2009.

[80] Dusan Stevanovic, „Zigbee / IEEE 802.15.4 Standard," ZigBee Alliance, 2007.

[81] ZigBee Alliance, „ZigBee Cluster Library Specification," 2012.

[82] Texas Instruments, „Z-Stack Developer's Guide," 2011.

[83] ZigBee Alliance, „ZigBee IP Specification," 2013.

[84] J. Ploennigs, „Drahtlose und drahtgebundene Sensoren-Aktoren-Netzwerke," Technische Universität Dresden, 2012.

[85] digitalSTROM AG, „digitalSTROM," [Online]. Available: http://www.digitalstrom.com/. [Zugriff am 1 April 2014].

[86] eQ3, „HomeMatic," [Online]. Available: http://www.eq-3.de/homematic.html. [Zugriff am 1 April 2014].

[87] Siemens AG, „IP Gateway KNX/BACnet N 143 – zertifizierte Systemintegration," 2014.

[88] G.-U. Vack, „Mapping zwischen LON und BACnet," SysMik GmbH, 2006.

[89] S. C. Park, W. S. Lee, S. H. Kim, S. H. Hong und P. Palensky, „Implementation of a BACnet-ZigBee Gateway," *8th IEEE International Conference on Industrial Informatics (INDIN),* pp. 40-45, 2010.

[90] G. Wütherich, N. Hartmann, B. Kolb und M. Lübken, Die OSGi Service Plattform, Heidelberg: dpunkt.verlag GmbH, 2008.

[91] B. Weber, P. Baumgartner und O. Braun, OSGi für Praktiker, München: Carl Hanser Verlag, 2010.

[92] Materna GmbH, „Java Multi Edition DPWS Stack Documentation“.

[93] Bundesamt für Sicherheit in der Informationstechnik , „Anforderungen an die Interoperabilität der Kommunikationseinheit eines intelligenten Messsystems für Stoff- und Energiemengen,“ Technische Richtlinie BSI TR-03109, 2011.

[94] Ernst & Young, „Kosten-Nutzen-Analyse für einen flächendeckenden Einsatz intelligenter Zähler,“ 2013.