Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Marco Schneider

Entwicklung und Realisierung eines Sensornetzwerkes für das Living Place Hamburg

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Marco Schneider

Entwicklung und Realisierung eines Sensornetzwerkes für das Living Place Hamburg

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Gunter Klemke
Zweitgutachter : Prof. Dr. rer. nat. Kai von Luck

Abgegeben am 18.08.2010

Marco Schneider

Thema der Bachelorarbeit

Entwicklung und Realisierung eines Sensornetzwerkes für das Living Place Hamburg

Stichworte

Sensornetzwerk, ZigBee, verteilte Systeme, Smart Dust, Ubiquitous Computing, Funknetzwerke, Arduino, Living Place Hamburg

Kurzzusammenfassung

Diese Arbeit beinhaltet eine mögliche Realisierung eines Sensornetzwerkes im Kontext des Living Place Hamburg. Es sollen Sensor-Informationen einfach per Funk an das Rückgrat der Wohnung übertragen werden. Es wird dabei ausschließlich mit Arduino-Standardhardware gearbeitet, sodass eine einfache Erweiterbarkeit gewährleistet ist.

Die Implementierung umfasst sämtliche Schritte vom eigentlichen Sensor bis hin zur ActiveMQ-Schnittstelle der Wohnung.

Marco Schneider

Title of the paper

Development and implementation of a WSN for the Living Place Hamburg project

Keywords

sensor networks, ZigBee, distributed systems, smart dust, ubiquitous computing, wireless networks, Arduino, Living Place Hamburg

Abstract

This paper discusses the possible implementation of a sensor network for the "Living Place Hamburg" project. Sensory information is to be transmitted via radio to the technological backbone of the apartment. Only Arduino standard hardware is used to ensure extensibility.

The implementation of the network contains all steps ranging from the sensor itself to the ActiveMQ interface of the apartment.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

1.1 Motivation

Heutzutage sind Computer nicht mehr aus dem Leben wegzudenken: nichts funktioniert mehr ohne sie. Durch die rasante Entwicklung der letzten Jahrzehnte wurden sie immer kleiner und leistungsfähiger. Sie sind fast überall - aber nur der kleinste Teil steht sichtbar unter dem Schreibtisch. Wo sind also all die unsichtbaren Computer, wer bedient sie und wofür sind sie überhaupt?

Die Antwort steckt in unscheinbaren Alltagsgegenständen wie zum Beispiel dem Auto. Die Klimaanlage steht auf 21°, der Tempomat ist aktiv. Lieschen Müller hat gerade Feierabend gemacht und ist auf dem Heimweg. Plötzlich läuft ein Tier vor das Auto - Vollbremsung. Lieschen Müller spürt das Pulsen des ABS im Fuß. Gerade nochmal gutgegangen. Dies lag auch an der Vielzahl der Computer, die in diesem Moment unterstützend tätig waren.

Eigentlich mag Lieschen Müller keine Computer: „nichts als Ärger mit den Dingern!“ sagt sie immer wieder. Auch mit Technik hat sie es nicht so, denn es sei alles viel zu kompliziert. Lieschen gehört zu den Menschen, die Technik benutzen, die nicht als lästig oder störend empfunden wird. Diese Eigenschaft macht Lieschen zur idealen Testperson für den Living Place in Hamburg.

Der Living Place Hamburg ist eine ganz normale Wohnung - zumindest für Lieschen Müller. Für Informatiker ist sie ein Mekka: sie ist ein Versuch, Marc Weiser’s Vision vom Ubiquitous Computing (allgegenwärtigen Rechnen) zu realisieren. Seine Vision war, dass die Interaktion zwischen Mensch und Computer nicht nur zwischen Mensch und Standard-PC möglich ist, sondern dass sie durch intelligente Gegenstände in den normalen Alltag integriert wird.

Um diese Vision zu erreichen, werden große Mengen von Daten benötigt. Die Wohnung kann vielerlei Unterstützungen bieten, dies fängt bei der automatischen Lichtsteuerung an, welche Nachts das Licht langsam hochdimmt. Es geht über einfache Sachverhalte wie das automatische Anschalten der Kaffemaschine basiert auf einem Durchschnittswert der normalen

Duschdauer des Benutzers bis hin zu komplexen Zusammenhängen wie der Stimmungserkennung und -widerspiegelung durch Lichtfarbtöne. All diese Steuerungen benötigen zur korrekten Ausführung viele Daten. Doch woher kommen diese Daten?

Sensoren aller Art sind die „Wahrnehmungsintstrumente“ des Computers. Es gibt sie vom einfachen Türkontaktschalter über Bewegungssensoren bis hin zu hochauflösenden Kameras. Die Wohnung verfügt über eine Vielzahl der unterschiedlichsten Sensoren um möglichst genaue und viele Informationen zu erhalten. Sensoren müssen irgendwie eingelesen werden, traditionell geschieht dies per Kabel. Doch es gibt auch Stellen und Orte, an denen es einfach nicht möglich ist, ein Kabel zu legen: z.B. in beweglichen Objekten wie Stühlen. Hier ist eine Funktechnologie gefragt.

Im Rahmen dieser Bachelorarbeit soll ein solches Funknetzwerk aus Sensorknoten geplant und aufgebaut werden.

1.2 Gliederung

Die Arbeit ist gegliedert in 4 verschiedene Bereiche.

Zu Beginn wird im Kapitel Grundlagen auf die für diese Arbeit notwendigen Grundlagen und Hintergrundinformationen eingegangen. Hier werden neben der Hardware auch die Grundlagen eines Sensornetzes erklärt; Es wird ebenfalls auf das Software-Rückgrat der Wohnung - das Message Broker System ActiveMQ eingegangen.

Die Analyse bildet mit dem Design den Kern dieser Arbeit. Es wird zuerst auf die notwendigen Anforderungen eingegangen, welche danach ausführlich diskutiert und erörtert werden. Es reicht von der Auswahl der benötigten Hardware über ein geeignetes Funkprotokoll bis hin zur Datenaufbereitung. Es werden weiter interne und externe Schnittstellen spezifiziert, um eine Erweiterbarkeit zu gewährleisten.

Der folgende Bereich ist der Realisierung gewidmet. In diesem Kapitel werden die konkreten Umsetzungen des Designs erläutert. Es werden Tests der Software durchgeführt und dokumentiert und es wird auf Probleme während der Realisierung eingegangen.

Abschließend bildet das Resümee mit einer kurzen Zusammenfassung den Schlussteil dieser Arbeit. Es wird ein Fazit gezogen und es werden mögliche Verwendungsszenarien und Verbesserungsmöglichkeiten im Ausblick vorgestellt.

2 Grundlagen

In diesem Kapitel soll erklärt werden, welche Hardware in dieser Bachelorarbeit verwendet wurde. Es ist nicht wichtig, sämtliche Details der einzelnen Komponenten zu verstehen oder zu erörtern; Vielmehr ist ein Überblick der Module und deren Funktionen das Ziel dieses Kapitels.

Es wird auch kurz auf die „Standardhardware“ eingegangen - da Arduino’s kaum Grenzen an die eigene Phantasie vorgeben, würden die Kombinations- und Einsatzmöglichkeiten den Rahmen dieser Arbeit sprengen.

2.1 Living Place Hamburg

Der „Living Place Hamburg“ ist ein ca. 140qm großes, vollausgestattetes Apartment im Loft-Stil, in welchem Experimente unter Realbedingungen ermöglicht werden. Es existiert seit Anfang 2009 und wurde in dem alten Maschinenbaugebäude in den ehemaligen AStA-Räumen der Hochschule für Angewandte Wissenschaften Hamburg untergebracht.

Die Wohnung wurde bis vor Kurzem komplett entkernt und renoviert. Da bei diesem „Umbau“ auf nichts Rücksicht genommen werden musste, konnte extrem viel Technik und Kabel in der Wohnung untergebracht werden. Dies ist erforderlich, um einen fließenden Übergang von dem tangible (anfassbaren) zum ubiquitous (allgegenwärtigen) computing zu gewährleisten.

Ziel ist es, die schon 1991 von Weiser aufgestellte Vision des allgegenwärtigen Computing zu realisieren. Dabei soll aber entgegen der früheren (an der komplizierten Interaktion gescheiterten) Konzepte des intelligenten Hauses besonderer Schwerpunkt auf der Seamless Interaction sowie der Human Computer Interaction liegen (vgl. [Gregor u. a., 2009]).

Insbesondere soll die Wohnung dazu dienen, in realitätsnahen Experimenten im Zeitfenster von Stunden bis hin zu mehreren Tagen das tangible Computing zu erforschen und neue Technologien einschätzen und bewerten zu können. Dabei wurde großer Wert darauf gelegt, dass die Wohnung nicht das Image eines „Laborkäfigs“ bekommt - ganz im Sinne des ubiquitous computing: *Computer sind überall, aber man sieht sie nicht*.

Da das Projekt gerade erst gestartet ist, stecken die Wünsche und Ideen noch in den Kinderschuhen und werden gerade von ca. 20 Master-Studenten der HAW-Hamburg entwickelt. Dabei geht es um Themen wie ein „intelligentes Bett“, das z.B. Leichtschlafphasen erkennen soll um diese dem „Wecker 2.0“ mitzuteilen, wann ein geeigneter Zeitpunkt wäre, den Benutzer zu wecken. Es werden aber auch z.B. Stimmungen erkannt und entsprechend in RGB-Lichtsteuerungen umgesetzt. Wem auch dies nicht reicht, der findet vielleicht Gefallen an der Idee, dass per Laser oder Beamer z.B. auf den Küchentisch ein Auswahlmenü projiziert wird, sobald es an der Tür klingelt. Hier kann sich der Benutzer entscheiden, ob der Türöffner aktiviert werden soll oder nicht. Man sieht, dass nur die eigene Phantasie die Möglichkeiten in dieser Wohnung zu begrenzen scheint.

Die Wohnung lebt von Informationen, die eine Vielzahl von Sensoren an ein Blackboard (ActiveMQ - siehe Kommunikationsschnittstelle ActiveMQ) liefern. Da es nicht überall möglich ist, Sensoren per Kabel mit einem Server kommunizieren zu lassen, lag die Idee nahe, dass man einen einfachen Funksensor gebrauchen könnte. Dieser Sensor soll idealerweise klein, stromsparend und bidirektional sein: es sollen Sensorwerte eingelesen werden können, aber es soll auch (wenn sowieso schon ein Mikrocontroller vorhanden ist) über die Ferne etwas gesteuert werden können. Durch diesen Ansatz wäre es nicht nur möglich, in beweglichen Gegenständen wie Stühlen oder Kaffemaschinen Sensoren zu platzieren, sondern entfällt auch bei einem nachträglichen Einbau eines Sensors die Verkabelung. Es muss also ein Funknetzwerk realisiert werden, das die gewünschten Eigenschaften bereitstellen kann.

2.2 Sensornetzwerke

[Eriksson, 2009] beschreibt ein Sensornetzwerk als eine Ansammlung kleiner, autonomer Knoten. Dabei hat jeder Knoten einen kleinen Mikroprozessor, einen Funkchip, ein paar Sensoren und normalerweise eine Batterie, von der die Lebenszeit des Knotens abhängig ist.

Andere Quellen (z.B. [Haenselmann, 2006]) legen keinen Wert darauf, dass ein Sensornetzwerk zwangsläufig drahtlos sein muss: der Aspekt der Kommunikationswege und der Strom-

versorgung sollte außer acht gelassen werden. Fakt ist, dass es wie bei einem verteilten System keine Definition, sondern nur Annäherungen an den Begriff gibt.

Alle Quellen sind sich einig, dass es sich bei einem Sensornetzwerk um kleine Controller handelt, die mit Sensoren ausgestattet sind. Diese Sensorknoten (auch: *Motes*) bezeichnen Mikroprozessoren, die in großen Mengen verfügbar sind. Damit einhergehend müssen sie klein, stromsparend und dem Einsatzgebiet angepasst sein. *Einwegprodukt* bzw. *Wegwerfartikel* wären treffende Bezeichnungen - jedoch kommt es hier auch wieder stark auf das Einsatzgebiet an.

Die einzelnen Knoten bauen eigenständig das Sensornetzwerk auf. Typischerweise gibt es einen Coordinator, in jedem Fall aber sog. AccessPoints - also Schnittstellen vom Sensornetzwerk zum Betreiber. Über diese teilt das Sensornetz Ereignisse mit - somit kann der Betreiber die Informationen auswerten.

Je nach Anwendungsgebiet können bis zu mehrere tausend Knoten ein Sensornetz bilden - dies kommt z.B. beim Militär zur Anwendung, um ein großes Gebiet zu überwachen. Diese Vielzahl kleiner Knoten wird auch als *SmartDust* (=intelligenter Sensorstaub) bezeichnet.

Der Vorgänger der heutigen Sensornetzforschung war das Sound Surveillance System *(SOSUS)*, welches Mitte der 50er Jahre im kalten Krieg von Amerika eingesetzt wurde, um U-Boote aufzuspüren. Dazu wurden diverse Bojen mit Schallsensoren ausgerüstet und einzeln ausgewertet. Dies war kein Rechnernetz im heutigen Sinne, brachte aber die Idee der flächendeckenden Sensoranordnung hervor.

2.2.1 Sensorknoten

Sensorknoten bestehen wie oben schon angedeutet aus einem Mikroprozessor, einem Sensormodul und aus einer Netzwerkschnittstelle (meist drahtlos). Grundsätzlich werden diese Sensorknoten autark betrieben, d.h. sie besitzen eine Batterie und somit spielt der Faktor Energieverbrauch eine große Rolle bei der Auswahl der Hardware und bei der Programmierung der Software. Durch den Zusammenschluss mehrerer Sensorknoten zu einem Sensornetzwerk, bildet sich ein verteiltes System (vgl. [Tanenbaum und van Steen, 2006]), welches die Kontexte aller beteiligten Knoten zur Verfügung stellt. Die Knoten sollen im Idealfall konfigurationsfrei - oder zumindest konfigurationsarm sein. Das heißt, wenn neue Knoten in das Netzwerk kommen, dass diese einfach integrierbar sein sollen. Die Integration in das Netzwerk an sich ist recht simpel: Der Knoten meldet sich bei dem nächsten Nachbarn und registriert sich bei diesem. Damit ist er Teil des Sensornetzwerkes. Tritt nun ein Ereignis auf,

welches der Knoten wahrnimmt, propagiert er dieses Ereignis über das Sensornetzwerk. Eine bestimmte Zieladresse für dieses Ereignis ist nicht notwendig, da die Knoten das Ereignis per Broadcast verbreiten. Somit wird sichergestellt, dass das gesamte Netz die Information erlangt hat.

Die Kosten pro Stück sind wieder ganz vom Einsatzzweck abhängig. So spricht die DARPA (US Verteidigungsministerium) z.B. in Größenordnungen von hunderttausenden Sensorknoten. Im Fahrzeugbau sind Stückpreise von 2 EUR sicherlich noch tragbar, bei der Anwendung der DARPA z.B. als Grenzkontrolle lohnt sich dies erst im unseren Cent-Bereich. Die teuersten Sensorknoten heutzutage kosten selten über 100 EUR pro Stück und sind dann mit modernster Überwachungstechnik z.B. für das Militär ausgestattet.

Einhergehend mit dem Preisverfall der Sensorknoten gibt es das Problem der Umweltverschmutzung: Selbstverständlich ist es einfach, in einem Waldgebiet tausende von Sensoren zu verteilen, jedoch ist irgendwann auch die stärkste Batterie an ihrer Leistungsgrenze angekommen. Daher versucht man die Sensorknoten künftig ökologisch verträglicher zu machen, sodass sie nach ihrem Einsatz sprichwörtlich „verrotten“. Leider ist dies noch komplette Zukunftsmusik und es wird noch Jahrzehnte dauern, bis diese Idee umgesetzt wird.

2.2.2 Kommunikation

Grundsätzlich gibt es 2 verschiedene Ansätze in verteilten Netzen: es gibt dezentrale Netztopologien wie Ad-hoc-Netz (Mesh-Network), aber auch zentrale Ansätze wie die Sterntopologie. Normalerweise bilden Sensornetze immer Ad-hoc-Netze, d.h. Netzwerke ohne feste Infrastruktur zwischen den einzelnen Sensorknoten. Die einzelnen Sensorknoten sind nur mit den Nachbarn verbunden und kommunizieren über Broadcasts. Daraus ergibt sich eine sog. Multi-Hop-Kommunikation, bei der die Nachrichten von Knoten zu Knoten weitergegeben werden, bis das Ziel erreicht ist.

Durch diese Eigenschaften nennt man die Netze *unsicher*, da durch die dynamische Konfiguration weder Struktur, Antwortzeit noch „Leitungsqualität“ vorhersagbar sind - hinzu kommt, dass bei der Vielzahl der Knoten auch mit Ausfällen gerechnet werden muss (diese statistischen Ausfälle lassen sich berechnen und werden so zu einem handhabbaren Problem).

Sensornetzwerke verfügen über spezielle Netzprotokolle, da sich gezeigt hat, dass sich herkömmliche Netzwerkprotkolle wie der 802.11-Standard einfach viel zu verschwenderisch mit der Energie sind - oder einfach nur begrenzt viele Teilnehmer in einem Netz beherbergen

können, wie es bei Bluetooth der Fall ist. Auch gibt es in den Sensornetzwerken keine Konkurrenz um Ressourcen - das Sensornetz hat nur eine einzige, netzweite Anwendung, die sogesehen mit sich selber nicht konkurriert. Das Gesamtziel der Anwendung hat oberste Priorität - nicht die Gleichbehandlung einzelner Knoten. Der Begriff der Fairness in Rechnernetzen muss also umdefiniert werden (vgl. [Tanenbaum, 2003]).

Interessanterweise lassen sich die Routingprotokolle aus dem normalen Netzwerkleben ohne bzw. mit minimalen Anpassungen auf ein Sensornetzwerk adaptieren. Besonders relevant für Sensornetze sind jedoch geographische Routingprotokolle: viele Anwendungen benötigen Informationen von einem Gebiet. Zum Beispiel könnten diese Informationen für einen Mehrheitsentscheid benutzt werden, um auszuschließen, dass ein Sensorknoten falsche Werte liefert; das Routingprotokoll muss dem Benutzer aber in jedem Fall die Auswahl der Knoten abnehmen. Der Benutzer weiß nicht, welche Knoten einem Bestimmten am nächsten sind - das weiß nur der Routingalgorithmus. Ein Verfahren heißt Geo-Cast - dieses visualisiert das Sensornetzwerk z.B. auf eine Landkarte. So ist es möglich, einen Sensor mittels GUI schnell zu finden.

2.2.3 Ortsbestimmung

Es ist möglich, die Position relativ zueinander in einem Sensornetzwerk bestimmen zu können, sofern 2 Sensorknoten ihren absoluten Standort in dem Netzwerk kennen. Dies funktioniert so, dass jeder Sensorknoten sein Koordinatensystem aufstellt - dabei muss er im Ursprung sein und zwei benachbarte Knoten als X und Y-Achse festlegen. Durch Verfahren wie Triangulation können nun weitere, benachbarte Knoten nach selbem Verfahren ihre Position zu dem Punkt bestimmen und sich so in dem Koordinatensystem einordnen. Leider garantiert dies keinen Erfolg, denn wenn zu wenig Knoten vorhanden sind oder sie ungünstig angeordnet sind, so bleibt die Ortsbestimmung auch ungenau.

2.2.4 Synchronisation

Oftmals ist es wichtig in Sensornetzen, dass eine genaue Uhrzeit bei jedem Knoten vorhanden ist um z.B. transitive Kausalketten von Ereignissen zu erstellen. Da jeder Sensorknoten einen eigenen Quarz hat, der unterschiedlich schnell läuft, muss man irgendwie die Knoten synchronisieren. Dabei ist es unerheblich die Knoten auf die Uhrzeit zu synchronisieren - Hauptsache sie laufen irgendwie in ihrem System synchron. Zu diesem Standardproblem von verteilten Systemen gibt es mehrere Ansätze um das Problem zu lösen, einer davon ist

die Messung der Round Trip Time. Dabei zieht man die Zeitstempel von 2 Knoten im Netz ab, zieht noch einmal die Round Trip Time ab und erhält so die Differenz der beiden Zeitstempel. Diesen Wert kann zurückschicken und somit wissen beide Knoten, welche Differenz es gibt. Dieses Verfahren hat immer das Problem der unterschiedlichen Laufzeiten und wird so nie wirklich genau.

Ein weiterer Ansatz ist die Reference Broadcast Synchronisation (RBS). Bei diesem System sendet eine zentrale Stelle im Netzwerk einen Zeitstempel aus. Dieser wird von jedem Knoten empfangen und der Knoten schickt dann an seine Nachbarn diesen Zeitstempel raus. Der Nachbar vergleicht den eigenen Stempel mit dem vom Nachbarn und stellt dann eine mögliche Abweichung fest. Der Unterschied zu den anderen Algorithmen ist, dass jetzt die Uhren nicht zwingend umgestellt werden müssen, sondern die Knoten mit dieser Verzögerung weiterrechnen können. Natürlich können die Knoten auch die Zeit untereinander anpassen - dies hat den Vorteil, dass bei starken Varianzen der Laufzeit auch eine genaue Uhrzeit im Mittel erreicht werden kann (vgl. [Tanenbaum und van Steen, 2003]).

Ein hierarchischer Ansatz ist das Timing-Sync Protocol for Sensor Networks (TPSN). Hier ist der Initiator der Wurzelknoten, welcher sich die Priorität 0 gibt. Dieser sendet nun einen Broadcast an alle Nachbarn mit einer *level_discovery-Nachricht*. Die Knoten, die diese Nachricht empfangen, setzen die eigene Priorität um 1 höher als in der empfangenen Nachricht und senden auch einen Broadcast. Wenn ein Knoten nun eine Nachricht erhält, die eine kleinere Priorität enhält als die eigene, wird eine zufällige Zeit gewartet und dann wird erst der Broadcast an die eigenen Nachbarn weitergeschickt.

Die Synchronisation funktioniert nun so, dass der Wurzelknoten die *time-sync-Nachricht* an die Nachbarn schickt, und sie so auffordert, ihn nach der Uhrzeit zu fragen. Die Knoten senden nun ein *synchronization-pulse*, das vom Vaterknoten die aktuelle Zeit (mit einem ACK) zurückgibt. Alle anderen Knoten der selben Priorität veranlasst diese Nachricht, ebenfalls beim Vaterknoten anzufragen. So wird hierarchisch die Zeit synchronisiert (vgl. [Kulakli und Erciyes, 2008]).

2.2.5 Netztopologien

In Abbildung 2.1 sehen wir die nach IEEE 802.15.4 möglichen Netztopologien. Es gibt grundlegend drei verschiedene Arten, die Stern (Star)-, Punkt-zu-Punkt (Mesh)- und die Baumtopologie (Cluster-Tree).

Eine Sterntopologie ist denkbar einfach aufgebaut: der Koordinator in der Mitte hat eine direkte Verbindung zu jedem Knoten, Routing zwischen den Knoten ist nur über den Koordinator möglich. Dies empfiehlt sich, wenn nur Daten von den Knoten zum Coordinator übermittelt werden müssen, aber nicht untereinander. Ein Nachteil dieses Konzepts ist der Koordinator. Im Falle eines Fehlers funktioniert das gesamte Netz nicht mehr und ist somit ein single point of failure.

In einer Mesh-Topologie bilden die full function devices (FFD) ein Netzwerk aus Verbindungen, jeder ist mit jedem verbunden. Es gibt natürlich auch wieder einen Koordinator, welcher Teil dieses Netzes ist, jedoch kann sein Standpunkt innerhalb des Netzes nicht vorhergesagt werden, da die Netztopologie sich selber aufbaut. Dies hat den großen Vorteil, dass das Netz „selbstheilend“ ist: Falls ein Knoten ausfällt, erkennt das Netz dies und leitet die Daten automatisch über einen anderen Weg weiter. Das Netz ist somit sehr robust und sehr gut geeignet für Sensornetzwerke.

Die Baumtopologie ist wie ein typischer Informatik-Baum aufgebaut. Die Wurzel bildet der Koordinator, welcher über direkte Verbindungen zu den Routern verfügt. Die Router können wie der Koordinator mehrere Kinder haben, aber jeweils nur einen Elternknoten. Hier stoßen wir auf selbiges Problem wie bei den Stern-Netzen, denn sobald ein Router ausfällt, ist es seinem gesamtem Ast nicht mehr möglich, mit dem Koordinator zu kommunizieren. Der Vorteil besteht darin, dass das Netzwerk sehr genau segmentiert werden kann und so z.B. auf einer Landkarte genau die Standorte und Routingwege der einzelnen Knoten nachvollziehen kann.

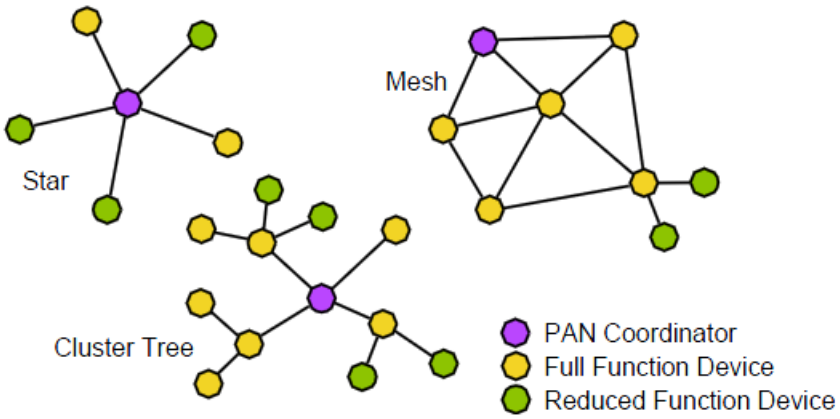

Abbildung 2.1: Netztopologien bei Sensornetzen [Legg, 2004]

2.3 Arduino-Plattform

Die Marke „Arduino“ gehört der Firma tinker.it, welche das Arduino-Projekt weit vorangetrieben hat. Es handelt sich um eine aus Soft- und Hardware bestehende Physical-Computing Plattform. Sie basiert auf einem quelloffenen Standard mit einem Atmel AVR-Mikrocontroller - das heißt, dass jeder diese Plattform nachbauen kann ohne zusätzliche Lizenzgebühren zahlen zu müssen.

Das Ziel der Arduinos war es, einen einfachen Mikrocontroller für Nicht-Informatiker (z.B. Designer) zu entwickeln. Hierbei sollte eine sehr einfache Programmiersprache verwendet werden, die keine großen Programmierkenntnisse erfordert und trotzdem annähernd das gesamte Funktionsspektrum eines Mikrocontrollers repräsentiert. Dieser Kompromiss wurde in der Programmiersprache *Processing* gefunden.

Diese Programmiersprache ist sehr funktional - d.h. es kommt nicht darauf an, in welcher Reihenfolge entsprechende Register beschrieben werden (wie bei C und dem normalen AVR), sondern vielmehr geht es um einfachste Prozeduraufrufe. Als einfaches Beispiel hierfür kann die serielle Schnittstelle dienen: mit einem einfachen *serial.begin(9600)* wird die gesamte serielle Schnittstelle konfiguriert. Bei C-Code und einem normalen AVR ist dies bekanntermaßen nicht so einfach, da dort Einstellungen in x Registern vorgenommen werden müssen, welche man sich aus dem Datenblatt raussuchen muss.

Für die Arduinos werden keine speziellen JTAG- oder ISP-Controller benötigt, wie dies bei üblichen Mikroprozessoren zum Programmieren nötig ist. Dies funktioniert über einen eingebauten ISP (*In-System-Programmer*), welcher als mini-Bootloader auf dem Arduino vorhanden ist. Dieser bootet ein minimales System, welches zulässt, dass wir den Arduino über die USB-Schnittstelle programmieren können.
Trotzdem besitzt der Arduino eine ISP-Schnittstelle, mit dem man „von Hand“ den Bootloader wieder installieren kann, falls dort mal etwas schief gegangen ist. Der Atmel ist also nicht verloren, sondern kann mit etwas Aufwand eigentlich immer wieder zum Leben erweckt werden.

Die Stromversorgung der Arduinos erfolgt auch über die USB-Schnittstelle - oder wahlweise über den externen Stromanschluss. Dieser kann in der Regel von 6 V bis 20 V betrieben werden - dies ist durch einen auf dem Board verbauten Spannungskonstanter möglich. Es ist allerdings nicht ratsam, den Arduino mit einer Spannung über 9 Volt zu betreiben, da der Spannungskonstanter über seinen längsgeregelten Transistor nur die überflüssige Spannung in Wärme umsetzen kann.

Durch das festgelegte Design der einzelnen Arduinos besitzt jeder Mikroprozessor eine vorgegebene Anzahl an Schnittstellen - diese sind nicht nur in Form von Pins irgendwo vorhanden, sondern auch in einer bestimmten Reihenfolge auf dem Board angeordnet. Weiterhin bilden diese ein Stecksystem: es gibt sog. Shields, mit welchen die Arduinos leicht um diverse Module und Fähigkeiten erweitert werden können. Dazu aber später mehr.

Bisher wurde nur von Arduinos gesprochen, nun sollen kurz die verschiedenen Versionen vorgestellt werden. Jede Version hat Vor- und Nachteile, welche sich aus dem Design ergeben.

2.3.1 Die Arduino-Hardware

Mega

Der Arduino Mega (Bild 2.2) ist der größte aller Arduinos. Er basiert auf einem Atmel ATmega1280 und bietet neben den typischen Standardschnittstellen für die Shields noch viele weitere I/O-Ports. Genauer: 54 digitale I/O-Pins, von denen 14 PWM-Signale (Pulsweitenmodulation oder neudeutsch pulse-with-modulation) erzeugen können, sowie weitere 16 Analogpins.

Der Mega verfügt nicht nur über deutlich mehr Ports, sondern auch über mehr Speicher. Insgesamt 128 kB stehen zur Verfügung - das ist deutlich mehr als die anderen Arduinos zur Verfügung stellen. Das hat den Vorteil, dass auf dem Mega sehr komplexe Programme ausgeführt werden können, für die ein normaler Arduino nicht reicht.

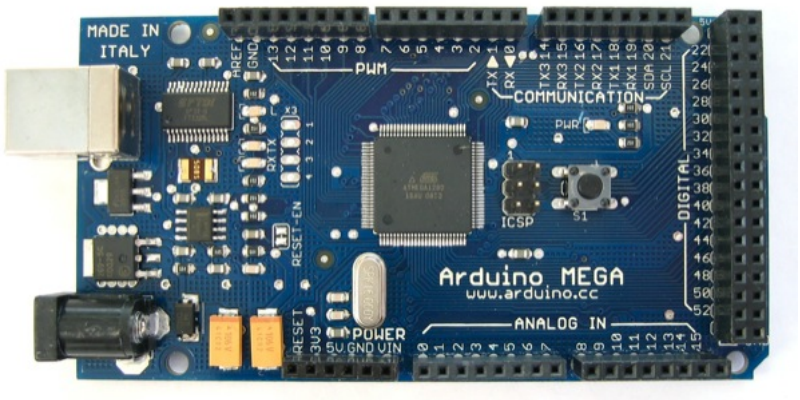

Abbildung 2.2: Arduino Mega[1]

Duemilanove

Der Duemilanove (Bild 2.3, italienisch für *2009*) ist der Standard-Arduino. Wie der Name schon sagt ist es ein Design aus dem Jahre 2009, jedoch gab es ab März 2009 eine kleine Änderung im Design: der ATmega168 wurde gegen einen ATmega328p getauscht. Einer der Unterschiede ist z.B. der doppelt so große Speicher auf dem 328, der Rest ist für diese Arbeit nicht von Bedeutung.

Der Duemilanove besitzt einen 16 MHz Quarz und verfügt über 14 digitale Ein- und Ausgänge. Von diesen 14 I/O-Pins können 6 als PWM-Ausgang benutzt werden. Allerdings ist hier zu beachten, dass es sich um Software-PWM's handelt und nicht um 6 Hardware-Counter, die auf dem Chip vorhanden sind. Weiterhin verfügt der Duemilanove über 6 analoge Input-Pins.

Außerdem besitzt der Duemilanove einen USB-Schutz, der den PC vor einer Überlast schützt: sobald mehr als 500 mA von dem Arduino inkl. der angeschlossenen Hardware verbraucht werden, bricht der Arduino die Verbindung ab, bis das Problem behoben ist.

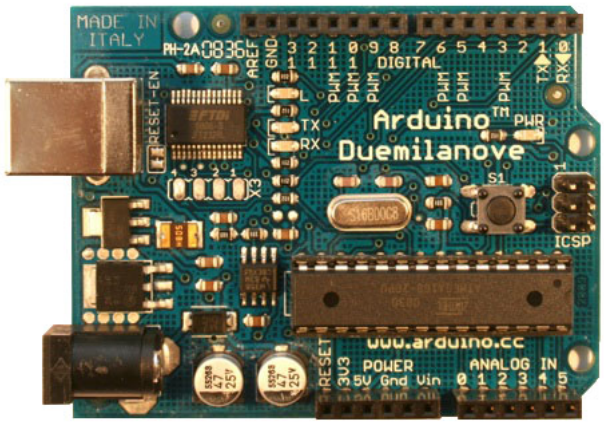

Abbildung 2.3: Arduino Duemilanove[2]

[1] Bild: „Arduino Mega“: http://www.robotronic.co.nz
[2] Bild: „Arduino Duemilanove“: http://www.freeduino.de

Diecimila

Der Diecimila (Bild 2.4, italienisch für *10.000*) ist der Vorgänger des Duemilanove und ist sehr ähnlich aufgebaut, besitzt aber standardmäßig nur den ATmega168 Chip. Der Name wurde dem Board gegeben, um zum Ausdruck zu bringen, dass bereits über 10.000 Stück hergestellt wurden. Von der Programmierung unterscheidet sich der Diecimila nicht von dem Duemilanove, auch verfügt er über die bewährte Pinanordnung für diverse Shields.

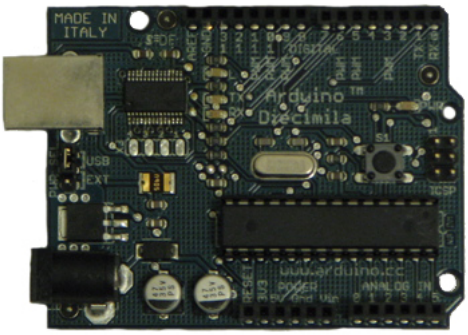

Abbildung 2.4: Arduino Diecimila[3]

Nano

Der Nano (Bild 2.5) ist - wie der Name schon sagt - sehr klein. Er besitzt nur einen Mini-USB Anschluss, hat aber trotz der geringen Größe einen ATmega168 bzw. einen ATmega328 Chip und ist somit von der Rechenleistung her äquivalent zu dem Duemilanove. Auch, wenn man es nicht glauben möchte: der Nano verfügt sogar über die gleiche Anzahl an Pins wie der große Bruder - nur die Anordnung ist nicht „Shield-kompatibel“.

Mini

Der Mini (Bild 2.6) ist der kleinste aller Arduinos - noch kleiner als der Nano. Er besitzt keinen USB-Anschluss und muss deswegen über einen Adapter programmiert werden. Durch die Baugröße hat er keinen Spannungskonstanter und muss mit einer Spannung von 7 - 9 Volt

[3] Bild: „Arduino Diecimila“: http://arduino.cc
[4] Bild: „Arduino Nano“: http://www.coolcomponents.co.uk

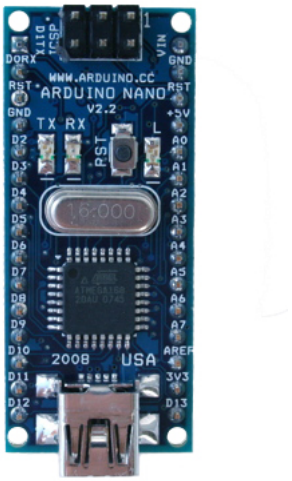

Abbildung 2.5: Arduino Nano[4]

betrieben werden. Sollte die Spannung über 9 Volt liegen, wird der Mini beschädigt. Er besitzt ebenfalls 14 digitale I/O-Pins und einen Speicher von 16 kB. Wie auf dem Bild zu sehen ist, wird er wie alle anderen Arduinos auch über einen 16 MHz-Quarz angetrieben.

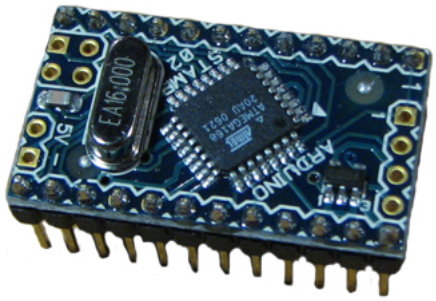

Abbildung 2.6: Arduino Mini[5]

LilyPad

Das Lilypad (Bild 2.7) definiert das Wort „tragbarer Computer“ etwas um: es wurde hauptsächlich für wearable-applications entwickelt. Es klingt erstmal kurios, dass man einen Mikroprozessor in Kleidung einnähen soll - durch das Konzept von den Entwicklern Leah Buechley

[5]Bild: „Arduino Mini“: http://www.nuengineering.com

(MIT) und der Firma Sparkfun gibt es an den Platinen keine Ecken, es kann leicht eingenäht werden und es ist *abwaschbar*.

Das LilyPad besteht aus einer speziellen low-power-Version des ATmega168. Das liegt daran, dass man den relativ kleinen Prozessor zwar gut in der Kleidung verstecken kann, aber nicht mit einem Akkupack auf dem Rücken umherlaufen möchte. Deswegen wird bei diesen Arduino besonders auf das Stromparen geachtet. Der Prozessor läuft auch als einziger nur mit 8 MHz, kann von 2,7 bis 5,5 Volt betrieben werden. Die berühmten 14 digitalen I/O-Pins sind selbstverständlich auch auf diesem Arduino zu finden.

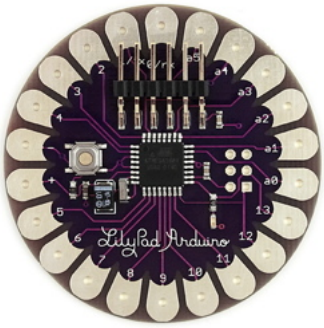

Abbildung 2.7: Lilypad Arduino[6]

2.3.2 Shields

Bisher wurden nur die Arduinos selber vorgestellt. Diese sind vergleichsweise langweilig - man kann sie aber mit sog. Shields bestücken. Dies sind von den Anschlüssen her genormte Aufsteckmodule, die auf den Arduino passen. Mit diesen Shields wird die Funktionalität der Arduinos einfach erweitert und sie werden somit sehr mächtig. Auf arduino.cc befindet sich eine Liste aller Shields. Da es sich aber wie oben gesagt um ein Open-Source Projekt handelt, kommen ständig neue Shields dazu. Aus diesem Grund wird nur kurz auf die Wichtigen eingegangen.

XBee

Das XBee-Shield (Bild 2.8) ermöglicht es dem Arduino über eine serielle Schnittstelle mit anderen Arduinos drahtlos zu kommunizieren. Das XBee-Modul von Firma Digi an sich ist

[6] Bild: „Lilypad Arduino“: http://nuengineering.com

relativ klein und wird mittels des Shield's auf den Duemilanove gesteckt. Dies ist allerdings nicht zwingend erforderlich, da das Modul quasi nur seine Versorgungsspannung und 2 Pins für die serielle Schnittstelle braucht. Einen großen Nachteil hat das Shield leider: sporadisch schickt das Shield auf den Reset-Pin des Duemilanove ein Signal, sodass dieser damit neustartet. Deswegen hat sich gezeigt, dass das Shield eher unpraktisch ist und bei kritischen Anwendungen darauf verzichten sollte.

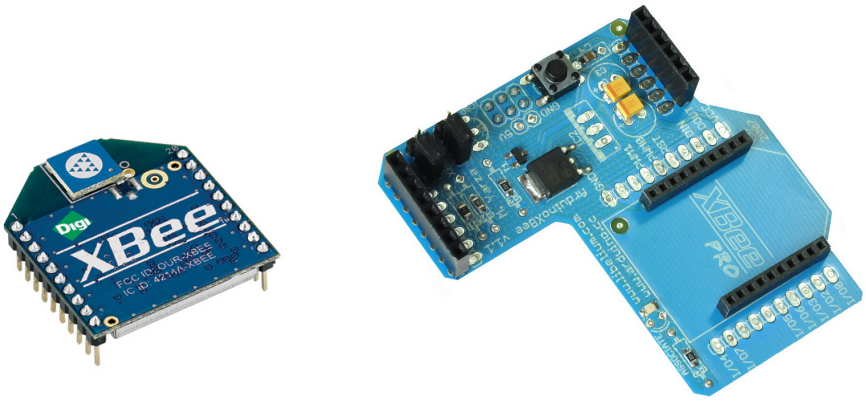

Abbildung 2.8: XBee Modul[7] und Shield[8]

Ethernet

Das Ethernet-Shield (Bild 2.9) verwandelt den Arduino in ein vollwertigen Ethernet-Netzwerkteilnehmer. Es ist ohne weiteres Möglich, kleine Webserver zu implementieren und so Informationen im Netzwerk bereit zu stellen. Es ist ein Wiznet W5100-Chip verbaut, welcher neben dem IP-Stack auch TCP und UDP bereitstellt. Es sind maximal 4 gleichzeitige Verbindungen möglich.

Der Arduino kommuniziert über die Pins 10 bis 13 mit dem W5100-Chip, sodass diese Ports für weitere I/O-Anwendungen nicht mehr verfügbar sind.

[7] Bild: „XBee Modul“: http://www.electronics-lab.com
[8] Bild: „XBee Shield“: http://store.gravitech.us/
[9] Bild: „Ethernet Shield“: http://www.coolcomponents.co.uk

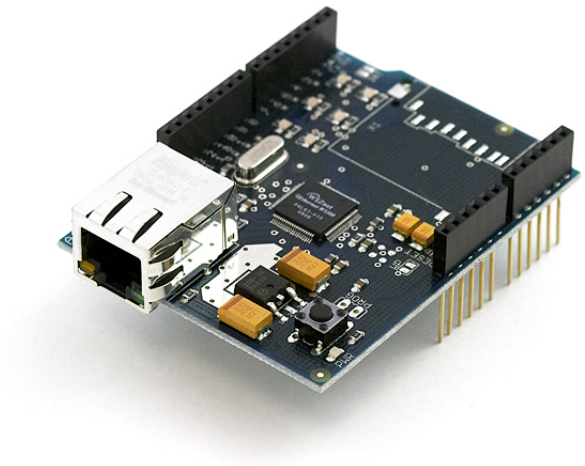

Abbildung 2.9: Ethernet Shield[9]

Bluetooth

Selbstverständlich ist der Arduino auch zu Bluetooth kompatibel (Bild 2.10). Das Modul wird auf dem selben Shield wie die XBee-Module verwendet oder kann (genau wie die XBee-Module auch) direkt angeschlossen werden. Es kann Bluetooth 2.0, ist natürlich auch zu 1.2 abwärtskompatibel. Es hat eine Reichweite von ca. 40 Metern in Gebäuden und 60 Metern außerhalb in direkter Sichtlinie. Der Bluetoothstandard ermöglicht, dass nicht nur Arduino-Arduino-Verbindungen möglich sind, sondern dass man auch den Arduino mit dem PC verbinden kann - oder HID-Geräte wie eine Tastatur.

GPS

Selbst ein GPS-Empfänger (Bild 2.11) ist für die Arduinos erhältlich. Es ist zwar nur ein 9-Kanal Empfänger, das reicht für die meisten Anforderungen jedoch aus. Da das Shield auch eine Log-Funktion besitzt, kann es z.B. beim Sport oder Geocachen mitgenommen werden und später in Google-Earth die Route zu veranschaulichen. Mit einer normalen 9

[10] Bild: „Bluetooth-Modul“: http://www.libelium.com/

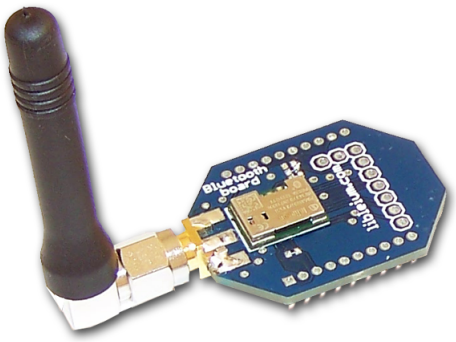

Abbildung 2.10: Bluetooth-Modul[10]

Volt-Batterie hält das Shield ca. 3 Stunden. Das klingt wenig, jedoch zeichnet das Shield jede Änderung auf und verfügt später über eine sehr genaue Route. Auch soll nicht unerwähnt bleiben, dass die 3 Stunden verlängert werden können, wenn Powersave-Maßnahmen ergriffen werden.

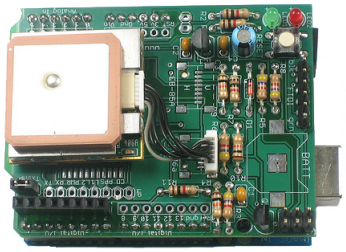

Abbildung 2.11: GPS-Modul[11]

2.3.3 weitere Hardware

Neben den genormten Arduinos und den Shields gibt es noch ein paar weitere, sinnvolle Zusatzteile für das „Projekt Arduino“. Es handelt sich dabei um offiziell produzierte Hardware, also nichts selber Gebasteltes.

[11] Bild: „GPS-Modul“: http://www.navigadget.com/

XBee Explorer USB

Mit diesem kleinen Stück Technik (Bild 2.12) kann direkt vom PC aus mit den anderen XBee's kommuniziert werden. Der PC emuliert eine serielle Schnittstelle, über diese können z.B. andere Arduinos gesteuert werden oder es kann einfach nur etwas mitloggt oder debuggt werden.

Es empfiehlt sich, den Koordinator mit diesem Modul zu betreiben, da dann kein Arduino verloren geht und der Koordinator z.B. fürs Flashen over-the-air benutzt werden kann. Dies stellt sich allerdings als nicht praktikabel heraus, da die Arduinos vor einem Flash neugestartet werden müssen - und dies ist nicht über Software realisierbar. Man muss also zu den Arduinos gehen, den Reset-Knopf drücken und dann in kurzer Zeit das Flashen vom Rechner aus iniziieren. In [Fried, 2010] findet man eine Anleitung fürs OTA-Flashen eines Arduinos.

Weiterhin können die XBee-Module mit dem Explorer einfach konfigurieren werden - denn den normalen Arduino muss man für dieses Procedere etwas „hardwaretechnisch umbauen", sodass er die USB-Anforderungen direkt an das XBee-Shield weitergibt und nicht selber versucht zu verarbeiten.

Abbildung 2.12: XBee Explorer USB[12]

2.4 Kommunikationsschnittstelle ActiveMQ

Die Kommunikationsschnittstelle im LivingPlace Hamburg soll dazu dienen, eine Kommunikationsbasis für die verschiendenen Sensoren bereitzustellen. Hierzu war anfangs der iROS EventHeap angedacht. Nachdem einige Tests von den beiden Masterstudenten Otto und

[12]Bild: „XBee Explorer USB": http://www.robotgear.com.au

Voskuhl durchgeführt wurden bezüglich der Zuverlässigkeit und Geschwindigkeit, wurde sich letztendlich gegen diese Lösung entschieden - nicht zuletzt, weil das Projekt seit längerer Zeit nicht mehr weiterentwickelt wird.

Die Kommunikationsschnittstelle besteht nun hauptsächlich aus dem Message Broker System ActiveMQ. Das System bietet sog. Message Queues, welche der Punkt-zu-Punkt Verbindung von Kommunikationspartnern dienen. Weiterhin kann man von dem Publisher/Subscriber Prinzip Gebrauch machen um seine Daten zu verteilen (vgl. [Otto und Voskuhl, 2010]).

Das ActiveMQ-System wurde von Apache entwickelt und benötigt lediglich einen Apache-Webserver sowie eine Java-Installation. Es gibt vorgefertigte Schnittstellen für diverse Programmiersprachen wie Java, C, C++, C#, Ruby, Perl, Python und PHP (um nur die bekanntesten zu nennen). Diese findet man nebst Beschreibung in [The Apache Software Foundation, 2010].

Es ist möglich, über den ActiveMQ auch objektorientiert zu arbeiten und so Objekte über die Queues serialisiert zu übertragen. Dies klappt allerdings nur per default bei Java-Anwendungen; sollte man z.B. ein serialisiertes Objekt in C empfangen wollen, muss man selber die entsprechende Schnittstelle schreiben. Dies ist allerdings auch keine unüberwindbare Hürde, da alles im gesamten Living Place einheitlich in JSON geschrieben wird.

Der aktuelle Stand im Living Place ist ein ActiveMQ-Server in der Version 5.3.1, welcher auf einem Blade unter der IP-Adresse 192.168.14.2 erreichbar ist. In dem Admin-Bereich des ActiveMQ Servers kann man die Anzahl der aktiven Clients sehen, weiter ist es möglich die einzelnen Topics einzusehen mit Anzahl der empfangenen und weitergeleiteten Nachrichten.

2.4.1 Producer/Consumer über Message Queues

Da das Producer/Consumer-Paradigma in der Informatik hinreichend bekannt ist, wird nur kurz auf diesen Punkt eingegangen. Als erstes verbindet man sich mit dem Server über einen wohldefinierten Port und erstellt danach eine Nachrichtenqueue. In diese Queue kann man nun nach belieben Textnachrichten schicken - um diese Nachrichten wieder lesen zu können, funktioniert das Prinzip andersherum: als erstes wird sich wieder mit dem Server verbunden und die Queue angegeben. Danach kann man mit getMessage() die Nachrichten aus der Queue abholen. Der Nachteil ist hierbei, dass die Nachrichten verbraucht werden und sich diese Topologie somit nur eignet, wenn keine persistenten Informationen gespeichert werden sollen. Es ist möglich, sich mit mehreren Clients an einer Queue anzumelden, jedoch werden

die Nachrichten nicht für alle Clients gespeichert. Das heißt, sobald ein x-beliebiger Client einen getMessage() ausführt, ist das Element aus der Queue verschwunden und die anderen Clients haben keine Chance mehr, die Information zu bekommen.

2.4.2 Publisher/Subscriber über Topics

Auch das Publisher/Subscriber-Paradigma sollte hinreichend bekannt sein. Der Unterschied zu dem Producer/Consumer-Ansatz ist der, dass sich die Publisher und Subscriber über ein Topic auf dem Server verbinden. Man hat also ein Keyword, das man bei der Verbindung angibt und bekommt so die Informationen von dem ActiveMQ zu genau diesem Topic. Der Vorteil bei diesem Ansatz ist, dass man beliebig viele Subscriber auf einem Topic haben kann. Sobald ein Publisher eine Änderung für das Topic schickt, wird an alle Subscriber eine Kopie geschickt.

Nach diesem Ablauf wird die Nachricht verworfen, denn das Publisher/Subscriber-Prinzip speichert keine Nachrichten! Wenn kein Subscriber aktiv ist und der Publisher eine Änderung schickt, ist diese verloren. Dies kann mit dem im folgenden Kapitel beschriebenen Pull-Mechanismus umgangen werden. Allerdings ist es auch möglich, über das Topic abzufragen, ob aktuell Subscriber auf dem Topic warten. So könnte der Publisher zumindest feststellen, ob seine Nachrichten „ins Leere“ gehen oder verarbeitet werden.

2.4.3 Pull Mechanismus mit ActiveMQ

Das Problem beim Publisher und Subscriber-System über Topics ist, dass keinerlei Informationen gespeichert werden. Würde man dieses Problem über eine Queue lösen wollen, so hat man alle nicht abgerufenen Änderungen zu verarbeiten - in Kontext dieser Bachelorarbeit ist dies nicht relevant, da es für niemanden von Interesse ist, ob vor 10 Minuten eine Schublade geöffnet wurde oder nicht.

Der Pull-Mechanismus ist eine Kombination aus Topic’s und MessageQueues. Die Idee ist, dass der Publisher eines Topics über eine MessageQueue als Consumer verfügt. Über diese Queue ist es möglich, dass sich Subscriber des Topics „bemerkbar“ machen und den Publisher informieren können, wenn sie den aktuellen Sensorwert haben wollen. Der neu angemeldete Subscriber meldet sich also am Topic an und schickt als Producer eine Aktualisierungsanforderung über die Message Queue an den Publisher. Dieser erkennt die

Anforderung und schickt die aktuellen Sensorwerte nun ganz normal über das Topic an alle Subscriber. Der anfordernde Subscriber erhält die gewünschte Information, alle anderen Subscriber verwerfen die Information, sofern sie unwichtig für sie ist.

Über diese Rückkopplung ist es möglich, dass der Publisher nicht dauernd ohne Anforderung die aktuellen Werte schicken muss und macht die Topics für fast alle Anwendungen interessant.

2.4.4 Namenskonventionen für Topics & Queues

Da der ActiveMQ-Server gerade erst eingeführt wurde, gibt es noch keine expliziten Anforderungen für die Namenskonventionen. Wenn das System aber produktiv in den Einsatz geht und hunderte von Topics verwalten muss, wird diese Richtlinie kommen - deswegen wird in dieser Arbeit schon der erste Ansatz realisiert werden.

Ziel ist es, durch eine Segmentierung der Namen die Übersicht nicht zu verlieren und das System wartbar zu halten. Deswegen wird ein Topic im Stil einer Java-Klasse erstellt und mit Punkten unterteilt. Dies macht gerade bei einer Anwendung wie in dieser Bachelorarbeit Sinn, da eine Vielzahl von Sensoren so besser zugeordnet werden kann.

Ein Beispiel: der erste Sensor des Knoten 7 wird später im ActiveMQ zu finden sein unter **Sensornetz.Node7.Sensor1**

3 Analyse

Nachdem in dieser Arbeit die Grundlagen und die zur Verfügung stehende Hardware vorgestellt wurde, beschäftigt sich das folgende Kapitel mit den Anforderungen an das Sensornetzwerk. Weiterhin gibt es einen kurzen Ausblick über verwandte Arbeiten und Projekte in diesem Gebiet, sodass am Ende dieses Kapitels für das folgende Design alle wichtigen Fakten geklärt sein sollen.

3.1 Anforderungen

Um das Sensornetzerk zu spezifizieren werden die Anforderungen in harte und weiche Anforderungen aufgeteilt. Harte Anforderungen müssen von dem Sensornetz erfüllt werden, während die weichen Anforderungen als Optionen zu verstehen sind, die den Nutzwert oder die Bedienung erleichtern können. Auf die korrekte Notation im Requirement-Stil wurde an dieser Stelle der Übersichtlichkeit wegen verzichtet. Anschließend folgt eine Erklärung, warum diese Anforderungen festgelegt wurden.

3.1.1 harte Anforderungen

H1 Das Sensornetzwerk soll drahtlos sein.

H2 Das Sensornetzwerk soll selbstkonfigurierend, also einfach erweiterbar sein.

H3 Das Sensornetzwerk soll auf Standardhardware basieren.

H4 Die Knoten sollen analoge und digitale Werte einlesen können.

H5 Die Knoten sollen über fernsteuerbare, digitale Ausgänge verfügen.

H6 Die Anwendung soll die Sensorwerte im ActiveMQ bereitstellen.

H7 Die Anwendung soll eine Schnittstelle für andere Anwendungen bereitstellen, um die Digitalausgänge der Knoten zu steuern.

3.1.2 weiche Anforderungen

W1 Das Sensornetzwerk soll im Fehlerfall weiter funktionieren.

W2 Die Knoten sollen drahtlos neu programmiert werden können.

W3 Die Anwendung soll eine einfach zu verwaltende, grafische Oberfläche bereitstellen.

W4 Die Kosten sollen beachtet werden.

W5 Die verwendeten Protokolle sollen einfach gehalten werden.

3.1.3 Erläuterung der Anforderungen

Im Living Place Hamburg ergeben sich durch die künftige Anwendung des Sensornetzwerkes gewisse Anforderungen, die in einem fremden Kontext durchaus fragwürdig erscheinen können, deswegen wird kurz auf jeden Punkt der Anforderungen eingegangen.

Durch die flexible Einsatzmöglichkeit müssen die Knoten drahtlos sein [H1], anders wäre z.B. der Einsatz in beweglichen Objekten nicht möglich. Eine Selbstkonfiguration des Netzwerkes ist ebenfalls gefordert [H2], damit später einfach Knoten hinzugefügt werden können, ohne die Netztopologie ändern zu müssen. Dies ist fast einhergehend mit [H3], denn die Standardhardware soll verwendet werden, um später keine Kompatibilitätsprobleme zu bekommen bei der Erweiterung des Netzes.

Die Funktionsbeschreibung der Knoten setzt voraus, dass sowohl analoge als auch digitale Werte im Rahmen der Hardwaregenauigkeit eingelesen werden können [H4]. Dies ist notwendig, um eine Kompatibilität zu Sensoren zu schaffen, die über ein digitales ein/aus hinaus gehen (z.B. Abstandssensoren). Weiterhin sollen die nicht genutzten Ports am Arduino als ferngesteuerte Ausgänge dienen [H5]. Da keine weitere Anforderung nötig ist, wird der Strom und die Spannung durch den Arduino begrenzt.

Die Anwendung soll alle Sensorwerte von den Knoten entgegennehmen, diese verarbeiten, zuordnen und dem ActiveMQ der Wohnung zur Verfügung stellen (Subscriber-Dienst) [H6].

Weiterhin soll die Anwendung eine Schnittstelle für Fremdanwendungen zur Verfügung stellen, in der über die ID's und den I/O-Port über eine einfache Semantik die digitalen Ausgänge gesteuert werden können [H7].

In den weichen Anforderungen wird in [W1] gefordert, dass das Sensornetzwerk im Fehlerfall weiter funktionieren soll. Dies gilt für die Sensorknoten, damit diese bei einem Ausfall die anderen Knoten nicht stören.

Ein drahtloses Update der Sensorknoten [W2] wäre wünschenswert, damit man alle Knoten bequem von einer Stelle aus mit einer neuen Firmware flashen kann. Die Möglichkeit besteht bereits, bedingt aber einen manuellen Reset des Knotens vor Beginn des Flashvorganges.

Für die einfache und intuitive Bedienbarkeit wurde [W3] gefordert. Eine Textanwendung würde komplizierter zu bedienen sein als eine grafische Oberfläche. Um keine Darstellungsfehler zu bekommen, wäre eine plattformunabhängige Programmierung (z.B. in Java) sinnvoll.

Die Erweiterbarkeit soll gewährleistet werden[H2], bei diesem Punkt sind die Kosten der Hardware ein nicht vernachlässigbarer Grund. Somit wurde [W4] mit in die Anforderungen aufgenommen. Es ist jedoch nur eine weiche Anforderung, da es sich hier um ein Pilotprojekt handelt, welches nur eine begrenzte Stückzahl an Knoten einsetzt.

Auch der Erweiterbarkeit geschuldet ist die Anforderung [W5]. Durch einfache Protokolle wird gewährleistet, dass spätere Änderungen problemlos verarbeitet werden können.

3.2 verwandte Arbeiten // existierende Konzepte

In der Masterarbeit von Stefan Meißner zum Thema „Realisierung von Sound und Event Awareness durch verteilte Sensorknoten“ ([Meißner, 2010]) wird ein Sensornetzwerk aufgebaut um verschiedene Geräusche zu visualisieren. Die Motivation dieser Arbeit ist die Hilfestellung für gehörlose Menschen: jeder Knoten besitzt ein Mikrofon, welches mögliche Geräusche (z.B. das Schreien eines Kindes) erkennt. Zusätzlich gibt es Konventionen über eine Zuordnung von Farben und Räumen. So steht z.B. die Farbe rot für die Küche, die Farbe blau für das Schlafzimmer.

Tritt nun ein Geräusch in der Küche auf, sendet der Küchenknoten dieses Ereignis an alle Knoten des Netzes. Diese verfügen über RGB-LED's und können nach dem Empfang des Küchen-Ereignisses die Farbe rot visualisieren. Dadurch, dass die Sensoren in jedem Raum

vorhanden sind, weiß nun die gehörlose Person, dass in der Küche ein Geräusch aufgetreten ist und sie ggf. dort nach dem Rechten sehen sollte.

Die Arbeit beschäftigt sich mit dem ZigBee-Protokoll, basierend auf Arduino-Hardware. Die Sound-Awareness-Mote's bestehen aus diesen Standardkomponenten; die Logik ist jedoch um ein Vielfaches komplexer als nur ein Sensornetzwerk. Es werden Rückschlüsse gezogen auf eine Art Kategorie des Geräusches, schließlich macht es einen großen Unterschied, ob ein Kind schreit, oder ob die Kühlschranktür vom Lebenspartner geschlossen wurde. Dies kann mit einer Dringlichkeit ebenfalls zum Ausdruck gebracht werden: leichtes Fading der Farben kann auf ein spielendes Kind hindeuten, ein konstantes Ein- und Ausschalten auf ein Telefonanruf oder die Türklingel und ein kontinuierliches Aufblitzen könnte für einen Feueralarm stehen.

Der Schwerpunkt dieser Arbeit ist gänzlich anders; trotzdem ist die Technik im Hintergrund vergleichbar. Es soll ebenfalls mit Standardhardware gearbeitet werden, lediglich die spätere Verarbeitung der generierten Events wird in dieser Arbeit zentral erfolgen und nicht auf sämtlichen verfügbaren Knoten.

3.2.1 andere Projekte

Das Projekt i2home ist vergleichbar mit dem Living Place Hamburg, nur mit dem Unterschied, dass die Zielgruppe in diesem Projekt ältere Menschen, bzw. Menschen mit geringer Behinderung sind. Das Projekt ist ein europäisches Gemeinschaftsprojekt internationaler Hochschulen, die auf ihrem jeweiligen Gebiet forschen. Der Forschungsschwerpunkt liegt auf der intuitiven Interaktion mit Elektronik, daher werden in dieser Arbeit die Sensornetzwerke nur für sekundäre Aufgaben verwandt.

Interessanter scheint das Projekt Amigo (Ambient intelligence for the networked home environment) zu sein. Dies ist ein Gemeinschaftsprojekt von über 15 Firmen, welche das Zusammenspiel von Hard- und Software, Consumer-Elektronik und mobile und stationäre Netzwerke vereinen um das Wohnerlebnis zu verbessern. Es gibt wie beim Living Place Hamburg eine Musterwohnung, diese ist im High-Tech Campus Eindhoven untergebracht.

Das Projekt beruht auf einer umfassenden Middleware, welche sämtliche Ereignisse von Benutzer, Computer und sonstigen elektronischen Geräten zusammenträgt und vereint. Die gesammelten Informationen ermöglichen die Analyse der Bewohner und sollen bei der Interaktion behilflich sein, den gewünschten Kontext des Benutzers zu erfahren.

Leider war nur zu erfahren, dass dieses Projekt mit mehreren Sensornetzwerken zur Datenübertragung zwischen der Middleware und den einzelnen Komponenten arbeitet, jedoch wurden trotz intensiver Suche kein Paper oder andere Informationen zu diesem Thema gefunden.

Im Allgemeinen können Sensornetzwerke überall dort eingesetzt werden, wo man daran interessiert ist, etwas zu überwachen. Die Technik scheint prädestiniert für militärische Anwendungen und wird dort auch z.B. von der US Army eingesetzt. Dieses System nennt sich REMBASS (Remote Battlefield Sensor System) und ist dafür zuständig, Bewegungen am Boden festzustellen. Diese Technik ermöglicht es, Personen im Bereich von 3 - 50 Metern zu erkennen, Fahrzeuge bis zu 350 Meter. Der Nachteil ist, dass die Technik sehr groß und auffällig ist, deswegen geht auch das Militär zur sog. SmartDust-Lösung über. SmartDust ist „intelligenter Sensorstaub“, welcher so unauffällig klein ist, dass feindliche Einheiten ihn nicht wahrnehmen. Natürlich braucht man im Vergleich zu dem REMBASS-System deutlich mehr Sensorknoten, dennoch ist je nach Einsatzgebiet die SmartDust-Variante vorteilhaft.

Im zivilen Leben werden Sensornetze an allen nur denkbaren Orten eingesetzt. BP zum Beispiel setzt in einem Chemielager auf die Technik von Particle Computer - einem Projekt der Universität Karlsruhe. Dort wurden in einem Pilotprojekt Gefahrstoffe mit Sensoren bestückt, die über verschiedenste Sensoren für Position, Temperatur, Bewegung, Vibration und Licht verfügen. Sobald ein Sensor eine Auffälligkeit feststellt, werden die Nachbarknoten informiert und ggf. Alarm ausgelöst.

Doch es gibt auch viel größer angelegte Sensornetze. In großen Waldstücken werden z.B. Sensoren platziert, die über einen möglichen Waldbrand informieren sollen. Anhand dieser Daten können Löscharbeiten effektiver und präziser durchgeführt werden. Jedoch sollen diese Geräte nicht nur über den worst-case informieren, sondern melden vorher schon gefährliche Werte von Temperatur und Luftfeuchtigkeit.

Seit 2002 benutzt Intel in Zusammenarbeit mit kalifornischen Winzern auch Sensornetze um in den Rebstöcken minütlich die Temperatur an einen Zentralrechner zu übermitteln. Anhand dieser Daten kann der Winzer entscheiden, ob die Felder bewässert werden müssen und wann er mit der Ernte beginnen kann.

Das Gebiet der Sensornetzwerke explodiert förmlich, die Kosten-pro-Stück fallen immer weiter und somit ein noch massiverer Einsatz von Sensornetzwerken in der Zukunft durchaus wahrscheinlich.

4 Design

Nachdem nun die Anforderungen an das Sensornetzwerk festgelegt wurden, muss nun ein geeignetes Konzept erarbeitet werden, in dem die Anforderungen umgesetzt werden. Dazu werden verschiedene Aspekte miteinander verglichen und ein für diese Arbeit optimales Ergebnis erzielt. In Bild 4.1 sieht man den geplanten, schematischen Aufbau des Sensornetzes.

Neben der Hardwareauswahl wird über ein geeignetes Funkprotokoll diskutiert; es wird aber auch über die eigenen Protokolle, Konventionen und Übertragungslogiken gesprochen, da diese im Mittelpunkt der Arbeit stehen.

4.1 Hardwareauswahl

In den Grundlagen wurden diverse Standardplattformen des Arduino vorgestellt. Die meisten der Arduino's basieren auf dem selben Chip, sodass die Funktionalität nicht im Vordergrund der Entscheidung steht.

[H3] der Anforderungen besagt, dass in dieser Arbeit nur mit Standardhardware gearbeitet werden soll. Dies würden alle Arduinos erfüllen, jedoch soll die Funk-Komponente nicht über Drähte verbunden, sondern es sollen die dafür vorgesehen Shields verwendet werden. Fällt also die Wahl auf Bluetooth, so soll das Bluetooth-Modul nicht mittels eines Breadboardes verbunden werden, sondern über das passende Shield.

Dies hat zur Folge, dass uns nur noch drei Arduino's zur Auswahl bleiben: der Diecimila, der Duemilanove und der Mega. Alle drei sind kompatibel zu den Shields und kommen deswegen in die engere Auswahl.

Der Diecimila fällt aufgrund seines Alters raus, er besitzt einen alten Chip und wird auch nicht mehr produziert, da er letztes Jahr durch den Duemilanove abgelöst wurde. Somit bleiben

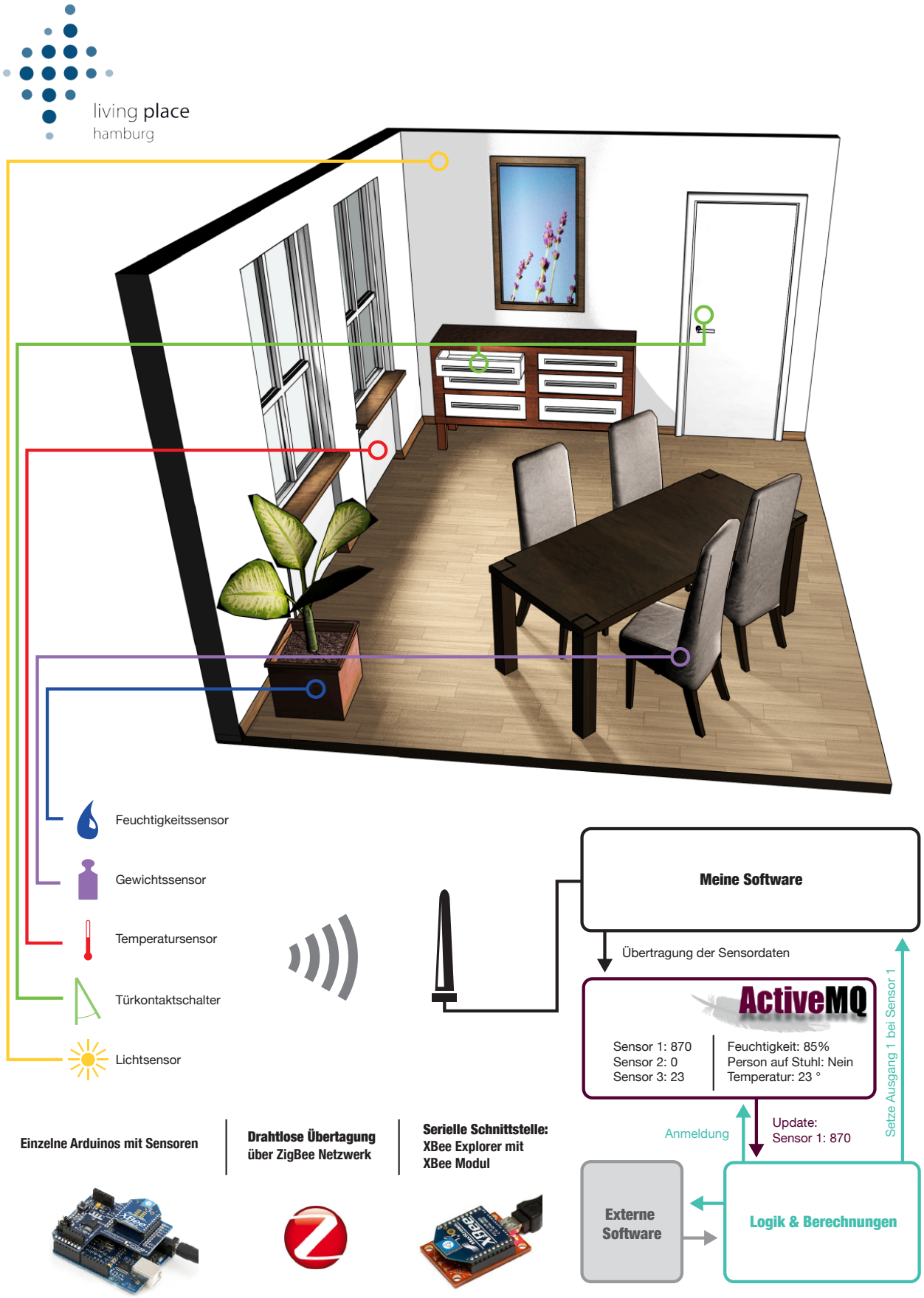

Abbildung 4.1: schematische Darstellung des Sensornetzwerkes

nur noch der Duemilanove und der Mega zur Auswahl. Beide Boards sind fast identisch, nur dass der Mega einen größeren Chip besitzt und über deutlich mehr I/O-Ports verfügt.

Die Entscheidung zwischen Mega und Duemilanove ist also abhängig von der Anzahl der benötigten I/O-Ports. Da wir pro Knoten nur einen Sensor vorgesehen haben, wird als Eingang maximal ein analoger Eingang und ein digitaler Eingang benötigt. Letztlich würde der Mega also nur Vorteile bringen, wenn wir extrem viele Ausgänge benötigten - dies ist aber nicht der Fall. Somit wird aus wirtschaftlichen Gründen der Duemilanove ausgewählt, da er sämtliche Anforderungen erfüllen kann und die günstigere Variante des Boardes ist.

Selbstverständlich ist eine Kompatibilität zu dem Mega angestrebt, sollten also irgendwann mehr Ein- oder Ausgänge benötigt werden, als der Duemilanove zur Verfügung stellen kann, ist ein einfacher Umbau auf den Mega kein großes Problem.

4.2 Wireless

Es wurde kabelloser Ansatz bei der Realisierung gefordert, dies zieht die Auswahl eines geeigneten Standards nach sich. Da es diverse drahtlose Übertragungsmechanismen gibt, wird an dieser Stelle kurz der Standard erklärt und es wird kurz auf die Vor- und Nachteile eingegangen.

Den Standard liefert uns die IEEE (*Institute of Electrical and Electronics Engineers*) 802. Dort werden sämtliche für Netzwerktechnik benötigten Protokolle beschrieben - natürlich interessieren uns speziell nur die kabellosen Standards wie der 802.11 (WLAN) und der 802.15 (WPAN) (vgl. [Tanenbaum, 2003]). Beide Standards beschreiben die unteren ISO/OSI-Schichten und ermöglichen so die drahtlose Kommunikation für geringe Reichweiten bis zu 100 Metern.

4.2.1 Bluetooth

Bluetooth (IEEE 802.15.1) wurde von Ericsson maßgeblich entwickelt und ist heutzutage eigentlich jedem bekannt: quasi jedes Handy, moderne Autos und die meisten Notebooks 'sprechen' Bluetooth. Eigentlich war dieser Standard für medizinische Bereiche entwickelt worden, hat sich aber auf Grund der nutzbaren Profile schnell zu einem Allgemeinstandard ausgeweitet.

Bluetooth läuft im lizenzfreien ISM-Band (industry, scientific & medical band) - genauer gesagt zwischen 2,402 GHz und 2,480 GHz. Bei dieser Frequenz können einem sofort diverse „Probleme“ einfallen: Mikrowellen, WLAN, Funktelefone, etc. sind ebenfalls in diesem Frequenzbereich zu Hause. Damit diese Fremdanwendungen keinen Einfluss auf die Verbindungsqualität haben, nutzt Bluetooth ein Frequenzsprungverfahren, welches 1.600 mal pro Sekunde die Frequenz wechselt. Durch dieses Verfahren ist eine Störung quasi ausgeschlossen, da das gesamte Band aufteilt wird in 79 Unterfrequenzen mit je 1 MHz Abstand. Man bräuchte also, um Bluetooth komplett lahm zu legen, einen Störsender mit 40 MHz Frequenzhub - das ist technisch realisierbar, jedoch nicht in gewöhnlichen Umgebungen zu finden.

Bluetooth galt als abhörsicher durch den verwendeten Schlüssel zwischen den kommunizierenden Stationen. Die Erfahrung (bzw. die Security-Gemeinde im Internet) hat aber gezeigt, dass auch längere Schlüssellängen durchaus ohne Probleme in Echtzeit geknackt werden können. Für kurze Schlüssel reicht dafür ein normaler RISC-Prozessor, für längere Schlüssellängen muss man auf die GPU zurückgreifen.

Durch die verschiedenen Profile könnte man Bluetooth auch im Rahmen dieser Arbeit gut einsetzen, hier käme das SPP (serial port profile)-Profil zum Einsatz, mit dem man einfach Daten übertragen kann. Auch einen der drei Energiesparmodi ist für diese Anwendung optimal, somit ist ein geringer Energieverbrauch gewährleistet. Jedoch gibt es einen klaren Nachteil bei dem Energiesparen: Die Bluetoothmodule können unter Umständen mehrere Sekunden brauchen, bis sie aufwachen.

Der große Nachteil von Bluetooth ist die Skalierbarkeit: es kann in einem Netzwerk nur maximal 255 Geräte geben. Von diesen 255 Geräten können maximal 8 Geräte (3 Bit zur Adressierung) zeitgleich aktiv sein, also senden. Die restlichen 247 Geräte müssen dann geparkt werden, was einen erheblichen Nachteil darstellt. Ein weiterer Nachteil wäre, dass Bluetooth schnell unwirtschaftlich wird bei größerer Stückzahl; weiterhin ist die Komplexität und die Systemarchitektur alles andere als angemessen für die Nutzdaten im Kontext dieser Bachelorarbeit.

4.2.2 IrDA

In der *Infrared Data Association* haben sich 1993 ca. 50 Unternehmen zusammengeschlossen, um einen gemeinsamen Standard zu entwickeln. Infrarot ist kein neues Thema gewesen, jedoch wurde es bisher nur z.B. für Fernbedienungen mit einem einfachen RC5-Code

verwendet. Dieser Zusammenschluss der Unternehmen (unter maßgeblicher Führung von HP, IBM und Microsoft) wollte eine Datenübertragungsschnittstelle im Infrarotbereich von 850 - 900 nm. Dies sollte eine reine Punkt-zu-Punkt-Verbindung werden und im Nahbereich von bis zu einem Meter funktionieren.

Der Vorteil dieser Technik war die vergleichsweise hohe Datenübertragungsrate von bis zu 4 MBit/s (bei FIR - fast infrared), spätere Entwicklungen gehen bis zum Giga-IR mit bis zu 512 MBit/s bzw. 1 GBit/s. Angewandt wurde die Infrarottechnik hauptsächlich im Bereich des Datenaustausches zwischen Handys und Notebooks/PDA's. Auch HP baute eine Zeit lang Drucker mit Infrarotschnittstelle, welche einfaches, kabelloses Drucken ermöglichte.

Ob die „Abhörsicherheit“ durch die geringe Reichweite von Pro- oder Contra-Argument ist, bleibt ein Streitpunkt; Fakt ist jedoch, dass keine Verschlüsselungsalgorithmen angewandt wurden. Der größte Nachteil von Infrarot ist jedoch die benötigte Sichtverbindung zwischen Sender und Empfänger. Dies ist im Kontext dieser Arbeit ein großes Problem, da die Übertragung über diese Technik nicht immer gewährleistet werden kann. Somit ist der Anwendungsbreich deutlichst eingeschränkt und würde nur in ortsunveränderlichen Installationen Sinn machen.

4.2.3 ZigBee

ZigBee ist ein von der ZigBee Alliance™entwickelte Funkstandard. Die 2002 gegründete ZigBee Alliance™ist ein Zusammenschluss von mehr als 230 Unternehmen, welche ein gemeinschaftliches Protokoll für Haushaltsgeräte, Sensoren, etc. auf Kurzstrecken bis 100 Meter zu verbinden. Ziel war es, dies möglichst einfach und kostengünstig zu realisieren.

Der Standard wurde aufbauend auf der IEEE 802.15.4 geschaffen und benutzt die beiden unteren Schichten PHY (Bitübertragungsschicht) und MAC (Sicherungsschicht). ZigBee ist also nur ein Protokoll, welches auf dem 802.15.4-Standard aufsetzt und ihn erweitert.

Dieser ZigBee-Standard bietet sehr vielen möglichen Anwendungen ein einfaches Protokoll, welches für den Anwender sehr einfach zu konfigurieren ist. Es ist möglich mit Prioritäten zu arbeiten, d.h. es kann z.B. ein geringerer Stromverbrauch erreicht werden zu Lasten der Nachrichten-Latenz und umgekehrt. Das ZigBee-Netzwerk kann eine theoretische Anzahl von 65.536 Knoten aufnehmen und ist im Hinblick auf die Skalierbarkeit zu Bluetooth somit deutlich mächtiger.

In einem ZigBee-Netzwerk gibt es verschiedene Aufgaben der einzelnen Knoten, dazu zählen:

Koordinator (ZigBee coordinator, ZC): Der Koordinator ist in jedem ZigBee-Netzwerk genau einmal vorhanden. Er gibt grundlegende Parameter für das PAN (personal area network) vor und verwaltet das Netz. Er ist auch für die Aufnahme neuer Knoten zuständig und soll nach [Digi International Inc., 2010] eine ständige Stromzufuhr haben.

Router (ZigBee Router, ZR): Ein Router ist ein sog. FFD (full function device), d.h. sie bauen das Netzwerk mit auf und können zeitgleich als Endgerät Daten liefern oder verarbeiten.

Endgerät (ZigBee End Device, ZED): Das Endgerät ist ein RFD (reduced function device). Es meldet sich nur beim nächsten Router an und kann selber keine Routingaufgaben oder sonstige Netzwerkaufgaben erledigen.

Die Vorteile von ZigBee liegen auf der Robustheit, dem geringen Energieverbrauch und nicht zuletzt bei den geringen Kosten. Es ist ein sehr einfaches Protokoll und sehr einfach erweiterbar. Die Netztopologie baut sich selbstständig auf und verfügt über sichere Routingalgorithmen: wenn ein Knoten ausfällt, wird automatisch über andere Knoten geroutet.

Die Nachteile liegen in dem single-point-of-failure, dem Koordinator. Wenn dieser aus irgendeinem Grund abstürzt oder nicht mehr erreichbar ist, funktioniert das gesamte Netz nicht mehr. Abhilfe schaffen bei diesem Problem die FFD's: Diese sind so konfigurierbar (vgl. [Digi International Inc., 2010]), dass die Aufgabe des Koordinator automatisch übernommen werden kann bei Ausfall von selbigem.

4.2.4 WLAN

WLAN ist heutzutage in jedem Notebook Standard, eine Vielzahl von Wohnungen verfügt über die Wireless-LAN Technik. Natürlich verfügt auch der Living Place Hamburg über WLAN, sodass die Infrastruktur teilweise schon vorhanden wäre.

Da der WLAN Standard (IEEE 802.11) hinreichend bekannt sein sollte, werde ich nicht weiter auf die Technik an sich eingehen, sondern gehe direkt über zu den Vor- und Nachteilen dieses Ansatzes.

Wie schon erwähnt, wäre die Infrastruktur in der Wohnung bereits vorhanden; es ist bekannt, dass die Versorgung überall ausreichend ist und somit ein Sensor überall funktionieren wird.

Die Kosten sind vergleichsweise akzeptabel, solange man keine Großserie von Sensoren in der Wohnung unterbringen möchte.

Der größte Nachteil dieser Technik ist der erhebliche Protokollaufwand, der implementiert werden muss. Das WLAN-Protokoll ist viel zu umfangreich, „um ein paar Daten zu übertragen“. Ein weiterer Nachteil wäre auch, dass das Netzwerk durch mögliche Kollisionen jedes mal einen SlowStart machen würde, sodass die Performance des gesamten WLANs extrem sinken würde (vgl. [Tanenbaum, 2003]). Da sich dies auf alle anderen Netzwerkteilnehmer auswirken würde, müsste man hier einen eigenen AccessPoint installieren, was wiederum Mehrkosten nach sich ziehen würde.

4.2.5 Zusammenfassung und Auswahl

WLAN ist für die geforderten Zwecke überdimensioniert, es bietet Funktionen, die nicht gebraucht werden. Außerdem ist es nicht für den gewünschten Zweck konzipiert worden, sodass wir es gewissermaßen zweckentfremden würden. Dieser Overhead in der Programmierung und im Protokoll stehen in keinem Vergleich zu dem Nutzen, den WLAN uns bietet. Somit fällt diese mögliche Lösung aus.

Infrarot ist eine Technik um schnell Daten zu übertragen, dies wäre sicherlich eine denkbare Lösung für Festinstallationen. Da aber z.B. die Sensoren in beweglichen Dingen wie Stühlen eingebaut werden könnte, ist diese Technik aufgrund der benötigten Sichtlinie zum Empfänger auch nicht realisierbar. Man könnte sicherlich überall in der Wohnung IR-Empfänger einbauen, jedoch würde dies den Vorteil der drahtlosen Übertragung quasi zunichte machen und es wäre nicht gewährleistet, dass trotz einer Vielzahl von Empfängern die Sensordaten ankommen.

Noch zur Auswahl stehen also Bluetooth und ZigBee - die Entscheidung zwischen diesen beiden annähernd identischen Protokollen könnte man mit einer Nutzwertanalyse ohne Probleme zu einer Entscheidung bringen. Da wir aber auf dieses Mittel verzichten wollen, wird die Entscheidung anhand von Kriterien in Bezug auf die Problemstellung bezogen. Bluetooth hat mit 255 möglichen Endgeräten zwar mehr Geräte, als jemals in dem Living Place verbaut werden würden, jedoch stört der Ansatz, dass nur 8 Geräte zeitgleich aktiv sein können. Bei ZigBee sind mit 65.536 Geräten noch weniger Grenzen gesetzt in Sachen Skalierbarkeit, jedoch ist der gesamte Protokollaufbau von ZigBee deutlich schlanker als der von Bluetooth. Bei Bluetooth muss z.B. beim Verbindungsaufbau ein Handshake gemacht werden für den Schlüssel, dies entfällt alles bei ZigBee. Auch die verschiedenen Profile, die bei Bluetooth

schon implementiert sind, werden nicht benötigt. Es sollen nur einfachste Daten übertragen werden, dies wäre mit einem Bluetoothprofil ohne Probleme möglich; jedoch ist das Netz nicht problemlos erweiterbar.

Man kann also zusammenfassen, dass Bluetooth und ZigBee jeweils Vor- und Nachteile haben. In dem Kontext des Living Place Hamburg wird jedoch nur ein einfachstes Protokoll benötigt. Es gibt keine Anforderungen an die Sicherheit, also muss keine Verschlüsselung mit gemeinsamen Schlüsseln erfolgen, weitere Anwendungen, wo die Bluetooth-Profile zur Anwendung kämen, sind nicht geplant und würden in diesem Kontext auch keinen Sinn machen. Es gibt also von der Anforderung her keinen Grund, warum Bluetooth zu bevorzugen wäre - die nicht nutzbaren Vorteile, die es bietet, würden einen erheblichen Mehraufwand nach sich ziehen.

Letztendlich fällt die Wahl auf das ZigBee-Protokoll, da es durch sein schlankes Protokoll ideal für unsere Anwendung erscheint und zusätzlich alle Anforderungen erfüllen und übertreffen kann durch z.B. optionale Verschlüsselung. Die Nachteile, z.B. der single-point-of-failure lassen sich durch entsprechende Konfiguration umgehen, sodass sich diese Nachteile kompensieren lassen.

4.3 Sensornetzwerk

Damit das Sensornetzwerk funktioniert, müssen alle Teilnehmer (in unserem Fall steht das Wort Teilnehmer für die XBee-Module) entsprechend konfiguriert werden. Dazu sind einige Konventionen und Erklärungen notwendig, die in diesem Kapitel erörtert werden.

4.3.1 Konfigurationsprogramm X-CTU

Die Einstellungen an dem XBee-Modul „XBee Series 2“ werden über das Tool X-CTU (Bild 4.2) von Firma Digi ausgelesen und verändert. Da für diese Arbeit fabrikneue Hardware bereitgestellt wurde, mussten die Module nicht auf den selben Firmwarestand gebracht werden und somit gibt es auch keine Unterschiede in der Konfiguration.

Sämtliche dieser Einstellungen sind über die sog. AT-Commands auch per Software direkt vom Arduino abruf- und veränderbar. Da aber diese Einstellungen im laufenden Betrieb nicht relevant sind, wird einmal das Modul entsprechend konfiguriert und ist dann bereit für den Betrieb.

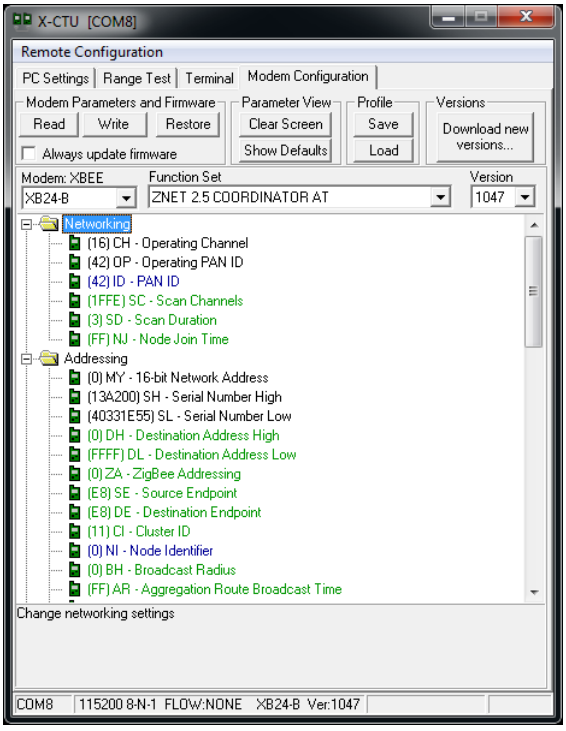

Abbildung 4.2: X-CTU Konfigurationsprogramm von Digi

Die Einstellungen sind in verschiedene Bereiche wie Networking, Addressing und Security (als Beispiel) unterteilt. Jede Einstellmöglichkeit hat ein eindeutiges Kürzel, z.B. steht „ID“ für die gewünschte Netzwerk-ID; „NI“ steht für den Node Identifier - also eine eindeutige Kennung des Knotens (vgl. Bild 4.2).

4.3.2 Konfiguration XBee

Für ein funktionierendes Netzwerk braucht man nur einen Bruchteil der verfügbaren Einstellmöglichkeiten, dazu zählen:

ID PAN ID: Diese Einstellung ist die Netzwerk-ID, die bei jedem Knoten im selben Netzwerk gleich sein muss. Die Einstellung reicht von 0 bis 0x3FFF, was 16.383 verschiedenen Netzwerken entspricht. In dieser Arbeit wird die PAN-ID 42 verwendet und ist auf allen XBee-Modulen gleich.

DH Destination Address High: sind die oberen 32 Bit der Zieladresse. Diese sind normalerweise immer 0, da die unteren 32 Bit ausreichen für normale Anwendungen.

DL Destination Address Low: sind die unteren 32 Bit der Zieladresse. Wenn man den Koordinator erreichen möchte, setzt man die Bits auf 0, somit empfängt nur der Koordinator die Nachricht; wenn man hingegen die Bits auf 0xFFFF setzt, ist es ein Broadcast und alle Teilnehmer des Netzwerkes empfangen die Nachricht. Die Standardeinstellung in dieser Arbeit wird 0 sein, da die Endknoten nur mit dem Koordinator kommunizieren sollen und nicht untereinander.

NI Node Identifier: dies ist die eindeutige ID im Netzwerk. Diese muss eindeutig sein, da später die ID per Software ausgelesen wird und für die Zuordnung verwendet wird. Der Koordinator erhält die ID 0, die Router erhalten die nachfolgenden ID's ab „1". Diese Einstellung ist eigentlich ein String, man kann also auch „router1" verwenden - in dieser Arbeit wird dies jedoch auf Zahlen beschränkt.

BD Baud Rate: ist die Geschwindigkeit, mit der das XBee-Modul Daten über die serielle Schnittstelle empfängt und über Funk versendet. Diese Einstellung muss ebenfalls überall gleich sein, damit die Kommunikation funktioniert. Hier wird die Maximalgeschwindigkeit von 115.200 verwendet.

Weiterhin muss das „Function Set" (=die Aufgabe, die das Modul übernehmen wird) einheitlich auf „ZNET 2.5 ROUTER/END DEVICE AT" stehen. Die einzige Ausnahme davon bildet der Koordinator, welcher als „ZNET 2.5 COORDINATOR AT" konfiguriert wird. Diese Konfiguration ermöglicht den Sensorknoten, als Router zu agieren und so die Reichweite des gesamten Netzes zu erhöhen.

Wichtig für die problemlose Kommunikation ist auch ein gleicher Firmwarestand: 1047 beim Koordinator und 1247 bei den Routern.

Mit diesen Einstellungen ist es möglich, von jedem Router Daten an den Koordinator zu schicken. Da der Koordinator aber auch Daten zurückschicken muss, wird hier die DL-Adresse auf 0xFFFF gesetzt - somit empfangen alle Router die Nachrichten vom Koordinator. Weitere Einstellungen wie PowerLevel, Encryption etc. lassen wir an dieser Stelle erst einmal unberücksichtigt, da sie nicht für die Funktion des Sensornetzwerkes verantwortlich sind.

4.3.3 Einfügen eines neuen Knotens

Hier wird exemplarisch der Ablauf vom Einfügen eines neuen Teilnehmers ins Sensornetzwerk erläutert.

1. Das XBee-Modul wird mittels X-CTU (s. 4.3.2) auf die entsprechende PAN-ID und einen freien Node-Identifier konfiguriert. Dies ist erforderlich, damit der Arduino später den Node-Identifier auslesen kann und für die weitere Kommunikation verwenden kann.
2. Der Arduino wird mittels der in dieser Arbeit zur Verfügung gestellten Firmware geflasht. Der Code beinhaltet keine individuellen Einstellungen, sodass nichts angepasst werden muss.
3. Die programmierte Hardware wird mittels des XBee-Shieldes zusammengesteckt und mit einer Stromquelle verbunden (entweder über USB oder dem Adapter).
4. Der Arduino liest die ID aus dem XBee-Modul aus und schickt über das Sensornetzwerk eine Nachricht an die Software, dass er sich am Netzwerk anmeldet
5. Die Software registriert den Knoten; der Benutzer kann nun die Einstellungen für diesen Knoten in der Software vornehmen.

4.3.4 Ausfall eines Knoten

Ein spezielles Problem in einem Sensornetzwerk ist die Frage nach der Ausfallerkennung. Bei einem einfachen sende-bei-Aktualisierung-Paradigma würde ein Ausfall nicht erkannt werden, da der Sensor keine Änderungen mehr schickt. Dies bedeutet, dass sich nichts ändert, nicht dass der Knoten ausgefallen wäre.

Nun kann man dieses Problem auf zwei einfache Arten angehen: die erste Möglichkeit ist eine Art Ping - also das periodische Anfragen des aktuellen Sensorwertes. Bleibt die Antwort auf die Aktualisierungsanforderung aus, kann die Software zur Sicherheit noch eine zweite Anforderung schicken um mögliche Kollisionen auszuschließen. Sollte die Antwort auf die zweite Anfrage auch nicht kommen, kann die Software davon ausgehen, dass der Knoten ausgefallen oder nicht (mehr) in Reichweite ist. Dies könnte man entsprechend verarbeiten und in der Software z.B. durch eine Meldung oder einen Hinweis kenntlich machen.

Die andere Alternative wäre das periodische Senden des Knotens. Dies könnte über Interrupts des Arduino's realisiert werden: der 16 Bit Timer löst ca. alle 4,2 Sekunden bei maximaler Einstellung und Prescaler aus. Eine selbstgeschriebene ISR (Interrupt Service Routine) kann diesen Wert noch beliebig langsamer machen, indem man z.B. einen Counter im Programm mitlaufen lässt und den Arduino nur jeden 143. Durchlauf (entspricht 10 Minuten) eine Statusnachricht verschicken lässt.

Ohne diese (oder andere XBee-basierte) Mechanismen ist es nicht möglich, einen möglichen Ausfall eines Knoten zu erkennen. Natürlich muss man abwägen, ob es sinnvoll für die Anwendung ist, diese Ausfallerkennung zu verwenden. Jede Nachricht, die verschickt wird und jeder Rechenzyklus des Arduino kostet Strom. Wenn der Arduino mit einer Batterie betrieben wird, muss erörert werden, ob sich der Mehrverbrauch gerechtfertigt ist, um einen möglichen Ausfall erkennen zu können.

4.3.5 Drahtloses Firmwareupdate

Wenn im laufenden Betrieb des Sensornetzes ein Firmwareupdate für die Sensorknoten durchgeführt werden muss, wird dies so realisiert, dass man mit einem Notebook von Knoten zu Knoten geht, das USB-Kabel ansteckt und die Module einzeln flasht. Das ist bei einer übschaubaren Anzahl Knoten einfach, wird jedoch mit steigender Anzahl mehr und mehr unpraktisch.

In der Masterarbeit von Stefan Meißner (s. [Meißner, 2010]) wurde für genau diesen Fall das Programm „RemoteAVR“ (s. Bild 4.3) geschrieben. Es ist ein kleines Tool, welches alle verfügbaren Knoten im Netzwerk auflistet, um sie neu programmieren zu können. Dazu ist es notwendig, dass die Knoten einen speziellen Startup-Code haben, welcher es ermöglicht, den vom RemoteAVR verschickten HEX-Dump entgegen zu nehmen und in den Speicher zu schreiben.

Der Dump selber wird vom Compiler generiert und wird von RemoteAVR erkannt. Um den Updatevorgang nun zu starten, müssen die Knoten neu gestartet werden. Dies ist manuell einfach möglich mittels des Reset-Knopf. Eine bessere Variante bietet jedoch der Watchdog des Mikrocontrollers. Dieser ermöglicht einen echten Software-Reset, sodass selbst der manuelle Reset nicht mehr nötig ist.

Es muss also auf dem Knoten der Startup-Code von [Meißner, 2010] übernommen werden, außerdem muss eine Methode sicherstellen, dass der Software-Reset ausgelöst werden kann (Genaueres ist in [RN-Wissen, 2010] nachzulesen). Mit dieser Konfiguration ist es per RemoteAVR möglich, direkt vom PC aus die Knoten zu flashen.

[1] Bild: „Software RemoteAVR“: [Meißner, 2010]

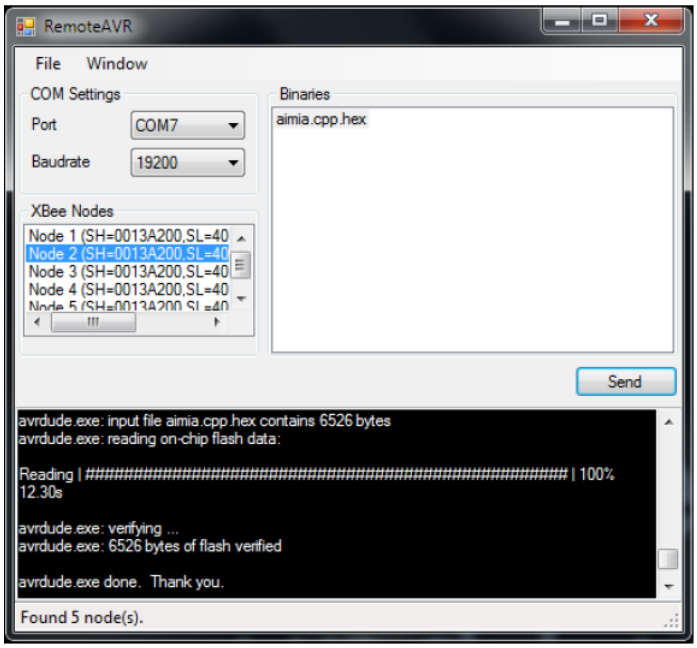

Abbildung 4.3: Software RemoteAVR[1]

4.4 Datenaufbereitung

Ein allgemeins Problem von Sensoren ist, dass sie mehr oder weniger ungenau arbeiten. Kein Sensor dieser Welt arbeitet immer exakt gleich und in jedem Arbeitsbereich mit der selben Genauigkeit. Daraus resultieren auch Probleme für diese Arbeit, denn schließlich sollen die Sensoren auch kompatibel sein zu Analogsensoren. Im folgenden Abschnitt werden die Probleme diskutiert und es wird eine Lösung für eventuelle Probleme geschaffen.

4.4.1 Sensordaten

Der ATmega328, welcher auf dem Duemilanove verbaut ist, verfügt über einen 10 Bit ADC (Analog/Digital-Wandler). Die maximale Genauigkeit liegt bei 4,9 mV bei einer Samplingrate von maximal 10 kHz (ohne Verwendung der seriellen Schnittstelle). Daraus resultiert, dass unser Arduino Spannungen von 0 Volt bis knapp über 5 Volt digitalisieren kann und diese repräsentiert als Integer von 0 bis 1023.

Zum Vergleich: ein normales Oszilloskop verfügt in der Regel über einen 8 Bit ADC, Geräte mit sehr hohen Auflösung haben 12 oder 16 Bit. Damit ist der Arduino für den Messbereich

von 0 bis 5 Volt sehr genau und kann durchaus mit einem Standard-Oszilloskop mithalten. Dies ist Motivation dafür, die Daten nicht künstlich durch Rundungen etc. zu verfälschen.

Leider bringt der gute ADC nicht nur Vorteile. Durch die kleine Genauigkeit von ca. 5 mV treten ständig Schwankungen beim eingelesenen Wert auf - auch wenn dieser eigentlich konstant bleiben soll. Hier schafft ein 10 kΩ-Widerstand, welcher zwischen GND (Masse) und dem analogen Eingang geschaltet wird, Abhilfe (s. Bild 4.4). Dieser sorgt als Pull-Down-Widerstand dafür, dass der Eingang keine Phantomspannung anzeigt. Wenn an den Eingang ein Sensor angeschlossen wird, muss sichergestellt sein, dass der Sensor genug Strom durchlässt um gegen den Widerstand anzuarbeiten - sonst muss der Widerstand entfernt werden.

Ähnlich wie bei dem Analogeingang ist es beim Digitaleingang. Zwar vermutet man, dass dieser Eingang deutlich „unempfindlicher“ wäre als der Analogeingang, doch das ist ein Irrtum. Schon das bloße Berühren des Arduino an den Pins reicht aus, um den Eingang von 0 nach 1 zu bringen. Dies wird ebenfalls mit einem Pull-Down-Widerstand gelöst (s. Bild 4.5).

Mittels dieser beiden Widerstände ist es möglich, genaue Werte mit dem Arduino zu erzielen. Dies ist wichtig, wenn man später bei Änderung des Wertes automatisch eine Nachricht an die Software schicken möchte. Dies wäre ohne die Widerstände nicht möglich, da die Arduinos dauernd Änderungen am Wert erkennen würden und diesen an die Software schicken würden.

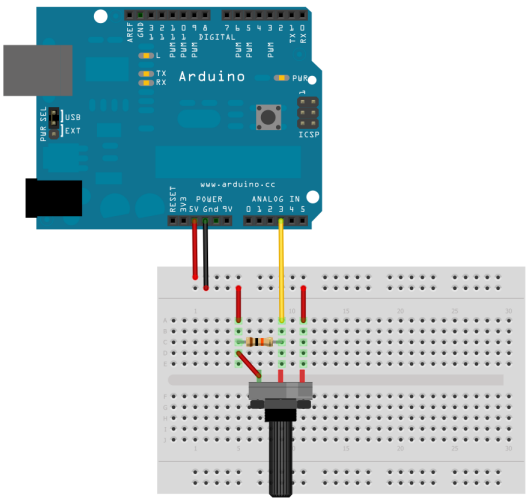

Abbildung 4.4: schematische Darstellung von einem Analogeingang

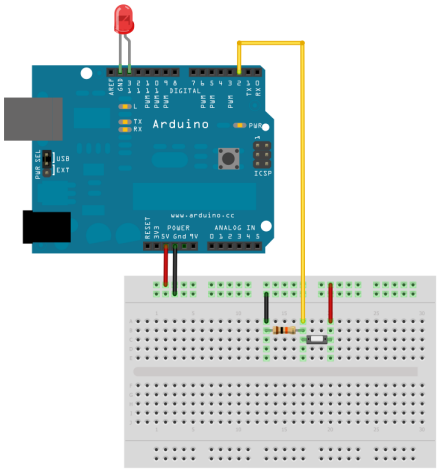

Abbildung 4.5: schematische Darstellung von einem Digitaleingang

4.4.2 Konfiguration der Ein- und Ausgänge für Sensoren

Nicht in jeder Anwendung kommt man mit einem analogen und einem digitalen Eingang aus, sodass ein dynamischer Ansatz wünschenswert wäre. Es wäre nicht sinnvoll, in einem Schrank jede Schublade von einem einzelnen Sensorknoten überwachen zu lassen, deswegen ist ein dynamischer Ansatz in manchen Fällen sinnvoller. Natürlich begrenzt der verwendete Arduino die Anzahl der Ports, dennoch ist es manchmal sinnvoller, nur einen digitalen Ausgang und dafür 10 Eingänge zu haben. Damit dies realisiert werden kann, müssen entsprechende Konventionen getroffen werden, welche die Kommunikation und Konfiguration betreffen.

Um die Ports entsprechend zu konfigurieren, muss eine Nachricht von der Software an den Knoten geschickt werden. Diese kann mit dynamischer Länge die einzelnen Ports im CSV-Stil (kommagetrennte Werte) übertragen werden, alternativ kann auch ein Bitvektor verwendet werden und als einzelne Zahl übertragen werden. Diese Nachricht wird vom Sensorknoten empfangen und verarbeitet. Die Konfiguration der Ein- und Ausgänge erfolgt dann entsprechend.

Softwareseitig könnte die Konfiguration der einzelnen Ports wie in Bild 4.6 aussehen. Digitalport 1 und 2 sind dabei nicht schaltbar, da es sich um die serielle Schnittstelle zu dem XBee-Modul handelt.

Da bei dem dynamischen Ansatz nicht klar ist, wie viele Sensoren ihre Information übertragen werden, kann in der Datenstruktur nicht mit festen Werten gearbeitet werden. Es muss

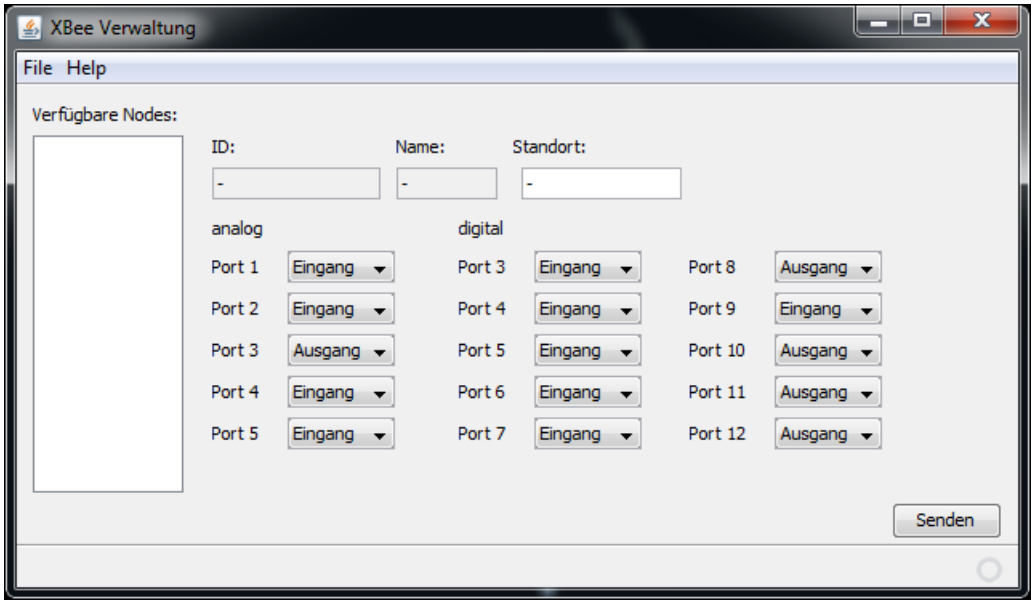

Abbildung 4.6: Konfiguration Ein- und Ausgänge

also idealerweise am Anfang mitgeteilt werden, aus wie viel Byte die Nachricht besteht, um sie so komplett verarbeiten zu können. Bei der Konfiguration der Sensorknoten besteht selbiges Problem: der im Vergleich zur Java-Software „einfache“ Sensorknoten muss am Anfang der Nachricht mitgeteilt bekommen, wie lang die Nachricht wird um z.B. ein Byte-Array anlegen zu können.

Die Speicherung der Daten im EEPROM kann sich auch als sinnvoll erweisen, wenn der Sensorknoten lange an seinem Standort verbleiben wird. Die begrenzte Anzahl der Schreibezyklen ist für diesen Fall jedoch mehr als ausreichend. Es können Daten über die eigene ID, Sensortypen, Portkonfigurationen, etc. gespeichert werden, sodass diese bei einem Neustart des Arduino sofort zur Verfügung stehen und nicht erst über das Sensornetzwerk übertragen werden müssen.

4.4.3 Übertragungsintervalle

Eine wichtige Frage ist, wie oft die Sensordaten übertragen werden sollen. Hierbei kommt es ganz auf den Einsatzzweck an - während ein Schubladensensor nur seine Werte übertragen soll, wenn die Schublade geöffnet oder geschlossen wird, wäre es bei einem Temperatursensor oder Abstandssensor z.B. sinnvoll, dass bei einer gewissen Änderung des Messwertes automatisch ein Update geschickt wird.

Durch die Charakteristik der Sensortypen lässt sich festlegen, dass diese Unterteilung *digital - nur bei Änderung* und *analog - ab einem gewissen Delta* die einfachste und sinnvollste Methode ist. Dieses Delta sollte dem Knoten übermittelt werden und nicht fest einprogrammiert sein, da unterschiedliche Sensoren auch unterschiedliche Werte liefern. Eine mögliche GUI-Konfiguration ist in Bild 4.7 zu sehen. Es kann pro Port ausgewählt werden, ob das Update nach Ablauf einer Zeitperiode, bei einem über- oder unterschreiten des Messwertes um die eingegebene Toleranz oder bei einer Kombination aus beidem erfolgen soll.

Es sollte aber nicht außer Acht gelassen werden, dass es auch eine Nachricht geben sollte, welche den Aktualisierungsprozess auf dem Node aktiviert. Somit kann man direkt von der GUI eine Aktualisierung anfordern um den aktuellen Wert des Sensors zu bekommen, falls z.B. der Wert sich nur minimal geändert hat und deswegen noch keine automatische Aktualisierung geschickt wurde.

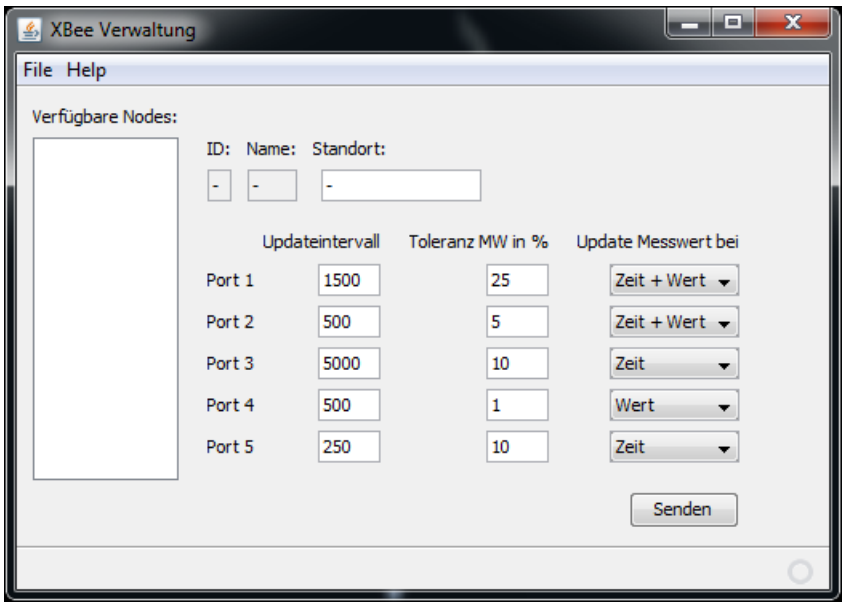

Abbildung 4.7: erweiterte Konfiguration Analogeingänge

4.4.4 Zuordnung der Sensoren

Der große Nachteil eines drahtlosen Netzwerkes ist, dass man die Position der einzelnen Knoten nicht wie bei einer Festverkabelung festlegen kann. Die Sensoren können naturgemäß transportiert und schnell an anderen Orten eingesetzt werden. Somit ist es nicht

sinnvoll, direkt in dem Sensor die Position „einzuprogrammieren“, sondern dies muss über die Software geschehen.

Dort muss es einen String geben mit einem eindeutigen Namen des Nodes (welcher durch „Node + ID“ beschrieben wird), weiterhin muss ein String für den Standort vorhanden sein. Da dieser nicht vorinitialisiert werden kann, muss dieser von Hand in der Software vergeben werden.

Beide Informationen können später für das Topic im ActiveMQ benutzt werden - sinnvoller erscheint jedoch der Name des Nodes. Dadurch ist gewährleistet, dass es nicht zu Verwechselungen mit anderen Sensoren kommt - schließlich ist eine Vielzahl von Sensoren in der Wohnung verbaut.

4.5 Schnittstellen und Datenorganisation

4.5.1 interne Datenverwaltung

Zur einfachen Verwaltung der Daten existiert ein Speicher, in dem sämtliche Informationen über die Sensoren gespeichert werden. Anforderung [H2] (einfache Erweiterbarkeit) schränkt zwar nichts ein, jedoch vereinfacht ein zentraler Datenspeicher die Umsetzung der Anforderung. Die Software wird mit Java programmiert werden, sodass sehr einfach mit Referenzen gearbeitet werden kann. Dies hilft, Inkonsistenzen zu vermeiden.

Um diesen Datenspeicher zu ermöglichen, muss eine eigene Klasse her, diese könnte wie in Listing 4.1 aussehen. Jeder Knoten beinhaltet die gleichen Informationen, somit kann später einfach über die vorhandenen XBee’s iteriert werden um die entsprechenden Informationen zu erhalten.

```
1  public class XBee {
2
3      private int ID;                  // ID des Nodes
4      private String name;             // Name des Nodes
5      private String standort;         // Standort des Nodes
6      private String sensortyp;        // Sensortyp ist am Arduino
7      private int messwert_analog;     // Messwert des Analogeinganges
8      private int messwert_digital;    // Messwert des Digitaleinganges
9
10
11     //konstruktor
```

```
12      public XBee(int ID){
13          this.ID = ID;
14          name = "Node " + ID;
15      }
16  // es fehlen sämtliche get/set-Methoden
17  }
```

Listing 4.1: Beispielklasse eines XBee

4.5.2 Sensor -> Koordinator

Die Anforderung [W5] besagt, dass die Protokolle einfach gehalten werden sollen. Somit wurde für die Übertragung vom Sensor zum Koordinator menschenlesbarer Text verwendet und keine kodierten Pseudo-Protokolle. Dies hat zwar den Nachteil, dass ein gewisser Overhead entsteht, beim Debuggen bzw. bei der Prüfung in der Software ist dieser Ansatz jedoch hilfreich.

Der Sensor kann 3 verschiedene Nachrichten an den Koordinator schicken:

Anmeldung

Nachdem der Arduino betriebsbereit ist, sendet er den folgenden String an den Koordinator:
Node 7 meldet sich in Ihrem Sprechfunkverkehrskreis an, betriebsbereit!
Dieser Standardsatz stammt aus dem BOS-Funk (Behörden und Organisationen mit Sicherheitsaufgaben) und wird von jeder Funkstation benutzt, um der Leitstelle die eigene Station zu registrieren. Somit weiß die Leitstelle, wer per Funk erreichbar ist.

In diesem Satz wird nur die ID von dem Arduino dynamisch eingefügt, der Rest ist rein statisch. Deswegen kann es auch verwendet werden, um abzuprüfen, ob der String komplett übertragen wurde.

Aktualisierung

Bei einer Aktualisierung (sowohl selbst-initiiert als auch angefordert von der Software) sendet der Arduino:
Node 7 sendet Aktualisierung: 0 0 abgeschlossen.

Hierbei fügt der Arduino seine Node-ID ein sowie die beiden Sensorwerte. Der erste Sensorwert steht dabei für den analogen Eingang und kann Werte von 0 bis 1023 repräsentieren, während die zweite Ziffer für den digitalen Eingang steht und nur 0 oder 1 werden kann.

Bestätigung

Sollte ein Ausgang am Arduino gesetzt werden, wird ein entsprechender Befehl zum Arduino geschickt. Da man nicht weiß, ob die Nachricht angekommen ist, antwortet der Arduino mit:
Node 7 hat Port 13 auf 1 gesetzt.
Dabei antwortet der sendende Arduino mit seiner internen ID (nicht die ID, die in der Nachricht stand!), dem empfangenen Port und dem Zustand. Die Nachricht wird erst versendet, wenn der Arduino den Port bereits modifiziert hat. Anhand dieser Nachricht kann festgestellt werden, ob eine Änderung am Gerät komplett angekommen ist.

4.5.3 Koordinator -> Sensor

Um die Sensoren zu steuern, gibt es ein kleines Protokoll, welches die genaue Reihenfolge von Prüfzeichen, IDs und weiteren Einstellungen festlegt. Diese sind notwendig um zu prüfen, ob bei der Übertragung alle Zeichen angekommen sind bzw. ob das Paket überhaupt für den eigenen Knoten bestimmt ist.

Hintergrund ist, dass der Koordinator nur broadcasten kann, während die Sensoren direkt an den Koordinator senden können. Daraus ergibt sich das Problem, dass alle Knoten zeitgleich die selbe Nachricht vom Koordinator empfangen und sie so alle gleich verarbeiten können müssen.
***00007|13|1#**
Das ist das Schema, nach dem die Datenübertragung zwischen Koordinator und Sensor erfolgt. Jede Nachricht beginnt mit einem Sternchen, danach folgt 5-stellig die Ziel-ID. Diese ID muss mit führenden Nullen aufgefüllt sein um die Gesamtlänge der Nachricht von 12 Zeichen nicht zu verkürzen. Die Länge ist gewählt um die maximal mögliche Anzahl von etwa 65000 Arduinos verwalten zu können. Nach der ID folgt ein senkrechter Strich, danach folgt zweistellig (ebenfalls mit führender Null falls erforderlich) die Port-ID, welche geschaltet werden soll. Es folgt wiederum ein senkrechter Strich, nachdem der Zustand folgt, in den der Port versetzt werden soll (also 0 oder 1). Abschließend erfolgt eine Raute, welche anzeigt, dass die Nachricht verarbeitet wurde.

Das Design macht es möglich, mehrere Nachrichten direkt hintereinander zu hängen ohne darauf achten zu müssen, dass sie einzeln verschickt werden.

Um eine Aktualisierungsanforderung an den Sensor zu schicken, wird das gleiche Schema verwendet, jedoch mit einer kleinen Änderung:
***00007|99|0#**
Die Ziel-ID ist angegeben, es wird also nur von einem Arduino verarbeitet. Der Port 99 ist nicht existent (auch nicht beim größeren Mega), sodass dies als Steuerzeichen für die Aktualisierungsanforderung benutzt werden kann. Ob danach eine 0 oder 1 folgt ist egal, dies wird nicht abgeprüft. Dieses Bit ist nur vorhanden, damit die 12 Zeichen nicht verletzt werden.

Da es unter Umständen müßig ist, jeden Arduino einzeln zu adressieren, kann der Befehl auch abgeändert werden:
***00000|99|0#**
Die Ziel-ID ist 0 - somit wissen alle Arduinos, dass es sich um eine globale Aktualisierungsanforderung handelt und jeder Knoten, der diese Nachricht empfangen hat, schickt seine Sensorwerte raus.

4.5.4 Koordinator <-> Software

Durch die Architektur mit dem XBee Explorer wird eine serielle Schnittstelle direkt an dem Zielrechner emuliert. Somit enfällt auch die Kommunikation zwischen dem Koordinator und der Software, da die Software direkt über die serielle Schnittstelle mit den Sensorknoten verbunden ist. Der Koordinator ist also nur der „Umsetzer“ zwischen dem Funknetzwerk und dem Kabel, welches softwaretechnisch in der seriellen Schnittstelle endet.

Es wäre möglich, den Koordinator-XBee als eine Art Router zu konfigurieren. Hierzu müsste ein Arduino über ein Ethernet-Shield und über ein XBee-Modul verfügen, was technisch gesehen kein Problem ist.

Um jedoch Ereignisse zu pushen (d.h. ein Ereignis tritt auf und die Knoten senden aktiv eine Nachricht ab), wird dieser Ansatz äußerst aufwendig. Es ist ohne Weiteres möglich, eine HTML-Seite mit den entsprechenden Sensorwerten zu generieren. Diese wären mit Java auch problemlos parsen, doch dies ermöglicht eben keine Push-Events. Dies müsste mit TCP/IP-Sockets realisiert werden, was den Umfang sehr komplex machen würde.

4.5.5 Fremdanwendung -> Software

Bisher kann nur die Software die Ausgänge der Arduinos steuern. Dies ist um die Funktion zu beweisen in Ordnung, bringt jedoch keinen sinnvollen Nutzen. Andere Programme, welche auf das ActiveMQ zugreifen und über mehr Informationen verfügen und so die Wohnung steuern, müssen Zugriff auf die Ausgänge haben.

Dies ist schnell in Java über TCP/IP-Sockets realisiert, dazu wird der Port 12358 geöffnet. Hier kann nun das Fremdprogramm verbinden und mittels eines Output-Streams Strings übermitteln. Dabei muss die Notation wie in Abschnitt 4.5.3 eingehalten werden. Ein eigenes Protokoll ist jedoch schnell realisiert, da die größte Schwierigkeit die Übertragung ist, nicht die spätere Verarbeitung im Programm.

4.5.6 statische vs. dynamische Übertragung

Wie in Kapitel 4.4.2 schon gesagt, macht ein dynamischer Ansatz unter gewissen Voraussetzungen mehr Sinn, als ein statischer. Der statische Ansatz ist einfacher abzuprüfen, es werden einfacher und schneller Fehler erkannt und die Umsetzung funktioniert für einen vorher festgelegten Anwendungsbereich genauso gut wie ein dynamischer Ansatz.

Sollte wie bei der Konfiguration der Sensorports ein dynamischer Ansatz sinnvoll sein, kann dieser selbstverständlich auch implementiert werden. Es ist allerdings zu beachten, dass unter gewissen Umständen die Nachrichten durcheinander geraten oder Teile davon vor bzw. nach anderen Teilen kommen. Dies lässt sich auf das ungenügende Broadcast-Verhalten der XBee's zurückführen. Wird aber ein dynamischer Ansatz gewünscht, muss zwingend eine Prüfsumme und die Länge des Datenpaketes vorhanden sein. Der Nachteil von dynamischer Programmierung ist der erhebliche Mehraufwand, der betrieben werden muss, um die Nachrichten entsprechend verarbeiten zu können.

Ob nun ein statischer oder dynamischer Ansatz sinnvoller ist, hängt wieder einmal vom Anwendungsgebiet ab. Werden immer die gleichen Nachrichten verschickt und ist nicht absehbar, dass etwas geändert werden muss, so ist ein statischer Ansatz das Mittel der Wahl. Werden häufig verschieden lange Werte übertragen (z.B. 10 Analogmesswerte mit einer möglichen Länge von 0 bis 1000), so kann es bei einer überschaubaren Anzahl von Knoten Sinn (in Hinblick auf das Übertragungsvolumen) haben, nur die Werte mit dynamischer Länge - also ohne führende Nullen - zu übertragen.

5 Realisierung

Zusammenfassend lässt sich sagen, dass das Design umgesetzt wurde. Es gab lediglich ein paar Feinheiten, die anders als geplant umgesetzt wurden.

In diesem Kapitel wird auf die technische Realisierung eingegangen, es werden die Änderungen vom Design angesprochen und erläutert. Abschließend werden die Testfälle erläutert und bewertet sowie auf Probleme während des Projektes eingegangen.

5.1 Programmierung

5.1.1 Hardware

Unter Hardwareprogrammierung ist nicht die Konfiguration der XBee-Module zu verstehen, sondern die Firmware für die Arduino-Knoten. Diese wurde mit der Standard-Entwicklungsumgebung vom Arduino geschrieben in der Sprache Processing. Es handelt sich dabei jedoch bei der Arduino-IDE nicht um ein „echtes“ Processing, sondern nur einen kleinen Teil. Dieser Teil wird von der Arduino-IDE ohne Erweiterungen interpretiert und umgesetzt. Erweiterungen sind jedoch mittels diverser Libraries möglich - bei der Entwicklung der Software wurde auf keine zusätzlichen Libraries zurückgegriffen.

Ein Arduino-Programm in Processing basiert auf den beiden Methoden setup() und loop(). Ein kleines Beispielprogramm, welches „Hello World“ über Funk versendet, ist in Listing 5.1 zu finden.

In der Methode setup() werden sämtliche Konfigurationen für den Arduino vorgenommen, z.B. für die Portmodi oder die serielle Schnittstelle. Die Methode wird beim Start des Arduino durchlaufen.

Die loop()-Methode ist wie der Name schon sagt die Methode, die immer wieder durchlaufen wird, vergleichbar mit einer while(1)-Schleife. Sie wird unmittelbar nach der setup() aufgerufen und beinhaltet den eigentlichen Programmcode.

In der Firmware wurde eine State-Machine implementiert, dies liegt an den in 4.5.3 spezifizierten Nachrichten. Mit der State-Machine wird die Überprüfung der eingehenden Nachrichten einfacher und übersichtlicher, vor allem aber wird die Software robuster gegenüber Fehlern, die während der Übertragung passieren. Es können z.B. direkt aufeinanderfolgende Nachrichten erkannt und bearbeitet werden, dies wäre in anderen Ansätzen nur schwer realisierbar durch die unbekannte Länge.

In Bild 5.1 ist ein Ablaufdiagramm der Software. Es ist in 4 States aufgeteilt, welche die gesamte Logik des Programms beinhalten. Dieser sehr flache Programmierstil ist wegen der Einfachheit gewählt worden, falls später etwas an der Software geändert werden muss.

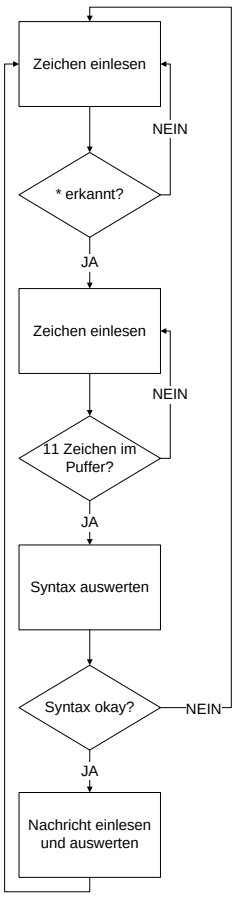

Abbildung 5.1: Ablaufdiagramm Arduino-Software

```
1  void setup()
2  {
3    // start serial port at 9600 bps:
4    Serial.begin(9600);
5  }
6
7  void loop()
8  {
9    Serial.println("Hello World!");
10   delay(2000);
11 }
```

Listing 5.1: einfaches Processing-Beispiel

5.1.2 Auslesen der ID bei XBee-Modulen

Um die ID aus den XBee-Modulen auslesen zu können, wurde der Code von [Meißner, 2010] leicht abgewandelt, da z.B. nicht das „OK“ abgefangen wurde, welches das Modul zurückgibt, sobald man in den Command-Mode geht. Der Code 5.2 wird in der Setup-Routine des Arduino's ausgeführt. Diese Routine wird bei jedem Start des Arduino ausgeführt, bevor das Programm in der loop()-Methode läuft. Trotzdem ist diese Methode, die ID auszulesen unsicher: sobald der Arduino einen Kaltstart macht, kann die ID aus einem nicht zu erklärenden Grund nicht ausgelesen werden. Damit die ID richtig ausgelesen werden kann, muss nach dem Starten des Arduinos einmalig der Reset-Knopf gedrückt werden. Danach wird die ID korrekt ausgelesen und kann für das Netzwerk verwendet werden.

Um die ID auslesen zu können, muss eine bestimmte Reihenfolge der Befehle erfolgen, wie sie in 5.2 ersichtlich ist. Das Modul verfügt über eine Guard-Time - die ist dafür gedacht, dass man nicht durch ein Versehen in den Command-Modus gelangt. Es bedingt, dass keine Aktivität eine Sekunde vor und nach dem Befehl erfolgt. Die Guard-Time kann beliebig gewählt werden und wird im XBee-Modul einprogrammiert. Außerdem gibt es noch eine Zeitspanne, bis der Command-Modus automatisch verlassen wird. Dies tritt ein, wenn die definierte Zeit kein AT-Command mehr erfolgt ist.

```
1    delay(1000);              // delay guard time
2    Serial.print("+++");      // enter command mode
3    delay(1000);              // delay guard time
4    Serial.println("ATNI");   // returns node identifier
5    delay(100);               // wait for answear
```

```
6   test = Serial.available();
7
8   char idString[Serial.available()]; // malloc
9   char idString2 = Serial.read();    // remove "OK\r"
10  idString2 = Serial.read();
11  idString2 = Serial.read();
12
13  for(i = 0; Serial.available() > 0; i++){
14      idString[i] = char(Serial.read());
15  }                                  // read ID
16
17  Serial.println("ATCN"); // exit command mode
18
19  myID = atoi(idString);    // cast string to int
```

Listing 5.2: Auslesen der XBee-ID

5.1.3 Software

Das Konfigurationsprogramm übernimmt einen Großteil der Koordination und Konfiguration der Knoten, aber auch das Aufgabengebiet der Datenaufbereitung. Die Software wurde in Java geschrieben und beinhaltet sowohl die Anbindung an das Sensornetzwerk als auch an das Message Broker System ActiveMQ.

interner Aufbau

Wichtig bei der Programmierung war, dass sämtliche Funktionen gekapselt sind, sodass man sie später aus der GUI ohne Probleme asynchron per eigenem Thread aufrufen kann. Die gesamte Funktionalität befindet sich in einer Klasse. Wird das Programm gestartet, so muss lediglich ein Objekt der Klasse erstellt werden und der gewünschte COM-Port gewählt werden.

Die Klasse erstellt dann alle benötigten Datenstrukturen sowie ein Objekt der seriellen Schnittstelle. Es sind sämtliche get und set-Methoden vorhanden, sodass externe Anwendungen nur auf den vorgesehenen Bereich zugreifen können.

Die Klasse besteht aber nicht nur aus Methoden, sondern beinhaltet auch einen Thread, welcher automatisch über den Konstruktor bei Erstellen eines Objektes erstellt wird. Dieser

Thread pollt alle 100 Millisekunden die Queue der seriellen Schnittstelle. Sind Daten vorhanden, werden sie übernommen und der Datenverarbeitungs-Methode übergeben.

Diese Methode erkennt und verarbeitet den String, indem sie z.B. bei Anmeldung eines neuen Moduls dieses erfasst und in den Datenspeicher legt. Ist der Nachrichtentyp eine Aktualisierung eines Sensorwertes, wird dieser Sensorwert ebenfalls im Datenspeicher abgelegt, jedoch auch direkt an das ActiveMQ übertragen. Dieser Mechanismus ist jedoch abschaltbar um eine Funktion der Software auch ohne aktiven ActiveMQ-Server zu gewährleisten.

serielle Schnittstelle

Man mag vermuten, dass eine Sprache wie Java ohne Probleme eine serielle Schnittstelle unterstützt. Leider ist dies eine komplette Fehlannahme, denn das javax.comm-Paket wurde bereits im Jahr 2002 (!) nicht mehr gepflegt und weiterentwickelt. Durch die Übernahme von Sun durch Oracle ist dieses Paket auch nicht mehr online verfügbar, sodass man nur noch auf das kompatible Paket von RXTX.org ausweichen kann. Dieses Paket bildet das gesamte javax.comm-Paket nach und wird auch heute noch weiterentwickelt.

Leider hat RXTX große Probleme was die Stabilität der Verbindung betrifft, auch sind viele Bugs enthalten, sodass eine effektive Nutzung quasi ausgeschlossen ist. Aus diesem Grund wurde eine alte Version der javax.comm verwendet, da diese trotz 8 Jahren ohne Wartung immer noch stabil und zuverlässig läuft.

Ein Codebeispiel aus dem Internet verfügte über je einen Thread zum Senden und Empfangen, dies wurde als nicht sinnvoll erachtet und anders implementiert. Dass die serielle Schnittstelle für das Empfangen über einen Thread verfügen muss, steht außer Frage. Jedoch kann man für das Senden eine normale Methode verwenden, da die Datenmenge nie so groß wird, dass das Blockieren während der Ausführung kritisch würde.

Der Empfangsthread legt die empfangenen Strings in eine LinkedList, welche sicherstellt, dass Liste transitiv ist - also alle Elemente in der selben Reihenfolge abgearbeitet werden können, wie sie empfangen wurden.

GUI

In Bild 5.2 ist ein Screenshot der Konfigurationssoftware zu sehen. Über diese Software werden die XBee-Module verwaltet, außerdem ist hierüber die exemplarische Steuerung der Ausgänge möglich.

Es werden die aktuellen Sensorwerte (Aktualisierungsrate: 500 Millisekunden) angezeigt, außerdem ist ein Standort des Moduls sowie eine Beschreibung der beiden Eingänge möglich. Im ActiveMQ ist der analoge Sensor zu finden als .Sensor1, der digitale als .Sensor2.

Die GUI wurde mit NetBeans geschrieben und läuft als eigenständiger Thread. Die Methoden der eigentlichen Logik werden asynchron aufgerufen als eigene Threads - so ist gewährleistet, dass bei einer Verzögerung die GUI nicht einfriert und weiter aktualisiert wird.

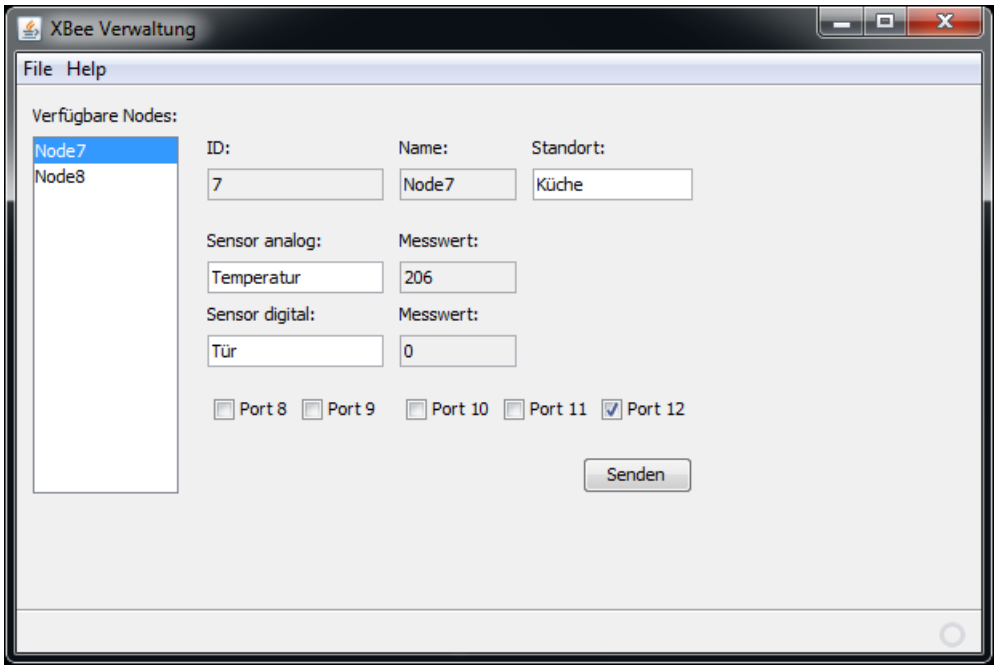

Abbildung 5.2: Screenshot Software: XBee Verwaltung

5.2 Testfälle

Im Vorfeld und während dieser Arbeit wurden sehr viele Tests zu dem Schwerpunkt XBee absolviert. Dabei ging es hauptsächlich darum, die Zuverlässigkeit des Protokolls zu erforschen und mögliche Grenzen zu erkennen. Dieses wurde mit einfachen Methoden (Echoserver als Client, etc.) stunden- bzw. tagelang getestet. Es wurde hierbei kein Wert auf die Einhaltung wissenschaftlicher Methoden, sondern nur auf repräsentative Ergebnisse gelegt.

Am Ende des Kapitels wird die Funktion der Software anhand von Anwendungsbeispielen erklärt.

5.2.1 Routingverhalten

Es gibt zwei Fälle, die man getrennt betrachten muss beim Routingverhalten der XBee's. Einmal den Fall, dass alle Knoten in Reichweite sind, und einmal den Fall, dass einige oder alle Knoten am Ende der Reichweite angekommen sind.

Im ersten Fall ist das Routing sehr zuverlässig, auch wenn Broadcasts ca. 4 Sekunden brauchen, um korrekt und zeitgleich bei allen Knoten anzukommen (abhängig von Signalstärke und Knotenanzahl). Werden schneller Nachrichten geschickt als die genannten 4 Sekunden, so gibt es eine Pause in der Übertragung von ca. 1-2 Sekunden. Danach werden die „angestauten" Pakete auf einmal übertragen. Es gehen aber keine Pakete verloren, sie werden nur mit einer Verzögerung übertragen.

Im zweiten Fall, also wenn der Knoten sich am Ende der Reichweite befindet, kann es zu einem packet-loss kommen. Es ist nicht vorhersagbar, wie und wann dieser Fall eintritt - es gehen sporadisch Pakete verloren. Das Phänomen mit dem Aufstauen der Pakete ist hier allerdings auch zu beobachten, wobei hier ausdrücklich Paketverluste auftreten können und nicht sichergestellt ist, dass alle Pakete empfangen werden. Je weiter das Modul am Ende der Reichweite ist, desto höher wird der Paketverlust. Interessant ist hierbei, dass diese Zone sehr abhängig von dem Wetter ist: so war bei schönem Wetter eine Verbindung quer über das Grundstück möglich mit wenigen Paketverlusten, während bei erhöhter Nahfelddämpfung (Regen und hohe Luftfeuchtigkeit) der Knoten überhaupt nicht mehr erreichbar war.

Man kann zusammenfassen, dass das Routing innerhalb des XBee-Netzes sehr zuverlässig funktioniert, sofern die Knoten nicht zu weit entfernt sind. Bei Broadcasts jedoch bringt man das Netz in große Schwierigkeiten, da das XBee-Protokoll grundsätzlich für Punkt-zu-Punkt-Verbindungen entwickelt wurde. Man darf auch nicht vergessen, dass das XBee-Protokoll recht einfach gehalten ist und nicht mit einem Industriestandard wie WLAN verglichen werden kann.

5.2.2 Belastungstest

Um zu erfahren, ob zeitgleich gesendete Nachrichten zu einer Kollision führen und ob das System auch nach längerer Zeit noch stabil läuft, wurden folgende zwei Tests durchgeführt:

Kollisionserkennung

Versuchsaufbau: bei 8 Knoten wurde der digitale Eingang an einen Rechtecksignalgenerator angeschlossen. Dieser wird mit einer Amplitude von 5 Volt betrieben und einem Offset von 2,5 Volt. Daraus resultiert, dass das Rechteck zwischen 0 und 5 Volt liegt und so einem Schalter gleicht.

Der gewünschte Effekt ist, dass alle 8 Arduinos zur selben Zeit eine Änderung am Eingang erkennen und so eine Statusänderung losschicken. So soll die Kollision provoziert werden.

Frequenz [Hz]	Zeit [min]	Fehler
0.1	600	0
0.5	600	0
1	300	0
4	300	0*
100	300	0*

* bei dieser Einstellung sind nicht alle erwarteten Antworten eingegangen, weil das XBee-Netzwerk scheinbar nicht schnell genug war. Es wurde auch nicht überprüft, ob Daten vom Netzwerk verworfen wurden - sondern lediglich auf Kollisionen zwischen Nachrichten.

Langzeittest

Bei diesem Test wurden 8 Module im Haus und auf dem Grundstück verteilt. Da bei diesem Szenario eine externe Triggerung mit viel Aufwand verbunden wäre, wurde für diesen Test die Ansteuerung der Ausgänge verwendet. Ein Javaprogramm schickt an alle Knoten nacheinander die Anforderung, Ausgang 13 zu setzen, macht dann 4 Sekunden Pause. Danach erhalten die Knoten den Befehl, Ausgang 13 auszuschalten. Nach weiteren 4 Sekunden Pause beginnt die Schleife erneut.

Da die Knoten auf die Anforderung antworten mit ihrer ID und dem Status, kann diese Antwort abgefragt und als Auswertung benutzt werden.

Zeit [h]	Fehler
1	0
12	28*
46	4.591**

* die Fehler wurden von unterschiedlichen Knoten verursacht, es ist also mit einer gewissen Anzahl von Fehlern auch bei optimalen Bedingungen zu rechnen.

** von diesen 4.591 Fehlern wurden 4.467 von einem Knoten verursacht - dieser war auch am weitesten entfernt. Es ist davon auszugehen, dass durch äußere Einflüsse (Störstrahlung, Dämpfung) der Knoten nicht mehr optimal arbeitet. Die verbleibenden 124 Fehler entsprechen in etwa dem selben Prozentsatz wie beim 12-Stunden-Test mit 0,3% Fehler. Rechnet man die Fehler des einzelnen Knoten mit ein, liegt man bei 11%. Dies wäre eine Größenordnung, die nicht mehr tolerierbar wäre. Da es sich aber bei dem Fehler nicht um einen Hardware- oder Softwarefehler handelte, sondern der Knoten einfach nur an der physikalischen Grenze arbeitete (die in dem Living Place nie erreicht werden würde), kann man diesen Fehler als gegenstandslos betrachten.

Bewertung

Lässt man den fehlerhaften Knoten aus dem zweiten Test aus der Statistik, kann man sagen, dass das XBee-Netzwerk sehr zuverlässig arbeitet mit einem Fehler von ca. 0,3%. Um diesen verbleibenden Fehler zu erkennen und zu eliminieren, wird man sehr viel Aufwand betreiben müssen, sodass man diesen Fehler auf der Hardwareseite in Kauf nehmen kann und ihn versucht, softwareseitig zu lösen.

5.2.3 Anwendungsfälle

In Kapitel 4.3.3 wurde schon exemplarisch der Ablauf vom Einfügen eines neuen Knoten erklärt. Dieses Szenario wird nun vom Programmieren der neuen Hardware bis hin zur Konfiguration der Sensoren durchgespielt.

Die Reihenfolge der Programmierung ist vertauschbar, in diesem Beispiel fangen wir mit dem XBee-Modul an. Die Software X-CTU (Bild 4.2) wird dabei zum Flashen der Module verwendet. Dazu muss der COM-Port des XBee-Explorers (auf dem das XBee-Modul steckt) ausgewählt, und die richtige Übertragungsgeschwindigkeit eingestellt werden. Dies ist in dieser Arbeit immer 115.200 Baud. Wichtig ist die Konfiguration des Function Set (s. Kapitel 4.3.2), welches als Router/Enddevice AT in der Version 1247 eingestellt sein muss. Danach wird die PAN-ID auf 42 (das PAN, welches in dieser Arbeit verwendet wurde) und die Node-ID auf einen freien Wert gestellt wird (in diesem Beispiel: 1337). Nach einem „write“ wird das

Modul überschrieben, welches X-CTU mit einem „done.“ quittiert. Das XBee-Modul ist nun bereit für den Einsatz.

Nun kümmern wir uns um den Arduino. Nur der reine Duemilanove (ohne XBee-Shield!) wird an einen freien USB-Port des Rechners gesteckt, welcher darauf einen neuen COM-Port erkennt. In der Software (Bild: 5.3) lädt man nun den Sketch für den Router, wählt über „tools -> serial port“ den entsprechenden COM-Port aus und klickt auf „Upload“. Der Screenshot (5.3) zeigt den Zustand nach dem Flashen, das Programm quittiert mit einem „Done uploading.“ den Erfolg.

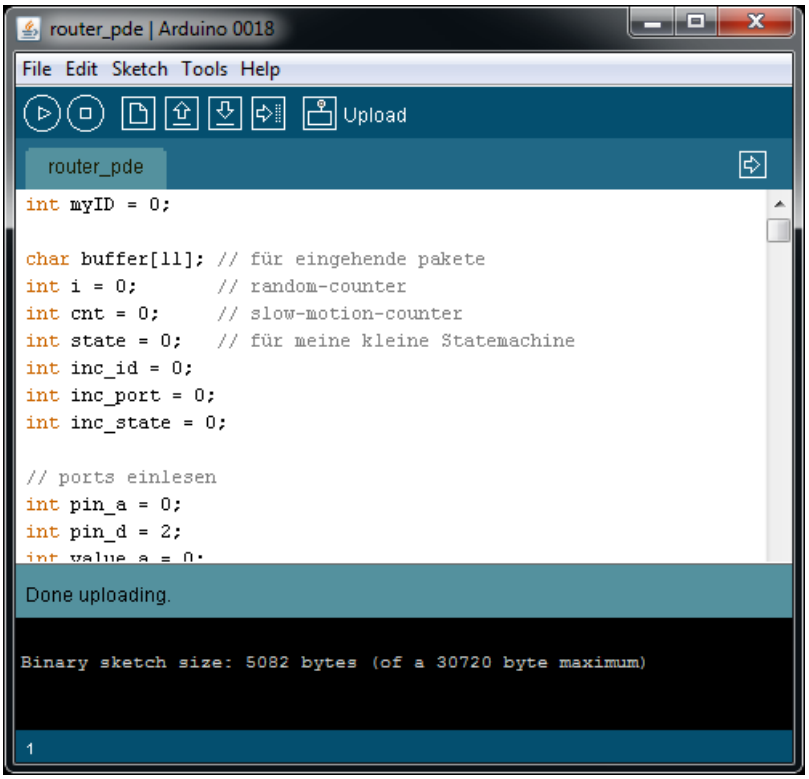

Abbildung 5.3: Arduino-IDE 0018

Nachdem sowohl das XBee-Modul, als auch der Arduino programmiert wurde, werden beide Komponenten mittels XBee-Shield verbunden. Dabei empfiehlt es sich, Pull-Down-Widerstände zu verwenden, da sonst die Werte zu stark schwanken und eine Auswertung faktisch unmöglich ist. In Bild 5.5 ist ein Testaufbau mit Widerständen (je 10kΩ) zu sehen. Auf den langen Beinen des Widerstandes wird der Sensor bzw. Schalter angeschlossen und mit Vcc verbunden.

Nachdem die Hardware nun einsatzbereit ist, schließen wir den Arduino an ein Netzteil an

und drücken nach 2-3 Sekunden den Reset-Knopf. Dadurch wird es dem Ardunio erst ermöglicht, die ID des XBee-Modules auszulesen.

Die Software aktualisiert sich selbstständig:

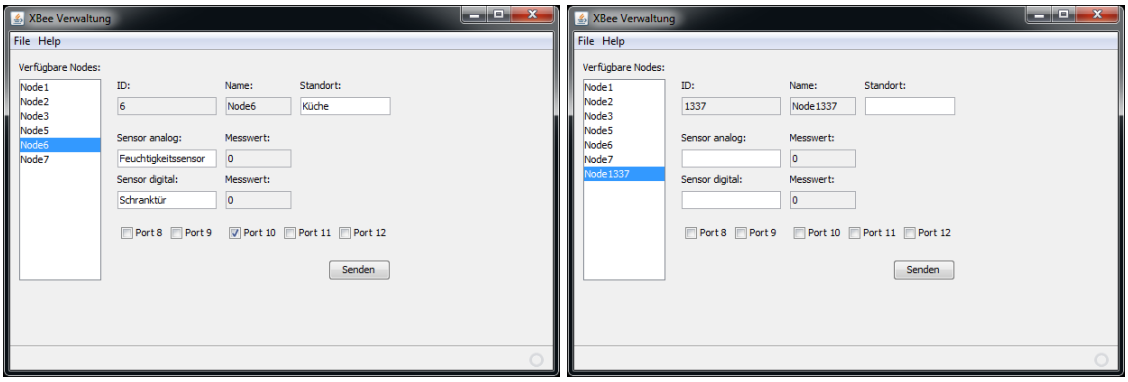

Abbildung 5.4: Aktualisierungsvorgang der Software

Wenn der neue Knoten (ID: 1337) in der Liste ausgewählt wird, ist es möglich, ihn zu konfigurieren. Die Textfelder können ausgefüllt werden mit Zusatzinformationen, z.B. dem Standort oder dem Sensortyp. Dies dient einzig und allein der besseren Zuordnung in der Software, diese Informationen werden nicht an das ActiveMQ übertragen. Wenn Änderungen vorgenommen werden, muss zum Übernehmen der Daten der „Senden“-Knopf gedrückt werden. Über diesen lassen sich auch exemplarisch die Ausgänge steuern.

Nun können Sensoren angeschlossen werden, die Werte werden automatisch ins ActiveMQ übertragen. Dabei findet keine Überprüfung statt, ob es Subscriber gibt - jede Änderung wird übertragen.

5.3 Probleme und Schwierigkeiten

Während der Implementierung der Software gab es selbstverständlich auch Probleme, welche in diesem Kapitel kurz angesprochen werden.

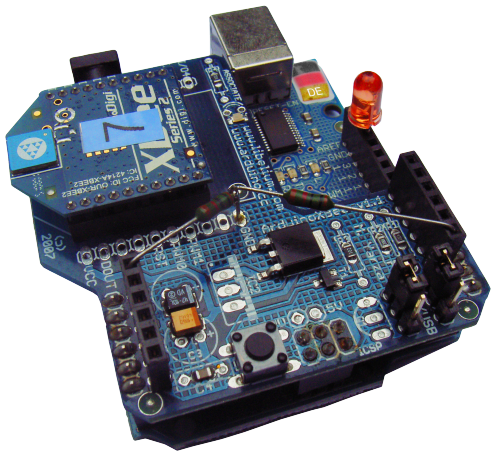

Abbildung 5.5: Sensorknoten mit Pull-Down-Widerständen

5.3.1 Hardware

Arduino

Bei den Arduino's tritt das Problem auf, dass nach einem Kaltstart die ID nicht ausgelesen werden kann (bereits in Kapitel 5.1.2 beschrieben). Für dieses Problem konnte keine Lösung außer einem Reset gefunden werden, dies scheint ein generelles Problem zu sein.

Wird ein USB-Gerät an den PC angeschlossen, starten die Arduinos durch und haben selbigen Fehler mit der ID. Der Reset ist ein Design-Bug (oder -Feature), da wie bei der seriellen Schnittstelle ein Pin auf dem Reset des Mikrocontrollers liegt. Die falsche ID lässt sich nicht erklären. Wenn man die serielle Schnittstelle mit der Arduino-IDE debuggt, werden ein paar kryptische und nicht druckbare Zeichen angezeigt. Der Grund für dieses Verhalten ist aller Wahrscheinlichkeit nach der FTDI-Treiber, der umgeschrieben werden müsste, um dieses Problem zu beheben.

Beide Probleme könnte man mit dem in 4.3.5 beschriebenen Software-Reset über den Watchdog des Mikrocontrollers möglicherweise beseitigen.

Ein weiteres, gravierendes Problem sind die Schwankungen der Eingänge. Um hier ein verwertbares Ergebnis zu erzielen, mussten Pull-Down-Widerstände auf den Boards verbaut werden. Ohne diese Widerstände schwanken die Werte vom vollen Wertebereich von 0 bis 1023 - und dies ohne erkennbaren Grund.

XBee

Beim Flashen der XBee-Module tritt sporadisch das Problem auf, dass die Module kein Firmwareupdate annehmen. Es wird mittendrin abgebrochen oder schlägt fehl. Eine Lösung für dieses Problem ist die massive Wiederholung des Flashvorganges - ein Modul musste im Rahmen dieser Bachelorarbeit ca. 20 Mal geflasht werden, bevor es die neue Firmware übernahm.

5.3.2 Software

Bei der seriellen Schnittstelle läuft ein Thread, welcher die serielle Schnittstelle dauernd abfragt. Dieser Thread verursachte so eine CPU-Last von 100% - um dieses Problem zu „lösen“, wurde ein Sleep() eingebaut von 100 Millisekunden. Dadurch sinkt die CPU-Nutzung auf ein normales Maß und es konnten keine Beeinträchtigungen durch einen Pufferüberlauf festgestellt werden.

5.4 Evaluation

Um die Realisierung bewerten zu können, wird die Software mit den Anforderungen in Kapitel 3.1.1 verglichen und bewertet. Die Funktionen der Software werden kurz angesprochen und zusammengefasst, sodass ein umfassendes Bild der in diesem Kapitel entstandenen Software entsteht.

[H1] Das Sensornetzwerk soll drahtlos sein. Durch die Verwendung des ZigBee-Netzwerkes verfügt das Sensornetzwerk über eine drahtlose Verbindung unter den Knoten. Sie funktionieren als Router, d.h. jeder einzelne Sensorknoten erweiter die Reichweite des gesamten Netzes. Lediglich der Koordinator verfügt über eine feste Verbindung zum Server - in diesem Falle ist der der Access-Point und ermöglicht der Software, auf das Sensornetzwerk zuzugreifen.

[H2] Das Sensornetzwerk soll selbstkonfigurierend, also einfach erweiterbar sein. Ebenfalls durch die Verwendung des ZigBee-Protokolls, ist die Funkkomponente selbstorganisierend und bis zu über 65.000 Knoten erweiterbar. Durch die Verwendung von Arduino-Standardhardware ist die Erweiterbarkeit gewährleistet, da die neuen Sensorknoten nur eine

spezielle Firmware bekommen müssen und somit ohne Probleme funktionieren. Die Software ist eine grafische Anwendung und übernimmt die Verwaltung der Knoten.

[H3] Das Sensornetzwerk soll auf Standardhardware basieren. Wie schon angesprochen, wird in dieser Arbeit ausschließlich Standardhardware der Firma tinker.it verwendet. Dabei handelt es sich um den Arduino Duemilanove, welcher mit seiner Bauform die Verwendung von Zusatzmodulen ermöglicht. Über ein solches Zusatzmodul ist es dem Mikroprozessor auch möglich, die Funkanwendung ZigBee zu betreiben. Lediglich zu Testzwecken wurde die Hardware mit ein paar Widerständen bestückt, um die Ergebnisse verwertbar zu machen.

[H4] Die Knoten sollen analoge und digitale Werte einlesen können. Der Arduino Duemilanove besitzt 5 analoge Ports, welche zum Einlesen verwendet werden können. Weiterhin besitzt er 13 digitale Ports, von denen aber nur 11 nutzbar sind. Dies liegt an der Kommunikation mit dem XBee-Modul, welches eine serielle Schnittstelle, also 2 digitale Pins benötigt. Im Prototyp dieser Arbeit wurde ein statischer Ansatz implementiert, welcher den ersten Analogport und den dritten Digitalport ausliest. Ein dynamischer Ansatz wurde in dem Design diskutiert und stellt so eine mögliche Erweiterung dieser Arbeit dar.

[H5] Die Knoten sollen über fernsteuerbare, digitale Ausgänge verfügen. Um Aktoren zu betreiben, sah es die Anforderung vor, dass die digitalen Ausgänge steuerbar sein sollen. Dies ist realisiert worden in der Form, dass die digitalen Ausgänge 8 bis 13 mittels der entwickelten Software testweise geschaltet werden können. Außerdem bietet eine Schnittstelle für externe Programme die Möglichkeit, die Ausgänge zu steuern. Dies ist nötig, da Fremdanwendungen mittels ActiveMQ an die Sensorwerte kommen, aber nicht über das ActiveMQ die Aktoren schalten können. Somit bietet diese Schnittstelle die Möglichkeit einer Rückkopplung. Die Ausgänge können in diesem Stand der Arbeit nur auf die Werte 0 und 1 gesetzt werden, um z.B. ein Analogsignal zu erzeugen, muss Pulsweitenmodulation (PWM) verwendet werden - dies wäre auch eine mögliche Fortführung dieser Arbeit.

[H6] Die Anwendung soll die Sensorwerte im ActiveMQ bereitstellen. Die Sensorwerte sollen im ActiveMQ abgebildet werden, damit Fremdanwendungen Zugriff auf diese Daten erhalten. Dies ist Kernthema dieser Arbeit: ein Sensor soll an einem beliebigen Platz installiert werden können, damit andere Anwendungen Zugriff auf diese Sensordaten bekommen. Bei jeder Aktualisierung, die die Sensorknoten senden, wird automatisch aus der entwickelten Software ein Update an das ActiveMQ geschickt. Somit bekommen alle Subscriber des jeweiligen Sensors eine Nachricht, sobald der Sensorknoten ein Update geschickt hat. Dieser Mechanismus ist jedoch abschaltbar, um ein Testen ohne einen ActiveMQ-Server zu ermöglichen.

[H7] Die Anwendung soll eine Schnittstelle für andere Anwendungen bereitstellen, um die Digitalausgänge der Knoten zu steuern. Wie im Punkt vorher schon genannt, steht die Bereitstellung der Sensordaten an andere Anwendungen im Mittelpunkt. Damit diesen Anwendungen die Möglichkeit der Rückkopplung gegeben wird, wurde in diesem Projekt wie in [H5] schon genannt eine Schnittstelle definiert, auf die Fremdanwendungen zugreifen - und somit die Ausgänge schalten können. Um die Ausgänge zu schalten, braucht das Sensornetzwerk im schlechtesten Fall (große Reichweite, viele Daten, etc.) bis zu 4 Sekunden um die Änderung zu übernehmen. In der Regel geht dies allerdings „sofort“, d.h. ohne merkliche Verzögerung.

[W1] Das Sensornetzwerk soll im Fehlerfall weiter funktionieren. Sofern ein Fehler auftritt in Form eines Ausfalls von einem Sensorknoten oder wenn ein Knoten falsche oder unvollständige Werte schickt, funktioniert das Netzwerk weiter. Falsche bzw. deformierte Nachrichten werden verworfen, sodass keine falschen Werte im ActiveMQ ankommen werden. Wenn ein Knoten ausfällt kann dies in der implementierten Software nicht erkannt werden, da die Sensorknoten bisher nur dem sende-bei-Änderung-Paradigma folgen. Es ist zwar möglich, auf Anforderung den aktuellen Sensorwert zu erhalten, jedoch wird nicht überprüft, ob der Knoten antwortet.

[W2] Die Knoten sollen drahtlos neu programmiert werden können. Damit die Firmware der Sensorknoten übernommen werden kann, ist eine Änderung der Firmware über Funk ratsam. Diese Möglichkeit wurde schon in anderen Arbeiten verwendet, sodass die Möglichkeit an sich besteht und getestet ist. Trotzdem wurde in dieser Arbeit dieses Feature noch nicht implementiert, da Grundlegende Probleme, wie das „falsche“ Starten des Arduinos viel Zeit bei der Fehlersuche gekostet haben. Somit wurde der Mechanismus für ein drahtloses Update erstmal außen vor gelassen, da Stefan Meißner durch seine Masterarbeit und das Tool RemoteAVR samt Code für die Knoten alles bereitstellt und es kein großer Aufwand mehr sein sollte, dieses Feature zu implementieren.

[W3] Die Anwendung soll eine einfach zu verwaltende, grafische Oberfläche bereitstellen. Durch die Anwendung von Java als Programmiersprache ist auch eine grafische Oberfläche entstanden, welche die Verwaltung der Sensoren übernimmt. Die Software zeigt die aktuellen Sensorwerte in einem Abstand von 500 ms an, weiter ist es möglich, die digitalen Ausgänge von 8 bis 13 direkt aus der Software zu steuern. Es können Standort und Namen für die einzelnen Sensoren vergeben werden, diese Informationen sind jedoch nur in der Software selber abrufbar und werden nicht mit in das ActiveMQ übermittelt. Man könnte die Topic’s des ActiveMQ natürlich auch mit den Namen der Sensoren anlegen, jedoch liegt

die Vermutung nahe, dass es bei wechselnden Sensoren schnell zu Verwirrungen bei der Zuordnung kommen wird.

[W4] Die Kosten sollen beachtet werden. Die in dieser Arbeit benutzte Hardware ist mit ca. 70 EUR pro Knoten sehr teuer. Wenn man aber bedenkt, dass keinerlei Entwicklungsarbeit für die Hardware notwendig war, sind diese Kosten für ein Projekt mit niedriger Stückzahl durchaus vertretbar. Würde man ein eigenes Layout verwenden, werden Kosten pro Stück deutlich höher sein und erst mit einer hohen Stückzahl auf ein wirtschaftliches Maß sinken.

[W5] Die verwendeten Protokolle sollen einfach gehalten werden. Damit die Erweiterbarkeit gewährleistet bleibt, wurde bei der Implementierung der Software auf einfache Protokolle zwischen den verschiedenen Punkten im Netzwerk geachtet. Sowohl die „interne“ Kommunikation im Sensornetz, als auch die Schnittstelle für externe Anwendungen haben eine sehr einfache Syntax. Somit ist sichergestellt, dass andere Programmierer sich nicht lange mit der Notation beschäftigen müssen und so die Erweiterbarkeit gewährleistet bleibt.

6 Resümee

6.1 Zusammenfassung & Fazit

Rückblickend kann gesagt werden, dass aus dieser Arbeit ein funktionsfähiges und robustes Sensornetzwerk realisiert wurde. Es genügt den Anforderungen des Living Place Hamburg und ist durch seine Struktur einfach erweiterbar. Als Sensorknoten wurde die standardisierte Open-Source Plattform „Arduino Duemilanove“ verwendet, welche ein einfache Erweiterbarkeit gewährleistet und sich preislich attraktiv gestaltet. Als Funkanwendung wurde ZigBee auf Grund des schlanken Protokolls, der Robustheit und Erweiterbarkeit gewählt. Die Kommunikation und Steuerung des Sensornetzes erfolgt über einen PC mit einer USB-Schnittstelle. Ein Java-Programm ist für die Verwaltung zuständig und sorgt dafür, dass die Sensorwerte ins ActiveMQ geschrieben werden.

Die gesteckten Ziele in Form der Anforderungen wurden mit Ausnahme der weichen Anforderung [W2] (drahtloses Firmwareupdate) erreicht.

Da das System noch nicht im Produktivbetrieb im Living Place getestet wurde, kann an dieser Stelle noch keine Aussage darüber getroffen werden, ob das System den tatsächlichen Anforderungen gerecht wird. Aus Sicht der aus den Anforderungen in Kapitel 3.1.1 entstandenen Kriterien und Erwartungen, kann gesagt werden, dass das System im vollen Umfang funktioniert. Sämtliche Tests wurden ohne Fehler erfolgreich absolviert.

Es ist jedoch festzustellen, dass die Sprache Processing wirklich eine Sprache für Nicht-Informatiker zu sein scheint. Sie ist sehr leicht zu bedienen, doch wenn man das Potenzial von einem Mikrocontroller kennt, möchte man am liebsten alle Funktionen neu- oder umschreiben, da manche einfach sehr ineffizient sind, das Potenzial nicht ausschöpfen oder schlicht falsch oder fehlerhaft geschrieben sind.

6.2 Ausblick

Die hier konzipierte und realisierte Arbeit bietet viele Möglichkeiten für Erweiterungen sowie Fortführungen. Einen sehr interessanten Aspekt bietet der ATmega128RFA1 der Firma Atmel. Dies ist ein Mikrocontroller mit eingebauten ZigBee-Modul - das heißt, man muss nur noch eine kleine Antenne an den Mikrocontroller anschließen und braucht keine weitere, externe Hardware. Durch diese On-Chip-Lösung könnten die Sensorknoten noch deutlich kleiner gebaut werden, außerdem könnten sie länger per Batterie laufen, da keine zusätzliche Hardware nötig ist. Die Sleep-Modes könnten effektiver ausgenutzt werden und es wäre eine enorme Kostenersparnis.

Ein weiterer Aspekt für die Erhöhung der Reichweite wären Module mit erhöhter Sendeleistung oder eine bessere Antenne. Die XBee-Module erlauben bei hardwaretechnischer Veränderung den Anschluss einer externen Antenne. Dies hätte auch nicht das in Funkkreisen bekannte Krokodil-Problem (großer Mund - kleine Ohren, also viel Sendeleistung und wenig Empfangsempfindlichkeit) zur Folge, sondern würde effektiv die Reichweite erhöhen. Bei Tests kamen die Module mit 2 mW Sendeleistung deutlich weniger weit, als das Datenblatt angab. Dies ist in jedem Fall auch ein Punkt, der verbessert werden kann.

Aufbauend auf dieser Arbeit könnten ein Shield entstehen, welches bestimmte Standardsensoren für Temperatur neben dem XBee-Modul eingebaut hat. Damit würde die Verdrahtung entfallen und man hätte ein sauberes Layout und würde somit auch weniger Fehler durch Übergangswiderstände, falsche Sensorwerte, etc. erhalten.

Es gibt noch viele weitere Möglichkeiten, diese Arbeit zu erweitern oder fortzuführen - durch den Einsatz von Open-Source Hardware und einfachen Kommunikationsschnittstellen scheint das begrenzende Element die eigene Phantasie zu sein.

Literaturverzeichnis

[Digi International Inc. 2010] DIGI INTERNATIONAL INC.: *XBee & XBee-PRO ZB ZigBee PRO RF Modules*. 11001 Bren Road East, Minnetonka, MN 55343. 2010. – URL http://www.digi.com/products/wireless/zigbee-mesh/xbee-zb-module.jsp. – accessed on 02.08.2010

[Eriksson 2009] ERIKSSON, Joakim: *Detailed Simulation of Heterogeneous Wireless Sensor Networks*. 2009. – URL http://www.it.uu.se/research/publications/lic/2009-001/2009-001.pdf. – accessed on 21.07.2010

[Fried 2010] FRIED, Limor: *Wireless Arduino programming/serial link*. 2010. – URL http://www.ladyada.net/make/xbee/arduino.html. – accessed on 18.07.2010

[Gregor u. a. 2009] GREGOR, Sebastian ; RAHIMI, Mohammadali ; VOGT, Matthias ; SCHULZ, Thomas ; LUCK, Kai von: *Tangible Computing revisited: Anfassbare Computer in Intelligenten Umgebungen*. Hochschule für Angewandte Wissenschaften Hamburg. 2009. – URL http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/papers/MMWismar2009.pdf. – accessed on 20.07.2010

[Haenselmann 2006] HAENSELMANN, Thomas: *An FDL'ed Textbook on Sensor Networks*. 2006. – URL http://pi4.informatik.uni-mannheim.de/~haensel/sn_book. – accessed on 21.07.2010

[Kulakli und Erciyes 2008] KULAKLI, A.B. ; ERCIYES, K.: Time synchronization algorithms based on Timing-sync Protocol in Wireless Sensor Networks. In: *Computer and Information Sciences, 2008. ISCIS '08. 23rd International Symposium on* (2008), S. 1 – 5

[Legg 2004] LEGG, Gary: *ZigBee: Wireless Technology for Low-Power Sensor Networks*. 2004. – URL http://www.eetimes.com/design/communications-design/4017853/ZigBee-Wireless-Technology-for-Low-Power-Sensor-Networks. – accessed on 11.08.2010

[Meißner 2010] MEISSNER, Stefan: *Realisierung von Sound und Event Awareness durch verteilte Sensorknoten*, Hochschule für Angewandte Wissenschaften Hamburg, Masterarbeit, 2010. – URL http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/meissner.pdf. – accessed on 27.05.2010

[Otto und Voskuhl 2010] OTTO, Kjell ; VOSKUHL, Sören: *Kommunikationsschnittstelle*. 2010. – URL http://livingplace.informatik.haw-hamburg.de/wiki/index.php/Kommunikationsschnittstelle. – accessed on 21.07.2010

[RN-Wissen 2010] RN-WISSEN: *Avr-gcc*. 2010. – URL http://www.rn-wissen.de/index.php/Avr-gcc#Reset_ausl.C3.B6sen. – accessed on 11.08.2010

[Tanenbaum 2003] TANENBAUM, Andrew S.: *Computernetzwerke - 4., überarbeitete Auflage*. Prentice Hall, 2003. – ISBN 978-3-8273-7046-4

[Tanenbaum und van Steen 2003] TANENBAUM, Andrew S. ; STEEN, Marten van: *Verteilte Systeme - Grundlagen und Paradigmen*. Prentice Hall, 2003. – ISBN 3-8273-7057-4

[Tanenbaum und van Steen 2006] TANENBAUM, Andrew S. ; STEEN, Marten van: *Distributed Systems: Principles and Paradigms. 2nd Edition.* Prentice Hall, 2006. – ISBN 978-0-1323-9227-3

[The Apache Software Foundation 2010] THE APACHE SOFTWARE FOUNDATION: *Apache ActiveMQ > Connectivity > Cross Language Clients*. 2010. – URL http://activemq.apache.org/cross-language-clients.html. – accessed on 21.07.2010

[Weiser 1991] WEISER, Mark: The Computer for the Twenty-First Century. In: *Scientific American* 265 (1991), Nr. 3, S. 94–104

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 18.08.2010	
Ort, Datum	Unterschrift