BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Analyse, Planung und Implementierung einer internetfähigen Steuerung für elektrische Verbraucher im eHome-Bereich

Masterarbeit

von

Frank Schwarze

EDV.Nr.:753 569

Fachbereich VIII – Maschinenbau, Veranstaltungstechnik, Verfahrenstechnik –
Fernstudieninstitut
der Beuth Hochschule für Technik Berlin

zur Erlangung des akademischen Grades
Master of Engineering (M. Eng.)
im Master-Fernstudiengang
Industrial Engineering

Tag der Abgabe 12.06.2013

Betreuer:	Prof. Dipl.-Inform. Thomas Scheffler
Zweitgutachter:	Prof. Dr.-Ing. Peter Gober

Kurzfassung

Diese Arbeit beschreibt die Erstellung einer internetfähigen Steuerung für elektrische Verbraucher. Anforderungen an die Steuerung werden nach dem Kano-Modell definiert. Über eine Nutzwertanalyse werden vorhandene Techniken und Standards bewertet. Exemplarisch wird die Lösung mit dem größten Nutzwert implementiert.
Ein Zigbit-Modul, bestehend aus einem AVR Mikrocontroller und einem IEEE 802.15.4 Funkchip, bildet die Basis für die Hardware. Zusammen mit einem selbst dimensioniertem Kondensatornetzteil wird das Modul in einem Steckdosengehäuse verbaut.
Um zukunftssicher zu sein, wird das Protokoll IPv6 eingesetzt. Die Adaptionsschicht übernimmt das Protokoll 6LoWPAN. Das verwendete Betriebssystem Contiki besitzt eine fertige Webserver-Applikation, die für die eigenen Zwecke angepasst wird. Das Protokoll IEC 60870-5-104 wird neu implementiert. Es basiert auf dem TCP/IP-Modell und wird vor allem im Umfeld von Energieleitsystemen eingesetzt. Es eignet sich besonders für einen automatisierten Zugriff.
Über eine öffentliche Adresse des IPv6-Tunnelbrokers SixXS ist die Steuerung weltweit erreichbar und der elektrische Verbraucher kann über einen Webbrowser oder von einem Energieleitsystem ein- und ausgeschaltet werden.
Die Anforderungen nach dem Kano-Modell wurden nahezu vollständig erfüllt. Die Implementierung eines Webservers und einer IEC 60870-5-104-Applikation ist mit den gegebenen limitierten Ressourcen möglich. Anwendungsmöglichkeiten für die Steuerung liegen im Bereich eHome und Smart Grid.

Abstract

This Master Thesis describes the implementation of a solution to control and monitor electric consumers via the Internet. Needs of this solution are defined by use of the Kano model. Existing technologies and standards are benchmarked by means of a cost-utility analysis. The solution that scores the highest value of benefit will be implemented typically.
A Zigbit Module forms the basis of the hardware. It bundles an AVR microcontroller and an IEEE 802.15.4 transceiver. Together with a self-dimensioned capacitive power supply it is mounted in a socket housing.
To be future-proof, the IPv6 protocol is used. The 6LoWPAN protocol handles the adaptation layer. Contiki is used as operating system. It is delivered with a ready-to-use web server application which is customized for the own purposes. The IEC 60870-5-104 protocol is implemented from scratch. It is based on TCP/IP and is used in the field of energy management systems. It is particularly suitable for automated access.
Via a public address given by IPv6 tunnel broker SixXS the solution is accessible worldwide. The electric consumer can be switched on and off by the means of a web browser or an energy management system.
The needs according to the Kano model are almost completeley achieved. It is possible to implement a solution consisting of a web server and IEC 60870-5-104 application in ressource constraint environments. Possible applications for such a solution are in the field of home automation and smart grid.

Erklärung

Ich versichere, dass ich meine Prüfungsarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Datum Unterschrift

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

6LoWPAN	IPv6 over Low power Wireless Personal Area Network
AC	Alternating Current (Wechselstrom)
ANSI	American National Standards Institute
APCI	Application Protocol Control Information
APDU	Application Protocol Data Unit
API	Application Programming Interface
APS	Application Support Sublayer (ZigBee)
ASDU	Application Service Data Unit
BAN	Body Area Network
BLIP	Berkeley Low-power IP stack (TinyOS)
BSD	Berkeley Software Distribution
CA	Common Address
CB	Circuit Breaker
COT	Cause Of Transmission
CPU	Central Processing Unit
DAD	Duplicate Address Detection
DC	Direct Current (Gleichstrom)
EEPROM	Electrically Erasable Programmable Read-Only Memory
ELF	Executable and Linkable Format
EMV	Elektromagnetische Verträglichkeit
EUI-64	64-Bit Extended Unique Identifier
FCS	Frame Check Sequence
FIFO	First In First Out
FTP	File Transfer Protocol
GPRS	General Packet Radio Service
HAL	Hardware Abstraction Layer
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IOA	Information Object Address
IP	Internet Protocol
IPSO	Internet Protocol for Smart Objects (Alliance)
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
ISA	International Society of Automation
ISDN	Integrated Services Digital Network

ISM	Industrial, Scientific and Medical, ISM-Band
ISP	In-System Programming
JTAG	Joint Test Action Group
LAN	Local Area Network
LED	Light-Emitting Diode
LOC	Lines of Code
LSB	Least Significant Byte / Least Significant Bit
M2M	Machine-to-Machine
MAC	Media Access Control
MIB	Management Information Base
MRTG	Multi Router Traffic Grapher
MSB	Most Significant Byte / Most Significant Bit
MTU	Maximum Transmission Unit
NTP	Network Time Protocol
OID	Object Identifier
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
OUI	Organizationally Unique Identifier, Teil der EUI-64
PAN	Personal Area Network
PC	Personal Computer
PCB	Printed Circuit Board
PHP	Hypertext Preprocessor
PLC	Power Line Communication
RAM	Random Access Memory
RFC	Request For Comments
RIP	Routing Information Protocol
ROLL	Routing Over LoW power Lossy networks (IETF-Arbeitsgruppe)
ROM	Read Only Memory
RPL	Routing Protocol for Low power and Lossy Networks
RTU	Remote Terminal Unit
SCADA	Supervisory Control and Data Aquisition
SDM	Source Data Management
SFD	Start Frame Delimiter
SIM	Subscriber Identity Module
SNMP	Simple Network Management Protocol
SPI	Serial Peripheral Interface
SRD	Short Range Devices, Kurzstreckenfunk
SSI	Server Side Includes
TA	Technological Address
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
URL	Uniform Resource Locator
VAC	Volt Alternating Current (Wechselspannung)
VDC	Volt Direct Current (Gleichspannung)
VPN	Virtual Private Network
WAN	Wide Area Network
WBAN	Wireless Body Area Network
WLAN	Wireless Local Area Network
WPAN	Wireless Personal Area Network
WSN	Wireless Sensor Network

Kapitel 1

Einleitung

1.1 Aufgabenbeschreibung

Im Zuge dieser Masterarbeit soll eine Lösung erarbeitet und entwickelt werden, mit der ein beliebiger elektrischer Verbraucher über einen Fernzugriff gesteuert werden kann. Eine im Handel erhältliche Steckdose soll so modifiziert werden, dass die Stromzufuhr des Verbrauchers über das Internet ein- und ausschaltbar ist. Für den Zugriff auf die Steckdose soll keine zusätzliche Verkabelung notwendig sein. Das vorhandene Steuergerät der Steckdose soll durch eine selbst entwickelte Mikrocontrollerschaltung ersetzt werden. Der Mikrocontroller soll eine Applikation bereitstellen, damit menschliche Benutzer möglichst einfach auf die Steckdose zugreifen können.
Es sollen verschiedene existierende Technologien, die für diese Anwendung in Frage kommen, recherchiert und bewertet werden. Welche Technologien existieren bereits für ähnliche Anwendungen und welche Technologien lassen sich für diesen Anwendungsfall am Besten einsetzen? Eine Lösung soll beispielhaft implementiert und nach einem Funktionstest kritisch begutachtet werden. Dadurch soll gezeigt werden, in wie weit es möglich ist sehr kleine Geräte mit besonders limitierten Ressourcen an ein lokales Netzwerk und an das Internet anzubinden.

1.2 Herangehensweise

In Kapitel 2 werden zuerst zentrale Begriffe, wie Smart Objects und Internet-of-Things, erläutert. In Kapitel 2.1.4 wird erläutert, warum es sich bei der zu implementierenden Lösung nach dem eigenen Verständnis um ein Smart Object handelt. Deswegen wird in Kapitel 2.2 der schematische Aufbau der Hardware eines Smart Objects behandelt. Verschiedenen Technologien und Standards, die im Zusammenhang mit der Aufgabenstellung verwendet werden, werden vorgestellt. Diese Technologien werden eingeteilt in Übertragungstechniken und Kommunikationstechnologien. Eine Auswahl an Applikationen wird erläutert, die für den Zugriff auf die zu implementierende Lösung verwendet werden können. Betriebssysteme und Softwareumgebungen werden vorgestellt, die für die Bearbeitung der Aufgabenstellung in Frage kommen. Im Abschluss werden allgemein technische und nicht-technische Herausforderungen behandelt.
Die Technologien, die beispielhaft implementiert werden sollen, werden in Kapitel 3 ausgewählt. Dazu werden zuerst, unabhängig von bestimmten Technologien, Anforderungen beschrieben, die an eine implementierte Lösung gestellt werden. Die Erfüllung dieser Anforderungen wird am Ende dieser Arbeit geprüft. Für die Auswahl der Technologien werden verschiedene mögliche Komplettlösungen miteinander verglichen. Anhand einer Nutzwertanalyse wird die Komplettlösung ermittelt, die für die Aufgabenstellung am geeignetsten ist. Am Ende wird die Entscheidung hergeleitet, welche Lösung beispielhaft implementiert wird.
Das Kapitel 4 behandelt die Vorbereitungen für die Implementierung. Es wird die Hardware ausgewählt, die verwendet werden soll. Außerdem werden die Kommunikationstechnologie und die

Entwicklungsumgebung genauer erläutert.
Die Beschreibung der eigentlichen Implementierung ist in zwei Teil geteilt. Das Kapitel 5 behandelt die Hardwareimplementierung, das Kapitel 6 die Softwareimplementierung.
Nach Abschluss der Implementierung wurden Funktionstests der implementierten Applikationen durchgeführt. Der Testaufbau und die Funktionstests werden in Kapitel 7 behandelt.
Zum Ende der Arbeit wird die implementierte Lösung kritisch begutachtet. In Kapitel 8 wird geprüft, welche vorher beschriebenen Anforderungen erfüllt wurden. Der Nutzwert der implementierten Lösung wird ermittelt und mit dem Ergebnis der vorherigen Nutzwertanalyse verglichen. Verschiedene Anwendungsmöglichkeiten für die Implementierung und ähnliche Lösungsansätze werden behandelt. Ein Resümee der Arbeit wird gezogen.

Kapitel 2

Grundlagen

In diesem Kapitel wird nach anfänglichen Begriffserklärungen der schematische Hardwareaufbau eines Smart Objects erläutert. Es wird ein Überblick über verschiedene Technologien, die im Zusammenhang mit der Aufgabenstellung verwendet werden können, gegeben. Diese Technologien werden hier kurz angesprochen und einsortiert. Technische Details werden, soweit notwendig, in späteren Kapiteln erörtert.
Um darzustellen, wie ein Benutzer auf die zu implementierende Lösung zugreifen kann, werden verschiedene Applikationen vorgestellt. Außerdem werden verschiedene Betriebssysteme vorgestellt, die für eine Softwareimplementierung in Frage kommen.
Zum Abschluss dieses Kapitels werden technische und nicht-technische Herausforderungen angesprochen.

2.1 Begriffserklärungen

Um die später vorgestellten Übertragungstechniken und Kommunikationstechnologien besser einordnen zu können, wird zu Beginn auf das OSI-7-Schichtenmodell eingegangen.
Außerdem werden weitere Begriffe erläutert, die helfen sollen, die zu implementierende Lösung einzugruppieren. Teilweise handelt es sich um englische Begriffe, weil diese auch in der deutschen Literatur gebräuchlicher sind.

2.1.1 OSI-7-Schichtenmodell

Das OSI-7-Schichtenmodell wird als bekannt vorausgesetzt. Weil es aber verschiedene Interpretationen und Namensgebungen gibt, werden in der Tabelle 2.1 Begriffe zugeordnet, wie sie in dieser Arbeit verwendet werden. Es werden die deutschen und englischen Begriffe für die sieben Schichten aufgelistet. Zusätzlich wird die Einordnung nach dem TCP/IP-Modell [RFC1122] gezeigt und es werden für jede Schicht Beispielprotokolle genannt.

Schicht			**Beispielprotokoll**	**TCP/IP-Modell**
7	Anwendungsschicht	Application Layer	HTTP, FTP, SNMP	Application Layer
6	Darstellungsschicht	Presentation Layer		
5	Sitzungsschicht	Session Layer		
4	Transportschicht	Transport Layer	TCP, UDP	Transport Layer
3	Vermittlungsschicht	Networking Layer	IPv4, IPv6, ICMP	Internet Layer
2	Sicherungsschicht	Data Link Layer	Ethernet, WLAN, ISDN	Link Layer
1	Physikalische Schicht	Physical Layer		

Tabelle 2.1: OSI-7-Schichtenmodell

Das OSI-7-Schichtenmodell wird in dieser Arbeit verwendet, um Übertragungstechniken und Kommunikationstechnologien einordnen zu können.

2.1.2 Eingebettete Systeme

Eingebettete Systeme – oft wird auch der englische Begriff embedded systems verwendet – sind kleine Computer, die Teilaufgaben in einem größeren technischen Kontext übernehmen. Sie bestehen häufig aus Mikrocontrollern oder digitalen Signalprozessoren und sind dafür ausgelegt eine bestimmte Funktion, oder auch mehrere bestimmte Funktionen, zu übernehmen. Im Gegensatz zu herkömmlichen Computern ist es nicht möglich, die Funktion eines eingebetteten Systems zu ändern [Heath 2003].

2.1.3 Drahtlose Sensornetze

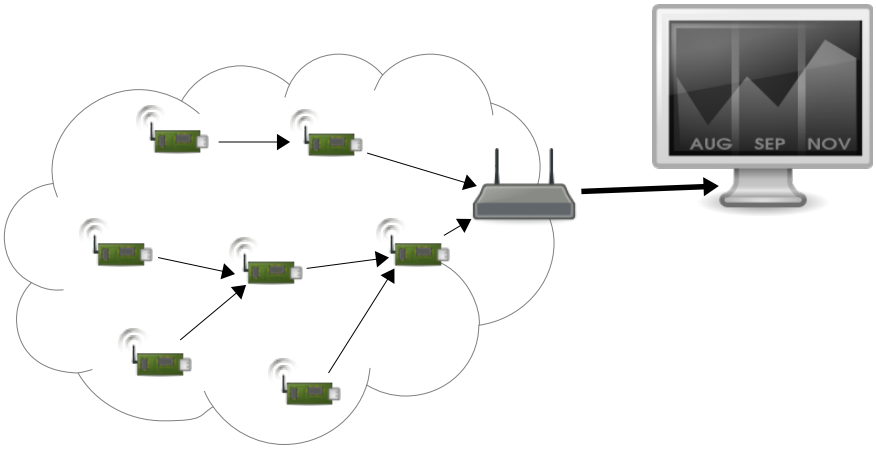

Abbildung 2.1: Aufbau eines drahtlosen Sensornetzwerkes

Bei drahtlosen Sensornetzen, international auch Wireless Sensor Networks (WSN) genannt, handelt es sich um Netzwerke aus kleinen Sensoren, die ihre Messergebnisse zu einer zentralen Station senden. Die Sensoren helfen dabei einander, die Informationen weiterzureichen (siehe Abbildung 2.1). So ein drahtloses Sensornetz kann zum Beispiel in einem Gebäude Temperatur- und Luftfeuchtigkeitswerte an eine zentrale Station liefern, die dann die Klimaanlage entsprechend ansteuert. Eine komplette Verkabelung jedes einzelnen Sensors kann dabei je nach Anzahl sehr aufwendig und kostspielig sein. Wenn die Sensoren dazu noch mobil sein sollen, empfiehlt sich eine drahtlose Kommunikation.
Ein Sensor in einem drahtlosen Sensornetz ist ausgestattet mit einer Kommunikationseinheit, die sich selbstständig innerhalb des Netzes konfiguriert und darüber die eingelesenen Messgrößen zu einer zentralen Stelle transportiert.

2.1.4 Smart Objects

Diese Definition von Smart Objects beruht auf [Vasseur und Dunkels 2010]. Dabei handelt es sich um kleine Objekte, die mit folgenden Einheiten ausgestattet sind:

- eine Form von Sensor und/oder Aktor
- ein kleiner Mikrocontroller
- eine Kommunikationseinheit
- eine Energieversorgung

Ein Sensor und/oder Aktor wird benötigt, um mit der Umwelt interagieren zu können. Ein Sensor kann bestimmte Messgrößen erfassen, die dann verarbeitet werden können. Ein Aktor ist das Gegenstück zu einem Sensor. Über ihn kann aktiv die Umwelt beeinflusst werden. Die Kommunikationseinheit befähigt das Smart Object über ein Netzwerk kommunizieren zu können. Es ist dabei möglich mit anderen Smart Objects zu kommunizieren als auch mit weit entfernten Geräten, im Falle des Internets weltweit. Ein kleiner Mikrocontroller übernimmt die Informationsverarbeitung. Dieser nimmt Daten von Sensoren entgegen bzw. steuert den Aktor. Die Kommunikation wird ebenso von ihm geregelt, von einem Sensor gemessene Werte können über das Netzwerk weitergegeben werden. Der Mikrocontroller kann als Kern des Smart Object angesehen werden. Die Energieversorgung ist notwendig, um alle Einheiten ausreichend mit elektrischer Energie zu versorgen.
Alle diese technischen Eigenschaften machen ein Objekt aber noch nicht zu einem Smart Object. Erst die Anwendung und sein Verhalten machen aus einem Objekt mit den obigen Eigenschaften ein Smart Object. Allerdings ist es sehr schwierig, dieses Verhalten zu definieren. Die Anwendungen sind sehr unterschiedlich und es ist völlig unbekannt, wie Smart Objects in der Zukunft eingesetzt werden. Gemeinsam ist allen Anwendungen allerdings, dass Smart Objects mit der physikalischen Umwelt interagieren und über ein Netzwerk kommunizieren. Dieses Verhalten und die obigen technischen Eigenschaften machen ein Smart Object aus.
Die Technologie von Smart Objects ist größtenteils keine neue Erfindung. Geräte mit ähnlichen oder teilweise sogar gleichen technischen Eigenschaften lassen sich auch bei eingebetteten Systemen finden. Diese können Sensoren oder Aktoren besitzen und verwenden in der Regel eine Schnittstelle, mit der sie angesteuert werden können. Der Unterschied liegt in der Anwendung, beziehungsweise wo der Schwerpunkt eines Systems bzw. Objekts liegt. Eingebettete Systeme verarbeiten strikt die Funktion für die sie entwickelt worden sind. Sie agieren meistens mit einem zentralen Steuergerät, sind aber abgeschottet in ihrem technischen Kontext. Smart Objects können mit einem großen Netzwerk kommunizieren und sind, wenn das Internet als Infrastruktur verwendet wird, weltweit erreichbar.
Von der Kommunikationstechnologie her ähnelt ein Smart Object einem Sensor in einem Sensornetz. Allerdings ist im Gegensatz zu einem Sensornetz ein Netz aus Smart Objects nicht allein auf die Datenübermittlung fokussiert. Bei einem Sensornetz ist der Datenfluss immer vom Sensor ins Netzwerk, wohingegen der Datenfluss bei einem Smart Object bidirektional ist. Der Fokus liegt dabei auf eine Vielzahl von Funktionen insbesondere der Steuerung und Überwachung [Vasseur und Dunkels 2010, Seite 12]. Es ist also die Anwendung, die ein Gerät zu einem Smart Object macht. Aber Entwicklungen, hauptsächlich in der Energieversorgung und Kommunikationstechnik, kommen beiden Forschungsgebieten zugute.
Anhand der Definition wird deutlich, dass es sich bei dieser Arbeit um ein Smart Object handelt. Es wird ein Aktor benötigt, um über ein Relais den elektrischen Verbraucher ein- oder ausschalten zu können. Über den Zustand des Aktors ist auch gleichzeitig bekannt, ob der elektrische Verbraucher ein- oder ausgeschaltet ist. In diesem Sinne ist der Aktor auch gleichzeitig ein Sensor. Die Arbeit könnte aber zum Beispiel zusätzlich um einen Energieverbrauchssensor erweitert werden, der die aktuelle Leistungsaufnahme des Verbrauchers misst. Für den Fernzugriff wird eine Kommunikationseinheit verwendet. Der Mikrocontroller wird benötigt, um eine für den Benutzer einfach zu handhabende Applikation zur Verfügung zu stellen. Eine Energieversorgung ist selbstverständlich auch notwendig. Idealerweise wird bei dieser Arbeit die notwendige Energie mithilfe der verwendeten Steckdose aus dem angeschlossenen Stromnetz bezogen.

2.1.5 Internet-of-Things

Das Internet-of-Things, auch Internet der Dinge genannt, ist eine Vision, wie sich das Internet in Zukunft entwickeln kann. Es soll aus der Vernetzung vieler Kleinstgeräte – auch oder vielleicht hauptsächlich Smart Objects – bestehen. Shelby und Bormann [Shelby und Bormann 2009] be-

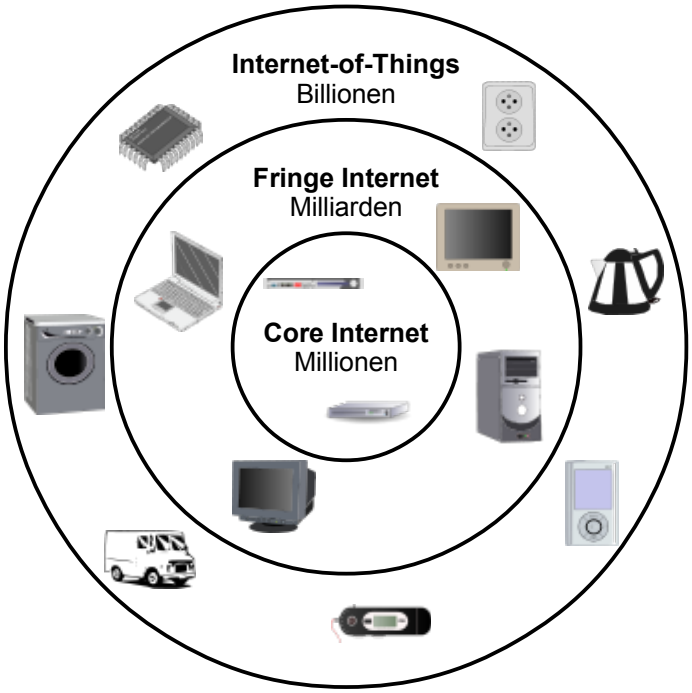

Abbildung 2.2: Die Internet-of-Things Vision [Shelby und Bormann 2009, Figure 1.2]

trachten das Internet-of-Things als eine weitere Schale des Internets (Abbildung 2.2), die gerade anfängt sich heraus zu bilden. Der Kern des Internets (Core Internet) besteht heute aus vielen Routern und Servern, die zusammen Millionen von Teilnehmern ausmachen. Um diesen Kern herum liegt das sogenannte Fringe Internet, was beispielsweise aus privaten Rechnern oder Laptops besteht, aber auch aus lokalen Netzwerken, die zum Internet verbunden werden. Die Teilnehmer des Fringe Internet gehen in die Milliarden. Um dieses Fringe Internet herum beginnt sich eine weitere Schale zu bilden, das sogenannte Internet-of-Things. Auf lange Sicht kann hier mit Billionen von Teilnehmern gerechnet werden. Hauptsächlich bestehen diese Teilnehmer aus IP-fähigen eingebetteten Systemen (embedded systems). Eine so große Zahl an Teilnehmern kann nur mithilfe des Internet Protokolls der Version 6 ans Internet angeschlossen werden, da das bisherige Internet Protokoll der Version 4 keine freien Adressräume mehr zur Verfügung hat.

2.2 Hardware

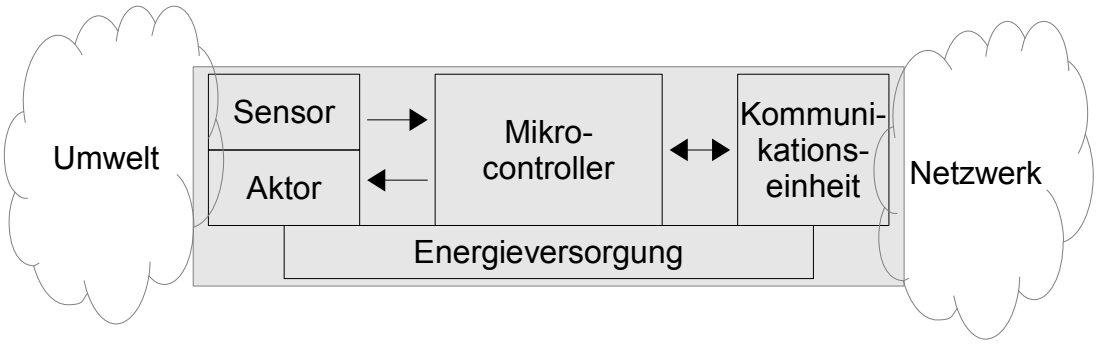

Abbildung 2.3: Schematischer Aufbau eines Smart Object

Wie schon in Kapitel 2.1.4 erwähnt, bestehen Smart Objects aus Sensoren und/oder Aktoren, einem Mikrocontroller, einer Kommunikationseinheit und einer Energieversorgung. In Abbildung 2.3 ist dies schematisch dargestellt. Die Elemente eines Smart Objects werden näher erläutert und zusätzlich wird mit dem AVR Ravenboard von Atmel beispielhaft ein Entwicklungskit aus dem Bereich Smart Objects vorgestellt.

2.2.1 Sensoren und Aktoren

Sensoren und Aktoren werden benötigt, um Smart Objects die Möglichkeit zu geben, mit ihrer Umwelt zu interagieren. Ein Sensor ist ein technisches Bauteil, das physikalische Eigenschaften aus der Umgebung misst und in ein elektrisches Signal umwandelt. Dieses Signal kann dann weiterverarbeitet werden. Ein Aktor ist das Gegenstück dazu. Ein Aktor wandelt elektrische Signale in physikalische Größen beziehungsweise mechanische Bewegung um. Es gibt eine Vielzahl von Sensoren und Aktoren, die teilweise sehr einfach funktionieren, aber auch hoch kompliziert sein können.
Ein typisches Beispiel für einen Sensor ist ein Pt100-Messwiderstand [Weichert und Wülker 2010]. Dies ist ein Temperaturfühler bestehend aus dem Element Platin, der abhängig von der Temperatur seinen elektrischen Widerstand ändert. Platin hat den Vorteil, dass die Abhängigkeit zwischen Temperatur und elektrischen Widerstand linear ist. Der Pt100 ist normiert nach IEC 60751 [IEC60751], bei einer Temperatur von 0°C beträgt sein Widerstand 100 Ohm und bei einer Temperatur von 100°C 138,51 Ohm. Wenn so ein Pt100-Widerstand über eine entsprechende Messschaltung an einen analogen Eingangspin eines Mikrocontrollers angeschlossen wird, kann über die anliegende Spannung die Temperatur berechnet und weiterverarbeitet werden.
Die einfachste Variante eines Sensors ist ein binärer Eingangspin am Mikrocontroller, an dem über einen Spannungspegel eine logische 1 oder eine logische 0 anliegt. Der Mikrocontroller kann dieses Bit auslesen und weiterverarbeiten.
Bei der Entwicklung der Steckdose wird als Aktor ein Relais verwendet, mit dem der Stromfluss ein- und ausgeschaltet werden kann. Angesteuert wird das Relais über einen Ein- bzw. Ausgangspin des Mikrocontrollers. Wenn dieser Pin von der Software auf logisch 1 gesetzt wird, zieht das Relais an und der Strom kann fließen. Bei logisch 0 ist das Relais offen und der Stromfluss unterbrochen.

2.2.2 Mikrocontroller

Der Mikrocontroller ist das Herzstück eines Smart Objects. Auf ihm ist die Software gespeichert und hier läuft das erstellte Programm ab. Die Sensoren und Aktoren werden angesteuert und mithilfe der Kommunikationseinheit wird mit der Außenwelt kommuniziert. Ein Mikrocontroller besteht im Gegensatz zu einem Mikroprozessor oder auch CPU (Central Processing Unit) aus mehreren Modulen, die auf einem Chip integriert sind. Eine CPU benötigt immer externe Peripherie, während ein Mikrocontroller die Basisperipherie bereits integriert hat [Schäffer 2008]. Außer der Recheneinheit befindet sich auf dem Mikrocontroller-Chip der Speicher, steuerbare Ein- und Ausgänge und andere optionale Zusatzmodule wie serielle Schnittstellen oder Analog/Digital-Wandler. Beim Speicher wird zwischen flüchtigen Speicher und dauerhaften Speicher unterschieden. Ein flüchtiger Speicher ist der Arbeitsspeicher des Mikrocontrollers oder auch RAM (Random Access Memory). Das Programm eines Mikrocontrollers befindet sich im dauerhaften Speicher. Üblicherweise besitzt ein Mikrocontroller zwei dauerhafte Speicher. Ein Flash-Speicher, in dem das Programm gespeichert wird und einen internen EEPROM (Electrically Erasable Programmable Read-Only Memory), in dem Konfigurationsparameter und ähnliches gespeichert werden. Der Programmcode befindet sich nur im Flash-Speicher, er lässt sich in der Regel während der Programmlaufzeit nicht ändern. Auf dem EEPROM-Speicher kann lesend und schreibend zugegriffen werden. Da beides jedoch zeitintensiv ist, wird er hauptsächlich für Konfigurationsparameter verwendet, die nur einmal eingelesen werden.

2.2.3 Kommunikationseinheit

Mithilfe der Kommunikationseinheit kann ein Smart Object mit anderen Geräten kommunizieren. Diese Kommunikation kann je nach verwendeter Technologie drahtgebunden oder drahtlos sein. Zwei Geräte müssen die gleiche Kommunikationstechnologie verwenden, um miteinander kommunizieren zu können. Im Kapitel 2.3 und 2.4 werden ausgewählte Übertragungstechniken und Kommunikationstechnologien beschrieben.
Die Kommunikationseinheit hat einen großen Anteil an der Leistungsaufnahme eines Smart Objects. Um ein Smart Object möglichst energiesparend zu konzeptionieren, ist die Wahl der Kommunikationseinheit besonders wichtig.
Auf dem Markt sind auch fertige Module erhältlich, die den Mikrocontroller, die Kommunikationseinheit und deren Verschaltung vereinen. Als Beispiel für solche single chip Lösungen sei das Radiocraft RC2400 [RC2400] genannt und das später noch näher beschriebene ATZB-24-A2 von Atmel [ATZB-24-A2].

2.2.4 Energieversorgung

Die Energieversorgung wird benötigt, um den Mikrocontroller und die Kommunikationseinheit mit Energie zu versorgen. Eine weit verbreitete Energiequelle für Smart Objects ist die Batterie. Um eine möglichst lange Lebenszeit auch mit kleinen Batterien zu gewährleisten, wird bei Smart Objects auf eine energieeffiziente Implementierung geachtet. Energie kann aber auch aus der Umwelt heraus bezogen werden. Das kann zum Beispiel eine eigene kleine Solarzelle sein, ein Thermoelement oder ein Piezoelement. Ein RFID-Chip [Finkenzeller 2008] bezieht seine Energie aus dem Funksignal des Abfragegerätes.

2.2.5 AVR Ravenboard

Abbildung 2.4: Atmel AVR RZ Raven Evaluierungskit [AVR RZRaven]

Zu Evaluierungszwecken bietet die Firma Atmel mit dem AVR RZ Raven [AVR RZRaven] ein Packet an, bestehend aus zwei AVR Ravenboards und einem sogenannten RZUSBSTICK (Abbildung 2.4).
Die beiden Ravenboards sind fertig zusammengestellte Einheiten bestehend aus einem Mikrocontroller vom Typ ATmega1284P und einer Kommunikationseinheit vom Typ AT86RF230. Es handelt sich dabei um einen 2,4GHz Funkchip, der auf dem Standard IEEE 802.15.4 basiert. Zusätzlich enthält es noch ein LCD Display, ein Eingabe-Joystick und ein Audiogerät. Diese drei Einheiten werden von einem zweiten Mikrocontroller ATmega3290P angesteuert. Betrieben wird das Ravenboard über eine externe 5-12V Spannungsquelle oder alternativ über zwei handelsübliche 1,5V Batterien.

Der RZUSBSTICK basiert auf einem Mikrocontroller AT90USB1287 und der gleichen Kommunikationseinheit AT86RF230. Über eine USB Schnittstelle ist es möglich, den RZUSBSTICK an einen PC anzuschließen.
Dieses Evaluierungskit von Atmel soll dazu verwendet werden, Anwendungsmöglichkeiten von Protokollen wie IEEE 802.15.4, 6LoWPAN oder ZigBee zu betrachten. Es kann zu Entwicklungszwecken, zum Austesten oder als Gegenstand von Vorführungen verwendet werden. Aufgrund der Architektur der Ravenboards und des RZUSBSTICK kann es ebenso zur Evaluierung von Smart Objects verwendet werden.

2.3 Übertragungstechniken

Es werden verschiedene Übertragungstechniken vorgestellt. Bezogen auf das OSI-7-Schichtenmodell (Tabelle 2.1) werden hier die ersten beiden Schichten, die physikalische Schicht und die Sicherungsschicht behandelt. Die Reihenfolge der aufgezählten Übertragungstechniken ist nicht von Bedeutung.

2.3.1 Ethernet

Die Kommunikationsstandardfamilie IEEE 802.3 [IEEE802.3] wird gemeinhin als Ethernet bezeichnet. Es spezifiziert die Linkschicht und die physikalische Schicht für kabelgebundene Verbindungen. Dabei stehen verschiedene Übertragungsformen, u.a. Übertragungsgeschwindigkeiten, Stecker und Kabeltypen zur Verfügung. Ethernet kann als Basis von mehreren höheren Protokollen eingesetzt werden, wird aber heutzutage hauptsächlich für TCP/IP eingesetzt.
Es stehen verschiedene Ethernet Controller mit SPI-Schnittstelle (Serial Peripheral Interface) für den Anschluss an einen Mikrocontroller zur Verfügung, z.B. der ENC28J60 von Microchip [ENC28J60]. Ethernet ist eine sehr verbreitete und ausgereifte Technologie. Ihr Einsatz mit Smart Objects ist dagegen nicht sehr weit verbreitet. Zum einem liegt das am zusätzlichem Platzbedarf. Eine Ethernet-Buchse benötigt zusätzlichen Platz, der bei einem Smart Object entscheidend sein kann. Zum anderen liegt es daran, dass Ethernet eine kabelgebundene Technologie ist. Sie kann nur eingesetzt werden, wenn das Smart Object mit einem Ethernet-Kabel ins Netzwerk angeschlossen werden kann. Oft werden Smart Objects aber in großen Netzwerken oder auch ortsungebunden betrieben. Da wäre eine komplette Verkabelung zu kostspielig.

2.3.2 WLAN

WLAN (Wireless Local Area Network) – oder international auch WiFi genannt – ist in der Standardfamilie IEEE 802.11 [IEEE802.11] spezifiziert. Es ist eine drahtlose Übertragungstechnik, die vor allem im Heimbereich und bei öffentlichen Internetzugängen verbreitet ist. Hauptsächlich wird das 2,4 GHz Frequenzband verwendet, das in mehrere Kanäle aufgeteilt wird. Je nach Protokollspezifikation können Reichweiten von ca. 100 Metern und Datenraten von bis zu 600 MBit/s erreicht werden.
Vor allem kleinere Geräte wie Smartphones und Tablets verwenden häufig WLAN. Mittlerweile ist die Anwendung sehr einfach, was zu einer großen Verbreitung beigetragen hat. Es sind wenig bis teilweise keine Netzwerkkenntnisse notwendig, um ein WLAN-fähiges Gerät in ein bestehendes WLAN anzuschließen.
Für den Anschluss an einen Mikrocontroller stehen WLAN-Module von verschiedenen Herstellern zur Verfügung. Weil für Mikrocontroller-Anwendungen der Energieverbrauch oft entscheidend ist, bieten viele Anbieter ein Low-Power WLAN an. Damit kann der Energieverbrauch von üblicherweise ca. 800mW auf ca. 300mW reduziert werden [Vasseur und Dunkels 2010, Seite 162].

2.3.3 IEEE 802.15.4

Der Standard IEEE 802.15.4 [IEEE802.15.4] beschreibt eine Form von drahtloser Kommunikation mit einer relativ geringen Reichweite. Er wird in WPANs (Wireless Personal Area Networks) eingesetzt und gewährleistet eine kostengünstige und energiesparende Lösung für verschiedene Gerätetypen.
Ähnlich wie bei WLAN wird auch hier hauptsächlich das 2,4 GHz Frequenzband verwendet, das in mehrere Kanäle unterteilt ist. Eine parallele Verwendung von WLAN und IEEE 802.15.4 kann zu einer Verschlechterung der Linkqualität führen, wenn Kanäle verwendet werden, die sich überlagern. Im Gegensatz zu WLAN ist die Reichweite und die Datenrate geringer. Es können nur Datenraten bis zu 250 kBit/s erreicht werden. Dafür ist der Energieverbrauch zum Senden und Empfangen deutlich reduziert. Typische Werte liegen bei ca. 50mW [Vasseur und Dunkels 2010, Seite 158].
Zusätzlich zum 2,4 GHz Frequenzband, das weltweit gilt, steht in Europa die Frequenz 868 MHz und in den USA das Frequenzband 915 MHz zur Verfügung. Auf diesen Frequenzen werden geringere Datenraten, 20 kBit/s bzw. 40 kBit/s, erreicht.
IEEE 802.15.4 ist ähnlich wie Ethernet und WLAN in zwei Schichten aufgebaut. Die physikalische Schicht spezifiziert, wie Datenframes übertragen und empfangen werden sollen. Sie spezifiziert das Funkmedium, welche Frequenzbänder zur Verfügung stehen und wie ein Datenframe moduliert wird. Die zweite Schicht, die Linkschicht oder auch MAC-Layer (Media Access Control Layer), spezifiziert die Datenframestruktur und wie Daten von einer höheren Schicht in ein IEEE-802.15.4-Datenframe eingekapselt werden.
Das Protokoll IEEE 802.15.4 wird von verschiedenen höheren Protokollen verwendet, insbesondere von IPv6 mittels 6LoWPAN und von ZigBee. Beide werden in den nächsten Kapiteln erläutert.

2.3.4 Power Line Communication

PLC (Power Line Communication) ist eine Technik, um Informationen über ein bestehendes Stromnetz zu übertragen. Das kann ein Stromnetz innerhalb eines Gebäudes sein und damit ein lokal begrenztes, es ist aber auch möglich Informationen über Hochspannungsleitungen und damit über lange Strecken zu übertragen. Angewandt werden dabei verschiedene Technologien, die aber alle nach dem gleichen Prinzip arbeiten. Es wird auf der vorhandenen Wechselspannung, je nach Region 50Hz oder 60Hz, eine zusätzliche höhere Frequenz mit geringer Amplitude aufmoduliert, in der die zu übertragende Information kodiert ist. Für den Stromverbraucher ist diese Frequenz nicht wahrnehmbar. Allerdings kann durch große Änderungen im Stromfluss die aufmodulierte Frequenz gestört werden.
Als Beispiel für eine offen spezifizierte Power Line Communication kann IEEE 1901 [IEEE1901] „IEEE Standard for Broadband over Power Line Networks: Medium Access Control and Physical Layer Specifications“ genannt werden. Umgangssprachlich wird dieser Standard oft als Powerline oder Homeplug bezeichnet.

2.3.5 GPRS/UMTS

GPRS (General Packet Radio Service) und UMTS (Universal Mobile Telecommunications System) sind Möglichkeiten das bestehende Mobilfunknetz für die Kommunikation zu verwenden. Es gibt GPRS- und UMTS-Module im Handel, die an einen Mikrocontroller angeschlossen werden können. Zum Betrieb eines GPRS- oder eines UMTS-Moduls ist eine SIM-Karte von einem Mobilfunkanbieter notwendig.
Für Anwendungen mit einer großen Anzahl von Smart Objects ist diese Technologie nicht geeignet, da jedes Smart Object eine eigene SIM-Karte benötigt. In der Regel sprengt dies den preislichen Rahmen. Für einzelne Smart Objects ohne anschließende Infrastruktur ist dies aber durchaus eine Alternative.

2.4 Kommunikationstechnologien

Im folgenden Unterkapitel werden verschiedene Kommunikationstechnologien aufgezählt. Nach dem OSI-7-Schichtenmodell (Tabelle 2.1) kann grob gesagt werden, dass hier die Vermittlungsschicht und die Transportschicht behandelt werden. Ganz exakt ist diese Einordnung allerdings nicht, weil mehrere vorgestellte Protokolle und Standards bis in die Anwendungsschicht gehen oder sich nicht an dem klassischen OSI-7-Schichtenmodell orientieren.

2.4.1 TCP/IP

TCP/IP bezeichnet in dieser Arbeit die Internetprotokollfamilie. Sie wird von der IETF (Internet Engineering Task Force) mittels sogenannter RFCs (Request For Comments) spezifiziert. Die Basis von TCP/IP umfasst das Protokoll IP, das in der Vermittlungsschicht angesiedelt ist, und das Protokoll TCP oder alternativ das Protokoll UDP, das die Funktionen der Transportschicht übernimmt. Das heutige Internet basiert auf TCP/IP.
Bei IP werden zwei Versionen unterschieden: IPv4 und IPv6. Der markanteste Unterschied beider Versionen ist die mögliche Anzahl an IP-Adressen. Eine IP-Adresse dient zur Identifikation eines IP-Knotens innerhalb eines Netzwerk. Bei IPv4 besteht eine IP-Adresse aus 32 Bit. Damit sind rein rechnerisch – de facto ist ein kleiner Teil für spezielle Aufgaben reserviert – ca. 4,3 Milliarden Adressen möglich. Bei IPv6 besteht eine IP-Adresse aus 128 Bit. Damit sind ca. 3,4 * 10^{38} Adressen möglich. Auch die Schreibweise ist unterschiedlich. IPv4-Adressen werden als vier dezimale 8-Bit-Blöcke geschrieben, wohingegen IPv6-Adressen hexadezimal in 16-Bit-Blöcken geschrieben werden (siehe Listing 2.1). Im Jahr 2011 sind von der IANA (Internet Assigned Numbers Authority) im Internet-Netzwerk die letzten freien IPv4-Adressblöcke vergeben worden [Web:ipv4-address-space]. Die Verwendung von IPv6 ist also zwingend erforderlich, um dem wachsenden Internet gerecht zu werden.

Listing 2.1: Schreibweise IP-Adressen

```
IPv4-Adresse: 173.194.69.94
IPv6-Adresse: 2a00:1450:4008:c01::5e
```

Das Protokoll IP vermittelt zwischen zwei Teilnehmern innerhalb eines Netzwerks. Um Daten zwischen beiden Teilnehmern transportieren zu können, werden die Protokolle UDP und TCP verwendet. UDP ist eine verbindungsloses Protokoll. Mit UDP werden Daten übertragen ohne sicherzustellen, dass diese korrekt von der Gegenseite empfangen werden. Entweder ist dies nicht notwendig, oder eine Sicherstellung wird von der Anwendungsschicht übernommen. Im Gegensatz dazu ist TCP verbindungsorientiert. Bei TCP wird eine Verbindung aufgebaut, innerhalb der Daten übertragen werden. TCP übernimmt dabei eine Sicherung, dass die Daten auf der Gegenseite empfangen werden. Falls einzelne Pakete verloren gehen, werden sie wiederholt gesendet.
TCP/IP hat den Vorteil, dass es sehr weit verbreitet ist. Mit dem Internet gibt es ein sehr großes weltweites Netzwerk. Aber auch viele kleine geschlossene Netzwerke, in denen früher auch andere Protokolle verbreitet waren, verwenden mehr und mehr TCP/IP.

2.4.2 6LoWPAN

Der Standard IEEE 802.15.4 zielt auf eine kostengünstige und energiesparende Kommunikation in einem drahtlosen PAN (Personal Area Network). 6LoWPAN ist eine Spezifizierung der IETF, wie über IEEE 802.15.4 IPv6-Packete übertragen werden können. Es wirkt damit wie eine Brücke zwischen IEEE 802.15.4 und TCP/IP. Für die Verwirklichung mussten mehrere Schwierigkeiten beachtet werden, die in [RFC4919] formuliert wurden. Eine Schwierigkeit dreht sich um die MTU (Maximum Transmission Unit), die vom IPv6 Standard mit mindestens 1280 Bytes gefordert wird [RFC2460, Section 5]. Der Standard IEEE 802.15.4 lässt nur Datenframelängen von maximal 127

Bytes zu, inklusive MAC-Header. Dieser hat eine variable Größe, abhängig vom Adressierungsverfahren und von optionalen Sicherheitsfunktionen, und kann ohne Sicherheitsfunktionen bis zu 25 Bytes betragen. Damit bleiben 102 Bytes für das IPv6-Packet. Da dies sehr viel geringer als die geforderten 1280 Bytes sind, wurde mit 6LoWPAN eine Adaptionsschicht eingeführt, die es ermöglicht, ein IPv6-Packet in fragmentierten IEEE 802.15.4 Datenframes zu übertragen [RFC4944, Section 4]. Diese Adaptionsschicht wird durch einen eigenen 6LoWPAN-Header abgebildet.
Allerdings bedeutet eine Fragmentierung immer zusätzlichen Overhead. Um eine Fragmentierung überhaupt zu vermeiden bzw. zu verringern bietet 6LoWPAN eine Header-Komprimierung. Diese Komprimierung zielt hauptsächlich auf den IPv6 Header. Dieser Header hat eine Größe von 40 Bytes. Durch eine ausgeklügelte Komprimierung kann – unter bestimmten Voraussetzungen – der Header auf wenige Bytes verringert werden. Aber die Komprimierung kann auch auf höhere Header, wie zum Beispiel UDP oder IPv6-Erweiterungsheader (Extension Header), wirken.
Für den Standard 6LoWPAN (IPv6 over Low power WPAN) gibt es eine eigene IETF Arbeitsgruppe. Zur Zeit gibt es mehrere Richtungen, an denen gearbeitet wird. So wird zum Beispiel an einer Implementierung von 6LoWPAN über Bluetooth Low Energy [ID-6lowpan-btle] gearbeitet. Auch neue Komprimierungsarten und damit 6LoWPAN-Header werden vorgeschlagen [ID-6lowpan-ghc]. Eine Übersicht über laufende Arbeiten liefert [ID-6lowpan-roadmap].

2.4.3 ZigBee

ZigBee ist ein Quasi-Standard, der von der ZigBee Alliance – ein Zusammenschluss von verschiedenen Firmen – herausgegeben und laufend weiterentwickelt wird. Er basiert auf dem Kommunikationsstandard IEEE 802.15.4 und spezifiziert darauf aufbauend eine Netzwerkschicht und eine Anwendungsschicht, bestehend aus einem sogenannten Application Support Sublayer (APS) und einem Application Framework [Hersent et al. 2012, Seite 93ff]. Für deren Verwendung stehen je nach Anwendungsgebiet verschiedene Applikationsprofile [Web:ZigBee] zur Verfügung:

- ZigBee Building Automation (Efficient commercial spaces)
- ZigBee Remote Control (Advanced remote controls)
- ZigBee Smart Energy (Home energy savings)
- ZigBee Health Care (Health and fitness monitoring)
- ZigBee Home Automation (Smart homes)
- ZigBee Input Device (Easy-to-use touchpads, mice, keyboards, wands)
- ZigBee Light Link (LED lighting control)
- ZigBee Retail Services (Smarter shopping)
- ZigBee Telecom Services (Value-added services)

Ein Applikationsprofil umfasst dabei eine Menge an Meldungen und Attributen für die Verwendung im jeweiligen Anwendungsumfeld. Ziel ist es unter anderem, eine gewisse Kompatibilität zwischen verschiedenen Herstellern zu gewährleisten.
Weil ZigBee mit IEEE 802.15.4 auf eine energiesparende und kostengünstige Kommunikation aufbaut, wird es oft in drahtlosen Sensornetzwerken eingesetzt. Dabei wird auch häufig Mikrocontroller-Technik eingesetzt.

2.4.4 ISA 100.11a

Von der ISA (International Society of Automation) wurde der Standard ISA 100.11a [ISA-100.11a] entwickelt. Dieser Standard ist fokussiert auf den Industrieautomatisierungsbereich und liefert eine drahtlose Kommunikationstechnik zur Prozesssteuerung. Er kommt zum Beispiel bei Stahlwerken zum Einsatz, in denen es oft Probleme mit der Verdrahtung von Sensoren gibt. Durch

die raue Umgebung müssen Kabelstränge oft erneuert werden. Sensoren fallen aus oder liefern verfälschte Werte. Die drahtlose Kommunikation ist in solchen Umgebungen ein Vorteil. Allerdings muss sie verlässlich und sicher sein. Das zu gewährleisten ist Ziel von ISA 100.11a.
Der Standard orientiert sich am OSI-7-Schichtenmodell. In verschiedenen Abschnitten wird die physikalische Schicht (Physical Layer), die Sicherungsschicht (Data Link Layer), die Vermittlungsschicht (Network Layer), die Transportschicht (Transport Layer) und die Anwendungsschicht (Application Layer) definiert. Dabei wird auch auf vorhandene internationale Standards verwiesen. Die unteren beiden Schichten verwenden IEEE 802.15.4, die weiteren beiden Schichten basieren auf 6LoWPAN und UDP [Shelby und Bormann 2009, Seite 167].

2.4.5 Z-Wave

Das Protokoll Z-Wave wurde von der Firma Zensys entwickelt und wird mittlerweile von der Z-Wave Alliance [Web:Z-Wave] gepflegt. Es bietet eine drahtlose Kommunikation zwischen Geräten von unterschiedlichen Herstellern im Bereich der Heimautomatisierung und Gebäudeautomation. Verschiedene Einrichtungen und Geräte können über Z-Wave überwacht und gesteuert werden.
Z-Wave verwendet für die beiden unteren Schichten im OSI-7-Schichtenmodell den Standard ITU-T G.9959 [G.9959]. ITU-T G.9959 verwendet das 900 MHz Frequenzband und ist weniger störanfällig gegenüber WLAN oder IEEE 802.15.4, die beide hauptsächlich 2,4 GHz Frequenzbänder verwenden. Allerdings ist das Z-Wave Protokoll selber kein offenes Protokoll, es ist nur Mitgliedern der Z-Wave Alliance zugänglich.

2.4.6 KNX

Die KNX Association [Web:KNX] ist ein kommerzielles Konsortium aus verschiedenen Firmen, die sich zusammengeschlossen haben, um im Bereich der Heimautomatisierung und der Gebäudeautomation einen einheitlichen Standard zu entwickeln. Der Standard KNX ist auch als offener Standard in IEC 14543-3 spezifiziert. Sein Aufbau orientiert sich am OSI-7-Schichtenmodell. Er ist unterteilt in physikalische Schicht, Sicherungsschicht, Vermittlungsschicht, Transportschicht und Anwendungsschicht. Innerhalb der Anwendungsschicht können dem KNX Gerät Funktionsblöcke zugeordnet werden. Ein Funktionsblock ist dabei eine logische Gruppe von Eingaben, Ausgaben und Parameter, die für die jeweilige Funktionsweise nützlich sind [Hersent et al. 2012, Seite 89].
Es stehen im KNX Standard vier verschiedene Kommunikationsmedien zur Verfügung:

- TP-1 (Twisted Pair): Kommunikationskabel mit verdrillten Adernpaaren. Bandbreite: 9600 bits/s
- PL110 (Powerline): Hausstromnetz als Kommunikationsmedium. Bandbreite: 1200 bits/s
- RF (Radio Frequency): Funkübertragung im 868MHz Frequenzband. Maximale Bandbreite: 16 384 kBit/s
- IP (Ethernet): KNX Telegramme eingekapselt in IP-Paketen, üblicherweise – aber nicht notwendigerweise – Ethernet-basiert

2.4.7 EnOcean

Der Standard EnOcean wurde ursprünglich von der Firma EnOcean GmbH entwickelt. Mittlerweile ist der Standard in zwei Teile geteilt. Die Kommunikationstechnik ist im offenen Standard IEC 14543-3-10 spezifiziert. Die Anwendungsschicht wird von der EnOcean-Alliance in sogenannten EnOcean Equipment Profiles (EEP) gepflegt. Die EnOcean-Alliance ist ein Zusammenschluss von der EnOcean GmbH und vielen weiteren Firmen [Web:enocean].

Das Hauptmerkmal von EnOcean ist der Verzicht auf konventionellen Energiequellen. EnOcean-Geräte beziehen ihre Energie aus sich selber, sei es durch kinetische Energie, Solarenergie oder thermische Energie. Weil mit IEC 14543-3-10 auch eine drahtlose Kommunikationstechnik verwendet wird, kann auf Batterien und jegliche Verkabelung verzichtet werden.
Eingesetzt wird EnOcean hauptsächlich in der Gebäudeautomation, aber vereinzelt auch in anderen Anwendungen.

2.5 Applikationen

Auf ein Smart Object können Benutzer oder Maschinen zugreifen. In diesem Unterkapitel werden Zugriffsmöglichkeiten von Benutzern betrachtet. Es werden Applikationen vorgestellt, wie auf Smart Objects zugegriffen werden kann. Es soll dabei möglich sein, dass Internet als Infrastruktur verwenden zu können.

2.5.1 Webserver

Webserver stellen anderen Teilnehmern innerhalb eines Netzwerkes verschiedene Dienste zur Verfügung. Die Verbindung ist dabei eine klassische Server-Client-Verbindung. Die Clients greifen üblicherweise über einen Webbrowser auf den Webserver zu und rufen Dokumente ab, die dann im Webbrowser dargestellt werden. Zur Datenübertragung wird das Protokoll HTTP (Hypertext Transfer Protocol) [RFC2616] bzw. für verschlüsselte Verbindungen HTTPS (Hypertext Transfer Protocol Secure) verwendet. Die übertragenen Dokumente sind größtenteils in HTML-Format (Hypertext Markup Language), es können aber auch Bilddateien oder andere Daten über HTTP/HTTPS übertragen werden.
Die Webserver-Technologie ist durch das Internet extrem stark verbreitet. Es existieren mehrere Webbrowser-Anwendungen, die teilweise auch betriebssystemübergreifend lauffähig sind. Im Allgemeinem wird vom Webserver ein HTML-Dokument abgerufen, das im Webbrowser dargestellt wird. Der Webbrowser kann auch Informationen zum Webserver senden, der darauf entsprechend reagieren kann.
Webserver im Internet sind normalerweise große Rechnersysteme. Sie benötigen genügend Rechenleistung, um alle Anfragen beantworten zu können. Nichtsdestotrotz existieren auch Webserver-Anwendungen im Mikrocontrollerbereich. Sie sind aufgrund der geringen Ressourcen sehr eingeschränkt. Sie können nur eine geringe Anzahl von Anfragen gleichzeitig beantworten und haben auch nur eine beschränkte Rechenleistung, um serverseitige Verarbeitungen, wie zum Beispiel rechenintensive Skriptsprachen, zu unterstützen.

2.5.2 SNMP-Agent

Ein SNMP-Agent stellt Informationen zur Verfügung, die über das Protokoll SNMP (Simple Network Management Protocol) [RFC1157] abgefragt werden können. Jede Information setzt sich durch eine OID (Object Identifier), einen Datentyp und einen Wert zusammen. Mit SNMP ist es sowohl möglich den Wert einer OID abzufragen als auch zu setzen. Zusätzlich ist es möglich, dass ein geänderter Wert spontan vom SNMP-Agent gesendet wird, sogenannte SNMP-Traps.
Im Standard sind Gruppen von OIDs in verschiedenen MIBs (Management Information Base) strukturiert. Ein SNMP-Agent legt fest, welche MIBs er unterstützen will. Das geht von Standard-MIBs, über MIBs für bestimmte Geräte und Technologien, hin zu herstellerspezifischen MIBs. Verwaltet werden die MIBs von der IETF in RFCs und von der IANA (Internet Assigned Numbers Authority).
SNMP ist ein sehr verbreitetes Protokoll im Bereich Netzwerkmanagement. Nahezu jedes netzwerkfähige Gerät (Switches, Router, Server, Drucker) unterstützt SNMP. Es gibt eine Vielzahl von kommerzieller und freier Software zum Verwalten eines Netzwerkes mithilfe von SNMP.

Beispiele für kommerzielle Software sind HP OpenView von Hewlett-Packard oder Network Performance Monitor von Solarwinds. Für freie Software können Zabbix, Nagios oder MRTG (Multi Router Traffic Grapher) als Beispiele genannt werden. Solche Art Software wird SNMP-Manager genannt.
In der Version SNMPv3 [RFC3414] wurde das Protokoll um eine Verschlüsselung und eine Authentifizierung erweitert. So ist es möglich, SNMP in ungesicherten Netzwerken einzusetzen.

2.5.3 IEC60870-5-104 Slave

Das Protokoll IEC 60870-5-104 [IEC104], kurz IEC104, ist ein Fernwirkprotokoll aus dem Bereich der Energieautomatisierung, das in SCADA (Supervisory Control and Data Aquisition) Systemen weit verbreitet ist. Es dient dazu, Fernwirkgeräte (Remote Terminal Units) über eine TCP/IP-Schnittstelle an eine Leitstelle anzubinden. Das Protokoll unterscheidet dabei zwischen Unterstation – das Fernwirkgerät – und Zentralstation – die Leitstelle. Manchmal wird die Unterstation auch als IEC104-Slave und die Leitstelle als IEC104-Master benannt.
Das Fernwirkgerät liefert zum Beispiel den Zustand von einem Hochleistungsschalter und nimmt von der Leitstelle Befehle entgegen, diesen Schalter zu öffnen oder zu schließen. In der zu erstellenden Lösung soll die Ansteuerung eines Schaltrelais realisiert werden. Dies ist ein vergleichbarer Anwendungsfall.
Die zur Verfügung gestellten Informationen setzen sich aus einer Informationsobjektadresse, einem Datentyp, dem Wert, Qualitätsmarker und gegebenenfalls einem Zeitstempel zusammen. Über das Protokoll IEC104 kann die Änderung einer solchen Information vom Fernwirkgerät spontan gemeldet werden. Informationen können aber auch von der Leitstelle abgefragt oder gesteuert werden.
Außer bei Herstellern von Leitstellen und Herstellern von Fernwirkgeräten in der Energieautomation ist das Protokoll IEC104 nicht sehr verbreitet. Es ist nicht bekannt, ob dieses Protokoll bisher in Smart Objects eingesetzt wurde. Auch ein Einsatz im Zusammenhang mit IPv6 ist bisher nicht bekannt.

2.6 Betriebssysteme für Smart Objects

Allgemein auf Computer bezogen hat ein Betriebssystem zwei markante Aufgaben. Zum einem ist es die Schnittstelle zwischen Mensch bzw. Anwendungssoftware und der darunterliegenden Hardware. Ein Betriebssystem bietet eine leicht verständliche und gut handbare Schnittstelle für den Zugriff auf die eigentliche komplexe Maschine an. Zum anderen hat ein Betriebssystem die Aufgabe, die zur Verfügung stehenden Ressourcen zu verwalten. Die Zuteilung von Prozessoren, Speichereinheiten und Eingabe- und Ausgabe-Geräten zwischen verschiedenen Anwendungen muss sichergestellt werden [Herold 1999, Seite 7f].
Anwendungen oder Programme während der Ausführung werden als Prozesse bezeichnet [Herold 1999, Seite 569]. Um mehrere Prozesse gleichzeitig verarbeiten zu können, benötigt ein Betriebssystem ein Prozessmanagement. Es verwaltet den gleichzeitigen oder nebenläufigen Ablauf der Prozesse und koordiniert den Zugriff auf die unterliegende Hardware.
Dutta und Dunkels [Dutta und Dunkels 2012] beschreiben Anforderungen an Betriebssysteme für drahtlose Sensornetze. Ihre Überlegungen sind ebenso für Smart Objects und teilweise auch für eingebettete System gültig. Im Gegensatz zu herkömmlichen Betriebssystemen, gibt es in diesen Bereichen viele unterschiedliche Prozessorarchitekturen, Kommunikationstechnologien und Sensoren/Aktoren. Dies ist eine der Herausforderung für Betriebsysteme für diese Bereiche.
Dabei stehen nur sehr geringe Ressourcen zur Verfügung. Die Zuteilung dieser wenigen Ressourcen an mehrere Prozesse ist eine weitere Herausforderung für Betriebssysteme dieser Art. Weil ein Betriebssystem selbst auch Ressourcen benötigt, ist es im Bereich von eingebetteten Systemen eine Abwägungssache, ob sich der Einsatz eines Betriebssystems lohnt. Bei Smart Objects ist dies

in der Regel der Fall. Die Programmierung wird – besonders wegen der Netzwerktechnik – um ein Vielfaches erleichtert. Es kann auf vorhandene Anwendungen und Libraries zugegriffen werden und ein Entwickler kann auf eine fertige Prozessverarbeitung aufsetzen. Das Grundgerüst, das bei komplexeren Programmen wie bei Smart Objects unbedingt notwendig ist, ist schon fertig.

Um die Verwendung von TCP/IP zu erleichtern, stellen die meisten Betriebssysteme einen TCP/IP-Stack zur Verfügung. Über Systemaufrufe können Anwendungen vereinfacht TCP/IP-Verbindungen steuern ohne die eigentliche Netzwerkhardware berücksichtigen zu müssen. In der Netzwerkprogrammierung wird die TCP/IP-Verbindung zwischen zwei Endknoten auch eine Socket-Verbindung genannt. Ein Socket ist dabei eine Kombination aus IP-Adresse und TCP-Port von einem Endknoten [Stevens et al. 2004, Seite 52]. Dies ist ein Beispiel, wie ein Betriebssystem den Programmierer von Anwendungssoftware unterstützen kann.

Nachfolgend werden verschiedene Betriebssysteme und Softwareumgebungen vorgestellt, die im Zusammenhang mit Smart Objects zum Einsatz kommen.

2.6.1 Atmel BitCloud

Atmel BitCloud ist weniger ein Betriebssystem als vielmehr eine Entwicklungsumgebung. Sie stellt einen vollständigen, fertigen ZigBee Stack zur Verfügung, um einfach und effektiv Anwendungen für Atmel Mikrocontroller und Funkchips entwickeln zu können. Über eine API kann mittels der Programmiersprache C auf den ZigBee Stack zugegriffen werden. Das reduziert die Komplexität und vereinfacht den Entwicklungsprozess.

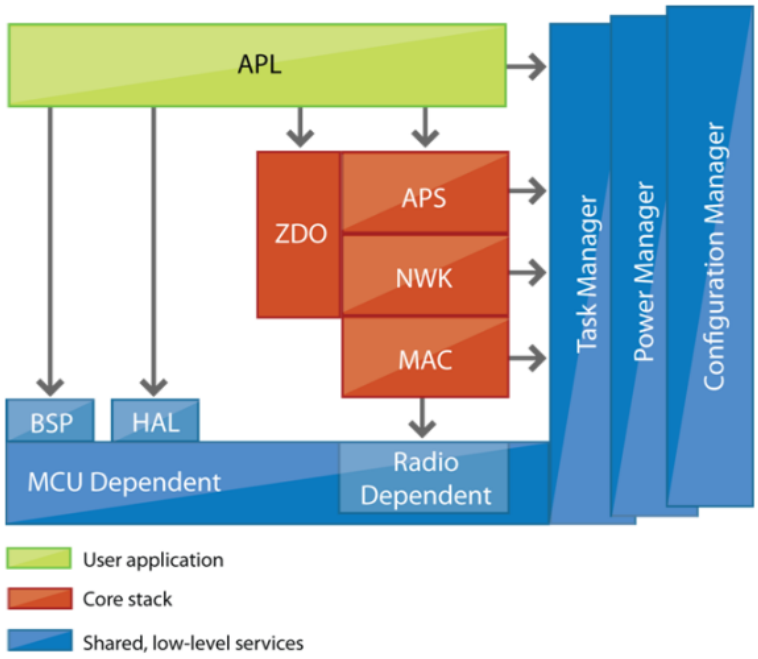

Abbildung 2.5: BitCloud Software Stack Architecture [Atmel BitCloud]

Neben dem ZigBee Stack stellt die Atmel BitCloud API auch weitere Dienste zur Verfügung (siehe Abbildung 2.5). Ein Task Manager koordiniert den Ablauf verschiedener Tasks. Ein Power Manager stellt Routinen zur Verfügung, um geregelt in einem Sleep-Modus zu gehen bzw. aus einem Sleep-Modus aufzuwachen. Dieser Modus reduziert den Energieverbrauch auf ein Minimum. Über HAL (Hardware Abstraction Layer) und BSP (Board Support Package) wird der Zugriff auf die Hardware und die Peripherie vereinheitlicht.

2.6.2 Contiki

Contiki ist ein Betriebssystem, das speziell für kleine netzwerkfähige Geräte entwickelt wurde, die nur über eingeschränkte Ressourcen verfügen. Es wurde unter der Führung von Adam Dunkels [Vasseur und Dunkels 2010, Seite 129f] mit der Idee entwickelt, Geräte an das Internet anzubinden, die so nicht im Internet erwartet werden. In einem Projekt wurde ein Commodore 64, ein 8-Bit Heimcomputer aus dem Jahre 1982, als Webserver ans Internet angeschlossen. Er ist öffentlich unter der Internetadresse http://www.c64web.com erreichbar. Das Haupteinsatzgebiet von Contiki sind aber drahtlose Sensornetzwerke. Als Plattformen werden mehrere Mikrocontroller-Typen, wie der Texas Instruments MSP430 und der Atmel AVR, unterstützt [Dunkels et al. 2004]. Contiki ist in der Programmiersprache C geschrieben. Es unterstützt den Programmierer mit einem zweckdienlichen Kompilierungssystem. Verschiedene fertige Applikationen stehen zur Verfügung und können über das Kompilierungssystem in das zu erstellende Programm eingebunden werden. Außerdem kann auf eine Vielzahl von Systemfunktionen und verschiedene Libraries zurückgegriffen werden.
Als leistungsfähiger TCP/IP Stack wird uIP (micro IP) verwendet. uIP und sein naher Verwandter lwIP (lightweight IP) werden auch in anderen Lösungen, kommerziell und nicht kommerziell, eingesetzt. Es wird besonders darauf geachtet, so wenig Speicher wie möglich zu verbrauchen. Deswegen ist nur das absolute Minimum an Funktionen implementiert, die zum Betrieb eines vollständigen TCP/IP Stacks notwendig sind. Der Speicherverbrauch des Programmcodes von uIP auf einer Atmel AVR Plattform beträgt 5164 Bytes [Dunkels 2003]. Im Jahr 2008 wurde aufbauend auf dem uIP Stack der uIPv6 Stack veröffentlicht. Er benötigt auf der Atmel AVR Plattform 11,5 KBytes für seinen Programmcode und 1,8 KBytes RAM [Durvy et al. 2008]. In Contiki stehen beide Varianten zur Verfügung.
Contiki basiert auf einem ereignisgesteuerten Programmierungsmodell (Zustandsautomat). Im Gegensatz dazu bieten viele Betriebssysteme, die in nicht ressourcenbeschränkten Umgebungen eingesetzt werden, die Möglichkeit zur Verwendung von Prozessen und Threads. Als Thread ist dabei ein zusammenhängender Kontext von Programmieranweisungen zu verstehen. Ein Prozess besteht mindestens aus einem Thread, aber es können innerhalb eines Prozesses mehrere Threads existieren. Um diese parallel ausführen zu können, müssen die Daten eines Threads im Betriebssystem gespeichert werden. Dies geschieht über eine Stackverwaltung, die wiederum Speicher-Overhead erzeugt. Deswegen wird die Thread-Programmierung in ressourcenbeschränkten Umgebungen kaum verwendet. Allerdings ist dieser Stil der Programmierung einfacher als beim ereignisgesteuerten Programmierstil. Mit seinen Protothreads [Dunkels und Schmidt 2005]; [Dunkels et al. 2006] kombiniert Contiki die Vorteile von beiden Stilen. Protothreads bestehen aus Präprozessor-Makros, die mehrere Threads ereignisgesteuert über eine Switch-Anweisung ausführen. Programmiert werden sie ähnlich der Thread-Programmierung. Sie verzichten auf eine Stackverwaltung, es werden allerdings 2 Byte Speicher-Overhead je Thread benötigt. Dafür kann die Anzahl der Zeilen Quellcode um ein Drittel reduziert werden.
Auf Basis der Protothreads bietet Contiki eine Netzwerkprogrammierung mit Protosockets. Sie werden verwendet, um über ein Netzwerk empfangene Daten zu lesen oder um ausgehende Daten zu schreiben. Sie haben die Eigenschaften eines Protothreads mit zusätzlichen Funktionen zum Empfangen und Senden von Daten. Protosockets bestehen ebenfalls aus Präprozessor-Makros, die eine Programmierung ähnlich der BSD Socketprogrammierung erlauben. Im Hintergrund läuft aber wie bei den Protothreads ein Zustandsautomat.

2.6.3 TinyOS

TinyOS [Hill et al. 2000] ist ein quelloffenes BSD-lizenziertes Betriebssystem, was hauptsächlich in drahtlosen Sensornetzwerken eingesetzt wird. Es wird gepflegt von verschiedenen Arbeitsgruppen innerhalb der TinyOS Community [Web:tinyos]. Das Design von TinyOS wurde durch vier Voraussetzungen beeinflusst [Levis 2006]:

- begrenzte Ressourcen, wie Speicher und Rechenleistung:
  eine typische TinyOS Anwendung benötigt etwa 15 KBytes, wovon 400 Bytes vom Betriebssystem verwendet werden.

- Management von gleichzeitigen Prozessaufgaben (Concurrency Management):
  TinyOS verwendet dazu ein ereignisgesteuertes Modell, um auf eine Stack-Speicherung verzichten zu können.

- Flexibilität in Bezug auf unterstützte Hardware und Anwendungen:
  TinyOS baut auf eine komponentenbasierte Architektur. Dabei können einzelne Komponenten selektiv ausgetauscht werden, um neue Funktionen oder Hardware zu unterstützen.

- geringer Energiebedarf:
  batteriebetriebene Geräte sollen möglichst lange wartungsfrei laufen. Der TinyOS Scheduler, Teil des Betriebssystems, kann den Prozessor automatisch in einen Sleep-Modus setzen, wenn keine Prozessaufgaben anstehen.

TinyOS verwendet die selbst entwickelte, an C angelehnte Programmiersprache NesC. Die TinyOS Community verwaltet Mailinglisten, ein Wiki und unterstützt Entwickler mit verschiedenen Handbüchern. Als Protokollstack wird der Berkeley Low-power IP stack (BLIP) verwendet.

2.6.4 FreeRTOS

FreeRTOS [Web:freertos] ist ein quelloffenes Echtzeitbetriebssystem für eingebettete Systeme (embedded systems). Echtzeitbetriebssystem bedeutet, dass das Zeitverhalten von Prozessen vorhergesagt werden kann. Eine Anwendung, die auf FreeRTOS läuft, kann exakt bestimmen, wann eine Anweisung ausgeführt werden soll. Contiki und TinyOS unterstützen dies nicht in dieser Form.
FreeRTOS ist hauptsächlich in der Programmiersprache C implementiert worden, kleinere Teile sind in Assembler geschrieben. Insgesamt werden über 30 Plattformen unterstützt.
Da FreeRTOS eher ein Kernel ist als ein ganzes Betriebssystem, wird TCP/IP nicht direkt unterstützt. Allerdings gibt es mehrere Beispielprojekte, enthalten im FreeRTOS Quellcode, die beide TCP/IP-Stacks uIP und lwIP verwenden [Vasseur und Dunkels 2010, Seite 131f].

2.7 Herausforderungen

Nachdem verschiedene Technologien vorgestellt worden sind, werden jetzt allgemein Herausforderungen betrachtet, die bei der Entwicklung von Smart Objects auftreten können. In [Vasseur und Dunkels 2010] sind diese Herausforderungen in zwei Kategorien eingeteilt. Zum einen sind technische Herausforderungen definiert. Sie behandeln die technischen Eigenschaften von Smart Objects, die Art und Weise, wie sie über ein Netzwerk miteinander kommunizieren und die Anforderungen an die Entwicklung von Smart Objects. Zum anderen gibt es aber auch nicht-technische Herausforderungen, die im darauf folgenden Abschnitt behandelt werden.

2.7.1 Technische Herausforderungen

Die drei markanten technischen Herausforderungen eines Smart Objects sind

- Energieverbrauch

- physische Größe

- Kosten

Der Energieverbrauch sollte möglichst gering sein. Oft wird ein Smart Object drahtlos verwendet. Dann kann als Energiequelle nur die Selbstversorgung, zum Beispiel durch Sonnenenergie oder kinetische Energie, oder eine gespeicherte Energieform, klassisch die Batterie, verwendet werden. Durch Reduzierung des eigenen Energieverbrauchs sinken die Ansprüche – und damit die Komplexität und Kosten – an eine Selbstversorung. Beim Einsatz einer Batterie kann durch Energieeinsparungen eine höhere Laufzeit erreicht werden bzw. es können kleinere Batterien eingesetzt werden.

Smart Objects werden sich durchsetzen, wenn sie in ihrer Umwelt nicht oder zumindest kaum auffallen. Für viele Anwendungen muss ein Smart Object in einem vorhandenen Gerät, sei es eine Steckdose, eine Waschmaschine oder ein Messgerät, eingebaut werden. Dafür muss ein Smart Object möglichst klein sein.

Die Kosten eines Smart Objects sind entscheidend. Es wird nur dann eingesetzt werden, wenn der Mehrnutzen, den man durch ein Smart Object erhält, größer ist als die Kosten, die für ein Smart Object aufgebracht werden müssen. Nutzen und Kosten sind dabei hauptsächlich aber nicht nur finanzieller Natur. Komfort zum Beispiel ist auch ein Nutzen, der in diese Rechnung einfließt. Es bedeutet aber, dass Smart Objects sehr günstig, sowohl in der Herstellung als auch während des Betriebs, sein müssen.

Um diese drei Herausforderungen zu meistern, müssen Hardware- und Software-Designer eng zusammen arbeiten. Der Energieverbrauch zum Beispiel lässt sich durch den Einsatz der richtigen Bauteile reduzieren, aber auch dadurch, dass der Mikrocontroller oder die Kommunikationseinheit so oft wie möglich von der Software in einen Sleep-Modus gesetzt wird. Ein Software-Designer muss das bei seiner Implementierung berücksichtigen. Ebenfalls müssen die limitierten Ressourcen beachtet werden. Ein Mikrocontroller besitzt meistens nur wenige Kilobytes an Speicher und hat auch nur eine begrenzte Rechenleistung. Ein Software-Designer muss seine Implementierung speicherschonend, effizient und energiesparend anlegen.

Eine besondere Anforderung wird auch an die Kommunikationstechnologie gestellt. Sie ist ein markanter Baustein eines Smart Objects und hat einen großen Einfluss auf Energieverbrauch, physikalische Größe und Kosten. In der Regel hat die Kommunikationseinheit den größten Anteil am Energieverbrauch. Eine Antenne bei drahtloser Kommunikation oder ein Steckverbinder bei drahtgebundener Kommunikation benötigt Platz auf dem Smart Object. Die Bauteile der Kommunikationseinheit machen einen Teil der Hardwarekosten aus. Andere Herausforderungen an die Kommunikationstechnologie betreffen das Routing. Netzwerke mit Smart Objects können potentiell eine Vielzahl von Teilnehmern haben. Wie kann das Kommunikationsmedium effizient genutzt werden, um die Bandbreite und die Reichweite des Netzwerkes möglichst hoch zu halten? Dies ist noch schwieriger bei unzuverlässigen Netzwerken, wie sie auch von Smart Objects verwendet werden können.

Um Kosten niedrig zu halten und die Akzeptanz einer Lösung zu gewährleisten, müssen Smart Objects ohne großen Aufwand betrieben werden können. Das bedeutet, dass sich Smart Objects mit geringen Maßnahmen in eine bestehende Infrastruktur integrieren lassen. Das betrifft den physikalischen Anschluss an ein bestehendes Netzwerk, der bei drahtloser Kommunikation quasi wegfällt, aber auch die Konfiguration des Smart Objects, die idealerweise ganz automatisch passiert. Als Beispiel für eine gute Lösung kann WLAN genannt werden. Mittlerweile zeichnet sich diese Technologie mit einer auch für Laien einfachen Installation aus. Ein neues Gerät kann in der Regel mit wenig Aufwand in ein bestehendes WLAN Netzwerk integriert werden. Dennoch ist eine manuelle Interaktion erforderlich, üblicherweise muss mindestens ein Passwort bzw. ein WLAN-Schlüssel konfiguriert werden. Bei Smart Objects besteht aber nicht immer diese Möglichkeit.

Smart Objects sollen unter anderem Daten sammeln und an externe Geräte weitergeben, aber sie sollen auch Steuerbefehle entgegen nehmen können. Es müssen Methoden erarbeitet werden, wie von außen auf Smart Objects zugegriffen werden kann und wie Daten abgefragt werden können. Dabei ist die Sicherheit auch eine große Herausforderung. Wie kann ein Smart Object wissen, ob ein Zugriff berechtigt ist und wie kann dieser Zugriff von unerlaubtem Mithören geschützt

werden? Authentifizierung und Verschlüsselung sind zwei markante Herausforderungen, die auch sehr wichtig für die Akzeptanz einer Smart Object Lösung sein werden.

2.7.2 Nicht-technische Herausforderungen

Neben den vorgestellten Technologien gibt es zur Zeit sehr viele proprietäre Lösungen für Smart Object Anwendungen, besonders im Bereich der Heimautomatisierung. Der große Nachteil an solchen Lösungen ist, dass sie nicht kompatibel zueinander sind. Geräte von einer Lösung können nicht mit Geräten einer anderen Lösung kombiniert werden.
Eine große nicht-technische Herausforderung an eine Smart Object Lösung ist die Interoperabilität. Dabei muss die Interoperabilität zwischen verschiedenen Herstellern als auch die Interoperabilität mit existierenden Lösungen bzw. existierender Infrastruktur betrachtet werden. Um eine Interoperabilität überhaupt gewährleisten zu können, wird eine gute Standardisierung benötigt. Mit einer standardisierten Lösung, idealerweise durch einen offenen Standard, wird die Technologie unabhängig vom Hersteller und lässt sich leichter kombinieren. Allerdings verwenden Smart Objects Technologien auf ganz verschiedenen Ebenen, angefangen von der Kommunikationstechnologie bis hin zu der Frage, wie auf ein Smart Object zugegriffen wird. Jede Ebene hat ihre eigenen Anforderungen, die auch je nach Anwendung sehr vielschichtig sind. Umso wichtiger ist eine durchgängige Standardisierung auf allen Ebenen.
Eine Standardisierung ist besonders wichtig, wenn es um die Integration in eine große Infrastruktur geht. Dies zeigt das Beispiel von IPv6 und dem Internet deutlich. Der Standard von IPv6 wurde bereits im Jahr 1998 als IETF Draft Standard veröffentlicht [Hagen 2009, Seite 4]. Trotzdem ist der Anteil an IPv6 im Internet nach Messungen von [Web:google] immer noch unter 2%. Große Infrastrukturen wie das Internet sind sehr träge und benötigen eine lange Zeit für tiefe Veränderungen. In so einem Umfeld ist es umso wichtiger, Standardisierungen so früh wie möglich und vorausschauend wie möglich vorzubereiten, um die Akzeptanz für eine Lösung zu gewährleisten. Dazu gehört auch, solche Lösungen zu verbreiten. Eine Organisation, die sich zum Beispiel für die Nutzung des Internet Protokoll für Smart Objects einsetzt ist die IPSO Alliance [Dunkels und Vasseur 2008].

Kapitel 3

Auswahl der Technologien

Die vorgestellten Technologien sollen nun methodisch verglichen werden. Dies geschieht unter dem Aspekt, dass nach dem Vergleich eine Lösung beispielhaft implementiert werden soll. Zuerst wird dabei die zu implementierende Lösung als Produkt begriffen. Eine weit verbreitete Methode, den Einfluss von Produktmerkmalen auf die Kundenzufriedenheit festzustellen, ist das Anforderungsmodell nach Kano. Diese Methode wird im ersten Abschnitt angewandt. Die verschiedenen Technologien werden hier noch nicht mit betrachtet.
Da die verschiedenen Technologien aufeinander aufbauen, manchmal zueinander kompatibel sind, manchmal inkompatibel sind, werden nachfolgend mögliche Komplettlösungen hergeleitet. Sie werden zuerst kurz vorgestellt, miteinander verglichen und bewertet. Um einen systematischen Entscheidungsprozess zu gewährleisten, werden die möglichen Komplettlösungen mithilfe der Nutzwertanalyse verglichen und bewertet. Der Vorteil der Nutzwertanalyse ist es, dass die Bewertung nachvollziehbar und überprüfbar ist. Nach der Auswertung wird die Lösung ausgesucht, die implementiert werden soll.

3.1 Anforderungsbeschreibung

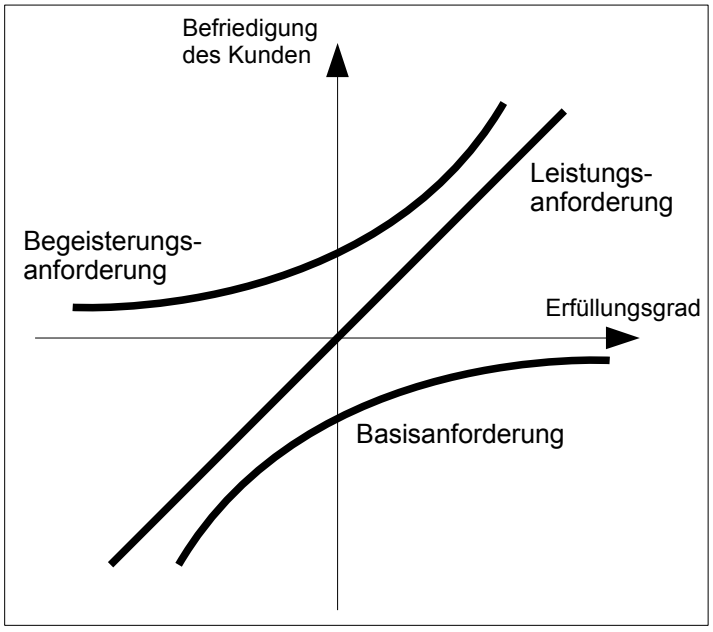

Abbildung 3.1: Kano-Modell

Die allgemeine Aufgabenbeschreibung befindet sich in Kapitel 1.1. In diesem Kapitel werden dagegen einige Anforderungen an eine mögliche Implementierung definiert. Die Anforderungen werden nach dem Kano-Modell zur Kundenzufriedenheit [Kano et al. 1984] eingeordnet (siehe Abbildung 3.1).
Dieses Modell wird dazu verwendet, die Kundenzufriedenheit in Bezug auf ein Produkt zu analysieren. Dabei werden Produktanforderungen in drei Kategorien eingeteilt:

- Basisanforderungen
- Leistungsanforderungen
- Begeisterungsanforderungen

Anhand des Erfüllungsgrades der einzelnen Anforderungsmerkmale lässt sich die Kundenzufriedenheit ablesen. Vor der Formulierung der Anforderungen wird erläutert, wie die Begriffe Produkt und Kunde in unserem Fall verstanden werden. Das Produkt ist die zu implementierende Steuerung nach der Aufgabenbeschreibung in Kapitel 1.1. Da sie nicht kommerziell vertrieben werden soll, gibt es keinen Kunden im klassischen Sinne. Als Kunde oder Zielgruppe werden private Anwender von Heimautomatisierungssystemen gesehen, aber auch Entwickler von Produkten dieser Art.
Nachfolgend werden Anforderungen der drei Kategorien nach eigenen Annahmen definiert.

3.1.1 Basisanforderungen

Basisanforderungen beschreiben die minimalen Anforderungen an ein Produkt. Das Fehlen einer dieser Anforderungen führt zu Unzufriedenheit beim Kunden. Allerdings kann sich ein Produkt anhand von Basismerkmalen kaum gegenüber anderen Vergleichsprodukten differenzieren.
Basisanforderung der zu entwickelnden Steuerung sind:

- Der Zugriff auf die Steuerung soll über das Internet und über ein lokales Netzwerk möglich sein.
- Um zukunftssicher zu sein, soll der Zugriff über IPv6 erfolgen.
- Es sollen keine proprietären Technologien bzw. Protokolle verwendet werden.
- Für den Netzwerkzugriff soll keine zusätzliche Verkabelung notwendig sein. Der Netzwerkzugriff soll drahtlos erfolgen.
- Für die Stromversorgung der Steuerung soll kein zusätzlicher Anschluss gebraucht werden. Sie soll über das angeschlossene Stromnetz erfolgen.
- Der Netzwerkzugriff soll über eine Funkübertragung mit einer Reichweite von 5 Metern erfolgen.
- Die Leistungsaufnahme der Steuerung soll weniger als 1 Watt betragen.
- Der Zugriff soll über eine Benutzerschnittstelle erfolgen (Mensch-Maschine-Schnittstelle).
- Die Steuerung soll ohne Umparametrierung auch an anderen Orten betreibbar sein.

3.1.2 Leistungsanforderungen

Leistungsanforderungen sind Anforderungen, die ein Kunde bewusst an ein Produkt stellt. Je nach Erfüllung dieser Anforderungen wird Unzufriedenheit beseitigt oder Zufriedenheit geschaffen. Über Leistungsanforderungen kann sich ein Produkt gegenüber Vergleichsprodukten differenzieren.
Leistungsanforderungen der zu entwickelnden Steuerung sind:

- Der Netzwerkzugriff soll über eine Funkübertragung mit einer Reichweite von 10 Metern erfolgen.
- Die Leistungsaufnahme der Steuerung soll weniger als 100 Milliwatt betragen.
- Der Zugriff auf die Steuerung soll über Standard-Software bzw. Standard-Applikationsprotokollen erfolgen.
- Die Bedienung soll einfach und intuitiv sein.
- Der Zugriff soll zusätzlich zu einer Benutzerschnittstelle auch automatisch erfolgen können (M2M).
- Die Materialkosten sollen weniger als 50 Euro betragen (Einzelstückpreise).
- Um Entwicklungskosten gering zu halten, soll auf vorhandene frei verfügbare Software aufgebaut werden. Es sollen nur kleine Erweiterungen bzw. Anpassungen notwendig sein. Für die Implementierung sollen weniger als 1000 Zeilen Quellcode (LOC) selbst programmiert werden.
- Die verwendete Kommunikationstechnologie soll einfach in ein bestehendes Heim-Netzwerk einzubinden sein.

3.1.3 Begeisterungsanforderungen

Begeisterungsanforderungen sind Produktmerkmale, mit denen der Kunde nicht rechnet. Über diese Merkmale kann primär eine Differenzierung zu Vergleichsprodukten erfolgen.

Begeisterungsanforderung der zu entwickelnden Steuerung sind:

- Weitere Applikationen bzw. Zugriffsmöglichkeiten sollen einfach zu entwickeln sein bzw. schon in der verwendeten Software verfügbar sein.
- Die Materialkosten sollen weniger als 10 Euro betragen.
- Verschiedene Zugriffsmöglichkeiten sollen parallel möglich sein.

3.2 Herleitung möglicher Komplettlösungen

Um mögliche Komplettlösungen zu erstellen, werden zunächst die Abhängigkeiten der vorgestellten Übertragungstechniken und Kommunikationstechnologien in Abbildung 3.2 dargestellt. Dabei befinden sich die Technologien, die sich näher an der physikalischen Schicht befinden, weiter unten und die Technologien, die sich näher an der Applikationsschicht befinden, oben. Die Abbildung zeigt zum Beispiel, dass ZigBee und 6LoWPAN auf der selben Übertragungstechnik, nämlich IEEE 802.15.4, aufbauen.
Im weiteren Verlauf werden die möglichen Komplettlösungen als eine Kombination von Kommunikationstechnologie, Applikation und Betriebsystem beschrieben. Mögliche Applikationen wurden bereits in Kapitel 2.5 beschrieben, mögliche Betriebsysteme in Kapitel 2.6.

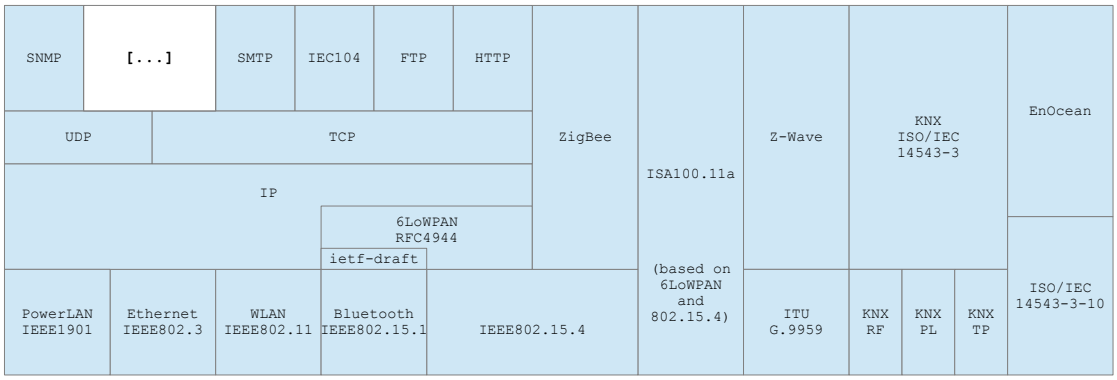

Abbildung 3.2: Kommunikationstechnologien

3.2.1 6LoWPAN + HTTP + Contiki

Als Kommunikationstechnologie soll hier 6LoWPAN verwendet werden. Als Betriebssystem wird Contiki gewählt, weil hier bereits eine verbreitete Implementierung von 6LoWPAN vorhanden ist. Die Applikation soll ein Webserver sein, auf den über das Protokoll HTTP von jedem beliebigen Webbrowser zugegriffen werden kann. Eine Webserver-Implementierung ist ebenfalls schon in Contiki vorhanden – diese muss allerdings angepasst werden.
Als Hardware kann zum Beispiel von Texas Instruments ein MSP430 Mikrocontroller [MSP430] mit einem CC2420 Funkchip [CC2420] oder von Atmel ein AVR Mikrocontroller [Web:Atmel AVR] mit einem RF230 Funkchip [AT86RF230] verwendet werden.

3.2.2 6LoWPAN + SNMP + Contiki

Diese Lösung ist identisch mit der vorherigen. Allerdings wird als Applikation ein SNMP-Agent statt eines Webservers verwendet. Eine SNMP-Implementierung ist nicht Bestandteil von Contiki, allerdings gibt es bereits eine Implementierung, auf die aufgesetzt werden kann [Kuryla und Schönwälder 2011].

3.2.3 WLAN + HTTP + Contiki

Diese Lösung ist ähnlich der ersten Lösung. Allerdings soll als Kommunikationstechnik nicht 6LoWPAN, sondern WLAN nach IEEE 802.11 verwendet werden. Um diesen Standard verwenden zu können, muss ein anderer Funkchip eingesetzt werden. Zusätzlich sind Anpassungen notwendig, weil Contiki diesen Standard nicht ohne Weiteres unterstützt. Eine Recherche der Contiki-Developers-Mailing-Liste [Web:contiki-developers, 802.11 wifi (Feb 2012)] ergab, dass es zwei Möglichkeiten gibt, diesen Standard in Contiki zu realisieren. Entweder man verwendet Funkchips, die einen integrierten MAC Layer verwenden. Diese sind preislich teuer, aber die Anbindung an Contiki ist einfacher zu realisieren. Oder man verwendet Funkchips, bei denen der MAC-Layer in der Software der Steuereinheit realisiert werden muss. Diese sind preislich günstiger, allerdings ist dann der Anpassungsaufwand im Contiki größer. Es ist ein Nachteil, dass es nicht viel Erfahrung mit diesem Standard in Verbindung mit Contiki gibt.

3.2.4 6LoWPAN + HTTP + TinyOS

Im Gegensatz zu den vorherigen Lösungen wird hier das Betriebssystem TinyOS verwendet. Ansonsten wird wie bei der ersten Lösung die Kommunikationstechnologie 6LoWPAN und das Applikationsprotokoll HTTP verwendet. 6LoWPAN wird von TinyOS seit der Version 2.1.1 unter-

stützt. Für das Protokoll HTTP gibt es bereits mehrere Beispielimplementierungen eines Webservers in TinyOS.

3.2.5 ZigBee + HTTP + BitCloud

Diese Lösung verwendet als Kommunikationstechnologie ZigBee. ZigBee ist keine reine Kommunikationstechnologie, sondern beschreibt als quasi-Standard alle Schichten bis hoch zu verschiedenen Applikationsprofilen. Das Betriebssystem bzw. die Softwareimplementierung, auf die aufgesetzt werden kann, kann selbst entwickelt sein oder vom Hersteller als API zur Verfügung gestellt werden. BitCloud ist eine Implementierung von Atmel, die mit verschiedenen Geräten kompatibel ist. Als Hardware kann zum Beispiel das Modul ATZB-24-A2 [ATZB-24-A2] verwendet werden. Um vom Internet aus auf ein ZigBee-Modul zugreifen zu können, muss ein Gateway zwischengeschaltet werden. Der Webserver läuft nicht nativ auf dem Modul, sondern auf dem Gateway. Ein Beispiel für solch einen Gateway ist der Z-202 [Web:Z-202] von der Firma Netvox.

3.3 Nutzwertanalyse möglicher Komplettlösungen

Der Vergleich zwischen den vorher ausgewählten Komplettlösungen wird anhand einer Nutzwertanalyse erstellt. Nach der Definition von [Zangemeister 1976, Seite 45] ist die Nutzwertanalyse „die Analyse einer Menge komplexer Handlungsalternativen mit dem Zweck, die Elemente dieser Menge entsprechend den Präferenzen des Entscheidungsträgers bezüglich eines multidimensionalen Zielsystems zu ordnen. Die Abbildung dieser Ordnung erfolgt durch die Angabe der Nutzwerte (Gesamtwerte) der Alternativen."
Die Nutzwertanalyse dient dazu verschiedene Entscheidungsalternativen einzuordnen, also zu bewerten. Dies geschieht durch die Aufstellung mehrerer Zielkriterien, die dann gewichtet werden. Für jedes Zielkriterium zusammen mit jeder Alternative wird ermittelt, in wie weit das Kriterium von der Alternative erfüllt wird. So werden die einzelnen Teilnutzwerte ermittelt (multidimensionales Zielsystem). Der gesamte Nutzwert einer Alternative ermittelt sich durch die Summe aus den jeweiligen Produkten von Teilnutzwert und Gewichtung. Bei dem Vergleich der Alternativen wird nur der ermittelte Nutzwert betrachtet. So können Defizite in einzelnen Zielkriterien durch eine Übererfüllung von anderen Kriterien kompensiert werden.

Mathematisch kann das folgendermaßen ausgedrückt werden:

g_k	Gewichtung des Kriteriums k
t_{ak}	Teilnutzwert der Alternative a für das Kriterium k
N_a	Nutzwert der Alternative a

Jedes Kriterium wird gewichtet zu g_k. Die Summe der Gewichtung aller Kriterien soll 1 bzw. 100% betragen. Bei einer Anzahl von n Kriterien bedeutet dies:

$$\sum_{k=1}^{n} g_k = 1 \tag{3.1}$$

Von jedem Kriterium wird für jede Alternative ein Teilnutzwert t_{ak} ermittelt. Durch die Summierung des Produktes aus Teilnutzwert t_{ak} und Gewichtung g_k ergibt sich der Nutzwert einer Alternative N_a.

$$N_a = \sum_{k=1}^{n} t_{ak} * g_k \tag{3.2}$$

Als nächsten Schritt werden in Kapitel 3.3.1 Kriterien definiert nach denen die vorher hergeleiteten Komplettlösungen bewertet werden sollen. Die einzelnen Kriterien werden in Kapitel 3.3.2

gewichtet. In Kapitel 3.3.3 wird für jedes Kriterium ein Teilnutzwert für jede Komplettlösung ermittelt. Daraus wird der jeweilige Nutzwert einer möglichen Lösung berechnet und miteinander verglichen.

3.3.1 Definition der Kriterien

Auf Basis der vorher definierten Anforderungen werden Kriterien definiert, nach denen die Komplettlösungen bewertet werden sollen. Die Kriterien beziehen sich auf die Kommunikationstechnologie, die verwendete Applikation oder das verwendete Betriebssystem beziehungsweise die verwendete Software.

- **Kommunikationstechnologie**

  1. **Reichweite der Funkübertragung**
     Die Reichweite besagt, in welchem Umkreis die Steuerung vom Empfangsgerät entfernt platziert werden kann.
  2. **typische Leistungsaufnahme zum Senden und Empfangen**
     Die Steuerung soll unter anderem dazu beitragen, den Energieverbrauch im Haushalt zu optimieren. Dazu muss durch sie mehr Energie eingespart werden, als sie selbst verbraucht. Ausschlaggebend für die Leistungsaufnahme der gesamten Steuerung ist die verwendete Kommunikationstechnologie.
  3. **Verbreitung der Technologie (Interoperabilität)**
     Eine weite Verbreitung der Kommunikationstechnologie wird der Steuerung helfen, besser und schneller am Markt akzeptiert zu werden. Hinzu kommt, dass es dadurch mehr Erfahrungen im Umgang mit der Technologie gibt. Dies hat auch einen Einfluss auf die Interoperabilität mit anderen Technologien.
  4. **Integrationsaufwand in eine existierende Infrastruktur**
     Die Steuerung soll in eine bestehende Infrastruktur (nämlich das jeweile Haus-Netzwerk bzw. das Internet) eingebunden werden. Ein typisches Haus-Netzwerk hat einen Internetanschluß über DSL, Kabel oder ähnliches. Endgeräte (PCs, Laptops, u.a.) sind in der Regel über ein Ethernet- oder ein WLAN-Netzwerk angeschlossen. Abgeschätzt werden muss der Aufwand, der notwendig ist, die für die Steuerung verwendete Kommunikationstechnologie in dieses Haus-Netzwerk einzubinden.
  5. **Implementierungsaufwand**
     Es soll der Aufwand abgeschätzt werden, der notwendig ist, die verwendete Kommunikationstechnologie zu implementieren. Im günstigsten Fall wird die Technologie bereits von der verwendeten Software unterstützt.
  6. **Unterstützung von Applikationen**
     Es soll bewertet werden, ob – außer der verwendeten Applikation – auch noch andere Applikationen von der Kommunikationstechnologie unterstützt werden. Dies ist ein Indikator für die Akzeptanz der Kommunikationstechnologie. Außerdem kann die Steuerung dann auch einfacher für andere Bereiche angepasst werden, in denen weitere Applikationen Verwendung finden.
  7. **typische Hardwarekosten**
     Die Kosten sollen möglichst gering sein. Im Idealfall betragen die Kosten für die gesamte Steuerung weniger als mit ihr – über ihre Lebenszeit gesehen – an Leistungsverbrauch eingespart werden kann. Bei einer sehr großen Stückzahl geht der Entwicklungsaufwand gegen Null. Die Hardwarekosten je Steuerung bleiben aber relativ konstant.

- **Applikation**

  8. **Verbreitung möglicher Zugriffssoftware**
     Der Zugriff auf die Steuerung läuft über Software. Diese Software muss auf dem Gerät installiert sein, von dem aus auf die Steuerung zugegriffen wird. Umso verbreiteter die notwendige Software ist, umso höher die Akzeptanz der Applikation. Der Idealfall wäre ein Zugriff über einen Webbrowser, da fast jedes internetfähige Gerät einen Webbrowser besitzt.
  9. **Möglichkeit für einen manuellen Zugriff (Benutzerschnittstelle)**
     Mit diesem Kriterium soll die Benutzerschnittstelle bewertet werden. Der manuelle Zugriff soll für den Benutzer möglichst einfach und intuitiv sein.
  10. **Möglichkeit für einen automatischen Zugriff (M2M)**
      Wenn die Applikation einen automatischen Zugriff zulässt, soll dieser auch in die Bewertung mit einfließen.
  11. **Implementierungsaufwand der Applikation auf dem gewählten Betriebssystem**
      Der Aufwand für die Implementierung der Applikation auf dem gewählten Betriebssystem soll abgeschätzt werden. Im Idealfall wird die Applikation schon vom Betriebssystem unterstützt und es sind nur kleinere Anpassungen notwendig.
  12. **Anforderung an Kommunikationstechnologie**
      Die Anforderungen der Applikation an die Kommunikationstechnologie sollen bewertet werden. Als Anhaltspunkt soll dabei die die notwendige Bandbreite betrachtet werden.

- **Betriebssystem**

  13. **Lizenz**
      Eine frei verfügbare Software ist günstiger als eine kostenpflichtige Software. Eventuell müsste eine Lizenz auf den Stückpreis hinzugerechnet werden. Es soll auch betrachtet werden, ob eine kommerzielle Nutzung ohne weiteres möglich ist oder nicht.
  14. **Verbreitung des Betriebssystems**
      Eine weite Verbreitung des Betriebssystems bedeutet in der Regel, dass es eine Entwicklergemeinde gibt, die sich über mögliche Probleme austauscht. Das weist darauf hin, dass das Betriebssystem in verschiedenen Umgebungen eingesetzt und so Erfahrungen gesammelt wurden, die wieder in neue Versionen eingeflossen sind.
  15. **bereits unterstützte Applikationen**
      Bewertet werden soll hier die Anzahl und Art bereits unterstützter weiterer Applikationen. Dadurch wird es einfacher, die Steuerung in anderen Bereichen einzusetzen.
  16. **Aufwand der Implementierung neuer Applikationen**
      Hier soll der Aufwand abgeschätzt werden, der benötigt wird, um eine neue Applikation zu implementieren.
  17. **Unterstützung verschiedener Kommunikationstechnologien**
      Es sollen möglichst mehrere Kommunikationstechnologien unterstützt werden. So ist es einfacher, die Steuerung für weitere Einsatzmöglichkeiten anzupassen.

3.3.2 Gewichtung der Kriterien

Manche Kriterien sind besonders wichtig, andere sind nicht sehr ausschlaggebend für das Ergebnis. Um das zu berücksichtigen, werden die Kriterien nach eigenen Annahmen gewichtet. Um eine möglichst ausgewogene Gewichtung zu bekommen, wird jedes Kriterium mit jedem anderen verglichen und abgewogen, welches wichtiger ist. Nach folgenden Punktesystem wird bewertet:

- 2 = wichtiger
- 1 = gleich wichtig
- 0 = unwichtiger

		Kommunikations-technologie							Applikation					Betriebssystem				
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Kommunikations-technologie	1		0	1	1	1	1	0	0	0	1	0	1	1	1	1	1	0
	2	2		1	1	2	2	1	1	1	2	1	2	2	1	1	1	1
	3	1	1		1	1	1	1	1	1	2	1	2	1	1	1	1	1
	4	1	1	1		1	1	1	0	1	1	1	1	2	1	1	1	2
	5	1	0	1	1		1	1	0	1	1	1	1	2	1	1	1	1
	6	1	0	1	1	1		1	0	1	1	1	1	1	1	1	1	1
	7	2	1	1	1	1	1		0	1	1	1	1	2	1	1	1	1
Applikation	8	2	1	1	2	2	2	2		1	1	1	1	2	1	2	2	1
	9	2	1	1	1	1	1	1	1		1	1	1	2	1	1	1	1
	10	1	0	0	1	1	1	1	1	1		1	1	1	1	1	1	1
	11	2	1	1	1	1	1	1	1	1	1		1	1	1	1	1	1
	12	1	0	0	1	1	1	1	1	1	1	1		1	1	1	1	1
Betriebssystem	13	1	0	1	0	0	1	0	0	0	1	1	1		1	1	1	1
	14	1	1	1	1	1	1	1	1	1	1	1	1	1		1	1	1
	15	1	1	1	1	1	1	1	0	1	1	1	1	1	1		1	1
	16	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1		1
	17	2	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	

Abbildung 3.3: Gewichtung der Kriterien

Abbildung 3.3 zeigt die Bewertung nach dem Punktesystem dargestellt in einer Matrix. Die einzelnen Punkte je Kriterium werden zeilenweise zusammengezählt. Die einzelnen Summen werden jeweils durch die Gesamtpunktzahl aller Kriterien (272) dividiert. Die Ergebnisse ergeben die Gewichtungen je Kriterium, dargestellt in Prozent (siehe Abbildung 3.4). Nur diese Prozentzahl geht als Gewichtung in den nächsten Schritt der Nutzwertanalyse ein.
Durch die vergleichende Methode ragen die beiden Kriterien „Verbreitung möglicher Zugriffssoftware“ mit 8,8% und „typische Leistungsaufnahme zum Senden und Empfangen“ mit 8,1% am deutlichsten heraus. Beides sind sehr wichtige Kriterien für die Akzeptanz einer technischen Lösung. Es ist ein großer Nachteil, wenn eine notwendige Zugriffssoftware nicht stark verbreitet ist und erst auf den jeweiligen Zugriffsgeräten installiert werden muss. Wenn es anders herum eine Zugriffssoftware gibt, die auf nahezu jedem internetfähigen Gerät vorhanden ist, ist kein zusätzlicher Aufwand notwendig. Die Voraussetzungen an ein Zugriffsgerät, also z.B. das verwendete Betriebssystem, sind dann minimal. Die Wichtigkeit des zweiten Kriteriums wird deutlich, wenn die Anwendungsgebiete einer internetfähigen Steuerung für elektrische Verbraucher betrachtet werden. Ein Ziel einer Fernsteuerung ist es auch, Energie zu sparen. Geräte können – auch wenn man nicht zuhause ist – ein- und ausgeschaltet werden. Damit dieses Ziel erreicht werden kann, darf die Steuerung aber nicht mehr Energie verbrauchen, als mit ihrer Hilfe eingespart werden kann. Sie muss also eine möglichst geringe Leistungsaufnahme haben. Ausschlaggebend dabei ist die verwendete Kommunikationstechnologie.

		Kriterien	Gewicht %	Summe
Kommunikations-technologie	1	Reichweite der Funkübertragung	**3,7**	10
	2	typische Leistungsaufnahme zum Senden und Empfangen	**8,1**	22
	3	Verbreitung der Technologie (Interoperabilität)	**6,6**	18
	4	Integrationsaufwand in eine existierende Infrastruktur	**6,3**	17
	5	Implementierungsaufwand	**5,5**	15
	6	Unterstützung von Applikationen	**5,1**	14
	7	typische Hardwarekosten	**6,3**	17
Applikation	8	Verbreitung möglicher Zugriffssoftware	**8,8**	24
	9	Möglichkeit für einen manuellen Zugriff (Benutzerschnittstelle)	**6,6**	18
	10	Möglichkeit für einen automatischen Zugriff (M2M)	**5,1**	14
	11	Implementierungsaufwand der Applikation auf dem gewählten Betriebssystem	**6,3**	17
	12	Anforderung an Kommunikationstechnologie	**5,1**	14
Betriebssystem	13	Lizenz	**3,7**	10
	14	Verbreitung des Betriebssystems	**5,9**	16
	15	bereits unterstützte Applikationen	**5,5**	15
	16	Aufwand der Implementierung neuer Applikationen	**5,5**	15
	17	Unterstützung verschiedener Kommunikationstechnologien	**5,9**	16
		Summen:	100,0	272

Abbildung 3.4: Auswertung der Gewichtung

Die zwei Kriterien mit der geringsten Gewichtung mit 3,7% sind die „Reichweite der Funkübertragung“ und die „Lizenz“ der verwendeten Software. Die Reichweite muss größer sein als die Entfernung der Steuerung zum Funkempfänger. Beides wird sich im Heimbereich aber wahrscheinlich im selben Raum befinden. Solange ein bestimmtes Mindestmaß gegeben ist, ist die Reichweite kaum ausschlaggebend. Die Lizenz der verwendeten Software ist eher für eine kommerzielle Nutzung interessant. Deswegen soll sie hier mit betrachtet werden, ist aber wenig ausschlaggebend.

3.3.3 Bewertung der Alternativen

Im nächsten Schritt wird bewertet, inwieweit die jeweilige Alternative aus Kapitel 3.2 jedes Kriterium erfüllt. Dabei werden Punkte von 0 (erfüllt das Kriterium gar nicht) bis 6 (erfüllt das Kriterium sehr gut) vergeben. Durch die Multiplikation mit der Gewichtung in Prozent und das Zusammenzählen der Punkte ergibt sich für jede Alternative, also für jede betrachtete Komplettlösung, ein Endwert – der sogenannte Nutzwert.
Die Bewertung erfolgt nach eigenen Annahmen. Die vergebenen Punkte – der Teilnutzwert – wird dabei jeweils kurz begründet:

- **Kommunikationstechnologie**

  1. **Reichweite der Funkübertragung**
     6LoWPAN und ZigBee basieren beide auf IEEE 802.15.4, welches im PAN-Bereich (Personal Area Network) eingesetzt wird. Es hat eine Reichweite von mehreren Metern. Ohne Störungen kann die Reichweite bis ca. 20 Metern betragen. WLAN – oder genauer IEEE 802.11 – wird im LAN-Bereich (Local Area Network) eingesetzt. Je nach verwendeter Variante kann die Reichweite ohne Störungen bis ca. 100 Meter betragen. Die Alternativen mit 6LoWPAN und ZigBee bekommen 3 Punkte, die Alternative mit WLAN bekommt 5 Punkte.

2. **typische Leistungsaufnahme zum Senden und Empfangen**
   Den größten Anteil am Energieverbrauch hat die verwendete Kommunikationstechnologie. Hier muss wieder IEEE 802.15.4 und IEEE 802.11 verglichen werden. Als Vergleich werden die Herstellerangaben eines typischen Funkchips herangezogen. Für IEEE 802.15.4 wird der AT86RF230 von der Firma Atmel [AT86RF230] genommen. Er verbraucht ca. 16mA bei 3V Eingangsspannung. Das ergibt einen Energieverbrauch von ca. 50mW. Für IEEE 802.11 wird der CC3000 der Firma Texas Instruments [CC3000] genommen. Er benötigt ca. 200mA bei 3,6V Eingangsspannung. Das ergibt einen Energieverbrauch von ca. 700mW. Die Alternativen mit 6LoWPAN und ZigBee bekommen 5 Punkte, die Alternative mit WLAN bekommt 2 Punkte.

3. **Verbreitung der Technologie (Interoperabilität)**
   WLAN ist mit Abstand am meisten verbreitet, hat aber aufgrund seiner höheren Leistungsaufnahme bisher wenig Einsatz in Kleinststeuerungen gefunden. ZigBee ist im Bereich von Wireless Sensor Networks sehr verbreitet, allerdings ist ZigBee ein Industriestandard und kein frei verfügbarer Standard wie die IEEE Standards. Das könnte eine weitreichende Verbreitung bremsen. 6LoWPAN ist standardisiert in der RFC4944, allerdings noch nicht sehr weit verbreitet. Alternativen mit 6LoWPAN bekommen 3 Punkte, die Alternative mit WLAN 6 und die mit ZigBee 4 Punkte.

4. **Integrationsaufwand in eine existierende Infrastruktur**
   Weil WLAN in vielen Haushalten schon verwendet wird, ist da der Aufwand minimal. Generell muss ein Router vorhanden sein, der ans Internet angeschlossen ist und WLAN-fähig ist. Bei 6LoWPAN ist statt WLAN ein 6LoWPAN-Empfänger notwendig. Für ZigBee wird ein ZigBee-Empfänger benötigt, der zusätzlich als Gateway fungiert. Die Alternativen mit 6LoWPAN bekommen 4 Punkte, die Alternative mit WLAN 6 Punkte und die mit ZigBee 1 Punkt.

5. **Implementierungsaufwand**
   6LoWPAN ist in Contiki und TinyOS vollständig implementiert. Mehrere IEEE 802.15.4 Funkchips werden von Contiki unterstützt. Hier ist der Implementierungsaufwand minimal. Für ZigBee und BitCloud gilt das ebenso. Der Implementierungsaufwand von WLAN besteht darin, den verwendeten Funkchip von Contiki unterstützbar zu machen. Die Alternativen mit 6LoWPAN und Contiki bekommen 6 Punkte, die Alternative mit 6LoWPAN und TinyOS bekommt 6 Punkte, die Alternative mit WLAN 3 Punkte und die mit ZigBee 6 Punkte.

6. **Unterstützung von Applikationen**
   6LoWPAN und WLAN basieren auf TCP/IP. Prinzipiell wird jede Applikation unterstützt, die auf TCP/IP aufbaut. ZigBee hingegen benötigt einen Gateway. Eine neue Applikation muss auf diesem Gateway implementiert werden. Die Alternativen mit 6LoWPAN und WLAN bekommen 6 Punkte, die mit ZigBee 2 Punkte.

7. **typische Hardwarekosten**
   Der Funkchip AT86RF230 (6LoWPAN und ZigBee) ist im freien Markt für ca. 5€ erhältlich, der Funkchip CC3000 (WLAN) für ca. 15€. Die Alternativen mit 6LoWPAN und ZigBee bekommen 4 Punkte, die Alternative mit WLAN bekommt 2 Punkte.

- **Applikation**

8. **Verbreitung möglicher Zugriffssoftware**
   Nahezu jedes internetfähige Gerät hat einen Webbrowser. Der Zugriff über HTTP ist also der Idealfall. Für einen Zugriff über SNMP wird ein SNMP-Manager benötigt. Ein SNMP-Manager ist für nahezu jedes Betriebssystem verfügbar. Die Alternativen mit HTTP bekommen 6 Punkte, die Alternative mit SNMP bekommt 4 Punkte.

9. **Möglichkeit für einen manuellen Zugriff (Benutzerschnittstelle)**
   Mit einem Webbrowser und in der Regel auch mit einem SNMP-Manager kann manuell auf die Steuerung zugegriffen werden. Bei einem Webbrowser wird die Benutzerschnittstelle, also die grafische Oberfläche, von der Steuerung bestimmt, beim SNMP-Manager von dem verwendeten SNMP-Manager. Die Alternativen mit HTTP bekommen 6 Punkte, die Alternative mit SNMP bekommt 5 Punkte.

10. **Möglichkeit für einen automatischen Zugriff (M2M)**
    Ein SNMP-Manager kann in der Regel den Zustand der Steuerung automatisch abfragen und setzen. Ein Webbrowser ist dazu normalerweise nicht in der Lage. Es gibt aber Möglichkeiten, HTTP-Anfragen zu automatisieren. Somit ist ein automatischer Zugriff über HTTP auch möglich. Die Alternativen mit HTTP bekommen 4 Punkte, die Alternative mit SNMP bekommt 6 Punkte.

11. **Implementierungsaufwand der Applikation auf dem gewählten Betriebssystem**
    Im Contiki-Quellcode ist bereits eine Webserver-Applikation verfügbar. Sie muss nur an die Anforderungen der Steuerung angepasst werden. Eine Implementierung eines SNMP-Agenten ist nicht in Contiki vorhanden, aber es gibt bereits eine verfügbare Implementierung, auf die aufgesetzt werden kann. Für TinyOS sind mehrere Beispielimplementierungen eines Webservers verfügbar. Bei der Alternative mit ZigBee muss das HTTP Protokoll im Gateway implementiert werden. Die Alternativen mit HTTP und Contiki bekommen 5 Punkte, die Alternative mit SNMP und Contiki bekommt 4 Punkte, die Alternative mit HTTP und TinyOS bekommt 4 Punkte und die Alternative mit ZigBee und HTTP bekommt 3 Punkte.

12. **Anforderung an Kommunikationstechnologie**
    Wenn man als Beispiel die Abfrage einer bestimmten Information betrachtet, wird mittels HTTP die Darstellung der Information in HTML-Format übertragen. Bei SNMP wird die Information als Wert mit OID und Datentyp übertragen. Die Darstellung wird dem SNMP-Manager überlassen. Das bedeutet, dass HTTP-Pakete in der Regel größer als SNMP-Pakete sind, vor allem, wenn auch Bilder mit übertragen werden. Sie erfordern deswegen eine höhere Bandbreite. Bei HTTP und ZigBee verläuft die HTTP-Übertragung nur bis zum Gateway, zwischen Gateway und Steuerung wird dann allein über ZigBee kommuniziert. Hier sind die Anforderungen wesentlich geringer. Die Alternativen mit HTTP bekommen 4 Punkte, die mit SNMP bekommt 5 Punkte und die mit Zibgee und HTTP bekommt 6 Punkte.

- **Betriebssystem**

13. **Lizenz**
    Contiki ist frei verfügbar unter einer BSD-ähnlichen Lizenz. So lange der Copyright-Text im Quellcode verbleibt, kann Contiki kommerziell und nicht-kommerziell verwendet werden. Das gleiche gilt auch für TinyOS. BitCloud ist eine Softwareumgebung von Atmel, die mit der Zustimmung der Atmel Lizenzvereinbarungen frei erworben werden kann. Die Alternativen mit Contiki und TinyOS bekommen 6 Punkte, die Alternative mit BitCloud bekommt 5 Punkte.

14. **Verbreitung des Betriebssystems**
    Zur Verbreitung der Betriebssysteme liegen keine Daten vor, allerdings gibt es eine große Entwicklergemeinde bei TinyOS und Contiki. Es gibt Mailing Listen, über die Fragen gestellt werden können. Für beide Betriebssysteme wird ein wiki gepflegt. Bei BitCloud gibt es verschiedene Foren und Unterstützung durch den Hersteller Atmel. Alle Betriebssysteme werden laufend aktualisiert. Alle Alternativen bekommen 4 Punkte.

15. **bereits unterstützte Applikationen**
    Bei dem BitCloud Quellcode sind mehrere Beispielapplikationen dabei, auf die aufgebaut werden kann bzw. von denen Quellcode-Teile übernehmen werden können. Bei TinyOS gibt es ein paar wenige vorgefertigte Applikationen und auch mehrere Beispielimplementierungen von verschiedenen Applikationen. Bei Contiki gibt es über 30 vorgefertigte Applikationen. Neben einem Webserver gibt es z.B. einen ftp-client, einen telnet-client und -server sowie eine twitter-Applikation. Die Alternativen mit Contiki bekommen 6 Punkte, die Alternative mit TinyOS bekommt 5 Punkte und die Alternative mit BitCloud bekommt 2 Punkte.

16. **Aufwand der Implementierung neuer Applikationen**
    Ohne genauer auf die Komplexität und die Art der Programmierung einzugehen, wurden für die drei unterschiedlichen Betriebssysteme die Quellcodezeilen für eine kleine Applikation zur Ansteuerung von LEDs verglichen. Der Contiki-Quellcode umfasst 34 Zeilen, TinyOS 36 Zeilen und BitCloud 93 Zeilen. Contiki und BitCloud werden in C programmiert, für eine Programmierung müssen aber die jeweiligen Systembefehle bekannt sein. TinyOS hat eine eigene Programmiersprache nesC, die an C angelehnt ist. Aufgrund der schwierigen Vergleichbarkeit bekommen alle Alternativen 4 Punkte.

17. **Unterstützung verschiedener Kommunikationstechnologien**
    Contiki unterstützt mit seinen uIP Stack die Protokolle der TCP/IP Suite, inklusive IPv6. Ebenso wird das Protokoll 6LoWPAN über IEEE 802.15.4 unterstützt. Ethernet und WLAN wird unterstützt, aber es ist ein Aufwand notwendig, um die jeweilige Hardware anzusteuern. Ähnlich gilt dies auch für TinyOS. BitCloud unterstützt kein IP. Die Alternativen mit Contiki und TinyOS bekommen 5 Punkte, die Alternative mit BitCloud bekommt 2 Punkte.

		6LoWPAN HTTP Contiki		6LoWPAN SNMP Contiki		WLAN HTTP Contiki		6LoWPAN HTTP TinyOS		Zigbee HTTP BitCloud	
	Gewicht %										
1	3,7	3	1,84	3	1,84	5	3,06	3	1,84	3	1,84
2	8,1	5	6,74	5	6,74	2	2,70	5	6,74	5	6,74
3	6,6	3	3,31	3	3,31	6	6,62	3	3,31	4	4,41
4	6,3	4	4,17	4	4,17	6	6,25	4	4,17	1	1,04
5	5,5	6	5,51	6	5,51	3	2,76	6	5,51	6	5,51
6	5,1	6	5,15	6	5,15	6	5,15	6	5,15	2	1,72
7	6,3	4	4,17	4	4,17	2	2,08	4	4,17	4	4,17
8	8,8	6	8,82	4	5,88	6	8,82	6	8,82	6	8,82
9	6,6	6	6,62	5	5,51	6	6,62	6	6,62	6	6,62
10	5,1	4	3,43	6	5,15	4	3,43	4	3,43	4	3,43
11	6,3	5	5,21	4	4,17	5	5,21	4	4,17	3	3,13
12	5,1	4	3,43	5	4,29	4	3,43	4	3,43	6	5,15
13	3,7	6	3,68	6	3,68	6	3,68	6	3,68	5	3,06
14	5,9	4	3,92	4	3,92	4	3,92	4	3,92	4	3,92
15	5,5	6	5,51	6	5,51	6	5,51	5	4,60	2	1,84
16	5,5	4	3,68	4	3,68	4	3,68	4	3,68	4	3,68
17	5,9	5	4,90	5	4,90	5	4,90	5	4,90	2	1,96
	100,0		**80,09**		**77,57**		**77,82**		**78,13**		**67,03**

Abbildung 3.5: Bewertung der Alternativen

Abbildung 3.5 zeigt, dass die Alternative „ZigBee + HTTP + BitCloud“ mit 67,03% die geringste Übereinstimmung hat. In vier Kriterien („Integrationsaufwand in eine existierende Infrastruktur“

und „Unterstützung von Applikationen“ in der Kommunikationstechnologie und „bereits unterstützte Applikationen“ und „Unterstützung verschiedener Kommunikationstechnologien“ im Betriebssystem) schneidet diese Alternative eindeutig schlechter ab.
Die anderen vier Alternativen liegen nicht sehr weit auseinander. Die Alternative „6LoWPAN + HTTP + Contiki“ hat mit 80,09% die größte Übereinstimmung der Kriterien.

3.4 Entscheidung für eine Alternative

Die Alternative „6LoWPAN + HTTP + Contiki“ hat bei der Nutzwertanalyse die größte Übereinstimmung mit den gestellten Kriterien erzielt. Bei der Nutzwertanalyse wurde nur der Betrieb einer Applikation betrachtet. Mit dem Betriebssystem Contiki ist es aber möglich mehrere Applikationen parallel als Prozesse laufen zu lassen. Es ist also denkbar, die beiden Alternativen „6LoWPAN + HTTP + Contiki“ und „6LoWPAN + SNMP + Contiki“ zu verknüpfen. In solch einer Lösung würde dann ein Prozess für HTTP, eine Webserver-Applikation, und ein Prozess für SNMP, ein SNMP-Agent, parallel laufen. Dabei muss aber geprüft werden, ob die knappen Ressourcen ausreichend sind.
Eine Webserver-Applikation ist im Contiki-Betriebssystem bereits vorhanden und auch ein SNMP-Agent-Implementierung ist bereits verfügbar. Um beispielhaft die Implementierung einer neuen Applikation im Betriebssystem Contiki zu prüfen, wird das Protokoll IEC104 entwickelt. Die Steuerung fungiert dabei als IEC104-Slave. Sie teilt einer IEC104-Master-Applikation den Zustand der Steuerung mit und nimmt Befehle entgegen. Weil aufgrund der knappen Ressourcen ein Parallelbetrieb von drei Applikationen nicht möglich ist, werden nur eine Webserver-Applikation und eine IEC104-Slave-Applikation implementiert. Eine SNMP-Agent-Implementierung wird nicht weiter verfolgt.
Nach der Implementierung wird die umgesetzte Lösung anhand der Kriterien der Nutzwertanalyse bewertet und mit der Übereinstimmung der Alternative „6LoWPAN + HTTP + Contiki“ verglichen. Ebenso wird die Erfüllung der Anforderungen nach Kano geprüft.

Kapitel 4

Konzeption und Entwurf

Die Wahl ist also auf die Kombination „6LoWPAN + HTTP + Contiki“ und einer zusätzlichen parallelen Anwendung IEC104-Slave gefallen. Bevor mit der eigentlichen Implementierung begonnen werden kann, muss die zu verwendende Hardware ausgewählt werden.
Die Kommunikationstechnologie 6LoWPAN und das damit verbundene Protokoll IEEE 802.15.4 werden detaillierter vorgestellt. Wie funktioniert 6LoWPAN im Detail und wie wird die Brücke zu TCP/IP geschlagen? Am Ende wird die Entwicklungsumgebung vorgestellt, die zur Implementierung verwendet wird. Dies beinhaltet auch eine kurze Einführung in das Betriebssystem Contiki.
Der Aufbau der Implementierung und die Einteilung in Hardware- und Softwareimplementierung wird am Ende dieses Kapitel behandelt.

4.1 Auswahl der Hardware

Weil schon Erfahrung mit dem AVR Ravenboard von Atmel besteht, soll die Hardware auf diesem basieren. Das Entwicklungsboard selbst kann aber nicht verwendet werden, weil es für die Einsatzzwecke zu groß ist. Auf manchen Komponenten, wie das LED-Display oder das Audiogerät, kann verzichtet werden. Die benötigten Komponenten des AVR Ravenboard sind:

- AVR Mikrocontroller ATmega1284P
- Kommunikationseinheit AT86RF230 mit Leiterbahnantenne (PCB antenna)
- Anschluss für die Energieversorgung
- Programmierschnittstelle (JTAG oder ISP)

Das schematische Leiterplattenlayout und eine Liste der Bauteile sind im Benutzerhandbuch für die Ravenboard Hardware [AVR RZRaven] nachzulesen. Darauf aufbauend war die erste Überlegung ein eigenes Leiterplattenlayout zu erstellen, das nur die oben genannten Komponenten enthält. Dadurch könnte eine angemessene Größe erreicht werden. Der Hauptgrund, warum diese Idee verworfen wurde, waren Probleme mit dem Layout der Antenne. Weil es sich hier um Hochfrequenztechnik handelt, gibt es eine Vielzahl von Schwierigkeiten, die berücksichtigt werden müssen. So beeinflussen Leiterbahnen durch die hohe Frequenz die elektrischen Eigenschaften. Eine spezielle Leiterbahnführung muss berechnet werden. Die Auswahl der Materialien spielt dabei auch eine Rolle. Aus Mangel an Erfahrung und aufgrund fehlender Ausrüstung wurde nach Alternativlösungen gesucht.
Der Schwerpunkt fiel auf sogenannte Single-Chip-Lösungen. Verschiedene Hersteller bieten fertige Module an, bestehend aus Mikrocontroller, Kommunikationseinheit und integrierter Antenne. Der Hersteller Atmel bietet mit dem ATZB-24-A2 ein solches Modul an. Es verwendet die gleiche Kommunikationseinheit und mit dem ATmega1281 einen ähnlichen Mikrocontroller wie das AVR

Ravenboard. Eine Antenne ist ebenso als Dual-Chip-Antenne auf dem Modul integriert. Damit unterscheidet sich die Hardware kaum vom AVR Ravenboard, aber es kann auf eine Entwicklung einer eigenen Leiterplatte verzichtet werden. Die Wahl fiel daher auf das Modul ATZB-24-A2 von Atmel.

4.2 Kommunikationstechnologie 6LoWPAN

Als Kommunikationstechnologie wird 6LoWPAN verwendet. Aus diesem Grund wird an dieser Stelle detaillierter die Struktur von IEEE 802.15.4 und 6LoWPAN erläutert.

4.2.1 IEEE 802.15.4 Datenframeaufbau

In Abbildung 4.1 ist der Aufbau eines IEEE-802.15.4-Datenframes zu sehen. Der Teil der physikalischen Schicht enthält eine Präambel, den Start Frame Delimiter und ein Feld, das die Datenframelänge angibt. Je nach verwendeter Modulation kann dieser Teil etwas abweichen. Der Teil der Linkschicht startet mit einem Kontrollfeld. Hier wird unter anderem angegeben, ob der optionale Security Header verwendet wird und wie das Adressfeld aufgebaut ist. Die maximale Größe eines MAC-Frames beträgt 127 Bytes.

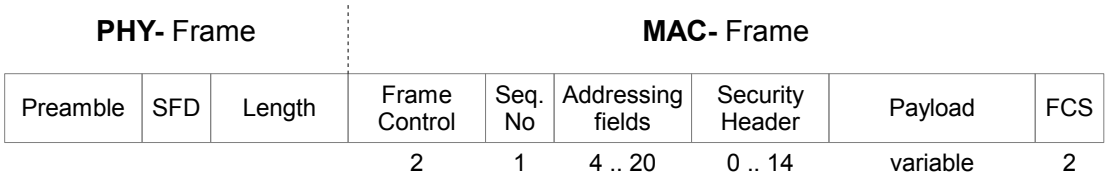

Abbildung 4.1: IEEE 802.15.4 Datenframestruktur

Generell besteht das Adressfeld aus der Zieladresse (Destination Address) und der Quelladresse (Source Address). Diese Adressen bestehen aus einem PAN (Personal Area Network) Identifier, auf den hier nicht näher eingegangen werden soll, und der eigentlichen Adresse. Diese können 16-bit Kurzadressen oder 64-bit Langadressen sein. Kurzadressen werden dabei dynamisch von einem sogenannten PAN Koordinator vergeben, wobei eine Langadresse eine dem Gerät fest zugeordnete Adresse nach EUI-64 ist. 24 Bit dieser Adresse bezeichnen den Hersteller, die sogenannte OUI (Organizational Unique Identifier). Die weiteren 40 Bit kann der Hersteller frei wählen, um das Gerät eindeutig zu adressieren, ähnlich einer Seriennummer. Die Langadresse eines Gerätes ist damit weltweit eindeutig.

4.2.2 6LoWPAN Paketaufbau

Neben der Fragmentierung ist die Hauptfunktion von 6LoWPAN eine Header-Komprimierung vorzunehmen. Auf diese Komprimierung soll hier näher eingegangen werden. So können der IPv6-Header und teilweise sogar weitere Header effektiv verkleinert werden, um mehr Platz für Nutzdaten verfügbar zu haben. Diese Komprimierung wurde in [RFC4944, Section 10] spezifiziert und in [RFC6282] aktualisiert und erweitert. Die Komprimierung und ebenso die Fragmentierung wird über das sogenannte Dispatch Byte gesteuert. Das Dispatch Byte ist das erste Byte in der IEEE 802.15.4 Payload und gibt den verwendeten Header-Typ an.
In Abbildung 4.2 wird anhand des Beispiels eines IPv6-Neighbor-Solicitation-Pakets gezeigt, wie effektiv der IPv6-Header komprimiert werden kann. Das Paket auf der linken Seite ist ein mitgeschnittenes Datenpaket eines AVR Ravenboards, das mit dem Betriebsystem Contiki in der Version

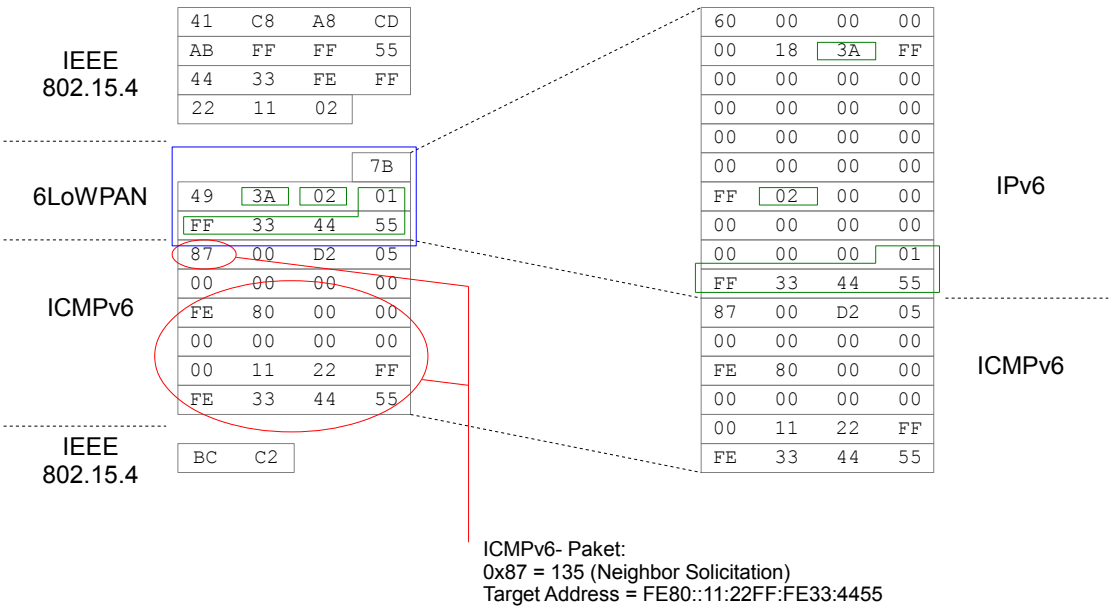

Abbildung 4.2: Beispiel 6LoWPAN-Paket: Neighbor Solicitation

2.5 programmiert wurde. Es zeigt ein ICMPv6-Paket mit einem 6LoWPAN-Header, eingekapselt in IEEE 802.15.4. Auf der rechten Seite ist das gleiche ICMPv6-Paket mit entschlüsselten IPv6-Header zu sehen. Die Informationen des ICMPv6-Teils sind unverändert. Die 40 Bytes des IPv6-Headers konnten in einem 9 Byte großen 6LoWPAN-Header komprimiert werden. Durch Komprimierung konnten 31 Bytes eingespart werden.

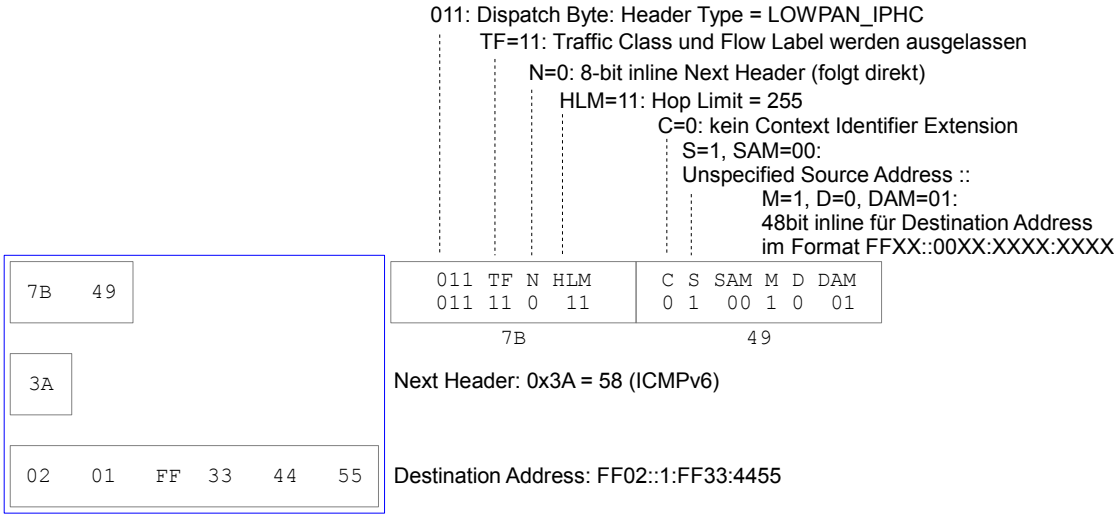

Abbildung 4.3: Beispiel 6LoWPAN-Header LOWPAN_IPHC

Der 6LoWPAN-Header ist in Abbildung 4.3 detailliert erklärt. Das erste Byte ist das Dispatch Byte, das anzeigt, welche Bedeutung die nachfolgenden Daten haben. In dem Beispiel handelt es sich um einen Header vom Typ LOWPAN_IPHC (siehe Tabelle 4.1). Dabei wird mit den ersten beiden Bytes das Basisformat des Headers beschrieben. Die IPv6-Quelladresse (Source Address) ist unspezifiziert, also (::) . Die IPv6-Zieladresse (Destination Address) ist im Format FFXX::00XX:XXXX:XXXX. Die fehlenden 6 Bytes (48 Bit), um die Zieladresse zu vervollstän-

Bit Pattern	Header Type	Reference
00 xxxxxx	NALP - Not a LoWPAN frame	RFC4944
01 000000	Reserved as replacement value for ESC	RFC6282
01 000001	IPv6 - uncompressed IPv6 Addresses	RFC4944
01 000010	LOWPAN_HC1 - LOWPAN_HC1 compressed IPv6	RFC4944
01 000011 through 01 001111	resered for future use	
01 010000	LOWPAN_BC0 - LOWPAN_BC0 broadcast	RFC4944
01 010001 through 01 011111	resered for future use	
01 1xxxxx	LOWPAN_IPHC	RFC6282
10 xxxxxx	MESH - Mesh Header	RFC4944
11 000xxx	FRAG1 – Fragmentation Header (first)	RFC4944
11 001000 through 11 011111	resered for future use	
11 100xxx	FRAGN – Fragmentation Header (subsequent)	RFC4944
11 101000 through 11 111111	resered for future use	

Tabelle 4.1: 6LoWPAN Dispatch Byte [Web:6lowpan-pars]

digen, folgen direkt nach den zwei Bytes Basisformat und einem Byte für das Next-Header-Feld. Das Next-Header-Feld zeigt mit dem Wert 58 (0x3A) an, dass als nächstes ein ICMPv6-Paket folgt.

Nachfolgend wird der ICMPv6-Teil interpretiert, um zu verstehen, warum das AVR Ravenboard dieses Paket sendet. Das erste Byte ist das Feld ICMPv6-Type. Hier beträgt der Wert 135 (0x87), es handelt sich also um ein Neighbor Solicitation Paket. Die nächsten Bytes bestehen aus einem Code-Feld (auf 0 gesetzt), einer 2-Byte-Checksumme und vier reservierte Bytes (auf 0 gesetzt). Bei der Bytefolge „FE80 0000 0000 0000 0011 22FF FE33 4455“ handelt es sich um das Target-Address-Feld. Ein Optionsfeld ist optional und wird hier nicht verwendet, weil die IPv6-Quelladresse unspezifiziet ist (::). Dieses ICMPv6-Paket wird während der IPv6-Autokonfiguration gesendet, um festzustellen, ob die selbstgenerierte Adresse vergeben ist. Dieser Vorgang nennt sich Duplicate Address Detection Test (DAD). In unserem Fall will sich das AVR Ravenboard die link-local-Adresse FE80::11:22FF:FE33:4455 zuweisen. Es wird ein Neighbor Solicitation mit dieser Adresse im Target-Address-Feld gesendet. Die Quelladresse von diesem IPv6-Paket ist die Unspecified-Adresse (::), die Zieladresse ist die Solicited-Node-Multicast-Adresse FF02::1:FF33:4455. Erst wenn dieses Paket nicht beantwortet wird, darf sich das AVR Ravenboard die selbstgenerierte link-local-Adresse zuweisen. Damit ist die IP-Verbindung am Link hergestellt und es kann mit einer Router Solicitation nach Router und IPv6-Präfixe fragen.

4.3 Entwicklungsumgebung

Um die Entwicklungsumgebung nicht Schritt für Schritt von Grund auf installieren zu müssen, stellt die Contiki-Community eine virtuelle Maschine namens Instant-Contiki [Web:Instant-Contiki] zur Verfügung. Sie läuft in einem VMWare Player und kann damit auf verschiedenen Betriebssystemen installiert werden. Die virtuelle Maschine selbst basiert auf Ubuntu Linux und hat bereits alle benötigten Entwicklungstools und -compiler installiert.

Diese virtuelle Maschine wurde auf einem Windows Host installiert und zur Erstellung des Programms verwendet. Der Windows Host und die virtuelle Maschine teilen sich ein Verzeichnis, in dem der Contiki-Quellcode liegt. Programmiert wurde mit dem Editor Notepad++ unter Windows. In der virtuellen Maschine wurde übersetzt und das erstellte Programm wurde unter Windows mit Atmel Studio und dem Programmiergerät STK500 bzw. später mit dem JTAGICE3 auf den Mikrocontroller geladen.

4.3.1 Contiki-Verzeichnisstruktur

Die Contiki-Verzeichnisstruktur besteht aus verschiedenen Unterverzeichnissen, die verschiedene Bedeutungen haben.

Listing 4.1: Contiki-Verzeichnisstruktur mit ausgewählten Unterverzeichnissen

```
/apps
    /ftp
    /ping6
    /telnet
    /telnetd
    /twitter
    /webserver
    /webserver-nano
    ...
/core
    /lib
    /net
    /sys
    ...
/cpu
    /arm
    /avr
    ...
/doc
/examples
    /webserver-ipv6-raven
    ...
/platform
    /avr-raven
    /avr-zigbit
    ...
/tools
```

Das Verzeichnis /core beinhaltet das Kernstück von Contiki. Hier wird zum Beispiel das System im Unterverzeichnis /sys definiert. Das beinhaltet das Prozesshandling und Protothreads sowie verschiedene Timer. Ebenfalls befindet sich hier im Unterverzeichnis /net der Netzwerk-Stack aufbauend auf uIP. Zum Netzwerk-Stack gehört auch IEEE 802.15.4 und 6LoWPAN. Im Unterverzeichnis /lib stehen verschiedene libraries zur Verfügung.
In den beiden Verzeichnissen /cpu und /platform ist die unterschiedliche Hardware beschrieben, die von Contiki unterstützt wird. Ein Programm, das auf einem Zigbit-Modul vom Hersteller Atmel geladen werden soll, verwendet die Verzeichnisse /platform/avr-zigbit und /cpu/avr.
Im Verzeichnis /apps befinden sich die Applikationen, die plattformunabhängig in ein Programm eingebunden werden können. Es gibt zum Beispiel eine webserver-, eine ping6-, eine telnetd- und eine twitter-Applikation. Insgesamt stehen in der Version 2.6 von Contiki 37 verschiedene Applikationen zur Verfügung. Es gibt auch plattformabhängige Applikationen. Diese befinden sich in den Verzeichnissen /platform/<platform>/apps.
Das Verzeichnis /examples beinhaltet Beispielprojekte. Hier gibt es ein Projekt webserver-ipv6-raven. Anhand dieses Beispiels wird deutlich, was notwendig ist, um eine Webserver-Applikation für das Ravenboard zu kompilieren.
Im Hauptverzeichnis gibt es eine Datei Makefile.include. Diese Datei ist Teil des Contiki-Makefile-Systems. Sie wird innerhalb eines Contiki-Projektes aufgerufen und bindet, abhängig von der Konfiguration des Projektes, die richtigen Dateien des Contiki-Systems ein.

4.3.2 Übersetzung eines Contiki-Programms

Ein Programm wird in Contiki mithilfe des Befehls „make“ übersetzt. Am Beispiel des Projekts webserver-ipv6-raven soll exemplarisch gezeigt werden, wie das Makefile System funktioniert.

Durch den Befehl „make“ wird die Datei Makefile im gleichen Verzeichnis abgearbeitet.

Listing 4.2: Auszug aus examples/webserver-ipv6-raven/Makefile

```
ifndef TARGET
  TARGET=avr-raven
  MCU=atmega1284p
endif
all:
  ${MAKE} -f Makefile.webserver TARGET=$(TARGET) NOAVRSIZE=1 webserver6.elf
```

Hier wird das TARGET, also die Plattform, und die MCU, die Microcontroller Unit, gesetzt und dann die Datei Makefile.webserver aufgerufen. Dabei wird der Parameter NOAVRSIZE gesetzt, um beim Übersetzen eine zusätzliche Ausgabe zur Speicherbelegung (avr-size) zu unterdrücken. Das Programm wird als Datei webserver6.elf erstellt.

Listing 4.3: Auszug aus examples/webserver-ipv6-raven/Makefile.webserver

```
all: webserver6
APPS=raven-webserver raven-lcd-interface
UIP_CONF_IPV6=1
CONTIKI = ../..
include $(CONTIKI)/Makefile.include
```

In der Datei Makefile.webserver werden die Applikationen raven-webserver und raven-lcd-interface über die Variable APPS eingebunden. Es wird eine Compiler Variable UIP_CONF_IPV6 gesetzt, um IPv6 zu aktivieren, und das Haupt-Makefile Makefile.include wird eingebunden.

Listing 4.4: Auszug aus examples/webserver-ipv6-raven/webserver6.c

```
#include "webserver-nogui.h"
/*---------------------------------------------------------------------------*/
AUTOSTART_PROCESSES(&webserver_nogui_process);
/*---------------------------------------------------------------------------*/
```

In der Datei webserver6.c wird ausgewählt, welche Contiki-Prozesse automatisch gestartet werden sollen. In unserem Beispiel ist das der Prozess „webserver_nogui_process“. Dieser Prozess ist Teil der Applikation raven-webserver.

Listing 4.5: Auszug aus Makefile.include

```
include $(CONTIKI)/core/net/rime/Makefile.rime
include $(CONTIKI)/core/net/mac/Makefile.mac
SYSTEM  = process.c procinit.c autostart.c elfloader.c profile.c \
          timetable.c timetable-aggregate.c compower.c serial-line.c
THREADS = mt.c
LIBS    = memb.c mmem.c timer.c list.c etimer.c ctimer.c energest.c rtimer.c \
         stimer.c print-stats.c ifft.c crc16.c random.c checkpoint.c ringbuf.c
DEV     = nullradio.c
NET     = netstack.c uip-debug.c packetbuf.c queuebuf.c packetqueue.c

ifdef UIP_CONF_IPV6
  CFLAGS += -DUIP_CONF_IPV6=1
  UIP   = uip6.c tcpip.c psock.c uip-udp-packet.c uip-split.c \
          resolv.c tcpdump.c uiplib.c simple-udp.c
  NET   += $(UIP) uip-icmp6.c uip-nd6.c uip-packetqueue.c \
          sicslowpan.c neighbor-attr.c neighbor-info.c uip-ds6.c
  ifneq ($(UIP_CONF_RPL),0)
    CFLAGS += -DUIP_CONF_IPV6_RPL=1
    include $(CONTIKI)/core/net/rpl/Makefile.rpl
  endif # UIP_CONF_RPL
else # UIP_CONF_IPV6
  UIP   = uip.c uiplib.c resolv.c tcpip.c psock.c hc.c uip-split.c uip-fw.c \
          uip-fw-drv.c uip_arp.c tcpdump.c uip-neighbor.c uip-udp-packet.c \
```

```
          uip-over-mesh.c dhcpc.c simple-udp.c
  NET   += $(UIP) uaodv.c uaodv-rt.c
endif # UIP_CONF_IPV6

### Include application makefiles
ifdef APPS
  APPDIRS += ${wildcard ${addprefix $(CONTIKI)/apps/, $(APPS)} \
             ${addprefix $(CONTIKI)/platform/$(TARGET)/apps/, $(APPS)} \
             $(APPS)}
  APPINCLUDES = ${foreach APP, $(APPS), ${wildcard ${foreach DIR, \
                $(APPDIRS), $(DIR)/Makefile.$(APP)}}}
  -include $(APPINCLUDES)
  APP_SOURCES = ${foreach APP, $(APPS), $($(APP)_src)}
  DSC_SOURCES = ${foreach APP, $(APPS), $($(APP)_dsc)}
  CONTIKI_SOURCEFILES += $(APP_SOURCES) $(DSC_SOURCES)
endif

### Include target makefile (TODO Unsafe?)
target_makefile :=$(wildcard $(CONTIKI)/platform/$(TARGET)/Makefile.$(TARGET)\
         ${foreach TDIR, $(TARGETDIRS), $(TDIR)/$(TARGET)/Makefile.$(TARGET)})
```

In der Datei Makefile.include werden die Contiki-Systemdateien eingebunden. Ebenso werden abhängig von der Compiler-Variable UIP_CONF_IPV6 die benötigten Netzwerkdateien eingebunden. Für jede parametrierte Applikation wird das zugehörige Makefile aufgerufen. Je nach ausgewählten TARGET werden die entsprechenden Makefiles eingebunden. In diesem Beispiel werden die Makefiles platform/avr-raven/apps/raven-webserver/Makefile.raven-webserver, platform/avr-raven/apps/raven-lcd-interface/Makefile.raven-lcd-interface und platform/avr-raven/Makefile.avr-raven eingebunden. Das Makefile.avr-raven wiederum bindet noch die Makefiles cpu/avr/Makefile.avr und cpu/avr/radio/Makefile.radio ein.
Das Makefile-System von Contiki ist sehr verschachtelt. Es hat aber den Vorteil, dass in einem Projekt nur die Variablen APPS und TARGET im eigenen Makefile richtig gesetzt werden müssen und es werden automatisch die richtigen Dateien eingebunden. Es ist somit sehr einfach, in einem Projekt eine Applikation hinzuzufügen oder zu entfernen. Zusätzlich wird eine .c-Datei benötigt, in der angegeben wird, welche Prozesse gestartet werden sollen. Dies wird über den Befehl AUTOSTART_PROCESSES ausgeführt. Daraufhin kann über „make“ das Programm erzeugt werden, das dann auf den Mikrocontroller geladen werden kann.

4.3.3 Programmieren eines Mikrocontrollers

Durch Übersetzung des Contiki-Beispielprojektes webserver-ipv6-raven wird eine Datei „webserver6.elf“ im ELF Format erzeugt. Hiervon wird eine Kopie „webserver6-avr-raven.elf“ erstellt. Diese Datei enthält Informationen darüber, was in den Flash und in den EEPROM des AVR Mikrocontrollers geladen werden muss. Ebenfalls beinhaltet es Informationen über das Setzen der Fuse Bits. Fuse Bits sind Einstellungen des Mikrocontrollers, die nicht von der Software geändert werden können. Sie schalten gewisse Funktionen, zum Beispiel woher die Taktfrequenz bezogen wird, ein oder aus.

Listing 4.6: Auszug aus examples/webserver-ipv6-raven/Makefile

```
TARGET=avr-raven
MCU=atmega1284p
OUTFILE=webserver6-$(TARGET)
avr-objcopy -O ihex -R .eeprom -R .fuse -R .signature \
            $(OUTFILE).elf $(OUTFILE).hex
avr-size -C --mcu=$(MCU) $(OUTFILE).elf
```

Durch zusätzliche Befehle im Makefile wird eine Datei „webserver6-avr-raven.hex“ erzeugt. Diese Datei enthält den Inhalt, der in den Flash geschrieben werden soll, im Intel-HEX-Format. Mit dem

„avr-size“-Befehl wird die Anzahl der Bytes angezeigt, die im Flash, im RAM und im EEPROM benötigt werden. Dies kann mit dem zur Verfügung stehenden Speicher verglichen werden.

Listing 4.7: Ausgabe vom Befehl „avr-size“

```
AVR Memory Usage
----------------
Device: atmega1284p

Program:   70874 bytes (54.1% Full)
(.text + .data + .bootloader)

Data:      13013 bytes (79.4% Full)
(.data + .bss + .noinit)

EEPROM:       63 bytes (1.5% Full)
(.eeprom)
```

Durch Hinzufügen eines zusätzlichen Befehls ins Makefile kann eine Datei „webserver6-avr-raven_eeprom.hex“ erzeugt werden. Diese Datei enthält dann den Inhalt im Intel-HEX-Format, der in den EEPROM geschrieben werden soll. Mit den beiden Dateien im Intel-HEX-Format ist es möglich, den Mikrocontroller mit einem Programmiergerät zu beschreiben, das kein ELF Format lesen kann.

Listing 4.8: Befehl um eeprom.hex zu erzeugen

```
avr-objcopy -O ihex -j .eeprom \
            -set-section-flags=.eeprom="alloc,load" --change-section-lma \
            .eeprom=0 $(OUTFILE).elf $(OUTFILE)_eeprom.hex
```

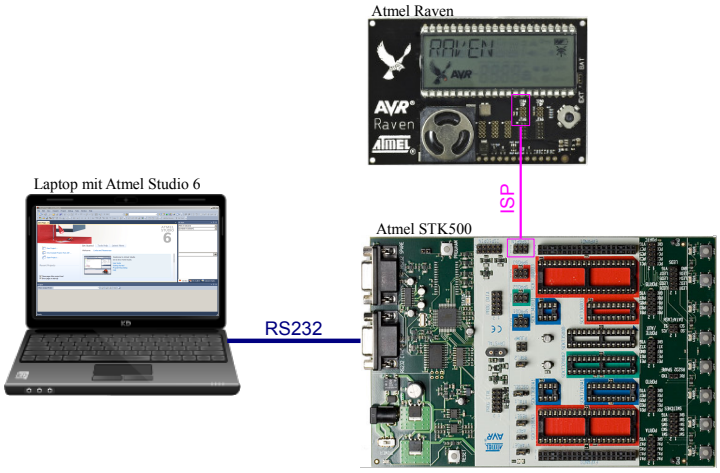

Abbildung 4.4: Aufbau Ravenboard mit STK500

Mittels des Programms Atmel Studio 6 werden wahlweise die erzeugten Dateien im Intel-HEX-Format oder die erzeugte Datei im ELF Format in den Flash und in den EEPROM des Mikrocontrollers programmiert. Zu Beginn wurde das Beispielprojekt webserver-ipv6-raven auf den Mikrocontroller AT-Mega1284P eines Atmel Ravenboard programmiert. Dabei wurde die Programmierschnittstelle ISP und das Programmiergerät Atmel STK500 verwendet (Abbildung 4.4). Später bei der entwickelten Hardware wurde die Programmierschnittstelle JTAG und das Programmiergerät Atmel JTAGICE3 verwendet.

4.4 Aufbau der Implementierung

Die Implementierung der Lösung erfolgte in zwei Teilen. Es wurde die Hardware erstellt und zeitlich parallel dazu wurde die Software entwickelt. Beide Teile konnten relativ unabhängig voneinander implementiert werden. Die Schnittstelle von Hardware und Software ist das Zigbit-Modul ATZB-24-A2. Auf diesem Modul wird die fertige Software einprogrammiert. Die Hardware versorgt das Modul mit Energie und schaltet nach Softwarevorgaben die Steckdose ein und aus.

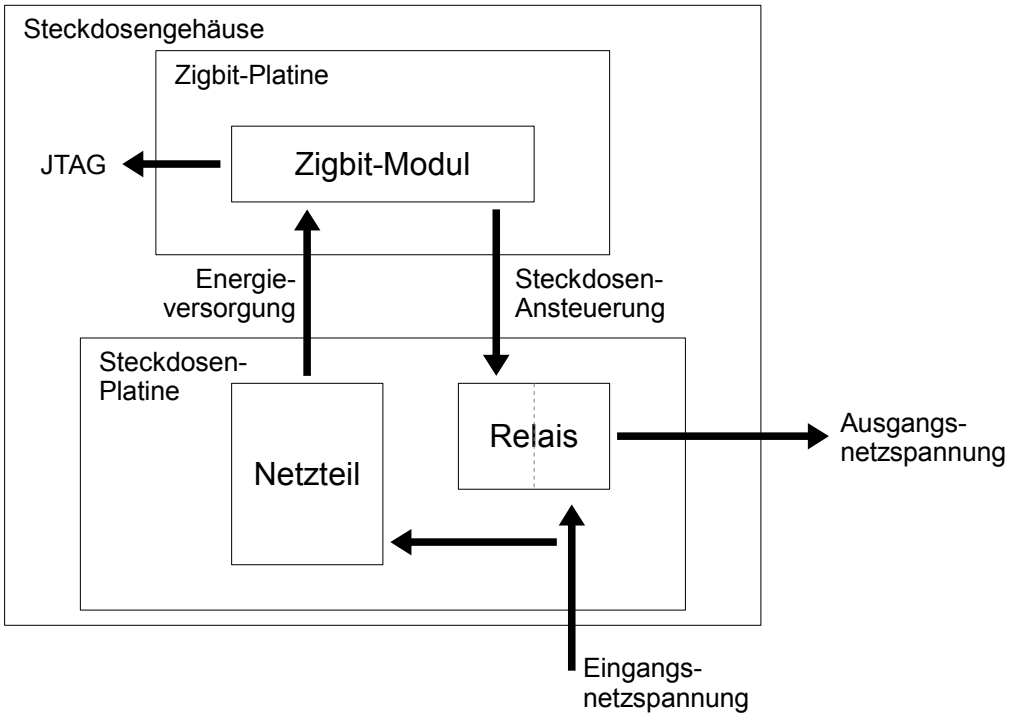

Abbildung 4.5: Architektur der Hardwareimplementierung

Die Hardware besteht aus einer Platine mit dem Zigbit-Modul und einer Platine mit einem Netzteil zur Energieversorgung und einem Relais, um die Steckdose schalten zu können. Beide Platinen sind in einem Steckdosengehäuse, das im Einzelhandel erworben wurde, untergebracht. Um das Zigbit-Modul programmieren zu können, wird eine Programmierschnittstelle benötigt. Dazu wurde auf der Platine ein JTAG-Anschluss implementiert. Beide Platinen und ihre jeweiligen Schnittstellen sind in Abbildung 4.5 zu sehen.

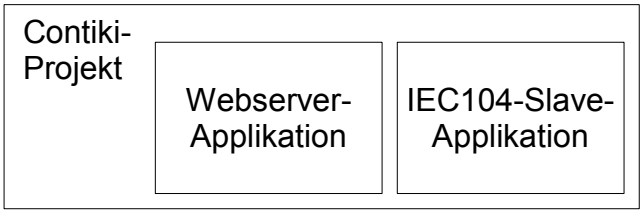

Abbildung 4.6: Architektur der Softwareimplementierung

Abbildung 4.6 zeigt, dass die Software aus zwei Contiki-Applikationen besteht, einer Webserver-Applikation und einer IEC104-Slave-Applikation. Beide Applikationen werden zusammen in einem Contiki-Projekt eingebunden, sie laufen dann parallel als eigene Prozesse. Durch Übersetzung des Projektes entsteht das fertige Programm, das auf das Zigbit-Modul geladen wird.

Kapitel 5

Implementierung der Hardware

Die Hardware besteht aus einer Steckdose, die im freien Handel erworben wurde. Sie liefert das Gehäuse, das Relais zur Ansteuerung der Steckdose und eine Gleichspannungsversorgung für den eingebauten Mikrocontroller. In dieses Steckdosen-Gehäuse soll das Zigbit-Modul ATZB-24-A2 eingebaut werden, das einen 8-bit AVR Mikrocontroller und ein Funkchip inklusive Antenne enthält.
Zuerst werden die Steckdose und das Zigbit-Modul vorgestellt und danach die notwendigen Veränderungen, die zum Einbau und zum Betrieb notwendig waren.

5.1 Steckdose

Die Steckdose wurde von dem Einzelhandelsunternehmen Tchibo [Web:Tchibo Steckdose] erworben. Die enthaltene Steuerung kann angeschlossene Verbraucher zeitgesteuert ein- und ausschalten. In Abbildung 5.1 sind die Steckdose und das Zigbit-Modul zu sehen. Das Gehäuse der Steckdose lässt sich nach Entfernen von vier Kreuzschrauben leicht öffnen.

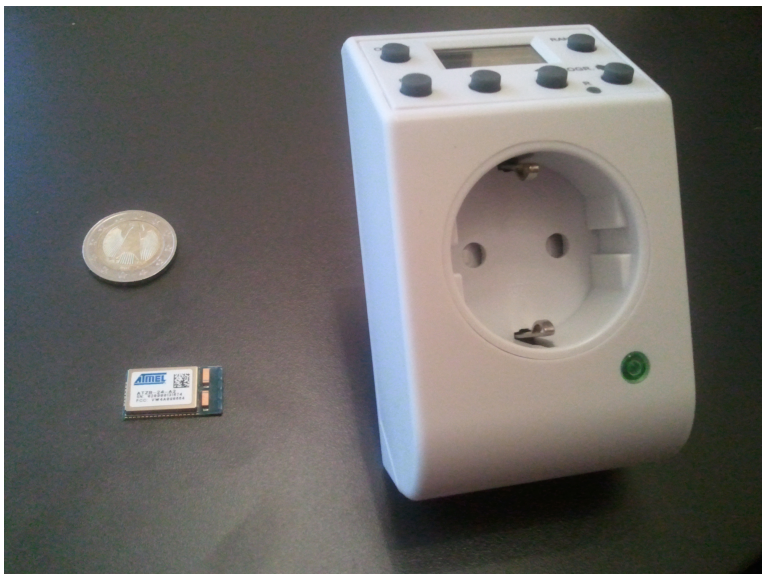

Abbildung 5.1: Tchibo Steckdose mit Zigbit-Modul

Das Innere besteht aus zwei Platinen. Die obere Platine enthält die Steuerung und die untere Platine enthält die Gleichspannungsversorgung und das Relais zum Schalten angeschlossener Verbraucher. Beide Platinen sind mit drei Leitungen miteinander verbunden. Sie sind farblich markiert (schwarz, rot und gelb) und haben folgende Funktionen:

- schwarz 0V (GND)
- rot Versorgungsspannung (VCC)
- gelb Steuerungssignal

Mit der schwarzen und der roten Leitung wird die obere Platine mit der notwendigen Spannung versorgt. Die gelbe Leitung wird von der Steuerung verwendet, um das Relais auf der unteren Platine anzusteuern.
Die Steuerung – also die obere Platine – soll durch das Zigbit-Modul ersetzt werden. Die untere Platine wird genauer analysiert. Dabei stehen zwei Fragen im Vordergrund: ist die Versorgungsspannung ausreichend dafür, das Zigbit-Modul anzusteuern? Kann das Zigbit-Modul die Steuerungssignal-Leitung so ansteuern, dass das Relais schaltet?

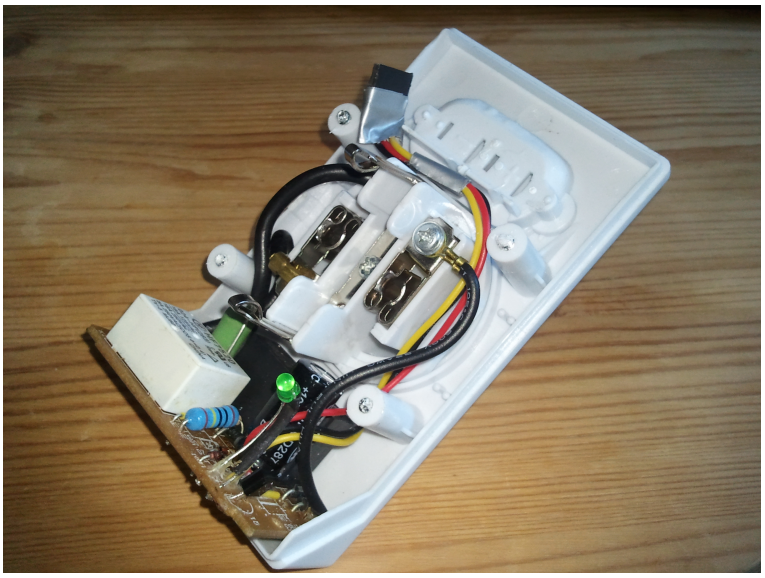

Abbildung 5.2: Innenleben der Tchibo Steckdose

Abbildung 5.2 zeigt das Innenleben der Steckdose. Die obere Platine ist bereits entfernt und nicht zu sehen. Die untere Platine wurde bereits bearbeitet. An den drei Verbindungsleitungen wurde eine Buchsenleiste als Steckverbindung angelötet.

5.1.1 Verschaltung der unteren Platine

Durch Verfolgung der Verbindungen auf der Platine wurde der Schaltplan in Abbildung 5.3 erstellt. Die Namensgebung der Bauteile stammt von der Beschriftung auf der Platine. Die Größe der Widerstände wurde anhand der aufgedruckten Farbenringe festgestellt. Einige Bauteile konnten aufgrund ihres Aufdrucks näher bestimmt werden – andere, zum Beispiel die Dioden und der Transistor Q1, konnten nicht näher bestimmt werden.
Bei dem Relais handelt es sich um ein Leistungsrelais HF7520 des Herstellers Hongfa. Bei einer Lastspannung von bis zu 250VAC kann es Lastströme bis zu 16 A schalten. Angesteuert wird es durch eine Sollspannung von 24VDC.
Bei dem Kondensator C12 handelt es sich um einen Entstörkondensator der Klasse X2. Er hat eine Nennspannung von 275VAC, allerdings konnte die Kapazität nicht festgestellt werden. Entstörkondensatoren sind für den Einsatz direkt am Stromnetz geeignet. Sie leiten hochfrequente Störsignale ab. Außerdem schützen sie die Schaltung vor netzseitigen Überspannungen und umgekehrt das Stromnetz vor Rückwirkungen aus der Schaltung. Die Klasse X2 bedeutet, dass der Kondensator nach DIN IEC 60384-14 geprüft und zugelassen ist. Der Kondensator verträgt Impulsspitzenspannungen bis 2,5kV.

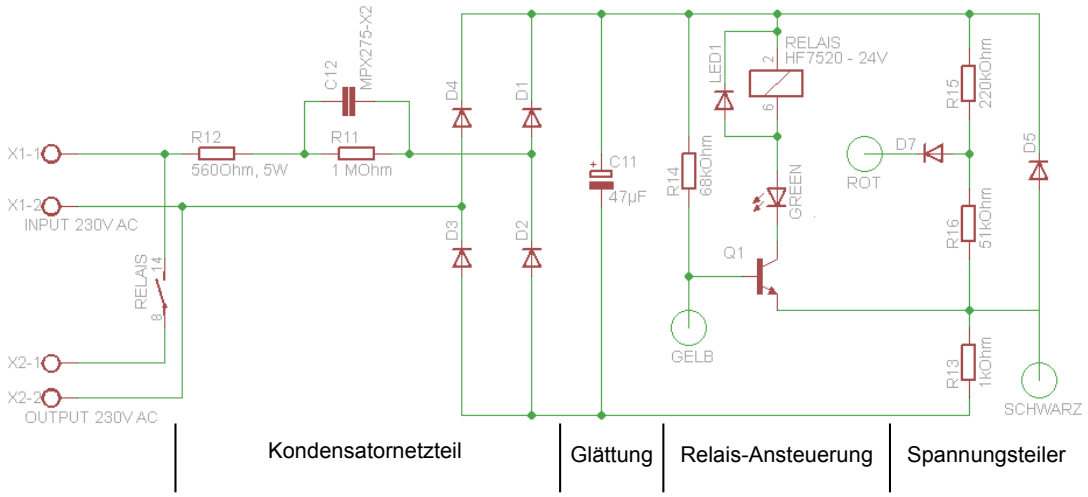

Abbildung 5.3: Schaltplan untere Platine

Die Schaltung kann in vier Bereiche unterteilt werden. Die Einteilung ist unterhalb des Schaltplans in Abbildung 5.3 dargestellt. Das Kondensatornetzteil dient dazu, aus der 230VAC Eingangsspannung eine Gleichspannung zu erzeugen. Die genaue Funktionsweise wird im nächsten Unterkapitel erläutert. Der Kondensator C11 ist für die Glättung der erzeugten Gleichspannung zuständig. Im dritten Bereich wird das Relais angesteuert. Je nach Eingangssignal, das die Steuerung über die gelbe Leitung vorgibt, wird über einen Transistor das Relais angesteuert. Bei einem Eingangssignal von logisch „0“ sperrt der Transistor und es kann kein Strom durch das Relais fließen. Bei einer logischen „1“ öffnet der Transistor und Strom fließt durch das Relais. Dadurch zieht das Relais an und schließt die 230VAC-Leitung zum Ausgang der Steckdose. Der elektrische Verbraucher an der Steckdose wird eingeschaltet. Dies wird durch eine grüne LED angezeigt. Der letzte Bereich ist ein Spannungsteiler für die Versorgungsspannung der Steuerung. Die Relais-Ansteuerung benötigt eine höhere Gleichspannung als die Steuerung auf der oberen Platine.

5.1.2 Kondensatornetzeil

Um aus der 230V Wechselspannung eine Gleichspannung zu erzeugen wird ein Kondensatornetzteil verwendet. Ein Kondensatornetzteil hat gegenüber eines klassischen Gleichrichters – bestehend aus Transformator und Brückengleichrichter – den Vorteil, dass die Bauteilekosten geringer sind und der Wirkungsgrad höher ist. Allerdings lässt sich mit einem Kondensatornetzteil nur ein geringer Ausgangsstrom erreichen. Außerdem gibt es keine galvanische Trennung.
Die Abbildung 5.4 zeigt eine typische Kondensatornetzteilschaltung. Die Funktionsweise ist identisch mit der Kondensatornetzteilschaltung der Steckdose. Die eigentliche Gleichrichtung übernehmen die vier Dioden D1 bis D4, die als sogenannter Brückengleichrichter verschaltet sind. Bei einer positiven Halbwelle an der Eingangsspannung Ue fließt der Strom durch die Bauteile C1, R1, D3, der Last bzw. C2 und D2. In diesem Fall sperren die Dioden D1 und D4. Bei einer negativen Halbwelle fließt der Strom durch D4, der Last bzw. C2, D1, R1 und C1. Hier sperren D2 und D3.
Der Widerstand R1 dient dazu den maximalen Strom zu begrenzen, damit die Dioden nicht überlastet werden. Der Kondensator C1 wirkt als kapazitiver Vorwiderstand. Dadurch wird die Verlustleistung reduziert. Der Widerstand R2 dient dazu, den Kondensator im Ruhezustand zu entladen.
Die Dimensionierung der Bauteile ist abhängig von verschiedenen Parametern. Die unten stehenden Formeln sind [Web:grosse] entnommen. Um den Strombegrenzungswiderstand auszuwählen, benötigt man die Eingangsspannung U_e und den maximal zulässigen Spitzenstrom durch die Dioden I_{zmax}. Dieser Wert muss aus dem Datenblatt der Dioden entnommen werden.

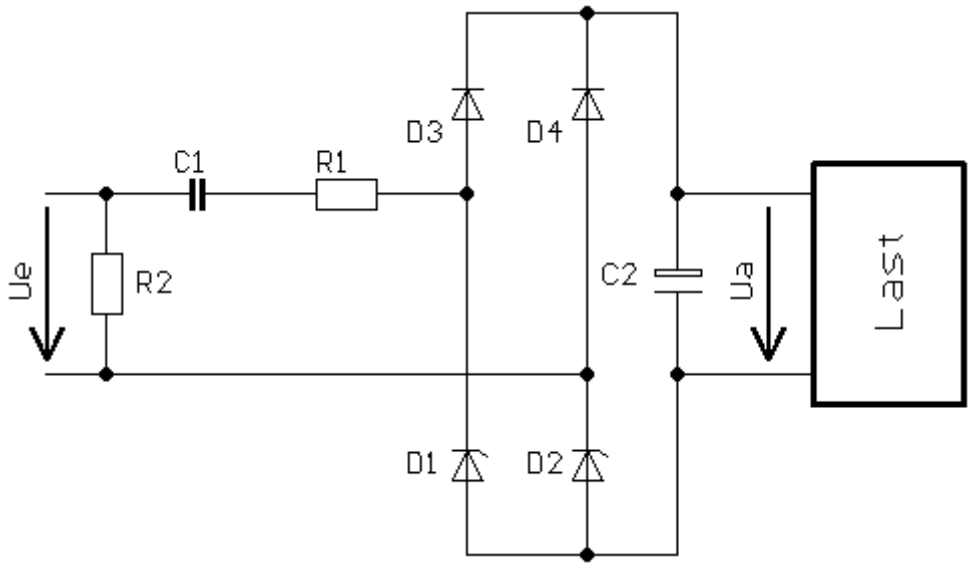

Abbildung 5.4: Kondensatornetzteil [Web:grosse]

$$\text{Strombegrenzungswiderstand:} \qquad R = \frac{U_e * \sqrt{2}}{I_{zmax}} \tag{5.1}$$

Für die Berechnung des kapazitiven Vorwiderstands wird der Ausgangsstrom I_a benötigt, der durch die Last fließen soll, und die Eingangsspannung U_e. In der Formel wird der Ausgangsstrom mit 1,1 multipliziert, weil dies dem Verhältnis zwischen Wechselstrom und Gleichstrom entspricht. Bei der Eingangsspannung wird davon ausgegangen, dass es maximal eine 10%ige Netzunterspannung geben könnte. Das Ergebnis entspricht der Kapazität, die mindestens notwendig ist. Gewählt wird ein Kondensator mit dem nächstgrößeren verfügbaren Wert.

$$\text{kapazitiver Vorwiderstand:} \qquad C = \frac{1}{2\pi f} * \frac{I_a * 1,1}{0,9 * U_e} \tag{5.2}$$

Der Entladewiderstand sollte möglichst groß gewählt werden. Je kleiner sein Wert ist, umso größer ist die Verlustleistung im normalen Betrieb. Je größer sein Wert ist, umso länger dauert der Entladevorgang des kapazitiven Widerstands im Ruhezustand. Ein typischer Wert ist 1MΩ.
Für die Berechnung des Glättungskondensators wird eine Restwelligkeit von 5% angenommen. Um diese auszugleichen, wird als Richtwert 10µF pro mA Ausgangsstrom angesetzt.

$$\text{Glättungskondensator:} \qquad C = I_a * 10\frac{\mu F}{mA} \tag{5.3}$$

In der Kondensatornetzteilschaltung der Steckdose entspricht der R12 dem Strombegrenzungswiderstand. Der C12 entspricht dem kapazitive Vorwiderstand und der R11 dem Entladewiderstand. Der C11 entspricht dem Glättungskondensator.

5.2 Zigbit-Modul

Abbildung 5.5 zeigt, dass das Zigbit-Modul ATZB-24-A2 aus drei Elementen besteht: einem Mikrocontroller ATmega1281, einem IEEE 802.15.4 Funkchip AT86RF230 und einer integrierten Dual Chip Antenne.
Der Mikrocontroller ATmega1281 ist ein 8bit AVR Mikrocontroller mit 128KByte Flash Speicher, 8KByte RAM Speicher und 4KByte EEPROM Speicher. Er wird mit einer 4MHz Frequenz betrieben.

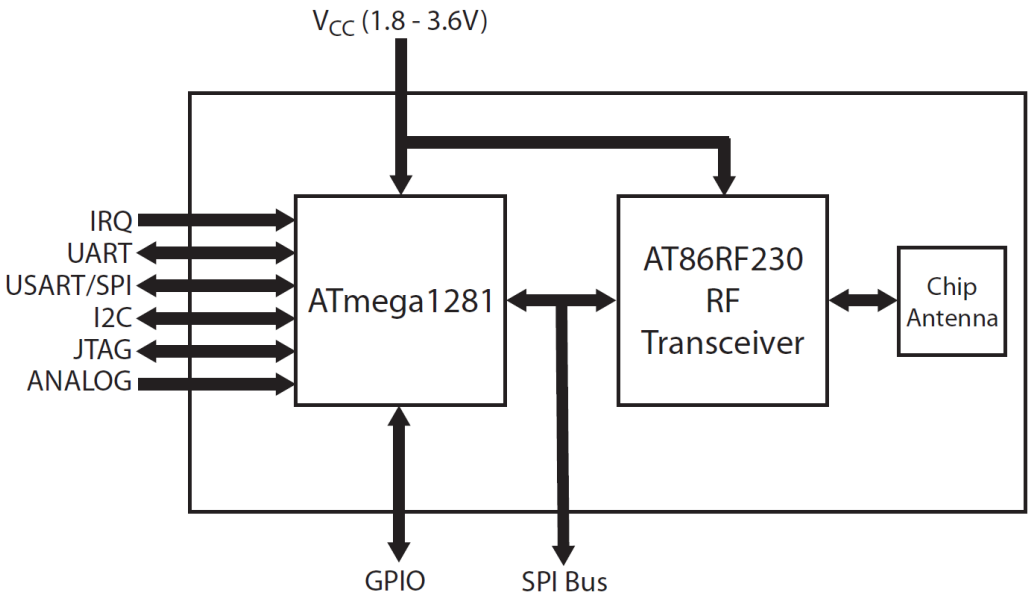

Abbildung 5.5: ATZB-24-A2 Block Diagramm [ATZB-24-A2]

Das Modul besitzt 42 Anschlusspins mit unterschiedlichen Funktionen. Zum Betrieb des Moduls müssen die Pins Vcc (Pin24 und Pin25), GND (Pin9, Pin22 und Pin23) und RESET (Pin8) verschaltet werden. Die Versorgungsspannung muss zwischen Vcc und GND angeschlossen werden, der RESET muss mit einem Pull-Up Widerstand von 100kΩ an Vcc verschaltet werden.
Zur Programmierung des Mikrocontrollers kann die JTAG-Schnittstelle verwendet werden. Die meisten Programmiergeräte verwenden für JTAG eine Steckerverbindung mit 10 Pins. In Tabelle 5.1 ist so eine Steckerverbindung aufgelistet. Es wird gezeigt, wie sie mit dem Zigbit-Modul verschaltet werden muss.

JTAG	Signal	ATZB-24-A2
1	JTAG_TCK	Pin29
2	GND	Pin9, Pin22, Pin23
3	JTAG_TDO	Pin28
4	Vcc	Pin24, Pin25
5	JTAG_TMS	Pin26
6	RESET	Pin8 (mit 100kΩ gegen Vcc)
7	N.C. (nicht verbunden)	-
8	N.C. (nicht verbunden)	-
9	JTAG_TDI	Pin27
10	GND	Pin9, Pin22, Pin23

Tabelle 5.1: JTAG Schnittstelle

Außerdem stehen verschiedene steuerbare Eingangs- und Ausgangspins des Mikrocontrollers zur Verfügung. Ein Pin davon wird später zur Ansteuerung des Relais in der Steckdose verwendet.
Zur Dimensionierung des Kondensatornetzteils wird der Energieverbrauch des Moduls benötigt. Laut Datenblatt fließt im Betrieb ein Eingangsstrom von ca. 18-19mA bei einer Betriebsspannung von 1,8V bis 3,6V.

5.3 Zigbit-Platine

Die Zigbit-Platine ersetzt die obere Platine in der existierenden Steckdose. In Abbildung 5.6 ist der zugehörige Schaltplan zu sehen. Die Hauptaufgabe der Zigbit-Platine ist es, die benötigten

Anschlusspins des Zigbit-Moduls über Steckerverbindungen abgreifbar zu machen. Es gibt eine Steckerverbindung, um über ein selbst gefertigtes Kabel ein JTAG-Programmiergerät anschließen zu können. Eine zweite Steckerverbindung mit 3 Pins ist für die Verbindung zur unteren Platine – der Steckdosen-Platine – zuständig. Hier werden die drei Leitungen (schwarz, rot und gelb) angeschlossen. Weil das Zigbit-Modul nur mit maximal 3,6V versorgt werden darf, ist ein Spannungsregler IC1 vom Typ LP2950CZ-3.0 [LP2950] davor geschaltet. Der Spannungsregler verkraftet Eingangsspannungen von 2,4V bis 30V und liefert eine konstante Ausgangsspannung von 3,0V.

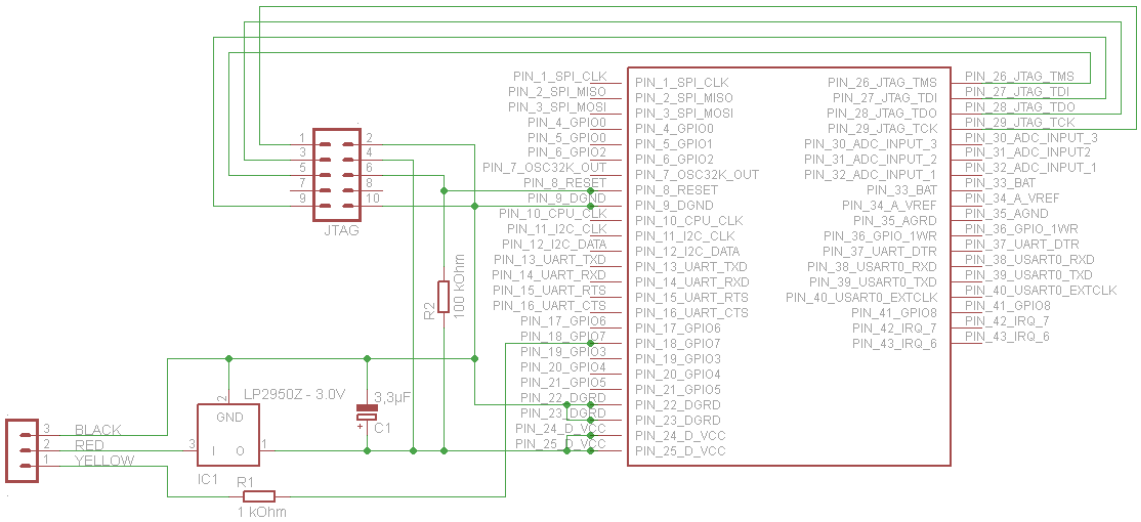

Abbildung 5.6: Schaltplan der Zigbit-Platine

So nah wie möglich an den Versorgungspins des Zigbit-Moduls befindet sich ein kleiner Kondensator C1 mit 3,3µF, um kleine Spannungsspitzen schnellstmöglich auszugleichen. An den RESET-Pin des Zigbit-Moduls ist ein Pull-Up Widerstand R2 mit 100kΩ verschaltet. Als Ansteuerung des Relais fungiert der Pin18 (GPIO7) des Zigbit-Moduls. Zwischen diesem Ausgangspin und der Steckerverbindung zur Steckdosen-Platine befindet sich ein Widerstand R1 mit 1kΩ. Dieser Widerstand wirkt als Basiswiderstand des Transistors auf der Steckdosen-Platine. Er ist aus Platzgründen auf der Zigbit-Platine platziert worden.

5.4 Steckdosen-Platine

Es wurde festgestellt, dass die Leistung des vorhandenen Kondensatornetzteils nicht ausreicht, das Zigbit-Modul vollständig zu versorgen. Ohne Last konnte eine Versorgungsspannung von 4V gemessen werden. Aber sobald das Zigbit-Modul angeschlossen wurde, fiel die Versorgungsspannung auf 0,8V ab. Die Kondensatornetzteilschaltung musste neu dimensioniert werden. Die Abbildung 5.7 zeigt den Schaltplan der Steckdosen-Platine.
Die Dioden in der Brückenschaltung wurden ersetzt durch Dioden vom Typ 1N4004 und durch Z-Dioden vom Typ ZPY24. Durch die Z-Dioden wird beim Gleichrichten der 230V Wechselspannung gleichzeitig die Spannung auf 24V begrenzt. Bei den Dioden wird von einem maximal zulässigen Spitzenstrom von 0,6A ausgegangen. Mit diesen Angaben ist es möglich, den Strombegrenzungswiderstand R12 zu berechnen.

$$R12 = \frac{U_e * \sqrt{2}}{I_{zmax}} = \frac{230V * \sqrt{2}}{0,6A} = 542.1\Omega \tag{5.4}$$

Es wird der nächstgrößere verfügbare Widerstand mit R12=560Ω gewählt. Aus Sicherheitsgründen wird ein Widerstand gewählt, der bis zu 5 Watt Leistung verträgt.

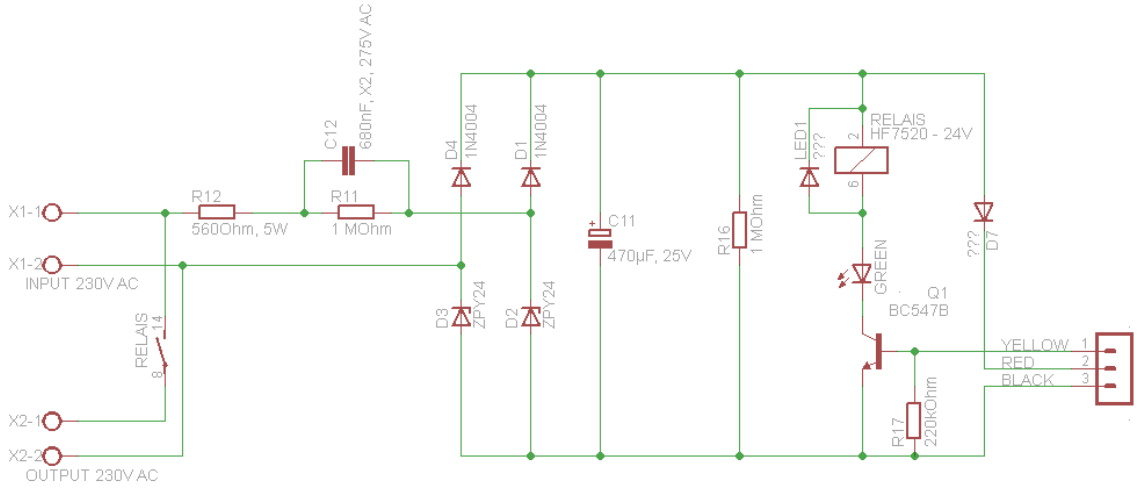

Abbildung 5.7: Schaltplan der Steckdosen-Platine

Um den kapazitiven Vorwiderstand C12 dimensionieren zu können, muss der maximale Ausgangsstrom bekannt sein. Durch Messungen wurde ermittelt, dass die Zigbit-Platine bei angelegten Spannungen von 3V bis 15V eine Stromaufnahme von 23,8mA hat. Es wird angenommen, dass die Stromaufnahme bei 24V ebenfalls 23,8mA beträgt. Dazu muss noch die Stromaufnahme der Relais-Ansteuerung addiert werden. Laut Datenblatt des Relais beträgt die Verlustleistung des Relais bis zu 200mW [HF7520]. Das entspricht bei 24V Spannungsversorgung einer Stromaufnahme von 8,4mA. Um noch Reserve zu berücksichtigen, wird von einem möglichen Ausgangsstrom I_a von 40mA ausgegangen.

$$C12 = \frac{1}{2\pi f} * \frac{I_a * 1,1}{0,9 * U_e} = \frac{1}{2 * \pi * 50Hz} * \frac{40mA * 1,1}{0,9 * 230V} = 677nF \tag{5.5}$$

Es wird der nächstgrößere verfügbare Entstörkondensator mit einer Nennspannung 230VAC und der Klasse X2 mit C12=680nF verwendet. Für den Kondensator wird ein Entladewiderstand R11 mit 1MΩ gewählt.
Der Glättungskondensator ist ebenfalls vom Ausgangsstrom I_a abhängig.

$$C11 = I_a * 10\frac{\mu F}{mA} = 40mA * 10\frac{\mu F}{mA} = 400\mu F \tag{5.6}$$

Es wird ein Elektrolytkondensator C11 mit 470µF gewählt. Um seine Entladung auch ohne angeschlossene Last zu gewährleisten, wird hier auch ein Entladewiderstand R16 mit 1MΩ gewählt.
Bei dem Abzweig für die Relais-Ansteuerung wurde der existierende Transistor ausgetauscht, weil keine Daten über ihn bekannt waren. Er wurde durch einen NPN-Transistor vom Typ BC547B [BC547] ersetzt. Zwischen Basis und Emitter wurde ein Pull-Down Widerstand R17 mit 220kΩ geschaltet, um den Transistor sicher zu sperren, falls sich der Mikrocontroller im Resetfall befindet und dann den Ausgangspin GPIO7 noch nicht ansteuert. Der Basiswiderstand für den Transistor befindet sich aus Platzgründen auf der Zigbit-Platine. Er lässt sich nach dem Ohmschen Gesetz berechnen. Die Spannung, die über den Basiswiderstand abfällt, ist die Spannung am Ausgangspin des Zigbit-Moduls minus die Steuerspannung, die am Transistor zwischen Basis und Emitter abfällt. Hier wird von einer Steuerspannung von 0,7V ausgegangen. Der Strom, der durch den Basiswiderstand fließt, ist abhängig vom Transistor-Typ und wird beim BC547B mit 2mA angenommen.

$$\text{Basiswiderstand} \qquad R1 = \frac{U - 0,7V}{I_b} = \frac{3V - 0,7V}{2mA} = 1,15k\Omega \tag{5.7}$$

Es wird ein Basiswiderstand R1 von 1kΩ gewählt.

5.5 Test und Fertigstellung der Hardware

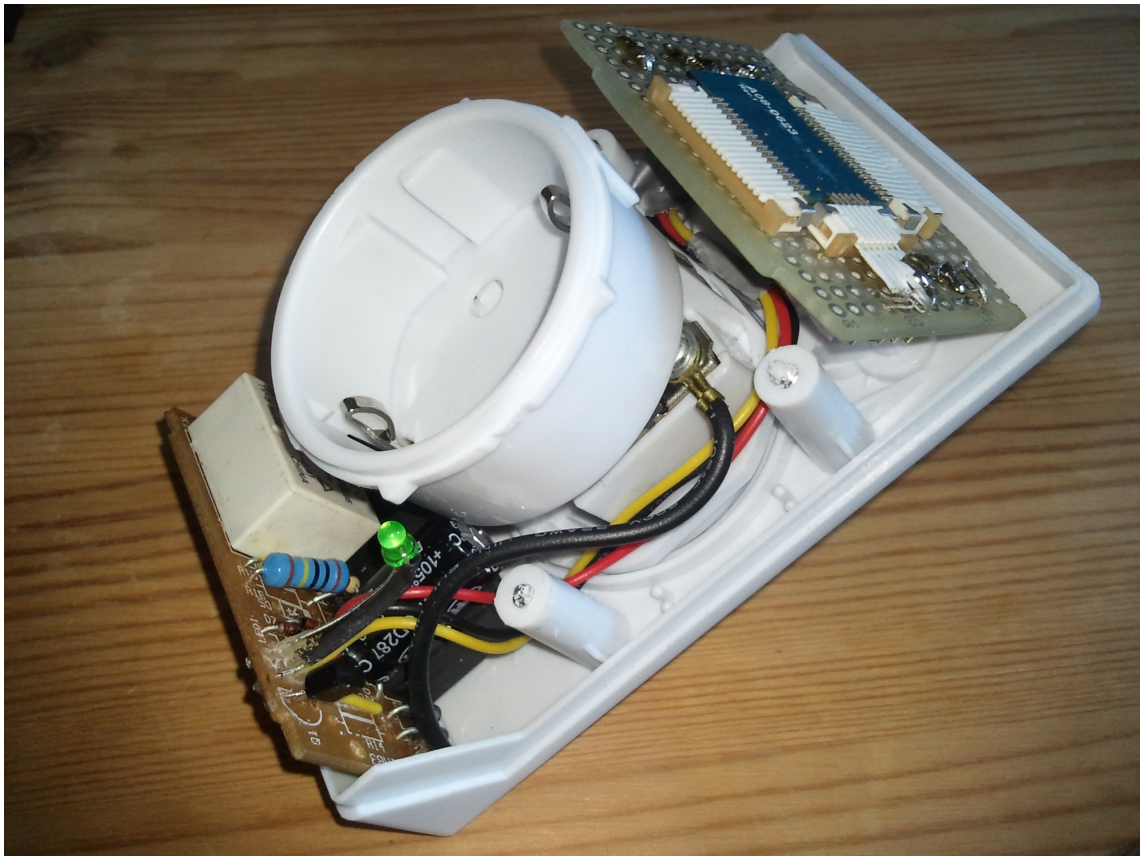

Abbildung 5.8: Zusammengebaute Steckdose ohne Deckel

Es wurde getestet, dass die Steckdosen-Platine genug Leistung liefert, um die Zigbit-Platine zu versorgen. Ebenfalls ist es möglich, von der Zigbit-Platine das Relais auf der Steckdosen-Platine anzusteuern. Nach dem Test konnten beide Platinen in das Steckdosen-Gehäuse eingebaut werden (siehe Abbildung 5.8). Nachdem das fertig implementierte Programm auf das Zigbit-Modul geladen wurde, wurde das Gehäuse verschlossen und verschraubt. Danach war die Steckdose einsatzfähig. Die Implementierung des Programms wird im nächsten Kapitel behandelt.

Kapitel 6

Implementierung der Software

Parallel zur Fertigstellung der Hardware wurde die Software entwickelt. Dabei wurde der vorhandene Contiki-Webserver webserver-nano als Vorlage genommen und an die Anforderungen der Steckdose angepasst. Es wurde eine eigene Contiki-Applikation iec104 entwickelt. Beide Applikationen wurden gemeinsam in einem Projekt kompiliert. Damit laufen sie parallel in einem Programm als eigenständige Prozesse.
Auf die Implementierung einer SNMP-Applikation soll hier nicht weiter eingegangen werden. Sie wurde bereits detailliert in [Zehl 2012] behandelt.

6.1 Webserver

Um die Anpassungen in dem Contiki-Quellcode vornehmen zu können, muss zuerst die Arbeitsweise des Contiki-Webservers verstanden werden. Vorlage dabei ist die webserver-nano Applikation von Contiki in der Version 2.6.

6.1.1 webserver-nano

Alle Quellcode-Dateien befinden sich in der Contiki-Verzeichnisstruktur im Unterverzeichnis apps/webserver-nano. Dort befindet sich auch die Datei Makefile.webserver-nano, die die notwendigen Informationen für die Kompilierung enthält. Im Makefile sind die Quellcode-Dateien aufgelistet, die für die Applikation übersetzt werden müssen. In der Datei webserver-nogui.c ist der Contiki-Prozess beschrieben.

Listing 6.1: Auszug aus apps/webserver-nano/webserver-nogui.c

```
PROCESS(webserver_nogui_process, "Web server");
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(webserver_nogui_process, ev, data)
{
  PROCESS_BEGIN();
  httpd_init();
  while(1) {
    PROCESS_WAIT_EVENT_UNTIL(ev == tcpip_event);
    httpd_appcall(data);
  }
  PROCESS_END();
}
```

In der ersten Zeile wird der Prozess definiert mit einem internen Namen und einem lesbaren Namen, hier „Web server“. Im unteren Teil wird der Ablauf des Prozesses beschrieben. Nach dem Start wird httpd_init() aufgerufen, um die Applikation zu initialisieren. Der Speicher wird initialisiert und der TCP-Port 80 wird geöffnet für mögliche eingehende Verbindungen, er wird in den Modus LISTEN gesetzt. Danach wartet der Prozess in einer Endlosschleife auf TCP/IP-Events

und ruft in diesem Fall die Funktion httpd_appcall() auf. Diese Funktion ist in der Datei httpd.c definiert.

Listing 6.2: Funktion httpd_appcall() aus apps/webserver-nano/httpd.c

```
void
httpd_appcall(void *state)
{
  struct httpd_state *s = (struct httpd_state *)state;
  if(uip_closed() || uip_aborted() || uip_timedout()) {
    /*~~Speicher freigeben~~*/
  } else if(uip_connected()) {
    /*~~Speicher allokieren~~*/
    /*~~Verbindung initialisieren~~*/
    handle_connection(s);
  } else if(s != NULL) {
    /*~~Timeout prüfen, wenn Verbindung im Ruhezustand ist~~*/
    handle_connection(s);
  } else {
    uip_abort();
  }
}
```

Um die Lesbarkeit der Funktion zu verbessern, wurden Teile des Quellcodes durch Kommentare ersetzt, die die dortige Programmierung zusammenfassen. Alle Informationen über eine TCP/IP-Verbindung wird in der Struktur struct httpd_state *s gespeichert. Ein Pointer auf diese Struktur wird an weitere Funktionen übergeben. Die Funktion httpd_appcall() reagiert auf Verbindungsabbrüche und Timeouts. Bei einem Verbindungsaufbau oder bei dem Empfang von Daten wird die Funktion handle_connection() aufgerufen.

Listing 6.3: Funktion handle_connection() aus apps/webserver-nano/httpd.c

```
static void
handle_connection(struct httpd_state *s)
{
  handle_input(s);
  if(s->state == STATE_OUTPUT) {
    handle_output(s);
  }
}
```

Es wird die Funktion handle_input() aufgerufen, um eingehende Daten zu verarbeiten. Wenn ausgehende Daten anstehen, wird die Funktion handle_output() aufgerufen.

Listing 6.4: Funktion handle_input() aus apps/webserver-nano/httpd.c

```
const char httpd_get[] HTTPD_STRING_ATTR = "GET ";
static
PT_THREAD(handle_input(struct httpd_state *s))
{
  PSOCK_BEGIN(&s->sin);
  PSOCK_READTO(&s->sin, ISO_space);

  if(httpd_strncmp(s->inputbuf, httpd_get, 4) != 0) {
    PSOCK_CLOSE_EXIT(&s->sin);
  }
  PSOCK_READTO(&s->sin, ISO_space);

  if(s->inputbuf[0] != ISO_slash) {
    PSOCK_CLOSE_EXIT(&s->sin);
  }

  if(s->inputbuf[1] == ISO_space) {
    httpd_strcpy(s->filename, httpd_indexfn);
```

```
  } else {
    uint8_t i;
    for (i=0;i<sizeof(s->filename)+1;i++) {
      if (i >= (PSOCK_DATALEN(&s->sin)-1)) break;
      if (s->inputbuf[i]==ISO_space) break;
     /* Query string is left in the httpd_query buffer
        until zeroed by the application! */
      if (s->inputbuf[i]==ISO_qmark) {
         strncpy(httpd_query,&s->inputbuf[i+1],sizeof(httpd_query));
         break;
      }
      s->filename[i]=s->inputbuf[i];
    }
    s->filename[i]=0;
  }

  s->state = STATE_OUTPUT;
  while(1) {
    PSOCK_READTO(&s->sin, ISO_nl);
  }
  PSOCK_END(&s->sin);
}
```

Die Funktion handle_input() ist als Protothread definiert. Das bedeutet, dass sie als eigener Thread läuft, der von anderen Prozessen unterbrochen werden kann. Die Aufgabe dieser Funktion ist es, die empfangene HTTP-GET-Abfrage zu dekodieren. Wenn die Abfrage anders ist als erwartet, wird die Verbindung abgebrochen (PSOCK_CLOSE_EXIT). Eine mögliche HTTP-GET-Abfrage könnte wie folgt aussehen:

Listing 6.5: HTTP-GET-Abfrage

```
GET /pins.shtml?pin18=ein HTTP/1.1\r\n
```

Die Abfrage wird ausgelesen. Dabei wird der abgefragte Dateiname in die Variable s->filename gespeichert. Wenn kein Dateiname angegeben ist, wird der Index-Dateiname „/index.html“ in diese Variable gespeichert. Es ist möglich, zusätzliche Parameter mit zu übergeben. Solche Parameter sind in der GET-Abfrage durch ein Fragezeichen vom Dateinamen getrennt und werden in der Variable httpd_query gespeichert. Wenn die GET-Abfrage gültig ist, werden also die beiden Variablen s->filename und httpd_query gesetzt und zusätzlich wird der Status s->state auf STATE_OUTPUT gesetzt. Das bedeutet, dass nun die Funktion handle_output() von der Funktion handle_connection() aufgerufen wird.

Listing 6.6: Funktion handle_output() aus apps/webserver-nano/httpd.c

```
const char httpd_indexfn [] HTTPD_STRING_ATTR = "/index.html";
const char httpd_indexsfn [] HTTPD_STRING_ATTR = "/index.shtml";
const char httpd_404fn   [] HTTPD_STRING_ATTR = "/404.html";
const char httpd_404notf [] HTTPD_STRING_ATTR = "404 Not found";
const char httpd_200ok   [] HTTPD_STRING_ATTR = "200 OK";
static
PT_THREAD(handle_output(struct httpd_state *s))
{
  char *ptr;

  PT_BEGIN(&s->outputpt);
  if(!httpd_fs_open(s->filename, &s->file)) {
    /* If index.html not found try index.shtml */
    if (httpd_strcmp(s->filename,httpd_indexfn)==0)
        httpd_strcpy(s->filename,httpd_indexsfn);
    if (httpd_fs_open(s->filename, &s->file)) goto sendfile;
    httpd_strcpy(s->filename, httpd_404fn);
    httpd_fs_open(s->filename, &s->file);
```

```
    PT_WAIT_THREAD(&s->outputpt, send_headers(s, httpd_404notf));
    PT_WAIT_THREAD(&s->outputpt, send_file(s));
  } else {
sendfile:
    PT_WAIT_THREAD(&s->outputpt, send_headers(s, httpd_200ok));
    ptr = strchr(s->filename, ISO_period);
    if((ptr != NULL && httpd_strncmp(ptr, httpd_shtml, 6) == 0)
       || httpd_strcmp(s->filename,httpd_indexfn)==0) {
      PT_INIT(&s->scriptpt);
      PT_WAIT_THREAD(&s->outputpt, handle_script(s));
    } else {
      PT_WAIT_THREAD(&s->outputpt, send_file(s));
    }
  }
  PSOCK_CLOSE(&s->sout);
  PT_END(&s->outputpt);
}
```

Die Funktion handle_output() ist ebenfalls als Protothread beschrieben. Diese Funktion sendet die Antwort auf die HTTP-GET-Abfrage zurück. Zuerst wird geprüft, ob eine Datei mit dem Namen aus der Variable s->filename vorhanden ist. Wenn dies nicht der Fall ist, wird ein „404 - file not found“-Fehler gesendet. Wenn die Datei vorhanden ist, wird der HTTP-Status-Header mit dem Text „200 OK“ aufgebaut. Danach wird geprüft, ob der Dateiname mit „.shtml“ endet. In diesem Fall wird die Funktion handle_script() aufgerufen, um die Antwort zurückzusenden, ansonsten die Funktion send_file(). Letztere Funktion sendet den Inhalt einer Datei zurück, ohne ihn zu bearbeiten. Auf diese Funktion soll hier nicht näher eingegangen werden.

Listing 6.7: Funktion handle_script() aus apps/webserver-nano/httpd.c

```
static
PT_THREAD(handle_script(struct httpd_state *s))
{
  static char scriptname[MAX_SCRIPT_NAME_LENGTH+1],*pptr;
  static uint16_t filelength;

  PT_BEGIN(&s->scriptpt);

  filelength=s->file.len;
  while(s->file.len > 0) {
    /* Check if we should start executing a script, flagged by %! */
    if(httpd_fs_getchar(s->file.data) == ISO_percent &&
       httpd_fs_getchar(s->file.data + 1) == ISO_bang) {

       s->scriptptr=get_scriptname(scriptname,s->file.data+2);
       s->scriptlen=s->file.len-(s->scriptptr-s->file.data);
       PT_WAIT_THREAD(&s->scriptpt,httpd_cgi(scriptname)(s, s->scriptptr));
       next_scriptstate(s);

      /* Reset the pointers and continue sending the current file. */
      s->file.data = s->scriptptr;
      s->file.len  = s->scriptlen;
    } else {
      /*~~Sende Inhalt der Datei bis zum nächsten Prozentzeichen~~*/
    }
  }

  PT_END(&s->scriptpt);
}
```

Die Funktion handle_script() geht den Inhalt der Datei durch. Sobald ein „%!“ gefunden wird, wird der zugehörige Skriptname ermittelt. Es wird dann die Funktion httpd_cgi() mit dem Skriptnamen als Parameter ausgeführt, um einen Teil der Ausgabe zu erstellen. Der Rest des Inhalts der

Datei wird ohne Bearbeitung gesendet. Das Verhalten entspricht dem von Server Side Includes (SSI). Es sind einige Skriptnamen bzw. Kommandos in der Quellcode-Datei httpd-cgi.c definiert. Wenn so ein Kommando mit vorangestellten „%!“ in der aufgerufenen .shtml-Datei steht, wird das zugehörige Skript auf Serverseite – also auf dem Mikrocontroller – ausgeführt. Die Skripte erstellen in der Regel eine Ausgabe in HTML-Format.

Listing 6.8: apps/webserver-nano/httpd-fs/status.shtml

```
%! header
<pre><big><b>Addresses</b></big>
%! addresses
<big><b>Neighbors</b></big>
%! neighbors
<big><b>Routes</b></big>
%! routes
<big><b>Sensors</b></big>
%! sensors
</pre>
%! file-stats .
```

In der Datei status.shtml werden einige Skripte verwendet. So liefert zum Beispiel das Skript „%! addresses“ die eigenen IP-Adressen. Das Skript „%! neighbors“ liefert die IP-Adressen benachbarter Geräte. Diese Informationen werden während der Laufzeit ausgelesen und entsprechend formatiert zurückgeliefert. In der Quellcode-Datei httpd-cgi.c werden der jeweilige Skriptname – also das Schlüsselwort – und die Funktion, die aufgerufen werden soll, definiert. Es soll hier am Beispiel von dem Schlüsselwort „sensors“ verdeutlicht werden.

Listing 6.9: Definition des Schlüsselwortes „sensors“ aus apps/webserver-nano/httpd-cgi.c

```
static const char sensor_name[] HTTPD_STRING_ATTR = "sensors";

HTTPD_CGI_CALL(sensors, sensor_name, sensor_readings);

void
httpd_cgi_init(void)
{
  ...
  httpd_cgi_add(&sensors);
  ...
}

static
PT_THREAD(sensor_readings(struct httpd_state *s, char *ptr))
{
  PSOCK_BEGIN(&s->sout);

  PSOCK_GENERATOR_SEND(&s->sout, generate_sensor_readings, s);
#if WEBSERVER_CONF_STATISTICS
  PSOCK_GENERATOR_SEND(&s->sout, generate_stats, s);
#endif

  PSOCK_END(&s->sout);
}
```

In der ersten Zeile wird das Schlüsselwort „sensors“ definiert und der Konstanten „sensor_name“ zugewiesen. Mit der Definition von HTTPD_CGI_CALL() und dem Aufruf httpd_cgi_add() während der Initialisierung, wird diesem Schlüsselwort die Funktion sensor_readings() zugewiesen. Sobald in einer .shtml-Datei dieses Schlüsselwort vorkommt mit einem „%!“ als Erkennung, wird die Funktion sensor_readings() aufgerufen.

Listing 6.10: Funktion httpd_cgi() aus apps/webserver-nano/httpd-cgi.c

```
httpd_cgifunction
httpd_cgi(char *name)
{
  struct httpd_cgi_call *f;
  /* Find the matching name in the table, return the function. */
  for(f = calls; f != NULL; f = f->next) {
    if(httpd_strncmp(name, f->name, httpd_strlen(f->name)) == 0) {
      return f->function;
    }
  }
  return nullfunction;
}
```

Die Funktion httpd_cgi(), die ja in der Funktion handle_script() aufgerufen wird, durchsucht alle definierten Schlüsselworte und ruft bei einer Übereinstimmung die zugehörige Funktion auf.

Listing 6.11: im Webbrowser empfangene Daten beim Aufruf der Datei status.shtml

```
<html><head><title>[2679]</title></head><body><pre><a href="/">Front page</a>|
<a href="status.shtml">Status</a>|<a href="tcp.shtml">Network connections</a>|
<a href="processes.shtml">System processes</a>|
<a href="files.shtml">File statistics</a>|
<a href="/ttt/ttt.shtml">TicTacToe</a>|
<a href="ajax.shtml">Ajax</a></pre><pre><big><b>Addresses</b></big>
2a01:1e8:e115::200:14:2679
fe80::200:14:2679
<big><b>Neighbors</b></big>
2a01:1e8:e115::2
fe80::12:13ff:fe14:1516
<big><b>Routes</b></big>
[None]
<big><b>Sensors</b></big>
<em>Battery    :</em> Not Available
<em>Uptime    :</em> 116 days 12:59:46
<big><b>Statistics</b></big>
</pre>
<p align="right"><br><br><i>This page has been sent 1 times (0.01 sec)</i>
</body></html>
```

Die HTTP-Antwort besteht dann zum Teil aus dem Inhalt der .shtml-Datei und zum Teil aus den Ergebnissen der Skripte. Im Webbrowser ist nur das Endergebnis zu sehen.

Abbildung 6.1 verdeutlicht beispielhaft, welche C-Funktionen durchlaufen werden, wenn eine Abfrage der Datei status.shtml verarbeitet wird. Es wird dabei von folgender HTTP-GET-Abfrage ausgegangen:

```
GET /status.shtml HTTP/1.1\r\n
```

Die empfangene HTTP-GET-Abfrage löst einen TCP/IP-Event im Prozess webserver_nogui_process aus. Dort wird die Funktion httpd_appcall() aufgerufen, die wiederum die Funktion handle_connection() aufruft. Die Funktion handle_input() dekodiert die GET-Abfrage und schreibt die angefragte Datei – in unserem Fall status.shtml – in die Variable s->filename. Die Funktion handle_output() baut den HTTP-Header für die Antwort auf und ruft die Funktion handle_script() auf. Hier wird der statische Teil der Datei verarbeitet, also zur HTTP-Antwort hinzugefügt. Für den dynamischen Teil, also die definierten Schlüsselworte, wird die Funktion httpd_cgi() aufgerufen, die wiederum die jeweilige Skript-Funktion aufruft. Diese Skript-Funktionen fügen ihren Teil der HTTP-Antwort hinzu.

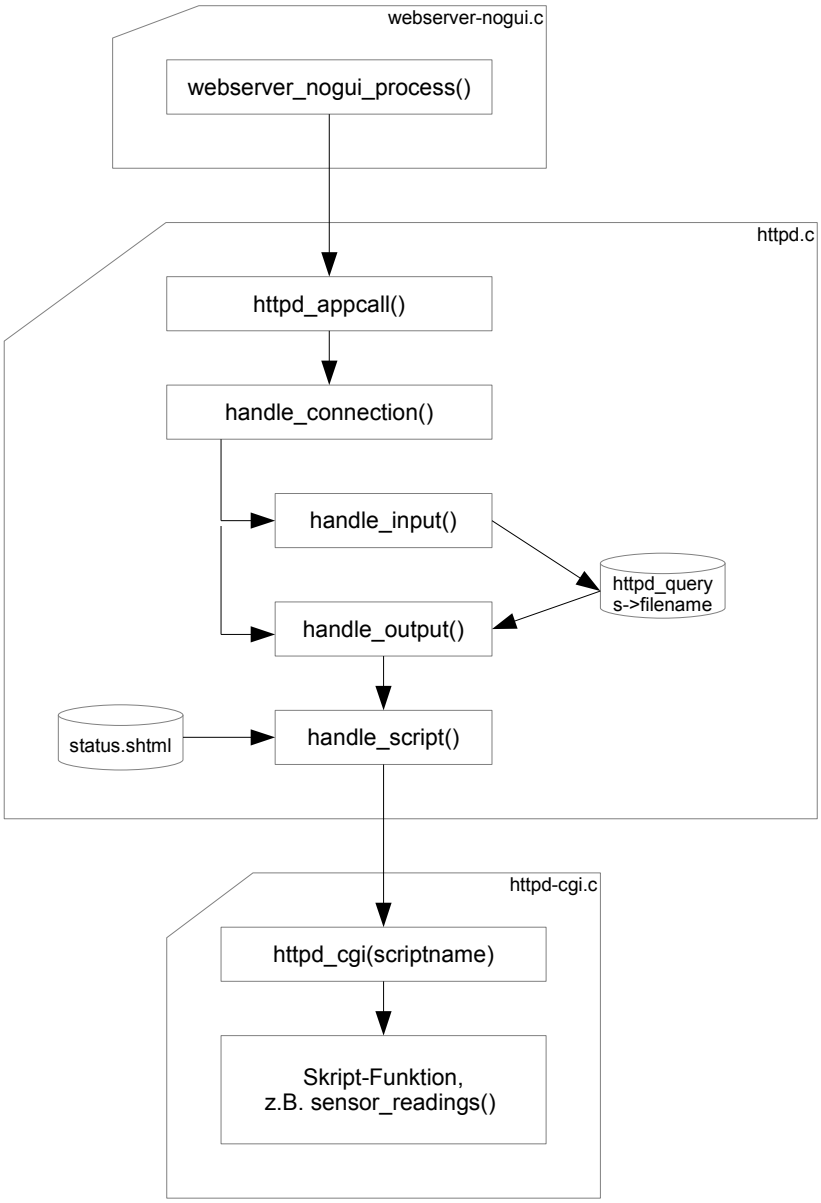

Abbildung 6.1: Durchlauf der Funktionen bei Abfrage von status.shtml

6.1.2 webserver-nano-steckdose

Um die Steckdose ansteuern zu können, müssen im Webserver zusätzliche Skript-Funktionen implementiert werden. Dazu sind verschiedene Anpassungen notwendig. Um die existierende Applikation webserver-nano nicht zu verändern, wurde eine neue Applikation webserver-nano-steckdose erstellt, die eine Kopie von webserver-nano ist. Zuerst wurden alle Quellcode-Dateien aus dem Verzeichnis apps/webserver-nano in das neue Verzeichnis apps/webserver-nano-steckdose kopiert. Das Makefile wurde entsprechend umbenannt.

```
mv Makefile.webserver-nano Makefile.webserver-nano-steckdose
```

Verschiedene Funktionen des Webservers lassen sich über Präprozessoranweisungen in der Datei httpd.h ein- oder ausschalten. Dabei gibt es drei Beispielkonfigurationen vom minimalen Funktionsumfang bis hin zum kompletten Funktionsumfang. Es wurde auf die Konfiguration des minimalen Funktionsumfang aufgesetzt. Diese wurde für die Steckdosen-Steuerung angepasst.

Listing 6.12: Auszug aus apps/webserver-nano-steckdose/httpd.h

```
// fschw: 120424 NAMESIZE 16->20, because of processes.shtml
#define WEBSERVER_CONF_NAMESIZE 20
// fschw 120424: enable TCPSTATS, PROCESSES and SENSORS for minimum webserver
#define WEBSERVER_CONF_TCPSTATS  1
#define WEBSERVER_CONF_PROCESSES 1
#define WEBSERVER_CONF_SENSORS   1
// fschw 120424: add WEBSERVER_CONF_STECKDOSE
#define WEBSERVER_CONF_STECKDOSE 1
// fschw 120424: enable WEBSERVER_CONF_PASSQUERY
#define WEBSERVER_CONF_PASSQUERY 10
```

Es wurden zusätzlich die TCP/IP-Statistiken, die Übersicht der Prozesse und die Sensoren für die Darstellung der Laufzeit eingeschaltet. Es wurde eine neue Präprozessoranweisung für die Steckdose WEBSERVER_CONF_STECKDOSE eingeführt und eingeschaltet. Das Ausschalten dieses Präprozessors würde die Änderungen in der Datei httpd-cgi.c zurücknehmen.

Listing 6.13: Neues Schlüsselwortes „steckdose“ in apps/webserver-nano-steckdose/httpd-cgi.c

```
// fschw: 120424: add Name for Steckdose
#if WEBSERVER_CONF_STECKDOSE
static const char   pins_name[] HTTPD_STRING_ATTR = "steckdose";
#endif

// fschw: 120424: add HTTPD_CGI_CALL for Steckdose
#if WEBSERVER_CONF_STECKDOSE
HTTPD_CGI_CALL( pins, pins_name, pins_call          );
#endif

void
httpd_cgi_init(void)
{
  ...
// fschw: 120424: add httpd_cgi_add for Steckdose
#if  WEBSERVER_CONF_STECKDOSE
  httpd_cgi_add( &pins);
#endif
  ...
}
```

Es wurde ein neues Schlüsselwort „steckdose“ eingeführt. Bei Aufruf dieses Schlüsselwortes in einer .shtml-Datei, wird die Funktion pins_call() aufgerufen.

Listing 6.14: Neue Funktionen in apps/webserver-nano-steckdose/httpd-cgi.c

```
// fschw: 120424 add WEBSERVER_CONF_STECKDOSE Part
#if WEBSERVER_CONF_STECKDOSE
/*---------------------------------------------------------------------------*/
static unsigned short
generate_pins_reading(void *arg)
{
  static const char httpd_cgi_pins_ein[] HTTPD_STRING_ATTR
         = "<td>Steckdose</td><td>EIN</td>";
  static const char httpd_cgi_pins_aus[] HTTPD_STRING_ATTR
         = "<td>Steckdose</td><td>AUS</td>";

  if (PORTD & (1 << PIN7)) {
    return httpd_snprintf((char *)uip_appdata, uip_mss(), httpd_cgi_pins_ein);
  } else {
    return httpd_snprintf((char *)uip_appdata, uip_mss(), httpd_cgi_pins_aus);
  }
}
/*---------------------------------------------------------------------------*/
```

```
static
PT_THREAD(pins_call(struct httpd_state *s, char *ptr))
{
  static const char httpd_cgi_pins_steckdose_ein[] HTTPD_STRING_ATTR
         = "pin18=ein";
  static const char httpd_cgi_pins_steckdose_aus[] HTTPD_STRING_ATTR
         = "pin18=aus";

  PSOCK_BEGIN(&s->sout);
  if (httpd_strncmp(httpd_query, httpd_cgi_pins_steckdose_ein, 9) == 0) {
    PORTD |=  (1 << PIN7);
  } else if (httpd_strncmp(httpd_query, httpd_cgi_pins_steckdose_aus, 9) == 0){
    PORTD &= ~(1 << PIN7);
  }
  httpd_query[0] = 0;
  PSOCK_GENERATOR_SEND(&s->sout, generate_pins_reading, s);
  PSOCK_END(&s->sout);
}
#endif /* WEBSERVER_CONF_STECKDOSE */
```

In der Funktion pins_call() wird zuerst die Variable httpd_query abgefragt. Wenn sie mit „pin18=ein“ identisch ist, wird der Ausgangspin vom Zigbit-Modul zum Relais auf logisch 1 gesetzt. Wenn sie mit „pin18=aus“ identisch ist, wird der Ausgangspin auf logisch 0 gesetzt. Jeder andere Wert wird ignoriert. Zum Schluss wird immer die Funktion generate_pins_reading aufgerufen, die den aktuellen Zustand des Ausgangspins abfragt und das Ergebnis im HTML-Format zurück gibt.

Listing 6.15: Neue Datei apps/webserver-nano-steckdose/httpd-fs/pins.shtml

```
%! header.html<h1>6LoWPAN Steckdose</h1><br><br>
<table width="100%"><tr><th>PIN</th><th>Status</th></tr>
<tr align="center">%! steckdose
<td><form action="pins.shtml" method="get" align="right">
<input type="hidden" name="pin18" value="ein">
<input type="submit" value="Einschalten"></form></td>
<td><form action="pins.shtml" method="get" align="right">
<input type="hidden" name="pin18" value="aus">
<input type="submit" value="Ausschalten"></form></td>
</tr></table>%! file-stats .
```

Es wurde eine neue Datei pins.shtml erstellt, die das neue Skript „%! steckdose“ aufruft und damit den aktuellen Zustand der Steckdose tabellarisch anzeigt. Außerdem gibt es zwei Schaltflächen, um die Steckdose ein- bzw. ausschalten zu können. Abbildung 6.2 zeigt die Darstellung der Datei pins.shtml im Webbrowser.

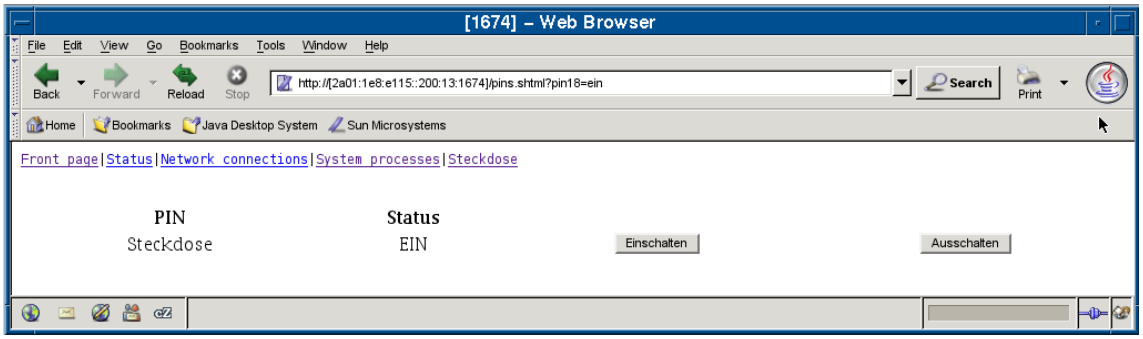

Abbildung 6.2: Anzeige von pins.shtml im Webbrowser

Damit ist die Applikation webserver-nano-steckdose erstellt. Sie kann in einem Projekt im zugehörigen Makefile mithilfe der Variable APPS eingebunden werden.

6.2 IEC104-Slave

Es wurde eine eigene Applikation iec104 entwickelt. Über diese Applikation kann mittels des Protokolls IEC 60870-5-104 auf die Steuerung zugegriffen werden. Die Steuerung wirkt dabei als IEC104-Slave. Es wurde ein neues Verzeichnis apps/iec104 erstellt. Dort befinden sich alle Quellcode-Dateien dieser Applikation. Sie kann in einem Projekt mithilfe der Variable APPS eingebunden werden. Der Name der Applikation ist „iec104“.
Die Applikation ist in fünf Module unterteilt:

- iec104 (Hauptmodul)
- iec104-buffer
- iec104-para
- iec104-appl
- iec104-link

Das Hauptmodul beschreibt den Contiki-Prozess und allgemeine Parameter. Das Modul iec104-buffer enthält einen eigenen FIFO-Puffer, der von anderen Modulen verwendet werden kann. Das Modul iec104-para übernimmt die IEC104-Parametrierung der Applikation. Wenn eine IEC104-Adresse geändert, eine neue Information hinzugefügt oder eine neue Verarbeitung implementiert werden muss, kann das in diesem Modul gemacht werden, ohne in die Funktionsweise des Protokolls eingreifen zu müssen. Das Modul iec104-appl übernimmt die Applikationsschicht und das Modul iec104-link die Linkschicht des Protokolls IEC104.

6.2.1 iec104

Das Hauptmodul besteht aus den Quellcode-Dateien iec104.h, iec104.c und iec104-codec.h. In der letzten Datei sind mehrere protokollspezifische Konstanten definiert. Diese Konstanten, z.B. der Wert des Startbyte in einem IEC104-Telegramm IEC_START=0x68, sind unabhängig von der Implementierung.

Listing 6.16: Auszug aus apps/iec104/iec104.c

```
PROCESS(iec104_process, "IEC60870-5-104 Slave");
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(iec104_process, ev, data)
{
        PROCESS_BEGIN();
        clock_init();
        etimer_set(&checktimer_iec104appl, CLOCK_SECOND * APPLICATION_TIMER);
        iec104link_init(&myIEC104State);
        iec104para_init();
        while(1) {
                PROCESS_WAIT_EVENT();

                if(ev == tcpip_event) {
                        iec104link_appcall(&myIEC104State);
                }
                if (etimer_expired(&checktimer_iec104appl)) {
                        iec104appl_appcall(&myIEC104State);
                        etimer_restart(&checktimer_iec104appl);
                }
        }
        PROCESS_END();
}
```

In der Datei iec104.c wird der Contiki-Prozess definiert. Ihm wird der lesbare Name „IEC60870-5-104 Slave“ zugewiesen. Die Aufgabe des Prozesses ist es, die Initialisierung von dem Modul

iec104-link vorzunehmen. Hier wird zum Beispiel der TCP-Port 2404 in den Modus LISTEN gesetzt. Ebenfalls wird ein Timer für das Modul iec104-appl mit zwei Sekunden Laufzeit initialisiert. Nach der Initialisierung läuft der Prozess in einer Endlosschleife und wartet auf Contiki-Events. Bei einem TCP/IP-Event wird die Funktion iec104link_appcall() aus dem Modul iec104-link aufgerufen. Alle zwei Sekunden (APPLICATION_TIMER) wird die Funktion iec104appl_appcall() aus dem Modul iec104-appl aufgerufen.

6.2.2 iec104-buffer

Dieses Modul besteht aus den Quellcode-Dateien iec104-buffer.h und iec104-buffer.c. Es wird vom Modul iec104-appl verwendet, um IEC104-Pakete (iFrames) zu speichern, bevor die Pakete von dem Modul iec104-link gesendet werden. Der Puffer ist notwendig, um mehrere iFrames in einem Telegrammblock übertragen zu können. iFrames werden erst aus dem Puffer gelöscht, wenn ihr Empfang von der Gegenstelle über die Sequenznummer bestätigt worden ist.

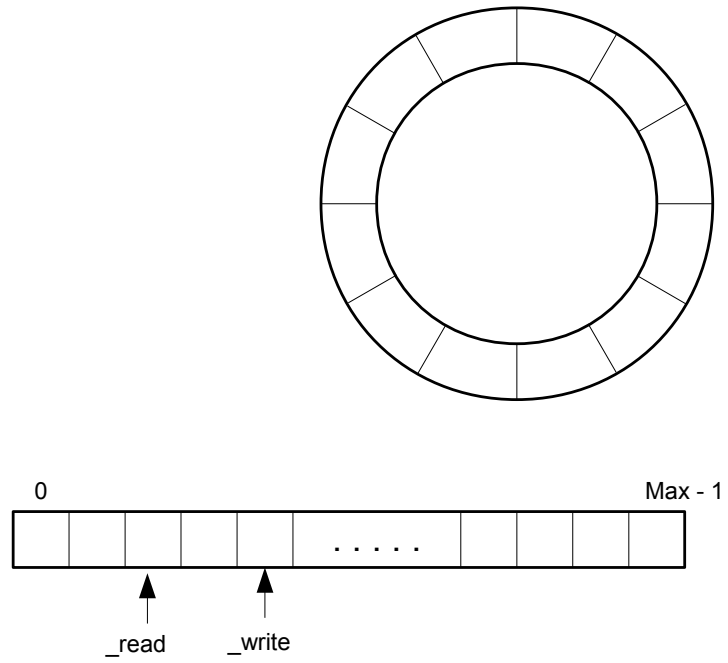

Abbildung 6.3: Typischer Ringpuffer

Aufgabe dieses Moduls ist es, den anderen Modulen über Makros einen einfachen und doch performanten Zugriff auf einen FIFO-Speicher (First In First Out) zu gewährleisten. Dieser Speicher ist als Ringpuffer konzipiert. In Abbildung 6.3 ist ein typischer Ringpuffer dargestellt. Es wird ein Speicher als Array mit einer bestimmten Anzahl Speicherzellen definiert. Über ein Index werden die einzelnen Speicherzellen adressiert. Erreicht der Index beim Hochzählen das Maximum, fängt er wieder am Anfang des Arrays an. So wird der Ringpuffer simuliert. Um zu erkennen, welche Speicherzellen in Verwendung sind und welche nicht, gibt es einen Schreib-Index (write) und einen Lese-Index (read). Beim Schreiben in den Ringpuffer (In) wird der Schreib-Index um eine Speicherzelle erhöht und dort der Inhalt geschrieben. Beim Lesen (Out) wird der Lese-Index um eine Speicherzelle erhöht und deren Inhalt ausgelesen. Zeigen beide Indizes auf die gleiche Speicherzelle, ist der Ringpuffer leer.
Der Ringpuffer des Moduls iec104-buffer funktioniert nach dem oben beschriebenen Prinzip, hat aber spezielle zusätzliche Eigenschaften. Die Abbildung 6.4 verdeutlicht, dass – im Gegensatz zu dem typischen Ringpuffer – zusätzlich ein Sende-Index (send) verwendet wird. Speicherzellen zwischen dem Lese-Index und dem Schreib-Index sind – wie bei dem typischen Ringpuffer – in

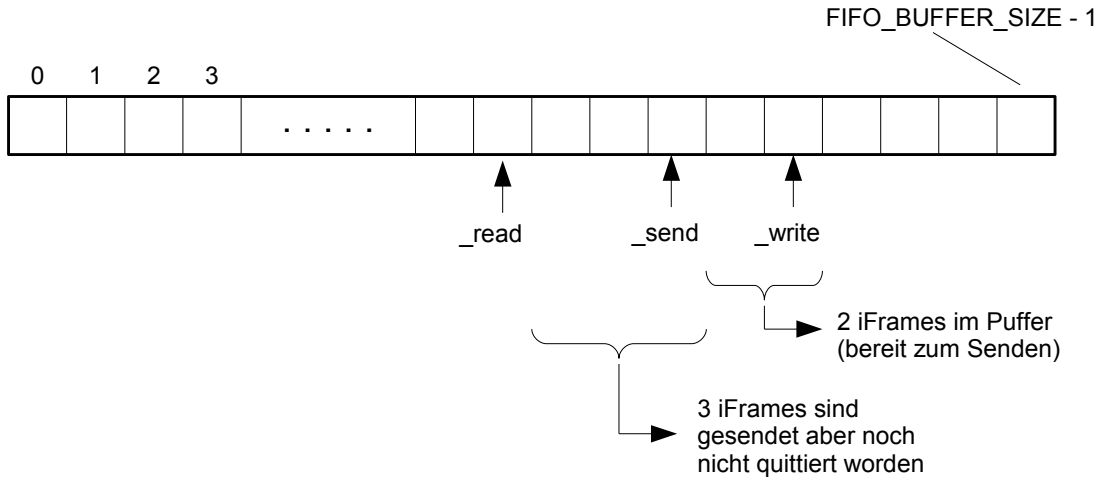

Abbildung 6.4: iec104-buffer Ringpuffer

Verwendung. Der Sende-Index gibt an, welche Speicherzellen schon gesendet worden sind. Erst wenn sie quittiert worden sind, werden sie wieder freigegeben. Speicherzellen zwischen dem Sende-Index und dem Schreib-Index enthalten Informationen, die noch nicht gesendet wurden. Speicherzellen zwischen dem Lese-Index und dem Sende-Index enthalten Informationen, die schon gesendet worden sind, die aber noch nicht quittiert wurden.
Die komplette Implementierung dieses Moduls wurde in der Headerdatei mittels Präprozessoranweisungen programmiert. Dadurch wird der Overhead einer C Funktion eingespart. Bei der Implementierung dieses Moduls wurde auf eine schnelle Pufferbearbeitung und gleichzeitig eine minimale Speicherauslastung Wert gelegt.

Listing 6.17: Auszug aus apps/iec104/iec104-buffer.h

```
#define FIFO_BUFFER_SIZE   32    /* Buffer Size for n iFrame          */
                                 /* Buffer Size must be 2^n !!!!      */

struct iFrame {
  uint16_t      NS;       /* set from link      */
  uint8_t       type;     /* rest set from appl */
  uint8_t       cause;
  uint8_t       byte1;
  uint8_t       byte2;
  uint8_t       byte3;
  uint8_t       value[5]; /* depending on type, must be casted  */
                          /* f.e. SPTYPE, MNTYPE                */
};

struct fifo_t {
  uint8_t       _read;
  uint8_t       _send;
  uint8_t       _write;
  struct iFrame _buffer[FIFO_BUFFER_SIZE];
};
```

Die Größe des Puffers wird über die Konstante FIFO_BUFFER_SIZE festgelegt. Durch die Einschränkung, dass die Puffergröße einer Zweierpotenz entsprechen muss, konnte die Pufferbearbeitung beschleunigt werden. Statt eines Vergleichs, ob ein Index die Obergrenze überschritten hat, wird eine Bitmaske über den Index gelegt. Eine Speicherzelle des Ringpuffers entspricht einem iFrame.

Folgende Makros können von anderen Modulen verwendet werden:

-	FIFO_init(fifo)	Initialisierung des FIFO-Puffers
-	FIFO_waiting(fifo)	gibt zurück, ob Daten zum Senden anstehen
-	FIFO_push(fifo, frameptr)	schreibt einen iFrame in den FIFO-Puffer bei einem Überlauf werden alte Daten überschrieben
-	FIFO_pull(fifo)	gibt ein iFrame aus dem FIFO-Puffer zurück, wenn Daten zum Senden anstehen
-	FIFO_ack(fifo, NSack)	löscht quittierte Daten aus dem Puffer
-	FIFO_waitingSame(fifo,retvalue)	gibt zurück, wieviel iFrames vom gleichen Typ zum Senden anstehen

6.2.3 iec104-para

In der Headerdatei dieses Moduls iec104-para.h werden die Stationsadresse (Common Address) der Steckdose und die Anzahl der verschiedenen Informationsobjekte, die übertragen werden sollen, parametriert. Hier ist auch die Struktur der Informationsobjekte definiert, wie sie in der Quellcodedatei iec104-para.c beschrieben werden.

Listing 6.18: Auszug aus apps/iec104/iec104-para.h

```
#define myPARA_COMMONADDRESS_LOW        14
#define myPARA_COMMONADDRESS_HIH        0
#define myPARA_SPanz                            9
#define myPARA_MNanz                            0
#define myPARA_SCanz                            1

struct InformationObject {
  uint8_t byte1;               /* Byte1         */
  uint8_t byte2;               /* Byte2         */
  uint8_t byte3;               /* Byte3         */
  void    (*fkt_value)(void *value);
};
```

Die Common Adresse hat den Wert 14 und es sind neun Einzelbitmeldungen (Single Point Information Objects) und ein Einzelbitbefehl (Single Command Information Object) parametriert. Zur Identifizierung eines Informationsobjektes wird eine 3-Byte-Adresse verwendet. Bei der Parametrierung der einzelnen Informationsobjekte wird die Informationsobjektadresse mit byte1 als höchstwertigstes Byte, byte2 als mittleres Byte und byte3 als niederwertigstes Byte parametriert. Zusätzlich wird zu jedem Informationsobjekt eine eigene Funktion parametriert. Bei Informationsobjekten in Monitorrichtung (Einzelbitmeldung) wird mithilfe dieser Funktion der Wert des jeweiligen Informationsobjektes festgestellt. In Kommandorichtung (Einzelbitbefehl) wird diese Funktion beim Empfang eines Kommandos ausgeführt.

Listing 6.19: Auszug aus apps/iec104/iec104-para.c

```
void
getvalue_Steckdose(void *value)
{
        if (PORTD & (1 << PIN7)) {
                *(SP_TYPE*)(value) = 1;
        } else {
                *(SP_TYPE*)(value) = 0;
        }
}

void
setvalue_Steckdose(void *value)
{
        if (*(SP_TYPE *)value) {
```

```
                PORTD |=  (1 << PIN7);
        } else {
                PORTD &= ~(1 << PIN7);
        }
}

const struct InformationObject myPARA_SP[myPARA_SPanz] = {
  {  0,   1,   4, getvalue_GPIO0},       /* GPIO0 - PB5 */
  {  0,   1,   5, getvalue_GPIO1},       /* GPIO1 - PB6 */
  {  0,   1,   6, getvalue_GPIO2},       /* GPIO2 - PB7 */
  {  0,   1,  19, getvalue_GPIO3},       /* GPIO3 - PG0 */
  {  0,   1,  20, getvalue_GPIO4},       /* GPIO4 - PG1 */
  {  0,   1,  21, getvalue_GPIO5},       /* GPIO5 - PG2 */
  {  0,   1,  17, getvalue_GPIO6},       /* GPIO6 - PD6 */
  {  0,   1,  18, getvalue_Steckdose},   /* GPIO7 - PD7, Pin18 on Zigbit */
  {  0,   1,  41, getvalue_GPIO8}        /* GPIO8 - PE3 */
};

const struct InformationObject myPARA_SC[myPARA_SCanz] = {
  {  1,   1,  18, setvalue_Steckdose}   /* GPIO7, Pin18 on Zigbit */
};
```

Der Wert der Einzelbitmeldung mit der Informationsadresse 0/1/18 wird über die Funktion getvalue_Steckdose() ermittelt. Dies geschieht alle zwei Sekunden in der Funktion iec104appl_appcall() oder durch eine Generalabfrage von der Gegenstelle. Wenn die Gegenstelle einen Einzelbitbefehl mit der Informationsadresse 1/1/18 sendet, wird die Funktion setvalue_Steckdose() ausgeführt. Abhängig von dem empfangenen Wert wird der Ausgangspin zum Relais auf logisch 1 oder auf logisch 0 gesetzt.

6.2.4 iec104-appl

Dieses Modul ist für die Applikationsschicht des Protokolls IEC104 zuständig. Hier werden empfangene ASDU Pakete dekodiert, um gegebenenfalls darauf zu reagieren. Wenn zum Beispiel ein Kommando empfangen wird, wird dieses, sofern es mit einem parametrierten Kommando übereinstimmt, ausgeführt. Das Modul sendet aber auch Informationen zur Gegenstelle, zum Beispiel wenn sich spontan der Wert eines Signals ändert. In dem Fall werden aus den Informationen iFrames erstellt und diese der Linkschicht, also dem Modul iec104-link, übergeben.

Listing 6.20: Auszug aus apps/iec104/iec104-appl.h

```
#define APPLICATION_TIMER 2       /* check period for application     */
void iec104appl_appcall(struct iec104_state *s);
void iec104appl_linkevent(struct iec104_state *s);
```

In der Headerdatei iec104-appl.h ist der APPLICATION_TIMER mit zwei Sekunden definiert. Das ist der Zyklus, in dem die Funktion iec104appl_appcall() vom Hauptmodul aufgerufen wird. Diese Funktion geht alle parametrierten Informationsobjekte in Monitorrichtung durch und führt die jeweilige parametrierte Funktion aus, um den aktuellen Wert zu bekommen. Nur falls sich ein Wert geändert hat, wird ein iFrame erzeugt und der Linkschicht übergeben.

Listing 6.21: Funktion iec104appl_linkevent() aus apps/iec104/iec104-appl.c

```
void
iec104appl_linkevent(struct iec104_state *s)
{
        switch(s->inputbuf[IEC_HEADSIZE + IEC_CONTSIZE]) {
                case C_SC_NA_1: dec_C_SC_NA_1(s);       break;
                case C_IC_NA_1: dec_C_IC_NA_1(s);       break;
                default:                ;
        }
}
```

Die Funktion iec104appl_linkevent() wird vom Modul iec104-link aufgerufen, wenn Applikationsdaten empfangen werden. Aktuell gibt es zwei IEC104-Datentypen, die die Applikationsschicht in Kommandorichtung verarbeitet: die Generalabfrage C_IC_NA_1 und den Einzelbitbefehl C_SC_NA_1.
Bevor weitere Funktionen dieses Moduls erörtert werden, soll die Telegrammstruktur auf der Applikationsschicht vorgestellt werden. Der Standard IEC104 bedient sich dabei dem Aufbau einer ASDU (Application Service Data Unit) nach dem Standard IEC 60870-5-101 (siehe Abbildung 6.5). Jedes Feld entspricht dabei einem Byte in der Telegrammstruktur. Im ersten Byte steht der Datentyp (Type Identification). Der Standard liefert eine Vielzahl von verschiedenen Datentypen, zum Beispiel eine Einzelbitmeldung ohne Zeitstempel M_SP_NA_1. Das zweite Byte (Variable Structure Qualifier) gibt an, wie viele Informationsobjekte innerhalb dieser ASDU übertragen werden. Es gibt einen Übertragungsgrund (Cause Of Transmission). Signale können zum Beispiel aufgrund einer spontanen Änderung übertragen werden oder weil ihr aktueller Zustand aufgrund einer Generalabfrage übertragen werden soll. Die Common Adresse ist die Stationsadresse der IEC104 Applikation. Sie wird in der Datei iec104-para.h parametriert. Bei den einzelnen Informationsobjekten kommt zuerst die Informationsadresse, bestehend aus drei Bytes. Danach kommt abhängig vom jeweiligen Datentyp der Wert mit eventuellen Qualifier Flags und einem möglichen Zeitstempel.
Folgende Datentypen sind implementiert:

- M_SP_NA_1 (Type Ident 1): Einzelbitmeldung ohne Zeitstempel
- M_ME_NA_1 (Type Ident 9): Messwert, normalisiert und ohne Zeitstempel
- C_SC_NA_1 (Type Ident 45): Einzelbitbefehl
- C_IC_NA_1 (Type Ident 100): Generalabfrage

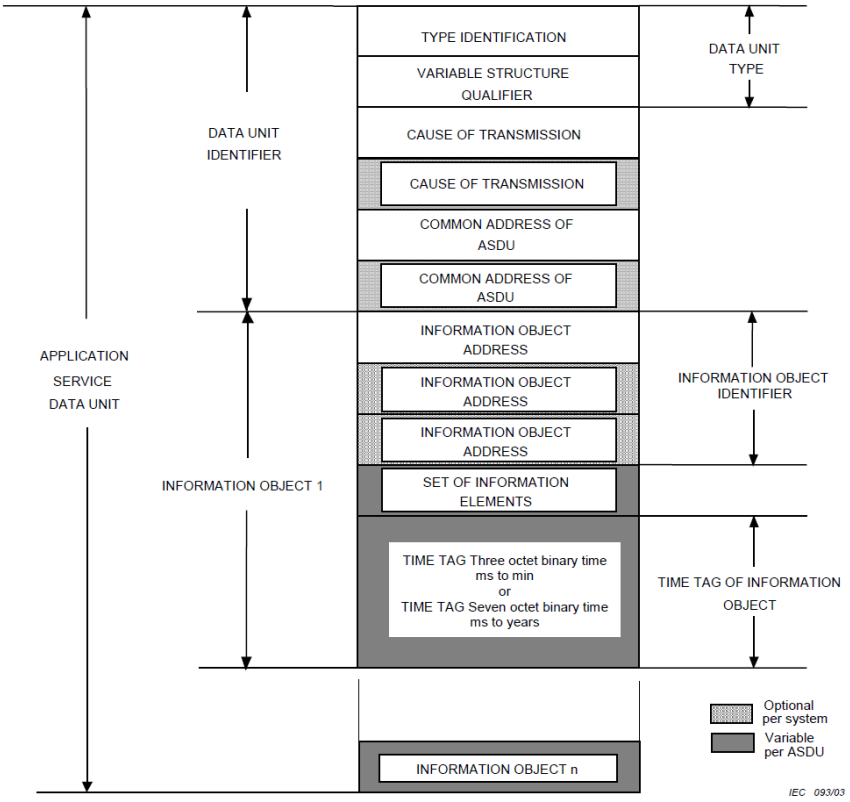

Figure 10 – Structure of an Application Service Data Unit ASDU

Abbildung 6.5: Telegrammstruktur ASDU [IEC101]

Listing 6.22: Weitere Funktionen aus apps/iec104/iec104-appl.c

```
push_M_SP_NA_1(iec104_state *s, uint8_t iNum, uint8_t iCause, SP_TYPE iValue)
push_M_ME_NA_1(iec104_state *s, uint8_t iNum, uint8_t iCause, MN_TYPE iValue)
push_C_SC_NA_1(iec104_state *s, uint8_t iNum, uint8_t iCause, SP_TYPE iValue)
push_C_IC_NA_1(iec104_state *s, uint8_t iCause, uint8_t iValue)
 dec_C_SC_NA_1(iec104_state *s)
 dec_C_IC_NA_1(iec104_state *s)
```

Bei den weiteren Funktionen dieses Moduls können zwei Arten unterschieden werden. Funktionen, die mit „dec_“ beginnen, haben die Aufgabe, empfangene Applikationsdaten zu dekodieren und entsprechend darauf zu reagieren. Funktionen, die mit „push_“ beginnen, erzeugen mit den übergebenen Informationen ein iFrame und senden diesen an die Linkschicht. Beide Arten von Funktionen werden nur innerhalb dieses Moduls verwendet.

Listing 6.23: Funktion push_M_SP_NA_1() aus apps/iec104/iec104-appl.c

```
static void
push_M_SP_NA_1(struct iec104_state *s, uint8_t iNum
               , uint8_t iCause, SP_TYPE iValue)
{
        struct iFrame frame;
        frame.type = M_SP_NA_1;
        frame.cause = iCause;
        frame.byte1 = myPARA_SP[iNum].byte1;
        frame.byte2 = myPARA_SP[iNum].byte2;
        frame.byte3 = myPARA_SP[iNum].byte3;
        frame.value[0] = iValue & VALUE_SINGLE;
        push_I_Frame(s, &frame);
}
```

Als Beispiel für eine „push_“-Funktion ist hier die Funktion zum Erzeugen einer Einzelbitmeldung M_SP_NA_1 aufgeführt. Der Ablauf einer solchen Funktion ist immer gleich. Es wird eine Variable vom Typ iFrame definiert. Von diesem iFrame wird der IEC104-Datentyp gesetzt. Der Übertragungsgrund (Cause Of Transmission) wird von den übergebenen Parametern gefüllt. Anhand der Variable iNum kann die parametrierte Informationsobjektadresse identifiziert werden. Über den Wert wird noch eine Bitmaske gelegt, weil bei Einzelbitmeldungen nur das letzte Bit den Wert enthält. Nachdem der iFrame aufgebaut ist, wird er über die Funktion push_I_Frame() der Linkschicht übergeben.

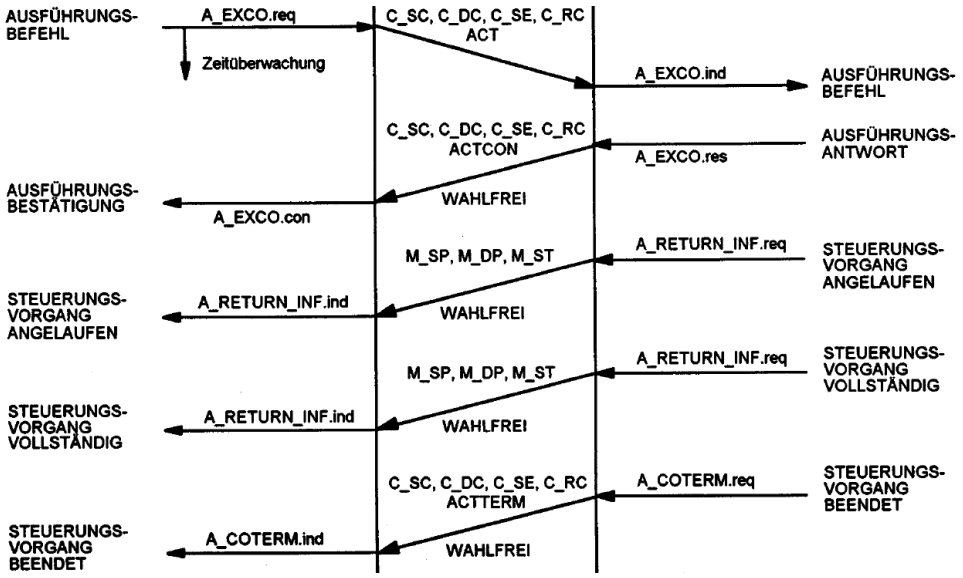

Abbildung 6.6: IEC104 Befehlsprozedur [IEC870-5-5]

Die Funktion dec_C_SC_NA_1() übernimmt die Verarbeitung eines Einzelbitbefehls, der von der Gegenstelle empfangen worden ist. Dabei wird zuerst geprüft, ob es sich um einen gültigen Befehl handelt. Die Stationsadresse muss übereinstimmen, die Informationsobjektadresse muss als Einzelbitbefehl parametriert sein und der Übertragungsgrund muss „Activation“ sein. Wenn das übereinstimmt, wird die Befehlsprozedur nach Abbildung 6.6 durchgeführt. Der Empfang des Einzelbitbefehls wird durch ein identisches Telegramm mit dem Übertragungsgrund „Activation Confirmation“ bestätigt. Der Befehl wird ausgeführt, die zugehörige parametrierte Funktion wird aufgerufen. Dann wird ein identisches Telegramm mit dem Übertragungsgrund „Activation Termination“ gesendet, um den Abschluss des Befehls anzuzeigen.

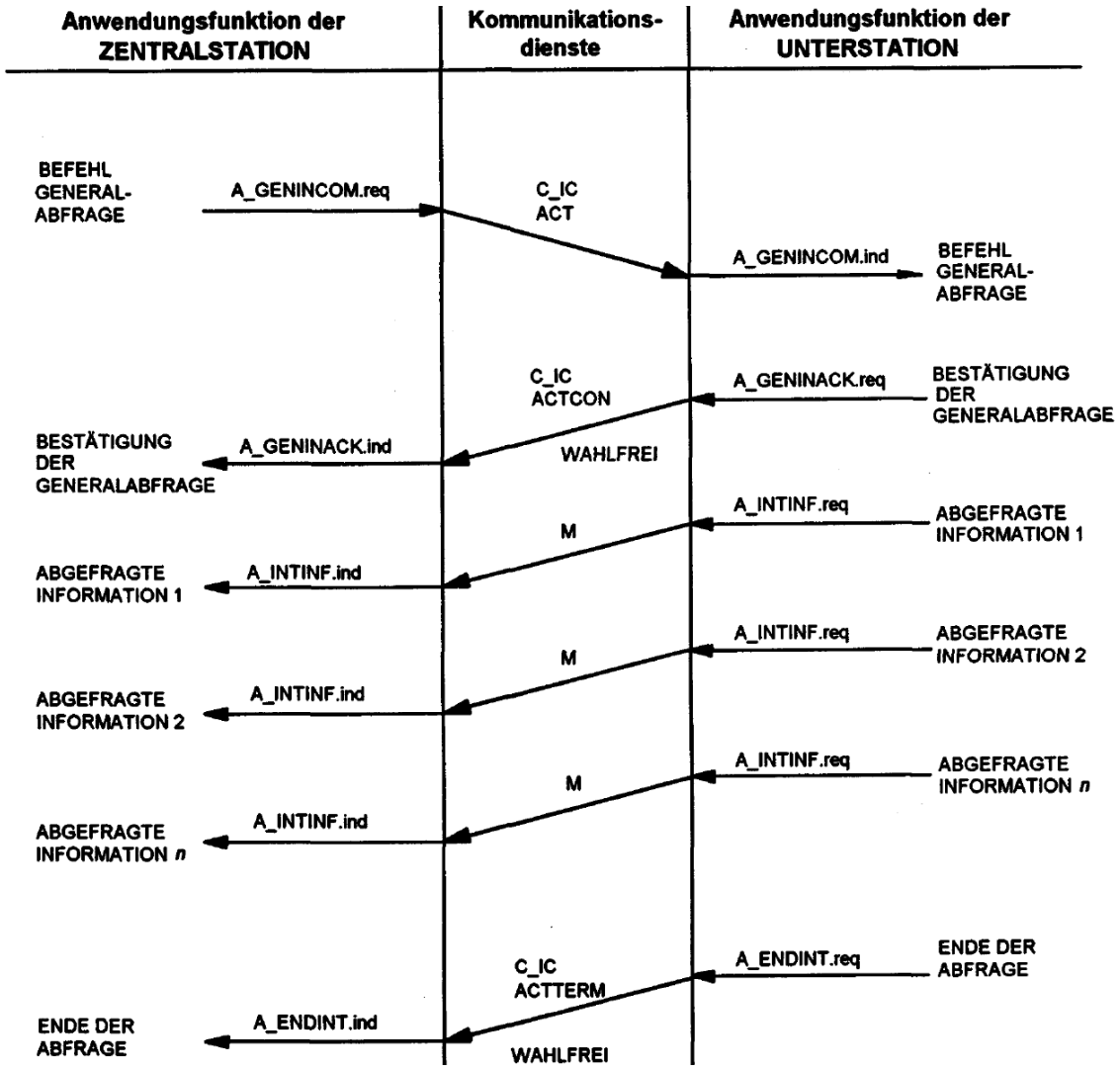

Abbildung 6.7: IEC104 Prozedur zur Unterstationsabfrage [IEC870-5-5]

Die Abbildung 6.7 zeigt die Prozedur einer Generalabfrage in IEC104. Die Zentralstation ist dabei der IEC104-Master, also die Gegenstelle. Die Unterstation ist diese Implementierung, also die Steckdose. Wenn ein solcher Befehl empfangen wird, übernimmt die Funktion dec_C_IC_NA_1() die Verarbeitung. Es wird zuerst geprüft, dass die Stationsadresse übereinstimmt. Bei einer Generalabfrage muss die Informationsobjektadresse gleich null sein. Ein Informationswert von 20 bedeutet, dass es sich um eine globale Generalabfrage handelt. Mit anderen Werten ist es möglich, verschiedene Gruppen abzufragen – in dieser Implementierung wird nur die globale Generalabfrage beantwortet. Wenn das übereinstimmt und der Übertragungsgrund gleich „Activation“ ist, wird die Antwort erstellt und zurückgesendet. Zuerst wird der Empfang der Generalabfrage durch

ein „Activation Confirmation“ bestätigt, dann wird zu jedem parametrierten Informationsobjekt in Monitorrichtung die zugehörige Funktion ausgeführt, um den aktuellen Zustand zu ermitteln. Dieser wird dann mit dem Übertragungsgrund „Generalabfrage“ übertragen. Das Ende der Generalabfrage wird durch ein „Activation Termination“ gekennzeichnet.

6.2.5 iec104-link

Die Linkschicht des Protokolls IEC104 wird von diesem Modul übernommen. Dazu gehören die Kodierung und Dekodierung der endgültigen IEC104-Telegramme, die Quittierungen empfangener Telegramme durch sogenannte Sequenznummern (Sequence Number) und die Behandlung von Linksteuerungsfunktionen.
Die Implementierung der TCP/IP-Verarbeitung ist angelehnt an die Applikation webserver-nano. Beim Starten des Contiki-Prozesses im Hauptmodul wird die Funktion iec104link_init() aufgerufen. Hier wird der TCP-Port für IEC104 (2404) für eingehende Verbindungen geöffnet. Während des normalen Betriebs wird bei einem TCP/IP-Event die Funktion iec104link_appcall() aufgerufen.

Listing 6.24: Funktion iec104link_appcall() aus apps/iec104/iec104-link.c

```
void
iec104link_appcall(void *state)
{
        struct iec104_state *s = (struct iec104_state *)state;
        if(uip_closed() || uip_aborted() || uip_timedout()) {
                /*~~IEC104-Status initialisieren~~*/
        } else if(uip_connected()) {
                /*~~IEC104-Status initialisieren~~*/
                /*~~Verbindung initialisieren~~*/
                handle_connection(s);
        } else if(s->state & STATE_ESTABLISHED) {
                /*~~Timeout prüfen, wenn Verbindung im Ruhezustand ist~~*/
                s->state |= STATE_INPUT;
                handle_connection(s);
        } else {
                /*~~Verbindung abbrechen und IEC104-Status initialisieren~~*/
        }
}
```

Der Status der IEC104-Verbindung wird in der Struktur struct iec104_state *s gespeichert. In dieser Struktur befindet sich auch die Instanz des FIFO Puffers iec104-buffer. Wenn TCP/IP-Daten empfangen werden, wird die Funktion handle_connection() aufgerufen. Diese Funktion wiederum ruft die beiden Funktionen handle_input() und handle_output() auf, sofern eingehende Daten oder ausgehende Daten anstehen.
Bevor die Funktion handle_input() genauer erläutert wird, wird die Telegrammstruktur des IEC104 vorgestellt.

6.2.5.1 IEC104 Telegrammstruktur

Wie in Abbildung 6.8 dargestellt, hat ein IEC104-Telegramm ein festes Startbyte mit dem Wert 0x68. Als zweites Byte wird die Länge des Telegramms übertragen, wobei die ersten beiden Bytes nicht mit gezählt werden. Vier weitere Bytes übernehmen verschiedene Linkkontrollaufgaben und vervollständigen den ersten Abschnitt eines IEC104-Telegramms zur APCI (Application Protocol Control Information). Zusammen mit der ASDU aus der IEC104-Applikationsschicht ergibt das ein APDU (Application Protocol Data Unit). Generell können sowohl komplette APDUs übertragen werden als auch – zur Linkkontrolle – reine APCIs. Wenn nur die APCI übertragen wird, enthält das Längenbyte immer den Wert vier.

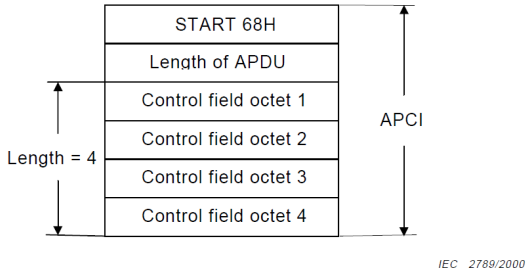

Figure 5 – APCI of the defined telecontrol companion standard

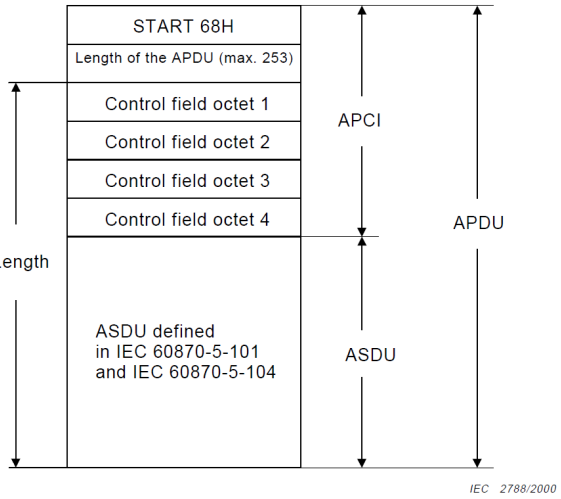

Figure 4 – APDU of the defined telecontrol companion standard

Abbildung 6.8: IEC104 Telegrammstruktur APCI und APDU [IEC104]

Bei den vier Bytes für die Linkkontrolle ist die Interpretation abhängig vom jeweiligen Format. Es gibt dabei drei verschiedene Typen:

- U-Format (unnumbered): Kontrollfunktionen ohne Nummerierung
- S-Format (supervisory): Überwachungsfunktion mit Nummerierung
- I-Format (infomation): Informationstransfer mit Nummerierung

Die letzten beiden Bits im ersten Kontrollbyte definieren, um welche Art von Telegrammformat es sich handelt. Telegramme in U-Format und in S-Format werden nur als reine APCI übertragen. Telegramme in I-Format können komplette APDUs, APCI plus ASDU, übertragen.

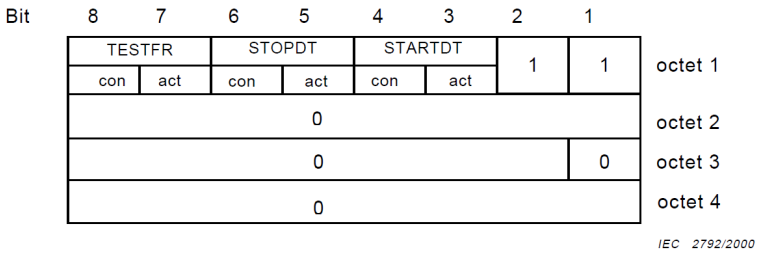

Figure 8 – Control field of type unnumbered control functions (U format)

Abbildung 6.9: IEC104 APCI Control Field in U-Format[IEC104]

Beim U-Format (Abbildung 6.9) hat nur das erste Byte eine Bedeutung, die anderen drei Bytes sind auf den Wert 0 zu setzen. Es stehen drei verschiedene Linkfunktionen zur Verfügung, von denen nur eine zur gleichen Zeit verwendet werden darf. Bei jeder Funktion gibt es ein Bit für die Aktivierung (act) und ein Bit für die Bestätigung (con).

- TESTFR (Test Frame): Diese Funktion wird verwendet, um den Link zu testen. Dies kann von beiden Seiten initiiert werden. Üblicherweise wird diese Funktion verwendet, wenn kein Datenverkehr erfolgt, der Link sich also für längere Zeit im Ruhezustand befindet.
- STOPDT (Stop Data Transmission): Diese Funktion wird vom IEC104-Master (Zentralstation) verwendet, um dem IEC104-Slave (Unterstation) anzuzeigen, dass keine Daten mehr gesendet werden dürfen. Der IEC104-Slave darf nach dem Empfang des STOPDT keine ASDUs mehr senden.

- STARTDT (Start Data Transmission): Diese Funktion ist das Gegenteil vom STOPDT. Hiermit zeigt der IEC104-Master dem IEC104-Slave an, dass Daten – also ASDUs – gesendet werden dürfen. Dieses Telegramm wird üblicherweise direkt nach dem TCP/IP-Linkaufbau vom IEC104-Master gesendet.

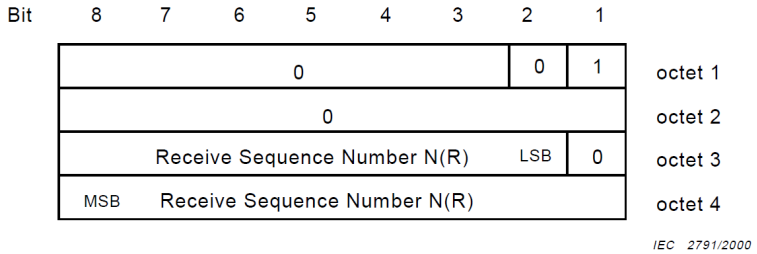

Figure 7 – Control field of type numbered supervisory functions (S format)

Abbildung 6.10: IEC104 APCI Control Field in S-Format[IEC104]

APCIs in S-Format (Abbildung 6.10) werden dazu verwendet, empfangene I-Format-Telegramme zu quittieren. Dazu wird eine sogenannte Sequenznummer verwendet. Das Prinzip hat der IEC104-Standard aus der X.25-Empfehlung der ITU-T [X.25] entnommen. Die Sequenznummer soll gewährleisten, dass keine Meldungen verloren gehen oder doppelt übertragen werden. Ebenfalls soll die Reihenfolge der Meldungen sichergestellt werden.

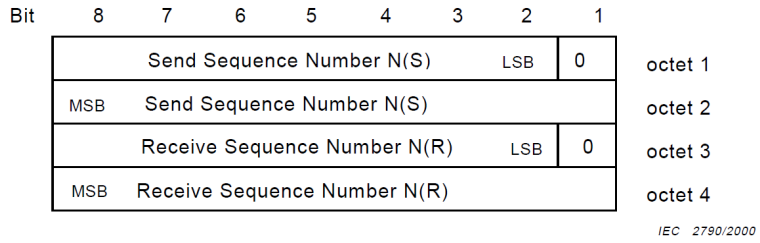

Figure 6 – Control field of type Information transfer format (I format)

Abbildung 6.11: IEC104 APCI Control Field in I-Format[IEC104]

Komplette APDUs, also zusammen mit Dateninformationen ASDUs, können nur im I-Format (Abbildung 6.11) gesendet werden. Es gibt eine sogenannte Send Sequence Number, die bei jedem Senden einer APDU um eins erhöht wird. Und es gibt eine Receive Sequence Number, die den Empfang einer APDU bestätigt. Bei einem TCP/IP-Linkaufbau werden beide Nummern auf null gesetzt. Mit einem I-Format wird gleichzeitig die Send Sequence Number und die Receive Sequence Number gesendet. Es kann also gleichzeitig ein neues APDU gesendet und der Empfang einer empfangenen APDU bestätigt werden.
Nicht jede Sequenznummer muss quittiert werden. Der IEC104 Standard definiert einen parametrierbaren Wert w. Vom Standard empfohlen, und in dieser Implementierung verwendet, ist der Wert w=8. Das bedeutet, dass nach acht empfangenen Sequenznummern, die letzte Sequenznummer bestätigt werden muss. Eine vorherige Quittierung ist erlaubt, aber nicht notwendig.

6.2.5.2 handle_input()

Die Funktion handle_input() wird aufgerufen, um empfangene Datenpakete zu verarbeiten. Der Programmablauf dieser Funktion ist als Struktogramm in Abbildung 6.12 dargestellt. Es wird zuerst das Startbyte und die Telegrammlänge geprüft. Wenn es sich um eine U-Format APCI handelt, werden die Linkkontrollfunktionen geprüft. Ein TESTFR act wird mit einem TESTFR con bestätigt. Beim Empfang eines STARTFR act wird ein Marker gesetzt, dass ASDUs gesendet

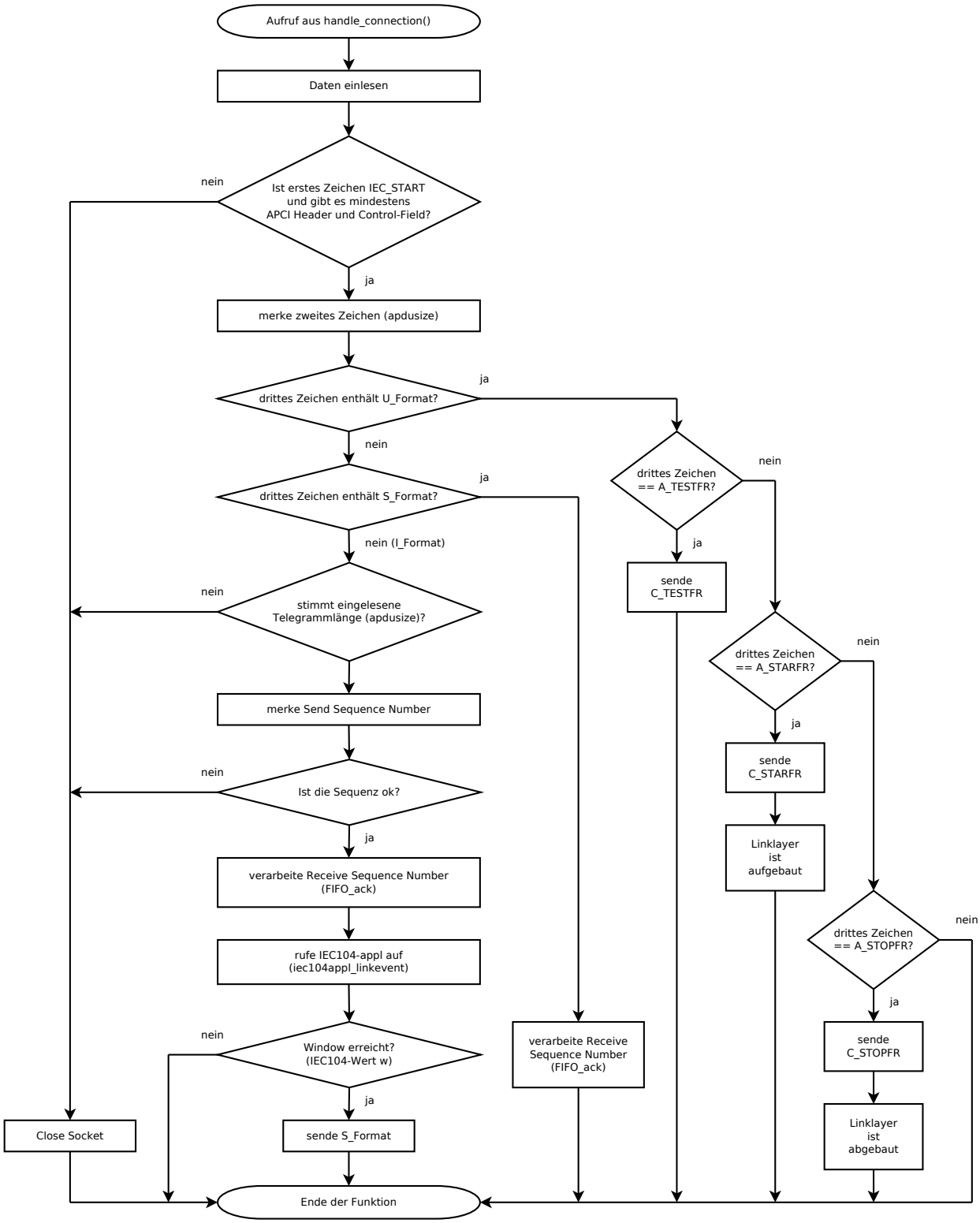

Abbildung 6.12: Struktogramm der Funktion handle_input()

werden dürfen. Sein Empfang wird mit einem STARTFR con bestätigt. Ein STOPDT act wird mit einem STOPDT con bestätigt und bewirkt, dass keine ASDUs mehr gesendet werden. Wenn anstatt einer U-Format APCI eine S-Format APCI empfangen wird, werden die quittierten ASDUs aus dem FIFO Puffer gelöscht. Wenn weder eine U-Format APCI, noch eine S-Format APCI empfangen wurde, dann wurde eine I-Format APCI mit Dateninformationen empfangen. In dem Fall wird geprüft, ob die eingelesene Telegrammlänge stimmt. Es wird anhand der Send Sequence Number geprüft, dass die Reihenfolge der Meldungen stimmt. Die Receive Sequence Number wird verwendet, um quittierte ASDUs aus dem FIFO Puffer zu löschen. Die empfangene ASDU wird über die Funktion iec104appl_linkevent() der Applikationsschicht übergeben. Wenn bei der empfangenen Send Sequence Number und der zuletzt quittierten der Wert w überschritten wurde, wird eine Bestätigung als APCI im S-Format gesendet.

6.2.5.3 handle_output()

Die Funktion handle_output() ist für das Senden von Datenpaketen zuständig. Solange Daten zum Senden anstehen, wird die Funktion send_frames() aufgerufen. Diese Funktion wiederum prüft, ob eine APCI in U-Format oder in S-Format gesendet werden muss und ruft in diesem Fall die Funktion send_U_Frame() bzw. send_S_Frame() auf. Wenn Daten zum Senden im FIFO-Puffer anstehen, wird die Funktion send_I_Frame() aufgerufen. Die beiden ersten send_-Funktionen bauen die APCI entsprechend auf und senden diese. Mit der Funktion send_I_Frame() wird die komplette APDU, also auch die ASDU, aufgebaut und gesendet. Dafür wird das anstehende iFrame aus dem FIFO-Puffer gelesen. Ein iFrame enthält alle notwendigen Informationen, um die ASDU aufzubauen. Wenn mehrere iFrames vom gleichen Typ anstehen, werden diese zusammen in einer ASDU gesendet.

6.2.5.4 push_I_Frame()

Das Schreiben von iFrames in den FIFO-Puffer wird von der Applikationsschicht initiiert. Diese ruft dazu die Funktion push_I_Frame() auf und übergibt ein vorgefertigtes iFrame. In dieser Funktion wird das iFrame über die Funktion FIFO_push() in den FIFO-Puffer geschrieben. Ein Marker wird gesetzt, dass Daten zum Senden anstehen.

6.3 Contiki-Projekt

Um aus beiden Applikationen webserver-nano-steckdose und iec104 ein Programm zu erzeugen, das auf das Zigbit-Modul programmiert werden kann, muss ein Projekt erzeugt werden. Dazu wurde ein neues Verzeichnis projects/fs_both im Contiki-Hauptverzeichnis angelegt. Um das Contiki-Projekt übersetzen zu können, werden zwei Makefiles angelegt.

Listing 6.25: Auszug aus project/fs_both/Makefile

```
CONTIKI_PROJECT = both
ifndef TARGET
  TARGET=avr-zigbit
  MCU=atmega1281
endif

all:
  make -f Makefile.$(CONTIKI_PROJECT) TARGET=$(TARGET) $(CONTIKI_PROJECT).elf
  avr-objcopy -O ihex -R .eeprom -R .fuse -R .signature                    \
              $(CONTIKI_PROJECT).elf $(CONTIKI_PROJECT).hex
  avr-objcopy -O ihex -j .eeprom --set-section-flags=.eeprom="alloc,load"   \
              --change-section-lma .eeprom=0 $(CONTIKI_PROJECT).elf         \
              $(CONTIKI_PROJECT)_eeprom.hex
  avr-size -C --mcu=$(MCU) $(CONTIKI_PROJECT).elf
```

Durch Ausführen des Befehls „make“ im Projektverzeichnis wird das Makefile aufgerufen. Das Contiki-Projekt bekommt den Namen „both“ zugewiesen und die verwendete Plattform TARGET=avr-zigbit wird gesetzt. Es wird das zweite Makefile Makefile.both aufgerufen. Als Ausgabe wird die Datei both.elf erstellt, die das Programm für den Flash und für den EEPROM des Mikrocontrollers im ELF-Format enthält. Die beiden Befehle „avr-objcopy“ erzeugen aus dieser Datei zwei Dateien im Intel-HEX-Format. Die Datei both.hex enthält das Programm, das in den Flash-Speicher zu laden ist, und die Datei both_eeprom.hex enthält Konfigurationsparameter für den EEPROM-Speicher. Der Befehl „avr-size“ gibt die Speicherauslastung des Mikrocontrollers an.

Listing 6.26: Auszug aus project/fs_both/Makefile.both

```
APPS=iec104 webserver-nano-steckdose
CONTIKI = ../..
CFLAGS += -DPROJECT_CONF_H=\"project-conf.h\"
include $(CONTIKI)/Makefile.include
```

Das Wesentliche im zweiten Makefile ist die Auswahl der Applikationen. Mit der Anweisung „APPS=iec104 webserver-nano-steckdose“ werden beide Applikationen ausgewählt. Sie laufen dann parallel jeweils im eigenen Contiki-Prozess.

Listing 6.27: Auszug aus project/fs_both/both.c

```
#include "contiki.h"
#include "iec104.h"
#include "webserver-nogui.h"
AUTOSTART_PROCESSES(&iec104_process,&webserver_nogui_process);
```

Eine neue Quellcode-Datei both.c wurde erstellt. Mit der Anweisung „AUTOSTART_PROCESSES“ werden die Prozesse von beiden Applikation automatisch beim Anlauf gestartet. Es wurde zusätzlich eine Header-Datei project-conf.h erstellt, in der Präprozessoranweisungen gesetzt bzw. überschrieben worden sind.

Listing 6.28: Auszug aus project/fs_both/project-conf.h

```
#define WEBSERVER_CONF_NANO 1
#define RADIOSTATS 1
#undef UIP_CONF_BUFFER_SIZE
#define UIP_CONF_BUFFER_SIZE  1280
```

Durch den Befehl „make“ wurde das Programm übersetzt. Die Ausgabe des Befehls „avr-size“ ergab, dass 71350 Bytes Flash-Speicher verwendet werden. Das entspricht einer Auslastung von 54,4%. Im RAM-Speicher werden laut „avr-size“ 7282 Bytes verwendet, was 88,9% entspricht. Es wurde beobachtet, dass eine RAM-Auslastung von mehr als 90% Speicherüberläufe verursachen kann. Woran das genau liegt, konnte nicht ermittelt werden. Eine Vermutung ist, dass der benötigte Speicher für die Stackverwaltung nicht berücksichtigt wird.

Listing 6.29: Auszug aus platform/avr-zigbit/contiki-avr-zigbit-main.c

```
/* Put default MAC address in EEPROM */
uint8_t mac_address[8] EEMEM
                            = {0x02, 0x11, 0x22, 0xff, 0xfe, 0x33, 0x44, 0x55};
```

Listing 6.30: Inhalt von project/fs_both/both_eeprom.hex

```
:0800000021122FFFE334455FA
:00000001FF
```

Der EEPROM-Speicher wird nur für die Speicherung der 64-bit MAC-Adresse verwendet, er belegt also acht Bytes. Im Contiki-Quellcode ist diese MAC-Adresse in der Datei platform/avr-zigbit/contiki-avr-zigbit-main.c definiert. Sie ist vorbelegt mit der Adresse 0211:22ff:fe33:4455.

Wenn mehrere Endgeräte programmiert werden, muss gewährleistet sein, dass die MAC-Adresse eindeutig ist. Dabei gibt es zwei Möglichkeiten. Entweder wird für jedes Endgerät im Quellcode die Variable mac_address oben angepasst und das Projekt neu übersetzt. Oder für jedes Endgerät wird eine eigene Datei im Intel HEX Format erstellt, um den EEPROM unterschiedlich programmieren zu können. Es wurde die zweite Möglichkeit gewählt. Insgesamt wurden zwei Steckdosen hergestellt, also auch zwei Zigbit-Module programmiert. Es wurden also zwei Dateien erstellt, um den jeweiligen EEPROM-Speicher zu programmieren. Die MAC-Adresse wurde dabei an die Seriennummer des Zigbit-Moduls angelehnt. Die endgültige IPv6-Adresse des Zigbit-Moduls wird aus der Netzadresse und der Hostadresse zusammengesetzt. Die Netzadresse gibt der 6LoWPAN-Router vor, sie ändert sich, sobald das Modul in ein anderes Netzwerk eingebunden wird. Die Hostadresse wird aus der MAC-Adresse erstellt. Dabei bedeutet das zweitniederwertigste Bit des ersten Byte (02), dass es sich um eine lokale MAC-Adresse handelt. Es handelt sich nicht um eine global gültige MAC-Adresse, wie sie von der IEEE vergeben wird. Beim Umwandeln in den Hostadressen-Anteil der IP-Adresse wird dieses Bit zurückgesetzt.

Listing 6.31: Inhalt von project/fs_both/both_eepZigbit1.hex

```
:0800000002000200001316745 7
:00000001FF
```

Listing 6.32: Inhalt von project/fs_both/both_eepZigbit2.hex

```
:0800000002000200001426794 1
:00000001FF
```

Zigbit-Modul 1:	Ordering Code	ATZB-24-A2
	Serial Number	020000131674
	MAC Address	0200:0200:0013:1674
	Host IP Address	::200:13:1674
Zigbit-Modul 2:	Ordering Code	ATZB-24-A2
	Serial Number	020000142679
	MAC Address	0200:0200:0014:2679
	Host IP Address	::200:14:2679

Mit einer Netzadresse von 2a01:1e8:e115::/64 ergibt sich zum Beispiel für das Zigbit-Modul 2 eine IPv6-Adresse von 2a01:1e8:e115::200:14:2679.

Kapitel 7

Test

Nachfolgend wird beschrieben, wie die fertige Steckdose getestet wurde. Zuerst wird der Testaufbau vorgestellt. Die einzelnen Geräte, die zum Testen verwendet wurden, die Verbindungen zwischen ihnen und ihre Aufgaben werden erklärt. Der Test fand zuerst in einem privatem Netzwerk statt, später wurden öffentliche IP-Adressen vom IPv6-Tunnelbroker SixXS verwendet, die weltweit übers Internet erreichbar sind. Das Unterkapitel 7.2 widmet sich diesem erweiterten Testaufbau. Beide implementierten Applikationen wurden unabhängig voneinander getestet. Auf die Applikation webserver-nano-steckdose wurde mit verschiedenen Browser zugegriffen. Zum Test der Applikation iec104 wurde das Netzleitsystem Spectrum Power der Firma Siemens AG verwendet. Im Unterkapitel 7.5 werden verschiedene Sicherheitsaspekte der Implementierung untersucht.

7.1 Testaufbau

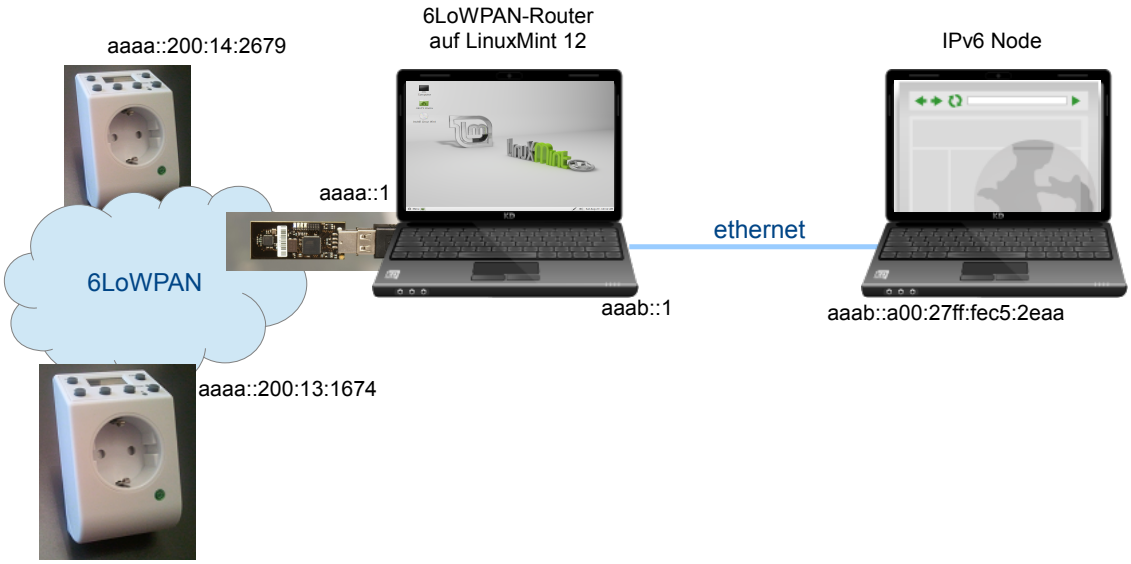

Abbildung 7.1: Testaufbau Steckdosen

In Abbildung 7.1 ist der Testaufbau dargestellt. Um auf die Steckdose zugreifen zu können, wird ein 6LoWPAN-Router benötigt. Verwendet wurde dafür ein handelsüblicher PC mit einer Linux Installation (Linux Mint 12 [Web:LinuxMint]). Dieser PC benötigt mehrere Schnittstellen. Die Schnittstelle in die 6LoWPAN-Wolke wurde durch einen USB-Stick RZUSBSTICK von Atmel [AVR RZRaven] realisiert. Der RZUSBSTICK wurde mit dem Programm ravenusbstick.elf aus dem Contiki 2.6 Verzeichnis examples/ravenusbstick geladen. Er arbeitet als Bridge, um zum PC

eine Ethernet-Schnittstelle zu simulieren. Zum 6LoWPAN verwendet er dazu die MAC-Adresse 0212:13ff:fe14:1516. Auf PC-Seite wird ein Ethernet-Interface usb0 mit der MAC-Adresse 02:12:13:14:15 und der local-link-IPv6-Adresse fe80::12:13ff:fe14:1516 angezeigt. Weil der PC als Router wirken soll, wird die globale IPv6-Adresse statisch eingestellt, hier auf aaaa::1 mit der Netzmaske /64. Damit alle Geräte in diesem Netzwerk über die Netzadresse informiert werden und wissen, dass der PC der Router ist, müssen sogenannte IPv6 Router Advertisements gesendet werden. Dazu muss der Linux-Daemon radvd [Web:radvd] auf dem PC installiert werden. Konfiguriert wird der radvd über die Datei /etc/radvd.conf. Dort wird das Interface usb0 eingetragen mit der Netzadresse bzw. den IPv6 Präfix aaaa::/64. Der PC sendet dann auf eine Anfrage der Steckdosen Router Solicitation oder periodisch Router Advertisments. So erkennen die Steckdosen, dass sie sich im Netzwerk aaaa::/64 befinden und bekommen dadurch die IP-Adressen aaaa::200:13:1674 bzw. aaaa::200:14:2679.
Damit ist es möglich, vom 6LoWPAN-Router eine Verbindung zu den Steckdosen aufzubauen. Damit auch andere Geräte, die z.B. über eine Ethernet-Schnittstelle zum 6LoWPAN-Router verbunden sind, auf eine Steckdose zugreifen können, muss die Ethernet-Schnittstelle ebenso konfiguriert werden. Am 6LoWPAN-Router wird an der Schnittstelle eth0 die statische IP aaab::1/64 konfiguriert und in der radvd-Konfigurationsdatei zusätzlich das Interface eth0 mit dem Präfix aaab::/64 eingetragen. Auf Geräten, die an diesem Netzwerk teilnehmen, muss das Netzwerkinterface auf IPv6-Autokonfiguration gesetzt werden. Sie bekommen dann automatisch eine IP-Adresse, die sich aus dem Präfix und einem lokalen Teil – in der Regel die MAC-Adresse – zusammensetzt. Mit dieser Konfiguration kann der IPv6-Node mit der IP-Adresse aaab::a00:27ff:fec5:2eaa auf die Steckdosen zugreifen. Der Netzwerkverkehr wird dann von dem 6LoWPAN-Router geroutet.

7.2 SixXS

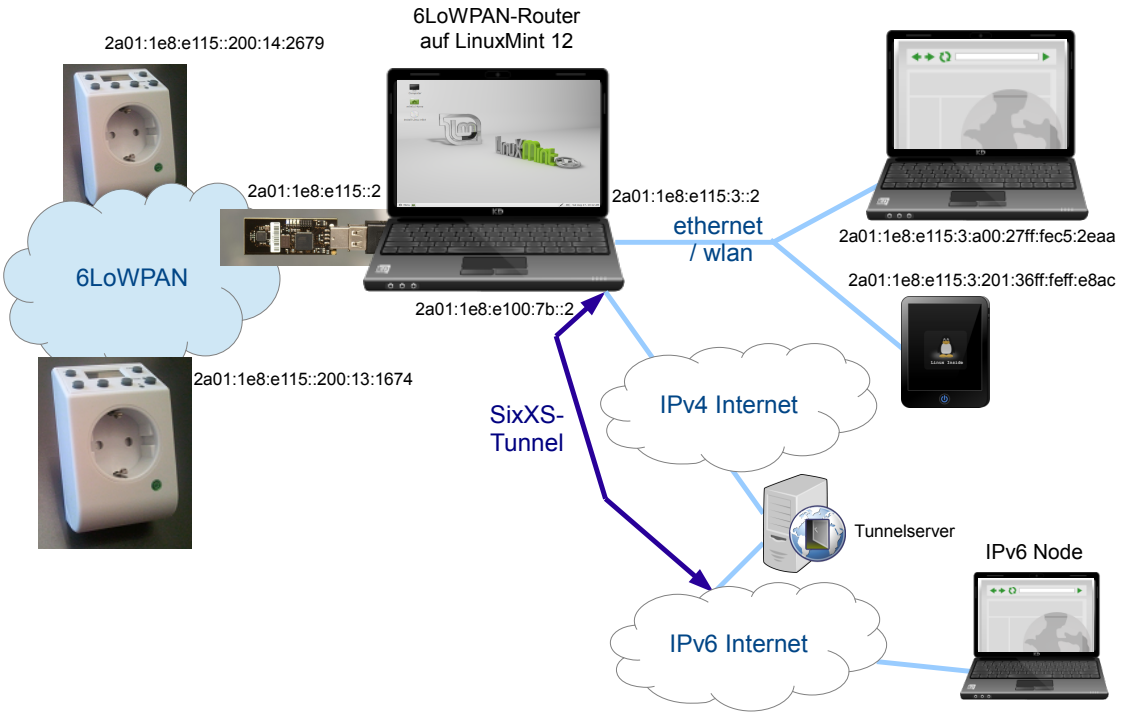

Abbildung 7.2: Testaufbau Steckdosen mit öffentlichen IP Adressen

Der Testaufbau wurde später erweitert (siehe Abbildung 7.2). Es wurden öffentliche IP-Adressen

von dem IPv6-Tunnelbroker SixXS [Web:SixXS] verwendet. Damit ist es möglich, auch mit einem nativen IPv4-Internetanschluss auf das IPv6-Internet zugreifen zu können. Es wird dabei über das IPv4-Internet eine Verbindung zu einem Tunnelserver aufgebaut und zwischen beiden Endpunkten ein IPv6-Tunnel [ID-v6ops-ayiya] aufgebaut. Dazu muss auf dem 6LoWPAN-Router die Software aiccu [Web:aiccu] installiert und über die Konfigurationsdatei /etc/aiccu.conf konfiguriert werden. Auf dem 6LoWPAN-Router wird dann ein virtuelles Interface sixxs erstellt mit der IPv6-Adresse, die von SixXS zugewiesen wurde. Damit ist diese IPv6-Adresse von allen IPv6-Nodes erreichbar, die ebenfalls eine Verbindung ins IPv6-Internet haben.
Es ist möglich, von SixXS nicht nur eine einzelne IPv6-Adresse, sondern ein ganzes Präfix zugewiesen zu bekommen. So können auch zusätzliche Geräte über den gleichen IPv6-Tunnel ins Internet angeschlossen werden. Der 6LoWPAN-Router fungiert dabei auch als IPv6-Router ins Internet. In dem Testaufbau wurden von dem zugewiesenen Präfix 2a01:1e8:e115::/48 zwei Subnetze erstellt. Das erste Subnetz 2a01:1e8:e115::/64 wird im 6LoWPAN-Netzwerk betrieben, das zweite Subnetz 2a01:1e8:e115:3::/64 in einem gemischten Ethernet-/WLAN-Netzwerk. Hier können bei Bedarf ein PC mit Windows 7, ein PC mit Solaris 10, ein PC mit Linux Mint 13 oder ein Tablet mit Android 2.3 angeschlossen werden.

7.3 Funktionstest der Webserver Applikation

Es wurde von verschiedenen Geräten aus dem Ethernet-/WLAN-Netzwerk auf die Steckdosen zugegriffen. Dabei wurde die IP-Adresse der Steckdose in die Adressleiste des Browsers eingegeben. Nachfolgende Abbildungen zeigen die Informationen, die per Browser von der Applikation webserver-nano-steckdose abgerufen werden können. Für die folgenden Abbildungen wurde der Browser Opera 12.15 von einem PC mit Windows 7 verwendet.

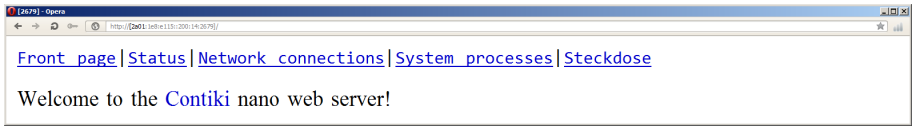

Abbildung 7.3: Startseite dargestellt im Webbrowser

Die Startseite (Abbildung 7.3) zeigt den Header mit den verschiedenen Seiten, auf die zugegriffen werden kann. Der Header ist bei allen Seiten identisch.

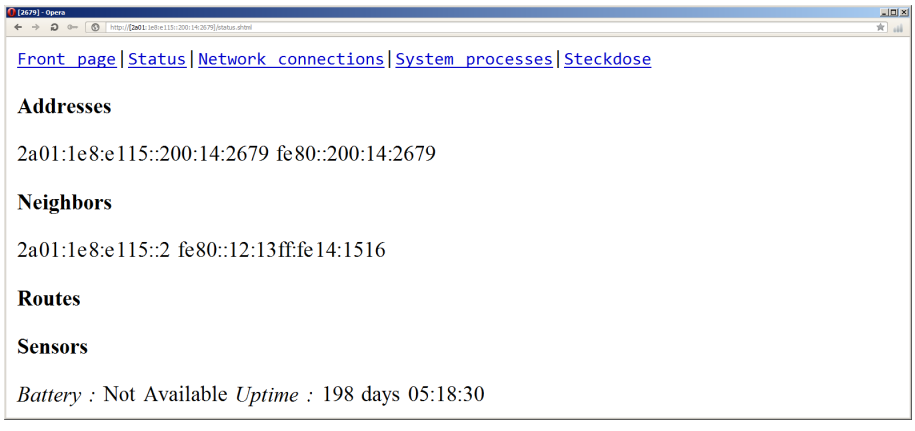

Abbildung 7.4: Webseite status.shtml dargestellt im Webbrowser

Hinter dem Link „Status“ verbirgt sich die Webseite status.shtml (Abbildung 7.4). Hier werden allgemeine Systeminformationen dargestellt. Unter „Addresses“ werden die eigenen IP-Adressen dargestellt. Die globale IP 2a01:1e8:e115::200:14:2679 und die link-lokale IP fe80::200:14:2679.

Die benachbarten IP-Adressen werden unter „Neighbors“ aufgelistet. Hier sind es die globale und die link-lokale IP-Adresse des RZUSBSTICK am 6LoWPAN-Router. Außerdem wird die „Uptime“ der Steckdose angezeigt, also die Betriebszeit der Steckdose. Hier sind es 198 Tage, 5 Stunden, 18 Minuten und 30 Sekunden.

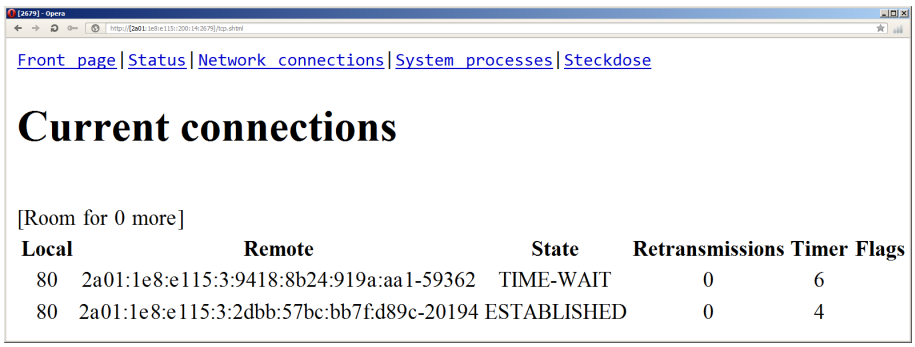

Abbildung 7.5: Webseite tcp.shtml dargestellt im Webbrowser

In Abbildung 7.5 wird die Darstellung der Webseite tcp.shtml angezeigt; sie verbirgt sich hinter dem Link „Network connections“. Hier sind alle aktuellen TCP/IP-Verbindungen der Steckdose zu sehen. Zu sehen sind der lokale TCP-Port und die Remote IP-Adresse mit dem zugehörigen TCP-Port und der Status der Verbindung. Die erste Verbindung zeigt einen Zugriff von einem PC unter Linux Mint auf die Steckdose. Die zweite Verbindung ist die Verbindung, mit der die Webseite tcp.shtml abgerufen worden ist. Weil der Zugriff von zwei verschiedenen Geräten erfolgte, sind zwei verschiedene IP-Adressen zu sehen.

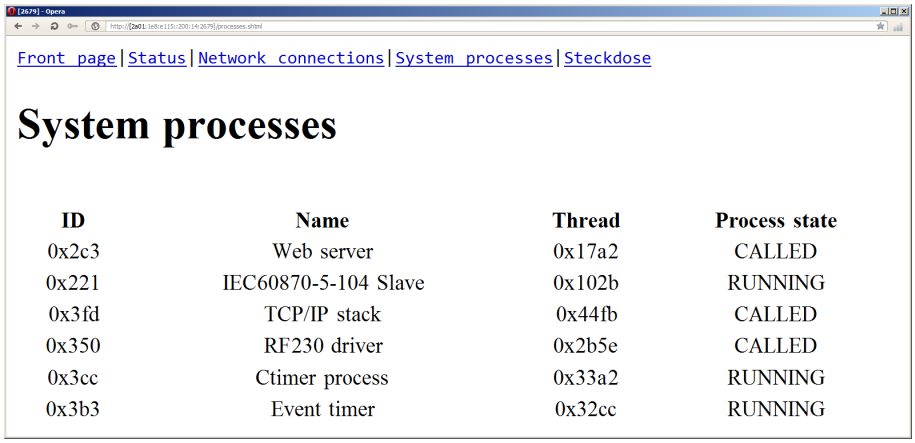

Abbildung 7.6: Webseite processes.shtml dargestellt im Webbrowser

Durch einen Klick auf „System processes“ wird die Webseite processes.shtml aufgerufen (Abbildung 7.6). Hier werden die Contiki-Prozesse mit ihren Namen und ihren aktuellen Status dargestellt. Die Applikation webserver-nano-steckdose läuft in dem Prozess mit dem Namen „Web server“. Die Applikation iec104 läuft im Prozess „IEC60870-5-104 Slave“. Der TCP/IP-Stack und der Treiber für den Funkchip AT86RF230 sind zwei eigene Prozesse. Außerdem laufen noch Timer-Prozesse, die vom Betriebssystem verwendet werden. Es ist erkennbar, dass bei einer Anfrage auf den Web Server die Prozesse „RF230 driver“, „TCP/IP stack“ und „Web server“ aufgerufen werden.
Hinter dem Link „Steckdose“ verbirgt sich die Webseite pins.shtml. Sie ist in Abbildung 7.7 dargestellt. Es wird der aktuelle Zustand der Steckdose bzw. des Relais dargestellt – EIN oder AUS. Über die beiden Schaltflächen rechts ist es möglich, das Relais zu schließen und zu öffnen, und

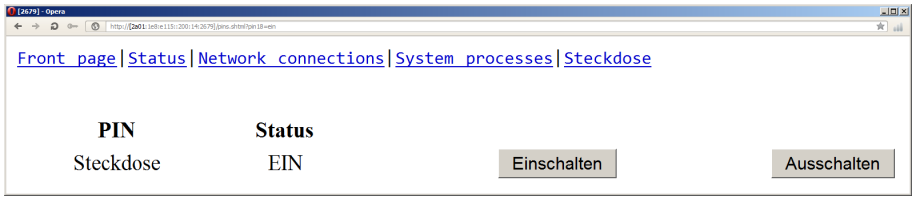

Abbildung 7.7: Webseite pins.shtml dargestellt im Webbrowser

damit den elektrischen Verbraucher, der an diese Steckdose angeschlossen ist, ein- und auszuschalten.
Folgende andere Webbrowser wurden ebenfalls getestet:

- Firefox 19.0.2 unter Windows 7
- Internet Explorer 10 unter Windows 7
- Firefox 19.0.2 unter Linux Mint 13
- Mozilla 1.7 unter Solaris 10
- Lynx 2.8.8dev.9 unter Linux Mint 12
- Browser unter Android 2.3.4

In allen Browsern wurden die Webseiten gut dargestellt und es war möglich, die Steckdose ein- und auszuschalten. Nur der Internet Explorer hatte einen kleinen Makel: bei ihm wurde bei der Webseite status.shtml unter dem Punkt „Addresses“ und „Neighbors“ nichts angezeigt. Der Grund dafür wurde nicht weiter verfolgt.

Listing 7.1: tcpdump (formatiert) bei Anfrage auf Webseite pins.shtml

```
16:28:08.302583 IP6 2a01:1e8:e115::2.56032 > 2a01:1e8:e115::200:14:2679.80:
    GET /pins.shtml HTTP/1.0
    Host: [2a01:1e8:e115::200:14:2679]
    Accept: text/html, text/plain, text/css, text/sgml, */*;q=0.01
    Accept-Encoding: gzip, compress, bzip2
    Accept-Language: en
    User-Agent: Lynx/2.8.8dev.9 libwww-FM/2.14 SSL-MM/1.4.1 GNUTLS/2.10.5

16:28:08.362450 IP6 2a01:1e8:e115::200:14:2679.80 > 2a01:1e8:e115::2.56032:
    HTTP/1.0 200 OK
    Server: Contiki/2.0 http://www.sics.se/contiki/
    Connection: close
16:28:08.381447 IP6 2a01:1e8:e115::200:14:2679.80 > 2a01:1e8:e115::2.56032:
    Content-type: text/html
16:28:08.420437 IP6 2a01:1e8:e115::200:14:2679.80 > 2a01:1e8:e115::2.56032:
    <html><head><title>[2679]</title></head><body><pre>
    <a href="/">Front page</a>|<a href="status.shtml">Status</a>|
    <a href="tcp.shtml">Network connections</a>|
    <a href="processes.shtml">System processes</a>|
    <a href="pins.shtml">Steckdose</a></pre>
16:28:08.439437 IP6 2a01:1e8:e115::200:14:2679.80 > 2a01:1e8:e115::2.56032:
    <br><table width="100
16:28:08.467453 IP6 2a01:1e8:e115::200:14:2679.80 > 2a01:1e8:e115::2.56032:
    %"><tr><th>PIN</th><th>Status</th></tr><tr align="center">
16:28:08.487439 IP6 2a01:1e8:e115::200:14:2679.80 > 2a01:1e8:e115::2.56032:
    <td>Steckdose</td><td>EIN</td>
16:28:08.541443 IP6 2a01:1e8:e115::200:14:2679.80 > 2a01:1e8:e115::2.56032:
    <td><form action="pins.shtml" method="get" align="right">
    <input type="hidden" name="pin18" value="ein">
    <input type="submit" value="Einschalten"></form></td>
    <td><form action="pins.shtml" method="get" align="right">
```

```
<input type="hidden" name="pin18" value="aus">
<input type="submit" value="Ausschalten"></form></td>
</tr></table>
```

Das Listing 7.1 zeigt eine Anfrage auf die Webseite pins.shtml. Es ist erkennbar, dass die Antwort von der Steckdose in mehrere TCP/IP-Pakete aufgeteilt ist. Das liegt an der Verarbeitung in der Applikation webserver-nano-steckdose. Die Antwort wird nicht lokal zusammengestellt und dann als ein ganzes Packet versendet, sondern während des Sendens erzeugt. Verschiedene Funktionen aus der Datei apps/webserver-nano-steckdose/httpd.c übernehmen dabei verschiedene Teile der Antwort. Der HTTP-Header ist in zwei Teilen aufgebaut. Zuerst kommt der HTTP-Status, der in der Funktion generate_status() erzeugt wird. Darauf folgt die Art der Information (Content-type), erzeugt in der Funktion generate_header(). Dann wird die angefragte Datei verarbeitet. Der Inhalt der Datei besteht aus statischen und aus dynamischen Informationen. Die dynamischen Informationen bestehen aus sogenannten Schlüsselwörter. Es wird abwechselnd der statische Teil mittels der Funktion send_part_of_file() und der dynamische Teil mittels der Funktion httpd_cgi() abgearbeitet. Bei jedem Wechsel wird ein neues TCP/IP-Paket erzeugt. Insgesamt beträgt die Zugriffszeit bei dem obigen Beispiel 0,25 Sekunden.

7.4 Funktionstest der IEC104 Applikation

Zum Test der Applikation iec104 wurde das Netzleitsystem Spectrum Power von der Firma Siemens AG verwendet. Nachfolgend wird das Netzleitsystem vorgestellt und wie die Steckdose darin konfiguriert wurde. Es wird die Bedeutung von verschiedenen IEC104 Time-out Parametern erläutert und wie sie im Spectrum Power konfiguriert sind. Zum Schluss wird auf die eigentliche IEC104-Verbindung zwischen Spectrum Power und der Steckdose eingegangen.

7.4.1 Netzleitsystem Spectrum Power

Spectrum Power ist ein SCADA (Supervisory Control and Data Aquisition) System, das vor allem bei Energieversorgungsunternehmen zum Einsatz kommt. Es dient dazu, das elektrische Netz zu überwachen und zu steuern. Ziel eines Netzleitsystems ist es, die Verfügbarkeit des Netzes zu steigern. Es soll schneller und gezielter auf Störfälle reagiert werden. Spectrum Power bietet ein großes Lösungsspektrum mit innovativen und erprobten Komponenten, die zu einer individuellen Lösung konfiguriert werden. Es bietet zahlreiche Werkzeuge, die beim Energiemanagement unterstützen sollen. Darunter zum Beispiel Lastmanagement, Prognoseberechnungen und Kraftwerksführung [siemens:netzleittechnik].
Beim Test kam die Version Spectrum Power 4.7 zum Einsatz. Es ist ein redundantes Mehrrechnersystem auf Basis von Unix, das zum Leiten von großen vermaschten Netzen verwendet wird. Es richtet sich an große Energieverteilunternehmen und Energietransportunternehmen, sowie an Bahnunternehmen mit großen Bahnstromnetzen. Das zum Test verwendete System läuft auf einem PC mit dem Betriebsystem Solaris 10 und folgenden Komponenten:

- Basissystem
- Dateneingabesystem
- Archivsystem
- Prozessanschluss
- Bedienoberfläche

Der Prozessanschluss ist der Teil des Systems, der für die Verbindung zu den sogenannten Fernwirkgeräten – auch RTU (Remote Terminal Unit) genannt – zuständig ist. Das sind Geräte, die in Unterstationen installiert sind und Informationen über die Unterstation zur Leitstelle liefern

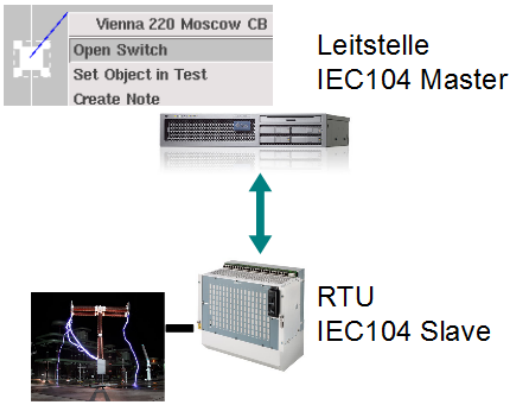

Abbildung 7.8: Verbindung Leitstelle und RTU

und Befehle von der Leitstelle entgegen nehmen und ausführen. Solche Informationen sind zum Beispiel der Schalterzustand eines Hochleistungsschalters oder ein Messwert über den aktuellen Stromfluss. Diese Informationen werden dann in der Bedienoberfläche für die Bediener des Netzleitsystems dargestellt. Es ist auch möglich, Befehle über die Bedienoberfläche auszuführen, die dann an eine RTU gesendet werden. So ein Befehl kann zum Beispiel sein, einen Hochleistungsschalter für einen bestimmten Abzweig ein- oder auszuschalten. Der Prozessanschluss – zusammen mit der RTU – ist das Bindeglied zwischen Bedienoberfläche und physikalischem Element im Feld. Die Kommunikation zwischen Prozessanschluss und RTU ist über Fernwirkprotokolle geregelt. Weltweit ist eine Vielzahl von standardisierten und proprietären Fernwirkprotokollen im Einsatz. Ein sehr verbreitetes Protokoll, das zum einem standardisiert ist und zum anderen auf TCP/IP aufbaut, ist IEC60870-5-104. Der Prozessanschluss von Spectrum Power unterstützt unter anderen dieses Protokoll. Eine Verbindung von Leitstelle zur RTU bzw. zum Element im Feld ist schematisch in Abbildung 7.8 dargestellt.

7.4.2 Konfiguration der Steckdose in Spectrum Power

Damit das Netzleitsystem eine Verbindung zur Steckdose aufbaut, muss diese im System konfiguriert werden. Das wird mithilfe des Dateneingabesystem SDM (Source Data Management) gemacht. Nachfolgend wird die Konfiguration der Steckdose vorgestellt, so wie sie im Prozessanschluss von Spectrum Power durchgeführt worden ist.

Pair Id: 1

Chan. Id	Backup Chan. Id	IP Address	Config Type	Baud Rate	Char Size	Parity	Rts Delay [msec]	Max Retries	Timeout [sec]	Fe1 B1 Name	Fe1 B2 Name	Fe1 B3 Name	Fe2 B1 Name	Fe2 B2 Name	Fe2 B3 Name
5	0	2a01:1e8:e115::	1	9600	8	0	0	1	1	IFS	IF11	Chan 05	IFS	IF12	Chan 05

Abbildung 7.9: SDM Konfiguration der IP Adresse

Es ist notwendig, für die Verbindung zur Steckdose einen virtuellen Kanal zu konfigurieren. Dies ist in Abbildung 7.9 zu sehen. Dieser virtuelle Kanal bekommt zuerst eine eindeutige Nummer (Chan. Id), damit er später identifiziert werden kann. Identifiziert wird er dabei über die Pair Id und die Channel Id. Die Pair Id repräsentiert den Prozessanschluss als redundantes Server-Paar. Bei großen Systemen reicht ein Server-Paar nicht aus, um alle RTUs anzuschließen. In dem Fall sind mehrere Server-Paare notwendig. Zum Testen wurde nur ein einzelner, nicht redundanter Server verwendet, weswegen nur die Pair Id=1 vorhanden ist. Der virtuelle Kanal in der Abbildung wird über die Parameter Pair Id=1 und Chan. Id=5 identifiziert. Der wichtigste Parameter ist die IP-Adresse der Steckdose, in Abbildung 7.9 leider nicht vollständig zu sehen, weil das Feld für IPv4-

Adressen dimensioniert ist. Es ist jedoch die vollständige IP-Adresse 2a01:1e8:e115::200:13:1674 konfiguriert worden.

Pair Id: 1 Channel Id: 5 Protocol_name: IEC104

RTU-Name	Protocol ID	Link & ASDU Size	Not in use (High | Low) Switch Struct		ASDU Addr (High | Low) Switch Struct		Gi Timeout	Gi Retries	Clock Sync	Old New	Obj Size	B1 Name	B2 Name	B3 Name	Master RTU-Name
Srv.002	5	6	0	0	0	14	30	1	0	1	3	IFS	RTU	Srv.002	0

Abbildung 7.10: SDM Konfiguration der RTU

Als nächstes wird die logische RTU Steckdose konfiguriert (Abbildung 7.10). Dazu wird der virtuelle Kanal ausgewählt, auf dem diese RTU liegen soll. Es wird ein Name vergeben, in diesem Fall „Srv.002“, das verwendete Fernwirkprotokoll wird ausgewählt. Protocol ID=5 steht für „IEC104“. Ein Parameter, der mit der Parametrierung der Steckdose abgestimmt werden muss, ist die Common Adresse oder auch ASDU Adresse. Sie wurde in der Contiki-Applikation iec104 in der Datei iec104-para.h parametriert (siehe Listing 6.18) und hat den Wert 14.

Pair Id: 1 Channel Id: 5 RTU Name: Srv.002 Structured address Mixed 1 Mixed 2 Unstructured address

B1-Name	B2-Name	B3-Name	Element	Info Show Text	Mon Address			Mon Typ.	Cur. In	Con Address			Con Type	Cur. Out	Sel Bef	Cmd Dur	GI	ISS	Dbl	Thr-esh	Inv	Zero range
Vienna	220	Rome	CB	Status	0	1	18	0	0	1	1	18	45	0	0	0	1	0	0	0	0	0.0
Vienna	220	Rome	Iso Eth	Status	0	1	20	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0.0
Vienna	220	Rome	Iso Line	Status	0	1	17	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0.0
Vienna	220	Rome	Iso Bb 1	Status	0	1	19	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0.0

Abbildung 7.11: SDM Konfiguration der Datenpunkte

In der Abbildung 7.11 ist ein Auszug der Parametrierung der Informationsobjekte – oder auch Datenpunkte – der Steckdose zu sehen. Es wird ausgewählt, auf welche RTU – mit Pair Id und Chan. Id – sich die Datenpunkte beziehen. Der linke Teil eines Datenpunktes (B1-Name, B2-Name, B3-Name, Element und Info) ist die interne Adressierung eines Datenpunktes im Spectrum Power System, auch technologische Adresse oder kurz TA genannt. Es handelt sich um eine 5-stufige Adresse, damit die einzelnen Datenpunkte im System strukturiert werden können. In der Regel bezeichnet der B1-Name eine Unterstation im elektrischen Netz, der B2-Name die Spannungsebene und der B3-Name den Abzweig. Element und Info repräsentieren das physikalische Element im Feld und die Information, die von diesem Element verfügbar ist. Der erste beschriebene Datenpunkt (Vienna / 220 / Rome / CB / Status) wurde verwendet, um das Relais in der Steckdose zu repräsentieren. Weitere Datenpunkte repräsentieren andere Pins des Zigbit-Moduls, die aktuell aber nicht verschaltet sind.

Der rechte Teil der Datenpunkt-Konfiguration behandelt verschiedene Verarbeitungsfunktionen des Prozessanschlusses, auf die hier aber nicht näher eingegangen wird. Der mittlere Teil ist die IEC104-Adressierung eines Datenpunktes, wie er in der Steckdose in der Datei iec104-para.c parametriert worden ist (siehe Listing 6.19). Bei dem ersten Datenpunkt ist die Adresse in Monitorrichtung 0/1/18 und in Kommandorichtung 1/1/18. In Kommandorichtung wird noch der IEC-Datentyp parametriert, Con Type=45 ist ein Einzelbitbefehl C_SC_NA_1.

Die obigen Abbildungen zeigen die Konfiguration der ersten Steckdose. Die zweite Steckdose ist analog dazu im Spectrum Power System konfiguriert. Es wurde die Channel Id=4 mit der IP-Adresse 2a01:1e8:e115::200:14:2679 konfiguriert. Der RTU-Name dieser Steckdose lautet „Srv.001“. Der Datenpunkt für das Informationsobjekt Relais wurde auf die technologische Adresse Vienna / 220 / Moscow / CB / Status gelegt.

7.4.3 IEC104 Time-out Parameter

Die TCP/IP-Verbindung wird bei IEC104 dauerhaft aufgebaut gehalten. Wenn keine Datentelegramme ausgetauscht werden, wird mithilfe von Testtelegrammen die Verfügbarkeit der Verbindung überwacht. Sobald eine Verbindung abbricht bzw. sobald eine RTU nicht mehr erreichbar ist, wird das als Alarm im Spectrum Power System signalisiert. Alle Datenpunkte, die von einer solchen RTU empfangen werden, werden in dem Fall im System als Nicht-Erneuert markiert.

Parameter	Standardwert	Bemerkungen
t_0	30 s	Time-out für Verbindungsaufbau
t_1	15 s	Time-out zum Senden oder Testen von APDUs
t_2	10 s	Time-out für Quittierungen im Falle von keiner Datenübertragung $t_2 < t_1$
t_3	20 s	Time-out zum Senden von Test Frames im Falle eines längeren Ruhezustandes
k	12 APDUs	Maximaler Unterschied zwischen empfangener Receive Sequence Number und zuletzt gesendeter Send Sequence Number
w	8 APDUs	Nach Emfpang von w APDUs in I-Format wird spätestens eine Quittierung gesendet

Tabelle 7.1: IEC104 Time-out Parameter [IEC104]

Für die Überwachung der Verbindung wurden im IEC104-Standard vier Zeitparameter und zwei APDU-Parameter definiert. Sie sind in der Tabelle 7.1 aufgelistet. Im Normalfall werden diese Parameter von beiden Seiten – vom IEC104 Master und vom IEC104 Slave – gesetzt und miteinander abgeglichen. Um die Implementierung im Contiki möglichst einfach zu halten, wurden diese Parameter größtenteils nicht berücksichtigt. Das kann aber nur funktionieren, indem von folgenden Voraussetzungen ausgegangen wird.

- Die Quittierung von gesendeten APDUs wird in der Steckdose nicht geprüft. Die Information, welche APDUs noch nicht quittiert worden sind, wird zwar im Puffer gehalten, aber es passiert nichts, wenn eine Quittierung ausbleibt. Hier wird angenommen, dass die Quittierung auf TCP/IP-Ebene ausreichend ist.

- Es wird angenommen, dass die Test Frames bei einem längeren Ruhezustand vom IEC104 Master gesendet werden. Die Steckdose quittiert diese Telegramme, sendet sie aber nicht von sich aus. Der Standard sieht vor, dass beide Seiten Test Frames senden dürfen.

- Empfangene APDUs in I-Format müssen von der Steckdose quittiert werden. Dies soll spätestens nach Empfang von w APDUs oder nach Ablauf der Zeit t_2 gemacht werden. Empfangene APDUs werden aber auch automatisch beim Senden von APDUs in I-Format quittiert. Der Parameter w=8 ist zwar in der Implementierung berücksichtigt, aber nicht die Zeit t_2. Empfangene APDUs können ja nur ein Einzelbitbefehl oder eine Generalabfrage sein (siehe Listing 6.21). Es wird davon ausgegangen, dass jedes empfangene APDU in I-Format auf Applikationsebene ein Senden von APDUs in I-Format auslöst und somit automatisch sofort quittiert wird.

Die letzte Voraussetzung bedeutet, dass der Empfang eines unbekannten ASDU-Typs nicht auf Linkebene quittiert werden würde. Das könnte zu einem Verbindungsabbruch führen. Um das zu kompensieren, könnte der Parameter w (cWIN_SIZE) in der Datei apps/iec104/iec104-link.h auf den Wert 1 gesetzt werden. Damit ist gewährleistet, dass jedes empfangene APDU in I-Format quittiert wird. Die Zeit t_2 hat dann keine Bedeutung mehr.

Listing 7.2: IEC104 Time-out Parameter in Spectrum Power

```
#  Timeout for connection establishment: t0, value in seconds                T0
#  Recomended value from IEC: 30 (sec).
TIMEOUT_CONNECT_EST=120
#  Timeout for send or test APDUs: t1, value in seconds                       T1
#  Recomended value from IEC: 15 (sec).
TIMEOUT_SEND_APDU=15
#  Timeout for acknowledges in case of no data messages: t2 (t2 < t1)         T2
#  Recomended value from IEC: 10 (sec).
TIMEOUT_ACKNOWLEDGE=40
#  Timeout for sending test frames in case of a long idle state, t3           T3
#  Recomended value from IEC: 20 (sec).
TEST_COMMAND_PERIOD=20
=============================================================================
/* Maximum number of outstanding I Frames: k                                 */
/* Defines the maximum number of sequentially numbered I frames that         */
/* may be outstanding (i.e. unacknowledged) at any given time.               */
/* Default value from Norwegian User Conventions 104: 12                     */
#define MAX_OUTSTANDING_I_FRAMES 12
/* Maximum number of received I frames before acknowledge: w                 */
/* Default value from Norwegian User Conventions 104: 8                      */
#define MAX_NOT_ACK_I_FRAMES 8
```

In Listing 7.2 sind die Parameter aufgelistet, wie sie beim Test im Spectrum Power gesetzt sind. Es wurden die Standardwerte verwendet. Nur der Parameter t_0 wurde auf 120 Sekunden erhöht, um dem Versuch zum Verbindsaufbau mehr Zeit zu geben. Es entlastet auch die Verbindung für den Fall, dass eine Steckdose nicht erreichbar ist.

7.4.4 Verbindungstest

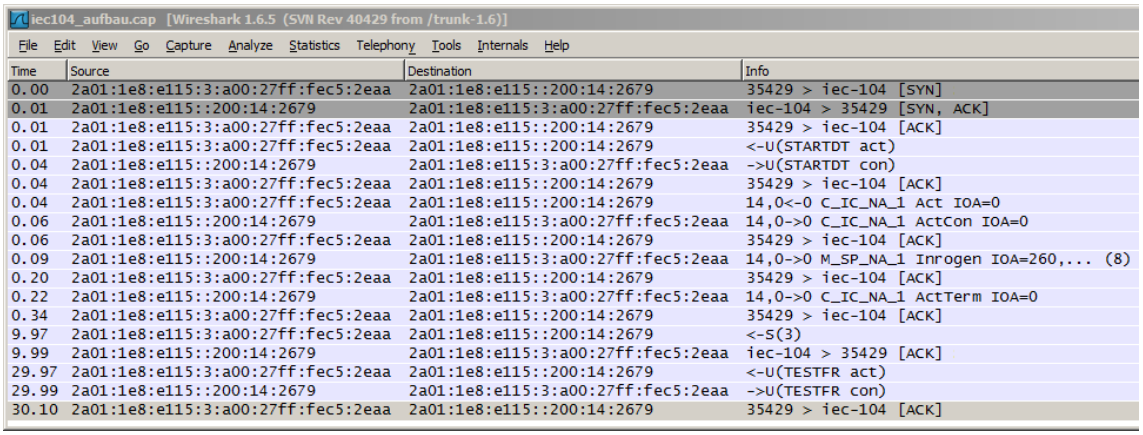

Abbildung 7.12: Verbindungsaufbau IEC104 im Wireshark

Nachdem die Steckdosen im Spectrum Power System konfiguriert und eingeschaltet worden sind, wird der Prozessanschluss einen Verbindungsaufbau versuchen. Wenn die Steckdose erreichbar ist, wird sie den Verbindungsaufbau zulassen. In Abbildung 7.12 ist der zugehörige Netzwerkverkehr im Wireshark dargestellt. Sobald mit einem SYN, SYN ACK, ACK die TCP/IP-Verbindung aufgebaut ist, ist das erste Telegramm vom Prozessanschluss das StartDT act, was von der Steckdose mit einem StartDT con bestätigt wird. Damit ist es der Steckdose erlaubt, Datentelegramme zu senden. Der Prozessanschluss sendet direkt danach eine initiale Generalabfrage, um den aktuellen Zustand von allen Informationsobjekten zu erhalten. Der Verbindungsaufbau besteht also aus drei Schritten:

1. TCP/IP-Verbindungsaufbau
2. StartDT (Start Data Transmission)
3. Initiale Generalabfrage

Die Generabfrage besteht dabei aus einem C_IC_NA_1 Act (Activation) vom Spectrum Power. Die Steckdose antwortet dann mit einem C_IC_NA_1 ActCon (Activation Confirmation) um anzuzeigen, dass der Generalabfragebefehl verstanden worden ist. Es folgen die Informationsobjekte vom Typ M_SP_NA_1 als geblocktes Telegramm, das heißt, es werden alle konfigurierten Informationsobjekte in einem Telegramm übertragen. Das letzte Telegramm des Verbindungsaufbaus ist ein C_IC_NA_1 ActTerm (Activation Termination), um dem Prozessanschluss das Ende der Generalabfrage anzuzeigen. Der gesamte Verbindungsaufbau beträgt 0,22 Sekunden. Es folgt noch ein APCI-Telegramm im S-Format von Spectrum Power, um die empfangenen Telegramme zu quittieren. Weil danach keine Datentelegramme mehr ausgetauscht werden, sendet Spectrum Power ein TESTFR act, was mit einem TESTFR con bestätigt wird.
Mit dem Ende der Generalabfrage werden dem Bediener vom Spectrum Power System signalisiert, dass die Verbindung zur RTU aufgebaut ist und dass die empfangenen Informationsobjekte den aktuellen Zustand aus dem Feld haben – die Datenpunkte werden als Aktuell markiert, vorher waren sie Nicht-Erneuert. Jetzt ist es dem Bediener erlaubt, Befehle zu senden bzw. der Steckdose ist es erlaubt, spontane Datenänderungen zu senden. Wenn keine Datentelegramme gesendet werden, wird der Status der Verbindung mittels Test Frames geprüft.

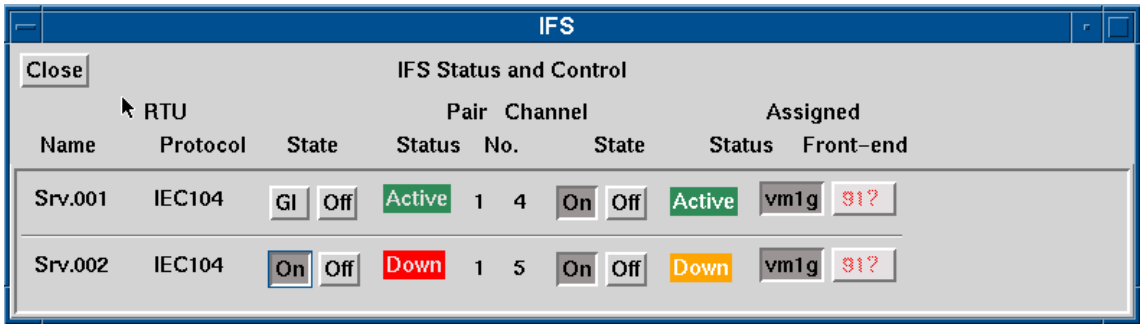

Abbildung 7.13: RTU Statusübersichtsbild im Spectrum Power

Der aktuelle Status der Verbindung wird in einem RTU Statusübersichtsbild angezeigt (Abbildung 7.13). Es wird für jede konfigurierte RTU der Status des virtuellen Kanals und der Status der logischen RTU angezeigt. Im Beispiel ist die Verbindung zur RTU „Srv.001“ aufgebaut und die RTU ist aktiv. Die RTU „Srv.002“ und die Verbindung dorthin ist nicht aktiv. Über diese Statusübersichtsbild ist es auch möglich, mittels der Schaltfläche „GI“ (General Interrogation) manuell eine Generalabfrage zu senden. Im Statusfeld der RTU wird dann angezeigt, dass eine Generalabfrage läuft.
Abbildung 7.14 zeigt das Stationsbild Vienna im Spectrum Power. Es werden die elektrischen Leitungen dargestellt und die Schalterelemente im Feld. Ein Kreis repräsentiert dabei einen Leistungstrennschalter und ein Quadrat einen Leistungsschalter. Ein geschlossener Kreis und ein geschlossenes Quadrat bedeuten, dass der Schalter geschlossen ist. Ein offener Kreis und ein offenes Quadrat zeigen einen offenen Schalter. Der ausgewählte Leistungsschalter Vienna / 220 / Moscow / CB / Status ist das Relais in der Steckdose mit der IP-Adresse 2a01:1e8:e115::200:14:2679 (RTU Srv.001). Im Moment zeigt es den Zustand „geschlossen“. Das bedeutet, der elektrische Verbraucher, der an der Steckdose verbunden ist, ist gerade eingeschaltet. Mit dem Befehl „Open Switch“ wird die Steckdose aufgefordert, das Relais zu öffnen.
In Abbildung 7.15 ist die Befehlsprozedur im Wireshark zu sehen. Sie beginnt mit dem Einzelbitbefehl C_SC_NA_1 Act mit der zugehörigen Informationsobjektadresse. Sie ist im Wireshark als

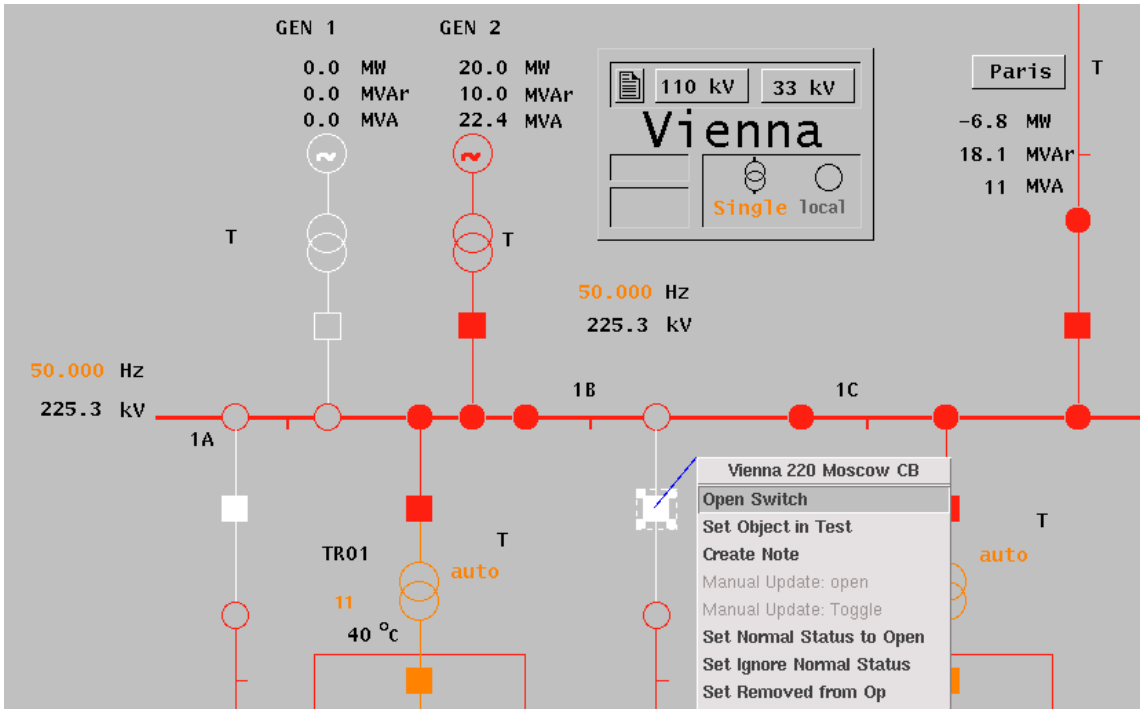

Abbildung 7.14: Bedienoberfläche der Station Vienna im Spectrum Power

unstrukturierte Adresse zu sehen. Es wurden die drei Bytes der Informationsobjektadresse als eine Nummer zusammenaddiert. IOA=65810 ist umgerechnet 1/1/18, was der Adresse entspricht, wie sie in der Steckdose in der Datei iec104-para.c bzw. im Spectrum Power System im SDM konfiguriert worden ist. Die Steckdose antwortet mit einem C_SC_NA_1 ActCon, um den Empfang des Befehls zu bestätigen. Mit dem Telegram C_SC_NA_1 ActTerm wird dem Prozessanschluss mitgeteilt, dass der Befehl ausgeführt worden ist. Die Änderung selbst kommt etwa zwei Sekunden später mit dem Telegramm M_SP_NA_1 und der Informationsobjektadresse IOA=274, was umgerechnet 0/1/18 entspricht. Es ist die Information, dass sich der Zustand des Relais geändert hat. Die Ausführung des Befehls bis zum Empfang der Rückmeldung hat 2,27 Sekunden gedauert.

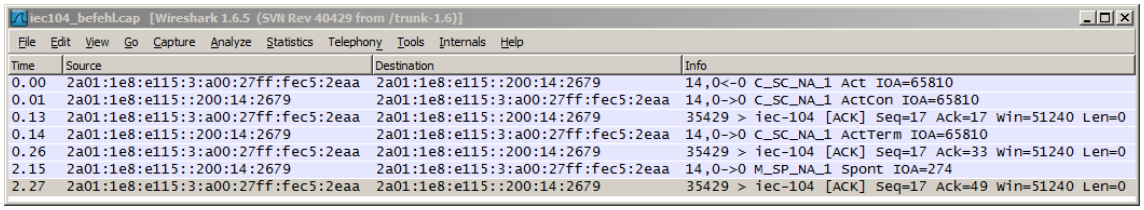

Abbildung 7.15: IEC104 Befehlsprozedur im Wireshark

Parallel zu der IEC104-Verbindung läuft noch der Webserver auf der Steckdose. Ein Schalten der Steckdose über den Webserver wurde nach ungefähr zwei bis drei Sekunden in der Bedienoberfläche von Spectrum Power als spontane Änderung angezeigt. Wenn über Spectrum Power der Zustand der Steckdose geändert worden ist und danach die Webseite der Steckdose neu aufgerufen wurde, wurde der geänderte Zustand dort auch angezeigt. Beide Applikationen laufen also parallel und können parallel Befehle zum Ändern des Relais-Zustandes entgegen nehmen und ausführen. Änderungen werden über IEC104 sofort spontan mitgeteilt, über den Webbrowser muss die entsprechende Seite neu aufgerufen werden.

7.5 Sicherheitsaspekte

Bei beiden Applikationen webserver-nano-steckdose und iec104 sind weder Authentifizierungs- noch Verschlüsselungsmechanismen implementiert. Bei der Webserver Applikation könnte das zum Beispiel durch die Verwendung von HTTPS statt HTTP erfolgen. Dies wurde aber bisher noch nicht im Contiki implementiert und soll auch nicht Bestandteil dieser Arbeit sein. Für das Fernwirkprotokoll IEC104 arbeitet der IEC Standard an verschiedene Sicherheitsmechanismen. Nach heutigem Stand ist es übliche Praxis, den Netzwerkverkehr über die Netztopologie abzusichern. Viele Energieversorgungsunternehmen unterhalten eine eigene Infrastruktur für den Anschluss von RTUs an die Netzleitstelle, getrennt von sämtlicher anderer interner und externer Kommunikation. Wenn doch auf eine Infrastruktur von einem Provider zugegriffen wird, wird diese über einen VPN-Tunnel abgesichert.
Zusätzlich ist es auch möglich, eine Authentifizierung und Verschlüsselung auf unteren Schichten der Kommunikationstechnologie durchzuführen. Mit ContikiSec [Casado und Tsigas 2009] existiert eine Implementierung für das Betriebsystem Contiki, die auf die Sicherungsschicht wirkt. Eine andere Implementierung Contiki-dtls [Perelman 2012] sichert auf der Transportschicht. Beide Implementierungen sind aber – soweit bekannt – nicht Teil des Betriebssystems Contiki.
Ein weiter Test der Steckdose wurde mithilfe des Programms Nmap 6.25 [Web:nmap] durchgeführt. Folgender Befehl führt einen intensiven Netzwerk-Scan aus:

Listing 7.3: nmap Befehl zum Scan der Steckdose

```
nmap -6 -p 1-65535 -T4 -A -v 2a01:1e8:e115::200:14:2679
```

Der Scan prüft unter anderem nach offenen TCP-Ports. Dazu versucht es eine TCP/IP-Verbindung auf alle möglichen TCP-Ports aufzubauen, indem zu allen Ports ein SYN-Packet gesendet wird. Dieser gesamte Scan hat etwa zwanzig Minuten gedauert. Während des Scans war die Steckdose via Webbrowser erreichbar und es war möglich, das Relais der Steckdose ein- und auszuschalten. Allerdings war die Bedienung der Steckdose sehr verzögert. Der Scan hat zwei offene TCP-Ports gefunden, das Betriebssystem konnte nicht ermittelt werden:

Listing 7.4: Ergebnis des nmap Scans

```
PORT     STATE SERVICE VERSION
80/tcp   open  http    Contiki httpd 2.0
2404/tcp open  unknown

No OS matches for host

TRACEROUTE (using port 199/tcp)
HOP RTT       ADDRESS
1   1.00 ms   2a01:1e8:e115:3::2
2   112.00 ms 2a01:1e8:e115::200:14:2679
```

Es wurde ein zweiter Netzwerk-Scan bei einer bestehenden IEC104-Verbindung durchgeführt. Die Verbindung verlief stabil, solange nur Test Frames ausgetauscht wurden. Aber selbst da konnten mehrere TCP-Retransmissions beobachtet werden. Eine manuelle Generalabfrage konnte nicht vollständig beantwortet werden. Das Spectrum Power System hat in dem Fall die Verbindung abgebaut und neu aufgebaut. Daraus folgt, dass massive Netzwerkanfragen eine bestehende IEC104-Verbindung instabil macht. Der Contiki-TCP/IP-Stack ist so beschäftigt damit, die Anfragen zu verarbeiten, dass IEC104-Anfragen nicht in der vorgegebenen Zeit beantwortet werden können.
Der Contiki-TCP/IP-Stack lässt maximal zwei gleichzeitige TCP/IP-Verbindungen zu. Weitere Anfrage werden nicht beantwortet. Bei einer aktiven IEC104-Verbindung kann maximal eine HTTP-Anfrage beantwortet werden, ansonsten können bis zu zwei HTTP-Anfragen gleichzeitig verarbeitet werden. Dadurch, dass eine HTTP-Anfrage in der Regel nach weniger als eine Sekunde beantwortet ist, werden weitere Anfragen nach einem TCP Retransmission verarbeitet. Eine Vielzahl von gleichzeitigen Anfragen kann aber dazu führen, dass Anfragen nicht beantwortet werden.

Nur die schnellere Anfrage wird hier beantwortet. Nach so einer massiven Attacke von Anfragen lief die Steckdose aber normal weiter. Die Attacke führte nicht zu einem Reboot oder einem Fehler in den Applikationen. Mittlerweile läuft eine Steckdose seit über 190 Tagen ohne Unterbrechung.

Kapitel 8

Nachbetrachtung

In diesem Kapitel wird die erstellte Lösung begutachtet. Es werden Anwendungsmöglichkeiten, allgemein von Smart Objects und speziell für die Implementierung, behandelt. Mögliche Erweiterungen werden vorgeschlagen und ein Fazit der Arbeit wird gezogen.

8.1 Analyse der Implementierung

Die erstellte Lösung wird kritisch begutachtet. Dafür werden zuerst Kostenbetrachtungen bezogen auf die Hardware und auf die Software durchgeführt. Die Ergebnisse fließen in die abschließende Bewertung mit ein. Mit den gemachten Erfahrungen wird die Lösung anhand der in Kapitel 3.3 aufgestellten Kriterien neu bewertet. Der ermittelte Nutzwert wird mit den anfänglichen Komplettlösungen verglichen. Die Erfüllung der Anforderungen nach Kano aus Kapitel 3.1 wird geprüft.

8.1.1 Kostenbetrachtung

Um in der Bewertung der Implementierung die Kosten einschätzen zu können, sollen hier zum einem die Hardwarekosten vorgestellt werden. Dazu sollen die einzelnen Materialkosten der verwendeten Bauteile zusammengefasst werden. Um die gesamten Hardwarekosten ermitteln zu können, müssen auch Entwicklungskosten und Fertigungskosten berücksichtigt werden. Fertigungskosten beinhalten dabei die eigentlichen Arbeitsstunden, die für die Fertigung aufgebracht werden müssen, als auch die Investitionskosten für den Fertigungsarbeitsplatz. Weil das sehr abhängig von der Art der Fertigung ist, sei es eine Einzelstückfertigung oder eine Serienfertigung, werden diese Kosten hier nicht näher betrachtet. Die Hardwarekosten werden rein durch die Materialkosten repräsentiert.
Zum anderen sollen die Softwarekosten abgeschätzt werden. Dies wird durch die Ermittlung der Entwicklungskosten für die Contiki-Applikation iec104 erreicht. Anhand eines angenommenen Stundensatzes für einen Softwareentwickler und den aufgewendeten Stunden werden die Kosten für die Entwicklung berechnet.

8.1.1.1 Materialkosten

In Tabelle 8.1 sind die Bauteile aufgelistet, die zum Erstellen der Steckdose verwendet wurden. Sie benennt das Bauteil und zeigt die Anzahl an, die verbaut wurden. Die Bauteile wurden – bis auf die Tchibo Steckdose – vom Handelsunternehmen Mouser Electronics bezogen. In der dritten Spalte ist die entsprechende Artikelnummer aufgelistet. Die vierte Spalte zeigt den Preis bei dem Bezug von nur einem Bauteil. Bei einer Massenfertigung der Steckdose werden andere Preise angeboten. Der Stückpreis bei einer Bestellung von 1000 Fertigungssätzen ist in der letzten Spalte aufgelistet. Die Preise wurden am 06.05.2013 abgefragt und sind inklusive Mehrwertsteuer.

#	Bauteil	Artikelnummer	Einzelstückpreis in €	Massenstückpreis in €
Zigbit-Platine				
1	ATZB-24-A2	556-ATZB-24-A2	25,43	14,42
1	Widerstand 100kOhm	71-CMF60100K00FKEB	0,206	0,099
1	Widerstand 1kOhm	71-CMF551K0000FHEK	0,116	0,005
1	Kondensator 3,3uF	667-EEU-HD1H3R3	0,165	0,104
1	Spannungsregler LP2950CZ-3.0	926-2950CZ-3.0/NOPB	0,676	0,27
1	Steckverbinder FFC 6 Pin	538-52271-0679	1,30	0,583
2	Steckverbinder FFC 18 Pin	538-52271-1879	1,71	0,94
Steckdosen-Platine				
1	Kondensator X2 275V	80-R46KN368050M2M	0,66	0,263
1	Widerstand 560Ohm, 5Watt	594-AC05W560R0J	0,38	0,198
2	Widerstand 1MOhm	594-MRS251M1%TR	0,074	0,038
2	Gleichrichterdiode	512-1N4004	0,076	0,025
2	Zener-Diode 24V	512-1N4749ATR	0,203	0,036
1	Kondensator 470uF	667-EEU-FR1E471YB	0,248	0,152
1	NPN-Transistor BC547B	512-BC547B	0,186	0,05
1	Widerstand 220kOhm	271-220K-RC	0,125	0,012
Steckdosen-Gehäuse				
1	Tchibo Digitale Zeitschaltuhr	4 043002 669758	5,99	5,99
Gesamtpreis			**38,61**	**24,27**

Tabelle 8.1: Materialkosten Steckdose

Die Gesamtmaterialkosten der verwendeten Bauteile betragen 38,61€. Nicht berücksichtigt sind Verbrauchsmaterialien. Dazu zählen die Leiterplatte für die Zigbit-Platine, die aus einer vorhandenen Lochrasterplatine ausgesägt wurde, Leitungen für die Verkabelung und Lötzinn. Diese Kosten sollen hier vernachlässigt werden. Bei einer Kalkulation für eine Fertigung in einem Unternehmen müssen diese als Materialgemeinkostenzuschlag zu den Materialkosten hinzugefügt werden. Zusammen mit den Fertigungskosten, also die Arbeitskosten, die zum Fertigen der Steckdose notwendig sind, und den Verwaltungs- und Vertriebsgemeinkostenzuschlag ergeben sich die Selbstkosten je Stück.

8.1.1.2 Entwicklungskosten iec104

Um den Aufwand von einer Contiki-Applikation abzuschätzen, sollen beispielhaft die Entwicklungskosten der implementierten Applikation iec104 ermittelt werden. Zum Ermitteln der Kosten, muss der Stundensatz des Entwicklers mit der Zeit in Stunden multipliziert werden, die dafür notwendig ist, eine solche Applikation zu programmieren. Dazu gehört eine Designphase, die eigentliche Programmierung und eventuelle Fehlerkorrekturen. Weil die Kosten proportional zur Zeit sind, soll nur diese betrachtet werden. Es wird geschätzt, dass für die Entwicklung der Applikation iec104 etwa drei bis vier Mannwochen benötigt wurden. Die Schätzung wird dadurch erschwert, dass nicht kontinuierlich an der Entwicklung gearbeitet wurde. Die Arbeit daran wurde öfter durch andere notwendige Arbeiten unterbrochen.
Deswegen soll noch eine quantitative Bewertung anhand der Anzahl geschriebener Zeilen Quellcode LOC (Lines Of Code) stattfinden. In der Praxis ist solch eine Bewertung, also der Rückschluss von den LOC auf den Aufwand, allerdings problematisch. Die Zeit, die für die Designphase verwendet wird, wird nicht berücksichtigt. Eine überlegte und optimierte Programmierung benötigt tendenziell weniger LOC. Die Qualität der Programmierung wird auch nicht bewertet. Verschiedene Programmiersprachen und individuelle Programmierstile wirken sich ebenso auf die LOC aus.

Trotzdem gibt es Faustformeln, die einen Zusammenhang zwischen der Anzahl geschriebener Zeilen Quellcode LOC und der eingesetzten Entwicklungszeit sehen. Nach [Ludewig und Lichter 2010, Seite 305] wird in einem Softwareprojekt nach n Stunden Aufwand – wobei hier der Gesamtaufwand gemeint ist, nicht die reine Programmierungszeit – ein System mit zwei mal n LOC erzeugt. Ein studentisches Projekt, das nicht kommerziell vertrieben werden soll, kann in der gleichen Zeit in etwa die dreifache Anzahl an LOC liefern. Dort können demnach mit jeder Stunde Aufwand in etwa sechs LOC entwickelt werden.
Um nach dieser Faustformel herauszufinden, wie viele Stunden in die Entwicklung der Contiki-Applikiation iec104 eingeflossen sind, werden die Anzahl an Zeilen Quellcode ermittelt. Die Contiki-Applikation iec104 besteht aus 14 Dateien:

- 1 Makefile
- 7 Header-Dateien
- 6 C-Dateien

Insgesamt enthalten diese Dateien 1011 Zeilen. Wenn die 211 Leerzeilen und 112 Kommentarzeilen abgezogen werden, bleiben 688 Zeilen Quellcode. Nach der Faustformel oben ergibt sich eine Entwicklungszeit von etwa 115 Stunden. Bei der Annahme von einem Stundensatz von 100€ ergeben sich Entwicklungskosten von 11 500€. Bei einer Wochenarbeitszeit von 40 Stunden, ergeben sich ungefähr 3 Mannwochen, die notwendig sind, um die Contiki-Applikation iec104 zu entwickeln. Das deckt sich in etwa mit den Erfahrungen aus dieser Arbeit.

8.1.2 Bewertung der Implementierung

Die erstellte Implementierung wird anhand der gewichteten Kriterien aus Abbildung 3.4 bewertet und das Ergebnis mit dem der vorgestellten möglichen Komplettlösungen (Abbildung 3.5) verglichen.

- **Kommunikationstechnologie**

  1. **Reichweite der Funkübertragung**
     Die implementierte Steckdose lässt sich ansteuern, wenn sie innerhalb 10 Metern vom Empfangsgerät (den RZUSBSTICK) entfernt ist. Es werden 3 Punkte vergeben.
  2. **typische Leistungsaufnahme zum Senden und Empfangen**
     Die implementierte Steckdose verwendet mit dem AT86RF230 die gleiche Kommunikationseinheit, die in der Bewertung vorher beispielhaft für die IEEE 802.15.4 Technologie herangezogen wurde. Deswegen werden hier ebenfalls 5 Punkte vergeben. Die gemessene Stromaufnahme der gesamten Zigbit-Platine beträgt 23,8 mA, was ca. 8mA mehr sind als im Datenblatt des AT86RF230 [AT86RF230] angegeben sind. Der Mehrverbrauch ergibt sich aus der Stromaufnahme des Mikrocontrollers und der Verschaltung auf der Platine. Bei einer Versorgungsspannung von 3V beträgt die Leistungsaufnahme der gesamten Steuerung 71,4 mW.
  3. **Verbreitung der Technologie (Interoperabilität)**
     Es wird die Punktzahl für 6LoWPAN in der ersten Bewertung vergeben, 3 Punkte.
  4. **Integrationsaufwand in eine existierende Infrastruktur**
     Die implementierte Steckdose wurde über einen 6LoWPAN-Router in ein bestehendes Netzwerk und ins Internet integriert. Der 6LoWPAN-Router war ein handelsüblicher PC mit Linux Betriebssystem, einer Ethernet-Schnittstelle und einem RZUSBSTICK von Atmel als 6LoWPAN-Schnittstelle. Die Installation und Konfiguration ist bei Linux-Grundkenntnissen relativ einfach. Der RZUSBSTICK muss mit einer neuen Firmware geladen werden, was Mikrocontrollerkenntnisse voraussetzt. Als Laie ist es nicht

so einfach möglich, die implementierte Steckdose in eine existierende Infrastruktur zu integrieren. Es werden 4 Punkte vergeben. Wenn sich 6LoWPAN weiter verbreitet, ist damit zu rechnen, dass der Integrationsaufwand ähnlich wie bei WLAN mit der Zeit abnimmt.

5. **Implementierungsaufwand**
   Ein Implementierungsaufwand für die Kommunikationstechnologie war praktisch nicht vorhanden. Alles war bereits im Betriebssystem Contiki vorhanden. In der Datei Makefile.both wurde mit dem Parameter UIP_CONF_IPV6 der uIPv6-Stack eingeschaltet. Es musste der verwendete uIPv6-Stackpuffer UIP_CONF_BUFFER_SIZE auf 1280 Bytes vergrößert werden. Bei der verwendeten Plattform avr-zigbit beträgt dieser Puffer 240 Bytes. Die MAC-Adresse wurde jeweils im EEPROM konfiguriert. Ansonsten wurde nichts an der Kommunikationstechnologie geändert. Es werden 6 Punkte vergeben.
6. **Unterstützung von Applikationen**
   Prinzipiell wird jede Applikation unterstützt, die auf TCP/IP aufbaut. Es werden, genau wie bei der Bewertung vorher, 6 Punkte vergeben.
7. **typische Hardwarekosten**
   Die Hardwarekosten für die gesamte Steckdose betragen 38,61€ zuzüglich Verbrauchsmaterial. Die Kommunikationseinheit AT86RF230 befindet sich zusammen mit der Antenne und dem Mikrocontroller auf dem Zigbit-Modul. Einzeln ist sie im freien Markt für ca. 5€ erhältlich. Es werden wie bei der Bewertung vorher 4 Punkte vergeben.

- **Applikation**

8. **Verbreitung möglicher Zugriffssoftware**
   Mit dem Webserver bietet die implementierte Lösung die wohl am weitesten verbreiteste Zugriffsmöglichkeit. Nahezu jedes internetfähige Gerät hat einen Webbrowser. Bei der Bewertung vorher wurden 6 Punkte vergeben. Zugriffssoftware für die Applikation iec104 ist wenig verbreitet. In der Regel wird dafür ein SCADA System oder ähnliches benötigt. Eine Verbreitung von Zugriffssoftware mit IEC104 wird mit 1 Punkt bewertet. Weil aber über beide Applikationen unabhängig voneinander und parallel auf die Steckdose zugegriffen werden kann, werden insgesamt 6 Punkte vergeben.
9. **Möglichkeit für einen manuellen Zugriff (Benutzerschnittstelle)**
   Für einen manuellen Zugriff ist wie bei dem Punkt vorher die Applikation Webserver entscheidend. Es werden wie bei der Bewertung vorher 6 Punkte vergeben.
10. **Möglichkeit für einen automatischen Zugriff (M2M)**
    Für einem automatischen Zugriff wurde der Webserver vorher mit 4 Punkten bewertet. Über IEC104 ist ein automatisierter Zugriff aber durchaus üblich. So verfügt das Netzleitsystem Spectrum Power von Siemens und auch andere Netzleitsysteme zum Beispiel über eine Lastabwurffunktion. Im Falle von einer Instabilität im elektrischen Netz können automatisch Teile des Netzes, die eher als unwichtig eingestuft sind, ausgeschaltet werden. Durch die gezielte Abschaltung einiger weniger Verbraucher, kann die Stabilität des gesamten Netzes gerettet werden. In so einem Fall kann über das Protokoll IEC104 der Befehl gesendet werden, einen bestimmten Leistungsschalter zu öffnen. Hier werden 6 Punkte vergeben.
11. **Implementierungsaufwand der Applikation auf dem gewählten Betriebssystem**
    Die Webserver-Applikation war bereits im Contiki-Quellcode vorhanden. Sie musste nur gezielt angepasst werden. Die Applikation iec104 wurde neu implementiert. Die Entwicklungszeit betrug etwa drei Mannwochen. Es werden 3 Punkte vergeben.

12. **Anforderung an Kommunikationstechnologie**
    Das mit Abstand größte IEC104-Paket wird bei der Antwort auf die Unterstationsabfrage gesendet, wenn acht Einzelbitmeldungen in einem Telegramm übertragen werden. Hier beträgt die IPv6-Payload 64 Bytes. Das größte HTTP-Paket bei Übertragung der Webseite pins.shtml hat eine IPv6 Payload von 329 Bytes. Bei der vorherigen Bewertung wurden für HTTP 4 Punkte vergeben. IEC104 kommt mit einer viel geringeren Paketgröße aus, deswegen werden hier 5 Punkte vergeben.

- **Betriebssystem**

13. **Lizenz**
    Es wird die Punktzahl von der vorherigen Bewertung für das Betriebssystem Contiki vergeben – 6 Punkte.
14. **Verbreitung des Betriebssystems**
    Die Verbreitung des Betriebssystems Contiki wird wie vorher mit 4 Punkten bewertet.
15. **bereits unterstützte Applikationen**
    Wie bei der vorherigen Bewertung werden 6 Punkte vergeben.
16. **Aufwand der Implementierung neuer Applikationen**
    Nach den gemachten Erfahrungen von der Entwicklung der Applikation iec104 kann gesagt werden, dass der Implementierungsaufwand relativ gering ist. Allerdings wird eine gewisse Einarbeitungszeit in das Betriebssystem Contiki vorausgesetzt. Es werden 4 Punkte vergeben.
17. **Unterstützung verschiedener Kommunikationstechnologien**
    Es werden, wie bei der vorherigen Bewertung, 5 Punkte vergeben.

		6LoWPAN HTTP Contiki		6LoWPAN SNMP Contiki		WLAN HTTP Contiki		6LoWPAN HTTP TinyOS		Zigbee HTTP BitCloud		6LoWPAN HTTP/IEC104 Contiki	
	Gewicht %												
1	3,7	3	1,84	3	1,84	5	3,06	3	1,84	3	1,84	3	1,84
2	8,1	5	6,74	5	6,74	2	2,70	5	6,74	5	6,74	5	6,74
3	6,6	3	3,31	3	3,31	6	6,62	3	3,31	4	4,41	3	3,31
4	6,3	4	4,17	4	4,17	6	6,25	4	4,17	1	1,04	4	4,17
5	5,5	6	5,51	6	5,51	3	2,76	6	5,51	6	5,51	6	5,51
6	5,1	6	5,15	6	5,15	6	5,15	6	5,15	2	1,72	6	5,15
7	6,3	4	4,17	4	4,17	2	2,08	4	4,17	4	4,17	4	4,17
8	8,8	6	8,82	4	5,88	6	8,82	6	8,82	6	8,82	6	8,82
9	6,6	6	6,62	5	5,51	6	6,62	6	6,62	6	6,62	6	6,62
10	5,1	4	3,43	6	5,15	4	3,43	4	3,43	4	3,43	6	5,15
11	6,3	5	5,21	4	4,17	5	5,21	4	4,17	3	3,13	3	3,13
12	5,1	4	3,43	5	4,29	4	3,43	4	3,43	6	5,15	5	4,29
13	3,7	6	3,68	6	3,68	6	3,68	6	3,68	5	3,06	6	3,68
14	5,9	4	3,92	4	3,92	4	3,92	4	3,92	4	3,92	4	3,92
15	5,5	6	5,51	6	5,51	6	5,51	5	4,60	2	1,84	6	5,51
16	5,5	4	3,68	4	3,68	4	3,68	4	3,68	4	3,68	4	3,68
17	5,9	5	4,90	5	4,90	5	4,90	5	4,90	2	1,96	5	4,90
	100,0		**80,09**		**77,57**		**77,82**		**78,13**		**67,03**		**80,58**

Abbildung 8.1: Nutzwertanalyse der implementierten Lösung

Abbildung 8.1 zeigt, dass die implementierte Lösung mit 80,58% ungefähr den Nutzwert der gewählten Option erzielt. Sie liegt sogar minimal höher. Die meisten Kriterien sind gleich bewertet.

Durch die parallele Applikation iec104 steigt der Implementierungsaufwand, was einen negativen Einfluß auf den Nutzwert hat. Aber die Möglichkeit für einen automatischen Zugriff (M2M) wird verbessert und die Anforderungen an die Kommunikation sind geringer bei Verwendung vom Protokoll IEC104.

8.1.3 Erfüllung der Anforderungen nach Kano

In Kapitel 3.1 wurden Anforderungen nach dem Kano-Modell definiert und in Basis-, Leistungs- und Begeisterungsanforderungen eingeteilt. In diesem Kapitel wird gezeigt, welche Anforderungen von der implementierten Lösung erfüllt werden und welche nicht. Der Erfüllungsgrad der einzelnen Anforderungsmerkmale beschreibt die Kundenzufriedenheit.
In Kapitel 3.1.1 wurden die Basisanforderungen definiert. Sie werden alle von der implementierten Lösung erfüllt. Der Zugriff auf die Steuerung erfolgt drahtlos mittels IPv6 über das Internet oder über ein lokales Netzwerk unter Verwendung einer Benutzerschnittstelle. Ein Zugriff über IPv4 ist nicht realisiert, war aber auch nicht gefordert. Die benötigte Leistung wird vom Netzanschluss der Steckdose über ein Kondensatornetzteil bezogen, es ist kein zusätzlicher Anschluss notwendig. Die mögliche Reichweite der Funkübertragung liegt bei etwa 10 Metern. Die Leistungsaufnahme der Steuerung beträgt bei 3V Versorgungsspannung 71,4 mW. Die Steuerung ist ortsungebunden, die MAC-Adresse ist fest programmiert, aber die IP-Adresse wird davon abgeleitet vom 6LoWPAN-Router vergeben.
Von den acht Leistungsanforderungen, die in Kapitel 3.1.2 definiert worden sind, werden sieben erfüllt. Die Anforderungen an die Reichweite und die Leistungsaufnahme werden knapp erfüllt. Mit HTTP und IEC104 findet der Zugriff über Standard-Applikationsprotokolle statt. Die Bedienung ist einfach und intuitiv. Ein automatischer Zugriff (M2M) ist über IEC104 möglich. Die Materialkosten für die einzelnen Bauteile betragen 38,61€. Die Contiki-Applikation iec104 besteht aus 688 Zeilen Quellcode (LOC). Die Anpassungen für die Applikation webserver-nano-steckdose bestehen aus 54 LOC. Die Beschreibung des Contiki-Projektes besteht aus 32 LOC. Insgesamt wurden weniger als 1000 LOC für die Implementierung selbst programmiert. Die Einbindung in ein bestehendes Heim-Netzwerk vollzieht sich noch nicht einfach. Einem Laien ohne Linux- und Mikrocontrollerkenntnisse wird es nicht ohne weiteres möglich sein, die implementierte Lösung in ein bestehendes Heim-Netzwerk zu integrieren.
Zwei von den drei Begeisterungsanforderungen aus Kapitel 3.1.3 werden erfüllt. Weitere Zugriffsmöglichkeiten, wie telnet oder SNMP, stehen zur Verfügung. Eine Anpassung auf den speziellen Anwendungsfall wird notwendig sein, aber nicht aufwendig. Bei der implementierten Lösung stehen zwei Zugriffsmöglichkeiten parallel zur Verfügung. Die Materialkosten liegen aber über den geforderten 10€.
Die Basisanforderungen werden komplett erfüllt, bei den Leistungs- und Begeisterungsanforderungen wird jeweils eine Anforderung nicht erfüllt. An diesen Anforderungen könnte angesetzt werden, um die Lösung weiter zu verbessern. Die Einbindung der Steuerung in ein bestehndes Heim-Netzwerk könnte durch einen nahezu fertigen und einfach zu konfigurierbaren 6LoWPAN-Router gegeben sein. Noch ist so etwas für den Normalverbraucher nicht verfügbar. Die Begeisterungsanforderung, dass die Materialkosten weniger als 10€ betragen sollen, ist in dieser Konstellation nicht zu erreichen. Der Preis für das verwendete Gehäuse beträgt 5,99€. Das Zigbit-Modul kostet auch bei einer Massenfertigung von 1000 Stück 14,42€. Materialgesamtkosten von unter 10€ sind mit diesem Ansatz nicht zu erreichen.

8.2 Anwendungsmöglichkeiten

Dieses Kapitel stellt zuerst allgemein verschiedene Anwendungsbereiche von Smart Objects vor. Darauffolgend wird erörtert, in welchen Anwendungsbereichen die erstellte Lösung eingesetzt

werden könnte. Anhand der Erfahrung, die bei der Implementierung gemacht wurden, werden mögliche Erweiterungen behandelt.

8.2.1 Anwendungsbereiche von Smart Objects

Smart Objects können in einer Vielzahl von Anwendungsbereichen eingesetzt werden, da sie sehr flexibel in ihrer Beschaffenheit sind. Die Programmierung kann individuell angepasst werden. Durch die Verwendung der richtigen Sensoren und Aktoren sind sie für verschiedenste Einsatzbereiche verwendbar. Nachfolgend sollen folgende fünf mögliche Anwendungsbereiche kurz beschrieben werden:

- eHome-Bereich
- Gebäudeautomation
- Industrieautomatisierung
- Logistik
- Smart Grid

Der eHome-Bereich ist ein Bereich in dem sich mehrere proprietäre Lösungen verbreitet haben, die oft nicht untereinander kompatibel sind. Dies kann auch ein Mitgrund dafür sein, dass sich die Heimautomatisierung bisher nicht so entwickelt hat wie vor etwa 7-9 Jahren prognostiziert [Vasseur und Dunkels 2010, Seite 360]. Wichtig ist auch, gerade im eHome-Bereich, eine einfache Installation der Geräte. Nur wenn ein Endnutzer ohne spezielles Expertenwissen Geräte in Betrieb nehmen und verwenden kann, wird sich eine Lösung durchsetzen können. Mögliche Einsatzgebiete von Smart Objects im eHome-Bereich sind zum Beispiel Steuerungen von Licht, der Heizung, von Fenster, von Rollläden und von Türschlössern.
Der eHome-Bereich wird für private Wohnhäuser eingesetzt. Im Gegensatz dazu zielt die Gebäudeautomation eher auf den professionellen Einsatz in großen Gebäuden meist im Zusammenspiel mit einem visualisierten Gebäudemanagementsystem. Vermesan und Friess [Vermesan und Friess 2011, Seite 304ff] beschreiben unter dem Titel „Smart IPv6 Building“ verschiedene Forschungsprojekte, die die Verwendung von IPv6 in der Gebäudeautomation untersuchen. Unter anderem wird auch das Hobnet-Projekt erwähnt, das spezielle Anwendungsfälle von Smart Objects in der Gebäudeautomation untersucht. Hier kommt auch 6LoWPAN zum Einsatz. Allgemein sind die Ziele einer Gebäudeautomation Energieeinsparungen, Sicherheit und Komfortgewinn.
In der Industrieautomatisierung unterstützen Smart Objects den Verarbeitungsprozess, indem sie Messgrößen erfassen und kleinere Steuerungsaufgaben übernehmen. Traditionell hat die Industrieautomatisierung auf verkabelte Kommunikationstechnologien gesetzt. Durch den Einsatz von drahtlosen Smart Objects sinkt aber der Aufwand zusätzliche Messgrößen zu erfassen. Vor allem in Umgebungen, in denen der Einsatz von Verkabelung beschwerlich oder unmöglich ist, werden mehr und mehr drahtlose Kommunikationstechnologien eingesetzt. Die großen Herausforderungen dabei sind die hohe geforderte Verfügbarkeit, die mögliche Überlagerung von Frequenzbändern, eine hohe Lebenszeit mit möglichst wenig Wartung, die Sicherheit und die Interoperabilität [Vasseur und Dunkels 2010, Seite 333].
In der Logistik können Smart Objects beim Tracking einzelner Waren eingesetzt werden. Jede Ware hat einen Sensor, der auf Anfrage an ein zentrales System seinen Aufenthaltsort und seinen Status nennen kann. So können innerhalb einer Lieferkette Warenströme besser synchronisiert werden. Das Smart-Objects-Innovation-Lab [Web:SOIL] ist eine Forschungseinrichtung, die verschiedene Anwendungsszenarien für Smart Objects in diesem Bereich prüft.
Der Bereich Smart Grid wird einer der größten Anwendungsbereiche für Smart Objects werden [Hersent et al. 2012, Seite 271ff]. Seit mehreren Jahren geht der Trend in der Stromerzeugung immer mehr in Richtung erneuerbaren Energiequellen wie Windkraftanlagen oder Photovoltaikanlagen. Diese sind aber im Gegensatz zu klassischen fossilen Kraftwerken dezentral aufgestellt.

Das führt zu einem erheblichen Umbruch in der Stromnetzführung. Die Energie wird nicht allein an wenigen zentralen Örtlichkeiten durch große Kraftwerke, sondern auch dezentral durch kleine teils privat teils kommerziell geführten Energiequellen erzeugt. Das erschwert die Kraftwerksregelung und die Leitung des Lastflusses. Um trotzdem eine gute Netzstabilität zu gewährleisten, muss das Stromnetz intelligenter und vielfältiger überwacht und gesteuert werden.
Eine Maßnahme dabei ist das Smart Metering. Stromverbrauchszähler liefern über eine Kommunikationsschnittstelle den aktuellen Energieverbrauch an das Energieversorgungsunternehmen. Solche Art intelligente Zähler existieren schon länger für große Energieverbraucher wie Produktionsbetriebe, seit einigen Jahren werden Smart Meter auch für Privathaushalte angeboten.
Eine andere Maßnahme ist die Elektromobilität. Hinter dem Begriff verbirgt sich die Nutzung von Elektrofahrzeugen. In der Regel besitzen diese große Batterien, die über das Stromnetz geladen werden. Aus Sicht von Energieversorgungsunternehmen ist ein Elektrofahrzeug ein mobiler Energiespeicher. Er kann an verschiedenen Stellen aufgeladen werden und damit Energie aus dem Stromnetz abnehmen, aber er kann theoretisch auch Energie ins Stromnetz zurückgeben und damit als Energieerzeuger wirken. Im Bereich der Elektromobilität werden derzeit viele Modellprojekte an verschiedenen Standorten erprobt.
Smart Objects können im Bereich Smart Grid helfen, Daten zu sammeln und kleine Steuerungsaufgaben zu übernehmen. Sie können sowohl in der Netzführung, im Bereich Smart Metering oder im Bereich der Elektromobilität eingesetzt werden.

8.2.2 Anwendungsmöglichkeiten für die Implementierung

Die klassische Anwendung der implementierten Lösung ist der eHome-Bereich. Im privaten Haushalt kann ein elektrischer Verbraucher über das Internet ein- oder ausgeschaltet werden. Hier wird hauptsächlich die Webserver-Applikation verwendet werden. Man kann von einem internetfähigen Gerät weltweit auf die Steuerung zugreifen.
Das Protokoll IEC104 ist ein Fernwirkprotokoll aus dem Bereich der Energieautomatisierung. Der Bereich Smart Grid ist also ein klassischer Anwendungsfall für dieses Protokoll. An ein zentrales Leitsystem werden Informationen übertragen und Steuerbefehle entgegen genommen. Allerdings wird der Zugriff auf eine Steuerung von einem elektrischen Verbraucher im Privathaushalt für ein Energieversorgungsunternehmen nicht von Bedeutung sein. Ein möglicher Anwendungsfall liegt eher in einem Smart Meter. Hier könnten mittels IEC104 Verbrauchsstände übertragen werden. Zusätzlich wäre in Absprache mit dem Privathaushalt eine Steuerung möglich. Genaue Anwendungsfälle müssen noch erprobt werden. Denkbar wäre eine Notabschaltung einer lokalen Photovoltaikanlage, wenn mehr Energie ins Stromnetz eingespeist wird als dort verbraucht wird bzw. in andere Stromnetzregionen abgeführt werden kann. Ebenfalls denkbar ist eine gezielte Steuerung von größeren Energieverbrauchern, bei denen der Einsatz zeitlich flexibel ist (Demand Side Management). Beispiele dafür sind die Waschmaschine oder die Batterie eines Elektrofahrzeugs. Bei einer großer Netzauslastung können diese steuerbaren Verbraucher zurückgefahren werden. Das Energieversorgungsunternehmen muss dann weniger Reserven für Spitzenlast vorhalten. So hilft eine intelligente Netzführung dabei, Kosten in der Energieerzeugung zu reduzieren.

8.2.3 Erweiterungsvorschläge für die Implementierung

Die erstellte Lösung ist ein Prototyp, der beispielhaft implementiert worden ist. Während der Arbeit sind mehrere Ideen entstanden, wie die Implementierung verbessert oder erweitert werden kann. Zuerst werden Vorschläge für die IEC104-Slave-Applikation aufgelistet.

- Informationsmeldungen mit Zeitstempel
  Die Statusänderung der Steckdose erfolgt aktuell über den Datentyp M_SP_NA_1 (Type Ident 1): Einzelbitmeldung ohne Zeitstempel. Eine Verbesserung wäre die Verwendung des Datentyps M_SP_TB_1 (Type Ident 30): Einzelbitmeldung mit dem Zeitstempel

CP56Time2a. Hier wird zusätzlich ein Zeitstempel mit Jahr, Monat, Tag, Stunde, Minute, Sekunde und Millisekunde übertragen. So ist der exakte Zeitpunkt der Statusänderung bekannt. Voraussetzung dafür ist aber, dass die Steuerung mit der aktuellen Zeit synchronisiert ist. Über IEC104 existiert dafür eine Möglichkeit mit dem Datentyp C_CS_NA_1 (Type Ident 103): Zeitsynchronisationsbefehl. Ein Zeitstempel wird vom IEC104-Master zum IEC104-Slave gesendet und dort übernommen. Weil die Übertragungszeit dabei aber nicht berücksichtigt wird, ist die Verwendung vom IEC-Standard nur bedingt empfohlen. In der Praxis wird oft das Protokoll NTP (Network Time Protocol) verwendet. Eine Implementierung von NTP im Betriebssystem Contiki existiert bereits [Web:Contiki-syslog].

- interne Statusinformationen des Contiki Betriebssystems
  Das Modul iec104-para kann mit zusätzlichen Informationsobjekten erweitert werden. Zusätzliche interne Statusinformationen wie die Anzahl der laufenden Prozesse, TCP/IP-Verbindungen oder die Betriebszeit könnten übertragen werden. Für Messwerte gibt es bereits den Datentyp M_ME_NA_1 (Type Ident 9): Messwert, normalisiert und ohne Zeitstempel. Weitere Datentypen können implementiert werden.
- IEC104-Parameter w=1
  In Kapitel 7.4.3 wurde festgestellt, dass Telegramme mit einem unbekannten ASDU-Typ auf Linkebene nicht quittiert werden. Das könnte zu einem Verbindungsabbruch führen. Durch Setzen das IEC104-Parameters w=1 wird das verhindert. Alternativ können die IEC104 Time-out Parameter ausprogrammiert werden.
- Befehlsrückmeldungen beschleunigen
  Vom Ausführen eines Befehls bis die Statusänderung zurückgesendet wird, können bis zu zwei Sekunden vergehen. Das ist der Zyklus, definiert über die Variable APPLICATION_TIMER, in dem geprüft wird, ob sich ein Wert in Monitorrichtung geändert hat. Eine Verbesserung ist es, wenn direkt nach der Befehlsausführung die Statusänderung gesendet wird. Dies kann erreicht werden, indem im Programmablauf direkt nach der Befehlsausführung die Werte in Monitorrichtung auf eine Änderung überprüft werden.

Eine weitere Verbesserungsmöglichkeit betrifft den Energieverbrauch der Implementierung. Es wurde das vorhandene Leistungsrelais HF7520 verwendet, das über eine Sollspannung von 24VDC angesteuert wird. Deswegen ist das Kondensatornetzteil auf 24VDC dimensioniert. Um das Zigbit-Modul zu schützen, wird die 24VDC Eingangsspannung über den Spannungsregler LP2950CZ-3.0 auf der Zigbit-Platine konstant auf 3VDC umgewandelt. Weil die Stromaufnahme jedoch konstant ist, entsteht dadurch Verlustleistung. Besser ist es, wenn das Relais mit der gleichen Spannung wie das Zigbit-Modul betrieben werden kann. In dem Fall kann auch auf den Spannungsregler verzichtet werden. Weil das Zigbit-Modul eine Betriebsspannung von 1,8V bis 3,6V zulässt kann durch eine Reduzierung der Spannung auf 1,8V zusätzlich noch Energie eingespart werden.
Die nächste Verbesserungsmöglichkeit betrifft nicht die Implementierung selbst sondern dem Testaufbau. Als 6LoWPAN-Router wurde ein handelsüblicher PC mit einer Linux-Installation verwendet. Als 6LoWPAN-Schnittstelle wurde ein USB-Stick RZUSBSTICK von Atmel verwendet, der auf dem PC eine Ethernet-Schnittstelle simuliert. Unter anderem hat das den Nachteil, dass auf dem PC der 6LoWPAN-Netzwerkverkehr nicht beobachtet werden kann, weil nur der simulierte Ethernet-Verkehr sichtbar ist. Es wird aber aktuell daran gearbeitet, 6LoWPAN direkt im Linux-Kernel zu implementieren [Ott 2012]. Eine eingeschränkte Variante wird schon seit der Kernel Version 3.2.46 unterstützt und laufend erweitert. Als Funkchips werden der MRF24J40 von Microchip und der AT86RF230 von Atmel, der auch im Zigbit-Modul verbaut ist, unterstützt. Ott stellt eine Hardware vor bestehend aus einem BeagleBone und einem MRF24J40MA Funkchip. Damit kann 6LoWPAN nativ unter Linux verwendet werden ohne zusätzliche Hardware und ohne ein zusätzliches Betriebssystem. Es ist allerdings nicht bekannt, ob es Interoperabilitätsprobleme zwischen der 6LoWPAN-Implementierung im Linux-Kernel und im Contiki-Betriebssystem gibt.

8.3 Fazit

Es konnte gezeigt werden, dass eine internetfähige Steuerung mithilfe eines 8-bit Mikrocontrollers implementiert werden kann. Das Betriebssystem Contiki stellt dazu den notwendigen TCP/IP-Stack und andere Werkzeuge und Hilfsmittel zur Verfügung. Eine Webserver-Anwendung und eine Vielzahl anderer Anwendungen sind Teil des Betriebssystems. Mit der Contiki-Appliktation iec104 wurde gezeigt, dass die Implementierung des Protokolls IEC104 in einer limitierten Umgebung durchaus möglich ist. Auch die Verwendung von IEC104 im Zusammenspiel mit IPv6 hat funktioniert. Bisher war keine Implementierung von IEC104 über IPv6 bekannt. Generell zeigt die Contiki-Applikation iec104, dass die Implementierung neuer internetfähiger Anwendungen möglich und nicht aufwendiger als bei anderen Betriebssystemen ist. Die geringen Ressourcen müssen allerdings bei der Programmierung berücksichtigt werden.
Bei der Implementierung ist der 8KByte große RAM-Speicher die markanteste Ressourcengrenze. Die Auslastung beträgt 88,9%. Das Hinzufügen von zusätzlichen Contiki-Applikationen oder das Erweitern der Applikation iec104 ist deswegen nicht ohne Weiteres möglich. Durch den Einsatz von anderer Hardware mit einem größeren RAM-Speicher kann dieses Problem umgangen werden. So steht bei dem AVR Ravenboard mit 16KByte doppelt so viel RAM-Speicher zur Verfügung. Der Hersteller Redwire bietet mit dem Econotag ein fertiges Modul mit USB-Schnittstelle an. Verwendet wird der Freescale MC13224V, der einen 32-bit ARM7 Mikrocontroller mit einer IEEE-802.15.4-Schnittstelle kombiniert. Er verfügt über 128KByte Flash- und 96KByte RAM-Speicher. Das Betriebssystem Contiki unterstützt diese Modul über die Platform redbee-econotag. Beide, das AVR Ravenboard und das Econotag, sind aber für den Einbau in die verwendete Steckdose zu groß.
Als Kommunikationstechnologie wurde 6LoWPAN verwendet. Es scheint ideal für den Einsatz bei ressourcenbeschränkten Geräten zu sein. Anfang dieses Jahres hat die ZigBee Alliance die Spezifikation ZigBee IP veröffentlicht [ZigBee-IP]. Damit werden unter der Verwendung von 6LoWPAN vermaschte drahtlose IPv6-Netzwerke unterstützt. Dies ist ein Beispiel dafür, dass die Verwendung von 6LoWPAN in vielen Bereichen voranschreitet. Eine weitere Entwicklung, die die Verbreitung von 6LoWPAN unterstützt, ist die Spezifizierung des Protokolls RPL [RFC6550]. RPL ist ein Routing-Protokoll, das für verlustbehaftete Sensornetze entwickelt worden ist. Die speziellen Anforderungen konnten von existierenden Routing-Protokolle wie OSPF oder RIP nicht erfüllt werden. Die Implementierung von RPL im Betriebssystem Contiki wird ContikiRPL genannt [Tsiftes et al. 2010].
Sehr wichtig für die weitere Verbreitung von 6LoWPAN ist die Interoperabilität von verschiedenen Implementierungen. Ko u. a. [Ko et al. 2011] haben zwei unabhängige Implementierungen, Contiki und TinyOS, zusammen getestet. Die Interoperabilität zwischen beiden war gegeben, allerdings hatten kleine Unterschiede im jeweiligen Protokollstack einen Einfluss auf die Gesamtsystemleistung. Neben der Interoperabilität ist die Leistungsfähigkeit bei dem Zusammenspiel verschiedener Implementierungen von Bedeutung.
Als weitere Schwierigkeit kommt hinzu, dass viele Implementierungen den Funkempfänger so oft wie möglich ausschalten. Im Betriebssystem Contiki steht diese Funktionalität unter dem Namen ContikiMAC zur Verfügung. So kann der Energieverbrauch um bis zu 80% reduziert werden [Dunkels 2011]. Das gesteuerte Ausschalten des Funkempfängers hat allerdings einen markanten Einfluss auf das Kommunikationsverhalten im Netzwerk. Weil keine Spezifikationen oder Standards für diese Funktionalität existieren, verhalten sich Implementierungen sehr verschieden. Dunkels, Eriksson und Tsiftes [Dunkels et al. 2011] beschreiben, dass das Zusammenspiel von Kommunikationstechnologie und Ausschaltverhalten des Funkempfängers ein wichtiges Forschungsgebiet für das Internet-of-Things ist.

Literatur

[AT86RF230] Atmel Corporation. *AT86RF230 - Low Power 2.4 GHz Transceiver for ZigBee, IEEE 802.15.4, 6LoWPAN, RF4CE and ISM Applications*. Englisch. URL: http://www.atmel.com/devices/AT86RF230.aspx (besucht am 29. 12. 2012).

[Atmel BitCloud] Atmel Corporation. *Atmel AVR2052: Atmel BitCloud Quick Start Guide*. Englisch. Application Note. URL: http://www.atmel.com/tools/BITCLOUD-ZIGBEEPRO.aspx?tab=documents (besucht am 23. 04. 2014).

[ATZB-24-A2] Atmel Corporation. *ZigBit™ 2.4 GHz Wireless Modules - ATZB-24-A2/B0*. Englisch. URL: http://www.atmel.com/tools/ZIGBIT2_4GHZMODULEWITHDUALCHIPANTENNA.aspx (besucht am 29. 02. 2012).

[AVR RZRaven] Atmel Corporation. *Atmel AVR2016: RZRAVEN Hardware User's Guide*. Englisch. URL: http://www.atmel.com/tools/RZRAVEN.aspx (besucht am 20. 02. 2012).

[BC547] Fairchild Semiconductor Corporation. *BC546/547/548/549/550*. Englisch. Datenblatt. URL: http://www.fairchildsemi.com/ds/BC/BC547.pdf (besucht am 01. 05. 2012).

[Casado und Tsigas 2009] Lander Casado und Philippas Tsigas. „ContikiSec: A Secure Network Layer for Wireless Sensor Networks under the Contiki Operating System". In: *NordSec*. 2009, S. 133–147. URL: http://www.cse.chalmers.se/research/group/dcs/masters/contikisec/ (besucht am 04. 04. 2011).

[CC2420] Texas Instruments Incorporated. *CC2420 - 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver*. Englisch. URL: http://www.ti.com/product/cc2420 (besucht am 29. 12. 2012).

[CC3000] Texas Instruments Incorporated. *CC3000MOD - TI SimpleLink™ CC3000 Module – Wi-Fi 802.11b/g Network Processor*. Englisch.

URL: http://www.ti.com/product/cc3000
(besucht am 12. 01. 2013).

[Dunkels 2003] Adam Dunkels. „Full TCP/IP for 8 Bit Architectures“.
In: *Proceedings of the First ACM/Usenix International Conference on Mobile Systems, Applications and Services (MobiSys 2003)*. usenix. San Francisco, Mai 2003. URL: http://www.sics.se/~adam/mobisys2003.pdf.

[Dunkels 2011] Adam Dunkels.
The ContikiMAC Radio Duty Cycling Protocol.
Techn. Ber. T2011:13.
Swedish Institute of Computer Science, Dez. 2011.
URL: http://www.sics.se/~adam/dunkels11contikimac.pdf.

[Dunkels et al. 2004] Adam Dunkels, Björn Grönvall und Thiemo Voigt.
„Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors“. In: *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*.
Tampa, Florida, USA, Nov. 2004. URL: http://www.sics.se/~adam/dunkels04contiki.pdf.

[Dunkels et al. 2006] Adam Dunkels u. a. „Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems“.
In: *Proceedings of the Fourth ACM Conference on Embedded Networked Sensor Systems (SenSys 2006)*.
Boulder, Colorado, USA, Nov. 2006. URL: http://www.sics.se/~adam/dunkels06protothreads.pdf.

[Dunkels et al. 2011] Adam Dunkels, Joakim Eriksson und Nicolas Tsiftes.
„Low-power Interoperability for the IPv6-based Internet of Things“. In: *Proceedings of the 10th Scandinavian Workshop on Wireless Ad-hoc Networks (ADHOC ´11)*.
Johannesberg Castle, Stockholm, Mai 2011. URL: http://www.sics.se/~adam/dunkels11adhoc.pdf.

[Dunkels und Schmidt 2005] Adam Dunkels und Oliver Schmidt.
Protothreads - Lightweight Stackless Threads in C.
Techn. Ber. T2005:05.
SICS – Swedish Institute of Computer Science, März 2005.
URL: http://www.sics.se/~adam/dunkels05protothreads.pdf.

[Dunkels und Vasseur 2008] Adam Dunkels und Jean-Philippe Vasseur.
IP for Smart Objects. IPSO Alliance White Paper #1.
Sep. 2008. URL: http://www.sics.se/~adam/dunkels08ipso.pdf.

[Durvy et al. 2008] Mathilde Durvy u. a. „Making Sensor Networks IPv6 Ready“.
In: *Proceedings of the Sixth ACM Conference on Networked Embedded Sensor Systems (ACM SenSys 2008), poster session*. Best poster award.
Raleigh, North Carolina, USA, Nov. 2008. URL: http://www.sics.se/~adam/durvy08making.pdf.

[Dutta und Dunkels 2012] Prabal Dutta und Adam Dunkels. „Operating Systems and Network Protocols for Wireless Sensor Networks“. In: *Philosophical Transactions of the Royal Society A* 370.1958 (13. Jan. 2012), S. 68–84.

[ENC28J60] Microchip. *ENC28J60 - Stand-Alone Ethernet Controller with SPI Interface*. Englisch. Datenblatt. URL: http://ww1.microchip.com/downloads/en/DeviceDoc/39662e.pdf (besucht am 07. 04. 2012).

[Finkenzeller 2008] Klaus Finkenzeller. *RFID Handbuch - Grundlagen und praktische Anwendungen von Transpondern, kontaktlosen Chipkarten und NFC*. Deutsch. 5. Aufl. Carl Hanser Verlag, 2008. ISBN: 978-3-446-41200-2.

[G.9959] International Telecommunication Union. *Short range narrow-band digital radiocommunication transceivers – PHY and MAC layer specifications*. Englisch. Feb. 2012.

[Hagen 2009] Silvia Hagen. *IPv6 : Grundlagen, Funktionalität, Integration*. Deutsch. 2. Aufl. Sunny Edition, 2009, S. I–XXIII, 1–507. ISBN: 978-3-9522942-2-2.

[Heath 2003] Steve Heath. *Embedded Systems Design*. Englisch. 2. Aufl. Elsevier Ltd., 2003, S. I–XX, 1–430. ISBN: 978-0750655460.

[Herold 1999] Helmut Herold. *Linux-Unix-Grundlagen, Kommandos und Konzepte*. Deutsch. 4. Aufl. Addison Wesley Verlag, 1999, S. I–VII, 1–1088. ISBN: 3-8273-1435-6.

[Hersent et al. 2012] Olivier Hersent, David Boswarthick und Omar Elloumi. *The Internet of Things: Key Applications and Protocols*. Englisch. 6. Aufl. John Wiley und Sons Ltd, 2012. ISBN: 978-1-1199943-5-0.

[HF7520] Hongfa. *HF7520 - Subminiature Power Relay*. Englisch. Datenblatt. URL: http://www.hongfa.com/pro/pdf/HF7520_en.pdf (besucht am 06. 03. 2012).

[Hill et al. 2000] Jason Hill u. a. „System architecture directions for networked sensors“. In: *Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*. ASPLOS IX. Cambridge, Massachusetts, USA: ACM, 2000, S. 93–104. ISBN: 1-58113-317-0. DOI: 10.1145/378993.379006. URL: http://doi.acm.org/10.1145/378993.379006.

[ID-6lowpan-btle] J. Nieminen, T. Savolainen, M. Isomaki, B. Patil, Z. Shelby, C. Gomez. *Transmission of IPv6 Packets over BLUETOOTH Low Energy*. Englisch. draft. Internet Engineering Task Force, Feb. 2013.

URL: http://tools.ietf.org/html/draft-ietf-6lowpan-btle-12.

[ID-6lowpan-ghc] C. Bormann. *6LoWPAN Generic Compression of Headers and Header-like Payloads*. Englisch. draft. Internet Engineering Task Force, März 2013. URL: http://tools.ietf.org/html/draft-bormann-6lowpan-ghc-06.

[ID-6lowpan-roadmap] C. Bormann. *6LoWPAN Roadmap and Implementation Guide*. Englisch. draft. Internet Engineering Task Force, Apr. 2013. URL: http://tools.ietf.org/html/draft-bormann-6lowpan-roadmap-04.

[ID-v6ops-ayiya] J. Massar. *AYIYA: Anything In Anything draft-massar-v6ops-ayiya-02*. Internet-Draft (Expired). Internet Engineering Task Force, Juli 2004. URL: http://www.sixxs.net/tools/ayiya/.

[IEC101] International Electrotechnical Commission. *IEC 60870-5-101, Second Edition*. Englisch. Feb. 2003.

[IEC104] International Electrotechnical Commission. *IEC 60870-5-104, First edition*. Englisch. Dez. 2000.

[IEC60751] International Electrotechnical Commission. *Industrial platinum resistance thermometers and platinum temperature sensors, Edition 2.0 2008-07*. Englisch. Juli 2008.

[IEC870-5-5] International Electrotechnical Commission. *IEC 870-5-5, Deutsche Fassung*. Deutsch. 1995.

[IEEE1901] IEEE. *IEEE Standard for Broadband over Power Line Networks: Medium Access Control and Physical Layer Specifications*. Institute of Electrical und Electronics Engineers, Inc. Dez. 2010.

[IEEE802.11] IEEE. *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. Institute of Electrical und Electronics Engineers, Inc. März 2012.

[IEEE802.15.4] IEEE. *Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*. Institute of Electrical und Electronics Engineers, Inc. Juni 2011.

[IEEE802.3] IEEE. *Part 3: Carrier sense multiple access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*. Institute of Electrical und Electronics Engineers, Inc. Dez. 2008.

[ISA-100.11a] International Society of Automation. *ANSI/ISA-100.11a-2011 Wireless systems for industrial automation: Process control and related applications*. Englisch. Mai 2011.

[Kano et al. 1984] KANO Noriaki u. a.
„Attractive Quality and Must-Be Quality“.
In: *Journal of the Japanese Society for Quality Control* 14.2 (1984-04-15), S. 147–156. ISSN: 03868230. URL: http://ci.nii.ac.jp/naid/110003158895/en/.

[Ko et al. 2011] JeongGil Ko u. a. „Beyond Interoperability: Pushing the Performance of Sensornet IP Stacks“.
In: *Proceedings of the ACM Conference on Networked Embedded Sensor Systems, ACM SenSys 2011*.
Seattle, WA, USA, Nov. 2011. URL: http://www.sics.se/~adam/ko11beyond.pdf.

[Kuryla und Schönwälder 2011] Siarhei Kuryla und Jürgen Schönwälder.
„Evaluation of the resource requirements of SNMP agents on constrained devices“. In: *Proceedings of the 5th international conference on Autonomous infrastructure, management, and security: managing the dynamics of networks and services*.
AIMS'11. Nancy, France: Springer-Verlag, 2011, S. 100–111.
ISBN: 978-3-642-21483-7.

[Levis 2006] Philip Alexander Levis. „TinyOS: An Open Operating System for Wireless Sensor Networks (Invited Seminar)“.
In: *Proceedings of the 7th International Conference on Mobile Data Management*. MDM '06.
Washington, DC, USA: IEEE Computer Society, 2006, S. 63–.
ISBN: 0-7695-2526-1. DOI: 10.1109/MDM.2006.151.
URL: http://dx.doi.org/10.1109/MDM.2006.151.

[LP2950] Texas Instruments Incorporated. *LP2950/LP2951 adjustable micropower voltage regulators with shutdown*. Englisch.
Datenblatt. URL: http://www.ti.com/lit/ds/symlink/lp2950-30.pdf (besucht am 01. 05. 2012).

[Ludewig und Lichter 2010] Jochen Ludewig und Horst Lichter. *Software Engineering - Grundlagen, Menschen, Prozesse, Techniken*. Deutsch.
2. Aufl. dpunkt.verlag, 2010, S. I–XXI, 1–665.
ISBN: 978-3-89864-662-8.

[MSP430] Texas Instruments Incorporated.
MSP430 Ultra-Low-Power Microcontroller. Englisch.
URL: http://www.ti.com/lsds/ti/microcontroller/16-bit_msp430/overview.page (besucht am 29. 12. 2012).

[Ott 2012] Alan Ott.
„Wireless Networking with IEEE 802.15.4 and 6LoWPAN“.
In: *Embedded Linux Conference Europe 2012 (ELC2012)*.
Nov. 2012.
URL: http://www.signal11.us/oss/elce2012/ (besucht am 10. 06. 2013).

[Perelman 2012] Vladislav Perelman. „Security in IPv6-enabled Wireless Sensor Networks: An Implementation of TLS/DTLS for the Contiki Operating System“. Master Thesis. Jacobs University | School of Engineering und Science, 2012.
URL: http://cnds.eecs.jacobs-university.de/archive/msc-2012-vperelman.pdf (besucht am 07. 06. 2013).

[RC2400] Radiocraft AS. *RC2400/RC2400HP - ZigBee®- Ready RF Transceiver Modules*. Englisch.
URL: http://www.radiocrafts.com/index.php?sideID=476 (besucht am 29. 02. 2012).

[RFC1122] R. Braden. *Requirements for Internet Hosts — Communication Layers*. Englisch. RFC 1122.
Internet Engineering Task Force, Okt. 1989.
URL: http://tools.ietf.org/html/rfc1122.

[RFC1157] Case, J.D. and Fedor, M. and Schoffstall, M.L. and Davin, J. *A Simple Network Management Protocol (SNMP)*. Englisch.
RFC 1157. Internet Engineering Task Force, Mai 1990.
URL: http://tools.ietf.org/html/rfc1157.

[RFC2460] S. Deering und R. Hinden.
Internet Protocol, Version 6 (IPv6) Specification. RFC 2460.
Internet Engineering Task Force, Dez. 1998.
URL: http://tools.ietf.org/html/rfc2460.

[RFC2616] Fielding, R. and Gettys, J. and Mogul, J. and Frystyk, H. and Masinter, L. and Leach, P. and Berners-Lee, T.
Hypertext Transfer Protocol – HTTP/1.1. Englisch. RFC 2616.
Internet Engineering Task Force, Juni 1999.
URL: http://tools.ietf.org/html/rfc2616.

[RFC3414] U. Blumenthal und B. Wijnen.
User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3). Englisch.
RFC 3414. Internet Engineering Task Force, Dez. 2002.
URL: http://tools.ietf.org/html/rfc3414.

[RFC4919] N. Kushalnagar, G. Montenegro und C. Schumacher.
IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals. RFC 4919.
Internet Engineering Task Force, Aug. 2007.
URL: http://tools.ietf.org/html/rfc4919.

[RFC4944] G. Montenegro u. a. *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*. Englisch. RFC 4944.
Internet Engineering Task Force, Sep. 2007.
URL: http://tools.ietf.org/html/rfc4944.

[RFC6282] J. Hui and P. Thubert. *Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks*. Englisch.
RFC 6282. Internet Engineering Task Force, Sep. 2011.
URL: http://tools.ietf.org/html/rfc6282.

[RFC6550] T. Winter and P. Thubert and A. Brandt and J. Hui and R. Kelsey and P. Levis and K. Pister and R. Struik and J. Vasseur and R. Alexander. *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*. Englisch. RFC 6550. Internet Engineering Task Force, März 2012. URL: http://tools.ietf.org/html/rfc6550.

[Schäffer 2008] Florian Schäffer. *AVR: Hardware und C-Programmierung in der Praxis*. Deutsch. 2. Aufl. Elektor-Verlag, 2008. ISBN: 978-3895762000.

[Shelby und Bormann 2009] Zach Shelby und Carsten Bormann. *6LoWPAN: the wireless embedded internet*. Englisch. John Wiley und Sons Ltd, 2009, S. I–XX, 1–223. ISBN: 978-0-470-74799-5.

[siemens:netzleittechnik] Siemens AG. *Netze steuern - Energieflüsse optimieren - Effizienz steigern: Spectrum Power Netzleittechnik*. Deutsch. Broschüre. URL: http://www.siemens.com/energy-automation (besucht am 01. 04. 2012).

[Stevens et al. 2004] Andrew M. Rudoff W. Richard Stevens Bill Fenner. *UNIX Network Programming: The Sockets Networking API - Volume 1 Third Edition*. Englisch. Addison Wesley Pearson Education, 2004, S. I–XXIII, 1–991. ISBN: 0-13-141155-1.

[Tsiftes et al. 2010] Nicolas Tsiftes, Joakim Eriksson und Adam Dunkels. „Poster Abstract: Low-Power Wireless IPv6 Routing with ContikiRPL“. In: *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN 2010)*. Stockholm, Sweden, Apr. 2010. URL: http://www.sics.se/~adam/tsiftes10rpl.pdf.

[Vasseur und Dunkels 2010] Jean-Philippe Vasseur und Adam Dunkels. *Interconnecting Smart Objects with IP - The Next Internet*. Morgan Kaufmann, 2010. ISBN: 978-0123751652. URL: http://TheNextInternet.org/.

[Vermesan und Friess 2011] Dr. Ovidiu Vermesan und Dr. Peter Friess. *The Internet of Things - Global Technological and Societal Trends*. Englisch. 1. Aufl. River Publishers, 2011. ISBN: 978-87-92329-73-8.

[Web:6lowpan-pars] Internet Assigned Numbers Authority. *IPv6 Low Power Personal Area Network Parameters*. Englisch. URL: http://www.iana.org/assignments/6lowpan-parameters/6lowpan-parameters.xml (besucht am 08. 06. 2013).

[Web:aiccu] SixXS. *Automatic IPv6 Connectivity Client Utility (aiccu)*. Englisch. URL: http://www.sixxs.net/tools/aiccu/ (besucht am 08. 06. 2013).

[Web:Atmel AVR] Atmel Corporation. *Atmel AVR*. Englisch. URL: http://www.atmel.com/products/microcontrollers/avr/default.aspx (besucht am 29. 12. 2012).

[Web:contiki-developers] The Contiki project and its contributors. *contiki-developers*. Englisch. URL: http://sourceforge.net/mailarchive/forum.php?forum_name=contiki-developers (besucht am 08. 06. 2013).

[Web:Contiki-syslog] Anuj Sehgal. *contiki-ntp-syslog*. Englisch. URL: http://cnds.eecs.jacobs-university.de/software/contiki-syslog/ (besucht am 10. 06. 2013).

[Web:enocean] EnOcean GmbH. *EnOcean-Funkstandard*. Deutsch. URL: http://www.enocean.com/de/enocean-wireless-standard/ (besucht am 08. 06. 2013).

[Web:freertos] Real Time Engineers Ltd. *FreeRTOS*. Englisch. URL: http://www.freertos.org/ (besucht am 08. 06. 2013).

[Web:google] Google Inc. *IPv6 Statistiken vom 08.06.2013*. Deutsch. URL: http://www.google.de/ipv6/statistics.html (besucht am 08. 06. 2013).

[Web:grosse] Ulrich Grosse. *Trafolose Stromversorgung*. Deutsch. URL: http://www.grosse-elektronik.de/das-elko/trlosestr/index.html (besucht am 08. 06. 2013).

[Web:Instant-Contiki] The Contiki project and its contributors. *Get Started with Contiki*. Englisch. URL: http://www.contiki-os.org/start.html%5C#install-instant-contiki (besucht am 08. 06. 2013).

[Web:ipv4-address-space] Internet Assigned Numbers Authority. *IANA IPv4 Address Space Registry*. Englisch. URL: http://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xml (besucht am 08. 06. 2013).

[Web:KNX] KNX. *KNX Association: International Standards*. URL: http://www.knx.org/knx-standard/standardisation/ (besucht am 08. 06. 2013).

[Web:LinuxMint] Linux Mint Community. *Linux Mint*. Englisch. URL: http://www.linuxmint.com/about.php (besucht am 08. 06. 2013).

[Web:nmap] Gordon Lyon. *Nmap - Network Mapper*. Englisch. URL: http://www.nmap.org/6/ (besucht am 08. 06. 2013).

[Web:radvd] Linux Community.
Linux IPv6 Router Advertisement Daemon (radvd). Englisch.
URL: http://www.litech.org/radvd/ (besucht am 08. 06. 2013).

[Web:SixXS] SixXS. *SixXS - IPv6 Deployment & Tunnel Broker*. Englisch.
URL: http://www.sixxs.net (besucht am 08. 06. 2013).

[Web:SOIL] Forschungsinstitut für Rationalisierung (FIR) e. V. an der RWTH Aachen. *Smart-Objects-Innovation-Lab*. Deutsch.
URL: http://www.fir.rwth-aachen.de/campus/smart-objects-innovation-lab (besucht am 08. 06. 2013).

[Web:Tchibo Steckdose] Tchibo direct GmbH.
Digitale Zeitschaltuhr bei Tchibo. Deutsch.
URL: http://www.tchibo.de/Digitale-Zeitschaltuhr-p200006363.html (besucht am 08. 06. 2013).

[Web:tinyos] TinyOS Alliance. *TinyOS Community*. Englisch.
URL: http://www.tinyos.net/community.html (besucht am 08. 06. 2013).

[Web:Z-202] NETVOX TECHNOLOGY CO., LTD.
Netvox Z-202. Englisch.
URL: http://www.netvox.com.tw/Z-202.asp (besucht am 08. 06. 2013).

[Web:Z-Wave] Z-Wave. *Z-Wave Alliance*. URL:
http://www.z-wavealliance.org/technology (besucht am 08. 06. 2013).

[Web:ZigBee] ZigBee Alliance. *ZigBee Standards Overview*. Englisch. URL:
http://zigbee.org/Standards/Overview.aspx (besucht am 08. 06. 2013).

[Weichert und Wülker 2010] Norbert Weichert und Michael Wülker.
Messtechnik und Messdatenerfassung. Deutsch.
Oldenbourg Wissenschaftsverlag, 2010, S. I–X, 1–323.
ISBN: 978-3486597738.

[X.25] International Telecommunication Union.
X.25, (10/96). Englisch. Okt. 1996.

[Zangemeister 1976] Christof Zangemeister.
Nutzwertanalyse in der Systemtechnik. Deutsch. 4. Aufl.
Wittemannsche Buchhandlung, 1976. ISBN: 3-923264-00-3.

[Zehl 2012] Sven Zehl. „Entwicklung und Test einer SNMP- basierten Automatisierungslösung für Sensorknoten in einem 6LoWPAN Funknetz“.
Bachelorarbeit. Beuth Hochschule für Technik Berlin, 2012.

[ZigBee-IP] ZigBee Alliance. *ZigBee IP Specification*. Englisch.
Feb. 2013.
URL: http://www.zigbee.org/Specifications/ZigBeeIP/Download.aspx (besucht am 13. 04. 2013).