Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Till Gerken

Design und Entwicklung eines WLAN-basierten verteilten Systems zur Messwerterfassung mit Cortex-M3-Mikrocontrollern unter Berücksichtigung der Energieeffizienz

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Till Gerken

Design und Entwicklung eines WLAN-basierten verteilten Systems zur Messwerterfassung mit Cortex-M3-Mikrocontrollern unter Berücksichtigung der Energieeffizienz

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. rer. nat. Hans Heinrich Heitmann
Zweitgutachter: Prof. Dr. Franz Korf

Eingereicht am: 24. August 2012

Till Gerken

Titel der Arbeit
Design und Entwicklung eines WLAN-basierten verteilten Systems zur Messwerterfassung mit Cortex-M3-Mikrocontrollern unter Berücksichtigung der Energieeffizienz

Stichworte
Aktive Objekte, ARM Cortex-M3, CoAP, Constrainded Application Protocol, Ereignisgesteuertes System, Internet der Dinge, Proxy, Quantum Platform, Ressource, WLAN, WLAN-Modul

Kurzzusammenfassung
Derzeit sind zahlreiche Produkte erhältlich, um eingebettete Systeme auf einfache Weise mit der Fähigkeit auszustatten, über WLAN Daten zu senden und zu empfangen. Zugleich gibt es unter dem Begriff „Internet der Dinge“ einen Trend, kleine batteriebetriebene Geräte (z. B. Messstationen) mithilfe von Funktechnologien an das Internet anzuschließen. Diese Entwicklungen gaben den Anlass für diese Arbeit, in der der Frage nachgegangen wird, wie Daten von batteriebetriebenen Mikrocontroller-Systemen über WLAN abgefragt werden können. Zur Datenabfrage werden Webbrowser benutzt, damit herkömmliche Computer wie Notebooks oder Smartphones verwendet werden können. Kern der Arbeit ist die Entwicklung eines prototypischen verteilten Systems zur Messwerterfassung. Dabei spielte der Aspekt der Energieeffizienz eine besondere Rolle. Die einzelnen Knoten bestehen jeweils aus einem ARM Cortex-M3-Mikrocontroller und einem WLAN-Modul.

Till Gerken

Title of the Thesis
Design and Development of a WLAN based Distributed System for Data Logging with Cortex-M3 Microcontrollers considering Power Efficiency

Keywords
active objects, ARM Cortex-M3, CoAP, Constrainded Application Protocol, event-driven system, Internet of Things, proxy, Quantum Platform, resource, WLAN, WLAN module

Abstract
Nowadays numerous products are available to equip embedded systems in a simple way with the ability for sending and receiving data via WLAN. At the same time there is a trend, known by the term "Internet of Things", to connect small battery-driven devices (e.g. smart meters) to the Internet by means of wireless technology. These developments gave rise to this work, in which the question is addressed of how data can be retrieved by battery-driven systems of microcontrollers via WLAN. For data retrieval web browsers are used, so that conventional computers such as notebooks and smartphones can be utilized. The very essence of this work is the development of a prototypical distributed system for data-logging. Thereby the aspect of power efficiency plays a major role. The single nodes consist each of an ARM Cortex-M3 microcontroller and a WLAN module.

Inhaltsverzeichnis

1 Einleitung

1.1 Problemumfeld

Wireless Local Area Network (WLAN) ist eine heutzutage weit verbreitete Technologie zur drahtlosen Datenübertragung im Nahbereich (bis 300 m) und eine gängige Methode, Personal Computer mit dem Internet zu verbinden. Da bei der Entwicklung des WLAN-Standards mehr Wert auf eine hohe Bandbreite gelegt wurde als auf die Unterstützung energieeffizienter Transceiver, war die Nutzung von WLAN lange Zeit Endgeräten mit stationärer Stromversorgung oder mit leistungsstarken Akkus vorbehalten, d. h. überwiegend Desktop-Rechnern und Notebooks, seit Kurzem auch Smartphones. In letzter Zeit ist unter dem Begriff „Internet der Dinge“ ein Trend zu verzeichnen, auch eingebetteten Kleinst-Computern eine Identität im Internet zu geben. Wichtige Voraussetzungen hierfür sind, dass jedes Gerät eine IP-Adresse erhält sowie Zugang zu einem (vorwiegend drahtlosen) Netzwerk hat, welches die Verbindung mit dem Internet herstellt. Mittlerweile bietet die Industrie eine Vielzahl von Produkten an, um batteriebetriebene Mikrorechnersysteme mit einem Zugang zu WLAN-Netzen auszustatten. Obgleich es andere Funkübertragungssysteme gibt, die weit besser für energieeffiziente kleine Geräte geeignet sind, gibt es durchaus gute Gründe, solche Geräte mithilfe von WLAN an IP-basierte Netzwerke anzuschließen. Eine wichtige Rolle spielt hierbei der hohe Verbreitungsgrad: WLAN-Netze sind an vielen Orten verfügbar (in Privathaushalten, öffentlichen Einrichtungen, Firmengebäuden usw.), daher liegt der Gedanke nahe, eingebettete Geräte, die im Empfangsbereich eines solchen Netzes eingesetzt werden, mithilfe von WLAN netzwerkfähig zu machen. Die vorhandene Infrastruktur kann genutzt werden und den Geräten stehen sämtliche Dienste und Weiterleitungsmöglichkeiten, die das jeweilige Netzwerk bietet, unmittelbar zur Verfügung. Werden hingegen andere Funktechnologien genutzt, entstehen entweder isolierte Netzwerke oder es müssen erst Übergangspunkte in Form von Routern zur Anbindung an die vor Ort bereits existierenden Netzwerke geschaffen werden.

1.2 Szenario

Grundlage dieser Bachelor-Arbeit ist das nachfolgend beschriebene Szenario. Kleine batteriebetriebene Mikrocontroller-Systeme „produzieren“ Daten, die von anderen Computern abgerufen werden können. Von der originären Aufgabe der Geräte hängt ab, um was für Daten es sich handelt und in welchem Format sie dargestellt werden. Beispielsweise könnte es sich bei solch einem Gerät um eine Sensorstation handeln, die periodisch Messungen vornimmt und die Messwerte in einem geeigneten Format zur Abfrage bereitstellt. Im Rahmen dieser Arbeit spielt der Anwendungszweck der Geräte und damit einhergehend das Zustandekommen der Daten keine wesentliche Rolle. Es wird lediglich davon ausgegangen, dass sich die Daten im Laufe der Zeit ändern und dass die Daten nur wenige Bytes groß sind. Im weiteren Verlauf der Arbeit wird ein Gerät, von dem Daten abgefragt werden können, einheitlich als Datenendpunkt (DEP) bezeichnet. Die Datenendpunkte sind mit einem WLAN-Transceiver ausgestattet und können sich mit einem an ihrem Einsatzort vorhandenen WLAN-Netz verbinden. Benutzer können die Daten abfragen, die die Datenendpunkte bereitstellen. Sie benötigen hierzu ein WLAN-fähiges Endgerät, das mit einem TCP/IP-Stack und einem Webbrowser ausgestattet ist (z. B. PC, Notebook oder Smartphone). Die Abfrage von Daten eines Datenendpunktes gestaltet sich folgendermaßen: Der Benutzer verbindet sein Endgerät mit demselben WLAN-Netz, in dem sich der DEP befindet.[1] Nach der Eingabe einer URL werden die angeforderten Informationen im Webbrowser angezeigt.

Das beschriebene Szenario weist drei prägnante Merkmale auf. Auf der Ebene des Netzzugangs (die unterste Schicht im TCP/IP-Referenzmodell) wird das WLAN-Protokoll nach IEEE 802.11-Standard eingesetzt. Weiter oben ist erläutert worden, warum diese Wahl sinnvoll sein kann. Das zweite Merkmal ist der Umstand, dass der Benutzer zur Datenabfrage heutzutage weit verbreitete Endgeräte verwenden kann, an die keine erweiterten Hardware- oder Software-Anforderungen gestellt werden. Vorausgesetzt werden lediglich ein WLAN-Transceiver ein TCP/IP-Stack und ein Webbrowser, womit sichergestellt ist, dass die allermeisten Desktop-Rechner, Notebooks und Smartphones geeignet sind. Das dritte charakteristische Merkmal stellt die Art der Computer dar, von denen Daten abgefragt werden können. Es handelt sich hierbei um kleine Mikrorechnersysteme mit beschränkten Ressourcen, sogenannte Constrained Devices. Darunter ist zu verstehen, dass die Geräte nur über wenig Arbeits- und Programmspeicher sowie über eine geringe Rechenleistung verfügen. Zudem sind die Geräte meist batteriebetrieben und dürfen daher nur einen geringen Stromverbrauch haben. Computer dieser Kategorie sind typischerweise gemeint, wenn im Zusamenhang mit dem Internet der Dinge von „Smart Things“ die Rede ist.

1 Sofern der DEP eine routbare Adresse hat, genügt es, wenn sich das Endgerät mit einem Netzwerk verbindet, von dem aus das WLAN-Netz des Datenendpunktes erreicht werden kann.

1.3 Zielsetzung

In dieser Arbeit wird die Praxistauglichkeit des dargestellten Szenarios untersucht. Hierzu werden zunächst theoretische Betrachtungen angestellt, um die Möglichkeiten und Grenzen der Idee zu erkunden. Darauf aufbauend wird ein realisierbares Konzept erstellt. Ein Großteil der Arbeit widmet sich dann der Entwicklung eines funktionsfähigen Prototypen. Der Prototyp trägt den Namen „verteiltes System zur Messwerterfassung“. Die Knoten (das sind die Rechner, aus denen das verteilte System aufgebaut ist) bestehen aus je einem ARM Cortex-M3-Mikrocontroller und einem daran angeschlossenen WLAN-Modul.

2 Anforderungsanalyse und Systemdesign

2.1 Struktur des verteilten Systems zur Messwerterfassung

In diesem Abschnitt wird die physische Struktur des verteilten Systems zur Messwerterfassung festgelegt. Es wird dargestellt, aus welchen Knoten das verteilte System aufgebaut ist, welche Rolle diese Knoten innerhalb des verteilten Systems einnehmen und welche Kommunikationsbeziehungen zwischen den Knoten bestehen.

Alle Komponenten kommunizieren drahtlos miteinander über ein WLAN-Netzwerk. Aus der Beschreibung des Szenarios geht unmittelbar hervor, dass Knoten existieren, die Daten zur Abfrage bereitstellen und solche, die diese Daten abfragen. Es ist bereits vereinbart worden, erstgenannte Knoten Datenendpunkte zu nennen. Die zweite Kategorie von Knoten sind Computer, die von menschlichen Benutzern zur Abfrage der von den Datenendpunkten bereitgestellten Informationen verwendet werden. Diese sind mit einem WLAN-Transceiver, einem TCP/IP-Stack und einem Webbrowser ausgestattet und werden in dieser Arbeit einheitlich Benutzerendpunkte (BEP) genannt. Die Anzahl der Datenendpunkte und die Anzahl der Benutzerendpunkte sind variabel, typischerweise jeweils größer als eins. Gleichwohl wird in dieser Arbeit der Begriff Datenendpunkt und Benutzerendpunkt oftmals im Singular verwendet, um präzisere Formulierungen zu ermöglichen.

Ein Datenendpunkt muss sehr energieeffizient arbeiten, da er in einer typischen Anwendung mit Batterien betrieben wird. Der größte Einspareffekt lässt sich erzielen, indem der WLAN-Transceiver eines Datenendpunktes möglichst oft und möglichst lange abgeschaltet ist. Anhand der Tabelle 2.1 lässt sich grob abschätzen, welches Potenzial diese Methode bietet. Für die

Betriebszustand	Stromstärke bei 3.3 V
Schlaf-Modus	4 μA
Standby-Modus	15 mA
Verbunden (Ruhezustand, RX)	40 mA
Verbunden (TX)	180 mA bei 10 dB

Tabelle 2.1: Stromverbrauch eines WLAN-Moduls (aus RN-1742012 (2012, S. 2))

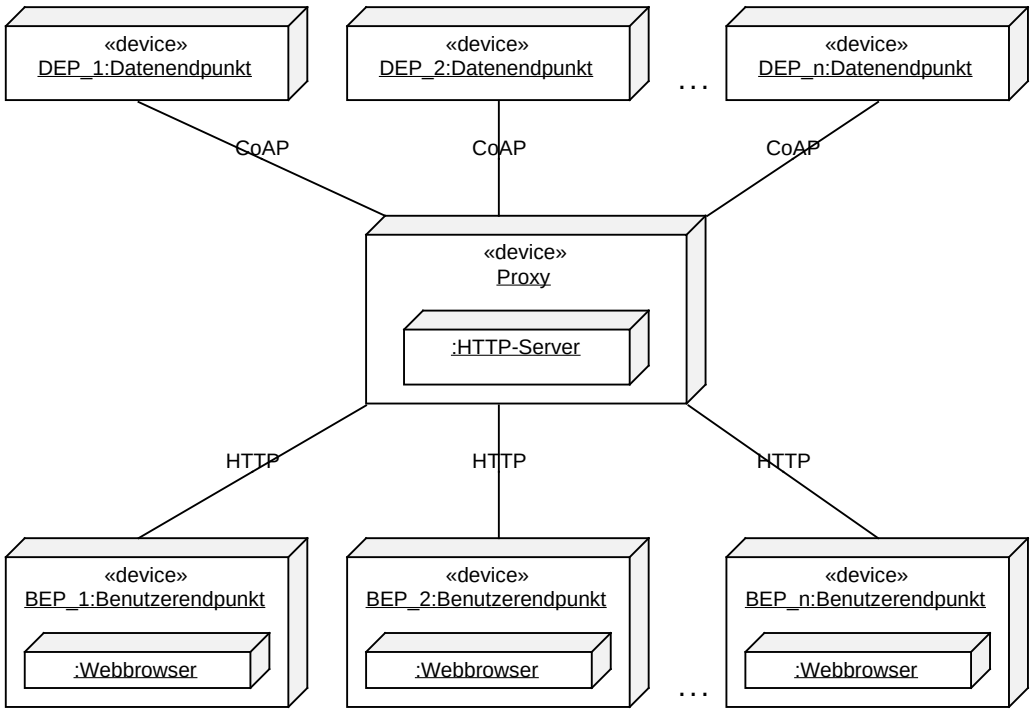

Abbildung 2.1: Verteilungsdiagramm des verteilten Systems zur Messwerterfassung

Stromversorgung des Datenendpunktes sollen herkömmliche Batterien verwendet werden können. Daher kann eine akzeptable Batterielaufzeit nicht erreicht werden, wenn der WLAN-Transceiver permanent empfangsbereit ist. Idealerweise sollte der Transceiver nur dann eingeschaltet sein, wenn Daten übertragen werden müssen.

Wesentlicher Bestandteil eines jeden Benutzerendpunktes ist ein Webbrowser. Ein Webbrowser ist ein HTTP-Client, daraus folgt unmittelbar, dass die Kommunikation eines Benutzerendpunktes mit einem anderen Knoten mithilfe des HTTP-Protokolls stattfindet und dass auf dem anderen Knoten ein HTTP-Server laufen muss. Da ein Benutzerendpunkt Daten eines Datenendpunktes abrufen soll, liegt der Gedanke nahe, den Datenendpunkt mit einem HTTP-Server auszustatten. Eine derartige Client-Server-Beziehung zwischen Benutzerendpunkt und Datenendpunkt hat jedoch einen entscheidenden Nachteil. Der Datenendpunkt muss permanent empfangsbereit sein, um jederzeit Anfragen eines Benutzerendpunktes beantworten zu können. Dies kann der Datenendpunkt nur gewährleisten, indem sein WLAN-Transceiver dauerhaft eingeschaltet ist, was jedoch aus Gründen der Energieeffizienz nicht realisierbar ist.

Eine zentrale Rolle bei der Konzeption des verteilten Systems zur Messwerterfassung spielt daher die Auflösung des nachfolgend formulierten Widerspruchs: Benutzerendpunkte sollen jederzeit auf die Informationen zugreifen können, die von Datenendpunkten bereitgestellt werden, obwohl diese die überwiegende Zeit ihrer Lebensdauer nicht erreichbar sind.

Beide Anforderungen können erfüllt werden, indem es innerhalb des verteilten Systems einen Knoten mit besonderer Funktionalität gibt: den Proxy. Der Proxy ist im Gegensatz zu den Datenendpunkten ständig empfangsbereit. Wesentliche Bestandteile des Proxy sind ein HTTP-Server und ein Cache. Ein Datenendpunkt kann eine Kopie seiner Daten (z. B. Messdaten) im Cache des Proxy speichern. Wenn ein Benutzerendpunkt die Daten eines Datenendpunktes abfragen möchte, kommuniziert er nicht direkt mit diesem, sondern richtet seine HTTP-Anfrage an den Proxy. Der Proxy generiert die Antwort aus den Daten, die in seinem Cache gespeichert sind.

Aus logischer Sicht findet also eine Kommunikation zwischen Benutzerendpunkt und Datenendpunkt statt. Auf Netzwerkebene ist diese Kommunikationsbeziehung jedoch durch den Proxy entkoppelt: Sowohl Benutzerendpunkt als auch Datenendpunkt kommunizieren mit dem Proxy, einen direkten Datenverkehr zwischen beiden gibt es hingegen nicht. Diese Entkoppelung bietet einige Vorteile:

- Ein Benutzerendpunkt kann jederzeit die Daten eines Datenendpunktes abfragen.
- Ein Datenendpunkt kann seine Schlaf-/Wachphasen selbst bestimmen. Sein WLAN-Transceiver muss nicht permanent empfangsbereit sein, um auf Anfragen reagieren zu können. Er muss

den WLAN-Transceiver nur dann einschalten, wenn er Kontakt zum Proxy aufnehmen möchte.

- Da der Datenendpunkt nicht mit dem Benutzerendpunkt kommuniziert, muss er nicht das HTTP-Protokoll implementieren. Für den Datenaustausch zwischen Datenendpunkt und Proxy kann ein einfacheres Protokoll verwendet werden, das einen datagrammorientierten Transportdienst wie UDP nutzt. Somit muss im Datenendpunkt kein TCP/IP-Stack vorhanden sein. Dies reduziert den Bedarf an Rechenleistung und Speicher, die die Hardware des Datenendpunktes für die Kommunikationsaufgaben bereitstellen muss.
- Zwischen Datenendpunkt und Proxy kann eine andere drahtlose Datenübertragungstechnologie zum Einsatz kommen, die den Anforderungen des Anwendungsszenarios besser gerecht wird.

Das UML-Verteilungsdiagramm in Abbildung 2.1 veranschaulicht die in diesem Abschnitt entworfene Struktur.[1] Das verteilte System zur Messwerterfassung besteht aus einem Proxy, einer variablen Anzahl an Benutzerendpunkten und einer variablen Anzahl an Datenendpunkten. Die Rolle der Benutzerendpunkte wird laut Aufgabenstellung von WLAN-fähigen Endgeräten mit Webbrowser übernommen. Die Kommunikation zwischen Benutzerendpunkt und Proxy erfolgt mittels HTTP. Datenendpunkt und Proxy werden im Rahmen dieser Arbeit entwickelt. Die folgenden Anforderungen müssen erfüllt werden:

- Datenendpunkte müssen zu beliebiger Zeit Kopien ihrer Daten beim Proxy speichern, aktualisieren und löschen können.
- Der Proxy muss einen HTTP-Server bereitstellen, der als Schnittstelle für die Anfragen der Benutzerendpunkte dient. Benutzerendpunkte müssen zu beliebiger Zeit mittels HTTP-GET-Requests die im Proxy gespeicherten Daten der Datenendpunkte abfragen können.

2.2 Verbindung mit einem WLAN-Netzwerk

In diesem Abschnitt werden ein paar Aspekte der Technologie WLAN näher erläutert, die für die Realisierung des verteilten Systems zur Messwerterfassung von Interesse sind. Anschließend wird beschrieben, welche Anforderungen Datenendpunkt und Proxy erfüllen müssen, damit sie sich mit einem WLAN-Netzwerk verbinden können.

1 In dem Verteilungsdiagramm ist das zwischen Datenendpunkt und Proxy eingesetzte Anwendungsschicht-Protokoll bereits eingetragen. Die Auswahl des CoAP-Protokolls wird in Abschnitt 2.4 beschrieben.

2.2.1 Wireless Local Area Network

WLAN-Netzwerke können in unterschiedlichen Modi betrieben werden, im Ad-hoc-Modus oder im Infrastruktur-Modus. Im Ad-hoc-Modus sind alle WLAN-Endgeräte gleichberechtigt und zwei Stationen kommunizieren auf direktem Wege ohne Beteiligung einer weiteren Station miteinander. Daraus folgt, dass nur diejenigen Stationen miteinander kommunizieren können, die sich gegenseitig empfangen können.

Die weitaus häufiger genutzte Betriebsart von WLAN ist der Infrastruktur-Modus. In einem derartigen Netzwerk gibt es einen oder mehrere Stationen mit besonderen Aufgaben, die Access Points (AP). Wenn ein WLAN-Endgerät ein Paket an ein anderes Endgerät senden möchte, so tut es das nicht auf direktem Wege, sondern schickt das Paket an einen Access Point. Dieser leitet es (ggf. unter Beteiligung weiterer Access Points) an den Empfänger weiter. Sender und Empfänger müssen sich nicht gegenseitig empfangen können, sie müssen sich lediglich im Empfangsbereich eines Access Points befinden. Dank der Vermittlung aller Pakete über einen Access Point kann ein Infrastruktur-WLAN eine größere räumliche Ausdehnung haben als ein Ad-hoc-Netzwerk. Zudem kann die Ausdehnung weiter erhöht werden, indem mehrere Access Points installiert werden. Die Access Points sind untereinander über Ethernet oder über eine Funkstrecke (Wireless Bridging) verbunden und tauschen auf diesem Wege Verwaltungsinformationen und Pakete aus, die an die Endgeräte zugestellt werden sollen. Dank der Access Points können in einem Infrastruktur-WLAN anders als in einem Ad-hoc-Netzwerk auch Stationen miteinander kommunizieren, die für eine direkte Kommunikation zu weit auseinanderliegen. Nachteilig an Infrastruktur-Netzwerken ist die Tatsache, dass die maximale Bandbreite nur halb so groß ist wie die eines Ad-hoc-Netzwerks, da jedes Paket wegen des „Umwegs“ über einen Access Point zweimal die Luftschnittstelle belegt.

Damit WLAN-Netzwerke von Stationen gefunden werden können, senden Access Points in regelmäßigen Abständen (typisch sind 100 ms) Beacon Frames, die u. a. eine eindeutige Bezeichnung des Netzwerks, die sogenannte Basic Service Set ID (SSID) enthalten. Durch die SSID ist eine Station in der Lage, verschiedene WLAN-Netzwerke zu unterscheiden, die sich am Standort der Station räumlich überschneiden.

WLAN bietet verschiedene Verschlüsselungsmethoden, damit die Kommunikation über die Luftschnittstelle nicht von unbefugten Stationen abgehört werden kann. Das derzeit sicherste Verfahren heißt WPA2, das in den Varianten WPA2 Personal Mode und WPA2 Enterprise Mode existiert. Eine Station muss zur Anmeldung an einem mit WPA2 Personal Mode gesicherten WLAN das Passwort des Access Points (Pre-Shared Key, PSK) kennen. Zur Anmeldung an einem mit WPA2 Enterprise Mode gesicherten WLAN ist hingegen eine Kombination aus Benutzername und Passwort erforderlich.

2.2.2 Anforderungen an Datenendpunkt und Proxy

- Der Datenendpunkt/Proxy muss sowohl in einem im Infrastruktur-Modus betriebenen als auch in einem im Ad-hoc-Modus betriebenen WLAN-Netzwerk einsetzbar sein.
- Der Datenendpunkt/Proxy soll selbsttätig die Verbindung zu einem WLAN-Netzwerk herstellen.
- Aufgrund der hohen Verbreitung sowie der hohen Sicherheit der WPA2-Verschlüsselung muss der Datenendpunkt/Proxy dieses Verfahren unterstützen, nach Möglichkeit sowohl in der Variante WPA2 Personal Mode als auch in der Variante WPA2 Enterprise Mode.
- Um die selbsttätige Einwahl in ein WLAN zu ermöglichen, muss die SSID des zu benutzenden Netzwerks sowie das Passwort (im Falle von WPA2 Personal Mode) bzw. der Benutzername und das Passwort (im Falle von WPA2 Enterprise Mode) vorab im Datenendpunt/Proxy konfiguriert werden können.
- In der Regel wird der Proxy seine IP-Adresse nicht frei wählen können. Das WLAN-Netzwerk, mit dem er sich verbindet, wird ein Verfahren zur Vergabe von IP-Adressen bereitstellen (z. B. durch DHCP oder Zeroconf). Damit der Proxy von Benutzerendpunkten und Datenendpunkten erreicht werden kann, muss seine IP-Adresse bekannt sein. Es muss eine Möglichkeit vorhanden sein, die IP-Adresse des Proxy zu erfahren, beispielsweise indem Broadcast-Nachrichten verschickt werden.

2.3 Schnittstellen zur Interaktion zwischen den Knoten

In diesem Abschnitt wird die Frage diskutiert, wie die Schnittstellen konzipiert sind, die Knoten anderen Knoten zur Nutzung ihrer Services anbieten. Anstatt individuelle Lösungen für die Interaktion zwischen Datenendpunkt und Proxy sowie zwischen Benutzerendpunkt und Proxy zu entwickeln, soll ein einheitliches Modell verwendet werden.

Guinard u. a. (2010) schlagen vor, das Internet der Dinge mit einer einheitlichen Applikationsschicht zu versehen, die es ermöglicht, „Smart Things“ in das World Wide Web (WWW) zu integrieren. Sie nennen ihren Ansatz das „Web der Dinge“. Kern ihrer Idee ist es, eines der wichtigsten Konzepte des Webs, den REST-Architekturstil, auch im Internet der Dinge einzusetzen. REST steht für Representational State Transfer und lässt sich folgendermaßen beschreiben: Das World Wide Web ist ein weltweites verteiltes System, das aus Endpunkten (Hosts) besteht, die über IP-basierte Netzwerke miteinander kommunizieren. Ein Endpunkt, der einen Service im

WWW anbietet, macht diesen in Form von Ressourcen verfügbar. Eine Ressource ist eine Abstraktion für eine Komponente einer Web-Anwendung, auf die andere Endpunkte zugreifen können. Um sie identifizieren zu können, besitzt jede Ressource eine eindeutige Adresse, den Uniform Resource Identifier (URI). Der Zugriff auf eine Ressource erfolgt nach dem Client-Server-Modell: Jede Ressource ist auf einem Server beheimatet (gehostet) und wird von diesem kontrolliert. Darunter ist insbesondere zu verstehen, dass der Zustand der Ressource ausschließlich von dem Server bestimmt/gespeichert wird, auf dem sie gehostet ist. Ein Endpunkt, der auf die Ressource zugreift, nimmt die Rolle des Client ein. Er schickt eine Anfrage (Request) an den Server, in der die URI und eine Anfragemethode (z. B. GET, PUT, POST, DELETE) enthalten ist. Der Server führt die durch die Anfrage identifizierte Aktion aus und schickt dem Client eine Antwort (Response).

Dieser Architekturstil soll für das verteilte System zur Messwerterfassung verwendet werden. Die Daten, die die Datenendpunkte bereitstellen, werden durch Ressourcen repräsentiert. Die Datenendpunkte speichern ihre Ressourcen beim Proxy. Dort können sie von Benutzerendpunkten abgerufen werden. Alle Interaktionen zwischen Datenendpunkten und Proxy sowie zwischen Benutzerendpunkten und Proxy basieren auf den REST-Prinzipien, d. h. es werden Anfragemethoden und URIs verwendet.

2.4 Protokoll zwischen Datenendpunkt und Proxy

In diesem Abschnitt werden geeignete Protokolle der Transportschicht sowie der Anwendungsschicht ausgewählt, die für die Kommunikation zwischen Datenendpunkt und Proxy eingesetzt werden. Die Protokolle der unteren Schichten sind aufgrund der Vorgabe WLAN zu verwenden bereits festgelegt.

2.4.1 Allgemeine Anforderungen

Ein verteiltes System wird in nicht unerheblichem Maße von den Protokollen geprägt, die für die Kommunikation zwischen den Knoten genutzt werden. Innerhalb eines verteilten Systems werden Protokolle für bestimmte Zwecke verwendet und diese Zwecke sollten sie möglichst gut erfüllen. Des Weiteren ist anzustreben, Protokolle einzusetzen, deren Implementierungs- und Ausführungsaufwand gering ist. In dieser Arbeit genießt letztere Forderung eine besonders hohe Priorität, weil Mikrorechnersysteme mit beschränkten Ressourcen an Rechenleistung und Speicher zum Einsatz kommen. Weitere Aufmerksamkeit bei der Protokollauswahl kommt der Erweiterungsfähigkeit zu: Wenn das verteilte System mit neuen Funktionen ausgestattet wird, soll das gewählte Protokoll weiter verwendet werden. Das Protokoll soll um neue Aufgaben

erweitert werden können und zugleich kompatibel mit bereits bestehenden Implementierungen bleiben.

2.4.2 Transportschicht

Auf der Ebene der Transportschicht stellt sich die Frage, ob das einfache User Datagram Protocol (UDP) oder das komplexere Transmission Control Protocol (TCP) zum Einsatz kommen soll. Im Gegensatz zu UDP stellt TCP der Anwendungsschicht einen zuverlässigen, verbindungsorientierten Transportdienst bereit inklusive Flusssteuerung und Staukontrolle. Anwendungsschicht-Protokolle, die auf derartige Eigenschaften angewiesen sind, können entweder TCP als Transportprotokoll nutzen oder die benötigten Funktionen in geeigneter Weise selbst implementieren. Auf den ersten Blick scheint die Bindung an TCP sinnvoll zu sein, weil dadurch ein Anwendungsschicht-Protokoll keine Funktionalität implementieren muss, die durch TCP schon bereitgestellt wird. Daher wird TCP von vielen Anwendungsschicht-Protokollen genutzt, beispielsweise auch von HTTP. Hingegen ist die Verwendung von TCP als Transportprotokoll für die Kommunikation zwischen Datenendpunkt und Proxy problematisch. Die Interaktion ist geprägt von periodischen, kurzzeitigen Transaktionen, bei denen jeweils nur wenige Bytes an Nutzdaten ausgetauscht werden. In derartigen Fällen weist TCP einen großen Protokoll-Overhead auf. Dies ist damit zu erklären, dass das Hauptaugenmerk beim Entwurf des TCP-Protokolls auf der effizienten und zuverlässigen Übertragung großer Datenmengen von einem Sender zu einem Empfänger lag. Ein weiteres Manko stellt die Tatsache dar, dass für den Betrieb eines TCP/IP-Stack auf einem Mikrocontroller eine verhältnismäßig große Menge an Arbeitsspeicher erforderlich ist. Es gibt speziell für Mikrocontroller optimierte Implementierungen, die nur wenig Arbeitsspeicher benötigen, z. B. den von Adam Dunkels entwickelten Open-Source-TCP/IP-Stack lwIP. Trotz der Optimierungen muss aber damit gerechnet werden, dass die Hardware-Anforderungen höher sind als bei Verwendung von UDP und ggf. ein leistungsfähigerer (und somit teurerer) Mikrocontroller eingesetzt werden muss. Daher sollte eine derartige Lösung nur dann in Betracht gezogen werden, wenn es triftige Gründe für den Einsatz von TCP gibt. Bei der Interaktion zwischen Datenendpunkt und Proxy ist TCP nicht vonnöten, um die Zuverlässigkeit der Datenübertragung zu garantieren. Denn bei WLAN wird jedes Datenpaket von der Gegenstelle nach korrektem Empfang durch ein Acknowledgement Frame bestätigt. Bleibt die Bestätigung aus, wird das Datenpaket erneut gesendet (Sauter 2011, S. 356).

2.4.3 Anwendungsschicht

Hypertext Transfer Protocol

Die Interaktion zwischen Benutzerendpunkt und Proxy basiert auf HTTP-Kommunikation. Zwischen diesen Knoten werden die gleichen Nutzdaten ausgetauscht wie zwischen Datenendpunkt und Proxy. Daher ist in Erwägung zu ziehen, HTTP auch für den Datenaustausch zwischen Datenendpunkt und Proxy zu verwenden. HTTP ist im heutigen World Wide Web das dominierende Anwendungsschicht-Transportprotokoll. Für die Machine-To-Machine-Kommunikation zwischen leistungsschwachen Mikrorechnersystemen ist HTTP hingegen aus mehreren Gründen nicht gut geeignet. HTTP nutzt TCP als Transportprotokoll. Ein TCP/IP-Stack kann jedoch auf vielen kleinen Geräten aufgrund der begrenzten Ressourcen an Speicher und Rechenleistung nicht implementiert werden. Wenn nur wenige Bytes an Nutzdaten übertragen werden sollen, ist der Protokoll-Overhead von HTTP groß (u. a. wegen der textbasierten Codierung der Nachrichten). Das HTTP zugrunde liegende Request-Response-Modell ist in drahtlosen Netzwerken für batteriebetriebene Endpunkte ungeeignet, die ihren Transceiver die meiste Zeit abgeschaltet haben, um Energie zu sparen.

Constrained Application Protocol

Um das Internet der Dinge voranzubringen, wird nach Ansicht der Internet Engineering Task Force (IETF) ein einheitliches Anwendungsschicht-Transportprotokoll benötigt, das im Bereich der Constrained Devices die Rolle einnimmt, die HTTP im heutigen WWW hat. Für die Entwicklung und Standardisierung eines solchen Protokolls hat die IETF am 9. 3. 2010 die Arbeitsgruppe Constrained RESTful Environments (CoRE) gegründet.[2] Das Constrained Application Protocol (CoAP) wird mit dem Ziel entwickelt, Kern-Konzepte des heutigen World Wide Web wie die REST-Architektur auch im Internet der Dinge nutzen zu können.

Der nachfolgenden Beschreibung des CoAP-Protokolls ist voranzuschicken, dass sich CoAP noch in aktiver Entwicklung befindet und alle bislang veröffentlichten Versionen den Zusatz Draft (Entwurf) tragen. Die einzelnen Versionen können sich in Teilbereichen voneinander unterscheiden. Das hat u. a. zur Folge, dass neuere Versionen nicht uneingeschränkt abwärtskompatibel sein müssen. Grundlage dieser Arbeit ist die Version 10 vom 25. 6. 2012 (Shelby u. a. 2012). Am 16. 7. 2012 ist bereits Version 11 veröffentlicht worden.

Der Name CoAP deutet bereits an, in welchem Umfeld das Protokoll zum Einsatz kommen soll: Es ist konzipiert für den Einsatz in leistungsschwachen Mikrorechnersystemen mit wenig Speicher, die über schmalbandige Netzwerke mit hohen Paketverlustraten miteinander kommunizieren.

2 http://datatracker.ietf.org/wg/core/history/

CoAP ist dafür ausgelegt, einen unzuverlässigen verbindungslosen Transportdienst wie UDP zu nutzen. Das entbindet die Endpunkte von der Aufgabe, einen TCP/IP-Stack bereitstellen zu müssen, was bei leistungsschwachen Geräten oftmals nicht darstellbar wäre. Dennoch können Anwendungen mit CoAP zuverlässig kommunizieren. Das Protokoll ermöglicht sowohl den Versand von Nachrichten, die vom Empfänger zu bestätigen sind, als auch den Versand von Nachrichten, deren Empfang nicht bestätigt wird. Das bedeutet, die Anwendung kann bei jedem Nachrichtenversand individuell festlegen, ob Zuverlässigkeit benötigt wird oder ob auf dieses Merkmal zugunsten einer geringeren Netzwerkbelastung verzichtet werden kann.

Um ineffiziente IP-Fragmentierung zu vermeiden, sollte ein UDP-Paket, das eine CoAP-Nachricht transportiert, nicht größer sein als die maximale Nutzdatengröße eines IP-Pakets. Das Nachrichtenformat von CoAP trägt dem Rechnung: Eine CoAP-Nachricht besteht aus einem vier Byte großen Header, einer variablen Anzahl von Optionen und ggf. den Nutzdaten. Header und Optionen sind platzsparend binär codiert. Da CoAP ein Transportprotokoll ist, das für diverse Anwendungszwecke genutzt werden kann, gibt es keine Vorgabe, wie die Nutzdaten zu codieren sind. Das Format wird von der Anwendung festgelegt. Mithilfe der Content-Type-Option ist es möglich, die in einer Nachricht verwendete Codierung explizit anzugeben. Anwendungsentwickler sollten effiziente Codierungen verwenden, damit die Nachrichten möglichst klein sind und der Aufwand zum Dekodieren (bzw. Parsen) auf Empfänger-Seite gering ist. Aus diesem Grund ist XML nicht gut geeignet. Stattdessen sollten Formate wie Extensible XML Interchange (EXI), Fast Infoset oder JavaScript Object Notation (JSON) genutzt werden (Shelby 2010, S. 54 f.). In dieser Arbeit werden die Nutzdaten als Plain-Text codiert.

2.5 Interaktion mit dem Proxy

Dieser Abschnitt beschreibt, wie Datenendpunkte und Benutzerendpunkte die Dienste des Proxy nutzen. In der CoAP-Terminologie wird der ursprüngliche Besitzer einer Ressource auch Origin Server genannt. Der Proxy muss bei den Anfragen, die er empfängt, unterscheiden, ob sie von einem Origin Server (d. h. von einem Datenendpunkt) oder von einem Benutzerendpunkt kommen.

2.5.1 Interaktion zwischen Datenendpunkt und Proxy

Der Proxy muss einen Dienst anbieten, den ein Datenendpunkt nutzen kann, um seine Ressourcen beim Proxy zu speichern, zu aktualisieren und zu löschen. Aufgrund der bereits getroffenen Entscheidung, für die Kommunikation zwischen Datenendpunkt und Proxy das Constrained Application Protocol einzusetzen, steht fest, dass dieser Dienst in Gestalt eines CoAP-Servers

realisiert werden muss. In (Fossati u. a. 2012) wird eine Protokollerweiterung von CoAP vorgeschlagen, die die benötigte Funktionalität bietet. Da die Lösung als zweckmäßig beurteilt werden kann, soll sie für den Proxy genutzt (implementiert) werden. Kern der Protokollerweiterung ist eine neue Publish-Option.

Ein Datenendpunkt (Origin Server), der die Dienste des Proxy nutzen möchte, sendet ihm eine Anfrage, die eine Proxy-Uri-Option und eine Publish-Option enthält. In der Proxy-Uri-Option ist die URI der Ressource des Datenendpunktes enthalten, auf die sich die Anfrage bezieht. Die Anfragemethode bestimmt, welche Operation der Datenendpunkt durchführen möchte: Mit einem PUT-Request speichert ein Datenendpunkt eine Ressource erstmalig beim Proxy oder aktualisiert eine bereits vorhandene Ressource. Die Repräsentation der Ressource ist im Nutzdatenteil der Nachricht enthalten. Mit einem DELETE-Request löscht der Datenendpunkt eine seiner Ressourcen beim Proxy. In der Publish-Option ist kodiert, welche Anfragemethoden erlaubt sind, wenn andere Endpunkte (z. B. Benutzerendpunkte) die auf dem Proxy gespeicherten Ressourcen abfragen, und welche vom Proxy als unerlaubte Methoden abgewiesen werden müssen. Die Publish-Option ist ein Byte groß. Nur die ersten vier Bits haben eine Bedeutung: Jede Bit-Position ist einer der vier möglichen CoAP-Anfragemethoden zugeordnet und der Wert des Bits bestimmt, ob die jeweilige Methode erlaubt ist oder nicht.

2.5.2 Interaktion zwischen Benutzerendpunkt und Proxy

Die Benutzerendpunkte, die die beim Proxy gespeicherten Ressourcen der Datenendpunkte abfragen wollen, greifen mithilfe eines Webbrowsers per HTTP auf den Proxy zu. Daher muss der Proxy einen HTTP-Server bereitstellen. Es stellt sich die Frage, wie der Webbrowser dem HTTP-Server mitteilt, auf welche Ressource er zugreifen möchte. Das allgemeine Konzept der Ressourcen sieht vor, dass jede Ressource durch eine eindeutige Zeichenkette identifiziert wird, den sog. Uniform Resource Identifier (URI). (Berners-Lee u. a. 2005) legt eine generische Syntax für URIs fest und konkrete Ausprägungen, sog. Schemata erweitern diese Syntax. Damit für jede URI festgestellt werden kann, nach welchem Schema sie gebildet worden ist, schreibt die generische Syntax vor, dass jede URI mit der einer Schema-Komponente beginnen muss (z. B. „http“, „coap“, „ftp“). Anschließend folgt (optional) die Authority-Komponente. Es ist wichtig zu verstehen, dass durch eine URI zunächst einmal eine Ressource nur eindeutig identifiziert wird. Die verbreitete Annahme, eine URI beschreibe auch die Art, wie auf die Ressource zugegriffen werden kann, ist ein Missverständnis (Berners-Lee u. a. 2005, Abschnitt 1.2.2.). Um diesen Sachverhalt zu verdeutlichen, wird beispielhaft angenommen, es existiert eine Ressource mit der URI „coap://messstation1:80/rssi“ . Dies impliziert nicht, dass auf die Ressource zugegriffen werden kann, indem ein CoAP-GET-Request an den Host „messstation1“ auf Port 80 geschickt wird. Es

kann durchaus sein, dass die Ressource abgerufen werden kann, indem ein anderer Rechner auf einem anderen Port über ein anderes Protokoll kontaktiert wird. Das Konzept, den Zugriff auf eine Ressource nicht auf direktem Wege, sondern durch eine oder mehrere Zwischenstationen zu ermöglichen, nennt sich Proxying. Eine Zwischenstation heißt Proxy. Proxying kann eine Reihe von Problemen lösen, die die Realisierung des Internets der Dinge erschweren.

Die Protokolle CoAP und HTTP unterstützen Proxying. Beide Protokolle sind in der Lage Anfragen zu transportieren, die URIs eines fremden Schemas beinhalten. Genau diese Art von HTTP-Anfragen wird im Rahmen dieser Arbeit benötigt. Ein HTTP-GET-Request der CoAP-Ressource aus obigem Beispiel sieht folgendermaßen aus:

```
GET coap://messstation1:80/rssi HTTP/1.1
```

Leider sind heutige Webbrowser nicht in der Lage, derartige HTTP-Anfragen zu erstellen. Sie ermitteln die Host-Adresse des Servers, an den die Anfrage geschickt werden soll, und die Request-URI anhand der in der Adresszeile eingegebenen Zeichenkette. In der Adresszeile werden jedoch nur Zeichenketten nach dem http-Schema akzeptiert.[3] Das Problem kann durch sogenanntes URI Mapping gelöst werden (Castellani u. a. 2012, Abschnitt 4): Der Proxy richtet für jede bei ihm gehostete CoAP-Ressource eine alternative URI ein, die mit heutigen Webbrowsern kompatibel ist. Angenommen der Proxy hat die Adresse 192.168.2.100, dann stellt er für die URI aus dem Beispiel folgende Alternative bereit: „http://192.168.2.100/coap/messstation1/80/rssi“. Diese URI kann ohne Probleme in der Adresszeile eines Webbrowsers eingegeben werden, und dieser erstellt daraus einen korrekten HTTP-GET-Request:

```
GET /coap/messstation1/80/rssi HTTP/1.1
```

Die Anfrage sendet der Webbrowser an den Host mit der IP-Adresse 192.168.2.100. Aus der Sicht des HTTP-Protokolls handelt es sich hierbei um eine Anfrage an einen normalen Server (Origin Server), nicht um eine Anfrage an einen Proxy.

3 Sofern in den Einstellungen des Browsers die Benutzung eines Proxyservers konfiguriert ist, wird die Zeichenkette ausschließlich zur Ermittlung der Request-URI herangezogen. Die HTTP-Anfrage wird immer an den konfigurierten Proxyserver gesendet. Auch in diesem Fall werden in der Adresszeile keine URIs akzeptiert, die nicht dem http-Schema entsprechen.

3 Hardware-Komponenten

Im letzten Kapitel wurde die grundlegende Struktur des verteilten Systems zur Messwerterfassung festgelegt sowie Anforderungen an die beteiligten Knoten formuliert. Das dient als Grundlage für dieses Kapitel, welches sich der Auswahl geeigneter Hardware-Komponenten für den Datenendpunkt und den Proxy widmet.

Eingangs sei nochmals auf die Zielsetzung dieser Arbeit verwiesen: Es soll ein Prototyp entwickelt werden, der die Machbarkeit der im Abschnitt 1.2 vorgestellten Idee demonstriert. Eine konkrete Anwendung liegt außerhalb des Betrachtungswinkels dieser Arbeit. In einem konkreten Einsatzszenario sind Randbedingungnen gegeben, die die Auswahl der Komponenten und evtl. Teilaspekte des Systemdesigns maßgeblich beeinflussen. Durch das Fehlen derartiger Randbedingungen bestand mehr Wahlfreiheit bei der Auswahl der Hardware-Komponenten. Somit war es möglich, den Kriterien Einarbeitungszeit, Entwicklungsaufwand und Wiederverwertbarkeit eine große Bedeutung beizumessen. Aufgrund der hohen Relevanz dieser Kriterien wurde entschieden, für die Realisierung von Datenendpunkt und Proxy die gleichen Hardware-Komponenten zu verwenden. Sowohl der Datenendpunkt als auch der Proxy bestehen jeweils aus einem Mikrocontroller und einem WLAN-Modul, die über eine kabelgebundene Schnittstelle miteinander verbunden sind.

3.1 Mikrocontroller

3.1.1 Anforderungen

In einer realen Anwendung ist es sinnvoll, die Auswahl eines geeigneten Mikrocontrollers für Datenendpunkt und Proxy getrennt vorzunehmen. Die Eignung eines Mikrocontrollers für den Datenendpunkt hängt stark von dem Anwendungszweck ab, dem der Datenendpunkt dienen soll. Anhand des Anwendungszwecks lässt sich ermitteln, welche Anforderungen hinsichtlich Rechenleistung, Speicherausstattung und integrierte Komponenten (z. B. Timer, A/D-Wandler, Kommunikationsschnittstellen) der Mikrocontroller erfüllen muss. Wichtige Aspekte bei der Auswahl eines Mikrocontrollers für den Proxy sind die Größe des Arbeitsspeichers bzw. die vorhandenen Möglichkeiten externe Speichermedien anzuschließen. Denn dadurch wird bestimmt,

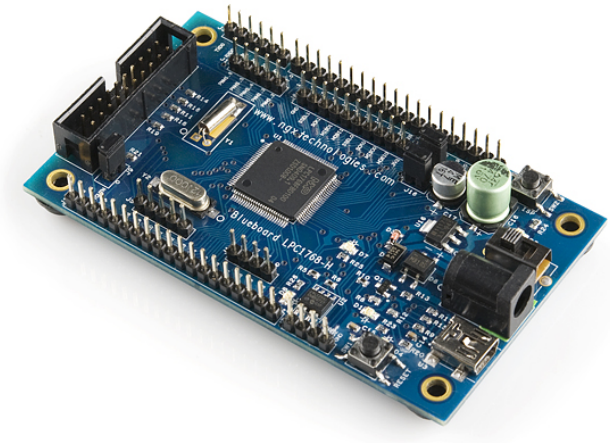

Abbildung 3.1: BlueBoard-LPC1768-H von NGX Technologies
(aus https://www.sparkfun.com/products/9931)

wie viele Ressourcen der Datenendpunkte der Proxy speichern kann. Nur wenn das konkrete Einsatzgebiet des Proxy bekannt ist, lässt sich die benötigte Speicherkapazität abschätzen. Wie bereits oben ausgeführt, fehlen derartige Randbedingungen in dieser Arbeit. Daher bestand große Wahlfreiheit bei der Auswahl eines Mikrocontrollers. Das einzige unabdingbare Kriterium war, dass der Mikrocontroller über die notwendigen Schnittstellen verfügen muss, um mit dem WLAN-Modul verbunden werden zu können.

3.1.2 BlueBoard-LPC1768-H

Es wurde das Development-Board *BlueBoard-LPC1768-H* der Firma *NGX Technologies* ausgewählt. Auf dem Board befindet sich der Cortex-M3-Mikrocontroller *LPC1768* von *NXP Semiconductors*. Der Chip ist mit 64 kB internem RAM und 512 kB ROM ausgestattet. Der Chip verfügt über zahlreiche Schnittstellen, darunter UART (viermal), I^2C (dreimal), SPI, USB (Device/Host/OTG) und Ethernet. Alle IO-Pins sind über Pfostenstecker leicht erreichbar. Der Mikrocontroller unterstützt Programmierung (Flashen) und Debuggen durch die JTAG-Schnittstelle. Um den Anschluss externer Debug-Hardware an die Debug-Schnittstelle des Cortex-M3 zu vereinfachen, befindet sich auf dem Development-Board ein 20-Pin-IDC-Stecker. Dies ist ein bei ARM-basierten Mikrocontrollern oft genutzter Steckertyp, um den Zugriff auf das Debug-System zu ermöglichen.

3.2 WLAN-Modul

3.2.1 Anforderungen

Die Industrie bietet zahlreiche Produkte an, um ein eingebettetes System mit WLAN-Funktionalität auszustatten. Bei der Auswahl eines geeigneten WLAN-Moduls wurde auf folgende Bewertungskriterien geachtet:

- Das WLAN-Modul soll einen geringen Energieverbrauch haben.
  - Der Stromverbrauch beim Senden und Empfangen soll gering sein.
  - Das WLAN-Modul soll in einen Energiesparmodus versetzt werden können, wenn keine Daten gesendet oder empfangen werden müssen.
  - Die Zeitspanne, die benötigt wird, um das WLAN-Modul aus dem Energiesparmodus in den normalen Sende-/Empfangsmodus zu versetzen, soll möglichst kurz sein.
- Das WLAN-Modul soll sich auf einfache Weise in ein eingebettetes System integrieren lassen.
  - Die Kommunikation zwischen dem Mikrocontroller und dem WLAN-Modul soll über eine im Embedded-Bereich verbreitete Schnittstelle erfolgen. Übliche Schnittstellen im Embedded-Umfeld sind UART, SPI, CAN.
- Das WLAN-Modul soll schnell einsetzbar für den Anwendungsentwickler sein.
  - Die Firmware des WLAN-Modul muss das WLAN-Protokoll implementieren.
  - Die Firmware des WLAN-Modul muss mit einem TCP/IP-Stack ausgestattet sein.

    oder
  - Das WLAN-Modul muss sich mit einem frei verfügbaren TCP/IP-Stack betreiben lassen.
  - Falls die Kommunikation zwischen Mikrocontroller und WLAN-Modul einen Treiber auf dem Mikrocontroller erfordert, müssen folgende Bedingungen gelten:
    * Der Treiber muss kostenneutral nutzbar sein, d.h. er muss entweder beim Kauf des WLAN-Moduls mitgeliefert werden oder er muss ohne Lizenzgebühren im Internet herunterladbar sein.
    * Der Treiber muss ohne Echtzeitbetriebssystem auf dem Mikrocontroller lauffähig sein.

Abbildung 3.2: RN-174 Development Board von Roving Networks (aus RN-1742012 (2012, S. 1))

3.2.2 RN-174 Development Board

Für diese Arbeit wurde das *RN-174 Development Board* mit dem darauf befindlichen *RN-171 WiFly-GSX Module* von *Roving Networks* ausgewählt. Mithilfe dieses Development Board kann ein Mikrocontrollersystem auf einfache Weise mit WLAN-Funtionalität ausgestattet werden. Das *RN-171 WiFly-GSX Module* besitzt einen WLAN-Transceiver nach Standard IEEE 802.11b/g und kann sowohl in Infrastruktur- als auch in Ad-hoc-Netzen eingesetzt werden. Das Modul zeichnet sich durch einen geringen Energieverbrauch aus und kann in einen Schlafmodus versetzt werden, wenn kein Netzwerkzugriff benötigt wird. Für den Anschluss an einen Mikrocontroller stehen wahlweise eine UART- und eine SPI-Schnittstelle zur Verfügung. Auf dem Mikrocontroller muss keine spezielle Software laufen (z. B. Treiber, Protokollstapel), da alle für die Anwendungsentwicklung benötigten Netzwerkdienste von der Firmware des Moduls implementiert werden. So ist insbesondere ein TCP/IP-Stack vorhanden. Um von einem Mikrocontroller aus mit der Software des Moduls zu kommunizieren, hat *Roving Networks* eine proprietäre Zeichen-basierte Kommandosprache entwickelt.

Um die Verwendung des *RN-171 WiFly-GSX Module* in Protoypen zu erleichtern, bietet *Roving Networks* das *RN-174 Development Board* an. Auf eine 51 mm × 28 mm großen Platine ist ein *RN-171 WiFly-GSX Module* gelötet. 4 LEDs geben Auskunft über den Betriebszustand des Moduls. Das *RN-171 WiFly-GSX Module* verfügt über keine integrierte Antenne, sondern lediglich über eine Anschlussmöglichkeit für eine externe Antenne. Aus diesem Grund ist auf dem *RN-174 Development Board* eine kleine Antenne vorhanden. Auf der Platine befinden sich Bohrlochreihen im Rastmaß 2 mm, in die Pfostensteckerleisten eingelötet werden können. Sobald die Platine mit diesen Leisten ausgerüstet worden ist, sind viele Eingangs-/Ausgangssignale des *RN-171 WiFly-GSX Module* per Steckverbindung erreichbar, u. a. Stromversorgung, UART RX, TX, CTS, RTS. Außerdem ist auf dem *RN-174 Development Board* ein Pegelwandler (UART/RS-232) und

eine zweireihige Pfostensteckerleiste (RS-232 RX, TX, CTS, RTS, GND) montiert. Dadurch kann das *RN-174 Development Board* auch direkt an die bei PCs gängige serielle Schnittstelle RS-232 angeschlossen werden.

3.2.3 Bedienung

Die Software, die auf dem angeschlossenen Mikrocontroller läuft, muss zur Steuerung des WLAN-Moduls die vom Hersteller vorgesehenen Schnittstellen nutzen. Da das Design der Software-Komponente, die mit dem WLAN-Modul kommuniziert, maßgeblich davon bestimmt ist, auf welche Art und Weise das WLAN-Modul bedient wird, wird im folgenden Abschnitt ein Überblick über das Bedienkonzept des *RN-174 Development Board* gegeben.[1] Alle diesbezüglich gemachten Angaben beruhen auf dem Manual (WiFly2012 2012) und auf den Erkenntnissen, die beim Umgang mit dem Modul gewonnen worden sind.

Die Bedienschnittstelle des WLAN-Moduls besteht aus zwei Teilen: Der weitaus größte Teil der Bedienhandlungen wird über ein Kommandozeilen-Interface (CLI) vorgenommen. Auf dieses Interface kann über UART bzw. RS-232 zugegriffen werden. Daher ist eine Steuerung des WLAN-Moduls per Kommandosprache sowohl von einem Mikrocontroller als auch von einem PC aus möglich. Außerdem ist die Kommandozeilen-Schnittstelle auch über eine Telnet-Verbindung aus der Ferne zugänglich. Der andere Teil der Bedienschnittstelle besteht aus mehreren Digital-Eingängen und -Ausgängen. Durch Setzen der Eingänge auf Eins oder Null können bestimmte Funktionen ausgelöst werden. Die Pegel der Ausgänge zeigen bestimmte Zustände des WLAN-Moduls an.

Für die Bedienung mithilfe des Kommandozeilen-Interface ist es wichtig zu wissen, dass sich das Modul grundsätzlich in einem von zwei Modi befindet: entweder im Kommandomodus oder im Datenmodus. Im Kommandomodus werden sämtliche Einstellungen vorgenommen und abgerufen. Dieser Modus arbeitet zeilenorientiert: Das WLAN-Modul signalisiert seine Eingabebereitschaft, indem es eine Eingabeaufforderung (Prompt) ausgibt.[2] Nun kann ein Kommando gemäß der Syntax der Kommandosprache eingegeben werden. Jedes Kommando wird mit einem Wagenrücklauf (carriage return, CR) abgeschlossen. Daraufhin wird das Kommando verarbeitet. Die erfolgreiche Verarbeitung wird vom Modul durch die Ausgabe von „AOK“ quittiert[3], ein ungültiges Kommando wird mit „ERR“ und einer Fehlermeldung beantwortet. Anschließend wird wieder die Eingabeaufforderung ausgegeben. Um den Kommandomodus zu verlassen, muss

1 Die Firmware der in dieser Arbeit benutzten WLAN-Module hat die Versionsnummer 2.32.

2 Das Aussehen der Eingabeaufforderung ist konfigurierbar. Standardwert ist die in spitze Klammern eingeschlossene Versionsnummer der Firmware, z. B. „<2.32>“.

3 Dies gilt laut WiFly2012 (2012, S. 7) für *die meisten* gültigen Kommandos. Im Manual sind keine Angaben zu finden, welche gültigen Kommandos nicht mit „AOK“ beantwortet werden.

das Kommando „exit“ gesendet werden. Das WLAN-Modul antwortet mit „EXIT“ und befindet sich von diesem Zeitpunkt an im Datenmodus.

Der Datenmodus dient dem Empfang und Versand von Daten über das WLAN-Netz. Dieser Modus funktioniert wie eine Datenquelle und eine Datensenke: Wenn das WLAN-Modul Daten über das Netzwerk empfängt, werden diese als Strom von Bytes ausgegeben. Bytes, die an das Kommandozeilen-Interface gesendet werden, werden über die WLAN-Schnittstelle verschickt. Um in den Kommandomodus zu wechseln, muss eine aus drei ASCI-Zeichen bestehende Escape-Sequenz an das Kommandozeilen-Interface gesendet werden.[4] Vor und nach der Escape-Sequenz darf innerhalb eines Zeitraums von 250 ms kein Byte gesendet werden. Andernfalls interpretiert das Modul die drei Zeichen nicht als Escape-Sequenz sondern behandelt sie als Nutzdaten und verschickt sie über das Netzwerk. Wird die Escape-Sequenz hingegen erkannt, antwortet das Modul mit „CMD“ und wechselt in den Kommandomodus. Wenn das Modul von einem menschlichen Benutzer bedient wird, ist eine Verzögerung von 2 mal 250 ms beim Wechsel in den Kommandomodus unkritisch. Erfolgt der Zugriff auf das Modul hingegen durch ein Computerprogramm, stellen Verzögerungen in einer derartigen Größenordnung eine erhebliche Performance-Einbuße dar. Offensichtlich hat auch der Hersteller des Moduls dieses Defizit erkannt und stellt mit der Firmware-Version 2.32 eine weitere Methode zur Verfügung, in den Kommandomodus zu wechseln. Diese Methode kann nur genutzt werden, wenn das Modul an einen Mikrocontroller angeschlossen ist, denn der Modus-Wechsel wird ausgelöst, indem ein Digitaleingang des Moduls auf Eins gesetzt wird.

Da das Senden und Empfangen von Daten über das WLAN-Netz zu den am meisten benutzten Anwendungsfällen gehört, wird auf die Fähigkeiten und Einschränkungen, die das Modul diesbezüglich aufweist, nachfolgend näher eingegangen.

Das Modul ist in der Lage, UDP- und TCP-Pakete zu verschicken. Zur Einstellung einer lokalen Portnummer steht nur ein Parameter zur Verfügung. Der Wert dieses Parameters legt fest, auf welchem Port UDP- und TCP-Pakete empfangen werden sowie von welchem Port aus Pakete gesendet werden. Dieser Umstand stellt eine Einschränkung der Nutzungsmöglichkeiten des Moduls dar: Mithilfe des WLAN-Moduls ist es zwar möglich, zeitgleich sowohl einen UDP als auch einem TCP basierten Dienst im Netzwerk anzubieten, jedoch müssen der UDP-Port und der TCP-Port die gleiche Nummer haben. Dies ist insbesondere dann von Nachteil, wenn beide Dienste auf sogenannten Well-Known-Ports horchen sollen. Auf dem Proxy, der in dieser Arbeit entwickelt wird, läuft sowohl ein HTTP-Server als auch ein CoAP-Server. In den Spezifikationen sind für beide Protokolle Standard-Ports vorgesehen. Es ist wünschenswert, wenn diese Portnummern verwendet werden könnten. Aufgrund der oben genannten Einschränkungen

4 Konfigurierbar, standardmäßig „$$$“.

wurde für beide Dienste der Port 80 gewählt. Somit ist sichergestellt, dass der HTTP-Server von einem Webbrowser ohne die explizite Angabe der Portnummer in der URL kontaktiert werden kann.

Soll vom WLAN-Modul ein UDP-Paket gesendet werden oder eine TCP-Verbindung aufgebaut werden, muss zunächst im Kommandomodus die IP-Adresse und der Port des Ziels eingestellt werden. Wenn ein UDP-Paket gesendet werden soll, kann anschließend in den Datenmodus gewechselt werden, um die Nutzdaten des Pakets an das Modul schicken zu können. Soll hingegen eine TCP-Verbindung aufgebaut werden, muss zunächst der Verbindungsaufbau initiiert werden. Dies geschieht wahlweise durch Senden des Befehls „open“ an das CLI oder indem der hierfür vorgesehene Digitaleingang des Moduls auf eins gesetzt wird. Das Modul gibt eine Rückmeldung über den erfolgreichen Verbindungsaufbau und wechselt selbstständig in den Datenmodus. Solange die Verbindung besteht, werden an das CLI gesendete Bytes zu TCP-Paketen zusammengefasst und übertragen.

3.2.4 Nachteile

Während der praktischen Beschäftigung mit dem WLAN-Modul traten einige Defizite der Bedien-Schnittstelle zutage. Verständlicherweise liegt es im Interesse des Herstellers eines Produktes, dessen Stärken hervorzuheben und nicht dessen Schwächen. Folgerichtig werden keine der nachfolgend aufgelisteten Mängel im Manual explizit erwähnt.

- Viele Informationen, die in Anwendungen oft benötigt werden, können mithilfe von Kommandos abgefragt werden, die mit „get“ beginnen. Anstatt zur Abfrage jeden Wertes ein eigenes Kommando bereitzustellen, gibt es nur Kommandos, die eine Kategorie von Werten abfragen. Beispielsweise liefert „get ip“ eine Reihe an Informationen, die mit IP-Adressen zu tun haben. Eine Anwendung, die einen bestimmten Wert benötigt (z. B. die IP-Adresse des WLAN-Moduls) muss den gelieferten Antwort-String nach den benötigten Informationen durchsuchen.

- Baut ein Remote-Host eine TCP-Verbindung zum WLAN-Modul auf und sendet Daten, werden über die Bedien-Schnittstelle nur der Aufbau und Abbau der Verbindung mitgeteilt sowie die Nutzdaten ausgegeben. Es gibt keine Möglichkeit, die IP-Adresse und den Port des Remote-Host abzufragen. Dieser Umstand ist erstaunlich, denn die IP-Adresse ist im IP-Header und die Portnummer im TCP-Header enthalten und beide Felder müssen vom Modul zur Verwaltung der TCP-Verbindung auch ausgewertet werden. Benötigt eine Anwendung die Adresse des Remote-Host, müssen die IP-Adresse und die Portnummer in den Nutzdaten des TCP-Pakets enthalten sein.

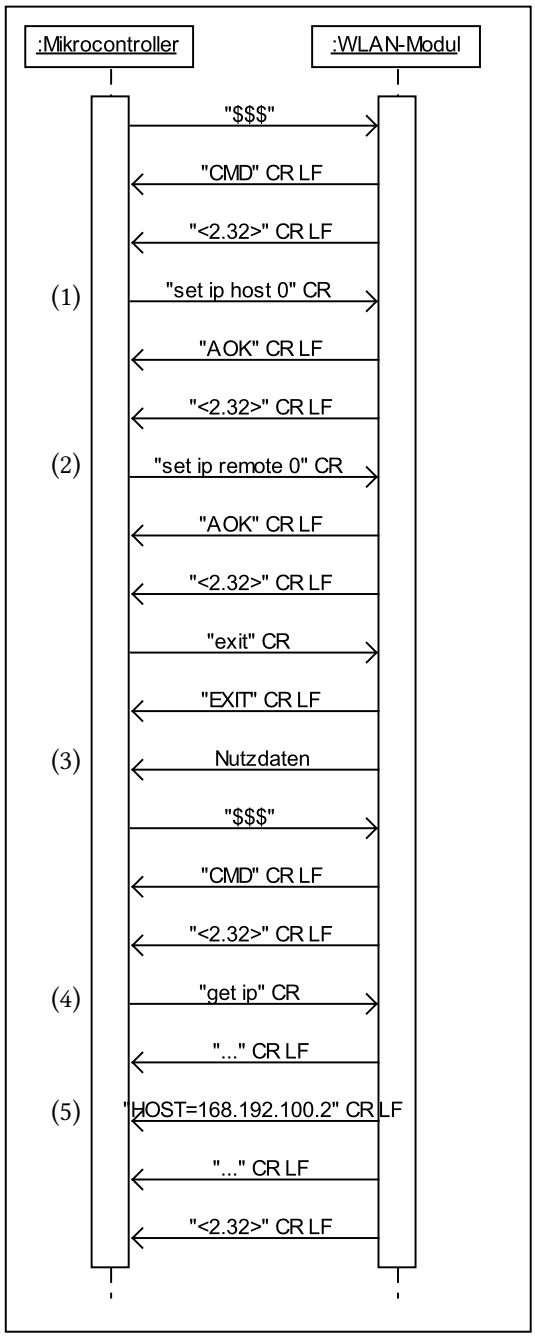

Abbildung 3.3: Sequenzdiagramm WLAN-Modul
Notation der Nachrichten in Angereicherter Backus-Naur-Form, siehe (Crocker und Overell 2008)

- Die Host-Adresse des Absenders eines empfangenen UDP-Pakets ist nur umständlich feststellbar. Voraussetzung ist, das die „UDP auto pairing“-Funktion des WLAN-Moduls aktiviert ist. Diese Funktion speichert die IP-Adresse und die Portnummer des Absenders, wenn ein UDP-Paket empfangen worden ist. Das Sequenzdiagramm in Abbildung 3.3 stellt die notwendigen Schritte dar, um die Remote-Host-Adresse zu ermitteln.

  (1, 2) Bevor ein UDP-Paket empfangen wird, müssen die internen Parameter, in denen die IP-Adresse und der Port des Remote-Host gespeichert werden, auf null gesetzt werden. Dies geschieht, indem die Kommandos (1) und (2) an das CLI gesendet werden.

  (3) Wenn das WLAN-Modul ein UDP-Paket empfängt, werden die Nutzdaten über das CLI ausgegeben.

  (4) Sobald das Paketende erkannt worden ist, kann die Host-Adresse des Absenders ermittelt werden. Das CLI bietet keine Kommandos, mit denen nur die IP-Adresse oder der Port abgefragt werden können. Daher muss das Kommando (4) verwendet werden.

  (5) Die Anwendung muss die gelieferte Antwort nach der Zeile durchsuchen, die mit „HOST“ beginnt, um die Remote-Host-Adresse zu ermitteln. Falls die Portnummer als Integer weiterverarbeitet werden soll, muss der als Zeichenkette vorliegende Wert noch konvertiert werden.

- Wenn das WLAN-Modul die Nutzdaten eines empfangenen UDP- oder TCP-Pakets über das CLI ausgibt, gibt es keine explizite Kennzeichnung des Anfangs und des Endes eines Pakets. Ein Server, der auf den Eingang von Nachrichten wartet, um dann entsprechend zu reagieren, ist daher nicht ohne Weiteres realisierbar. Von dieser Einschränkung ausgenommen sind Anwendungsschicht-Protokolle, die eine eindeutige Anfangs- und Endekennzeichnung der Nachrichten haben.

- Solange eine TCP-Verbindung besteht, können keine UDP-Pakete gesendet werden. Im Datenmodus werden alle an das CLI gesendeten Bytes über die TCP-Verbindung weitergeleitet. Erst nachdem die Verbindung beendet worden ist, können wieder UDP-Pakete gesendet werden.

- Wenn eine TCP-Verbindung besteht, gibt das CLI (sofern es sich im Datenmodus befindet) die Nutzdaten von empfangenen UDP- und TCP-Paketen aus. Die Anwendung, die die Daten entgegennimmt, kann jedoch nicht unterscheiden, ob die ausgegebenen Daten aus einem UDP- oder TCP-Paket stammt.

4 Entwicklung der Software für Datenendpunkt und Proxy

4.1 Dekomposition

Der folgende Abschnitt beschreibt die Dekomposition der Software-Systeme Datenendpunkt und Proxy in einzelne Komponenten. Es wird darauf eingegangen, welche Aufgaben die Komponenten erfüllen und welche Beziehungen zwischen den Komponenten bestehen. Legt man den Aufbau des verteilten Systems zur Messwerterfassung und die gewählte Hardware-Konfiguration zugrunde, lassen sich die benötigten Software-Komponenten leicht ermitteln. Die Namen der Komponenten sind zwecks leichterer Erkennbarkeit im Text in kursiver Schrift gesetzt.

4.1.1 Gemeinsame Komponenten

Der Mikrocontroller wird über die UART-Schnittstelle mit dem WLAN-Modul verbunden. Daher wird eine Komponente *UART-Treiber* benötigt. Wie im Abschnitt 3.2.3 beschrieben, stellt das Modul zur Kommunikation ein Command Line Interface zur Verfügung. Andere Komponenten, die die Funktionen des WLAN-Moduls nutzen, sollen keinen direkten Zugriff auf diese Schnittstelle haben, vielmehr soll der Zugriff über die Komponente *WLAN-Treiber* erfolgen. Diese zusätzliche Indirektionsstufe bietet mehrere Vorteile: Die aufwändige Nutzung der textbasierten Schnittstelle (i. d. R. Kommando senden, auf Antwort warten, Antwort-String interpretieren) wird nur einmal zentral im *WLAN-Treiber* implementiert. Der Treiber verbirgt die Komplexität und stellt anderen Komponenten, die die WLAN-Funktionalität nutzen, eine einfachere Schnittstelle zur Verfügung. Zudem wird der Austausch des WLAN-Moduls durch ein anderes Modell erleichtert: Es muss nur der *WLAN-Treiber* neu geschrieben werden, der Code der anderen Komponenten bleibt unverändert.

4.1.2 Komponenten des Proxy

Wie aus der Abbildung 2.1 hervorgeht, muss der Proxy zwei Netzwerkdienste anbieten: Zum einen einen HTTP-Server, der Anfragen der Benutzerendpunkte beantwortet, zum anderen einen

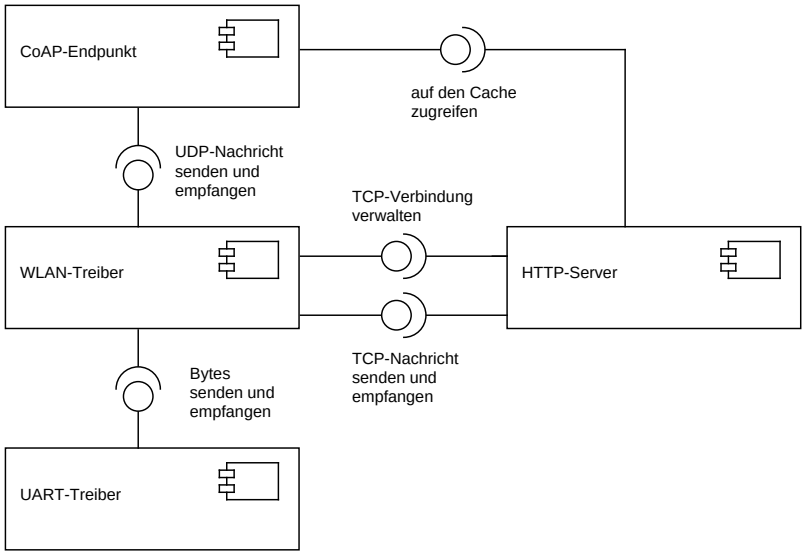

Abbildung 4.1: Komponentendiagramm Proxy

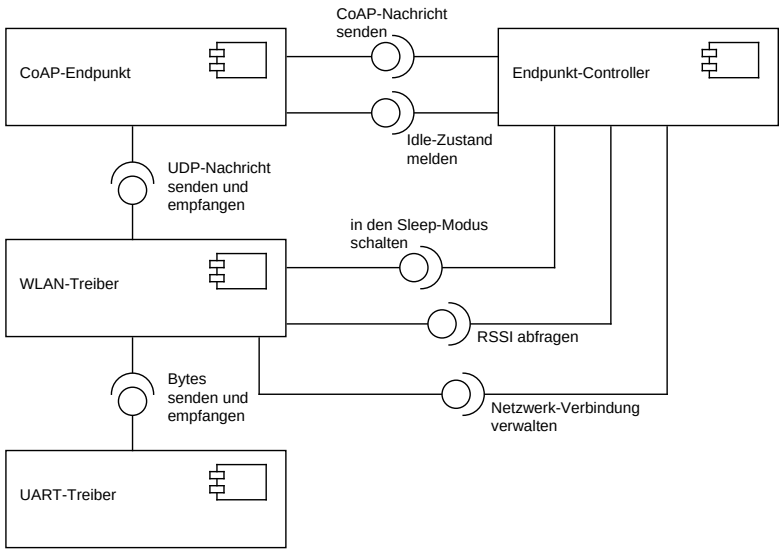

Abbildung 4.2: Komponentendiagramm Datenendpunkt

CoAP-Server, der Datenendpunkten die Nutzung des Proxy als Cache für ihre Ressourcen ermöglicht. Die Dienste werden durch die Komponenten *HTTP-Server* und *CoAP-Endpunkt* realisiert. Beide Komponenten benutzen für den Netzwerkzugriff die Dienste des *WLAN-Treibers*. Der *CoAP-Endpunkt* verwaltet einen Cache, in dem die Ressourcen der Datenendpunkte gespeichert werden. Der *HTTP-Server* greift auf den Cache zu, um Antworten auf die GET-Requests der Benutzerendpunkte zu generieren.

4.1.3 Komponenten des Datenendpunktes

Der Datenendpunkt benötigt eine Komponente *CoAP-Endpunkt*, die die Kommunikation mit dem Proxy abwickelt. Der *CoAP-Endpunkt* benutzt den *WLAN-Treiber* für den Netzwerkzugriff. Darüber hinaus hat ein Datenendpunkt noch eine oder mehrere applikationsspezifische Komponenten, die dem eigentlichen Anwendungszweck dienen. Von der Anwendung hängt maßgeblich ab, welche Ressourcen der Datenendpunkt überhaupt hat. Damit der Datenendpunkt, der in dieser Arbeit entwickelt wird, zumindest eine sinnvolle Ressource hat, die er beim Proxy zwischenspeichern kann, ist die Komponente *Endpunkt-Controller* vorhanden. Das CLI des WLAN-Moduls liefert nach Eingabe des Befehls „rssi“ eine Kennzahl, die die Feldstärke der empfangenen Beacon Frames angibt. Dieser Wert wird als Received Signal Strength Indication (RSSI) bezeichnet. Die Komponente *Endpunkt-Controller* ruft den RSSI-Wert periodisch beim *WLAN-Treiber* ab und stellt ihn in Form einer Ressource zur Verfügung. Nach jedem Abruf wird eine Repräsentation der Ressource im Plain-Text-Format an den Proxy übermittelt.

4.2 Auswahl eines Modells zur Zuteilung der Rechenzeit

Nachdem die Software-Komponenten ermittelt worden sind, aus denen Datenendpunkt und Proxy bestehen, kann entschieden werden, ob der Einsatz eines Realtime-Betriebssystems sinnvoll ist. Da diese Entscheidung weitreichende Folgen für die Implementierung hat, soll etwas weiter ausgeholt werden.

4.2.1 Eigenschaften der Software-Systeme Datenendpunkt und Proxy

Nachfolgend werden einige grundsätzliche Aussagen zu den Eigenschaften der Systemkomponenten Datenendpunkt und Proxy gemacht. Viele Aussagen werden an einem Beispiel verdeutlicht. Zwecks Übersichtlichkeit sind die Beispiele in einem separaten Abschnitt angeordnet. Auf das zu einer Aussage passende Beispiel wird mithilfe einer in Klammern stehenden Zahl verwiesen.

Eigenschaften

Datenendpunkt und Proxy sind ereignisgesteuerte Systeme. Das bedeutet, dass sie kontinuierlich auf externe oder interne Ereignisse warten (1), beim Eintreten eines Ereignisses entsprechend reagieren und anschließend wieder auf neue Ereignisse warten (2). Die Ereignisse treten vollständig asynchron zum Programmablauf auf (3) und die Reihenfolge, in der Ereignisse auftreten ist nicht vorhersehbar (4). Tritt ein Ereignis ein, werden nur bestimmte Software-Komponenten des Systems reagieren, wohingegen das Eintreffen dieses Ereignisses für die anderen Komponenten irrelevant ist und zu keiner Reaktion führt (5). Ereignisse lassen sich in Kategorien einordnen. Die Reaktion des Systems auf ein Ereignis wird typischerweise sowohl von der Art des Ereignisses als auch vom Systemzustand abhängen (6). Da es wesentlich für das Verständnis ereignisgesteuerter Systeme ist, sei es hier nochmals explizit verdeutlicht: Das System muss seinen aktuellen Zustand in geeigneter Weise intern darstellen. Der Systemzustand kann nicht anhand des empfangenen Ereignisses ermittelt werden. Jedoch kann das Auftreten eines Ereignisses zu einer Änderung des Systemzustands führen. Ereignisse gleichen Typs können in unterschiedlichen Systemzuständen unterschiedliche Reaktionen hervorrufen. Die interne Funktionsweise sowie der aktuelle interne Zustand einer Komponente sind für die anderen Komponenten nicht von Interesse. Im Sinne eines sauberen Software-Designs sollten die Interna einer Komponente den anderen Komponenten nicht zugänglich sein (Kapselung). Stattdessen werden die Dienste einer Komponente in Form einer wohl definierten Schnittstelle zugänglich gemacht (7). Die einzelnen Komponenten agieren unabhängig voneinander, was insbesondere bedeutet, dass nicht vorhersagbar ist, wann welche Komponente Rechenzeit benötigt (8). Die Entwicklung einer Komponente wird erheblich vereinfacht, wenn die Zuteilung von Rechenzeit an die Komponenten durch eine übergeordnete Instanz kontrolliert wird. Der Anwendungsentwickler wird von der komplexen und fehlerträchtigen Aufgabe entbunden, den konkurrierenden Zugriff auf die gemeinsam genutzte Ressource CPU korrekt handhaben zu müssen.

Beispiele

(1) Ereignisse sind beispielsweise das Ablaufen eines Timers, der Empfang eines UDP-Pakets oder das Auftreten eines Interrupts, wenn neue Daten im UART-Empfangspuffer bereitstehen. Nachrichten, die zwischen den Komponenten ausgetauscht werden, sind ebenfalls Ereignisse.

(2) Wenn der Proxy eine HTTP-Anfrage empfängt, muss diese interpretiert und gemäß den enthaltenen Informationen (Anfragemethode, Anfrage-URI, Header) eine Antwort generiert und gesendet werden.

(3) Während ein Datenendpunkt gerade das Acknowledgement einer zuvor gesendeten CoAP-Nachricht verarbeitet, läuft ein Timer ab, der eine Messung triggert.

(4) Nachdem ein Datenendpunkt eine CoAP-Nachricht vom Typ Confirmable gesendet hat, ist nicht vorhersehbar, ob als Nächstes die Bestätigung eintrifft, der Timer abläuft, der eine Neuübertragung der Nachricht auslöst, oder das WLAN-Modul die Unterbrechung der Netzwerkverbindung meldet. Die Liste der möglichen Ereignisse, die in unbekannter Reihenfolge eintreten können, lässt sich noch erweitern.

(5) Wenn die serielle Schnittstelle des Mikrocontrollers, an den das WLAN-Modul angeschlossen ist, einen Interrupt auslöst, ist dieses Ereignis ausschließlich für die Komponente UART-Treiber relevant. Der UART-Treiber muss den Grund für die Auslösung des Interrupt feststellen (z. B. Sendepuffer leer) und entsprechend reagieren.

(6) Angenommen der UART-Treiber hat ein neues Datum aus dem Empfangspuffer ausgelesen und sendet daraufhin eine entsprechende Nachricht (Ereignis) an den WLAN-Treiber. Aufgrund des Typs des Ereignisses ist der WLAN-Treiber in der Lage, anders zu reagieren als wenn er beispielsweise einen Auftrag (in Form eines Ereignisses) vom HTTP-Server erhält, eine TCP-Nachricht zu senden. Die Tatsache, dass das WLAN-Modul ein Byte über die serielle Schnittstelle gesendet hat, ist allerdings für sich genommen noch nicht ausreichend, um die korrekt Reaktion des WLAN-Treibers zu bestimmen: Der WLAN-Treiber muss zusätzlich den Kontext berücksichtigen, d. h. er muss wissen, ob sich das WLAN-Modul im Kommando- oder im Daten-Modus befindet, ob aktuell eine TCP-Verbindung besteht usw.

(7) Der WLAN-Treiber muss zur Kommunikation mit dem WLAN-Modul dessen aktuellen Zustand und Zustandsübergänge verwalten. Die Komponenten HTTP-Server und CoAP-Server nutzen die vom WLAN-Treiber bereitgestellten Dienste zum Senden und Empfangen von UDP- und TCP-Daten. Die interne Funktionsweise des WLAN-Treibers sollte vor beiden Komponenten verborgen werden.

(8) Es sei angenommen, dass der Benutzerendpunkt eine Komponente zur Durchführung von Messungen hat. Während diese Komponente zwecks Ermittlung eines neuen Messwerts rechnet, empfängt das WLAN-Modul eine CoAP-Nachricht, was zur Folge hat, dass auch die Komponente CoAP-Server die CPU benötigt. Da beide Komponenten voneinander unabhängig sind, wissen sie nicht, wenn sie zeitgleich um Rechenzeit konkurrieren.

4.2.2 Etablierte Modelle

Basierend auf den vorhergehenden Ausführungen wird nun evaluiert, welches der etablierten Modelle zum Management der CPU-Zeit für die Systemkomponenten Datenendpunkt und Proxy geeignet ist.

Sequentielles System

Das einfachste und jedem Software-Entwickler vertraute Modell zur Zuteilung von Rechenzeit ist die klassische Sequentielle Programmierung. Im Bereich der Programmierung von Mikrocontrollern wird dieses Modell in Form der Vordergrund/Hintergrund-Architektur (auch bekannt unter den Begriffen Super-Loop und main+ISR) realisiert. Im Hauptprogramm (in der main-Funktion) werden in einer Endlosschleife einzelne Funktionen aufgerufen (die ggf. ihrerseits weitere Funktionen aufrufen usw.). Die CPU wird während der gesamten Laufzeit des Systems vom Hauptprogramm kontrolliert, es sei denn es wird eine Interrupt Service Routine (ISR) aufgerufen. Dann wird das Hauptprogramm für die Dauer der Ausführung der ISR unterbrochen und anschließend fortgesetzt. Wenn innerhalb des Hauptprogramms auf ein bestimmtes Ereignis gewartet wird, geschieht dies durch Busy Waiting. Hier zeigt sich der größte Nachteil des Super-Loop-Modells: Solange im Hauptprogramm auf ein Ereignis gewartet wird, ist das gesamte System (mit Ausnahme der Interrupt-Service-Routinen) nicht in der Lage auf andere eintreffende Ereignisse zu reagieren. Da Datenendpunkt und Proxy jedoch den größten Teil ihrer Laufzeit auf das Eintreffen von Ereignissen warten und zugleich die zeitliche Reihenfolge, in der die Ereignisse eintreffen, nicht vorhersehbar ist, scheidet die Sequentielle Programmierung als CPU-Management-Modell aus.

Klassisches Multitasking-System

Der naheliegende Schritt, um das Problem der mangelnden Reaktionsfähigkeit auf Ereignisse zu lösen, ist der Einsatz eines herkömmlichen Multitasking-Realtime-Betriebsystems. Durch ein Multitasking-System können mehrere Threads/Tasks auf einem Ein-Prozessor-System quasi parallel ausgeführt werden, indem den einzelnen Tasks in kurzen Abständen Rechenzeit zugeteilt und wieder entzogen wird. Wenn ein Task eine Aufgabe über die gesamte Programmlaufzeit ausführen soll, wird er üblicherweise in Form einer Endlosschleife realisiert. Konzeptionell kann daher ein Multitasking-System als die natürliche Erweiterung eines sequentiellen Systems betrachtet werden: Letzteres besteht aus einem sequentiellen Task und ISRs, wohingegen erstgenanntes mehrere Tasks parallel und ISRs ausführen kann. Dieses Konzept der Zuteilung von Rechenzeit passt erheblich besser zu den ereignisgesteuerten Systemen Datenendpunkt und Proxy: Diejenigen

Software-Komponenten, die parallel auf das Eintreffen von Ereignissen warten, können durch eigenständige Tasks implementiert werden. Wartet eine Komponente im Busy-Waiting-Stil auf ein Ereignis, sind die anderen Komponenten weiterhin reaktionsfähig. Einen Nachteil, der bereits vom sequentiellen System bekannt ist, ist auch beim klassischen Multitasking-System vorhanden: Ein Task, der auf ein bestimmtes Ereignis wartet, ist blockiert und nicht in der Lage, auf andere Ereignisse zu reagieren.

Klassisches ereignisgesteuertes System

Eine weitere Kategorie von Modellen zur Zuteilung von Rechenzeit sind klassische ereignisgesteuerte Systeme. Grundlage solcher Systeme ist das Ereignis-Aktion-Paradigma, welches besagt, dass jedem Ereignis eine passende Aktion zugeordnet werden kann. Die Zuordnung erfolgt nicht dynamisch zur Laufzeit des Programms, sondern wird statisch in der Design-Phase festgelegt. Alle Ereignisse, die vorkommen können, werden in Kategorien eingeordnet und allen Ereignissen derselben Kategorie wird eine Aktion (in Form einer Event-Handler-Funktion) zugewiesen. Die Reihenfolge, in der Ereignisse eintreten, bestimmt die Reihenfolge, in der die zugehörigen Aktionen ausgeführt werden. Zur Entwicklung von Software-Systemen, die auf dem Ereignis-Aktion-Paradigma beruhen, stehen fertige Frameworks zur Verfügung. Das Framework ist für den Empfang und die Pufferung der Ereignisse und den Aufruf der entsprechenden Handler-Funktion zuständig. Dem Applikationsentwickler fällt die Aufgabe zu, die Ereignisse zu definieren, die Ereignisse an das Framework zu senden und die Event-Handler zu implementieren. Das Framework teilt abhängig vom aufgetretenen Ereignis der zugehörigen Handler-Funktion die Rechenzeit zu. Daher ist ein solches System im Gegensatz zu einem sequentiellen System oder einem Multitasking-System in der Lage, auf in beliebiger Reihenfolge eintreffende Ereignisse in geeigneter Weise zu reagieren. Doch auch das klassische ereignisgesteuerte Modell hat einen Mangel: Es bietet keine Unterstützung, den aktuellen Systemzustand bei der Reaktion auf ein Ereignis zu berücksichtigen. Es sei nochmals auf Beispiel (6) verwiesen um zu verdeutlichen, dass oftmals die Reaktion des Systems sowohl von der Art des Ereignisses als auch vom Systemzustand abhängt.

4.2.3 Active Objects Computing Model

Keines der etablierten Modelle zur Zuteilung der Rechenzeit erfüllt die Anforderungen von Datenendpunkt und Proxy wirklich gut. Alle drei Modelle haben ihre spezifischen Nachteile. Dahingegen ist das Active Objects Computing Model sehr gut geeignet, ereignisgesteuerte Systeme zu modellieren. Da Zustandsdiagramme ein wesentlicher Bestandteil des Modells sind,

wird zunächst eine kurze Einführung in diese Diagrammart der Unified Modeling Language (UML) gegeben. Anschließend wird das Active Objects Computing Model erklärt, bevor dann ein Open-Source-Framework vorgestellt wird, mit dem aktive Objekte in der Programmiersprache C realisiert werden können.

UML-Zustandsdiagramme

Zustandsautomaten sind eine etablierte Methode, das Verhalten ereignisgesteuerter Systeme zu modellieren, denn sie ermöglichen es, die Reaktion des Systems auf ein Ereignis sowohl von der Art des Ereignisses als auch vom Systemzustand abhängig zu machen. Die UML bietet mit den UML-Zustandsdiagrammen eine ausdrucksstarke Möglichkeit, Zustandsautomaten grafisch darzustellen. Zudem erweitern UML-Zustandsdiagramme die klassischen Zustandsautomaten um interessante Aspekte. Hierzu zählen u. a. Entry- und Exit-Aktionen, interne Transitionen, Wächterausdrücke, hierarchisch verschachtelte Zustände, Pseudozustände, Zurückstellen von Ereignissen und Zeitereignisse. Dies alles sind leistungsfähige Mechanismen, die insbesondere hilfreich sind, die Komplexität des Modells zu reduzieren. Bei der Modellierung der Software-Komponenten des Datenendpunktes und des Proxy wird intensiv von den Möglichkeiten Gebrauch gemacht, die UML-Zustandsautomaten zur Verfügung stellen. Eine detaillierte Beschreibung der Notation würde jedoch den Rahmen dieser Arbeit sprengen, in (Samek 2009, S. 55 ff. und S. 688 f.) lässt sich eine sehr gute Einführung finden.

Aktive Objekte

Das Active Objects Computing Model vereint die Vorteile, die klassische Multitasking-Systeme, klassische ereignisgesteuerte Systeme und UML-Zustandsdiagramme aufweisen, in einem Modell. Kern des Modells sind unabhängig voneinander agierende Software-Objekte, die auf asynchrone Weise miteinander kommunizieren, indem sie Nachrichten verschicken. Die UML nennt diese Objekte aktive Objekte und legt diesbezüglich fest: Ein aktives Objekt hat einen eigenen Ausführungsthread, die Kommunikation zwischen aktiven Objekten beruht auf dem Austausch von Ereignissen und das Verhalten eines aktiven Objekts wird durch ein UML-Zustandsdiagramm beschrieben. In (Samek 2009, S. 266) wird es kompakt und zutreffend beschrieben:

> Active object = (thread of control + event queue + state machine)

Anhand der Abbildung 4.3 soll das Modell etwas genauer erläutert werden. Ein aktives Objekt wird durch ein Rechteck mit fetter Umrandung dargestellt (1). Das stilisierte Zustandsdiagramm im Inneren des Rechtecks weist auf die Modellierung des Verhaltens durch einen Zustandsautomaten hin. Das Symbol (2) deutet an, dass das aktive Objekt seinen eigenen Ausführungsthread

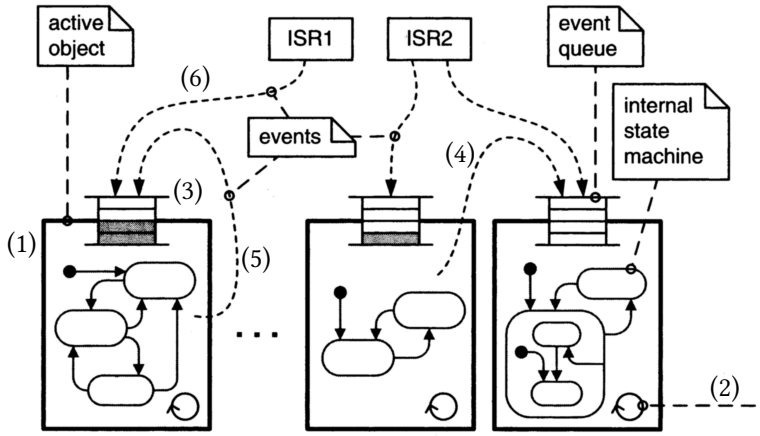

Abbildung 4.3: System mit aktiven Objekten

besitzt. Die fette Umrandung soll die strikte Kapselung des aktiven Objekts verdeutlichen: Die interne Struktur ist für andere Objekte nicht sichtbar. Ein aktives Objekt teilt keine Daten mit anderen Objekten, allgemeiner formuliert: In einem aus aktiven Objekten bestehenden System gibt es keine globalen Daten. Der Austausch von Ereignissen ist der einzige zur Verfügung stehende Mechanismus, zwischen aktiven Objekten zu kommunizieren (Daten auszutauschen). Jedes aktive Objekt besitzt eine Warteschlange (3), damit es Ereignisse empfangen kann. Ereignisse können von einem aktiven Objekt an ein anderes (4), von einem aktiven Objekt an sich selbst (5) oder von einer anderen Software-Komponente (z. B. von einer Interrupt Service Routine) an ein aktives Objekt (6) geschickt werden. Auch der Versand eines Ereignisses an mehrere Empfänger ist möglich. Das Senden von Ereignissen ist stets ein asynchroner Vorgang: Ein Objekt schickt ein Ereignis an eine Ereigniswarteschlange, wartet anschließend jedoch nicht, bis der/die Empfänger das Ereignis verarbeitet haben, sondern macht unmittelbar in seinem Ausführungskontext weiter.

Quantum Platform

Quantum Platform (QP) ist ein Open-Source-Framework, mit dem durch aktive Objekte modellierte Systeme implementiert werden können. Es stehen Versionen der Software in C und C++ zur Verfügung. Zahlreiche Portierungen sorgen dafür, dass QP sowohl direkt auf einem Prozessor („bare-metal“) als auch in Verbindung mit einem Betriebssystem eingesetzt werden kann. Es existieren u. a. Portierungen für ARM Cortex-M3 und für Linux. Aufgrund des modularen Aufbaus des Frameworks bleibt der applikationsspezifische Code (d. h. die Programmierung der Zustandsautomaten) immer gleich. Somit konnte die Software für den Datenendpunkt und

den Proxy zunächst auf einem Linux-PC entwickelt werden, was die Fehlersuche dank besserer Debug-Möglichkeiten als auf einem Mikrocontroller erleichterte.

4.3 Plattformunabhängigkeit

Bei der Implementierung eines Systems für eine PC-Plattform (z. B. Linux, Windows) kann der Software-Entwickler auf leistungsfähige Debug-Werkzeuge zurückgreifen. Für die Fehlersuche bei einem System, das auf einem Mikrocontroller läuft, muss der Programmierer hingegen einige Einschränkungen in Kauf nehmen. Hierzu zählt unter anderem, dass die Anzahl der Breakpoints beschränkt ist und printf-Ausgaben i. d. R. nicht möglich sind. Wie in Abschnitt 3.2.3 erläutert, kann das WLAN-Modul wahlweise an einen Mikrocontroller oder an einen PC angeschlossen werden. Da ein Großteil der Software von Datenendpunkt und Proxy plattformunabhängig ist, wird die Software zunächst für eine Linux-Plattform entwickelt und später auf die Cortex-M3-Plattform portiert. Die plattformabhängigen Teile des Codes werden in separaten Komponenten mit einheitlichen Schnittstellen gekapselt. Printf-Ausgaben können mithilfe von Compilerschaltern und define-Präprozessordirektiven ein- oder ausgeschaltet werden.

4.4 Beschreibung der Software-Komponenten

Nachfolgend werden die Software-Komponenten des Datenendpunktes und des Proxy beschrieben. Die abgebildeten Zustandsdiagramme sollen einen Eindruck vermitteln, wie das Verhalten der Komponenten modelliert worden ist. Aufgrund des nur begrenzt zur Verfügung stehenden Platzes konnten viele Details in den Diagrammen nicht dargestellt werden.

4.4.1 UART-Treiber

Der *UART-Treiber* ermöglicht anderen Komponenten die Benutzung der seriellen Schnittstelle. Der Zugriff auf die serielle Schnittstelle ist zwangsläufig plattformabhängig, daher wird Varianten des *UART-Treibers* für Linux und Cortex-M3 geben. Samek (2009, S. 461) empfiehlt, den plattformabhängigen Code der Anwendung in einem sog. Board Support Package (BSP) zu bündeln. Ein BSP besteht aus einer plattformunabhängigen Schnittstelle (in Form einer h-Datei) und der plattformspezifischen Implementierung (c-Datei). Der *UART-Treiber* erfüllt zwei Aufgaben. Zum einen nimmt er von anderen Komponenten Byte-Folgen an und sendet sie an die serielle Schnittstelle. Zum anderen leitet er Bytes, die über die serielle Schnittstelle empfangen werden, an andere Komponenten weiter. Der *UART-Treiber* für den Cortex-M3 arbeitet mit Sende- und Empfangs-Interrupts. Zur Programmierung des LPC1768 wurde die *LPC1700CMSIS Standard*

Peripheral Firmware Library von NXP eingesetzt. Die Schnittstelle der Bibliothek wird durch den *Cortex Microcontroller Software Interface Standard* (CMSIS) definiert. Dadurch wird der Austausch des Mikrocontrollers durch eine andere Cortex-M3-MCU erleichtert (Yiu 2010, S. 164 f- f. und S. 439 ff.). Die Linux-Version arbeitet mit den elementaren E/A-Funktionen nach dem POSIX-Standard (open, read, write) und der select-Funktion.

4.4.2 WLAN-Treiber

Der *WLAN-Treiber* wird als aktives Objekt modelliert. Um das WLAN-Modul korrekt bedienen zu können, muss der Zustand des *WLAN-Treibers* zu jedem Zeitpunkt den Zustand des WLAN-Moduls widerspiegeln. Zu diesem Zweck ist ein relativ komplexer Zustandsautomat nötig.

Wenn der *WLAN-Treiber* zur Bedienung des WLAN-Moduls ausschließlich dessen CLI nutzt, kann er vollständig plattformunabhängig implementiert werden. Das CLI ist über die serielle Schnittstelle zugänglich, d. h. der *WLAN-Treiber* nutzt die Komponente *UART-Treiber*. Leider ist die Bedienung des Moduls ausschließlich über das CLI mit einigen Mängeln behaftet, die vermieden werden können, wenn zusätzlich auch die Digital-Ein- und Ausgänge des Moduls zur Steuerung genutzt werden. Dann kann der *WLAN-Treiber* jedoch nicht mehr plattformunabhängig sein: Weil auf einem gewöhnlichen PC keine Ansteuerungsmöglichkeit für digitale Ein- und Ausgänge vorhanden ist, muss es dann eine Version des *WLAN-Treibers* geben, die die komplette Bedienschnittstelle des WLAN-Moduls nutzt und eine, die nur das CLI verwendet. Die erstgenannte Variante kann nur auf MCU-Plattformen zum Einsatz kommen, wohingegen die letztgenannte für PCs und Mikrocontroller nutzbar ist. Im Rahmen dieser Arbeit entsteht zunächst eine universell nutzbare Version des *WLAN-Treibers*.

4.4.3 CoAP-Endpunkt

Die Komponente *CoAP-Endpunkt* ist für das Versenden und Empfangen von CoAP-Nachrichten zuständig. Zu den wesentlichen Fähigkeiten, die implementiert werden müssen, gehören das Kodieren und Dekodieren von Nachrichten. Des Weiteren müssen empfangene Nachrichten erkannt werden, die ein fehlerhaftes Format haben oder die Optionen beinhalten, die der *CoAP-Endpunkt* nicht verarbeiten kann. Auf solche Nachrichten muss entsprechend der CoAP-Spezifikation reagiert werden. Die korrekte Reaktion ist oftmals vom Typ der CoAP-Nachricht abhängig: So werden derartige Nachrichten vom Typ Non-Confirmable oftmals ignoriert, wohingegen Nachrichten vom Typ Confirmable mit einer Response-Nachricht zurückgewiesen werden, die einen Fehlercode sowie optional eine für Menschen lesbare Fehlerbeschreibung enthält. Dieses

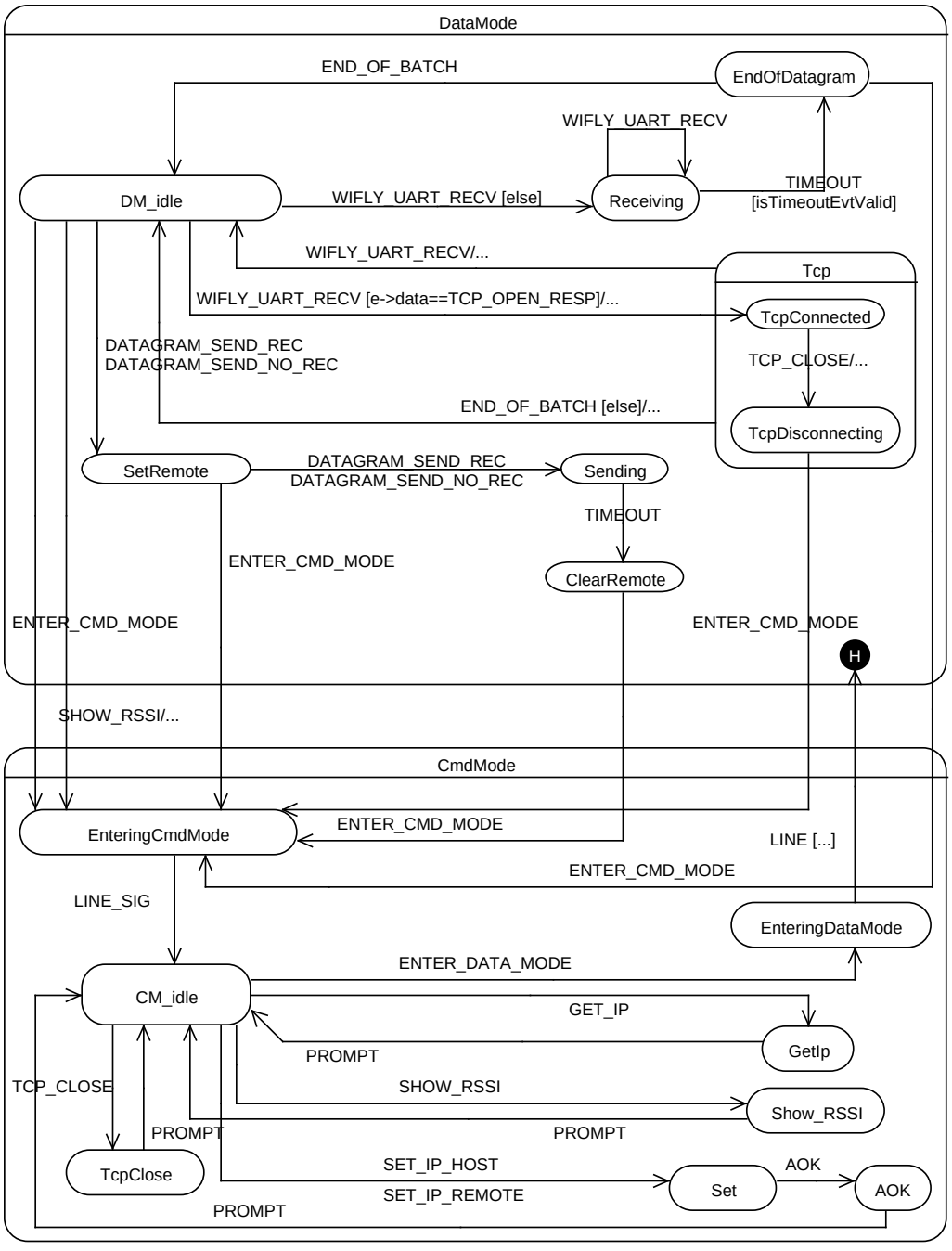

Abbildung 4.4: Zustandsdiagramm WLAN-Treiber

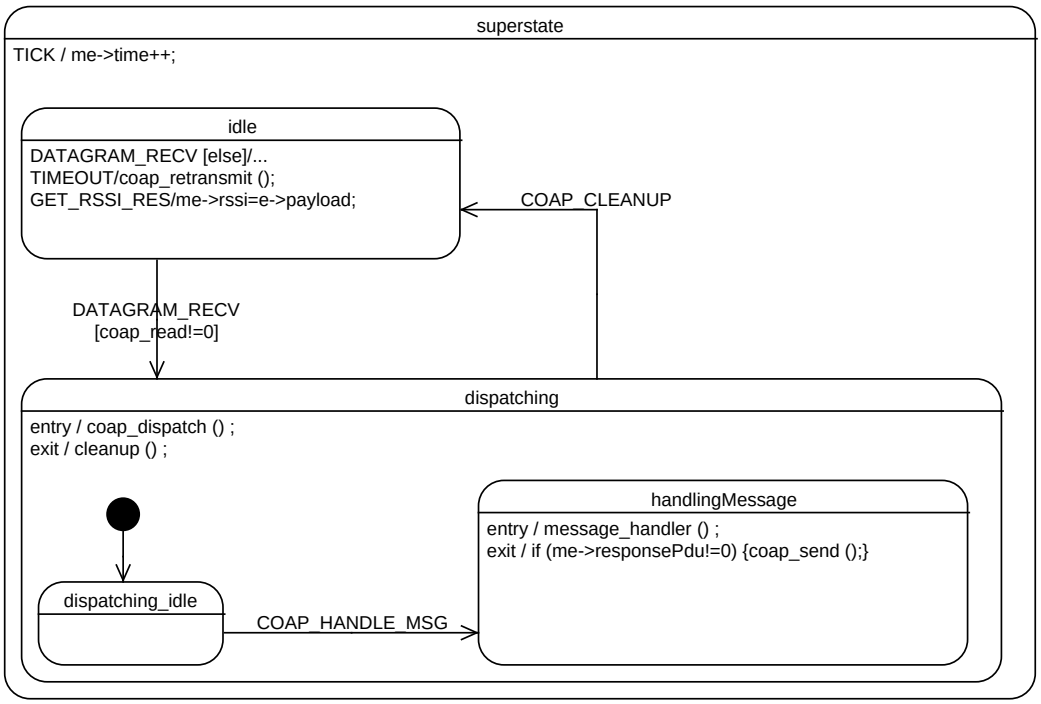

Abbildung 4.5: Zustandsdiagramm CoAP-Endpunkt

„Fehlermanagement“ macht einen nicht unerheblichen Teil der Implementierungsarbeit für den *CoAP-Endpunkt* aus.

Wie in Abschnitt 2.4.3 bereits erläutert, können mit CoAP sowohl Nachrichten verschickt werden, die vom Empfänger bestätigt werden müssen, als auch solche, die nicht bestätigt werden müssen. Daher muss im Rahmen der Entwicklung des Datenendpunktes die Frage gestellt werden, ob das Anmelden, Aktualisieren und Löschen von Ressourcen beim Proxy zuverlässig oder unzuverlässig sein soll. Für eine unzuverlässige Kommunikation zwischen Datenendpunkt und Proxy spricht, dass die Software-Komponente *CoAP-Endpunkt* des Datenendpunktes in diesem Fall den Versand von Confirmable-Nachrichten nicht unterstützen muss und somit der Implementierungsaufwand geringer ist. Andererseits kann nur durch den Versand von Nachrichten des Typs Confirmable sichergestellt werden, dass der Proxy zu jedem Zeitpunkt eine gültige Kopie der Ressourcen des Datenendpunktes besitzt. Da diese Forderung höher zu gewichten ist, muss der *CoAP-Endpunkt* einen Mechanismus zum Versand von Confirmable-Nachrichten implementieren. Der *CoAP-Endpunkt* darf Confirmable-Nachrichten anders als Non-Comfirmable-Nachrichten nicht unmittelbar nach dem Abschicken löschen, sondern muss sie solange speichern, bis der Empfänger den Eingang der Nachricht bestätigt hat. Falls innerhalb der vorgesehenen Wartezeit keine Bestätigung eingeht, muss die Nachricht neu übertragen werden. Die Wartezeit vergrößert sich nach jeder Neuübertragung. Nachdem eine im *CoAP-Endpunkt* eingestellte maximale Anzahl an Übertragungen erreicht worden ist, wird die Nachricht verworfen.

Um den *CoAP-Endpunkt* im zeitlichen Rahmen dieser Bachelorarbeit realisieren zu können, ist nach bereits vorhandenen Lösungen gesucht worden. Olaf Bergmann stellt mit libcoap Version 0.1.9 eine Implementierung des CoAP-Protokolls für PC-Betriebssysteme (Linux, Windows) bereit, die dank GNU General Public License frei genutzt werden kann. Bestandteil der Implementierung sind u. a. ein CoAP-Client und ein CoAP-Server, die von der Kommandozeile aus bedient werden. Im Quellcode von libcoap werden an sehr vielen Stellen Betriebssystem-Funktionen aufgerufen, die auf Mikrocontroller-Systemen nicht (bzw. nicht ohne Weiteres) zur Verfügung stehen, bzw. deren Verwendung problematisch ist. Im Einzelnen handelt es sich hierbei um die Funktionen der Socket-API, um die Funktionen der dynamischen Speicherverwaltung (malloc, free) sowie um die Funktionen printf und select. Insgesamt stellt Bergmanns Implementierung eines CoAP-Servers jedoch ein brauchbares Grundgerüst für die Programmierung des CoAP-Endpunktes dar, wenngleich Anpassungsarbeiten in erheblichem Umfang vonnöten sind.

Die Komponente *CoAP-Endpunkt* des Datenendpunktes und die des Proxy unterscheiden sich in einigen Details. Diesen Unterschieden wird durch bedingte Kompilierung (define-Anweisungen) Rechnung getragen, damit trotz der Unterschiede dieselbe Code-Basis verwendet werden kann.

Die wichtigsten Unterschiede seien nachfolgend kurz erwähnt. Der *CoAP-Endpunkt* des Datenendpunktes ist sowohl Client als auch Server, wohingegen der *CoAP-Endpunkt* des Proxy ein reiner Server ist. Der Datenendpunkt verwaltet ausschließlich die eigenen Ressourcen, der Proxy hingegen muss die Speicherung fremder Ressourcen handhaben können. Zu diesem Zweck besitzt letzterer einen Cache, der fremde Ressourcen aufnimmt. Die Komponente *HTTP-Server* muss zur Beantwortung der HTTP-Anfragen, die von den Benutzerendpunkten kommen, auf die Inhalte des Cache zugreifen können. Daher besitzt der *CoAP-Endpunkt* des Proxy eine Schnittstelle, die dem *HTTP-Server* den Zugriff auf den Cache ermöglicht.

4.4.4 HTTP-Server

Der typische Arbeitszyklus der Komponente *HTTP-Server* gestaltet sich wie folgt: Der *HTTP-Server* nimmt eine HTTP-GET-Anfrage über eine vom Benutzerendpunkt initiierte TCP-Verbindung entgegen, generiert unter Zuhilfenahme des beim *CoAP-Endpunkt* angesiedelten Cache eine Antwort, sendet diese an den Benutzerendpunkt und schließt die TCP-Verbindung. Der größte Aufwand verursacht das Parsen und Interpretieren der HTTP-Anfrage. Denn eine HTTP-Nachricht ist zwar vom Grundsatz her einfach aufgebaut, jedoch entsteht durch die große Anzahl der verschiedenen Header-Felder, die in der Nachricht vorkommen können, eine zusätzliche Komplexität. Ein HTTP/1.1-konformer Server muss alle Header-Felder einer HTTP-Anfrage parsen und entsprechend reagieren. Diese Forderung kann im Rahmen dieser Arbeit nicht erfüllt werden. Die Komponente *HTTP-Server* beachtet nur diejenigen Teile eines HTTP-Requests, die unbedingt notwendig sind, um die Anfrage beantworten zu können. Da sich diese Bachelorarbeit die Realisierung eines Prototypen zum Ziel gesetzt hat, kann diese nicht standardkonforme Implementierung eines HTTP-Servers akzeptiert werden.

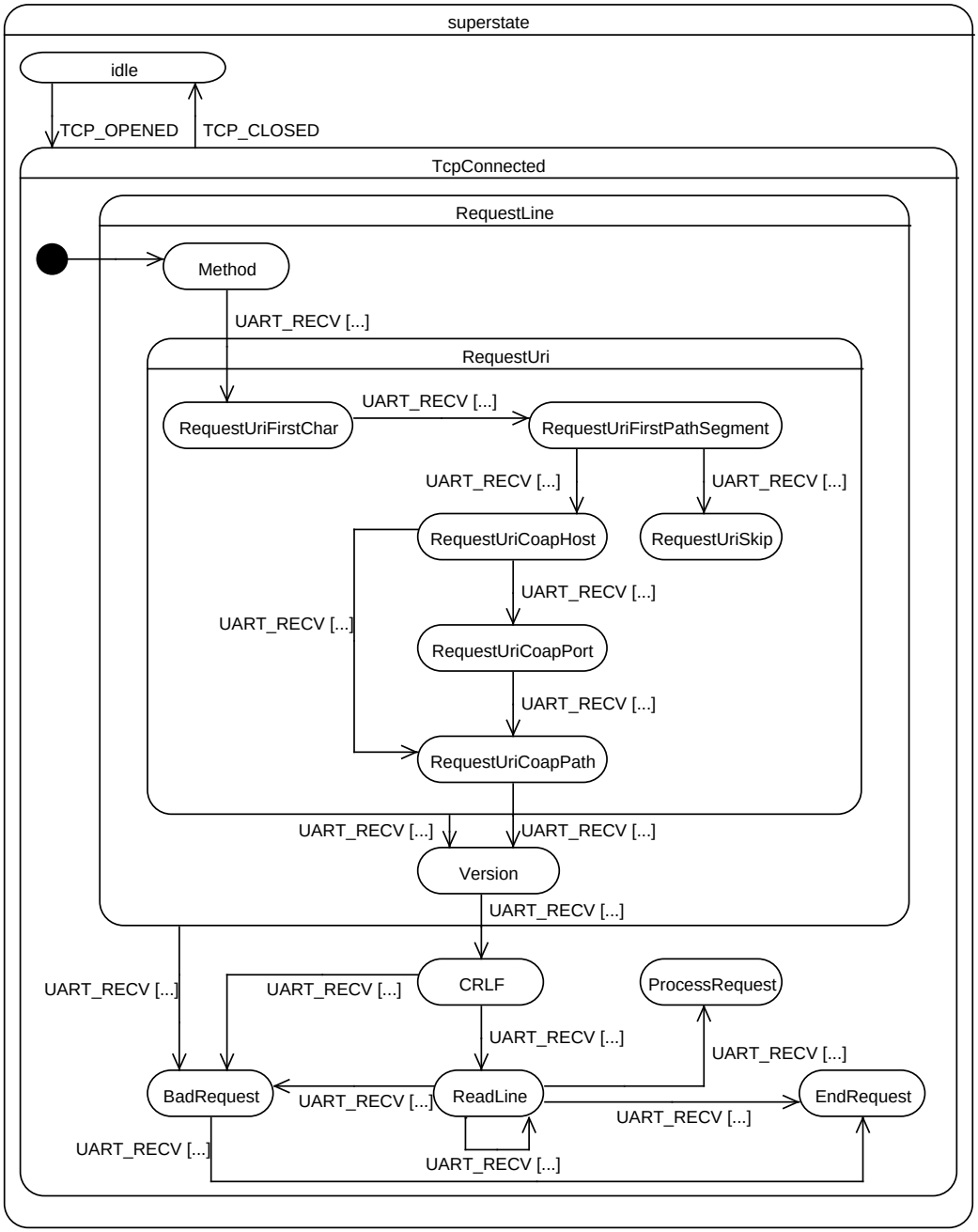

Abbildung 4.6: Zustandsdiagramm HTTP-Server

5 Abschließende Betrachtungen

5.1 Zusammenfassung

Es konnte ein verteiltes System zur Messwerterfassung aufgebaut werden, mit dem das in Abschnitt 1.2 beschriebene Szenario realisiert werden kann. Ein Benutzer kann mit einem Webbrowser über ein WLAN-Netz die Ressourcen von batteriebetriebenen Datenendpunkten abfragen. Zu diesem Zweck nimmt der Webbrowser nicht direkt mit dem Datenendpunkt Kontakt auf, sondern richtet seine Anfrage an den Proxy. Der Datenendpunkt nutzt den Proxy, dessen WLAN-Transceiver ständig empfangsbereit ist, als Zwischenspeicher für seine Ressourcen. Der WLAN-Transceiver des Datenendpunktes ist die meiste Zeit ausgeschaltet. Nur wenn der Datenendpunkt die Repräsentation seiner Ressourcen beim Proxy aktualisieren muss, schaltet er seinen WLAN-Transceiver kurzzeitig ein. Durch dieses Vorgehen ist sichergestellt, dass die Ressourcen jederzeit abgefragt werden können und zugleich der Datenendpunkt sehr energieeffizient betrieben werden kann.

5.2 Auswertung

Das wichtigste Ziel dieser Bachelorarbeit, das Design und die Entwicklung eines Prototypen, konnte erreicht werden. Vor dem Hintergrund der zur Verfügung stehenden Zeit mussten in der Realisierungsphase einige Kompromisse gemacht werden, die nun rückblickend bewertet werden sollen. Zugleich ergeben sich dadurch auch Anknüpfungspunkte für Weiterentwicklungsmöglichkeiten oder nachfolgende Bachelorarbeiten.

5.2.1 Auswahl des WLAN-Moduls

Es gibt keine standardisierte oder wenigstens herstellerübergreifende API zur Steuerung von WLAN-Modulen in Mikrorechnersystemen, d. h. die Bedienung ist von Modell zu Modell verschieden. In der Recherchephase dieser Arbeit konnte kein Dokument gefunden werden, das einen guten Überblick liefert über die von den verschiedenen Herstellern erdachten Konzepte zur software-seitigen Anbindung von WLAN-Modulen an Mikrocontroller-Systeme. So ist man

auf die Informationen angewiesen, die die Hersteller von WLAN-Modulen auf ihren Webseiten veröffentlichen. Angaben zu den Eigenschaften der Produkte hinsichtlich der unterstützten Protokolle, des Stromverbrauchs oder der Hardware-Schnittstellen sind immer leicht zu finden. Wie die Funktionen des WLAN-Moduls von einem auf dem Mikrocontroller laufenden Programm genutzt werden, wird auf den Webseiten erstaunlicherweise kaum erläutert. In der Anfangsphase des Projekts waren jedoch gerade diesbezügliche Informationen erforderlich, um eine begründete Auswahl eines WLAN-Moduls treffen zu können. Der Hersteller Roving Networks stellte hier eine Ausnahme dar: Auf der Webseite, auf der das WLAN-Modul *RN-174 Development Board* beschrieben wird, ist ein Link zu einem Manual vorhanden, das die Software-Schnittstelle des Moduls detailliert beschreibt. Anhand der vorab verfügbaren Beschreibungen wurde entschieden, dieses Modul zu beschaffen. Die Vorstellung, das WLAN-Modul ließe sich mit geringem Aufwand in ein Mikrocontroller-System integrieren, hat sich nicht bewahrheitet. Es musste zunächst eine Software-Komponente (*WLAN-Treiber*) entwickelt werden, die die Kommunikation mit der Bedienschnittstelle des WLAN-Moduls abwickelt. Lösungen anderer Hersteller, die eine fertige Treiber-Bibliothek mitbringen, lassen sich möglicherweise schneller integrieren. Rückblickend muss leider festgestellt werden, dass das *RN-174 Development Board* in seinem derzeitigen Entwicklungszustand für den Datenendpunkt und den Proxy nicht hinreichend gut geeignet ist. Insbesondere die Weiterleitung von empfangenen UDP- und TCP-Nachrichten an den angeschlossenen Mikrocontroller ist unzureichend. Einige Defizite könnten sicherlich durch Modifikationen der Firmware des Moduls beseitigt werden. Roving Networks bietet zum Preis von 2500 $ ein Software Development Kit an, mit dem es möglich ist, die Firmware des Moduls zu verändern. Eine Gegenüberstellung verschiedener WLAN-Module und eine Untersuchung der Eignung für den Einsatz in Mikrocontroller-Systemen konnte im Rahmen dieser Arbeit nicht durchgeführt werden. In diesem Zusammenhang sind auch Messungen zum zeitlichen Verhalten und zum Stomverbrauch der WLAN-Module von Interesse. Dies könnte in einer nachfolgenden Bachelorarbeit geleistet werden.

5.2.2 Verwendung des CoAP-Protokolls

Es wurde entschieden, für die Kommunikation zwischen Datenendpunkt und Proxy das CoAP-Protokoll einzusetzen. Die Verwendung von CoAP anstelle von HTTP ist aus fachlichen Gründen gerechtfertigt. Es ist jedoch zu bedenken, dass nur ein HTTP-Server und nicht zusätzlich noch ein CoAP-Endpunkt hätte entwickelt werden müssen, wenn auch zwischen Datenendpunkt und Proxy das HTTP-Protokoll zum Einsatz gekommen wäre. Dann hätte in der praktischen Phase der Bachelorarbeit mehr Zeit für andere Aufgaben zur Verfügung gestanden, u. a. für eine messtechnische Evaluation des WLAN-Moduls.

Für die Implementierung des CoAP-Protokolls im Datenendpunkt und im Proxy wurde die Bibliothek von Olaf Bergmann erheblich verändert. Rückblickend muss festgestellt werden, dass der Aufwand unterschätzt wurde, der nötig war, um diese Implementierung für PC-Betriebssysteme so anzupassen, dass sie auf einem Mikrocontroller lauffähig ist. Kritisch zu hinterfragen ist vor allem das angestrebte Ziel, sämtliche Aufrufe von malloc aus dem Code zu entfernen. Dieses Ziel konnte zwar erreicht werden, jedoch nur unter erheblichem Zeiteinsatz. Der Gebrauch der malloc-Funktion in Mikrocontroller-Anwendungen sollte zwar aus guten Gründen vermieden werden. Jedoch hätten die malloc-Aufrufe im Code zunächst akzeptiert werden können, zumal es sich bei dem erstellten System um einen Prototypen handelt.

5.2.3 Verwendung des QP Frameworks

Der Einsatz des QP Frameworks kann in der Rückschau als überaus sinnvoll angesehen werden. Durch diese Entscheidung war es möglich, die Softwaresysteme Datenendpunkt und Proxy auf einer höheren Abstraktionsebene zu entwerfen als es mit einem herkömmlichen Realtime-Betriebsystem möglich gewesen wäre. Die Modelle bestehen im Wesentlichen aus aktiven Objekten, Zustandsdiagrammen und Ereignissen. Sie lassen sich leicht in ausführbaren Code umsetzen und der Anwendungsentwickler muss sich keine Gedanken machen um all die Probleme, die im Zusammenhang mit Multitasking auftreten. Besonders bei der Modellierung des Verhaltens der Komponente *WLAN-Treiber* waren die Möglichkeiten der UML-Zustandsdiagramme von großem Vorteil. Die Komponente unterstützt derzeit nur diejenigen Funktionen des WLAN-Moduls, die im Rahmen dieser Arbeit benötigt wurden. Dank der verwendeten Modellierungsmethode kann der WLAN-Treiber jedoch leicht erweitert werden.

Die Verwendung der CoAP-Bibliothek von Bergmann als Grundlage für die Komponente *CoAP-Endpunkt* hat gezeigt, dass der Einsatz existierenden Quellcodes in aktiven Objekten schwierig sein kann: Aktive Objekte dürfen ausschließlich mit nicht blockierendem Code implementiert werden. Daher war es unabdingbar, die blockierenden Aufrufe von Funktionen der Socket-API aus der CoAP-Bibliothek zu entfernen. Für dieses Problem gibt noch einen anderen Lösungsansatz: Das QP-Framework kann auch mit einem herkömmlichen Echtzeitbetriebssystem zusammen betrieben werden. So ist es möglich, Software-Komponenten, die blockierenden Code enthalten, als eigenständige Tasks laufen zu lassen. Dieser Weg wurde jedoch nicht beschritten, da die Socket-API nicht mit dem Bedienkonzept des WLAN-Moduls kompatibel ist.

Literaturverzeichnis

RN-1742012 2012 *Roving Networks RN-174 WiFly Super Module.* Version 1.3r. Los Gatos, Sep 2012. – URL http://www.rovingnetworks.com/resources/download/14/RN_174

WiFly2012 2012 *Roving Networks User Manual and Command Reference: WIFLY GSX, WIFLY EZX.* Version 1.3r. Los Gatos, Feb 2012. – URL http://www.rovingnetworks.com/resources/download/93/WiFly_User_Manual

Berners-Lee u. a. 2005 Berners-Lee, T. ; Masinter, L. ; Fielding, R.: Uniform Resource Identifier (URI): Generic Syntax / RFC Editor. URL http://tools.ietf.org/html/rfc3986, Jan 2005 (3986). – RFC

Castellani u. a. 2012 Castellani, A. ; Loreto, S. ; Rahman, A. ; Fossati, T. ; Dijk, E.: Best Practices for HTTP-CoAP Mapping Implementation / IETF Secretariat. URL http://tools.ietf.org/html/draft-castellani-core-http-mapping-05, Jul 2012 (draft-castellani-core-http-mapping-05). – Internet-Draft

Crocker und Overell 2008 Crocker, D. ; Overell, P.: Augmented BNF for Syntax Specifications: ABNF / RFC Editor. URL http://tools.ietf.org/html/rfc5234, Jan 2008 (5234). – RFC

Fossati u. a. 2012 Fossati, T. ; Giacomin, P. ; Loreto, S.: Monitor Option for CoAP / IETF Secretariat. URL http://tools.ietf.org/html/draft-fossati-core-monitor-option-00, Jul 2012 (draft-fossati-core-monitor-option-00). – Internet-Draft

Guinard u. a. 2010 Guinard, Dominique ; Trifa, Vlad ; Wilde, Erik: A resource oriented architecture for the Web of Things. In: *IOT*, 2010

Samek 2009 Samek, Miro: *Practical UML Statecharts in C/C++.* Newnes, 2009

Sauter 2011 Sauter, Martin: *Grundkurs Mobile Kommunikationssysteme.* Vieweg + Teubner, 2011

Shelby u. a. 2012 Shelby, Z. ; Hartke, K. ; Bormann, C. ; Frank, B.: Constrained Application Protocol (CoAP) / IETF Secretariat. URL http://tools.ietf.org/html/draft-ietf-core-coap-10, Jun 2012 (draft-ietf-core-coap-10). – Internet-Draft

Shelby 2010 Shelby, Zach: The Internet of Things: Embedded Web Services. In: *IEEE Wireless Communications*, Dec 2010

Yiu 2010 Yiu, Joseph: *The Definitive Guide to the ARM Cortex-M3*. Newnes, 2010

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

Hamburg, 24. August 2012 Till Gerken