Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Patrick Büsselmann

Entwicklung einer M2M-Anwendung mit mobilen, Mikrocontroller-basierten Data-End-Points und UMTS-Datenübertragung

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Patrick Büsselmann

Entwicklung einer M2M-Anwendung mit mobilen, Mikrocontroller-basierten Data-End-Points und UMTS-Datenübertragung

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor 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.-Ing. Bernd Schwarz

Abgegeben am 28.07.2011

Patrick Büsselmann

Thema der Bachelorarbeit

Entwicklung einer M2M-Anwendung mit mobilen, Mikrocontroller-basierten Data-End-Points und UMTS-Datenübertragung

Stichworte

Machine-to-Machine, ARM Cortex-M3, UMTS-Stick, lwIP, USB, FreeRTOS, REST-Webservice, Play! Framework, HTTP, TCP/IP

Kurzzusammenfassung

Diese Arbeit behandelt die Entwicklung einer Machine-to-Machine Anwendung. Das Ziel ist es eine vielseitig einsetzbare Anwendung zu realisieren, die Informationen aus Gegenden ohne Energieversorgung an einen zentralen Server überträgt. Diese Informationen sollen es den Benutzern ermöglichen schnell auf externe Ereignisse reagieren zu können, umso beispielsweise Betriebskosten zu senken. Ein minimaler Energieverbrauch bei den Data-End-Points sowie geringe Kosten sind Merkmale, die zusätzlich von der Anwendung gefordert werden. Realisiert wurde ein Data-End-Point aus einem ARM Cortex-M3 Mikrocontroller und einem handelsüblichen UMTS-Stick. Dieser überträgt die Informationen AES-verschlüsselt über das Internet an den zentralen Server, welcher eine REST-basierte Webanwendung ausführt. Diese Webanwendung hat zudem die Aufgabe die empfangenen Informationen zu speichern und für die Benutzer darzustellen.

Patrick Büsselmann

Title of the paper

A M2M application with mobile, microcontroller based data end points and 3G data transmission

Keywords

Machine-to-Machine, ARM Cortex-M3, 3G USB Modem, lwIP, USB, FreeRTOS, REST Web Services, Play! Framework, HTTP, TCP/IP

Abstract

This thesis addresses the development of a machine-to-machine application. The aim is to realize a versatile application that transmits information from areas, which lack of a power supply, to a central server. This information should enable users to react quickly to external events, for instance, for cutting operating costs. Minimal power consumption by the data end points, and low costs are features, which are additionally required by the application. A Data Endpoint is carried out as a combination of an ARM Cortex-M3 microcontroller and a commercially available 3G USB Modem. The Data Endpoint transmits AES-encrypted information over the Internet to the central server, which executes a REST-based web application. This web application is also responsible for saving and displaying the received information for the users.

Inhaltsverzeichnis

Kapitel 1

Einführung

Die Popularität des mobilen Internets ist in den letzten Jahren stetig gewachsen und liegt im Trend. Die Möglichkeit Informationen abzurufen oder zu senden zu jeder Zeit und an fast jedem Ort, ist für fast ein Fünftel aller Internetbenutzer heutzutage schon zur Selbstverständlichkeit geworden [Bitkom (2011)]. Zusammen mit der wachsenden Beliebtheit des mobilen Internets ist zudem ein Mobilfunkmarkt entstanden, in welchem heutzutage viele Unternehmen konkurrieren. Die Folge dessen ist, dass die Preise in den letzten Jahren für den Zugang zum mobilen Internet als auch die der zugehörigen Hardware für Verbraucher stark gesunken sind.

Aufgrund der sich ständig weiterentwickelnden Mobilfunktechnik konnte zudem der Machine-to-Machine (M2M) Kommunikationsmarkt enorm zulegen. Machine-to-Machine beschreibt den automatisierten Informationsaustausch zwischen meist mobilen Clients (Data-End-Points) und einem zentralen Server. Das M2M Anwendungsspektrum ist vielfältig und reicht von der Übertragung von Sensormesswerten (Telemetrie) bis zu dem intelligenten Wohnen, bei welchem viele der Einrichtungsgegenstände untereinander vernetzt sind. Insbesondere der zuerst genannte Anwendungsbereich profitiert enorm von der günstigen aber zugleich auch technisch weit forgeschrittenen Mobilfunktechnik. Denn hierdurch ist es möglich geworden beispielsweise Umgebungsinformationen aus Gegenden bereitzustellen, wo zuvor aufgrund der Entfernung die Anwendung von Nahbereichfunksystemen nicht möglich war und Kabel-basierte Lösungen wirtschaftlich nicht sinnvoll waren. Ein Indiz für den blühenden drahtlosen Markt ist eine Studie der E-Plus Gruppe, die schätzt, dass 2010 2,3 Millionen SIM-Karten in Deutschland in Machine-to-Machine Anwendungen verwendet wurden. Für 2013 wird prognostiziert, dass 5 Millionen verwendet werden, was eine Verdopplung des Marktes bedeutet [Böning u. a. (2010)].

Trotz dieses wirtschaftlichen Booms sind Lösungen aus dem akademischen Bereich rar gesät, insbesondere solche, die das mobile Internet in Verbindung mit einem modernen Mobilfunkstandard zur Übertragung von Informationen nutzen. Das Ziel dieser Arbeit ist eine

Machine-to-Machine Anwendung zu entwickeln basierend auf mobilen Clients, welche echtzeitnah Informationen aus Gegenden ohne Energieversorgung an einen zentralen Server übermitteln. Dabei soll diese Anwendung sich als kostenwirksame und vielseitig einsetzbare Lösung darstellen. Zusätzlich ist es für dessen Clients erforderlich, dass diese möglichst energieeffizient arbeiten.

Der Nutzen, welcher aus den durch die Anwendung übertragenen Informationen gezogen werden kann, ist folgender: Mithilfe dieser Informationen soll der Benutzer schnell auf externe Ereignisse reagieren können, wodurch es möglich wird, Kosten zu senken und Geschäftsprozesse zu optimieren.

Gliederung

Im Anschluss an dieses Kapitel wird zunächst die beabsichtige Anwendung konzeptionell dargestellt (2) inklusive möglicher Anwendungsszenarien. Zudem wird die Zielsetzung dieser Arbeit in diesem Kapitel weiter konkretisiert.

Danach wird auf die Grundlagen (3) eingegangen, welche für das weitergehende Verständnis der nachfolgenden Kapitel benötigt werden. Zudem werden dort (3.1) auch die im Titel dieser Arbeit verwendeten Begriffe aus der Machine-to-Machine Terminologie weiter erläutert.

In Kapitel 4 werden mögliche Ansätze zur Lösung der Aufgabenstellung aufgezeigt basierend auf wissenschaftlichen Arbeiten und kommerziellen Produkten.

Beruhend auf diesen gewonnenen Erfahrungen werden im nächsten Kapitel (5) abstrakt die Komponententypen gewählt, welche in dieser Arbeit Verwendung finden sollen. Darauf aufbauend wird in Kapitel 6 der Ablauf der Anwendung spezifiziert und unter Wahl der konkreten Komponenten die komplette Anwendung entworfen.

Anschließend werden die relevanten Punkte der Realisierung (7) anhand von Quellcodeausschnitten vorgestellt. Im weiteren Verlauf wird noch ein Data-End-Point, welcher in Rahmen dieser Arbeit den Namen Sensorstation erhält, unter anderem in Hinblick auf dessen Energieverbrauch messtechnisch untersucht (8).

Abschließend werden noch Möglichkeiten zur Weiterentwicklung des realisierten Prototyps aufgezeigt (9) und eine Zusammenfassung über diese Arbeit aufgestellt (10).

Kapitel 2

Konzept der Anwendung

2.1 Mögliche Anwendungsszenarien

Zuerst werden mögliche Einsatzgebiete der beabsichtigten Anwendung in zwei Szenarien dargestellt.

2.1.1 Szenario A: Vermeidung von Bodenfrost

Die Notwendigkeit von genauen Temperaturmessungen besteht unter anderem im Obst-, Wein- und Gemüseanbau, wenn zum Beispiel die Gefahr von Frost besteht. Um Frostschäden entgegen zu wirken, werden bei Frostgefahr die Nutzpflanzen feinstäubig mit Wasser beregnet. Durch diese konstante Beregnung entsteht beim Gefrieren des Wassers die sogenannte Kristallisationswärme, wodurch die Temperatur innerhalb der Eishülle nicht unter den Nullpunkt fallen kann. Aufgrund des Mikroklimas[1], gegeben durch unterschiedliche Bodentemperaturen, können sich zudem schon auf einer Strecke von weniger als 100 Meter deutliche Unterschiede in der Lufttemperatur ergeben. Dadurch wird eine effiziente Aktivierung der Beregnung erschwert. Es ist daher erstrebenswert eine große Anzahl von Sensoren einzusetzen, um dadurch die Anlagen zur Beregnung effektiv zu steuern und dadurch Betriebskosten einzusparen.

2.1.2 Szenario B: Überwachung aus Gebieten ohne Energieversorgung

Der Bedarf nach Überwachung auch aus Gebieten, welche infrastrukturell gesehen keine Energieversorgung bieten, ist auf vielfältige Art gegeben. Um Vandalismus, Diebstahl und

[1] Beschreibt das Klima der bodennahen Luftschichten in einem kleinen Bereich.

Abbildung 2.1: Durch Frostschutzberegnung geschützte Apfelbaumknospen

andere Straftaten zu erschweren, könnte eine Kameralösung realisiert werden, welche entweder gesteuert durch einen Bewegungsmelder oder schlicht im Minutentakt Bilddaten an einen zentralen Server übermittelt. Weitere Anwendungsgebiete können aber durchaus auch angenehmere Hintergedanken haben, wie die Beobachtung von Tieren in freier Natur als auch die Bereitstellung eines Webcam-Dienstes, bei dem das Gesamtsystem mobil ist.

Anmerkung: In dem weiteren Verlauf der Arbeit wird nur noch von Sensoren gesprochen werden, worunter auch die hier genannte Kamera einzuordnen ist.

2.1.3 Ein erstes Resümee basierend auf den vorgestellten Szenarien

Eine vorstellbare Lösung für die beiden genannten Szenarien[2] wäre die Verwendung von Nahbereichsfunksystemen, die zum Beispiel auf der freien Frequenz 868 MHz senden. Resultierend dadurch ist die Reichweite deutlich begrenzt und erfordert somit eine hohe Dichte an Sensorknoten, um eine Verbindung zu dem Server herzustellen. Diesen Ansatz findet man in dem Forschungsfeld der drahtlosen Sensornetzwerke wieder, in denen die Sensorknoten ein vermaschtes Netz für die Kommunikation bilden. Jedoch nachteilig anzusehen ist

[2] Der mobile Webcam-Dienst ist hiervon ausgenommen.

bei dieser Lösung der hohe Wartungs- und Kostenfaktor, welcher durch die hohe und mit der Entfernung steigende Anzahl der Sensorknoten gegeben ist.

Weitere Ansätze sind die Nutzung von Mikrocontrollern oder Linux-basierten Einplatinen-Computern[3] in Verbindung mit jeweils einem Modem. Im Vergleich zwischen diesen beiden Systemen zeigen Mikrocontroller klare Vorteile zu der Bewältigung dieser Problemstellungen. Zum einen sind diese deutlich preiswerter, unter anderem wegen des geringeren Speicherbedarfs, als auch beträchtlich energieeffizienter. Aus diesem Grund ist die Verwendung von Mikrocontrollern den Linux-basierten Systemen vorzuziehen.

Basierend auf den beiden genannten Szenarien als auch diesen Vorüberlegungen stellt sich die Zielsetzung dieser Anwendung wie folgt dar:

2.2 Zielsetzung der Anwendung

Die Anwendung soll aus vielen mikrocontroller-basierten Sensorstationen und genau einem Server bestehen.

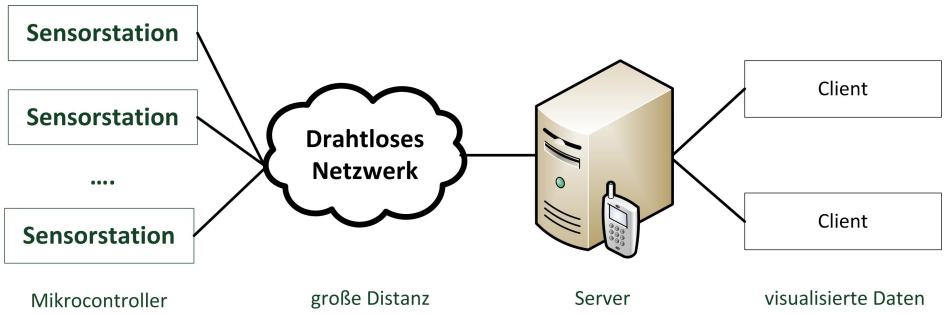

Abbildung 2.2: Schema der beabsichtigen Anwendung

- Die Aufgabe einer Sensorstationen ist es, die Daten der angeschlossenen Sensoren auszulesen und diese an den unter Umständen mehreren Kilometer entfernten Server zu senden. Die Sensorstationen sollen zudem einen möglichst niedrigen Energieverbrauch aufweisen, da die beabsichtigen Einsatzorte jeweils keine Stromversorgung bieten. Dadurch ist die Nutzung einer Batterie erforderlich.
- Die Funktion des Servers ist es die Sensorinformationen zu empfangen, speichern, überwachen und visuell für die Benutzer darzustellen.

[3] Das größte Unterscheidungskriterium zwischen diesen beiden Systemen ist, dass ein Mikrocontroller nur Steueraufgaben abwickelt. Der Einplatinen-Computer bietet jedoch unter anderem zusätzlich eine Schnittstelle für die Benutzerinteraktion.

2.3 Anforderungen an die Anwendung

Aus dieser Zielsetzung ergeben sich allgemeine als auch systemspezifische Anforderungen an die Sensorstationen und den Server.

2.3.1 Allgemeine Anforderungen an das Gesamtsystem

Es ist eine möglichst kostengünstige Lösung zu erarbeiten, welche zudem vielseitig einsetzbar sein soll. Aus diesem Grund empfiehlt sich der Einsatz von Hardware, die für den Massenmarkt entworfen wurde. Diese ist im Normalfall deutlich preiswerter als dessen industrielles Gegenstück. Des Weiteren sollen softwareseitig nur Open-Source-Bibliotheken verwendet werden, umso Lizenzgebühren zu vermeiden.

Neben den selbstverständlichen nicht-funktionalen Forderungen nach einem zuverlässigen und performanten System soll die Anwendung zudem auch die Option offen lassen möglichst einfach erweitert als auch auf unterschiedliche Plattformen portiert werden zu können.

Ferner soll das Gesamtsystem in Hinblick auf Usability-Aspekte möglichst einfach und selbsterklärend gestaltet werden. Die Inbetriebnahme als auch das Erweitern des Gesamtsystems mit neuen Sensorstationen oder Sensoren soll einfach und ohne großen Aufwand möglich sein.

Darüber hinaus soll die verteilte Anwendung sicher sein, was unter anderem bedeutet, dass alle Datenübertragungen verschlüsselt ablaufen müssen. Insbesondere ist eine verschlüsselte Datenübertragung zwischen den Sensorstationen und dem Server notwendig. Dies ist notwendig, da ansonsten Dritte dem Server manipulierte Sensorinformationen zusenden können, wodurch ein wirtschaftlicher Schaden entstehen könnte.

2.3.2 Anforderungen an eine Sensorstation

Die Sensorstation muss auf einem Mikrocontroller basieren. Zudem muss die Sensorstation Mobilität ermöglichen und mindestens folgende Anforderungen unterstützen.

Sensordaten Die Sensorstation muss die vorgegebenen und angeschlossenen Sensoren unterstützen. Zudem müssen diese periodisch ausgelesen und in ein vereinbartes Format umgewandelt werden.

Datenübertragung Die periodisch ausgelesenen Sensorinformationen müssen ebenfalls regelmäßig an den Server gesendet werden, welcher möglicherweise weit entfernt ist.

Energieeffizienz Die Sensorstation soll durch eine Batterie betrieben werden, wodurch ein möglichst geringer Energieverbrauch erforderlich ist. Eine Laufzeit von einigen Wochen ohne Batteriewechsel in der Zwischenzeit soll erreicht werden.

2.3.3 Anforderungen an den Server

Der Server muss mindestens folgenden Anforderungen gerecht werden:

Erreichbarkeit Der Server muss für die Sensorstationen als auch für die Benutzer möglichst ohne Ausfallszeiten 24 Stunden am Tag, 7 Tage die Woche erreichbar sein. Zudem muss der Server entweder eine feste Adresse besitzen oder es muss die Möglichkeit der Auffindung des Servers mithilfe eines Namensdienstes gegeben sein.

Datenempfang Für die durch die Sensorstationen gesendeten Informationen muss eine Schnittstelle zur Verfügung gestellt werden.

Persistenz Alle empfangenen Informationen müssen persistent in einem Datenbanksystem gespeichert werden.

Überwachung Die Aufgabe der Überwachung lässt sich in zwei Teilbereiche aufspalten: Zum einem muss der Server die Nutzer informieren, beispielsweise per E-Mail, sobald sich Sensoren ihrem kritischem Überwachungsbereich annähern. Auf Szenario A bezogen würde dies bedeuten, dass die Anwender informiert werden, sobald sich die Temperatursensoren der Frostgrenze nähern. Zum anderen hat der Server die Aufgabe sicherzustellen, dass alle Sensorstationen samt deren Sensoren sich in festen Abständen mit neuen Informationen melden. Bleibt eine Meldung über längere Zeit aus, kann man davon ausgehen, dass die entsprechende Sensorstation ein Problem hat.

Darstellung Alle in der Datenbank gespeicherten Sensorinformationen sollen visuell je nach Sensorart dargestellt werden können, umso den Anwendern die Möglichkeit zu geben Situationen schnell einzuschätzen. Des Weiteren sind Filtermöglichkeiten für unterschiedliche Darstellungen wünschenswert.

Administration Da schon in Kapitel 2.3.1 gefordert wurde, dass die Anwendung leicht in Betrieb genommen und erweitertet werden kann, ergibt sich zusätzlich noch eine weitere Anforderung. Alle gespeicherten Informationen müssen leicht verwaltet und editiert werden können.

2.4 Umsetzung eines Prototyps

In dieser Bachelorarbeit ist ein erster Prototyp der beschriebenen Anwendung zu entwickeln, wobei jedoch der Schwerpunkt bei der Umsetzung einer Sensorstation liegen soll. Durch diese soll die Machbarkeit dieser Aufgabenstellung nachgewiesen werden. Zudem sollen Abschätzungen und Erkenntnisse mit dem Prototyp gewonnen werden, insbesondere zu dem zeitlichen Übertragungsverhalten, dem benötigten Energiebedarf und dem möglichen Datendurchsatz.

Der Prototyp soll an das in Abschnitt 2.1.1 genannte Szenario A angelehnt sein. Bei dem Design jedoch soll die Anwendung so generisch wie möglich gestaltet werden, sodass der fertige Prototyp mit relativ geringem Aufwand auch beispielsweise für das zweite genannte Szenario genutzt werden kann.

Als Sensoren werden in Rahmen dieser Arbeit 1-Wire Temperatursensoren[4] eingesetzt, da diese ebenfalls im Praktikum der Veranstaltung „Grundlagen systemnahes Programmieren" verwendet werden und somit in der Hochschule vorhanden sind.

[4] Bei den Temperatursensoren handelt es sich um das Produkt D18B20 der Firma Maxim Integrated Products.

Kapitel 3

Grundlagen der genutzten Technologien

Dieses Kapitel zeigt kurz und prägnant die Grundlagen auf, welche erforderlich sind für den weiteren Verlauf dieser Arbeit.

3.1 Machine-to-Machine (M2M)

Machine-to-Machine (abgekürzt M2M) beschreibt den automatisierten Datenaustausch zwischen mindestens 2 Maschinen, Automaten, Fahrzeugen, Containern. Der Anwendungsschwerpunkt liegt in der Überwachung, Steuerung oder Wartung aus der Ferne.

Typischerweise ist die M2M Architektur aufgebaut ähnlich einer Client-Server-Architektur und standardmäßig durch 3 Komponenten beschrieben (vgl. [Glanz und Jung (2010)]):

Abbildung 3.1: Die M2M-Architektur

Data-End-Point (DEP) Kennzeichnet ein System, welches Mess-, Leistungs- und/oder Systemwerte von angeschlossenen Sensoren über das Kommunikationsnetz an den Data-Integration-Point versendet. Weitere Kennzeichen sind sehr oft minimale Interaktion mit Menschen und ein geringer Energieverbrauch. Im Normalfall enthält eine M2M Anwendung mehrere Data-End-Points.

M2M-Communication-Network (MCN) Beschreibt eine beliebige Übertragungstechnik die Data-End-Points und den Data-Integration-Point verbindet. Typischerweise ist diese drahtlos, zumindest auf Seite der Data-End-Points.

Data-Integration-Point (DIP) Ein zentraler Server, der gewöhnlich nur einmal in der Anwendung vorhanden ist. Dieser speichert persistent die Sensordaten der Data-End-Points und stellt diese optisch dar.

Machine-to-Machine gilt als Zukunftsmarkt, da Unternehmen dadurch Prozesse optimieren und vereinfachen können. Zudem können teure Ausfallzeiten von Maschinen durch die automatische Meldung von Störungen vermieden werden. Dies führt zu einer höheren Effektivität, Kosteneinsparungen und je nach Anwendungsfall zur Automatisierung. Obendrein kann die Wettbewerbsfähigkeit gesteigert werden mit neuen innovativen Produkten und Diensten, die durch die M2M Kommunikation ermöglicht werden.

Anmerkung: In dem weiteren Verlauf dieser Arbeit wird nur noch der Aspekt der Überwachung aus der Ferne in Zusammenhang mit M2M betrachtet.

3.2 UMTS Sticks

Unter einem UMTS Stick versteht man ein USB-Modem, welches eine UMTS-Internetverbindung herstellen kann. Andere Namen für einen UMTS Stick sind Surf-Stick oder Datenstick.

Abbildung 3.2: Ein UTMS Stick (Huawei 169)

3.2.1 Universal Serial Bus (USB)

Der Universal Serial Bus, kurz USB, ist ein serielles Bussystem und hat sich in den letzten Jahren zu der meistverwendeten Computerschnittstelle für externe Geräte entwickelt. USB verwendet eine Master-Slave-Architektur. Dies bedeutet jegliche Kommunikation geht vom Master aus, welcher auch Host genannt wird und dieser ist typischerweise der Computer. Nur nach Aufforderung sendet der Slave, welcher normalerweise das USB-Gerät repräsentiert, Daten an den Host.

Die USB Spezifikation definiert, dass ein USB Gerät mit einer Versorgungsspannung von 5 Volt betrieben wird und einen maximalen Stromverbrauch von 500 mA haben darf.

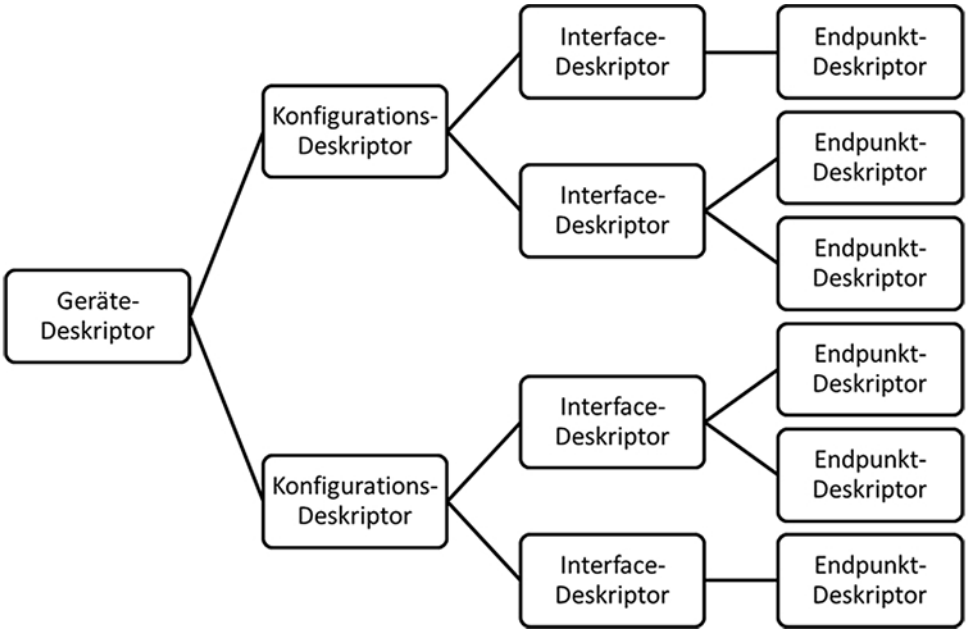

Abbildung 3.3: Übersicht der Standard-Deskriptoren

Ein USB Gerät wird durch Deskriptoren beschrieben. Die Abbildung 3.3 zeigt ein mögliches USB-Gerät und den hierarchischen Aufbau der Deskriptoren. Der Geräte-Deskriptor ist einmalig und enthält unter anderem die Vendor- und Product-ID, welche zusammen jedes USB-Produkt eindeutig kennzeichnen. Die im Baum tiefer gelegenen Deskriptoren können alle mehrfach vorkommen und enthalten jeweils Informationen über die direkt darunter gelegene Deskriptoren-Schicht. Es muss von jeder Art mindestens einer vorhanden sein und es kann immer nur ein Pfad in diesem Baum bis zur Ebene der Interface-Deskriptoren aktiv sein.

Der Konfigurations-Deskriptor beschreibt hauptsächlich das Stromprofil der Anwendung. Dies kann sich unterscheiden, wenn ein USB-Gerät zusätzlich mit einem externen Netzteil betrieben werden kann.

Ein Interface-Deskriptor beschreibt eine Funktionalität, die das USB-Gerät bietet. Die Funktionalitäten sind durch sogenannte Interface Geräteklassen standardisiert. Die bekanntesten Vertreter dieser Klassen sind sicherlich die Massenspeicher- und die HID-Klasse (Tastatur, Maus). Für diese Klassen sind standardmäßig Betriebssystem-Treiber verfügbar. Lässt sich die Funktionalität nicht in eine dieser Klassen einordnen, wird die herstellerspezifische Klasse gewählt.

Die Endpunkt-Deskriptoren sind die eigentlichen Kommunikationsports des USB-Geräts. Je nach Interface-Deskriptor sind einer oder mehrere Endpunkt-Deskriptoren vorhanden. Jeder Endpunkt-Deskriptor hat eine eindeutige ID zwischen 1 und 15 und lässt sich in einen IN-Endpunkt und einen OUT-Endpunkt aufteilen, die zusammen eine bidirektionale Kommunikation ermöglichen. Die Sicht auf die Richtung der Endpunkte ist immer vom Host ausgehend, dies bedeutet über einen OUT-Endpunkt werden Daten versendet. Zusätzlich muss jedes Gerät einen Endpunkt-Deskriptor mit der ID 0 anbieten, welchen der Host für die Konfiguration des Gerätes verwendet.

Die eigentliche Kommunikation findet paketbasiert mithilfe von logischen Pipes statt. Eine sogenannte USB-Pipe verbindet einen Endpunkt des Masters mit einem Endpunkt des Slaves. Für diese 1-zu-1 Verbindungen existieren unterschiedliche Transferarten. In dieser Arbeit kommen sogenannte Control- und Bulktransfers zum Einsatz. Controltransfers werden zum Konfigurieren des Gerätes verwendet und Bulktransfers sind für die einfache Übertragung der Nutzdaten zuständig.

Unter der USB-Enumerierung versteht man den Vorgang nach einem Reset. Dieser beinhaltet die Abfragen aller Deskriptoren über den Endpunkt 0. Nach der Abfrage hat der Host alle nötigen Informationen um das Gerät zu verwenden.

Anmerkung: Dieser Abschnitt hat den Universal Serial Bus nur sehr oberflächlich beschrieben. Eine detailliertere Beschreibung kann man unter anderem der Spezifikation entnehmen [USB IF (2000)].

3.2.2 USB Modeswitch

Nahezu alle UMTS Sticks besitzen neben der eigentlichen Modemfunktion auch noch einen Flashspeicher, der die normalerweise Windows-basierte Einwahlsoftware beinhaltet. Betrachtet man einen solchen UMTS Stick genauer, stellt man fest, dass dieser sowohl herstellerspezifische USB Interface-Deskriptoren, welche für die Modemfunktion zuständig sind, als auch einen Massenspeicher Interface-Deskriptor enthält. Nach dem Anschließen des Sticks an einen Computer ist standardmäßig das Massenspeicher-Interface aktiv, welches sich im Windows Explorer als CD Laufwerk darstellt. Dieses bietet die Möglichkeit, die Einwahlsoftware zu installieren. Durch die Verwendung der Einwahlsoftware und auch nur dann schaltet der UMTS Stick in den Modemmodus um und kann eine Internetverbindung herstellen.

Betrachtet man den USB-Bus während des Umschaltvorgangs, stellt man Folgendes fest: Zuerst wird ein spezielles Kommando gesendet, welches dazu führt, dass der UMTS Stick einen Reset auslöst. Nach dem Reset wird wieder die USB-Enumerierung durchgeführt, bei dem der UMTS Stick nun meldet, dass ein herstellerspezifischer Interface-Deskriptor aktiv ist. Dies bedeutet bei dem UMTS Stick ist der Modemmodus nun aktiv und nach der vollständigen Enumerierung reagiert dieser auf Befehle, welche dem AT-Befehlssatz entstammen.

Möchte man nun einen UMTS Stick mit einem nicht Windows-basierten System verwenden, so steht man vor dem Problem das Verhalten der Einwahlsoftware nachzuahmen, umso den Stick in den Modemmodus zu bringen. Glücklicherweise war die Linux Community insbesondere Josua Dietze hier aktiv, um diesem Problem Herr zu werden. Es wurde das Programm USB_ModeSwitch[5] entwickelt, welches auch in der aktuellen Ubuntu Distribution enthalten ist, um UMTS Sticks in den Modemmodus umzuschalten.

Untersucht man unterschiedliche UMTS Sticks oder das Programm USB_ModeSwitch genauer, stellt man fest, dass das Kommando zum Umschalten bei jedem Hersteller und teilweise sogar bei jedem Modell unterschiedlich ist. Möchte man einen UMTS Stick beispielsweise mit einem Mikrocontroller verwenden, ist USB_ModeSwitch ein guter Startpunkt, da der Umschaltvorgang für eine große Anzahl von Sticks implementiert wurde. Der Quellcode ist wie bei den meisten Linux-basierten Programmen frei verfügbar.

Manche UMTS Sticks lassen es zu, den Massenspeicher Modus zu deaktivieren und somit nur noch die Modemfunktionalität zu verwenden. Das Deaktivieren funktioniert mit einem AT-Befehl, der wiederum von Modell zu Modell variiert. Hier besteht aber das gleiche Problem wie beim Umschaltkommando: Der Hersteller veröffentlicht diese Befehle normalerweise nie. Dies führt wiederum dazu, dass das Herausfinden dieser Befehle sich oft sehr schwierig gestaltet.

3.2.3 AT-Befehlssatz

Der AT-Befehlssatz beschreibt eine Ansammlung von Befehlen für Modems zum Steuern, Konfigurieren und Auslesen von Statusinformationen. Der AT-Befehlssatz wird teilweise auch Hayes-Befehlssatz genannt, da dieser von der Firma „Hayes Microcomputer" in den 70er Jahren entwickelt wurde [Clark (2003)]. Dieser Befehlssatz hat sich zu einem Quasi-Standard entwickelt, wobei jeder Hersteller nicht alle Befehle unterstützt oder eigene hinzugefügt hat. Teilweise wurden diese Befehle durch das 3rd Generation Partnership Project für GSM Geräte standardisiert [3GPP TS 27.007 (2010)].

Grundlegend gesehen besteht jeder Befehl aus dem Kommando AT, auf welches das Modem

[5] http://www.draisberghof.de/usb_modeswitch/

entweder mit „OK” oder „ERROR”[6] antwortet. Des Weiteren muss jeder Befehl mit einem Carriage-Return Zeichen (\r) enden.

Die nachfolgenden Beispiele sollen das Prinzip etwas verdeutlichen, wobei hier aus Gründen der Lesbarkeit auf das Carriage-Return Zeichen verzichtet wird:

Befehl	Beschreibung
AT	Überprüft, ob das Modem betriebsbereit ist.
AT+CPIN?	Fragt den Status der SIM-Karte ab.
AT+CPIN=”1234”	Gibt den PIN „1234” ein.
AT+CGMI	Fragt den Hersteller des Modems ab.

3.3 Netzwerkprotokolle

Grundlegende Kenntnisse von Computernetzen werden in dieser Arbeit vorausgesetzt. Dennoch werden zwei für diese Arbeit essenzielle Protokolle beleuchtet und eine Eigenschaft von TCP aufgezeigt, welche die maximal mögliche Datenübertragungsrate stark beeinflussen kann.

3.3.1 Point-to-Point Protocol (PPP)

Das Netzzugangsprotokoll PPP ist für den Verbindungsaufbau zwischen zwei Netzwerkteilnehmern konzipiert. Verwendung findet es insbesondere für die Einwahl in das Internet und somit zur Kommunikation mit dem Internet-Service-Provider. Zudem ermöglicht es neben Authentifizierung der Teilnehmer auch den Transport von Datagrammen beliebiger Protokolle. Das Point-to-Point Protocol definiert eine Menge an Protokolle, die unter anderem zum Verbindungsaufbau verwendet werden. Typischerweise werden folgende Protokolle bei einem Verbindungsaufbau in das Internet verwendet:

LCP Zuerst werden Link Control Protocol (LCP) Pakete ausgetauscht. Mit diesen Paketen wird die Rahmenbedingung der Verbindung ausgehandelt, wie beispielsweise welches Authentifizierungsverfahren verwendet werden soll.

CHAP Das Challenge Handshake Authentication Protocol (CHAP) dient neben dem Password Authentication Protocol (PAP) zur Authentifizierung des Netzwerkteilnehmers. Das wesentliche Unterscheidungsmerkmal dieser beiden Protokolle ist, dass PAP Benutzername und Passwort in Klartext überträgt und CHAP hingegen für das Passwort

[6] Erweiterte Fehlermeldungen können aktiviert werden.

eine kryptografische Hashfunktion verwendet. Eines dieser Authentifizierungsverfahren wird im Anschluss an das LCP-Protokoll verwendet.

IPCP Nachdem der Teilnehmer sich authentifiziert hat, kommt das IP Control Protocol (IPCP) zum Einsatz, sofern das Zielnetz IP-basiert ist. Vereinfacht ausgedrückt hat dieses Protokoll den Zweck dem Teilnehmer seine künftige IP-Adresse zusammen mit der entsprechenden Subnetz-Maske und den IP-Adressen der DNS-Server mitzuteilen. Hiernach ist die Verbindung erfolgreich hergestellt.

3.3.2 Transmission Control Protocol (TCP)

Das Transmission Control Protocol (TCP) ermöglicht eine zuverlässige und verbindungsorientierte Datenübertragung. Auf das TCP-Protokoll bauen eine Vielzahl von Anwendungsprotokollen auf, wie zum Beispiel HTTP, FTP und IMAP. Damit die Zuverlässigkeit der Übertragung gewährleistet wird, muss, neben einem Dreiwegehandschlag für den Verbindungsaufbau und -abbau jedes versendete Netzwerkpaket von der Gegenseite bestätigt werden das es korrekt angekommen ist. Geht ein Netzwerkpaket verloren oder wird nicht korrekt übertragen, wird es vom Sender erneut gesendet.

Um eine möglichste effiziente Datenübertragung zu ermöglichen, enthält das TCP-Protokoll diverse Mechanismen zur Datenflusskontrolle. Dies führt dazu, dass in Hinblick auf die Ressourcen eines typischen Mikrocontrollers das TCP-Protokoll sehr fordernd ist. Insbesondere das "Sliding-Window" Verfahren hat einen sehr hohen Speicherbedarf sorgt aber für eine höhere Übertragungsgeschwindigkeit und zusätzlich das der versendete Overhead minimiert wird.

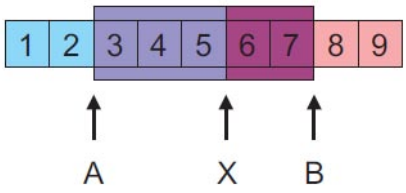

Abbildung 3.4: TCP Sliding-Window Beispiel

Die Abbildung 3.4 stellt die Funktionsweise des "Sliding-Window" Verfahren vereinfacht dar. Die Pakete 1 bis 9 sollen versendet werden. Der Pfeil A beschreibt das letzte bestätigte und damit erfolgreich übertragene Paket. Bei dem "Sliding-Window" Verfahren sendet der Empfänger mit jeder Paketbestätigung die aktuelle Anzahl der freien Bytes seines Empfangspuffers zusätzlich mit. Diese Größe wird Fenster genannt. Dies wird in der Abbildung als Strecke zwischen den Pfeilen A und B dargestellt. Der Vorteil dieser Vorgehensweise ist, dass der Sender nun weitere Pakete senden kann, ohne auf die Bestätigung des direkt

zuvor gesendeten Pakets zu warten, da der Sender weiß, wie viele Bytes der Empfänger noch entgegennehmen kann. In der Abbildung ist dies der Fall und es wurden die Pakete 4 und 5 gesendet (Pfeil X), obwohl Paket 3 noch nicht bestätigt wurde. Der andere Vorteil dieser Methodik ist, dass der Empfänger mehrere Pakete auf einmal bestätigen kann und so Protokolloverhead eingespart wird. Wenn ein Paket durch den Empfänger bestätigt wurde, verschiebt sich das Fenster und damit hier der Pfeil A nach rechts. Aufgrund dieser Verfahrensweise rührt auch der Name dieses Verfahren.

Der erhöhte Speicherbedarf bei diesem Verfahren ist aus folgendem Grund gegeben: Der Sender versendet nun mehrere Netzwerkpakete und muss diese so lange zwischenspeichern, bis die Pakete von der Gegenseite bestätigt wurden. Die Belohnung dafür ist eine deutlich erhöhte Übertragungsgeschwindigkeit.

Die mögliche Datenübertragungsrate ist neben der Fenstergröße von der Round Trip Time (RTT) abhängig. Diese beschreibt die Laufzeit von Punkt A nach Punkt B plus den Rückweg von Punkt B nach A. Bei einer UMTS-Internetverbindung muss man nach dem Verbindungsaufbau mit einer durchschnittlichen Round Trip Time von 300 Millisekunden rechnen. Diese sinkt nach wenigen Sekunden bei konstanter Datenübertragung auf einen Mittelwert von 100 Millisekunden ab.

Der Datendurchsatz ist wie folgt linear definiert, solange dieser kleiner ist als die maximale Übertragungsrate:

$$Datenübertragungsrate = \frac{Segementgröße \,/\, Fenstergröße}{Round\,Trip\,Time}$$

Wird das "Sliding-Window" Verfahren nicht verwendet, muss nach jedem Senden auf die Bestätigung der Gegenseite gewartet werden. Diese Vorgehensweise lässt sich bei manchen TCP/IP Stacks für Mikrocontroller wiederfinden. Dies ergibt am Anfang folgende maximale Datenübertragungsrate[7]:

$$\frac{1500\,Byte}{300\,ms} \approx 5\,\frac{kB}{s}$$

Diese sehr langsame Übertragungsgeschwindigkeit lässt sich durch das "Sliding-Window" Verfahren beschleunigen. Die Abbildung 3.5 zeigt den linearen Verlauf des Datendurchsatzes in Abhängigkeit der Fenstergröße.

Die Datenübertragungsrate steigt mit zunehmender Fenstergröße, bis die maximale Übertragungsgeschwindigkeit erreicht ist. Die minimale Fenstergröße, die benötigt wird, lässt sich aus der vorhergegangenen Formel herleiten und ist das Produkt aus dem Datendurchsatz und der Round Trip Time. Dies bedeutet, dass man für die maximale Upload-Geschwindigkeit

[7] Diese Übertragungsgeschwindigkeit kann deutlich geringer ausfallen falls der Empfänger zusätzlich den „Delayed ACK" Algorithmus [Braden (1989)] verwendet, da dann die Verzögerungszeit auf die Round Trip Time addiert wird.

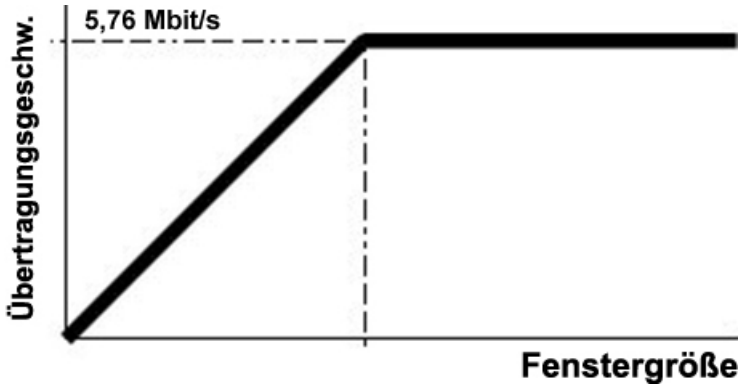

Abbildung 3.5: Die Datenübertragungsrate unter Verwendung des "Sliding-Window" Verfahrens.

von 5,76 $\frac{MBit}{s}$, welche UMTS in Verbindung mit HSUPA bietet, mindestens folgende Fenstergröße zwischenspeichern können muss:

$$5,76\frac{MBit}{s} * 300ms = 216kB$$
$$5,76\frac{MBit}{s} * 100ms = 72kB$$

Im optimalen Fall wird hier ein 72 Kilobyte großes Fenster benötigt, um die maximale Übertragungsgeschwindigkeit zu erreichen. Das Zwischenspeichern von 72 kB klingt in heutiger Zeit nach einer trivialen Aufgabe. Dies ist jedoch nicht der Fall bei einer Mikrocontroller-basierten Anwendung, denn Mikrocontroller besitzen typischerweise nur einen kleinen RAM-Speicher[8]. Dies sorgt dafür, dass die maximale Datenübertragungsrate nicht ohne weiteres in Kombination mit einem Mikrocontroller nutzbar ist.

3.3.3 Hypertext Transfer Protocol (HTTP)

Das Hypertext Transfer Protocol (HTTP) ist ein Übertragungsprotokoll und die Basis des Internets. Es ist der Anwendungsschicht im OSI-Schichtenmodel zuzuordnen. Grundlegend werden bei HTTP zwei Nachrichtenarten unterschieden. Zum einen die Anfrage, welche der Client dem Server stellt, und die Antwort, die der Server zurücksendet. Jede Nachricht wiederum lässt sich in zwei Teile spalten. Der eine Teil ist der Header, welcher mindestens die HTTP-Request-Methode, die angeforderte Ressource und den Hostnamen enthält. Der andere Teil wird als Body bezeichnet, welcher die Nutzdaten enthält. Typischerweise ist HTTP verbindungsorientiert und verwendet somit TCP als Übertragungsprotokoll.

[8] Hierfür gibt es zwei Gründe. Zum einen wird für die meisten Anwendungen nicht mehr benötigt. Zum anderen ist der Mikrocontroller im Preis umso günstiger desto weniger Speicher dieser verwendet.

In den meisten Fällen werden nur zwei HTTP-Request-Methoden verwendet:

GET Fordert eine Ressource vom Server an. Dies ist in den meisten Fällen eine HTML-basierte Webseite.

POST Sendet Daten, welche im Body stehen, an den Server. Ein typisches Beispiel hierfür ist das Senden von HTML-Formular Daten.

Zusätzlich können weitere HTTP-Request-Methoden verwendet werden, wie zum Beispiel PUT und DELETE, die für manche Anwendungsfälle benötigt werden ([Fielding u. a. (1999)] Kapitel 9).

Der Server sendet auf jede Anfrage eine Antwort mit einen HTTP-Response Code, welcher Aufschluss über Erfolg oder Fehlschlag der Anfrage gibt. Zusätzlich enthält der Body Nutzdaten, falls eine Ressource angefordert wurde und die Anfrage erfolgreich war.

Abschließend ein Beispiel einer GET-Anfrage, welche http://www.spiegel.de/politik/ anfordert:

```
GET /politik/ HTTP/1.1
Host: www.spiegel.de
```

3.4 REST Webservices

Neben den klassischen SOAP-basierten Webservices haben die sogenannten REST Webservices zunehmend an Beliebtheit gewonnen. REST, kurz für Representational State Transfer, ist kein Protokoll sondern ein Architekturmodell für die einfache Bereitstellung eines Webservices und kann durch folgende Punkte charakterisiert werden:

Eine solche Anwendung verwaltet Ressourcen. Die einzige Einschränkung die für Ressourcen gilt, ist, dass diese per URL abrufbar sein müssen. Ansonsten kann eine Ressource jede denkbare Art von Information darstellen. Als Schnittstellen bietet ein REST Webservices ausschließlich die HTTP-Request-Methoden an. Durch die Verwendung des HTTP-Protokolls ist die Kommunikation zwischen Client und Server zustandslos, genauso wie das HTTP-Protokoll selbst. Dies bedeutet eine Anfrage an den Server muss immer alle benötigten Informationen beinhalten.

Hält eine Anwendung die Prinzipien der REST Architektur ein, so kann diese 'RESTful' genannt werden (vgl. [Melzer (2010)] Kapitel 5.11.2).

Folgendes Beispiel soll das theoretische Prinzip eines REST Webservices verdeutlichen.

http://www.shop.xy/produkte

GET	POST	PUT	DELETE
Liefert eine Liste aller Produkte	Erstellt ein neues Produkt	-	-

http://www.shop.xy/produkte/2348

GET	POST	PUT	DELETE
Auslesen der Produkt-informationen	-	Editiert das Produkt mit der ID 2348	Löscht das Produkt mit der ID 2348

Die HTTP-Request-Methode GET dient zum Abfragen von Informationen, POST zum Erstellen einer neuen Ressource, PUT zum Editieren und DELETE zum Löschen. Das Antwortformat auf diese Anfragen ist meistens HTML oder XML.

Kapitel 4

Verwandte Arbeiten und Lösungen zur drahtlosen Messwertübertragung

In diesem Kapitel werden mögliche Ansätze zur Lösung der Aufgabenstellung gezeigt, sowohl aus dem akademischen Bereich als auch Lösungen der Wirtschaft. Daraufhin folgt eine kurze Evaluierung der Methoden in den vorgestellten Arbeiten und Lösungen.

4.1 Wissenschaftliche Ergebnisse

Eine Vielzahl von Forschungsarbeiten, welche die unterschiedlichsten Zielsetzungen besitzen, lassen sich in das Gebiet der Machine-to-Machine Kommunikation einordnen. Ein wichtiges Kriterium zur abstrakten Unterscheidung ist die Wahl des Kommunikationsnetzes und dessen Struktur. Je nach Distanz zwischen Data-End-Point und Data-Integration-Point ergeben sich unterschiedliche Anforderungen an die Anwendung. In diesem Abschnitt werden nun zwei Artikel exemplarisch vorgestellt, welche ebenfalls, wie in der Zielsetzung gefordert, größere Distanzen bei der Datenübertragung überbrücken müssen.

4.1.1 Embedded Wireless Network Control System

Das Team um Xiuhong Li stellte 2006 auf der IMACS Multikonferenz ein System zur Fernüberwachung von Umweltfaktoren in Gewächshäusern vor (vgl. [Li u. a. (2006)]). Ähnlich dem in Kapitel 2.1.1 vorgestelltem Szenario, benötigte man in den weitläufigen Landgegenden Chinas eine kostengünstige Möglichkeit Umgebungsinformationen echtzeitnah zu sammeln, um so die Produktivität zu steigern.

Die Umsetzung ist ähnlich dem Machine-to-Machine Konzeptes. Es wurde mindestens ein Mikrocontroller und ein Server verwendet. Der Mikrocontroller hat die Aufgabe mittels Analog-digital-Wandler aktuelle Daten von den angeschlossenen Sensoren auszulesen und diese via GPRS an den Server zu versenden. Für die GPRS-Verbindung wird ein per RS-232 angeschlossenes Modem genutzt. Um mehrere externe Datenquellen performant verwenden zu können, als auch die Möglichkeit offen zu lassen das System später einfach zu erweitern, wurde das proprietäre Echtzeitbetriebssystem µC / OS-II von der Firma Micriμm[9] verwendet.

Die Datenübertragung zwischen Mikrocontroller und Server erfolgt über eine TCP-Verbindung. Der bekannte TCP/IP Stack uIP findet Verwendung auf der Seite des Mikrocontrollers. Die Gegenstelle der Verbindung, der Server, bietet einen durch das Internet erreichbaren TCP Socket an. Der Server hat zu dem die Aufgabe die Sensordaten als Webseite, unter Verwendung von ASP.NET, für die Benutzer zu veröffentlichen.

4.1.2 A Mobile GPRS-Sensors Array for Air Pollution Monitoring

Im Jahr 2010 veröffentlichten A. R. Al-Ali und dessen Kollegen einen Artikel über eine Anwendung zur echtzeitnahen Messung der Luftverschmutzung aus der Ferne (vgl. [Al-Ali u. a. (2010)]). Das Ergebnis war ein System, welches mobil den Verschmutzungsgrad der Luft mit Sensoren misst und diese Messdaten samt aktueller Position periodisch an einen zentralen Server übermittelt. Das Serversystem, welches über das Internet ansprechbar ist, hat die Aufgabe, den aktuellen Grad der Luftverschmutzung für die Nutzer durch eine Webseite darzustellen und daneben die Daten persistent zu speichern.

Für das mobile System wurde ein 16-Bit-Mikrocontroller und ein industrielles GPRS-Modem verwendet, welches die bekanntesten Vertreter der Internetprotokolle zur Verfügung stellt. Zusätzlich kam ein GPS-Modul zum Einsatz, welches genauso wie das Modem via RS-232 mit dem Mikrocontroller verbunden ist. Der Verschmutzungsgrad der Luft wird mit Kohlenstoffmonoxid, Stickstoffdioxid und Schwefeldioxid Sensoren gemessen.

Das Serversystem besteht hier aus einem MySQL-Datenbankserver und einen Windows-basierten Apache-Webserver (WAMP[10]). Die Sensordaten des mobilen Systems werden durch ein Java Programm mit Hilfe von TCP/IP Berkeley Sockets empfangen. Die Luftverschmutzung wird danach entweder als Karte mithilfe der GPS-Daten und Google Maps dargestellt oder als Graph. Die Umsetzung erfolgt mit der Skriptsprache PHP.

Getestet wurde das System in der Stadt Schardscha[11], indem das mobile System auf einem Bus angebracht wurde. Dort sendete es dann zuverlässig für 12 Stunden Daten aus unter-

[9] http://micrium.com
[10] http://www.wampserver.com/
[11] Eines der sieben Vereinigten Arabischen Emirate

schiedlichen Gebieten der Stadt über den Verschmutzungsgrad der Luft an den zentralen Server.

4.2 Kommerzielle Lösungen

Durch den stetig wachsenden und sehr breit aufgestellten Markt der Machine-to-Machine Kommunikation findet man natürlich auch diverse Hersteller und Systemhäuser, die Komplettlösungen anbieten. Hier sind beispielhaft folgende Produkte auf dem deutschen Markt zu nennen:

- e-ControlNet[12] - Avantgarde Business Solutions GmbH
- CenterSight[13] - Device Insight GmbH
- GPRS Wireless Connect Professional[14] - ConiuGo GmbH

Die kommerziellen Lösungen weisen sehr oft Ähnlichkeiten mit den zuvor genannten akademischen auf. Die Server stellen die Daten normalerweise webbasiert für die Nutzer dar. Die Data-End-Points sind häufig Produkte, die einen Mikrocontroller oder einen Linux-basierten Einplatinen-Computer mit einem Industriemodem vereinen (siehe beispielgebend Abbildung 4.1).

Abbildung 4.1: GPRS Wireless Connect Professional von der Firma ConiuGo GmbH

Viele dieser Unternehmen bieten zudem Starterpakete an, die einen erschwinglichen Einstieg in den Machine-to-Machine Bereich bieten. Durch ein solches Paket wäre auf jeden Fall Szenario A der Aufgabenstellung umsetzbar.

[12] http://www.e-ControlNet.de
[13] http://www.device-insight.com/de/centersight-plattform.html
[14] http://www.coniugo.com/html_e/e_gsm_transmitter.html

Klarer Vorteil dieser Lösungen ist, dass diese dafür ausgelegt sind, schnell und einfach in Betrieb genommen zu werden. Auch serverseitig steht ein breites Spektrum an Diensten zur Verfügung, die gegen eine monatliche Gebühr genutzt werden können. Dies kann vorteilhaft sein, da man sich dann als Nutzer normalerweise keine Gedanken machen muss bezüglich des Servers in Hinsicht auf Konnektivität, Sicherheit und Ähnlichem.

Nachteilig anzusehen ist dagegen, dass diese kommerziellen Lösungen im Allgemeinen deutlich teurer sind als die angestrebte Lösung. Des Weiteren entsprechen diese Starter-Produkte oft nicht 100% den gewünschten Anforderungen, wie zum Beispiel dem Wunsch nach Mobilität. Der in Abbildung 4.1 vorgestellte Transmitter erfordert laut Datenblatt ein Steckernetzteil, womit man nur mit weiteren Modifikationen zu einer mobilen Lösung kommen würde.

4.3 Zusammenfassung

In diesem Kapitel wurden mehrere mögliche Ansätze vorgestellt, um die vorgebende Aufgabenstellung umzusetzen. Interessant zu sehen ist, dass sich im Abstrakten alle Lösungen gewissermaßen ähneln. Im Detail unterscheiden sich diese Lösungen jedoch, sodass die im kommerziellen Bereich mangelnde Flexibilität und der höhere Kostenfaktor, welcher gerade für Einsteiger in das Machine-to-Machine Gebiet gegeben ist, Motivation für eine eigene Lösung bietet.

Betrachtet man die in Kapitel 4.1 vorgestellten Arbeiten so zeigen diese in Hinblick auf die Zielsetzung dieser Arbeit noch offene Fragen auf und bieten auch Kritikpunkte. Die Frage, wie beispielsweise die Energieversorgung umgesetzt werden soll, wird von beiden Arbeiten nicht beantwortet. Die Anwendung zur Messung der Luftverschmutzung führt zwar einen möglichst geringen Energieverbrauch als Anforderung auf, gibt aber weiter keine Auskünfte wie und wie gut dieses umgesetzt wurde.

Im folgenden Kapitel wird die Architektur zusammen mit den benötigten Komponenten gewählt, die eine Realisierung des in Kapitel2 vorgestellten Konzepts ermöglicht, wobei sich die Auswahl auf die in diesem Kapitel gewonnene Wissensgrundlage stützen wird.

Kapitel 5

Evaluierung der Systemkomponenten

Bei erneuter Betrachtung des Konzeptes kommt man zu dem Schluss, dass sich diese Anwendung ebenfalls in das Gebiet der Machine-to-Machine Anwendungen einordnen lässt. Das vorhergegangene Kapitel veranschaulichte zudem ein abstraktes Konzept für Machine-to-Machine Anwendungen. Es ist daher sinnvoll dieses Konzept ebenfalls zu verwenden.

Abbildung 5.1: Die drei Grundelemente der Anwendung entsprechend der M2M-Architektur.

Dadurch lässt sich die geplante Anwendung durch die drei Grundelemente beschreiben, welche in Abbildung 5.1 gezeigt werden. Die Namen der Elemente werden entsprechend des vorgestellten Konzeptes beibehalten, wobei jedoch die einzelnen Komponenten jeweils den Komponenten der M2M-Architektur entsprechen. Dies bedeutet, die Sensorstationen entsprechen den Data-End-Points und der Server dem Data-Integration-Point.

Neben diesen drei dargestellten Grundelementen haben alle bisher gezeigten Lösungsansätze des vorherigen Kapitels eine weitere Gemeinsamkeit, nämlich das der Server webbasiert ist. Diese Entscheidung scheint auch im Hinblick auf die Zielsetzung dieser Arbeit sehr sinnvoll, denn dadurch kann der Nutzer sehr komfortabel aus aller Welt auf die Sensorinformationen zugreifen. Ein weiterer Vorteil ist, dass durch diese Vorgehensweise keine besonderen Anforderungen an die Benutzer gestellt werden. Mit diesem Argument kann man bereits jetzt festlegen, dass der Server webbasiert sein soll.

In dem weiteren Verlauf dieses Kapitels werden nun die einzelnen benötigten Komponententypen für diese Anwendung evaluiert und anschließend gewählt.

5.1 Das Kommunikationsnetz

Jede Machine-to-Machine Anwendung benötigt ein Kommunikationsnetz, um die Verbindung zwischen den Sensorstationen und dem Server herzustellen.

5.1.1 Auswahl des Kommunikationsnetzes

Wie schon in Kapitel 2.1.3 thematisiert müssen aus der Auswahl Funktechnologien, welche auf kurze Distanzen ausgelegt sind, für diese Anwendung ausgeschlossen werden. Mit dem Wegfall von Protokollen wie ZigBee, Bluetooth, RFID oder W-LAN bleibt nur noch das Mobilfunknetz übrig, wenn man im Hinterkopf behält, dass die angestrebte Lösung zugleich kostengünstig als auch energieeffizient sein soll.

Über das Mobilfunknetz stehen unterschiedliche Datendienste zur Verfügung, die nachfolgend beschrieben werden:

5.1.1.1 Short Message Service

Der Short Message Service (SMS) ermöglicht den weltweiten Versand von Textnachrichten in alle Mobilfunknetze mit maximal 140 Byte Nutzdaten. Versendet werden SMS mit dem in Kapitel 3.2.3 vorgestellten AT-Befehlssatz. Zum Empfangen der SMS auf der Serverseite benötigt man normalerweise ein SMS Gateway[15], welches weitere Kosten verursacht. Als positive Argumente für den Short Message Service zu nennen sind die einfache Identifizierung der Endpunkte (anhand der Telefonnummer) und die Möglichkeit, dass beide Seiten die Kommunikation initiieren können.

5.1.1.2 Mobiles Internet

Mobiles Internet basiert heutzutage auf 3 Zugangstechnologien[16], welche je nach Ort und Mobilfunkprovider verfügbar sind:

[15] Ist mit der Kurzmitteilungszentrale (SMSC) des Mobilfunk-Providers verbunden. und bietet als Dienstleistung beispielsweise den Aufruf von Webservice-Schnittstellen der eigenen Anwendung, wenn eine SMS empfangen wird.

[16] Da LTE noch nicht marktreif zum Abgabedatum dieser Arbeit ist, wird diese Technologie in dieser Arbeit nicht betrachtet.

GPRS ist mit einer Upload-Geschwindigkeit von circa 40 $\frac{kbit}{s}$ die langsamste Technologie.

EGDE verbessert GPRS und steigert die Datenrate auf circa 118 $\frac{kbit}{s}$.

UMTS erreicht mit dessen Erweiterung HSUPA bei entsprechender Hardware in Deutschland eine Upload-Geschwindigkeit von bis zu 5,76 $\frac{Mbit}{s}$[17]. Die Angaben über die Netzabdeckung in Deutschland in Prozent variiereb stark, sie dürfte aber deutlich über 50% liegen, selbst außerhalb von Ballungsgebieten.

Im Vergleich mit einer SMS-basierten Lösung sind folgende Vor- und Nachteile erkennbar:

Vorteilhaft bei UMTS sind die hohen Datenraten, durch die große Datenmengen fast überall in Deutschland günstig und schnell übertragen werden können. Da die Verbindung hier auf standardisierten IP-basierten Protokollen basiert wird eine robuste und effiziente Verbindung ermöglicht.

Neutral anzusehen ist die wesentlich höhere Komplexität auf Seite der Sensorstationen durch die eben genannten Protokolle, denn diese sind auf einem Mikrocontroller standardmäßig nicht vorhanden. Dafür kann die Serverseite als auch die Schnittstelle zwischen den Sensorstationen und dem Server wesentlich flexibler gestaltet werden und benötigt zudem keine Zusatzkomponenten.

Als Nachteil zu nennen ist natürlich die gestiegene Übertragungszeit[18] der Anwendung, welche allein schon durch die Einwahl in das Internet zustande kommt. Dadurch steigt der Energieverbrauch stark im Vergleich zu einer SMS-basierten Lösung. Zusätzlich kann der Verbindungsaufbau zur Kommunikation nur noch von der Sensorstation ausgehen, aus folgendem Grund:

Mobile Geräte erhalten meist bei jedem Verbindungsaufbau ins Internet (PPP) eine IP-Adresse aus einem privaten Subnetz des Internet-Service-Providers (ISP) zugewiesen. Der Internetzugriff geschieht via ISP per Network Address Translation (NAT), somit ist es ohne bestehende Verbindung nicht möglich mit einer Sensorstation zu kommunizieren.

5.1.1.3 Fazit

Die Betrachtung der Energieeffizienz und die Möglichkeit des bi-direktionalen Verbindungsaufbaus zeigen Vorteile beim Short Message Service auf. Dennoch überwiegen die Vorteile des mobilen Internets mit UMTS als Lösung für diese Anwendung:

[17] In dieser Arbeit wird der Begriff UMTS umgangssprachlich angesehen und schließt somit dessen Erweiterungen, wie zum Beispiel HSUPA, mit ein.

[18] Dieser Vergleich bezieht sich auf kleine Datenmengen, die auch ohne Weiteres per SMS übertragbar sind.

Momentan sind sowohl die Kosten einer UMTS-Flatrate als auch die einer SMS-Flatrate äquivalent und bei circa 10€ pro Monat anzusiedeln[19].Bei einer SMS basierten Lösung würde höchst wahrscheinlich noch ein SMS-Gateway benötigt werden, das weitere monatliche Kosten verursacht.

Eine hohe Datenrate bei Übertragungen als auch ein großes Fenster für Nutzdaten ermöglicht es, die spätere Anwendung wesentlich flexibler für unterschiedliche Anwendungsszenarien einzusetzen. Zudem wäre das in Kapitel 2.1.2 beschriebene Szenario B, in dem es um die Übertragung von Bilddaten geht, mit einer SMS-basierten Lösung nicht möglich.

Aus diesem Grund wird in dieser Arbeit das Internet als Kommunikationsnetz verwendet. Auf Seite der Sensorstation kommt UMTS zum Einsatz. Eine Internetverbindung beim Server wird als unproblematisch in heutiger Zeit und damit als gegeben erachtet.

5.1.2 Wahl des Kommunikationsprotokolls

Nach der Entscheidung das Internet als Basis für die Kommunikation zwischen den Sensorstationen und dem Server zu wählen, fehlt noch die Wahl des Übertragungsprotokolls. Es stellt sich die Frage, ob ein Anwendungsprotokoll implementiert werden soll und falls nicht, ob ein TCP- oder ein UDP-Socket gewählt wird.

Beide akademischen Arbeiten, die in Kapitel 4 vorgestellt wurden, verwenden serverseitig zum Empfang von Sensordaten Anwendungsprotokoll-unabhängige TCP-Sockets. Dies hat den Vorteil, dass diese sehr einfach zu implementieren sind. Nachteilig anzusehen ist jedoch bei dieser Lösung ein erhöhter Aufwand auf der webbasierten Serverseite, da das Öffnen eines benutzerdefinierten TCP-Sockets normalerweise nicht mit einer Webanwendung umsetzbar ist. Dies bedeutet, dass ein weiteres Programm notwendig ist für den Empfang der Sensordaten und der Abspeicherung dieser in der Datenbank.

Als Lösung wäre auch ein auf UDP-basierender Anwendungsprotokoll-unabhängiger Socket denkbar. Dieser würde aber serverseitig genau das gleiche Problem verursachen wie sein stream-basiertes Gegenstück. Trotz des höheren Datendurchsatzes aufgrund des geringeren Protokoll Overheads sorgt das UDP-Protokoll zudem dafür, dass die Verbindung unzuverlässig wird. Diese Unzuverlässigkeit ist hier inakzeptabel dadurch, dass Messwerte nur alle X-Minuten übertragen werden[20]. Aus diesem Grund ist die Verwendung von UDP auszuschließen.

Vorteilhafter ist eine Lösung, die mit einer einzelnen Anwendung sowohl den Empfang der Sensordaten als auch die Visualisierung dieser Daten abdeckt. Da der Server ohnehin eine

[19] Für Geschäftskunden existieren momentan bei allen großen Mobilfunkbetreibern gesonderte Tarife speziell für die Machine-to-Machine Kommunikation, die unter Umständen günstiger sein können je nach Anwendung.

[20] Den Beweggrund für dieses Vorgehen legt der übernächste Abschnitt dar.

Webanwendung beherbergen wird und somit schlussfolgernd das HTTP-Protokoll verwendet, empfiehlt es sich, dieses Protokoll ebenfalls für den Datenversand zwischen den Sensorstationen und dem Server einzusetzen. Die Datenübertragung mittels HTTP ist wie folgt möglich:

Eine Sensorstation sendet eine HTTP-POST Anfrage an den Server, wobei der Body der Anfrage die Sensorinformationen in einem zuvor festgelegten Format enthält. Der Server empfängt dann diese Anfrage und verarbeitet dessen Inhalt. Zudem kann der Server mittels seiner Antwort auf die Anfrage die Sensorstation über Erfolg oder Fehlschlag informieren.

Diese Vorgehensweise sorgt zugleich dafür, dass die Datenübertragung verbindungsorientiert ist, da HTTP auf das TCP-Protokoll aufbaut. Betrachtet man das HTTP-Protokoll etwas genauer, zeigt sich zudem, dass der zusätzliche Overhead für die Übertragung mit wenigen Bytes minimal ist und zugleich eine maximale Übertragungsgröße nicht definiert ist[21].

5.1.3 Die zu übertragenden Daten und deren Format

Die zu übertragenden Sensorinformationen wurden schon vielfach genannt, aber es wurde noch nicht definiert, was man darunter genau zu verstehen hat im Sinne dieser Anwendung. Zunächst ist damit das Offensichtliche gemeint, nämlich ein Messwert eines Sensors. Da das Konzept aber beabsichtigt, dass eine Sensorstation mehrere Sensoren besitzt und die Gesamtanwendung mehrere Sensorstationen aufweist. Dieser Sachverhalt sorgt für ein Identifikationsproblem.

Dieses Problem ist jedoch lösbar, indem man zu jedem Messwert zusätzlich den Namen des Sensors und den der Sensorstation überträgt. Mit dieser Lösung kann jeder Messwert eindeutig identifiziert werden, wobei diese Lösung zugleich auch eine neue Problemstellung aufzeigt, nämlich die der Namensgebung der Sensorstation und der jeweiligen Sensoren. Aus diesem Grund wird die Namensgebung für die Sensorstationen wie folgt festgelegt:

Jede Sensorstation benötigt einen konstanten und eindeutigen Namen oder eine Identifikationsnummer, welche jeweils per Hand gewählt werden muss. Die Sensoren einer Sensorstation müssen jeweils auch einen eindeutigen Namen besitzen im Kontext der Sensorstation. Dieser lässt sich normalerweise automatisch generieren, wie zum Beispiel über den genutzten I/O-Pin. Zusätzlich muss jeder Sensor, einen vorher auf der Serverseite festgelegten Typ mitsenden, da ansonsten die unterschiedlichen Messwerte nicht entsprechend visualisiert werden können. Ein solcher Typ fast Sensoren des gleichen Typs zusammen und bietet eine Angabe darüber, ob sich der jeweilige Messwert in einem kritischen Bereich befindet. Da die generierten Namen zudem oft nicht aussagekräftig sind, muss der Server zudem die Möglichkeit bieten, dass diese lokal für die Visualisierung umbenannt werden können.

[21] Die maximale HTTP-POST Übertragungsgröße kann durch Einstellungen des Webservern limitiert sein.

Der Body-Bereich einer HTTP-POST Anfrage lässt offen in welchem Format die Nutzdaten übertragen werden und daher muss dieses selber gewählt werden. Bei Betrachtung der zuvor genannten Informationen, die übertragen werden müssen, lässt sich eine gewisse Struktur erahnen, wodurch eine XML-basierte Datenübertragung via HTTP als sinnvoll erscheint. Zusätzlich kann so ein XML-Format sehr leicht um Statusmeldungen der Sensorstationen wie beispielsweise die Batteriespannung erweitert werden. Ein passender XML-Standard[22] für diese Problemstellung existiert momentan nicht, sodass ein eignes Format im nächsten Kapitel definiert werden muss.

Abschließend muss noch erwähnt werden das die Datenübertragung und somit die hier genannten XML-Informationen laut Konzept verschlüsselt übertragen werden müssen.

5.2 Die Mikrocontroller-basierte Sensorstation

Die Zielsetzung fordert eine möglichst hohe Energieeffizienz bei den Mikrocontroller-basierten Sensorstationen. Diese Eigenschaft muss von vornherein bei der Wahl der Komponenten mitbedacht werden und beeinflusst zudem dessen Programmablauf.

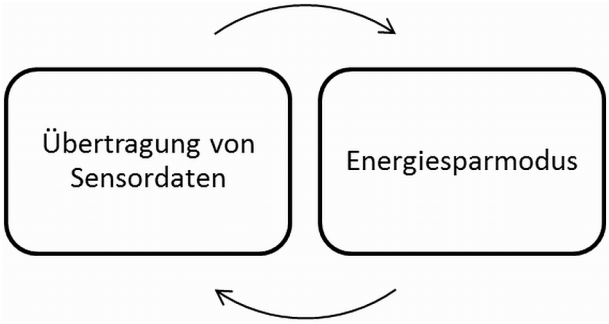

Abbildung 5.2: Abstrakter Programmablauf einer Sensorstation

Um den Energieverbrauch möglichst gering zu halten, ist folgender Ablauf erforderlich: Die Sensorstation befindet sich die meiste Zeit im Energiesparmodus, welcher dadurch gekennzeichnet ist, dass möglichst viele Strom verbrauchende Bauteile deaktiviert sind. Nach einer definierten Zeitspanne wechselt die Sensorstation in den aktiven Modus. In dieser Zeit liest sie die aktuellen Sensordaten ein und stellt eine Verbindung zum Server her. Sobald diese besteht, sendet sie die Sensordaten, trennt danach die Verbindung wieder und wechselt in den Energiesparmodus. Dies bedeutet, die Sensorstation wird periodisch zwischen einer Wach- und Schlafphase wechseln.

[22] Der quelloffene „Standard" M2MXML wurde als nicht passend für diese Anwendung erachtet.

Zudem müssen die Sensordaten bei jeder Wachphase übertragen werden, auch dann, wenn sich die Sensorinformationen seit der letzten Übertragung nicht geändert haben. Dies liegt daran, dass ansonsten auf der Serverseite nicht mehr festgestellt werden kann, ob eine Sensorstation ausgefallen ist.

5.2.1 Entscheidung für einen Modem-Typ

Auf der Suche nach einem günstigem UMTS-fähigem Modem für die Datenübertragung findet man momentan sehr schnell als Treffer UMTS Sticks. Als Alternative bieten sich noch Industriemodems an. Eingebettete UMTS Module hingegen, die teilweise auch schon in Oberklassefahrzeugen wie dem Audi A8 verbaut werden [Telit Wireless Solutions], können ausgeschlossen werden, da diese normalerweise nur in hoher Stückzahl abgenommen werden können.

Abbildung 5.3:
Links: O2 Surf Stick
Rechts: Industrielles UMTS Modem der Firma Maxon Australia

Die im Kapitel zuvor vorgestellten Lösungen verwendeten fast alle GPRS-Industriemodems. Ein solches Industriemodem ebenfalls für diese Anwendung hier zu verwenden ergibt zwei Nachteile. Zum einen wird die Flexibilität eingeschränkt, was sich durch das Szenario B (Kapitel 2.1.2) aufzeigen lässt. Dieser Anwendungsfall könnte beispielsweise fordern, dass periodisch zwei Megabyte Bilddaten übertragen werden müssen. Mit einer rein GPRS-basierten Übertragung würde die Dauer einer solchen Datenübertragung wesentlich länger dauern als mit UMTS aufgrund der niedrigeren Übertragungsgeschwindigkeit. Dadurch wiederum wird die Energieeffizienz stark negativ beeinflusst, wodurch man davon abraten müsste, dieses Szenario in Zusammenhang mit dieser Anwendung zu realisieren. Das zweite Argument gegen eine Industriemodem-basierte Lösung ist, dass die Preise dieser Modems durchschnitt-

lich bei 100€ starten. Bei Verwendung vieler Sensorstationen wird dies sehr schnell zu einer kostspieligen Angelegenheit, da jede Sensorstation ein Modem benötigt.

Mit dem Ziel eine kostengünstige Lösung zu schaffen, fällt die Entscheidung einen UMTS Stick zu verwenden nicht schwer. Insbesondere da UMTS Sticks heutzutage schon ab 10€ erhältlich sind. Zudem ist die USB-Schnittstelle bei modernen Mikrokontrollern heutzutage auch kein Hindernis mehr, wodurch der UMTS-Stick-Lösung technisch gesehen nichts im Weg steht.

Mit der Entscheidung UMTS Sticks zu verwenden, muss man natürlich auch Nachteile in Kauf nehmen. Da es sich bei diesen Sticks um Technik handelt, welche für den normalen Verbraucher gefertigt wurde, sind diese für eine geringere Temperaturspanne als Industriemodems ausgelegt[23]. Zudem finden sich normalerweise keine Datenblätter zu diesen Produkten, wie man es aus dem technischen Umfeld gewohnt ist. Problematisch zu sehen ist auch, dass der Markt für UMTS Sticks sehr dynamisch ist, wodurch nicht sichergestellt ist, dass der gewünschte UMTS Stick weiterhin in drei Monaten erhältlich ist.

Unterschiede bei dem maximalen Energieverbrauch sind im Normallfall zwischen UMTS Sticks und Industriemodems als gering anzusehen. Durch die USB-Spezifikation vorgegeben, kann ein USB-Gerät maximal 500 mA bei einer Versorgungsspannung von 5 Volt (vgl. [USB IF (2000)] Kapitel 7.3.2) für sich beanspruchen, was eine maximale Leistung von 2,5 Watt ergibt. Von diesem Verbrauch ist bei einem UMTS Stick auszugehen, da wie schon erwähnt keine Datenblätter zu diesen Produkten verfügbar sind. Industrielle Modems bieten dagegen in ihren Datenblätter Angaben zu den elektrischen Eigenschaften. Aus diesen ist meist eine ähnliche benötigte Leistung abzulesen.

Verwendete Modems in dieser Arbeit

Für die Sensorstationen wurde ein „O2 Surf Stick” (ZTE MF 190) zusammen mit einer UMTS Flat von Klarmobil[24] verwendet. Diese bietet ein mehr als ausreichendes monatliches Transfervolumen von 500 Megabyte zu einem monatlichen Preis von 10€. Um die Austauschbarkeit der UMTS Sticks zu beweisen, wurde zusätzlich der „XS Stick W14” des Hamburger Unternehmens 4G-Systems verwendet.

5.2.2 Benötigte Software-Komponenten

Um die Sensorstationen wie gewünscht zu realisieren, bedeutet dies, dass man neben dem UMTS Stick mindestens folgende Software-Komponenten benötigt:

[23] Vereinzelt findet man hierzu Angaben, die im Normalfall von -10 ° bis 55 ° Betriebstemperatur sprechen.
[24] http://www.klarmobil.de/tarife/datentarife/Internet_Flat_500/index.html

Ein USB-Stack wird benötigt, um den Stick softwaretechnisch ansprechen und verwenden zu können. Dieser hat explizit die Aufgabe der USB-Enumerierung und die darauf folgende Kommunikation zwischen dem UMTS Stick und dem Mikrocontroller abzuwickeln.

Darüber hinaus wird ein TCP/IP Stack benötigt, welcher im Idealfall folgende Protokolle enthält:

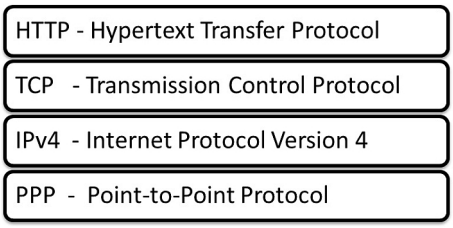

Abbildung 5.4: Benötigte Netzwerkprotokolle für den Betrieb einer Sensorstation.

PPP wird als Netzzugangsprotokoll benötigt und dient zur Kommunikation mit dem Mobilfunkprovider unter anderem bei der Einwahl in das Internet. HTTP wird, wie zuvor festgelegt, als Protokoll zur Kommunikation mit dem Server verwendet. TCP/IP wird grundlegend für die Vermittlung und die verbindungsorientierte Datenübertragung benötigt.

Zu guter Letzt steht noch folgende Frage im Raum. Benötigt die beabsichtigte Anwendung ein Echtzeitbetriebssystem (RTOS). Betrachtet man diese Fragestellung erst einmal allgemein ergeben sich folgende Vor- und Nachteile gegenüber einer Lösung, welche einen endlichen Automaten (FSM) verwendet. Als Vorteil zu nennen ist ganz klar die Möglichkeit parallele Tasks zu implementieren. Zudem wird durch ein Echtzeitbetriebssystem die Möglichkeit gegeben, eine bestehende Anwendung sehr einfach um neue Aufgaben zu erweitern. Nachteilig anzusehen ist dagegen, dass dadurch die Gesamtleistung der Anwendung sinkt und zugleich der Speicherverbrauch steigt. Dies liegt insbesondere daran, dass der Scheduler Laufzeit für sich beansprucht und die Kommunikation der Tasks untereinander synchronisiert werden muss, um Race Conditions zu vermeiden. Abschließend muss aber noch erwähnt werden, dass sich beide Lösungsansätze nicht gegenseitig ausschließen, was bedeutet, dass man diese durchaus auch kombinieren kann.

Begutachtet man nun den beschriebenen Ablauf der Mikrocontroller Anwendung (vgl. Abbildung 5.2) dann erscheint dieser Ablauf zunächst seriell und würde sich sehr gut durch eine Automaten-basierte Lösung umsetzen lassen. Dennoch empfiehlt sich auf den zweiten Blick die Verwendung eines Echtzeitbetriebssystems aus folgenden Gründen.

Zum einen soll die Anwendung möglichst kurze Wachphasen haben umso eine gute Energieeffizienz zu erzielen. Sollten Sensoren an die Sensorstation angeschlossen werden, welche die Eigenschaft haben, eine längere Zeit für eine akkurate Messung zu benötigen, dann würde sich die Zeit während der Wachphase deutlich erhöhen. Eine elegantere Lösung ist,

wenn sowohl der Einwahlvorgang in das Internet als auch das Auslesen der Sensoren parallel, im Sinne von präemptiven Multitasking, vonstattengeht. Dies wäre ohne ein Echtzeitbetriebssystem nur sehr schwierig zu realisieren. Der zweite Grund für ein RTOS entsteht aus der Zielsetzung der Arbeit, dass die Anwendung möglichst einfach erweiterbar sein soll, beispielsweise durch eine neue Art von Sensoren. Dies würde bei Verwendung eines Echtzeitbetriebssystems nur die Implementierung eines neuen Tasks bedeuten, ohne andere Codestücke erneut umschreiben zu müssen.

Aus diesen beiden Gründen wird ein Echtzeitbetriebssystem in dieser Arbeit verwendet. Dieses jedoch wird mit einem endlichen Automaten verknüpft, da der Sendevorgang sich nicht sinnvoll in mehrere parallele Tasks aufteilen lässt. Dadurch erscheint eine sehr performante Umsetzung der Anwendung möglich ohne große Kompromisse eingehen zu müssen.

5.3 Der Server

Bei der Vielfältigkeit des Internets ist es nicht es nicht verwunderlich, dass es ebenfalls eine Vielzahl an Technologien für dynamische Webanwendungen gibt. Beispielhaft zu nennen ist hier unter anderem ASP.NET, Java, PHP und Perl. Bei all diesen Technologien gibt es zudem nochmals eine Fülle an Frameworks, die sich wiederum im Detail unterscheiden. Man hat also die Qual der Wahl eine passende Basis für die serverseitige Webanwendung zu finden.

Betrachtet man erneut die Anforderungen, die die Zielsetzung fordert, so stellt man fest, dass diese meist sehr allgemein sind, wie beispielsweise, dass ein Datenbanksystem verwendet werden muss. Dadurch wird die Auswahl nicht sonderlich eingegrenzt. Die speziellste aller Forderungen an die zukünftige Webanwendung ist, dass es möglich sein muss, eine Funktion zu realisieren mit welcher Sensorinformationen via HTTP-POST empfangen werden können. Aber selbst diese Anforderung ist noch nicht außergewöhnlich, lässt aber einen auf die Idee kommen eine REST-basierte Webanwendung zu verwenden. Die Assoziation liegt nahe, da die in Kapitel 3.4 vorgestellten REST Webservices normalerweise nur die HTTP-Request-Methoden (GET, POST, usw.) als Schnittstellen verwenden.

Selbst mit der Wahl eine RESTful Webanwendung auf der Serverseite zu verwenden bleibt die bestehende Auswahl so groß, dass man die exakte Plattform nach persönlicher Präferenz wählen muss. Deswegen ist die Entscheidung gefallen eine Java-basierte Webanwendung für diese Arbeit zu wählen, die zudem RESTful ist. Java wurde gewählt aufgrund der hohen Plattform-Portabilität und der sehr großen Entwicklergemeinde.

5.4 Zusammenfassung

Die durch das Konzept beschriebene Anwendung wurde in diesem Kapitel als Machine-to-Machine Anwendung eingestuft. Als Folge dessen lässt sich die beabsichtigte Anwendung in drei Grundelemente aufteilen. Für jedes dieser Elemente wurden mögliche Lösungsansätze evaluiert. Im Anschluss wurde jeweils ein Fazit gezogen, aus welchem abstrakt die Komponenten hervorgehen, die für eine effiziente Lösung Verwendung finden sollen. Die Abbildung 5.5 zeigt das Resultat dieser Untersuchung und die grobe Struktur der zu realisierenden Anwendung:

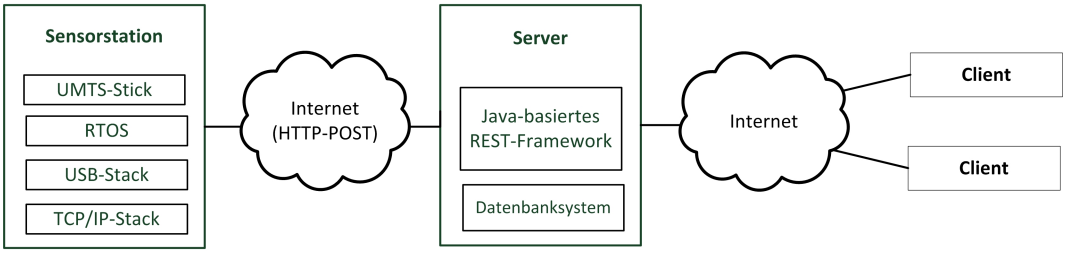

Abbildung 5.5: Struktur der beabsichtigen M2M-Anwendung

Die Datenübertragung zwischen den Sensorstationen und dem Server soll Internet-basiert via HTTP-POST realisiert werden. Die zu übertragenden Sensorinformationen sind in ein XML-Format zufassen, damit diese durch den Server zugeordnet werden können.

Für eine Mikrocontroller-basierte Sensorstation soll ein Echtzeitbetriebssystem zum Einsatz kommen, um möglichst einfach und effizient Sensoren hinzuzufügen und diese auszulesen. Als Modem soll ein handelsüblicher UMTS-Stick verwendet werden, welcher softwareseitig einen USB-Stack benötigt. Zusätzlich wird für die Datenübertragung ein TCP/IP-Stack benötigt.

Auf der Serverseite wird neben einem Datenbanksystem zur persistenten Speicherung der empfangenen Sensorinformationen, ein Java-basiertes Rest-Framework verwendet werden. Dieses stellt das Fundament für die beabsichtige Webanwendung dar.

Die Clients greifen ebenfalls über das Internet auf den Server zu und können dort die veranschaulichten Sensorinformationen schnell und effizient analysieren.

Kapitel 6

Design der M2M-Anwendung

In diesem Kapitel wird die Anwendung gestaltet und zugleich werden die konkreten Komponenten gewählt, die für Umsetzung der beabsichtigen Anwendung benötigt werden.

6.1 Design der Sensorstation

6.1.1 Architektur und Ablauf

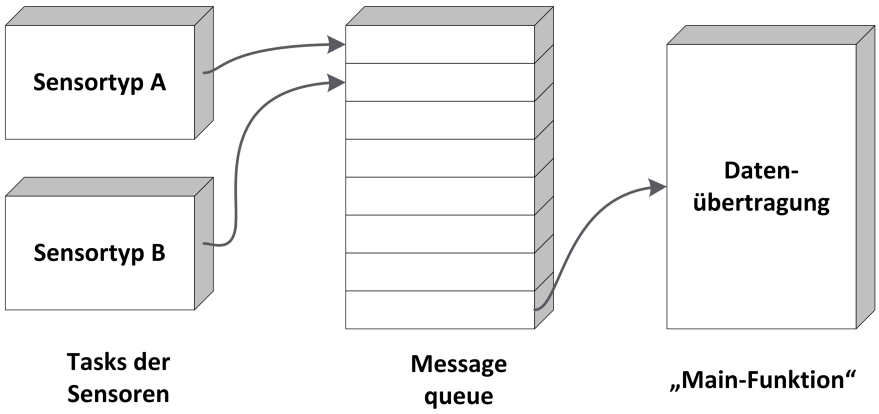

Abbildung 6.1: Architektur der Sensorstation

Die Verwendungsweise des Echtzeitbetriebssystems wurde schon im vorherigen Kapitel angedeutet. Die Abbildung 6.1 verdeutlicht diese Architektur nochmals, mit der die Zielsetzung verfolgt wird, die Sensorstationen mit minimalem Synchronisationsaufwand, hoher Modularität und Effizienz zu realisieren. Der nachfolgend beschriebener serielle Ablauf der Sensordatenübertragung ist hier in der Abbildung als „Main-Funktion” dargestellt und soll als ein im

System einzigartiger Task realisiert werden. Der Grund hierfür ist das sich der Vorgang nicht durch parallele Tasks beschleunigen lässt. Jeder Sensortyp soll zudem als eigener Task implementiert werden. Dies hat den Vorteil, dass man die Sensorstation sehr einfach um eine neue Art von Sensoren, welche jeweils ein anderes Protokoll zum Auslesen erfordern, erweitern kann, ohne den bisherigen Quellcode verändern zu müssen.

Die Kommunikation zwischen den einzelnen Sensortypen und dem Datenübertragungstask wird über eine synchronisierte Warteschlange („Message Queue") umgesetzt werden. Dies bedeutet, die Sensor-Tasks schreiben in jeder Wachphase die aktuellen Messwerte zusammen mit dessen Sensoridentifikation in die Warteschlange, wobei hier sichergestellt sein muss, dass die jeweiligen Messwerte OK sind. Der Datenübertragungstask entnimmt dann diese Informationen der Warteschlange und versendet diese an den Server. Dabei hat dieser Task gegeben durch die Architektur keine Kenntnisse, welche Sensoren in dem System vorhanden sind. Dies bedeutet, die Sensorstation kann nicht erkennen, ob ein Sensor ausgefallen ist. Diese Aufgabe obliegt dem Server, indem dieser prüft, ob von jedem bekannten Sensor regelmäßig neue Messwerte eintreffen. Durch diese Vorgehensweise wird die Komplexität auf der Seite der Sensorstation gesenkt und die Dauer der Wachphase verringert.

Der im vorherigen Absatz genannte Datenübertragungstask, welcher den Ablauf der Sensordatenübertragung beinhaltet, lässt sich mit dem Wissen welche Hardware- und Softwarekomponenten theoretisch verwendet werden sollen nun auch detaillierter als Zustandsdiagramm darstellen:

Die Abbildung 6.2 zeigt vereinfacht den Ablauf, der nötig ist, um Sensordaten mithilfe eines UMTS Sticks zu versenden. Nicht in der Abbildung gezeigt werden die Trigger der Transitionen, welche teilweise Event- als auch Lambda-basiert sind, und die Fehlerfälle, um den Ablauf möglichst übersichtlich darzustellen.

Der Prozess beginnt mit der Aktivierung des USB-Host-Controllers des Mikrocontrollers. Im Normalfall ist dieser mit dem UMTS Stick verbunden, was direkt zu der USB-Enumerierung des Sticks führt. In diesem Schritt kommt die in Kapitel 3.2.2 vorgestellte Modeswitch-Eigenschaft der UMTS Sticks zum Tragen. Falls der Stick sich als Massenspeichergerät bei der Sensorstation anmeldet, muss nach der Enumerierung der oder die Befehle gesendet werden zum Wechseln in den Modemmodus. Dies führt dazu, dass der Stick einen USB-Reset auslöst, was wiederum zur USB-Enumerierung führt. Bei dieser wird nun der Stick als Modem enumeriert und ist danach ansprechbar.

In dem nächsten Schritt muss der Stick konfiguriert werden. Hierbei muss beispielsweise der PIN eingegeben werden, um die SIM-Karte des UMTS Sticks freizuschalten. Dies wird mit AT-Befehlen bewerkstelligt, welche in dem Grundlagenkapitel 3.2.3 vorgestellt wurden. Nach der Konfiguration ist es notwendig das Einwahlkommando zu senden, um mit dem PPP-Verbindungsaufbau beginnen zu können. Nachdem dieser durchgeführt wurde und der Internet-Service-Provider der Sensorstation eine IP-Adresse zugewiesen hat, ist diese onli-

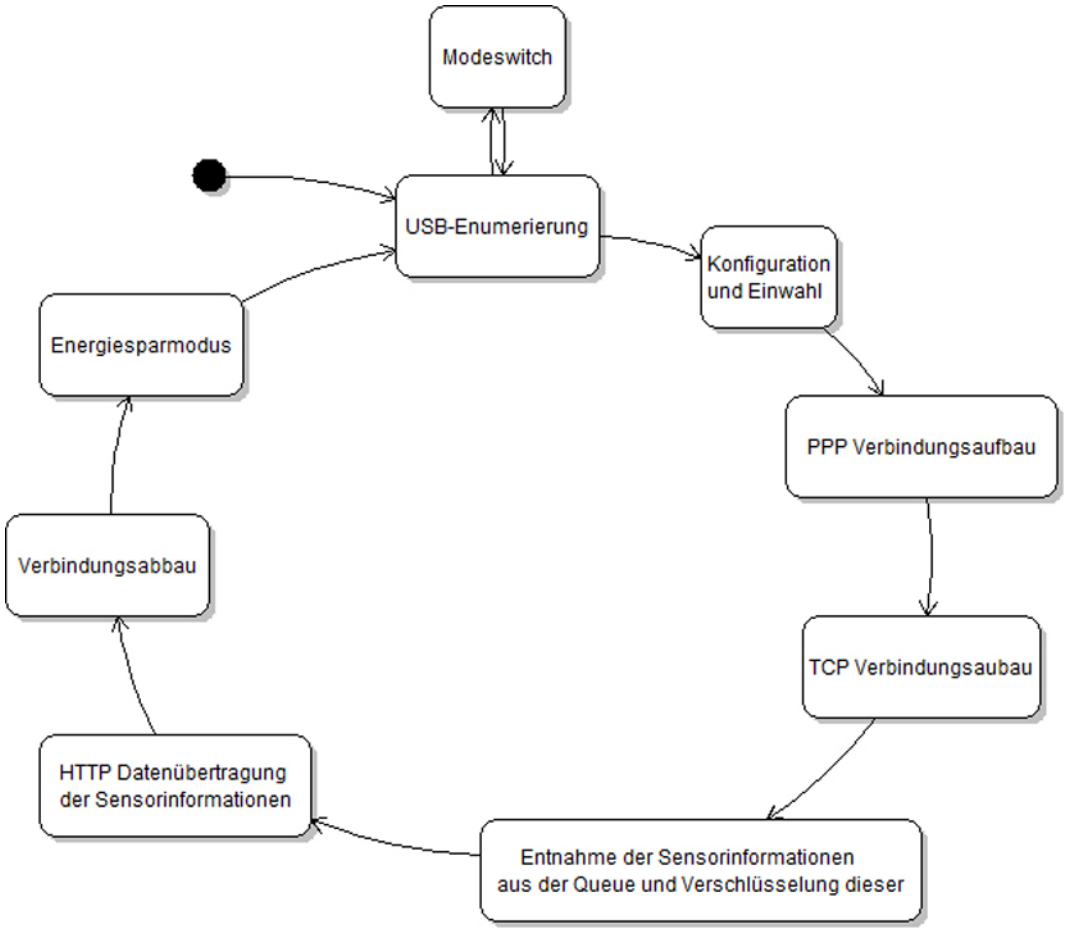

Abbildung 6.2: Vereinfachtes Zustandsdiagramm des Übertragungsvorgangs einer Sensorstation.

ne und kann mit der Sensordatenübertragung beginnen. Dazu wird eine TCP-Verbindung mit dem Server aufgebaut, da HTTP auf TCP basiert. Sobald diese erfolgreich hergestellt wurde, werden die bereitstehenden Sensorinformationen aus der Message Queue ausgelesen, verschlüsselt und mit einer einzigen HTTP-POST Anfrage, welche die verschlüsselten Informationen aller Sensoren im Body enthält, an den Server übertragen. Abschließend wird die TCP-Verbindung zum Server als auch die PPP-Internetverbindung beendet. Danach wird die Sensorstation in den Energiesparmodus versetzt, wodurch alle Hardware Komponenten der Sensorstation deaktiviert werden, was auch den UMTS Stick mit einschließt. Dieser Energiesparmodus gilt für die komplette Sensorstation, was bedeutet, dass auch die Tasks der Sensoren „schlafen”.

Einige Tests, die vorab durchgeführt wurden, haben Folgendes gezeigt. Ein UMTS Stick bei bestehender Internetverbindung, über welche keine Pakete versendet werden, zeigt trotzdem einen deutlichen Energieverbrauch auf. Aus diesem Grund ist es zwingend notwendig,

die USB-Schnittstelle während des Energiesparmodus zu deaktivieren. Dies aber hat jedoch zur Konsequenz, dass bei jeder Datenübertragung zuerst eine Internetverbindung via PPP hergestellt werden muss. Aus diesem Grund ergibt sich der in Abbildung 6.2 dargestellten Ablauf bei jeder Übertragung von Sensorinformationen.

Auf Fehlerfälle während dieses Ablaufes, beispielsweise wenn der Server nicht erreichbar ist, kann nur auf eine Weise reagiert werden, und zwar es später erneut zu versuchen. Dies bedeutet, dass jeder Fehler eine kontrollierte Transition in den Energiesparmodus-Zustand auslöst. Nach der Schlafphase beginnt der Sendezyklus erneut und ist hoffentlich fehlerfrei. Die Serverseite hat die Aufgabe festzustellen, dass eine Sensorstation ein Problem hat oder ausgefallen ist. Zusätzlich wird der Watchdog des Mikrocontrollers aktiviert, um den Programmfluss zu überwachen. Dies bedeutet, sollte man in einem Zustand für eine längere Zeit als erwartet stecken bleiben, weil beispielsweise Protokollantworten des ISP verloren gegangen sind, sorgt der Watchdog mit einem Reset für einen Neustart. Hierdurch wird sichergestellt, dass eine Sensorstation im Feld sich nicht aufhängen kann.

Um mit der Realisierung der Sensorstationen beginnen zu können, muss noch die Hardware Plattform, die konkreten Softwaremodule und das Datenübertragungsformat gewählt werden.

6.1.2 Die Hardware Plattform der Sensorstation

Die Cortex-M3 Architektur der Firma ARM ist eine moderne und aktuell sehr beliebte Mikrocontroller-Architektur. Diese vereint eine hohe Leistung mit einem geringen Energieverbrauch. Zudem wurde die Codedichte dank des verwendeten 16-Bit Thumb2-Befehlssatzes stark erhöht im Vergleich zu dem beliebten Vorgänger ARM7. Intern arbeitet der ARM Cortex-M3 weiterhin mit einem 32-Bit-Befehlssatz genauso wie alle anderen alle ARM-Architekturen. Obendrein sind Cortex-M3 basierte Mikrocontroller sehr günstig[25] erwerbbar und werden von einer Vielzahl von Entwicklungstools und Herstellern unterstützt. [vgl. Yiu (2009) Kapitel 1]

Aufgrund dieser Eigenschaften eignet sich ein ARM Cortex-M3 basierter Mikrocontroller hervorragend für diese Aufgabenstellung. Für eine effiziente Umsetzung eines ersten Prototyps der Sensorstation wurde ein Development-Board des Unternehmens Texas Instruments verwendet. Dieses Board ist in dem „Stellaris® LM3S9B96 Development Kit” enthalten und bietet eine Vielzahl von Komponenten. Das Blockschaltbild (Abbildung 6.3) zeigt diese auf[26] , wobei folgende besonders für die Umsetzung dieser Arbeit hervorzuheben sind:

[25] Teilweise ab einem Dollar erhältlich.

[26] Die 8 MB SDRAM befinden sich auf einem Erweiterungsboard, welches in dieser Arbeit nicht verwendet wurde.

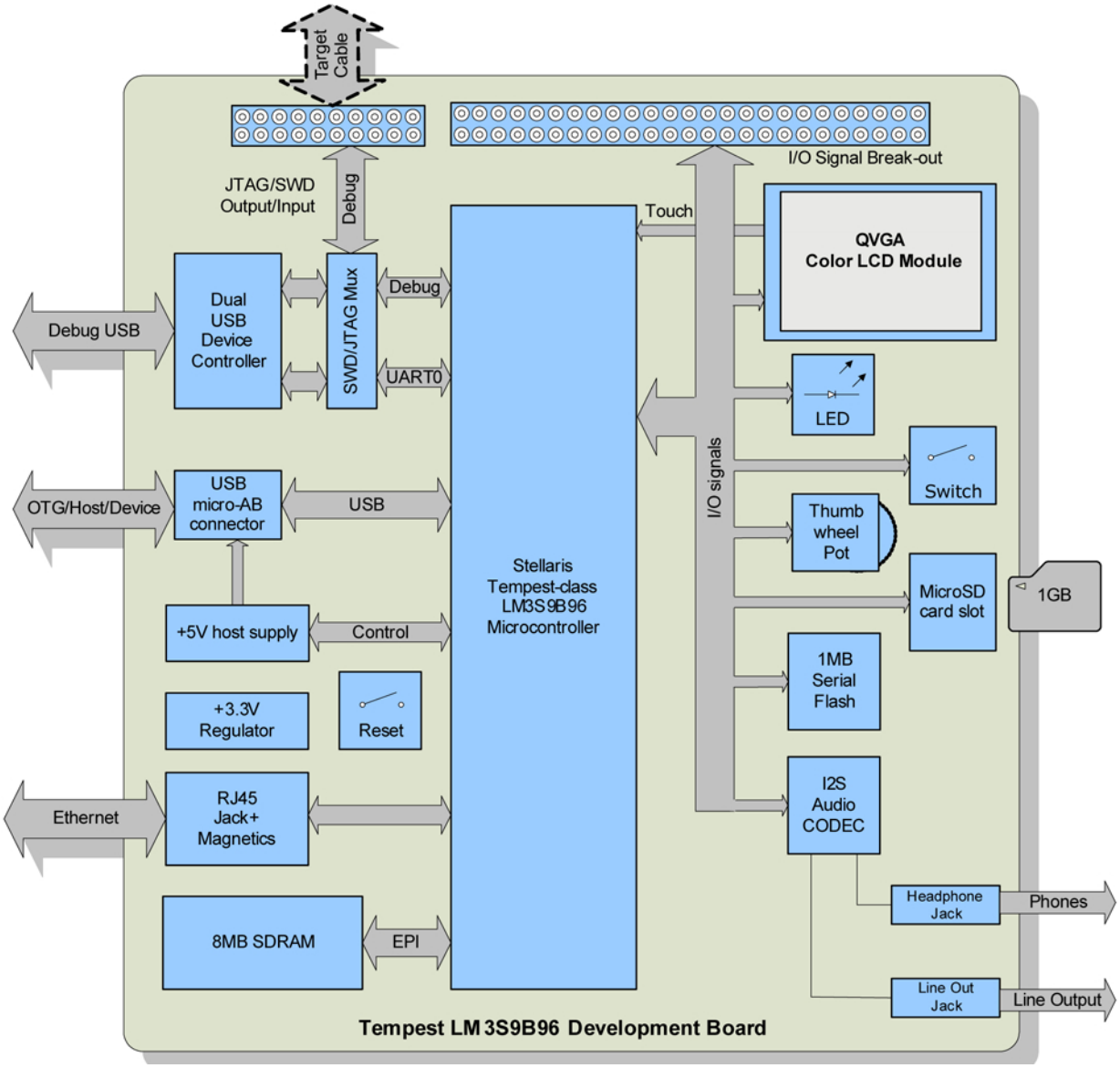

Abbildung 6.3: Blockschaltbild des LM3S9B96 Development-Boards

- 80 MHz LM3S9B96 Mikrocontroller
  - ARM Cortex-M3 Prozessor
  - 256 kB FLASH
  - 96 kB SRAM
- USB Controller mit Device, Host und On-The-Go Unterstützung
- Integrated In-circuit Debug Interface (ICDI)

6.1.2.1 Einschub: Entwicklungsumgebung der Sensorstation

Neben der freien GNU Compiler Collection[27] (GCC) bieten viele Unternehmen kommerzielle Entwicklungsumgebungen und Compiler für ARM Cortex-M3 Mikrocontroller an. Die kommerziellen Lösungen haben den Vorteil, dass man die unterstützen Mikrocontroller sehr schnell und einfach in Betrieb nehmen kann, da die meisten problematischen Ecken automatisch behandelt werden und zusätzlich Supportleistungen angeboten werden. Nachteilig anzusehen ist, dass diese Entwicklungsumgebungen und Compiler teilweise nur mit meist sehr hohen Lizenzkosten beziehbar sind. Des Weiteren wird durch Verwendung von proprietären Compilern[28], wie beispielsweise den Texas Instruments Code Generation Tools, die Möglichkeit die Anwendung auf andere Hardwareplattformen zu Portieren stark eingeschränkt. GCC unterstützt dagegen eine große Anzahl an unterschiedlichen Zielplattformen.

Aus diesen Gründen wird für diese Arbeit die GNU Compiler Collection verwendet trotz des erhöhten Aufwands beim Aufstellen der Toolchain. Genau genommen wurde die YAGARTO GNU ARM toolchain[29] für die Programmierung der Sensorstationen verwendet. Als Entwicklungsumgebung kam Eclipse CDT[30] zum Einsatz, unter anderem, da es sich durch viele hilfreiche Erweiterungen ergänzen lässt. Folgende Erweiterungen sollten hier genannt werden:

GNU ARM Eclipse Plug-In[31] Vereinfacht die Verwendung der YAGARTO Toolchain zusammen mit Eclipse

Zylin Embedded CDT[32] Ein GDB Debugging-Plug-In

STATEVIEWER Plug-In[33] Visualisiert alle laufenden Tasks und deren Zustände von FreeRTOS[34] während des Debuggens, wenn der Programmablauf pausiert ist.

Der freie On-Chip Debugger OpenOCD[35] wurde für das Flashen und Debuggen des Mikrocontrollers verwendet. Als Terminal für die UART-Debugausgaben des Mikrocontrollers kommt HTerm[36] zum Einsatz.

6.1.3 Das Echtzeitbetriebssystem

Die Sensorstation benötigt ein Echtzeitbetriebssystem an das nur diese einfachen Anforderungen gestellt werden:

[27] Enthält unter anderem neben dem gleichnamigen Compiler und Linker GCC den Debugger GDB.
[28] Aus diesem Argument sind proprietäre GCC-basierte Compiler ausgenommen.
[29] http://www.yagarto.de/
[30] http://www.eclipse.org/cdt/
[34] Neben FreeRTOS wird SafeRTOS und OpenRTOS unterstützt.
[35] http://openocd.berlios.de/web/
[36] http://www.der-hammer.info/terminal/

- Unterstützung der ARM Cortex-M3 Architektur
- Untersteht einer Open-Source-Lizenz
- „Message Queues" müssen als Synchronisationsmittel vorhanden sein.

Für diese Arbeit wurde FreeRTOS[37] als Echtzeitbetriebssystem gewählt aus der Menge an verfügbaren Lösungen [Wikipedia (2011)]. Die ausschlaggebenden Argumente für diese Wahl sind, dass es mit einem geringen Speicherverbrauch ein professionelles und zugleich einfach verwendbares Echtzeitbetriebssystem darstellt. Zusätzlich bietet FreeRTOS eine große Entwicklergemeinde, wodurch viele unterschiedliche Zielplattformen[38] unterstützt werden und ein guter Support geboten wird.

6.1.4 Der USB-Stack

Bei dem verwendeten Development-Board wird auch ein USB-Stack mitgeliefert mit dem Namen „Stellaris® USB Library". Dieser komplett in C geschriebene Stack bietet unter anderem die benötigte Host-Funktionalität. Zusätzlich bietet dieser eine Treiber-Architektur, was das einfache Einbinden unterschiedlicher USB-Geräte gestattet. Zusätzlich wird Interrupt-gestütztes Empfangen von Daten ermöglicht, wodurch Polling auf der Anwendungsebene verhindert wird[39]. Des Weiteren kann dieser Stack mit unterschiedlichen Toolchains übersetzt werden, wie beispielsweise mit der GCC-Toolchain. [Texas Instruments (2011)]

Weitere wünschenswerte Eigenschaften wären eine Hardwareabstraktionsschicht zum einfachen Portieren auf unterschiedliche Zielplattformen und die Unterstützung von USB-Hubs. Dieses wird leider nicht von der „Stellaris® USB Library" unterstützt. Anderseits ist die Auswahl an frei verwendbaren USB-Stacks minimal und diese bieten normalerweise eine geringere verfügbare Funktionalität. Zudem müsste ein solcher USB-Stack noch auf die hier verwendete Hardware portiert werden.

Aus diesen Gründen wird für einen ersten Prototyp der Sensorstation die „Stellaris® USB Library" zum Einsatz kommen, insbesondere da die genannten Nachteile hier nicht problematisch sind.

[37] http://www.freertos.org/
[38] Momentan sind es offiziell 27 und zusätzlich kommen viele weitere durch die Community hinzu.
[39] Diese Aussage stellt den Sachverhalt vereinfacht dar und wird später genauer erläutert.

6.1.5 Der TCP/IP Stack

Im Bereich der eingebetteten Systeme gelten uIP[40] und lwIP[41] (lightweight IP) als die populärsten freien die TCP/IP Stacks[42]. Beide Projekte wurden ursprünglich von dem Schweden Adam Dunkels entwickelt und haben jeweils eine große Entwicklergemeinde gefunden. Der Stack uIP wird heutzutage nur noch als Teil des Echtzeitbetriebssystem Contiki[43] weiterentwickelt, wobei beide Stacks schon von diversen Unternehmen eingesetzt wurden, wie zum Beispiel von Xilinx[44].

Beide Stacks bieten unter anderem die Protokolle IP, UDP und TCP. Dennoch lassen diese sich unterscheiden, denn uIP ist für 8- und 16-Bit-Mikrocontroller ausgelegt. Diese Art Mikrocontroller besitzen meist nur wenige Kilobyte RAM- als auch Flashspeicher, zudem sind 32-Bit-Rechenoperationen teilweise sehr aufwendig. Aus diesen Gründen wurde bei uIP das TCP-Protokoll nicht vollständig implementiert. Das in Kapitel 3.3.2 vorgestellte "Sliding-Window" Verfahren wurde nicht umgesetzt. Dies führt dazu, dass man mit dem uIP-Stack nur sehr geringe TCP-Übertragungsgeschwindigkeiten erreichen kann, welche im Bereich von wenigen Kilobyte liegen. Mit diesen Einsparungen benötigt der Stack bei der Nutzung der Protokolle IP, TCP und ICMP nur 5 kB für den Programmcode. [Dunkels (2003)]

Der TCP/IP Stack lwIP hingegen ist für Mikrocontroller mit deutlich mehr Flash- und RAM-Speicher ausgelegt. Zudem implementiert der Stack die meisten Internetprotokolle[45] und so auch das Point-to-Point Protocol. Die Protokolle IP, UDP und TCP wurden standardkonform implementiert was bedeutet, dass TCP hier auch das "Sliding-Window" Verfahren verwendet. Dadurch lassen sich hohe Übertragungsgeschwindigkeiten erreichen. Zusätzlich bietet lwIP eine Hardwareabstraktionsschicht und die Möglichkeit zusammen mit einem Echtzeitbetriebssystem Multithreading mit Berkeley-ähnlichen Sockets zu betreiben. Falls kein Multithreading benötigt wird, steht eine sehr performante „RAW-API” zur Verfügung, welche mit Callback-Funktionen arbeitet. Die Größe des Programmcodes im Flash des Mikrocontrollers lässt sich durch das Aktivieren der unterschiedlichen Protokolle sehr gut skalieren, dennoch muss man durchschnittlich mit einer Größe zwischen 20 und 50 Kilobyte rechnen.

Bei dieser Arbeit ist die Entscheidung zugunsten von lwIP gefallen, hauptsächlich aufgrund der deutlich höheren TCP-Übertragungsgeschwindigkeit. Ein weiterer Vorteil ist, dass bereits das PPP-Protokoll implementiert ist. Für die Realisierung wird die zuvor genannte „RAW-API”

[40] http://www.sics.se/~adam/old-uip/index.html

[41] http://www.sics.se/~adam/lwip/

[42] Es gibt noch eine Vielzahl an weiteren TCP/IP Stacks, die in dieser Arbeit nicht näher betrachtet werden, welche oft proprietär oder gar auf spezielle Anwendungsfälle zugeschnitten sind.

[43] http://www.sics.se/contiki/

[44] http://www.xilinx.com/support/documentation/application_notes/xapp1026.pdf
http://www.xilinx.com/support/documentation/application_notes/xapp807.pdf
beide abgerufen am 19.07.2011

[45] IP, ICMP, IGMP, UDP, TCP, DNS, SNMP, DHCP, AUTOIP, PPP und ARP

verwendet, da eine Sensorstation nicht die Fähigkeit benötigt mehrere TCP-Verbindungen gleichzeitig herzustellen.

Anmerkung: Die nachfolgenden zwei Abschnitte definieren die zu übertragenden Daten zwischen den Sensorstationen und dem Server. Aus diesem Grund haben diese Abschnitte auch Auswirkungen auf die Server-Implementierung.

6.1.6 Das Datenübertragungsformat

Das in Kapitel 5.1.3 aufgezeigte Problem die einzelnen Messwerte der Sensoren zu identifizieren, soll mit folgendem allgemeinem XML-Format gelöst werden. Dieses muss von den Sensorstationen erstellt und von dem Server geparst werden können.

```
1  <DataEndpoint>
2          <EndpointName>Sensorstation 2</EndpointName>
3          <EndpointDescription>Standortbeschreibung</EndpointDescription>
4          <SensorData type="Temperature">
5                  <SensorName>ID1</SensorName>
6                  <SensorValue>0x800FF000</SensorValue>
7          </SensorData>
8          <SensorData type="Temperature">
9                  <SensorName>ID2</SensorName>
10                 <SensorValue>0x000FF000</SensorValue>
11         </SensorData>
12 </DataEndpoint>
```

Alle Informationen die von einer Sensorstation übertragen werden sollen, werden in den *DataEndpoint*-Block gefasst. Dieser besitzt immer ein Namensattribut, welches die Bezeichnung der Sensorstation enthält. Diese Bezeichnung ist in der Anwendung eindeutig. Zusätzlich wird ein optionales Attribut aufgeführt in Zeile 3, dass beispielsweise eine Standortbeschreibung enthalten kann.

Danach kommen beliebige viele Sensorblöcke, wobei jeder Sensorblock exakt einen Sensor der Sensorstation beschreibt. Diese Sensorblöcke müssen neben einem dem Server bekannten Typ auch einen für die Sensorstation eindeutigen Namen besitzen. Für die Übertragung der Messwerte wurde ein Fixpunkt-Format gewählt. Dies aus dem Grund, dass das Floating-Point Operationen auf Mikrocontrollern nur sehr umständlich durch Software umgesetzt werden können. Das Format ist mit 32-Bit wie folgt definiert:

Signed Bit	$0...2^{15}$	$0...2^{-16}$

Mit dieser Auflösung lassen sich Messwerte fast jedes Sensortyps ausreichend genau darstellen in einer Auflösung von 2^{-16} großen Schritten. Auf der Seite des Servers können diese Werte mit einer einfachen Multiplikation mit dem Wert 2^{-16} in das Floating-Point Format *double* umgewandelt werden bei vorheriger Betrachtung des Vorzeichen-Bits[46].

Zusätzlich kann durch eine einfache Erweiterung dieses XML-Formats die Übertragung von Binärdaten, wie beispielsweise Kamerabilder, oder Statusmeldungen ermöglicht werden.

6.1.7 Die Übertragungssicherheit

Die Zielsetzung in Kapitel 2.3.1 fordert eine verschlüsselte Datenübertragung zwischen den Sensorstationen und dem Server, um Manipulation vorzubeugen. Um die Übertragung der Sensorinformationen zu verschlüsseln, gibt es verschiedene Möglichkeiten. Da die verwendete Mikrocontroller-Plattform LM3S9B96 im ROM standardmäßig Lookup-Tabellen für die Advanced Encryption Standard (AES) Verschlüsselung hat und zugleich auch noch die auf PolarSSL-basierte AES-Bibliothek mitgeliefert, ist es empfehlenswert diese auch zu verwenden.

Für den Prototyp wird zunächst nur der sogenannte „Electronic Code Book"-Modus verwendet mit einem konstantem 128 Bit Schlüssel, welcher für alle Sensorstationen gelten wird. Dieser Modus ist bietet zwar Angriffsstellen ist dafür jedoch vergleichsweise einfach zu implementieren[47]. Mit 83 $\frac{Zyklen}{Byte}$ ist die Verschlüsselung noch performant genug, sodass diese hier kaum die Energieeffizienz beeinflusst wird. Sollten jedoch je größere Datenmenge übertragen werden, muss geprüft werden, ob man die Verschlüsselung nicht gegen ein Authentisierungsverfahren austauschen kann, da sich dann die Rechendauer zum Verschlüsseln der Daten in Sekundenbereich verschiebt (vgl. [Texas Instruments (2010b)]).

6.2 Entwurf des Servers

Nach dem Entschluss ein Java-basiertes Rest-Framework zu verwenden steht nun die Entscheidung bevor ein konkretes zu wählen, um mit der Realisierung beginnen zu können. Im Optimalfall sollte dieses neben einer REST-basierten Architektur von Haus eine Vielzahl von Funktionen, wie beispielsweise einen objektrelationalen Mapper mitbringen, um so die Entwicklungszeit zu minimieren.

[46] Arithmetische Operationen sind mit diesem Zahlenformat nicht beabsichtigt, weswegen keine Guard-Bits zum Abfangen eines Überlaufs benötigt werden.

[47] Bei der Weiterentwicklung des fertigen Prototyps sollte eine stärkere Verschlüsselung verwendet werden, wie beispielsweise das CBC-Verfahren welches ebenfalls von der Bibliothek angeboten wird (dieses erfordert den Austausch von Initialisierungsvektoren zwischen beiden Seiten). Zusätzlich muss über ein ausgefeilteres Schlüsselmanagement nachgedacht werden.

Nach einer kurzen Recherche trifft man unter anderem[48] auf das Open-Source Web-Framework mit dem Namen „Play Framework" (kurz „Play!")[49]. Dieses implementiert das Model-View-Controller Pattern und verwendet eine Vielzahl von bekannten Java-Libraries, wobei es zugleich einem die üblichen Konfigurationsaufgaben abnimmt und deren Verwendung vereinfacht. Folgendes bietet unter anderem Play!:

- Einfache Konfiguration der Anwendung und der HTTP-Routen (REST)
- Ein HTTP-Server und eine SQL-Datenbank, welche beide leicht austauschbar sind
- Ein Groovy-basierte Template Engine
- Module für das einfache Generieren von Login- und administrativen Webseiten-Bereichen
- Eine Menge an weiteren Funktionen, wie beispielsweise ein Job-Scheduler

Zusätzlich bietet Play! einen eingebauten Eclipse-Compiler, welcher im Entwicklungsmodus Java-Dateien zur Laufzeit übersetzt. Dadurch kann man die Webanwendung entsprechend des Play!-Slogans „Fix the bug and hit reload!" entwickeln.

Durch die Möglichkeit Java-basierte Webanwendungen sehr einfach und komfortabel mithilfe des „Play Framework" zu entwickeln, welche zudem sehr performant[50] sind, ist es bestens geeignet, die Grundlage für den Server zu bilden.

Als Hardware Plattform wird für den Server ein typisches X86-basiertes System verwendet, auf welchem die Linux-Distribution Ubuntu läuft. Der Server ist zudem mit einer statischen IP-Adresse aus dem Internet erreichbar, als auch über einen DynDNS-Eintrag.

6.2.1 Die Funktionalität und das Datenmodell des Servers

Für eine erste prototypische Umsetzung werden die in Abbildung 6.4 dargestellten Anwendungsfälle implementiert. Die Webanwendung soll eine Log-in-Funktion bieten, um Zugang zu den Sensorinformationen nur berechtigten Nutzern zu Verfügung stellen. Um die Sensorinformationen der unterschiedlichen Stationen einfach und schnell analysieren zu können, sollen diese als über die Zeit aufgetragene Diagramme dargestellt werden. Zusätzlich soll die Möglichkeit vorhanden sein, die Informationen zu filtern.

[48] Ein umfassender Vergleich zwischen den unterschiedlichen Java-basierten REST Frameworks würde den Rahmen dieser Arbeit sprengen. Weitere bekannte Frameworks sind unter anderem Restlet und Jersey.
[49] http://www.playframework.org/
[50] Laut Entwicklerteam sind im Idealfall mehrere Tausend Requests pro Sekunde beherschbar.

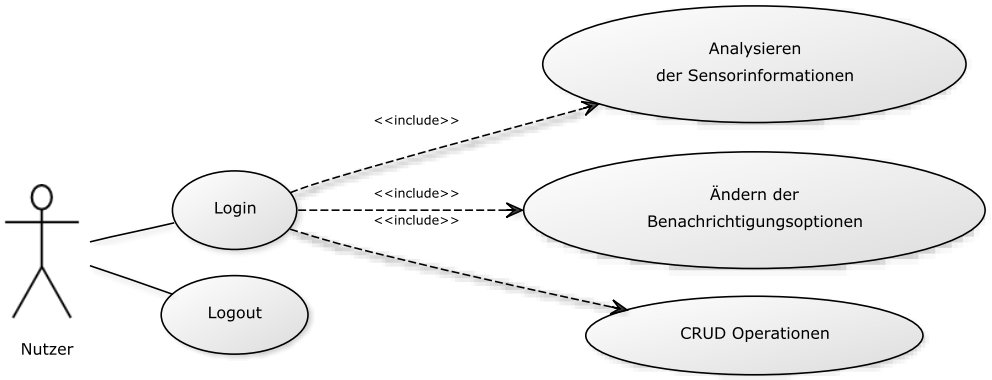

Abbildung 6.4: Beabsichtigte Nutzeroperationen des Prototyps

Ein einfaches Benachrichtigungskonzept soll ebenfalls realisiert werden. Sendet eine Sensorstation über längere Zeit keine neuen Daten oder ein Sensor nähert sich seinem kritischen Bereich[51] dann soll der Nutzer per E-Mail informiert werden. Obendrein sollen die typischen administrativen CRUD-Funktionen[52] für die gespeicherten Informationen zu Verfügung stehen. Die Webanwendung soll zunächst nur eine rudimentäre Benutzerverwaltung enthalten, was bedeutet, dass vorerst auf unterschiedliche Benutzergruppen verzichtet wird.

Aus der Zielsetzung und den gerade genannten Funktionalitäten lässt sich folgendes Datenmodell aufstellen, welches zudem mit dem Datenübertragungsformat harmoniert:

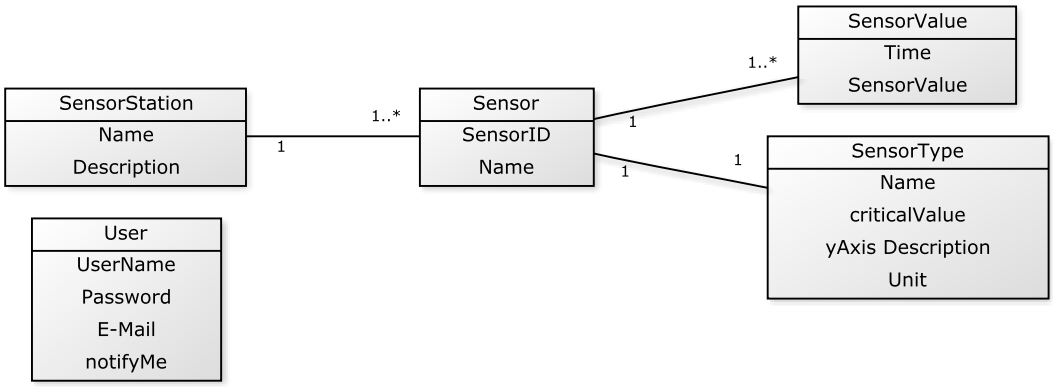

Abbildung 6.5: Fachliches Datenmodell des Servers

Jede Sensorstation besitzt neben einem Namen und einer (Standort-)Beschreibung typischerweise Sensoren. Jeder dieser Sensoren besitzt einen generierten Bezeichner. Da die-

[51] Bei einer Anwendung zum Frostschutz wäre der kritische Bereich knapp über dem Nullpunkt.
[52] Create, Read, Update and Delete (CRUD)

ser Bezeichner meist nicht sehr aussagekräftig ist, soll der Server die Möglichkeit offen lassen, dass die Sensoren lokal neu benannt werden können. Jeder dieser Sensoren besitzt zudem eine Liste über „*Sensorvalues*", welche alle bisherigen Messwerte des Sensors gepaart mit der jeweiligen Empfangszeit beinhaltet. Der Datentyp „*SensorType*" hat die Aufgabe den Sensoren eine physikalische Größe zuzuweisen und den kritischen Bereich anzugeben, damit die Benutzer, welche das Wünschen (*notifyMe*), benachrichtigt werden können.

Für den Empfang der Sensordaten wird eine HTTP-Route erstellt, über welche die Messwerte der Sensorstationen empfangen werden können. Bei Empfang einer neuer Nachricht mit Sensorinformationen muss diese zuerst entschlüsselt werden. Danach kann die Nachricht, welche im Klartext aus dem in Abschnitt 6.1.6 definierten XML-Format besteht, geparst werden. Anschließend muss geprüft werden, ob die Sensortypen bekannt sind und ob die empfangenen Messwerte im kritischen Bereich liegen. Zu guter Letzt werden die Sensorinformationen in der Datenbank abgespeichert und der HTTP-Statuscode 200, welcher OK bedeutet, zurückgesendet. Sollte einer der vorhergehenden Schritte fehlschlagen, antwortet der Server mit 400, was als „Bad Request" definiert ist.

Zusätzlich muss auf dem Server mindestens ein Job implementiert werden, welcher in festen Zeitabständen überprüft, ob jeder Sensor und damit auch jede Sensorstation sich in der letzten Zeit mit neuen Messwerten gemeldet hat. Ist dies nicht der Fall, ist davon auszugehen, dass je nach Fall entweder eine Sensorstation komplett oder ein Sensor einer Station ausgefallen ist. Tritt diese Situation auf, muss der Benutzer informiert werden, umso eine schnelle Reparatur der beeinträchtigten Sensorstation zu ermöglichen.

Kapitel 7

Realisierung des Konzeptes

Nach dem Entwurf der Anwendung im vorherigen Kapitel zeigt dieses hier nun die Implementierung der beabsichtigen M2M-Anwendung auf. Die Abbildung 7.1 zeigt die zu realisierenden Komponenten auf.

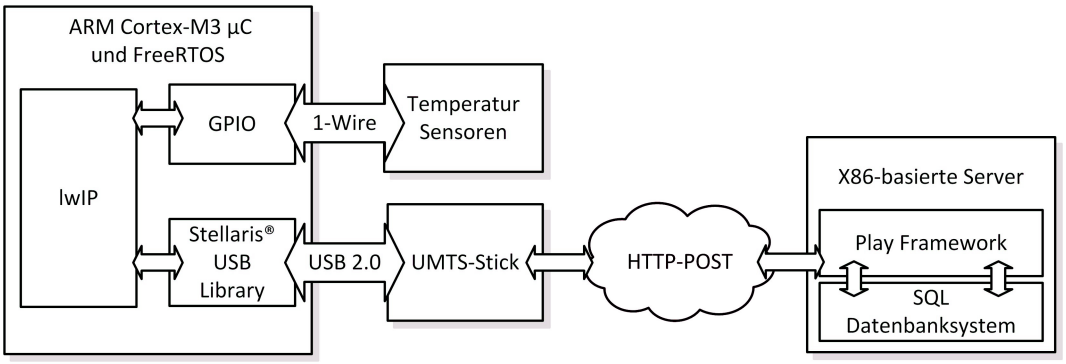

Abbildung 7.1: Struktur des zu realisierenden Prototyps

Für einen ersten Prototyp der Sensorstation kommen beispielhaft Temperatursensoren zum Einsatz, welche das 1-Wire Bussystem verwenden. Zusätzlich muss für die Realisierung der Sensorstation das Zusammenspiel zwischen FreeRTOS, lwIP und dem UMTS Stick implementiert werden, um die Sensordatenübertragung zu ermöglichen. Auf der Serverseite ist die beabsichtigte Webanwendung zu entwickeln, welche das Play-Framework als Grundlage verwendet.

7.1 Entwicklung der Sensorstation

Zusammen mit dem in Kapitel 6.1.2 vorgestellten Development-Board stellt Texas Instruments eine Reihe an Bibliotheken zur Verfügung. Darunter zu finden ist beispielsweise eine Treiberbibliothek mit welcher alle Komponenten des Development-Boards schnell und einfach angesprochen werden können. Von diesen Bibliotheken wurde Gebrauch gemacht bei dieser Umsetzung, falls sich deren Nutzung an den entsprechenden Stellen empfohlen hat, umso Entwicklungszeit einzusparen. Dazu wurden alle benötigten Bibliotheken in einer sogenannten „Static Library" zusammengefasst, welche dann von dem Sensorstation-Entwicklungsprojekt referenziert wird. Mit dieser Vorgehensweise ergibt sich bei dem Kompilieren der Sensorstation der Vorteil, dass nur noch die wirklich verwendeten Teile der „Static Library" mit übersetzt werden. Dadurch folgt eine kleinere Ausgangsgröße des resultierenden binären Programms (ELF), mit welchem weniger Platz im Flash-Speicher verbraucht wird.

7.1.1 Benötigte Debugging-Werkzeuge für die Realisierung

> Most debugging problems are fixed easily; identifying the location of the problem is hard. – unknown

Neben dem typischen Debuggen des Programmcodes mithilfe des GNU-Debuggers GDB muss auch der Ablauf diverser Protokolle auf ihre Korrektheit analysiert werden. Um die Kommunikation zwischen dem Mikrocontroller und dem UMTS Stick analysieren zu können, wurde der USB Explorer 200[53] von der Firma Ellisys eingesetzt. Hierbei handelt es sich um einen USB 2.0 Protokollanalysator. Dieser wird zwischen Mikrocontroller und UMTS Stick geschaltet. Dies bedeutet ein USB Kabel geht vom Mikrocontroller zum USB Explorer und ein weiteres vom USB Explorer zu dem UMTS Stick. Obwohl der USB Explorer dazwischen geschaltet ist, bemerken die beiden Komponenten dies nicht. Zusätzlich bietet der USB Explorer einen dritten Anschluss, um seiner Aufgabe gerecht zu werden. Über diesen kann der USB-Verkehr zwischen Mikrocontroller und UMTS Stick mit einem Computer und der Ellisys Visual USB Software aufgezeichnet und analysiert werden. Für die Analyse besteht neben diversen Filter- und Gruppierungsfunktionen, die Möglichkeit die empfangenen Informationen aufgeschlüsselt darzustellen.

Die Ellisys Visual USB Software ist am Anfang des Datenübertragungsablaufes ausreichend für das Debuggen. Sobald die Sensorstation aber Netzwerkpakete versendet, ist das Debuggen nicht mehr ohne weiteres möglich, da diese Pakete nicht mehr aus reinem ASCII-Text bestehen. Um die Netzwerk-Kommunikation der Sensorstationen dennoch analysieren zu

[53] http://www.ellisys.com/products/usbex200/index.php

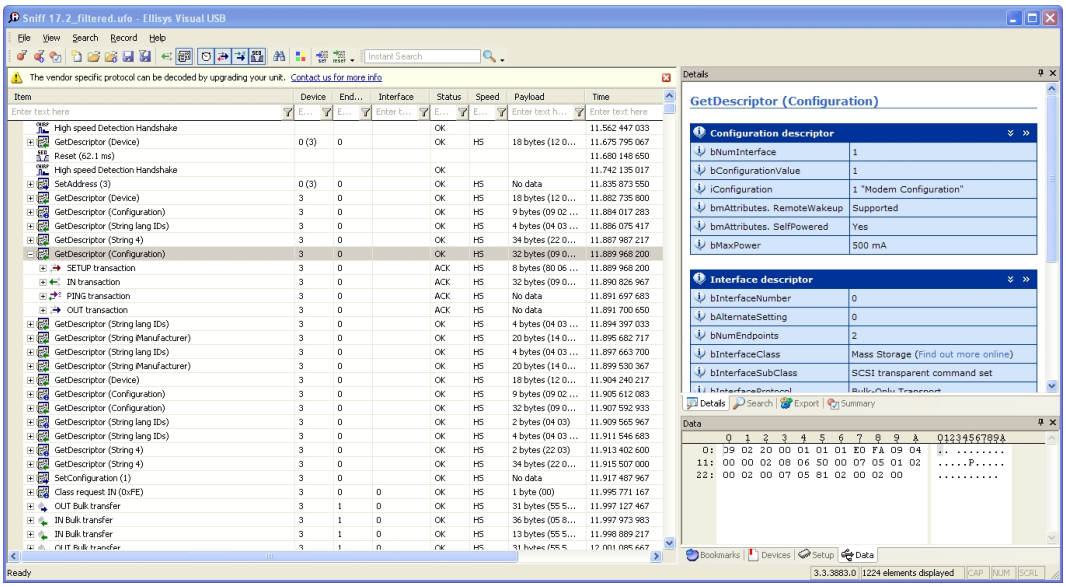

Abbildung 7.2: Oberfläche der Ellisys Visual USB Software

können, wurde ein Python-Skript entwickelt und das Programm Wireshark[54] verwendet. Die USB-Pakete können mithilfe der Visual USB Software als CVS-Datei exportiert werden. Das genannte Python-Skript hat dann die Aufgabe diese Datei zu parsen und in das für Wireshark lesbare pppdump-Format[55] umzuwandeln. Danach kann man mit Wireshark die Netzwerk-Kommunikation und die dabei versendeten Pakete genau analysieren.

7.1.2 FreeRTOS

Das Echtzeitbetriebssystem stellt die Grundlage der Sensorstation dar. Dessen Scheduler bietet präemptives als auch kooperatives Multitasking zusammen mit Priorität gesteuerten Scheduling. In dieser Arbeit wird präemptives Multitasking verwendet, wobei die unterschiedlichen Tasks gleich priorisiert sind.

FreeRTOS bietet in seiner aktuellen Version[56] neben dem eigentlichen Quellcode des Echtzeitbetriebssystems zugleich auch eine Portierung für ARM Cortex-M3 Mikrocontroller an. Mit diesen Dateien lässt sich FreeRTOS schnell implementieren. Hierfür muss zunächst ein periodischer Timer definiert werden, welcher dem Echtzeitbetriebssystem als Clock dient. Die ARM Cortex-M3 Architektur bietet für diese Problemstellung den sogenannten SYSTICK-Timer, welcher im Nested Vectored Interrupt Controller (NVIC) integriert ist. Zu-

[54] http://www.wireshark.org/
[55] Entspricht dem Ausgabeformat der pppd record Option (Linux).
[56] Version 7.0.1 wird in Rahmen dieser Arbeit verwendet.

sätzlich muss eine Header-Datei namens *FreeRTOSConfig.h* erstellt werden, welche das Echtzeitbetriebssystem konfiguriert. Diese enthält unter anderem die Einstellung nach wie vielen Ticks der Clock der Scheduler periodisch aufgerufen wird.

Die spätere Verwendung von FreeRTOS ist sehr einfach:

```
1  xQueueHandle sensorQueue = 0;
2  sensorQueue = xQueueCreate( SENSORQUEUE_SIZE , sizeof( sensorData * ) );
3
4  xTaskCreate(  mainTask, //function pointer
5                ( signed portCHAR * ) "mainTask", //debug name
6                1000, //stack
7                NULL, //task parameters
8                1, //task priority
9                NULL ); //handle variable
10
11 xTaskCreate( oneWireTask, ( signed portCHAR * ) "oneWireTask",  1000, NULL, 1, NULL );
12
13 // Starting the scheduler
14 vTaskStartScheduler();
15
16 // Should never be reached
17 while(1){}
```

Dieser Programmcode entstammt vereinfacht[57] aus der Mainfunktion der Sensorstation und ist für das Erstellen von zwei Tasks und der „Message Queue" zuständig entsprechend der im vorhergegangenen Kapitel definierten Architektur[58]. Die Warteschlange, welche für die Kommunikation zwischen den Sensor-Tasks und dem Datenübertragungstask Verwendung findet, wird so initialisiert, dass sie eine vorgegebene Menge folgender Structs aufnehmen kann:

```
1  typedef struct {
2      char * SensorName;  // sensor ID or something similar
3      char * SensorType;  // e.g. Temperature
4      unsigned int SensorValue; // S 15 . 16
5  } sensorData;
```

Dieses Struct enthält alle notwendigen Informationen um einen Messwert auf der Serverseite darzustellen. Im Anschluss wird der Scheduler gestartet, wodurch nur noch die hier zuvor definierten Tasks, die wiederum einfache Funktionen sind, nebenläufig ausgeführt werden. Hier ist dies der Task zum Auslesen der 1-Wire Temperatursensoren und der Datenübertragungstask.

[57] Die Fehlerbehandlung der einzelnen Aufrufe wurde hier ausgelassen.
[58] siehe Abbildung 6.1

7.1.3 Der Datenübertragungstask

Dieser Task, welcher auch „Main-Funktion” in dieser Arbeit genannt wird, behandelt alle Aufgaben der Sensorstation abgesehen von dem Auslesen der Sensorwerte. Dessen Ablauf wurde im vorherigen Kapitel ausführlich beschrieben und in Abbildung 6.2 dargestellt. Dieser Ablauf wurde im Datenübertragungstask als endlicher Automat implementiert. Die nachfolgenden Abschnitte zeigen kompakt die notwendigen Schritte, um den Ablauf des Automaten zu ermöglichen.

7.1.3.1 Die USB-Kommunikation mit dem UMTS Stick

Für die Kommunikation zwischen Mikrocontroller und UMTS Stick kommt die „Stellaris® USB Library” zum Einsatz mit dessen USB-Host Funktionalität. Diese Bibliothek bietet einen Treiber-basierten Ansatz. Dies bedeutet für die unterschiedlichen USB-Geräteklassen kann ein Treiber mit jeweils einer Open- und Close-Funktion definiert werden. Zusätzlich kann ein Default-Treiber definiert werden, welcher aktiv wird, wenn kein anderer Treiber passt.

```
1  // Drivers:
2  static const tUSBHostClassDriver UMTSStick_MS =
3   {USB_CLASS_MASS_STORAGE, UMTSStick_MS_Open, UMTSStick_MS_Close, 0};
4  static const tUSBHostClassDriver UMTSStick_Modem =
5  { USB_CLASS_VEND_SPECIFIC, UMTSStick_Modem_Open, UMTSStick_Modem_Close, 0};
6  DECLARE_EVENT_DRIVER(EventDriver, 0, 0, USBHCDEvents); // Default driver
7
8  static tUSBHostClassDriver const * const AllHostDriver[] =
9  { &UMTSStick_MS, &UMTSStick_Modem, &EventDriver };
```

Hier wurden ein Massenspeicher-Treiber, ein herstellerspezifischer Treiber und der Default-Treiber implementiert. Die Open-Funktion des jeweiligen Treibers wird durch die USB-Bibliothek aufgerufen nach einem USB-Reset je nachdem welche Interface Klasse momentan beim UMTS Stick aktiv ist. Der Massenspeicher-Treiber ist für das USB Modeswitch Verfahren zuständig, welches in Kapitel 3.2.2 vorgestellt wurde. Für den Modembetrieb kommt der herstellerspezifische Treiber zum Einsatz.

Eine Open-Funktion enthält das Auslesen und Überprüfen der unterschiedlichen Deskriptoren. Unter anderem wird hier überprüft ob die Vendor-ID und Product-ID des USB-Geräts bekannt ist. Dies ist notwendig, da UMTS Sticks sich unterschiedlich Verhalten und die Initialisierung von Stick zu Stick leider variiert.

```
1  typedef struct{
2      unsigned int vendor; //Vendor ID
3      unsigned int product; //Product ID
4      int (*init)(void); //Function to initialize the device
5  } USB_DEVICE;
```

```
6
7  static const USB_DEVICE modems[] =
8  {
9      { 0x1c9e, 0x9603, klarMobilModemInit },
10     { 0x19D2, 0x117, zteModemInit }
11 };
```

Aus diesem Grund wurde hier ein Array von Strukturen definiert, welches von dem jeweiligen Treiber durchlaufen wird, um die passende Initialisierungsmethode aufzurufen. Die Initialisierung ist notwendig um den UMTS Stick in einen betriebsbereiten Zustand zu versetzen, umso mit dem Modeswitch Vorgang oder dem Modembetrieb beginnen zu können. Initialisiert werden die Sticks mittels USB Control-Transfers. Das Wissen, welche Control-Transfers notwendig sind um den Stick zu initialisieren wurde erlangt in dem der USB-Verkehr gesnifft wurde, während der jeweilige Stick mit einem Windows Computer betrieben wurde. Dennoch war hier teilweise ein „Trial and error" Ansatz notwendig, da die Control-Transfers zum Teil herstellerspezifisch sind und mit dem in der Hochschule verfügbaren USB-Protokollanalysator nicht entschlüsselt werden konnten. Nach dem erfolgreichen Durchlaufen der Open-Funktion kann das Gerät verwendet werden. Dies bedeutet hier, dass im Massenspeicher Modus der Modeswitch-Befehl gesendet wird, was direkt zu einem USB-Reset und Aktiv werden des herstellerspezifischen Interfaces führt. Wenn der UMTS Stick mit einem aktiven herstellerspezifischen Interface betriebsbereit ist, kann mit dem Senden der AT-Befehle begonnen werden, um das Modem zu konfigurieren und in das Mobilfunknetz einzuwählen.

Die Kommunikation zwischen Host und UMTS Stick basiert auf Bulk-Transfers. Mit diesen werden Daten blockorientiert über eine Pipe versendet und auch empfangen. Das Schreiben von Daten, wie beispielsweise AT-Befehle, ist aufgrund des Master-Slave-Protokolls unproblematisch. Das Lesen von Daten ist sowohl blockierend als auch nicht-blockierend möglich. Insbesondere das zweitgenannte Verfahren wird hier verwendet. Dazu wird bei der Allokierung der entsprechenden IN-Pipe eine Callback-Funktion mit angegeben. Zusätzlich wird regelmäßig durch den Datenübertragungstask eine IN-Transaktion gestartet. In den meisten Fällen wird diese von dem UMTS Stick mit NAK beantwortet, was bedeutet das dieser momentan keine Daten senden kann. Sobald jedoch dieser Daten senden kann, wird durch die Interrupt Service Routine der USB-Bibliothek die zuvor genannte Callback-Funktion aufgerufen, welche die Aufgabe hat die Daten auszulesen und in einen synchronisierten Ringpuffer abzuspeichern. Die empfangenen Daten im Ringpuffer werden später durch den lwIP-Stack genutzt.

7.1.3.2 Die Architektur und der Verbindungsaufbau in das Internet

Nach dem erfolgreichen Senden des AT-Befehls *ATDT*99#*, mit welchem der UMTS Stick sich in das Mobilfunknetz einwählt, kann mit dem PPP-Verbindungsaufbau begonnen wer-

den. Diese Aufgabe übernimmt der TCP/IP Stack lwIP, welcher auch das PPP-Protokoll implementiert hat. Wie schon im vorherigen Kapitel erwähnt wurde, soll die RAW-API zum Einsatz kommen, da eine Sensorstation nicht die Fähigkeit benötigt mehrere Verbindungen gleichzeitig herzustellen und zu verwalten. Dadurch ist lwIP sehr effizient unter Verwendung von Callback-Funktionen und lässt sich schnell auf diese Plattform zu portieren. Skaliert und eingestellt wird lwIP über die Header-Datei *lwipopts.h*, welche man erstellen muss. In dieser werden sowohl die einzelnen Netzwerkprotokolle aktiviert oder deaktiviert, als auch Feineinstellungen vorgenommen, wie zum Beispiel die gewünschte TCP-Fenstergröße.

Zusätzlich muss für die Verwendung des Point-to-Point Protokolls zusammen mit einer seriellen Verbindung, wie es ein UMTS Stick ist, noch eine *sio.c* Quelldatei erstellt werden, welche es lwIP ermöglicht Daten zu versenden. Implementiert wurde die Datei so, dass diese die Daten an die USB-Bibliothek weitergereicht wo wo diese dann mithilfe des UMTS Sticks versendet werden. Die andere Richtung der Verbindung funktioniert bei lwIP im RAW-API Modus folgendermaßen: Die Funktion *pppos_input* muss regelmäßig mit den empfangenen Daten aufgerufen werden. Diese Aufgabe übernimmt der Datenübertragungstask ebenfalls, indem dieser die per USB empfangenen Daten aus dem im Abschnitt zuvor beschriebenen Ringpuffer entnimmt und diese mit dem *pppos_input*-Funktionsaufruf an lwIP weiterleitet.

Mit diesen Grundlagen kann mit dem PPP-basierten Verbindungsaufbau in das mobile Internet begonnen werden. Der folgende Codeausschnitt beinhaltet den Zustand des endlichen Automaten, welcher für den Start der Point-to-Point Verbindung mit dem ISP zuständig ist.

```
1  void startPPP(State* s){
2
3      // Starting PPP with CHAP authentication
4      pppSetAuth(PPPAUTHTYPE_CHAP, "o2", "o2");
5      pd = pppOverSerialOpen(0, pppCallback, 0);
6      if(pd < 0){
7          s->state = STATE_CRIT_ERROR;
8          s->arg = (void*)ppp_strerr(pd);
9          return;
10     }
11     s->state = STATE_WAIT4CALLBACK;
12     s->arg = "Waiting for PPP Callback";
13 }
```

Die eigentlichen Befehle zum Konfigurieren und Starten der Verbindung sind in Zeile 3 und 4 zu finden. Beim Aufruf der Funktion *pppOverSerialOpen* wird nicht blockierend eine Verbindung initiiert und zugleich eine eigens definierte Callback-Funktion übergeben. Wenn die Internetverbindung hergestellt ist, ruft lwIP diese Callback-Funktion auf. Dieses Verhalten ist typisch für lwIP im RAW-API Modus und alle anderen Protokolle des Stacks arbeiten ähnlich. Die Aufrufe der Callback-Funktionen entstehen aus der zuvor genannten *pppos_input* Funk-

tion. Diese wiederum ruft die Callback-Funktionen entsprechend der empfangenen Daten und des jeweiligen Zustandes der lwIP FSMs auf.

Die restlichen Code-Zeilen dienen der Fehlerbehandlung und dem Auslösen von Transitionen in der State-Machine des Datenübertragungstasks. Für die FSM wurde ein Funktionspointer Array implementiert, wobei jedes Element gepaart mit einem Enum-Wert einem Zustand entspricht. Zusätzlich wird eine Instanz der Struktur *State* genutzt, welche das Gedächtnis des Automaten darstellt und die Möglichkeit bietet Argumente an die unterschiedlichen Zustände zu übergeben. Die Hauptschleife des Datenübertragungstasks ruft jeweils den momentan in der Instanz aktiven Zustand auf und arbeitet zusätzlich die anderen Aufgaben ab, die regelmäßig durchgeführt werden müssen, wie zum Beispiel die Initiierung der USB-IN-Transaktionen.

Im Quellcode wird hier nun, falls der *pppOverSerialOpen* Aufruf erfolgreich war, in den „Wait for Callback" Zustand gewechselt. Dieser Zustand ist eine leere Funktion, welche aber dennoch eine wichtige Aufgabe erfüllt: Sie blockiert das System nicht. Es ist sehr wichtig, dass keiner der Zustände des Automaten blockiert, denn ansonsten kann die Hauptschleife des Datenübertragungstasks nicht mehr die angekommenen USB-Daten verarbeiten, was zu dem Einfrieren des Systems führen würde. Die Callback-Funktionen enthalten dann jeweils Anweisungen je nach Erfolg oder Misserfolg um den „Wait for Callback" Zustand wieder zu verlassen.

Hier wird nachdem erfolgreichem Verbindungsaufbau eine DNS-Anfrage gestartet, sofern die IP-Adresse des Servers noch nicht bekannt ist. Dies funktioniert analog zu dem PPP-Verbindungsaufbau nur enthält hier die Callback-Funktion anstatt der der Sensorstation zugewiesenen IP-Adresse nun die des Servers.

7.1.3.3 Die verschlüsselte Datenübertragung

Mit der momentan bestehenden Internetverbindung und der IP-Adresse des Servers kann mit der Datenübertragung begonnen werden. Hierzu wird eine TCP-Verbindung gestartet. Dieser Vorgang verhält sich ebenfalls ähnlich zu den zuvor gegangenen initiierten Verbindungen, nur dass diesmal zwei Callback-Funktionen angeben werden müssen. Einmal eine Callback-Funktion, mit dessen Aufruf signalisiert wird, dass das Herstellen der Verbindung durch Abschluss des Dreiwegehandshakes erfolgreich war und eine weitere Callback-Funktion, welche für alle Fehlerfälle dieser Verbindung zuständig ist.

Wenn nun eine TCP-Verbindung mit dem Server erfolgreich hergestellt wurde, kann mit den Vorbereitungen zur Datenübertragung begonnen werden. Der erste notwendige Schritt ist das Aufstellen des in Kapitel 6.1.6 vorgestellten XML-Datenübertragungsformats. Hierzu werden alle verfügbaren Sensorinformationen aus der synchronisierten Warteschlange entnommen und zusammen mit der Hilfe eines Stringbuilders wird das XML-Format aufgebaut.

Der nächste Schritt ist die Verschlüsselung. Der hier verwendete „Electronic Code Book"-Modus des AES-Verfahrens erfordert, das die zu verschlüsselnden Daten in 128-Bitblöcke aufteilbar sind. Diese Eigenschaft ist bei dem fertigen String, welcher die Sensorinformation gekapselt in dem vereinbarten XML-Datenformat enthält, zu überprüfen. Notfalls ist dieser String zu erweitern. Diese mögliche Erweiterung des Strings ist unkritisch, da auf Serverseite nur die Daten bis zum abschließenden XML-Block betrachtet werden. Danach wird der String blockweise mit dem konstanten 128-Bit Schlüssel verschlüsselt.

Der abschließende Schritt ist das Aufstellen des HTTP-POST-Formates, welches an Server per TCP übertragen werden soll. Dieses ist hier exemplarisch ausgefüllt und hat folgenden Aufbau:

```
1 POST /endpointUpdate HTTP/1.0
2 Host: m2mhaw.dyndns.org
3 Content−type: application
4 Content−length: 3412
5
6 [Nutzdaten]
```

Der Header und der Body werden durch die Leerzeile getrennt. Aus den ersten zwei Zeilen geht hervor, dass eine HTTP-POST Anfrage an die URL *m2mhaw.dyndns.org/endpointUpdate* gesendet wird. Nachfolgend wird die Aussage getroffen das Binärdaten übertragen werden. Im Body stehen dann die Nutzdaten, was hier die verschlüsselten XML-formatierten Sensorinformationen sind.

Versendet wird die HTTP-Anfrage, indem man sie an den lwIP Stack mit der Funktion *tcp_write* übergibt. Ein sofortiges Senden kann man mit der Funktion *tcp_output* erzwingen, welcher ähnlich wie ein Flush-Befehl wirkt. Mit einer zuvor definierten Callback-Funktion kann die Antwort des Servers empfangen werden, welche den HTTP-Statuscode enthält. Dieser Code sagt aus, ob die gesendeten Daten von dem Server akzeptiert wurden. Aufgrund des verwendeten HTTP-Protokolls in Version 1.0 wird automatisch nach der Antwort des Servers auch von diesem ein FIN-Flag gesendet, womit begonnen wird die TCP-Verbindung zu beenden. Zu guter Letzt beendet man auch noch die Internetverbindung mittels PPP, wonach die Sensorstation in den Energiesparmodus versetzt werden kann.

7.1.3.4 Der Energiesparmodus und der Watchdog

Die ARM Cortex-M3 Mikrocontroller Plattform bietet zwei Energiesparmodi und zwei Möglichkeiten aus diesen wieder zu erwachen. Der eine Modus nennt sich Sleep und der andere Deep-Sleep. Diese beiden unterscheiden sich darin, dass im Deep-Sleep-Modus zusätzlich zum Deaktivieren des Prozessor-Takts auch noch die PLL und der Flash-Speicher ausschaltet wird. Aufgewacht werden kann aus diesen Energiesparmodi entweder durch ein Interrupt

oder durch ein externes Signal [ARM (2010)]. Zusätzlich bietet die Plattform noch die Möglichkeit des Clock-Gatings. Dies bedeutet man kann via Software wählen, welche Komponenten während des Schlafens mit einem Takt versorgt werden sollen.

Hier wurde der Deep-Sleep-Modus gewählt aufgrund der besseren Energieeffizienz. Zusätzlich wurden alle Komponenten mithilfe des Clock-Gatings deaktiviert bis auf einen Timer-Baustein, welcher mit Auslösen seines Interrupts als Wecker dient. Zusätzlich wird der Systemtakt für die Zeit des Schlafens auf dessen Minimum gesenkt, was 468 Hertz sind, umso weitere Energie einzusparen.

Die Praxis hat gezeigt, dass die Abarbeitung des Datenübertragungstasks im Allgemeinen länger dauert als die Zeit, die die Tasks der Sensoren benötigen. Pro Datenübertragung ist zudem beabsichtigt nur eine Messung pro Sensor durchzuführen, was über einen Semaphore realisiert ist. Deswegen ist es unkritisch, dass der Datenübertragungstask global den Energiesparmodus auslöst.

Abschließend noch wenige Worte zur Ausfallsicherheit: Die Sensorstation besitzt einen Watchdog, welcher überprüft ob sich der Zustand des endlichen Automaten innerhalb einer gewissen Zeitspanne ändert. Ist dies nicht der Fall, löst dieser einen Reset aus. Während des Energiesparmodus ist auch der Watchdog zwischenzeitlich deaktiviert, sodass dieser während der Schlafphase nicht ablaufen kann. Sollte irgendwo im kompletten Ablauf der Sensorstation ein Fehlerfall auftreten, wird in den Fehlerzustand gewechselt. Dieser löst momentan direkt eine Transition in den Energiesparmodus aus, wobei jedoch hier eine Logging-Funktion, welche eine SD-Karte im UMTS Stick als Speicherort verwendet, denkbar wäre. Alle weiteren Fehlerfälle, wie beispielsweise der Ausfall der Sensorstation durch mangelnde Batteriespannung, sind vom Server zu erkennen.

7.1.4 Der 1-Wire Task

Wie schon mehrfach erwähnt, kommen in dieser Arbeit 1-Wire Temperatursensoren des Typs D18B20 zum Einsatz. 1-Wire beschreibt ein Bussystem basierend auf einer Datenader, welche zugleich auch die Spannungsversorgung darstellt, und einer Masseleitung.

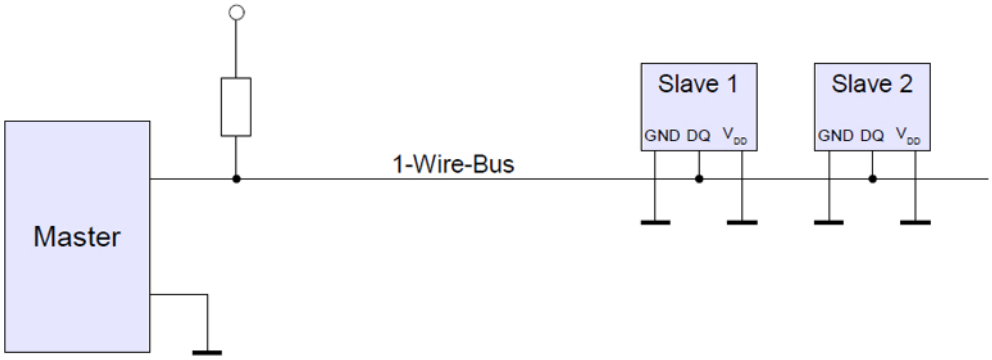

Abbildung 7.3: 1-Wire Temperatursensoren

Die Übertragung basiert auf dem One-Master/Multi-Slaves Prinzip. Übertragen wird seriell und im Halbduplexverfahren. Der 1-Wire-Bus wird über einen Pullup-Wiederstand mit Spannung versorgt und es gilt für den ganzen Bus die Wired-And Funktionsweise. Dies bedeutet, der Master als auch einer der Slaves können den Bus auf Masse schalten. Die Kommunikation zwischen Mikrocontroller und den Temperatursensoren basiert aus zeitlichen Sequenzen, in welchen der 1-Wire-Bus entweder logisch 1 oder 0 ist. Die hier verwendeten Temperatursensoren können zwischen -55℃ und +125℃ verwendet werden und bieten eine maximale Auflösung von $\frac{1}{16}$ Grad Celsius. Zusätzlich hat jeder dieser Temperatursensoren eine einzigartige 64-Bit lange ID, über welche der jeweilige Sensor auch bei der Kommunikation mit dem Master identifiziert wird. Die Temperaturmessung pro angeschlossenen Sensor benötigt circa 0.8 Sekunden.

Hier wurden für die Realisierung zwei GPIO-Pins verwendet. Einer dient als Spannungsversorgung, umso diese auch zu bestimmten Zeiten deaktivieren zu können. Der andere Pin ist für den 1-Wire-Bus zuständig. Pro Datenübertragungsdurchgang wird die Temperatur mit jedem der Sensoren gemessen, die an den Bus angeschlossenen sind. Je nach Anzahl der Sensoren kann dieser Vorgang mehrere Sekunden betragen, wodurch sich hier die Verwendung des Echtzeitbetriebssystems wahrlich auszahlt. Zusätzlich wird als Sensorname jeweils die einzigartige ID des Sensors verwendet. Abschließend werden die Sensorinformationen in die Warteschlange übergeben, damit diese vom Datenübertragungstask versendet werden können.

7.1.5 Konfiguration einer Sensorstation

Alle wichtigen Einstellungen einer Sensorstation wurden in einer Header-Datei namens *_globalSettings.h* zusammengefasst. Die notwendigen Einstellungen zum Betrieb sind folgende:

- Der APN[59] (Access Point Name)
- Die Zugangsdaten für die PPP-Authentifizierung (CHAP)
- Die IP-Adresse bzw. der DNS-Eintrag des Servers und dessen Port
- Der URL-Pfad zur Schnittstelle des Servers, über welche die Informationen empfangen werden.

Des Weiteren können dort noch diverse weitere Parameter eingestellt werden, die das Verhalten der Sensorstation beeinflussen. Zwei Beispiele hierfür sind die Dauer der Schlafphase als auch die Zeit, nach welcher der Watchdog abläuft.

Die Vorgehensweise die Einstellungen über eine Header-Datei zu verwalten wurde beim Prototyp der Einfachheit zuliebe gewählt. Diese ist aber für die Benutzer unpassend, da der Mikrocontroller der Sensorstation neu geflasht werden muss, um die Konfiguration ändern zu können. In dem Kapitel Ausblick (9) wird eine Möglichkeit skizziert, wie man diese Problemstellung benutzerfreundlich lösen könnte.

7.2 Umsetzung des webbasierten Servers

Das „Play Framework"[60], welches für die webbasierte Serveranwendung zum Einsatz kommt, ist RESTful was bedeutet, dass die zustandslosen HTTP-Request-Methoden als Schnittstellen genutzt werden. Play! setzt dies mit der *route* Konfigurationsdatei um, welche die Verbindung zwischen HTTP-Request und Anwendungsfunktion herstellt. Beispielhaft hier ein Eintrag aus einer *route* Datei:

```
GET     /pages/{id}         Application.showPage
```

Sendet ein Client eine GET-Anfrage mit dem URL-Pfad */pages/12* an den Server dann wird die Methode showPage der Klasse Application mit dem Parameter „12" aufgerufen.

Zusätzlich implementiert Play! das weitverbreitete Model View Controller (MVC) Entwurfsmuster, welches in Abbildung 7.4 näher dargestellt wird. Dieses spaltet die Anwendung in drei Teilbereiche, umso die Flexibilität und Erweiterbarkeit der Anwendung zu fördern.

Die drei Komponenten haben hier folgende Bedeutung und Aufgabe:

[59] Name eines Zugangspunktes, über welchen ein Provider im Mobilfunknetz Zugang zu paketorientierten Datendiensten erlaubt.

[60] In dieser Arbeit kommt die Version 1.2.1 zum Einsatz.

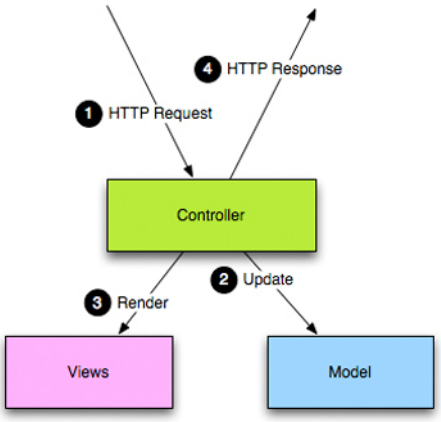

Abbildung 7.4: MVC Anwendungsmodell des Play Frameworks

Model Das Model enthält die Logik und die dazustellenden Daten. Die Daten des Models werden persistent in der Datenbank gespeichert.

Controller Alle in der route Konfigurationsdatei definierten Klassen, sind Klassen des Controllers. Dieser hat die Aufgabe je nach Anfrage die empfangenen Daten an das Model weiterzuleiten, damit dieses sich aktualisiert, oder die angeforderte Seite zu rendern mit den Daten des Models.

View Der View wird beim Play-Framework durch HTML-Seiten repräsentiert, welche sogenannte Groovy-Elemente enthalten. Mit diesem greift man auf die Java-Objekte zu, welche der Controller übergibt, und stellt diese dar.

7.2.1 Das Model

Entsprechend des Datenmodells, welches in vorherigem Kapitel aufgestellt wurde (vgl. Abbildung 6.5), wurde auch das Model realisiert. Eine solche Klasse lässt sich einfach bei Play! durch das Ableiten von der Play!-Basisklasse *Model* implementieren.

```
1  @Entity
2  public class Sensor extends Model {
3
4          @Required
5          @ManyToOne
6          public Sensorstation sensorstation;
7
8          @Required
9          public String sensorID;
10
```

```
11          public String name;
12
13          @Required
14          @OneToOne
15          public SensorType type;
16
17          @OneToMany(mappedBy="sensor", cascade=CascadeType.ALL)
18          public List<SensorValue> sensorValues;
19
20          [...]
```

Mit Annotationen werden die Beziehungen zwischen den Klassen des Models geregelt, wodurch die Hibernate XML-Mapping Dateien wegfallen. In Zeile 4 wird hier beispielsweise die Assoziation aufgestellt, dass zu einer Sensorstation ein oder mehrere Sensoren gehören. Zusätzlich können mit Annotation Eigenschaften der Attribute eingefordert werden, welche automatisch beim Erstellen oder Editieren dieser überprüft werden. Hier wird mit *@Required* gefordert, dass die Attribute nicht leer sein dürfen. Eine weitere sehr komfortable Eigenschaft ist, dass die Entitätstypen statische Methoden anbieten, um die einzelnen Entitäten einfach zu finden, löschen, zählen, erstellen oder zu editieren.

7.2.2 Die Controller

Für die Webanwendung werden zwei eigene Routen verwendet und zusätzlich Routen für die Module CRUD und Secure.

GET	/	Application.index
POST	/endpointUpdate	EndpointUpdate.endpointUpdate

Die Startseite hat die Funktion die Daten der Sensorstationen zu visualisieren. Dessen zugehöriger Controller ist in der Klasse *Application* implementiert und hat die Aufgabe die Sensorinformationen je nach Filtereinstellungen zusammenzustellen und an den View der Startseite zu übergeben. Für die Filteraktionen wurde die Bibliothek lambdaJ[61] verwendet, mit welcher sehr komfortabel Filteraktionen umsetzbar sind.

Die zweite wichtige Route ist /endpointUpdate. Diese ist für den Empfang der Sensorinformationen zuständig. Nach dem Empfang werden die Daten entschlüsselt, geparst und anschließend dem Model hinzugefügt. Zugleich werden die im empfangenen Messwerte der Sensoren überprüft, ob diese in Ihrem kritischen Bereich liegen, welcher von der Art des Sensors vorgegeben wird. Sollte dies der Fall sein, werden die Benutzer, welche die Benachrichtigungsoption aktiviert haben, via E-Mail informiert.

[61] http://code.google.com/p/lambdaj/

Das Play Framework ist um Module erweiterbar. Von Haus aus werden gleich die zwei sehr nützlichen Module CRUD und Secure mitgeliefert, mit welchen sowohl ein Log-in als auch ein administrativer Bereich sehr einfach generiert werden können.

```
1  package controllers;
2
3  import controllers.CRUD;
4  import controllers.Secure;
5  import play.*;
6  import play.mvc.*;
7
8  @With(Secure.class)
9  public class Sensors extends CRUD {
10
11 }
```

Mit der Klasse Sensors werden automatisch im administrativen Bereich Seiten angelegt um die Entitäten des Typs *Sensor* anzuzeigen, zu editieren und löschen zu können. Natürlich besteht auch die Möglichkeit neue Elemente per Hand hinzuzufügen. Die Konvention gibt hier vor, dass die CRUD-Klassen als Namen den Plural der zu betreuenden Entität tragen. Die Annotation in Zeile 8 bezieht sich auf das Secure-Modul. Diese Annotation kann vor jeder Klasse des Controllers stehen und bedeutet, dass man eingeloggt sein muss um die Seite, welche dieser Controller betreut, aufzurufen können. Falls dies nicht der Fall ist, wird man automatisch auf die Log-in-Seite weitergeleitet.

Bei dieser Webanwendung nutzen alle Controller-Klassen die Secure-Annotation, da Zugriff auf diese Webanwendung nur mit Benutzername und Passwort möglich sein soll. Zusätzlich haben alle Entitätstypen jeweils eine CRUD-Klasse, damit jede Entität auch über die Webseite verwaltet werden kann.

7.2.3 Die Views

An Views, welche HTML-basiert sind, musste mindestens die Startseite entworfen werden für diese Webanwendung. Diese hat die Aufgabe die Sensorinformationen in Diagrammen über die Zeit darzustellen und soll die Möglichkeit dem Nutzer geben die Daten nach Datum, Sensorstation und/oder Sensortyp zu filtern.

Die Abbildung 7.5 zeigt in Aktion das fertige Ergebnis der Startseite. Auf der linken Seite findet man die Filtermöglichkeiten und rechts werden die Informationen der Sensorstationen dargestellt. Diese Informationen werden nach Sensorstation geordnet und danach je nach Sensortyp dargestellt. Pro Diagramm fasst die Webseite automatisch bis zu fünf Sensoren des gleichen Sensortyps in einem Diagramm zusammen.

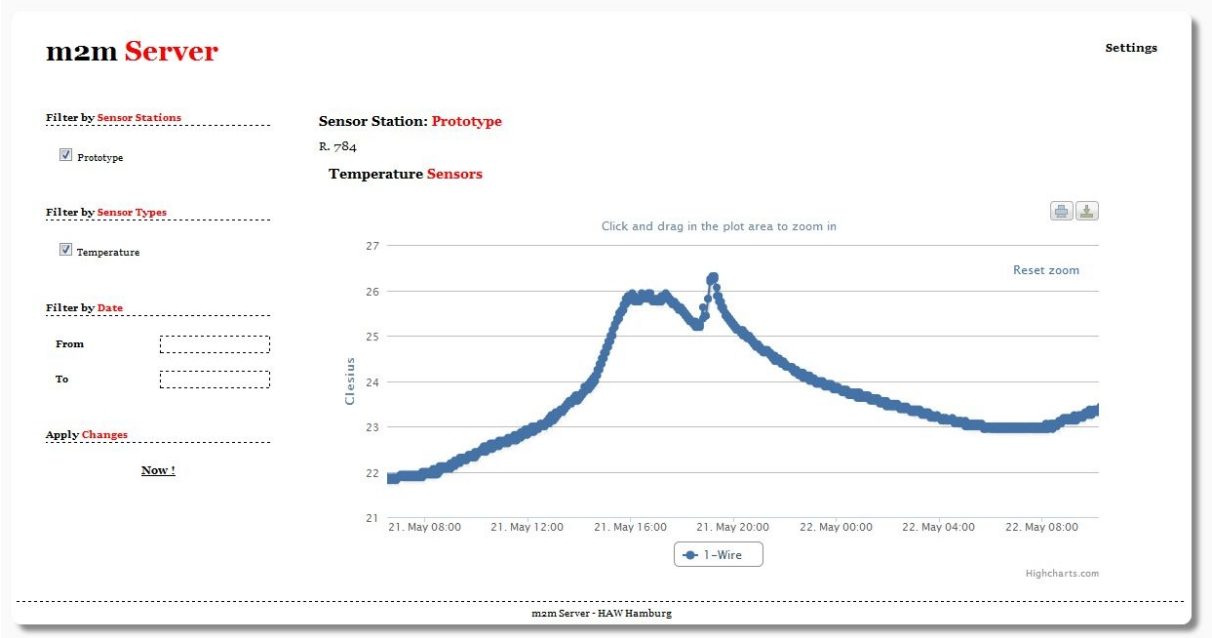

Abbildung 7.5: Screenshot der Startseite, welche den Temperaturverlauf des Raums 784 am 21.Mai 2011 zeigt.

Umgesetzt wurde die Webseite mit modernen Webtechnologien. Als Grundlage für die Diagramme wurde das JavaScript-basierte Framework Highcharts[62] gewählt. Dieses zeichnet basierend auf Datenpaaren ansprechende Diagramme, welche zudem noch die Möglichkeit bieten, dass man hineinzoomen kann[63]. Für weitere visuelle Filtermöglichkeiten und für eine einfache Auswahl des Filterdatums wird von der JavaScript-Klassenbibliothek jQuery[64] Gebrauch gemacht.

Die Views greifen mit Groovy-Elementen auf die Java-Objekte zu, welche durch den jeweiligen Controller mit dem Befehl *render* an sie übergeben werden. Groovy-Elemente bieten die Möglichkeit einfache Kontrollstrukturen zu definieren. Hier ein einfaches Beispiel, welches

[62] http://www.highcharts.com/

[63] Die Usability in Hinblick auf Smartphones wurde nicht überprüft. Insbesondere die mobilen Browser haben zum Teil Probleme mit diesem Framework. Beispielsweise unter Android in Verbindung mit Firefox werden die Graphen dargestellt, aber das Hineinzoomen funktioniert nicht, da der Browser die Multitouch Gesten nicht umsetzen kann. Sollte in der Zukunft die Notwendigkeit aufkommen, dass die Webseite auch mit einem Handy kompatibel sein muss, muss überprüft werden, ob man nicht eine technisch weniger fordernde Chart-Bibliothek wählt. Hierbei müsste man dann vermutlich aber bei der Funktionalität Kompromisse eingehen.

[64] http://jquery.com/
Highcharts benötigt diese Klassenbibliothek ebenfalls.

alle verfügbaren Sensorstationen für den Filterbereich der Startseite als HTML-Checkboxen darstellt:

```
1  <div class="filterBox">
2          <h3>Filter by <span>Sensor Stations</span></h3>
3          <div class="filterGroup">
4                  #{list items:sensorstations, as:'station'}
5                          <label>
6                                  <input type="checkbox" name="CheckedSensorstations" id="${
                                       station.name}" value="${station.name}" checked="checked">
7                                  ${station.name}
8                          </label>
9                          <br>
10                 #{/list}
11         </div>
12 </div>
```

Durch den Controller wurde eine Liste mit dem Namen *sensorstations* übergeben. In Zeile 4 wird über diese Liste mit Groovy iteriert. Mit dem Syntax, wie er in Zeile 7 zu sehen ist, kann man auf ein Objekt zugreifen. Hier wird auf die öffentliche Eigenschaft *Name* des jeweiligen Sensorstation-Objekts zugegriffen. Ein Aufruf von Methoden ist auch denkbar, zum Beispiel wenn diese einen String zurückliefern oder um einen Link zu erstellen. Zusätzlich bietet das Play Framework noch sogenannte Tags. Diese funktionieren ähnlich wie das Konzept #define bei der Programmiersprache C, nur das hier der Inhalt des Defines eine HTML-Seite ist[65].

Zusätzlich wurde auch noch ein einfacher administrativer Bereich gestaltet, in dem die automatisch generierten CRUD-Seiten integriert wurden.

7.2.4 Die Softwaretests

Die entwickelte Webanwendung kann natürlich auch mit diversen Softwaretests automatisiert getestet werden. Hierzu bietet das Play Framework drei Testarten an. Zum einen Unit-Tests, mit welchen die Softwaremodule des Models getestet werden können. Die zweite Testmöglichkeit sind Funktionstests. Mit diesen Tests kann man unterschiedliche HTTP-Anfragen an die Controller-Klassen testen, wobei man hier unter anderem den von der Webanwendung zurückgegebenen HTTP-Statuscode überprüfen kann. Diese beiden Testarten basieren beide auf der bekannten JUnit-Bibliothek[66], wodurch sich Tests in vertrauter Manier schreiben lassen. Die dritte Testart ist der sogenannte Selenium-Test. Selenium[67] ist ein Test-Framework,

[65] Parameter können ebenfalls an Tags übergeben werden
[66] http://www.junit.org/
[67] http://seleniumhq.org/

mit welchem es möglich ist Befehle, die man auf einer Webseite tätig, aufzunehmen. Diese Sequenz an Befehlen lässt sich dann automatisiert wiedergeben, wobei geprüft wird, ob der Weg durch die Webanwendung entsprechend des aufgezeichneten Pfades immer noch möglich ist.

Im Entwicklungsmodus lässt sich mit dem Aufruf des URL-Pfades *@tests* der „Tests runner" öffnen, mit welchem die unterschiedlichen Test-Suits gewählt und ausgeführt werden können. Die Abbildung 7.6 zeigt den erfolgreichen Abschluss aller Tests, die bei der Webanwendung implementiert wurden.

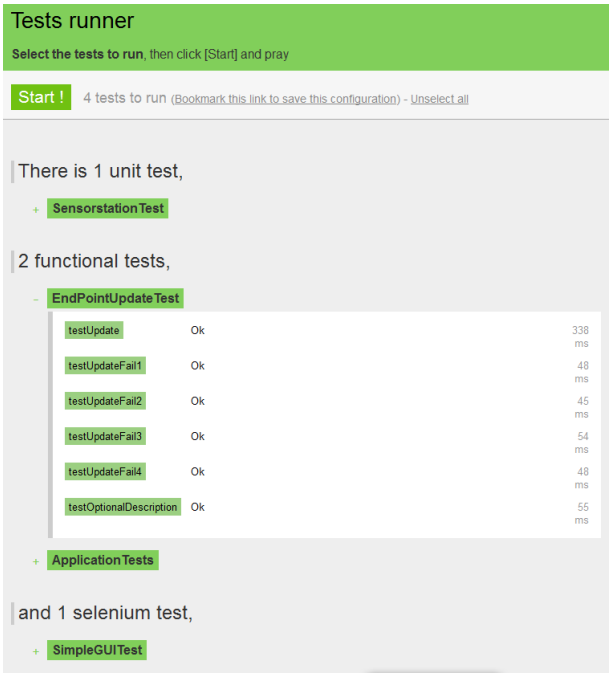

Abbildung 7.6: Die Tests des Servers

7.2.5 Sonstige Funktionen & Eigenschaften

Das Play Framework enthält standardmäßig die SQL-Datenbank H2[68], welche dateibasiert arbeitet. Dies ist mehr als ausreichend für diesen Prototyp, sollten jedoch sich die Anforderungen an die Datenbank ändern, kann diese durchaus sehr schnell ausgetauscht werden, beispielsweise durch eine Oracle-Datenbank.

Zusätzlich ist in Play! auch eine Scheduling-Funktion eingebaut. Mit dieser wurde ein asynchroner Job umgesetzt, welcher periodisch prüft, ob von jedem Sensor in der letzten Zeit neue Messwerte eingetroffen sind. Sollte ein Sensor oder gar eine ganze Sensorstation ausgefallen sein, werden die Nutzer per E-Mail darüber informiert.

Die Webanwendung wurde so konfiguriert, dass sie sowohl einen HTTPS-Zugang für die Benutzer als auch einen HTTP-Zugang für die Sensorstationen bietet. Diese zwei Zugänge sind notwendig, da die Sensorstationen HTTPS nicht unterstützen und deswegen die AES-Verschlüsselung nutzen. Mit dieser Vorgehensweise läuft nun jegliche Kommunikation verschlüsselt ab.

[68] http://www.h2database.com

Kapitel 8

Messtechnische Evaluation einer Sensorstation

In den bisherigen Kapiteln wurde wiederholt gefordert, dass eine Sensorstation energieeffizient arbeiten soll, umso eine möglichst lange Batterielaufzeit zu erzielen. Zudem wurde gefordert, dass diese eine möglichst hohe Übertragungsgeschwindigkeit erzielen soll, wodurch sich eine solche Sensorstation für eine Vielzahl an Szenarien einsetzen lässt. In diesem Kapitel wird nun eine Sensorstation auf dessen typischen Energieverbrauch untersucht in Zusammenhang mit dessen zeitlichem Übertragungsverhalten. Zusätzlich werden Angaben zu der hier maximal möglichen Übertragungsgeschwindigkeit gemacht.

8.1 Der Stromverbrauch und das zeitliche Verhalten

Die Abbildung 8.1 zeigt den Stromverbrauch einer Sensorstation und das zeitliche Verhalten eines Übertragungsvorgangs. Der Mikrocontroller wurde während dieser Messung mit 16 MHz[69] getaktet und es wurden circa 2 Kilobyte Nutzdaten übertragen. Das LCD-Display, welches sich auf dem Development-Board befindet, war während der Messung deaktiviert, um die Messwerte nicht zu verfälschen. Der UMTS Stick wurde so konfiguriert, dass kein Modeswitch-Vorgang als auch keine PIN-Eingabe vorgenommen werden muss.

Gemessen wurde der Stromverlauf indirekt über den Spannungsabfall eines niederohmigen Widerstandes, welcher in Reihe geschaltet wurde. Anschließend wurden die Messdaten gemäß des Ohmschen Gesetzes umgerechnet und durch ein geglättetes Spline dargestellt.

Die unterschiedlich markierten Teilbereiche sind folgende:

[69] 16 MHz ist das Maximum des internen Präzisions-Oszillators.

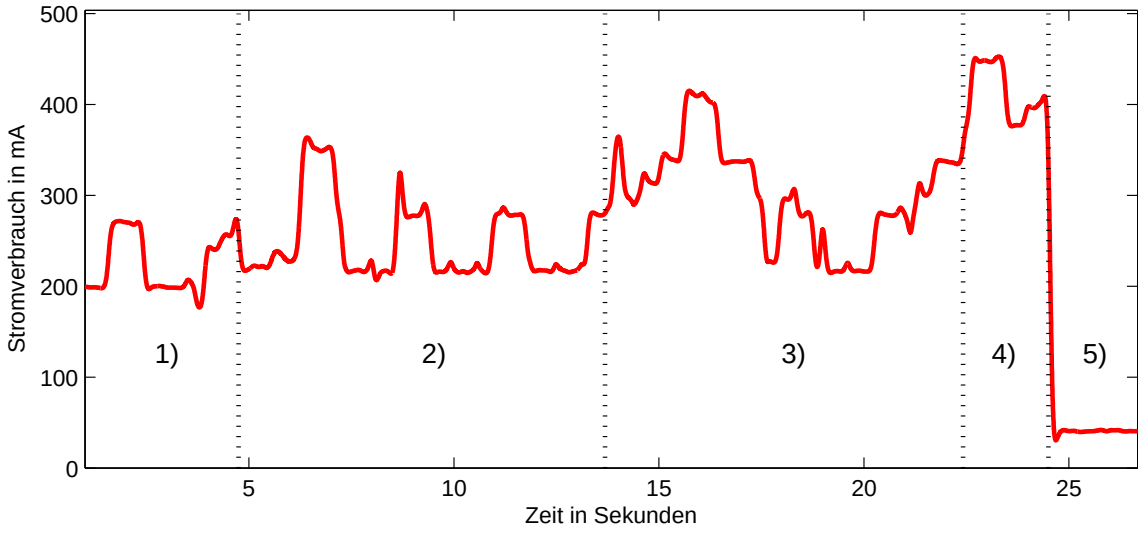

Abbildung 8.1: Der Stromverbrauch einer Sensorstation während eines Übertragungsvorgangs.

1) Die Zeit, die der UMTS Stick benötigt, bis dieser angesprochen werden kann mit AT-Befehlen. In diesem Intervall ist die Zeit enthalten, welche der UMTS Stick für den Reset und die USB-Enumerierung benötigt. Zudem wird in dieser Zeit die Messung der Temperatur mittels zwei Temperatursensoren vorgenommen.

2) Die Dauer, die die SIM-Karte des UMTS Sticks für die Initialisierung benötigt.

3) Zeitspanne des PPP-basierten Internetverbindungsaufbaus.

4) Dieses Intervall enthält hier folgende Aktionen: Eine DNS-Abfrage, die Herstellung der TCP-Verbindung zum Server, die Aufbereitung der Sensorinformationen und Verschlüsselung dieser, die Datenübertragung und der Empfang der Serverantwort, die Beendigung aller Verbindungen und abschließend der Wechsel in den Energiesparmodus.

5) Ab Sekunde 24.5 befindet sich die Sensorstation im Energiesparmodus.

Der Prototyp benötigt für die Übertragung der Temperaturmesswerte an den Server im Normalfall 23-28 Sekunden. Diese sehr lange Übertragungszeit, welche sich negativ auf die Energieeffizienz der Sensorstation auswirkt, ist technologiebedingt und bietet wenig Spielraum für weitere Optimierungen. Jeder UMTS Stick benötigt mehrere Sekunden, bis dieser angesprochen werden kann. Zusätzlich benötigt danach die SIM-Karte noch einige Zeit, um sich im Mobilfunknetz anzumelden. Nach diesem Vorgang ist der Stick betriebsbereit. Diese zwei technologiebedingten Vorgänge verlängern die Übertragungszeit hier in diesem Fall schon um 13,67 Sekunden.

Der zweite Aspekt, der die Übertragung wesentlich verzögert, ist der PPP-basierte Internetverbindungsaufbau. Insbesondere das IPCP-Protokoll, welches mit dem Internet-Service-Provider aushandelt, welche IP-Adresse der Sensorstation zugewiesen wird, sorgt für eine deutliche Verzögerung, da der ISP nur sehr langsam auf Anfragen antwortet. Das IPCP-Protokoll ist zudem auch dafür verantwortlich das die Übertragungszeit um bis zu 5 Sekunden variieren kann. Der letzte zeitliche Abschnitt vor dem Energiesparmodus enthält die eigentliche Übertragung. Ein Großteil dieser Zeit kommt auf Grund der Round Trip Time zustande, welche im dreistelligen Millisekundenbereich liegt.

Der Energiesparmodus, welcher ab Sekunde 25 aktiv ist, benötigt mit ungefähr 35 mA sehr viel Energie. Das Datenblatt des Mikrocontrollers jedoch gibt für den Deep-Sleep Modus in Kombination mit einer Taktrate von 468 Hz, wie dies hier auch für den Energiesparmodus implementiert wurde, einen durchschnittlichen Verbrauch von 550 µA an [Texas Instruments (2010a) - Kapitel 26.1.8.1]. Die nähere Betrachtung des Schaltplans von dem verwendeten Development-Board kann jedoch diese signifikante Differenz zwischen dem hier gemessenen Verbrauch und dem theoretischen Verbrauch erklären. Laut Schaltplan befinden sich auf diesem Development-Board eine Reihe von Komponenten, welche nicht für den Energiesparmodus benötigt werden aber trotzdem mit Spannung versorgt werden. Das offensichtlichste Beispiel ist hier die Power-LED, welche andauernd 10 mA verbraucht.

Diese Komponenten können nicht per Software deaktiviert werden, wodurch in dieser Arbeit kein niedrigerer Stromverbrauch erzielt werden kann mit der hier verwendeten Hardware-Plattform. Bei Einsatz einer eigens für die Sensorstation gestalteten Platine, welche nur die zwingend notwendigen Bauteile enthält, dürfte die Sensorstation einen Energieverbrauch von unter 1 mA während des Energiesparmodus aufweisen. Eine solche Platine konnte nicht im Rahmen dieser Arbeit entwickelt werden, soll jedoch in der Zukunft für die Sensorstationen zum Einsatz kommen.

Des Weiteren muss hier noch erwähnt werden, dass eine genauere Zuteilung des Stromverbrauchs zu den einzelnen verwendeten Komponenten an einem bestimmten Zeitpunkt nicht möglich ist. Das Datenblatt des Mikrocontrollers gibt nur eine vorläufige Stromverbrauch-Spezifikation für 50 MHz bei 25℃ Umgebungstemperatur an. Detailliertere Angaben in Bezug auf unterschiedliche Taktfrequenzen oder wie sich der Betrieb bei unterschiedlichen Temperaturen verhält, sind nicht gegeben. Zudem können auch keine Angaben zu dem UMTS Stick gemacht werden, da ein solcher kein Datenblatt besitzt. Es jedoch davon auszugehen, dass der Stromverbrauch des UMTS Sticks von der Nutzung, der Empfangsstärke als auch von der Umgebungstemperatur beeinflusst wird.

Es wurde zudem auch der Versuch unternommen die Sensorstation mit einer höheren Taktrate zu betreiben, umso die Wachphase zu verkürzen, was zu einem niedrigeren Energieverbrauch führen würde. Das Ergebnis dieser Untersuchung zeigt die Abbildung 8.2:

Eine Sensorstation, welche mit 80 MHz getaktet ist, hat im Schnitt einen 50mA höheren

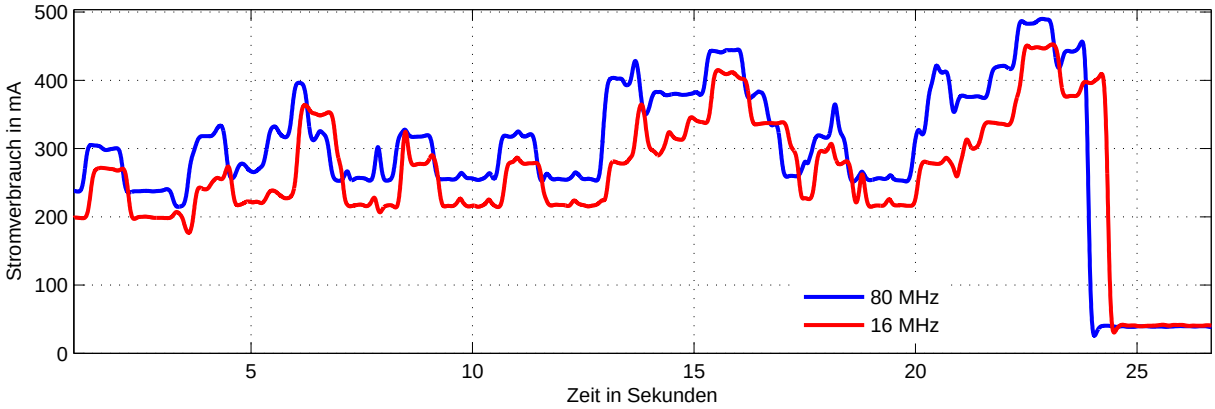

Abbildung 8.2: Untersuchung des Energieverbrauches in Abhängigkeit der verwendeten Taktrate.

Stromverbrauch und ist dabei nur um wenige Hundert Millisekunden schneller. Dies zeigt, dass man hier durch eine Erhöhung der Taktrate keine kürzeren Übertragungsgeschwindigkeiten erzielen kann und damit nur die Energieeffizienz negativ beeinflusst.

Dennoch muss man erwähnen, dass eine höhere Taktfrequenz in bestimmten Fällen auch sinnvoll sein kann. Hier ist die benötigte Zeit für das Auslesen der Sensoren als auch die zu übertragende Datenmenge sehr gering. Bei einem anderen Anwendungsfall kann dies durchaus anders aussehen. Müssten beispielsweise sehr viele Daten pro Wachphase übertragen werden, dann würde dies dazu führen, dass die Zeit, die für das Aufstellen des XML-Übertragungsformates und dessen Verschlüsselung notwendig ist, stark steigen wird. Ein anderer kritischer Fall ist, wenn die Tasks der Sensoren sehr viel Rechenzeit benötigen, da so der Datenübertragungstask verzögert werden würde. In beiden Fällen müsste man prüfen, ob eine höhere Taktrate sinnvoll ist, umso unter dem Strich weniger Strom zu verbrauchen.

Aufgrund der gewonnenen Daten bezüglich des Stromverbrauchs ist nun auch eine Hochrechnung über die mögliche Betriebsdauer einer Sensorstation unter Verwendung einer Batterie möglich. Im Einzelhandel sind vermehrt sogenannte USB-Akkupacks für Smartphones verfügbar mit einer Kapazität von beispielsweise 5000 mAh. Solch ein Akkupack soll hier die Grundlage der Hochrechnung sein, wobei die Verwendung der Sensorstation auch mit anderen Batteriearten denkbar ist. Die Übertragung mit 16 MHz ergibt einen durchschnittlichen Stromverbrauch von 250.02 mA. Bei der Übertragungszeit ist von durchschnittlich 25 Sekunden auszugehen. Zusätzlich wird für die Zeit des Energiesparmodus ein Verbrauch von einer Milliamperesekunde angenommen. Alle 10 Minuten sollen die Messwerte übertragen werden. Dies ergibt folgende Anzahl an Übertragungen und folgende Akkulaufzeit:

Benötigte Ladung für eine Schlafphase: ψ = Dauer einer Schlafphase * Mittelwert der Stromstärke im Energiesparmodus

Benötigte Ladung für eine Wachphase: ζ = Übertragungsdauer * Mittelwert der Stromstärke während einer Übertragung

$$\text{Anzahl Übertragungen}\, n = \frac{\text{Akku Kapazität}}{\psi + \zeta}$$

$$n = \frac{5000\,mAh}{(10min * 1\,mA) + (25s * 250.02\,mA)} \Rightarrow 2628\,\text{Übertragungen}$$

$$2628 * (10min + 25s) = 19.01\,\text{Tage}$$

Die errechnete Betriebsdauer einer Sensorstation von knapp 19 Tagen basiert auf einem Durchschnittswert, dennoch sollte ein Betrieb ohne Akkuwechsel für mehr als 15 Tage möglich sein. Dies unter der Vorrausetzung, dass die Sensorstationen in der Zukunft eine passende Platine erhalten wird, welche keine unnötigen Komponenten enthält. Mit einer solchen Batterielaufzeit stellt sich die komplette M2M-Anwendung wartungsarm dar und erfüllt die Anforderungen des Konzeptes.

Betrachtet man das Verhältnis zwischen der insgesamt benötigten Ladung für die Wachphase und die der Schlafphase, so ergibt sich folgendes Verhältnis, welches in Abbildung 8.3 dargestellt wird.

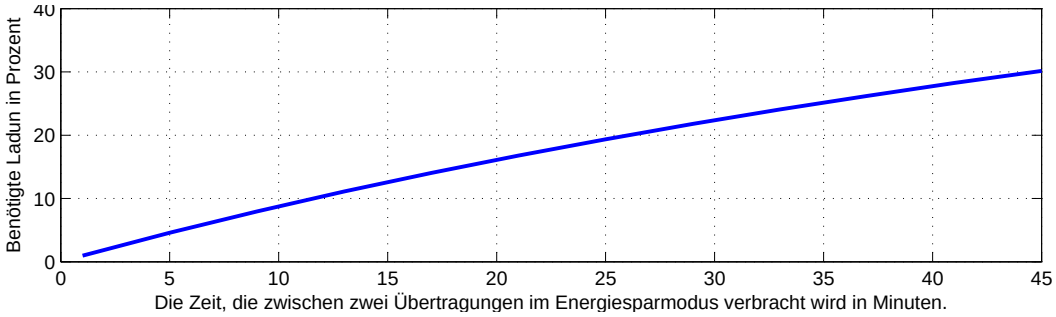

Abbildung 8.3: Prozentual dargestellt wird die insgesamt benötigte Ladung für den Energiesparmodus bei der Verwendung eines 5000-mAh-Akkus. Der prozentuale Anteil der Ladung, der von der Wachphase verbraucht wird, ergibt sich, indem man den Wert der Schlafphase von 100% subtrahiert. Diese Berechnung basiert auf der zuvor vorgestellten Formel.

In dem hier vorgestellten Beispiel, bei welchem im Takt von 10 Minuten Sensorinformationen übertragen werden, benötigt der Energiesparmodus gerade einmal 8,76%, obwohl die Sensorstation 95,55% der 19,1 Tage in diesem Modus verbringt. Ein solches oder ein noch kleineres Übertragungsintervall ist typisch für diese Art von Anwendung, was bedeutet, dass die benötigte Ladung für die Schlafphase als gering anzusehen ist.

8.2 Die mögliche Übertragungsgeschwindigkeit und die Zuverlässigkeit

Die Zuverlässigkeit der kompletten M2M-Anwendung wurde zudem mehrfach über längere Zeiträume getestet. Schon in einem der ersten Tests übertrug der Prototyp der Sensorstation zuverlässig und ohne Ausfall über 3 Tage lang alle 5 Minuten die aktuelle Temperatur an den Server. Die Abbildung 8.4 zeugt vom erfolgreichen Abschluss des Testes.

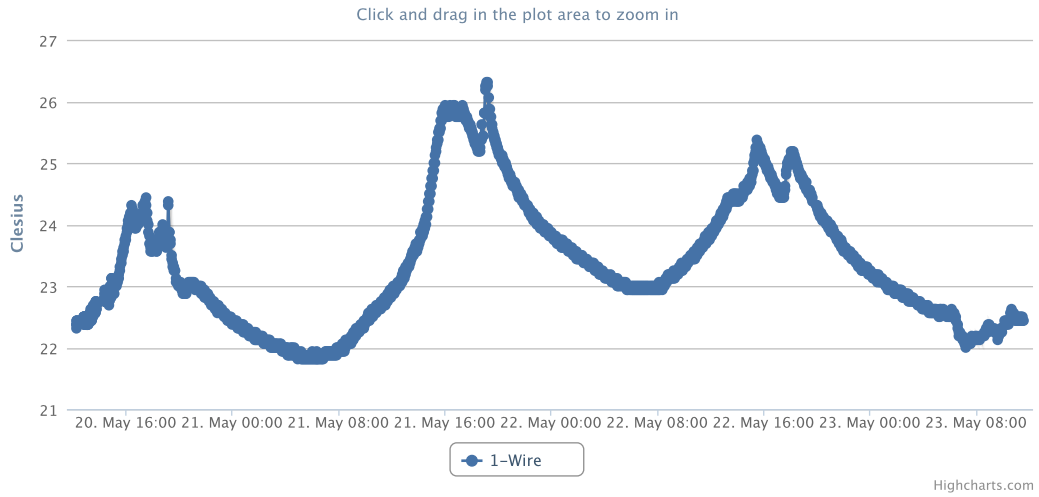

Abbildung 8.4: Empfangene Temperaturmesswerte vom 20. Mai bis 23. Mai im 5-Minutentakt.

Die Übertragungsgeschwindigkeit einer Sensorstation zusammen mit den Round Trip Zeiten der jeweiligen TCP-Pakete wurde mit dem verbreiteten Netzwerkanalyse-Tool Wireshark gemessen. Die Abbildung 8.5 zeigt, dass eine Übertragung mit durchschnittlich 100 $\frac{kB}{s}$ nach einigen Sekunden möglich ist. Dabei verwendet die Sensorstation ein 14 kB großes TCP-Fenster. Dieses klingt relativ klein im Verhältnis zu dem 96 Kilobyte großen Arbeitsspeicher des verwendeten Mikrocontrollers. Diese hat sich jedoch nach einigen Tests als optimale Größe erwiesen. Dies liegt daran, dass die Anwendung für viele andere Bereiche ebenfalls Speicherpools benötigt. Ein Beispiel hierfür ist der USB-Ringpuffer, welcher die durch die USB Interrupt Service-Routine empfangenen Daten des UMTS Sticks zwischenspeichert. Momentan wird der Arbeitsspeicher der Sensorstation fast vollständig genutzt, was kaum eine größere Fenstergröße erlaubt. Zusätzlich sollte hier noch erwähnt werden, dass hier konstante Daten versendet wurden. Anders als beim Datenübertragungstask wurde hier keine Rechenzeit aufgewendet um die Daten, welche versendet werden sollen, zu erstellen.

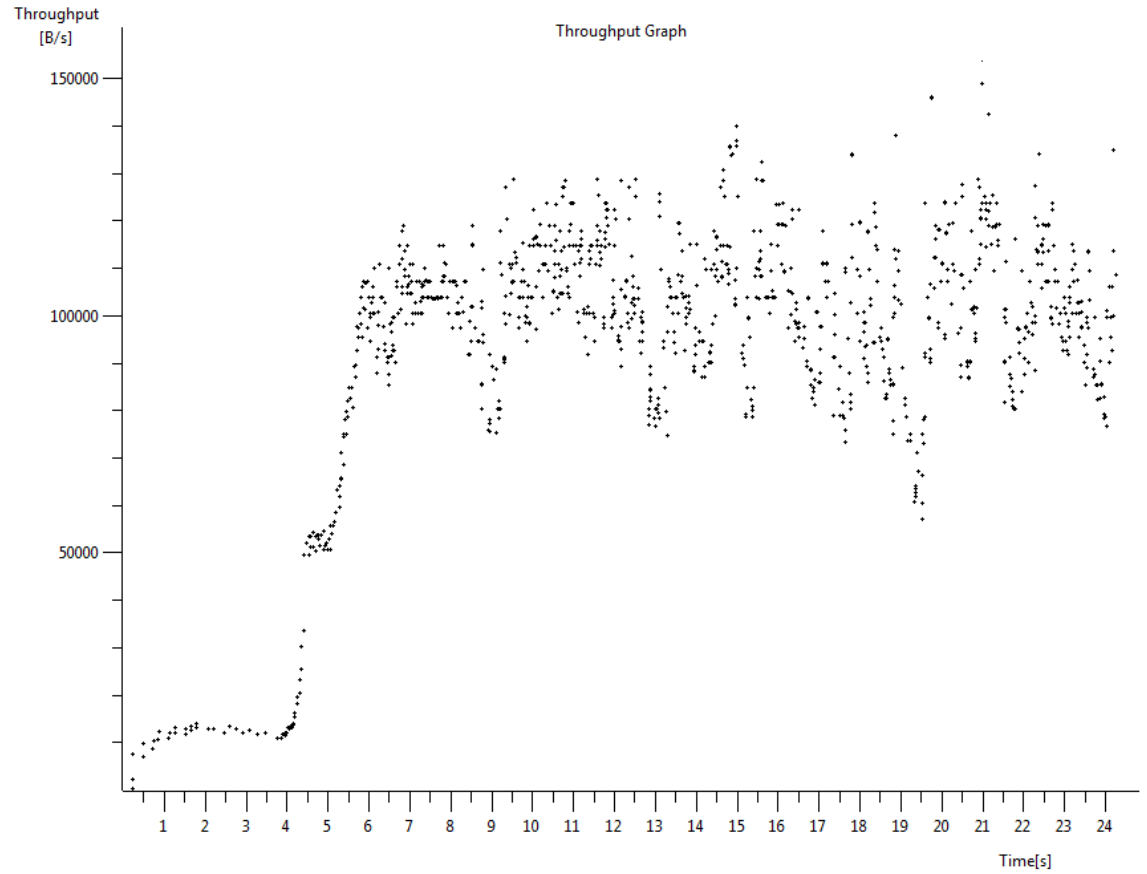

Abbildung 8.5: Messung der TCP-Übertragungsgeschwindigkeit einer Sensorstation mit Wireshark.

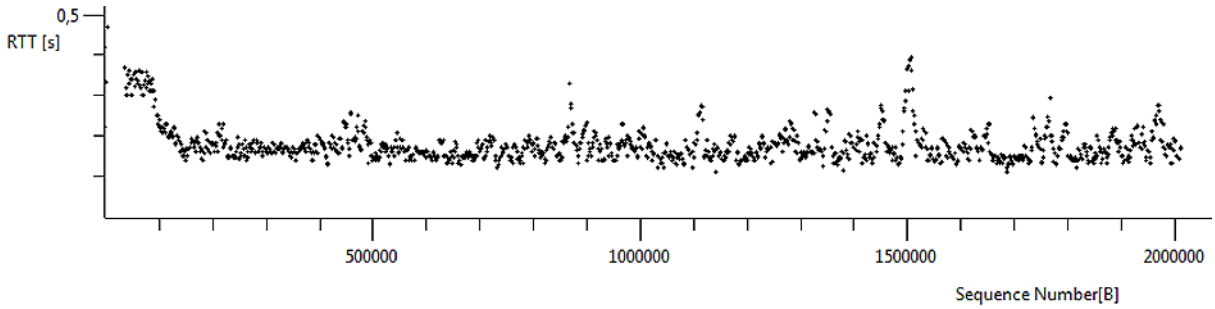

Abbildung 8.6: Die Round Trip Zeiten der einzelnen TCP-Pakete, welche bei der Messung der Datenübertragungsrate übertragen wurden. Die Sequenznummern werden durch Wireshark relativ dargestellt, was bedeutet, dass die Nummerierung bei dem ersten Paket mit null beginnt.

Das in Kapitel 3.3.2 vorgestellte "Sliding-Window" Verfahren zeigt den Zusammenhang zwischen dem Speicherverbrauch und möglichen Übertragungsgeschwindigkeiten bei geringen

TCP-Fenstergrößen auf. In Zusammenhang mit den Round Trip Zeiten der einzelnen übertragenen Pakete, wie sie in Abbildung 8.6 dargestellt sind, lässt sich der lineare Zusammenhang zwischen der Datenübertragungsrate der Pakete und der Round Trip Time grob bestätigen, sobald sich der Datendurchsatz ab Sekunde 6 auf die 100 $\frac{kB}{s}$ eingepegelt hat. Genauerer betrachtet hängt jedoch der TCP-Datendurchsatz von einer Vielzahl von weiteren Parametern ab, wobei man auf die meisten keinen Einfluss hat, da diese entweder in Hand des Mobilfunkproviders liegen oder aus der verwendeten Technologie resultieren. Zudem wird die Round Trip Time stark durch den Stau beim Mobilfunkbetreiber beeinflusst was dafür sorgt, dass unter anderem hohe Geschwindigkeiten am Anfang der Verbindung verhindert werden.

Als Fazit zu ziehen ist Folgendes: Die hier erreichte Übertagungsgeschwindigkeit mit rund 100 $\frac{kB}{s}$ dürfte ausreichend sein für fast alle Anwendungen. Sollten nur wenige Daten übertragen werden dann ist die geringe technologiebedingte Datenübertragungsrate, welche am Anfang vorherrscht, unkritisch, da die Übertragung sehr schnell beendet ist. Benötigt man im Mittel eine höhere Übertragungsgeschwindigkeit, umso beispielsweise die Energieeffizienz zu verbessern, muss man mehr Speicher und gegebenenfalls bessere Hardware verwenden. Hierbei muss jedoch in Kauf genommen werden, dass dadurch die Anschaffungs- bzw. Produktionskosten steigen.

Kapitel 9

Ausblick

Ein Prototyp zeigt die Machbarkeit des vorgestellten Konzeptes mit all dessen Eigenschaften auf, bietet jedoch aber normalerweise auch Raum für Weiterentwicklungen. Dies ist auch hier der Fall und dieser Abschnitt stellt mögliche Weiterentwicklungen vor, um aus diesem Prototyp ein Produkt zu schaffen.

Der wichtigste Schritt in diese Richtung wurde schon bei der Diskussion bezüglich des Stromverbrauchs einer Sensorstation deutlich. Die Sensorstationen benötigen eine eigens für sie entworfene Platine, welche nur die benötigten Bauteile enthält, umso den Energieverbrauch im Energiesparmodus auf ein Minimum reduzieren zu können. Zusätzlich muss die physische Form der Sensorstationen noch evaluiert werden. Eine dieser Fragenstellung ist, wie das Gehäuse für die Sensorstation aussehen und gestaltet sein muss, sodass es den Umwelteinflüssen trotzen kann.

Auf der Softwareseite sind auch noch Problemstellungen zu lösen. Bisher kann eine Sensorstation nur konfiguriert werden, indem der Mikrocontroller der Sensorstation neu geflasht wird. Dies ist für die meisten Nutzer eines solchen Systems nicht praktikabel. Eine denkbare Lösungsstrategie wäre Folgende: Die unabdingbaren Informationen, die für eine Internetverbindung benötigt werden, wie beispielsweise welcher APN zu nutzen ist, werden auf einer SD-Karte abgelegt, welche man in den entsprechenden UMTS Stick der Sensorstation steckt. Zusätzlich bietet jede Sensorstation einen Knopf, welcher einen Pairing-Vorgang auslöst. Während dieses Vorgangs stellt die Sensorstation eine Verbindung zum Server her und bietet dem Nutzer die Möglichkeit die Sensorstation via Web zu konfigurieren. Diese Konfigurationsdaten empfängt die Sensorstation und speichert diese dann persistent.

Ein weiteres Softwarefeature, welches vorteilhaft wäre, ist eine Hardwareabstraktionsschicht einzuführen, um es den Sensorstationen zu ermöglichen unterschiedliche Mikrocontroller-Plattformen als Basis zu verwenden. Das verwendete Echtzeitbetriebssystem FreeRTOS bietet bereits eine solche Schicht und unterstützt damit sehr viele unterschiedliche Plattformen. Die Aufgabe hier wäre eine Hardwareabstraktionsschicht sowohl für den USB-Stack

als auch die sonstigen angesprochenen Hardwarebausteine einzuführen, um so die Sensorstationen herstellerunabhängig gestalten zu können.

Weitere denkbare Verbesserungen wären folgende: Die Schlafzeiten der Sensorstationen könnten dynamisch durch den Server angepasst werden, je nachdem ob sich die empfangenen Messwerte nah oder weit entfernt von dem kritischen Bereich befinden. Eine weitere Idee wäre es, Statusmeldungen beispielsweise über den Batteriestand einzuführen, welche durch die Sensorstationen versendet werden. Zudem könnte noch geprüft werden, ob durch eine andere Taktung der Sensorstationen oder ein weiteres Auslagern von Aufgaben in neue Tasks noch mehr Energie eingespart werden kann. Auch auf der Serverseite besteht noch Erweiterungsbedarf beispielsweise durch die Einführung von Benutzergruppen.

Abschließend noch wenige Worte zur Sicherheit: Wie schon erwähnt wurde, bietet die Verschlüsselung, welche die Sensorstation zum Datenversand verwendet, Angriffsmöglichkeiten. Es ist zu prüfen, ob diese nicht gegen eine stärkere Verschlüsselung ausgetauscht werden kann. Zudem müssen noch Überlegungen angestellt werden bezüglich des Schlüsselmanagements und der physischen Sicherheit einer Sensorstation, damit zum Beispiel Diebstahl verhindert wird.

Kapitel 10

Resümee

In der vorliegenden Arbeit wurde eine Machine-to-Machine Anwendung entworfen und die Machbarkeit mit einem Prototyp nachgewiesen (Proof of concept). Die M2M-Anwendung ermöglicht es Informationen aus Gegenden ohne Energieversorgung den Benutzern zur Verfügung zustellen. Dabei ist der Benutzer beim Zugriff auf diese Informationen ortsungebunden und benötigt einzig allein einen Internet-fähigen Computer.

Strukturell gesehen basiert die Anwendung aus mindestens einer mobilen Sensorstation, welche für die Informationserfassung und deren Versand zuständig ist, und genau einem webbasierten Server. Als Sensorstation-Plattform wurde ein ARM Cortex-M3 Mikrocontroller zusammen mit einem handelsüblichen UMTS Stick gewählt. Diese Kombination ermöglicht eine hohe Effizienz bei niedrig bleibenden Hardwarekosten. Softwareseitig wurde das Echtzeitbetriebssystem FreeRTOS implementiert, um unter anderem die Modularität zu fördern. Zudem findet man bei einem Mikrocontroller-System keine Sockets, wie man sie beispielsweise von Linux kennt. Deswegen wurde zur Kommunikation mit dem Server ein USB-Stack mit dem TCP/IP Stack lwIP verknüpft. Die Kommunikation zwischen den Sensorstationen und dem Server geschieht über das Internet und wurde mittels HTTP-POST realisiert, wobei die Übertragung zudem AES verschlüsselt abläuft.

Da es die Zielsetzung erfordert Informationen, wie beispielsweise Sensormesswerte, aus Gegenden ohne Energieversorgung zu übertragen, führt dies sofort zu der Schlussfolgerung, dass die Sensorstationen energiesparend entworfen werden müssen. Um diese Forderung zu erfüllen, wechseln die Sensorstationen nach jeder Übertragung für eine bestimmte Zeit in den Energiesparmodus. Dieser ist dadurch gekennzeichnet, dass alle Hardware-Bausteine (bis auf einen Timer) deaktiviert sind. Diese Vorgehensweise hat sich als guter Weg herausgestellt, wobei hier zugleich auch technologiebedingte Probleme aufkamen. Da im Energiesparmodus auch die USB-Schnittstelle deaktiviert ist und somit auch der UMTS Stick, ist es erforderlich, bei jeder Übertragung den Stick neu zu initialisieren und die Internetverbindung neu aufzubauen. Es wurde festgestellt, dass ein UMTS Stick inklusive betriebsbereiter

SIM-Karte bis zu 15 Sekunden benötigt bis dieser verwendet werden kann[70]. Zusätzlich benötigt der PPP-Verbindungsaufbau in das Internet wegen des Mobilfunkproviders mehrere Sekunden. Somit muss mit circa 25 Sekunden für eine komplette Übertragung gerechnet werden. Trotz dieser langen Übertragungszeit sollte man eine Sensorstation in Kombination mit einem Akku, welcher eine vergleichbare Kapazität hat wie die eines Notebooks, je nach Betriebsweise[71] über mehrere Wochen betreiben können. Zusätzlich konnte durch die Verwendung von UMTS eine hohe Übertragungsgeschwindigkeit erzielt werden, wodurch es denkbar ist, die M2M-Anwendung für unterschiedlichste Szenarien einzusetzen.

Für die Implementierung des Servers wurde entschieden eine Java-basierte Webanwendung zu verwenden, die zugleich RESTful ist. Als Grundlage hierfür wurde das Play Framework eingesetzt, welches sich rückblickend als gute Wahl erwiesen hat. Durch die Architektur der Webanwendung konnte das Empfangen der Informationen, die durch die Sensorstationen versendet werden, einfach und effizient gelöst werden. Zudem hat der große Funktionsumfang des Frameworks dafür gesorgt, dass viele Anforderungen ohne großen Aufwand umsetzbar waren.

Im Ganzen betrachtet zeigt die hier in dieser Arbeit entworfene M2M-Anwendung samt deren Prototyp viel Potenzial auf. Ich bin überzeugt, dass zusammen mit den vorgeschlagenen Erweiterungen diese Anwendung ein konkurrenzfähiges Produkt im M2M-Markt darstellen könnte.

[70] Falls ein Modeswitch-Vorgang notwendig ist kann sich die Zeit noch etwas erhöhen.

[71] In Kapitel 8.1 ist eine Formel für die Hochrechnung der Batterielaufzeit angegeben.

Literaturverzeichnis

[3GPP TS 27.007 2010] 3GPP TS 27.007 ; MONRAD, Atle (Hrsg.): *AT Command Set for User Equipment (UE)*. 3GPP/TSG CT WG1, Release 10.1.0. September 2010. – URL http://www.3gpp.org/ftp/Specs/html-info/27007.htm

[Al-Ali u. a. 2010] AL-ALI, A. R. ; ZUALKERNAN, I. ; ALOUL, F.: A Mobile GPRS-Sensors Array for Air Pollution Monitoring. In: *IEEE SENSORS JOURNAL* 10 (2010), Nr. 10, S. 1666–1671

[ARM 2010] ARM: *Cortex-M3 Devices Generic User Guide*, 2010. – URL http://infocenter.arm.com/help/topic/com.arm.doc.dui0552a/DUI0552A_cortex_m3_dgug.pdf

[Bitkom 2011] BITKOM: *Zahl der Handy-Surfer in einem Jahr verdoppelt*. Presseinformation. 21. März 2011. – URL http://www.bitkom.org/files/documents/BITKOM_Presseinfo_Zugangsgeraete_ins_Web_21_03_2011.pdf

[Böning u. a. 2010] BÖNING, Steffen ; DÄMBKES, Andreas ; REICHHART, Philipp: Die M2M-Industry-Map Deutschland / E-Plus Gruppe. URL http://www.eplus-gruppe.de/download/filedownload.asp?folder=presse_studie&file=M2M_Industry_Map.pdf, 2010. – Forschungsbericht

[Braden 1989] BRADEN, R.: Requirements for Internet Hosts - Communication Layers RFC 1122. 1989. – Forschungsbericht

[Clark 2003] CLARK, Martin P.: *Data Networks, IP and the Internet: Protocols, Design and Operation*. 1. Wiley, 3 2003. – ISBN 9780470848562

[Dunkels 2003] DUNKELS, Adam: TCP/IP for 8-Bit Architectures. In: *In Proceedings of The First International Conference on Mobile Systems, Applications, and Services (MOBISYS 03)*, 2003

[Fielding u. a. 1999] FIELDING, R. ; GETTYS, J. ; MOGUL, J. ; FRYSTYK, H. ; MASINTER, L. ; LEACH, P. ; BERNERS-LEE, T. ; (IETF), Internet Engineering Task F. (Hrsg.): *Hypertext Transfer Protocol – HTTP/1.1 (RFC 2616)*. 1999. – URL http://www.ietf.org/rfc/rfc2616.txt

[Glanz und Jung 2010] GLANZ, Axel ; JUNG, Oliver: *Machine-to-Machine-Kommunikation*. 1. Campus Verlag, 3 2010. – ISBN 9783593392240

[Li u. a. 2006] LI, Xiuhong ; SUN, Zhongfu ; HUANG, Tianshu ; DU, Keming ; WANG, Qian ; WANG, Yingchun: Embedded Wireless Network Control System: an Application of Remote Monitoring System for Greenhouse Environment. In: *Proc. IMACS Multiconference Computational Engineering in Systems Applications* Bd. 2, 2006, S. 1719–1722

[Melzer 2010] MELZER, Ingo ; MELZER, Ingo (Hrsg.): *Service-orientierte Architekturen mit Web Services: Konzepte - Standards - Praxis*. 4. Aufl. Spektrum Akademischer Verlag, 4 2010. – ISBN 9783827425492

[Telit Wireless Solutions] TELIT WIRELESS SOLUTIONS: *Audi setzt bei neuem Infotainmentsystem auf UMTS/HSDPA-Technologie von Telit Wireless Solutions*. Pressemitteilung vom 05.05.2010. – URL `http://www.ptext.net/pressemitteilung/audi-infotainmentsystem-umtshsdpa-technologie-telit-wireless-solutions-7012`

[Texas Instruments 2010a] TEXAS INSTRUMENTS: *Stellaris LM3S9B96 Microcontroller DATA SHEET*, December 2010

[Texas Instruments 2010b] TEXAS INSTRUMENTS: *Using AES Encryption and Decryption with Stellaris Microcontrollers*. AN01251-03. Texas Instruments Incorporated (Veranst.), 01 2010

[Texas Instruments 2011] TEXAS INSTRUMENTS: *Stellaris USB Library*. SW-USBL-UG-6852. Texas Instruments Incorporated (Veranst.), 01 2011

[USB IF 2000] USB IF: *Universal Serial Bus Revision 2.0 specification*. 2.0. Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, and Philips (Veranst.), 2000. – URL `http://www.usb.org/developers/docs/`

[Wikipedia 2011] WIKIPEDIA: *List of real-time operating systems — Wikipedia, The Free Encyclopedia*. 2011. – URL `https://secure.wikimedia.org/wikipedia/en/w/index.php?title=List_of_real-time_operating_systems`

[Yiu 2009] YIU, Joseph: *The Definitive Guide to the ARM Cortex-M3, Second Edition*. 2. Newnes, 12 2009. – ISBN 9781856179638

Anmerkung: Alle Links wurden am 24.7.2011 erneut abgerufen und überprüft.

Abbildungsverzeichnis

Glossar & Abkürzungsverzeichnis

APN Access Point Name

CHAP Challenge Handshake Authentication Protocol

CRUD Create, Read, Update and Delete

FSM Finite-state machine (Endlicher Automat)

GCC GNU Compiler Collection

IPCP IP Control Protocol

lwIP lightweight IP

M2M Machine-to-Machine

MTU Maximum Transmission Unit

NVIC Nested Vectored Interrupt Controller

RS-232 Eine serielle Schnittstelle

RTOS Real-time operating system (Echtzeitbetriebssystem)

RTT Round Trip Time

Sensorstation Gewählte Bezeichnung, welche den Data-End-Point / Client beschreibt. Diese übertragen beispielsweise Sensormesswerte via UMTS an den Server.

UMTS Universal Mobile Telecommunications System. In dieser Arbeit wird der Begriff umgangssprachlich angesehen und schließt somit dessen Erweiterungen, wie zum Beispiel HSUPA, mit ein.

UMTS Stick Ein USB-Modem, welches eine UMTS-Internetverbindung herstellen kann.

ISP Internet-Service-Provider

PPP Point-to-Point Protocol

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw.§24(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 28.07.2011 Patrick Büsselmann