



Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

René Schultz

Selbstorganisierendes drahtloses Sensor- und Aktornetzwerk

*Fakultät Technik und Informatik
Studiendepartment Informatik*

*Faculty of Engineering and Computer Science
Department of Computer Science*

René Schultz

Selbstorganisierendes drahtloses Sensor- und Aktornetzwerk

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

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

Betreuender Prüfer: Prof. Dr. Gunter Klemke
Zweitgutachter: Prof. Dr. Birgit Wendholt

Eingereicht am: 12. Mai 2014

René Schultz

Thema der Arbeit

Selbstorganisierendes drahtloses Sensor- und Aktornetzwerk

Stichworte

Sensor, Aktor, selbstorganisierend, drahtloses Netzwerk, ZigBee, XBee, Raspberry Pi, interaktive Installation

Kurzzusammenfassung

Im Verlauf dieser Bachelorarbeit wurde ein selbstorganisierendes drahtloses Sensor- und Aktornetzwerkes mit Hilfe des ZigBee-Protokolls auf der Arduino Plattform und dem Raspberry Pi entwickelt. Dies kann im Bereich der interaktiven Installationen von Nicht-Informatikern mit wenig Programmierkenntnissen eingesetzt werden.

René Schultz

Title of the paper

Selforganising Wireless Sensor and Actor Network

Keywords

Sensor, Actor, self organising, wireless network, ZigBee, XBee, Raspberry Pi, interactive installation

Abstract

In this bachelor thesis a self organising wireless sensor and actuator network was developed. The technologies used include Arduino, Raspberry Pi and the ZigBee Protocol. It can be used by non-IT personnel with little knowledge of programming in the context of interactive installations.

Inhaltsverzeichnis

0.1	Danksagung	1
1	Einführung	2
1.1	Einführung	2
1.2	Szenario	3
1.3	Ziel der Arbeit	3
2	Gliederung	4
3	Grundlagen	5
3.1	Sensoren und Aktoren	5
3.2	Sensor- und Aktornetzwerke	6
3.2.1	Sensornetzwerke	6
3.2.2	Sensornetzwerke mit Aktoren	8
3.3	Technologie	10
3.3.1	Standards	10
3.3.2	Selbstorganisation	11
3.3.3	Routing	11
3.3.4	Netztopologien	13
3.3.5	Kommunikation	15
3.3.6	Zuverlässigkeit	16
3.3.7	Synchronisation	16
3.4	Zusammenfassung	17
4	Analyse	18
4.1	Ziel der Arbeit	18
4.2	Anwendungsfeld interaktive Installationen	18
4.3	Technische Anforderungen	20
4.4	Fehlerbehandlung	22
4.5	Abgrenzungen und Einschränkungen	22
4.5.1	Aktualität der Daten	22
4.5.2	Zustellungsbestätigungen	22
4.5.3	Ortsbestimmung	22
4.6	Vergleichbare Arbeiten	23
4.7	Zusammenfassung	24

5	Design	25
5.1	Konzept	25
5.2	Hardware	27
5.2.1	Arduino	27
5.2.2	Weitere Hardware	31
5.2.3	Auswahl	32
5.3	Drahtlose Standards	33
5.3.1	WLAN	33
5.3.2	Bluetooth	34
5.3.3	ZigBee	34
5.3.4	Auswahl	35
5.4	Netzwerk	36
5.4.1	XBee	36
5.4.2	Modulhardware	36
5.4.3	Vorkonfiguration	37
5.4.4	Architektur	38
5.4.5	Konfiguration der Knoten	38
5.4.6	Ablauf	39
5.4.7	Protokoll	44
5.5	Zusammenfassung	47
6	Umsetzung	48
6.1	Hardware	48
6.1.1	Identifizierung über LED	49
6.2	Software	49
6.2.1	Diagramme	49
6.2.2	Software auf den Knoten	52
6.2.3	GUI	53
6.2.4	Vergleichen von XBeeAddress64 Adressen auf dem Arduino	53
6.3	X-CTU und Arduino IDE	53
6.4	Mögliche Fehler	55
6.4.1	X-CTU	55
6.4.2	Arduino IDE	56
6.4.3	Senden des ersten Pakets schlägt fehl	56
6.4.4	Reset bei Verbinden der seriellen Schnittstelle	57
6.5	Proof of Concept	57
6.5.1	Aufbau	58
6.5.2	Ablauf	59
6.5.3	Test	60
6.5.4	Auswertung	60
6.6	Zusammenfassung	62

7	Zusammenfassung	63
7.1	Ergebnisse	63
7.2	Ausblick	64
7.3	Abschließendes Fazit	64
8	Glossar	66
9	Anhänge	67
9.1	Kurzanleitung	67
9.2	Identifikations-LED an Arduino	67
9.3	Vergleichen von XBee64Address	68
9.4	Zentrale starten	68
9.5	Konfigurationsdateien	68
9.5.1	nodes.js	68
9.5.2	config.js	69
9.5.3	userConfiguration.h	69
9.6	CD	70

0.1 Danksagung

Mein Dank gilt Prof. Dr. Birgit Wendholt für die Idee und Ermöglichung etwas über Sensor-Aktor-Netzwerke zu schreiben, die Betreuung in den ersten Monaten und die Zweitkorrektur, Prof. Dr. Gunter Klemke, der sich so schnell bereit erklärt hatte, die Betreuung während Birgits Auslandssemester zu übernehmen, aber natürlich auch meiner Familie und meinen Freunden, die mich tatkräftig unterstützten.

1 Einführung

1.1 Einführung

Während vor 30 Jahren Computer noch relativ selten und etwas für Spezialisten waren, erhält die moderne Computertechnik mittlerweile Einzug in fast jeden Aspekt unseres Lebens. In vielen Bereichen wird unsere Umwelt konstant sensorisch überwacht und vermessen, um repetitive Arbeiten automatisiert von Computern erledigen zu lassen. Dies umfasst jedoch nicht nur Aufgaben, die vorher von Menschen erledigt wurden, sondern eröffnet auch die Möglichkeit neue Daten zu erfassen, die ohne die moderne Technik nicht messbar und auswertbar wären.

Diese Sensoren werden häufig zu einem Netzwerk verbunden, um Daten auszutauschen und zu aggregieren. Sie sind unauffällig in alltägliche Gegenstände bzw. Gebäude integriert. Damit auf Sensorwerte eingegangen werden kann, werden Aktoren wie etwa Motoren hinzugefügt. Dies soll den Menschen vor allem dadurch das Leben erleichtern, indem es uns automatisierbare Überwachungsaufgaben abnimmt und sofort auf bestimmte Situationen eigenständig reagieren kann. Die teils mehr als 20 Jahre alten Ideen von Visionären wie dem Informatiker *Mark Weiser* werden zunehmend Wirklichkeit:

„Therefore we are trying to conceive a new way of thinking about computers in the world, one that takes into account the natural human environment and allows the computers themselves to vanish into the background.”

– Mark Weiser - *The Computer for the 21st Century*, 1991 [64]

Der Forschungsbereich der Sensor- und Aktornetzwerke widmet sich der technischen Realisierung solcher Netze, da sie anderen Anforderungen unterliegen als gut erforschte Netze wie etwa dem Internet. So können sie intelligenter gestaltet werden und den Menschen effizienter die Arbeit erleichtern. Sensornetzwerke bzw. Sensor- und Aktornetzwerke können jedoch nicht nur in der Industrie eingesetzt werden. Auch wenn sie ursprünglich aus dem technischen bzw. militärischen Bereich kommen, finden sich immer mehr Anwendungen im privaten Bereich (z.B. Smart Homes) sowie im Kunst- oder Hochschulbereich, da diese Netze diverse Vorteile gegenüber

drahtgebundenen Netzen bieten. Sie unterliegen jedoch anderen Anforderungen und können ggf. mit einfacheren Mitteln realisiert werden. Besondere Schwierigkeiten für Laien umfassen die Programmierung und Konfiguration eines solchen Netzes. Daher soll diese Arbeit Nicht-Informatikern Hilfsmittel geben, um Sensor- und Aktornetzwerke in ihrem Kontext anwenden zu können.

1.2 Szenario

Als Beispiel für einen Anwendungsfall im Kunst- bzw. Designbereich ist das folgende Ausgangsszenario dem Paper von Mark Weisers *The Computer for the 21st Century* [64] nachempfunden:

Die Künstlerin Sally wird von einer Marketingfirma über das Internet für ein Projekt kontaktiert. Ihr Auftrag ist es, sich ein Konzept zu überlegen, wie Kunden auf die im Schaufenster ausgestellten Kleidungsstücke aufmerksam gemacht werden können, selbst wenn das Geschäft geschlossen hat. Sie entscheidet sich schnell dafür, eine Vielzahl aus Sensoren zu benutzen, welche die Präsenz der Kunden vor dem Schaufenster feststellen soll. Kleine Sensorknoten sollen innen an der Schaufensterscheibe angebracht werden. Wenn diese Informationen vorhanden sind, braucht sie noch eine Möglichkeit auf sich aufmerksam zu machen. Sie fügt dem Aufbau Aktoren hinzu; dazu gehören unter anderem LEDs zur Beleuchtung der Schaufensterpuppen und Motoren, um diese zu bewegen. Eine Bewegung eines Passanten an einem Schaufenster soll auch eine Aktion in einem anderen Schaufenster hervorrufen können. Da sie nicht im gesamten Geschäft Kabel verlegen kann, würde Sally gerne ein drahtloses Netzwerk benutzen. Da Sally Design und Medien studiert hat, hat sie allerdings nur sehr begrenzt Ahnung von der Programmierung eines solchen Netzwerkes.

1.3 Ziel der Arbeit

Das Ziel dieser Arbeit ist es, ein drahtloses Sensor- und Aktornetzwerk zu entwickeln, das für interaktive Kunstinstallationen benutzt werden kann. Dabei soll besonderer Wert auf eine gute und eigenständige Bedienbarkeit durch Nicht-Informatiker mit wenigen Programmierkenntnissen gelegt werden. Durch die Benutzung eines Funkstandards sollen der Installationsaufwand und die Kosten niedrig gehalten sowie bestmögliche Flexibilität für die Künstler gewährleistet werden. Mithilfe der entwickelten Werkzeuge soll die Technik in den Hintergrund treten und die Konfiguration vereinfacht und so weit wie möglich automatisiert werden.

2 Gliederung

Die Bachelorarbeit ist in sechs Bereiche gegliedert. Nach der Einleitung und dieser Gliederung werden im dritten Kapitel die Grundlagen von Sensoren, Aktoren und der Netzwerktheorie, speziell im Bezug auf Sensor-Aktor-Netzwerke.

Im vierten Kapitel wird das Anwendungsgebiet *interaktive Installationen* vorgestellt sowie die technischen Anforderungen an das Projekt gestellt. Zudem werden technische Einschränkungen erläutert und vergleichbare Abschlussarbeiten aufgeführt und von diesen abgegrenzt.

Direkt thematisch angeschlossen ist das fünfte Kapitel, welches das Konzept und Design erläutert und die Architekturentscheidungen fällt und begründet. Hier werden auch die Hardware- und Protokollentscheidungen gefällt und erläutert. Es wird ebenfalls ein Plan für die zu erstellende Software entwickelt und ein eigenes Protokoll für die Kommunikation der Netzwerkknoten entwickelt.

Das fünfte Kapitel beinhaltet die Umsetzung des Netzwerkes. Das Design wird erläutert und verschiedene Probleme und Lösungen werden dargestellt. Da für diese Arbeit keine interaktive Installation zur Verfügung steht, wird ein Proof-of-Concept entwickelt und getestet.

Als Letztes wird in der Zusammenfassung ein Fazit gezogen und die eingesetzte Technik im Bezug auf das in der Einleitung genannte Ziel bewertet. Weiterhin gibt es einen Ausblick über die Grenzen dieser Arbeit hinaus.

3 Grundlagen

In diesem Kapitel werden die grundlegenden Konzepte und Begriffe des Bereiches Sensor- und Aktornetzwerke erläutert. Es umfasst eine generelle Definition von Sensoren und Aktoren sowie Sensornetzwerken, Sensor-Aktor-Netzwerken, deren Unterschiede, Anwendungen, das Internet of Things und Netzwerktheorie.

3.1 Sensoren und Aktoren

Sensoren sind elektrische Bauteile, die Eigenschaften ihrer Umgebung erfassen können und in elektrische Signale umwandeln. Diese Eigenschaften können z.B. physikalischen oder chemischen Ursprungs sein. Sie können Signale analog oder digital weitergeben. In letzter Zeit werden Sensoren zunehmend mit Intelligenz in Form eines Mikrocontrollers u.ä. ausgestattet. Man nennt diese Art von Sensoren dann Smart-Sensoren[42]. Beispiele für Sensoren sind Sensoren zur Abstandskontrolle (Infrarot, Sonar), Sensoren zur Messung von Temperatur und Druck, aber auch Mikrofone, Kameras oder Touchscreens gehören zur Kategorie der Sensoren. Im weitesten Sinne könnte man unter Sensoren auch andere Datenquellen z.B. aus dem Internet verstehen, die vielleicht sogar auf tatsächlichen Sensordaten beruhen und beispielsweise über APIs abgerufen werden können.

Aktoren sind das exakte Gegenteil von Sensoren. Sie setzen elektrische Signale in physikalische Größen bzw. Bewegung um. Häufig reagieren sie auf aufbereitete Messdaten, die von Sensoren stammen. Beispiele sind Motoren, Lautsprecher oder Leuchtmittel.

3.2 Sensor- und Aktornetzwerke

3.2.1 Sensornetzwerke

Definition

„Wireless sensor networks consist of many small nodes. Each node has a micro-processor, a radio chip, some sensors[...]”

–Joakim Eriksson[17]

Es gibt keine einheitliche Definition von einem Sensornetzwerk. Die meisten Definitionen gehen jedoch wie Eriksson davon aus, dass es ein Netzwerk von mehreren Knotenpunkten ist, die drahtlos miteinander kommunizieren. Dabei wird das Netzwerk häufig ad hoc aufgebaut, d.h. ohne feste Infrastruktur. Auch ob diese Knoten zwangsweise mobil und batteriebetrieben sein müssen oder ob es auch fest installierte Knoten geben kann, ist nicht festgelegt. Alle Definitionen beschreiben Sensorknoten aber als kleine, günstige und stromsparende Produkte, die man in großen Mengen bei mehreren Herstellern beschaffen kann. Um ein Netzwerk als Sensornetzwerk bezeichnen zu können, muss der überwiegende Teil dieser Punkte erfüllt sein.

Anwendungsgebiete

Ursprünglich wurden Sensornetzwerke als Frühwarnsysteme im militärischen Bereich genutzt, um große Landstriche oder Komplexe zu überwachen[8]. In den letzten Jahrzehnten ergaben sich jedoch vielseitige Nutzungen in der Industrie. Gerade in letzter Zeit werden verschiedenartige Sensoren in der Gebäudeautomation (Smart Home) genutzt und miteinander vernetzt. Dazu gibt es verschiedene Initiativen, z.B. „Intelligentes Wohnen” vom Zentralverband Elektrotechnik- und Elektronikindustrie (ZVEI)[18].

Sensornetzwerke können mit mehreren tausenden Knoten sehr groß skalieren, d.h. dass sowohl viele Knoten verwendet werden können oder auch, dass die Knoten über einen großen geographischen Raum verteilt sind. Dies ist vor allem im landwirtschaftlichen Bereich der Fall, etwa zur Erkennung von Waldbränden[55]. Das SISVIA (span. *Vigilancia y Seguimiento Ambiental*, Überwachung und Kontrolle der Umwelt) Projekt hat in Nordspanien auf einem Gebiet von 210 Hektar zahlreiche Gas-, Temperatur- und Feuchtigkeitssensoren installiert, um vorzeitig vor großen Waldbränden zu warnen, die Feuerwehr zu unterstützen und die lokale Bevölkerung ggf. evakuieren zu können. Ein Beispiel für ein geographisch weit verteiltes Sensor-Aktor-Netzwerk ist das DART-Projekt des US-amerikanischen National Data Buoy Center[7], das vor der Küste der USA Anzeichen für ein Erdbeben bzw. Tsunami misst. Mittlerweile ist es in der ganzen

Welt verteilt. Es besteht aus mehreren Bojen, die unter Wasser mit Sensoren verbunden sind. Die Bojen sind in der Lage, die Daten über Satellit an die Zentrale zu senden, in der dann Bedrohungsanalysen berechnet werden.

Ein Beispiel für den schon erwähnten militärischen Bereich ist das mobile *Unattended Ground Sensor System* (UGS) des *Brigade Combat Team Modernization Program* der US Army[59]. Hier werden großflächig akustische, Radioaktivitäts- und Bewegungssensoren sowie Kameras verteilt, die bei der Aufklärung von gefährlichen Gebieten helfen sollen. Außerdem können mit dem *Urban-Unattended Ground System* Gebäude überwacht werden, die im Hauptquartier der Streitkräfte, dem *Global Information Grid*, mit anderen Informationen zusammengefasst werden und somit z.B. eine schnelle Entscheidungshilfe beim Häuserkampf geben sollen.



Abbildung 3.1: Soldat der US Army mit einem Sensorknoten des UGS[4]

Ein Spezialfall der Sensornetzwerke mit mehreren tausend kleinen Knoten ist der sog. Smart Dust (engl. *intelligenter Staub*) [12]. Dabei können die einzelnen batteriebetriebenen Knoten kleiner als ein Reiskorn sein. Sie sind zwar extrem günstig, aber auch sehr anfällig gegenüber elektromagnetischen Störungen und Witterung. Aufgrund der Größe ist die Rechenkapazität und Sendeleistung meist zu klein um ein Betriebssystem darauf zu betreiben. Deswegen wurden im Zuge der Entwicklung von Smart Dust noch diverse andere Entwicklungen wie z.B. größere Knoten finanziert, die dann als Mittler zwischen den kleineren Endpunkten eingesetzt werden. Unter anderem entwickelt die Universität von Kalifornien in Berkeley in Kooperation mit Intel und Crossbow Technology das quelloffene Betriebssystem *TinyOS* und den C Dialekt *nesC*, der speziell auf eingebettete Netzwerkgeräte ausgelegt ist. Der Speicherverbrauch ist mit ca. 1kB RAM

und 4kB ROM extrem niedrig[50]. Dieses Betriebssystem läuft dann auf den größeren Knoten. Zur Speicherung bzw. zum Abruf der Daten von den Knoten wurde weiterhin eine Datenbank namens *TinyDB* entwickelt, mit der man mit SQL-ähnlichen Abfragen Daten im Netzwerk abfragen kann. Neben *TinyOS* sind noch die beiden Open Source Betriebssysteme *Contiki*[9] und *RIOT*[50] erwähnenswert. Inzwischen gibt es z.B. von der Firma *Willow Technologies* sogar kommerzielle Produkte[39].

3.2.2 Sensornetzwerke mit Aktoren

Definition

Wie schon erwähnt, können Aktoren (aufbereitete) Daten von Sensoren verwenden. Wenn heterogene Knoten in einem Netzwerk zusammen agieren, nennt man dies ein Sensor- und Aktornetzwerk, Sensor-Aktor-Netzwerk oder auch drahtloses Sensor-Aktor-Netzwerk. Im Englischen wird dies als *Wireless Sensor and Actor Network* oder auch *Wireless Sensor Actuator Network* (WSAN) bezeichnet. In dem Paper *Self-Recovering Sensor-Actor Networks* von der Abo Akademi Universität in Turku, Finnland beschreiben die Autoren WSANs folgendermaßen[32]:

„The sensor nodes are low-cost, low-power devices equipped with limited communication capabilities, while the actor nodes are usually mobile, more sophisticated and powerful devices compared to the sensor nodes.”

In einigen Punkten sind sich Sensornetzwerke und Sensor-Aktor-Netzwerke sehr ähnlich, haben aber zusätzlich besondere Eigenschaften. Dazu zählt häufig die Verarbeitung der Daten in Echtzeit, d.h. auf Sensor Input muss der Aktor sehr schnell reagieren können [53]. Da sich ggf. auch Aktoren untereinander absprechen können müssen, spielt die Koordination eine große Rolle. Das Forschen an neuen und effizienteren Protokollen sowie Routing-Protokollen (s. unten) ist Gegenstand aktueller Forschung an vielen Forschungseinrichtungen und Hochschulen. Dabei ist ein großer Teil der Herausforderungen, bestehende Protokolle für Sensornetzwerke auf die speziellen Bedürfnisse der Sensor-Aktor-Netzwerke anzupassen. Dabei geht es vor allem um effiziente Sensor-Aktor bzw. Aktor-Aktor Kommunikation in Echtzeit.[52]

Anwendungsgebiete

Ein Beispiel für ein Netzwerk aus dem Bereich Smart Home ist das Sensor- und Aktornetzwerk der Firma *Murata*, welches verschiedene Geräte steuern kann (Fensterläden, Licht, Ventilator), aber auch Daten erfasst (Temperatur, Licht, Bewegung)[38]. Diese Daten werden über ein Gateway zusammengefasst und über das Internet bzw. das lokale Netzwerk an weitere Geräte, z.B. Tablets,

gesendet. Dies soll vor allem ältere Menschen im täglichen Leben unterstützen. Genau genommen könnte man auch das schon genannte DART Projekt als Sensor-Aktor-Netzwerk begreifen, da die Bojen die Daten empfangen und eine Aktion ausführen: Das Senden der Daten über Satellit.

Ein weiteres Beispiel ist das Projekt *StickEar* der *Augmented Senses Group* der Singapore University of Technology and Design. Der Name ist dem englischen Wort für Haftnotizen, Sticky Notes, nachempfunden. Während Haftnotizen Daten visuell festhalten, sind StickEars kleine runde Geräte (s. Abb. 3.2), die mit einem Mikrofon und Lautsprecher ausgestattet Daten auditiv speichern, verarbeiten und weitergeben. Sie bilden ein Netzwerk und sollen unerfahrenen Nutzern einen intuitiven Zugang zu hilfreicher Technologie bieten[60]. Dies kann z.B. das auditive Festhalten von Notizen, die Soundüberwachung eines Raumes (ähnlich eines Babyfons) oder das autonome Reagieren auf bestimmte Soundevents sein.

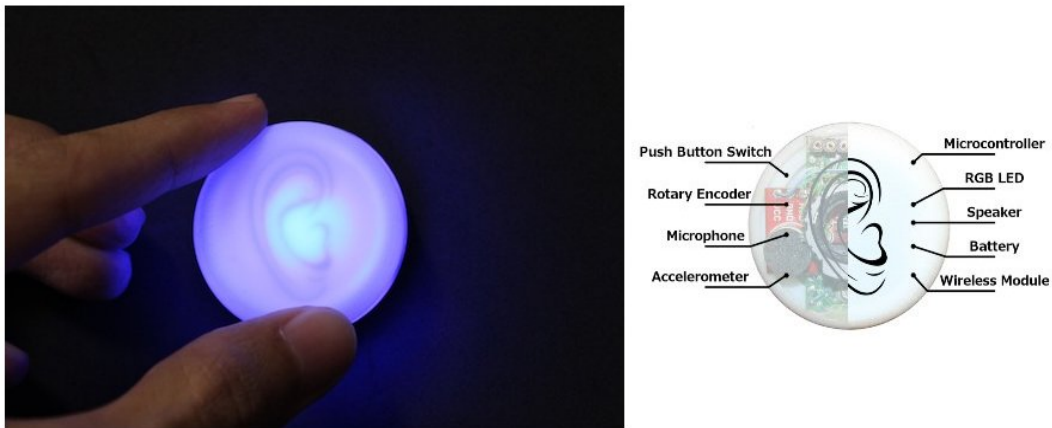


Abbildung 3.2: Ein StickEar Prototyp(l.) und Modell(r.)[44][61]

Internet of Things

Das Internet of Things (IoT) bzw. Internet der Dinge ist ein Stichwort, welches in der Diskussion um Sensor-Aktor-Netzwerke nicht unerwähnt bleiben darf. Der Begriff stammt aus dem Aufsatz *The Computer for the 21st Century*[64] des US-amerikanischen Wissenschaftlers Mark Weiser. Das Konzept IoT sieht vor, dass auch alltägliche technische Geräte an das Internet angeschlossen werden und nicht als solche zu erkennen sind. Dies nennt man auch Ubiquitous computing (engl. *allgegenwärtiges Rechnen*). Dabei haben die „Dinge“ eine virtuelle Repräsentation. Identifiziert werden können sie von außen z.B. durch RFID-Chips oder QR-Codes. Am meisten wird heutzutage daran geforscht, wie diese Dinge untereinander intelligent kommunizieren, um dem

Benutzer das alltägliche Leben zu erleichtern[16][27]. Ein häufiges Beispiel sind miteinander kommunizierende Haushaltsgeräte oder der mit Sensoren ausgestattete Drucker, welcher bei niedrigem Füllstand automatisch neue Patronen beim Hersteller ordert. Voraussetzung für die Kommunikation ist hierbei ohne Zweifel, dass die Hersteller sich auf Kommunikationsstandards einigen. Im Beispiel des Druckers wäre es aus Benutzersicht z.B. sinnvoll, die Größe und den Anbieter seiner Patrone wählen zu können. Jeder Anbieter müsste diesen Standard beherrschen, um den Auftrag entgegennehmen zu können. Die Hersteller haben aber verständlicherweise ein Interesse daran, nur ihr Produkt zu vertreiben.

Ein weiterer Kritikpunkt ist die Angreifbarkeit von alltäglichen Dingen von einem weit entfernten Ort. Da in der Praxis heutzutage internetfähige Geräte (außer Computer) fast nie ein Update bekommen, liegt die Vermutung nahe, dass sich dies auch in naher Zukunft nicht ändern wird. Somit könnten alltägliche Gegenstände z.B. von privaten Angreifern oder unterdrückenden Regierungen und Geheimdiensten als Wanzen genutzt werden. Aktive Gegenstände, also die Akteure im IoT, könnten zudem, gewollt oder ungewollt, eine aktive Bedrohung für den Benutzer darstellen.

3.3 Technologie

In den folgenden Kapiteln findet sich ein Überblick über die in Sensor- und Aktornetzwerken verwendeten Technologien und Standards.

3.3.1 Standards

Da für drahtlose Sensornetzwerke andere Ansätze gelten als für andere drahtlose Netze, gibt es spezielle offene und proprietäre Standards. Sonstige wichtige Merkmale sind vor allem der Energieverbrauch, die maximale Anzahl der Knotenpunkte und die Art der Kommunikation. Als Beispiele für offene Standards sollen hier ZigBee und ZWave hervorgehoben werden, die von größeren Zusammenschlüssen von Firmen entwickelt wurden, aber auch das proprietäre ANT(+), welches von der Firma *Dynastream* hergestellt wird. Die MAC und PHY Layer von Z-Wave sind ein Standard der Internationalen Fernmeldeunion (ITU), einer Sonderorganisation der UNO, wogegen die MAC und PHY Layer von ZigBee auf dem 802.15.4 Standard des Institute of Electrical and Electronics Engineers (IEEE) basieren.

ZigBee und Z-Wave sind direkte Konkurrenten, da sie ähnliche Anwendungsgebiete haben und beide Mesh-Netzwerke (s. Kap. 3.3.4) bilden. Es gibt technisch nur marginale Unterschiede, allerdings hat ZigBee mit 250 kBit/s die 2,5-fache Übertragungsrate von Z-Wave. ANT hingegen

hat eine viermal größere Übertragungsrate als ZigBee und ist damit eher für Netzwerke mit hohem Datendurchsatz, aber kurzer Reichweite (bis zu 30 Meter) ausgelegt. Auch Z-Wave hat eine auf 30 Meter beschränkte Reichweite, verbraucht dafür aber weniger Energie als ZigBee Geräte, die bis zu 100 Meter weit senden können. Der große Vorteil von ZigBee gegenüber Z-Wave ist die hohe Anzahl an möglichen Teilnehmern. Während Z-Wave eine maximale Anzahl von 232 Knoten beherrscht, kann ZigBee mit 65536 Teilnehmern mehr als 280 mal so große Netze aufbauen.

3.3.2 Selbstorganisation

Selbstorganisation bezeichnet in diesem Kontext die Vorgehensweise und Technologie, die von Netzwerken benutzt wird, um aus sich heraus den Aufbau, die Konfiguration, das Management und die Optimierung des Netzwerkes durchzuführen. In Netzwerken muss jeder Knoten eindeutig identifizierbar sein. Umsetzbar ist dies z.B., indem der Hersteller eine genügend große Seriennummer vergibt (z.B. MAC Adressen). Zur weiteren Identifikation muss nicht zwingend diese Seriennummer genutzt werden. Nachdem sich alle Knoten zuvor über ihre Seriennummer gemeldet haben, können nur für das Netzwerk gültige Adressen spontan von einer zentralen Instanz vergeben werden. Erst dann kann sich auf Protokollebene ein Netzwerk ad-hoc selbst organisieren.

3.3.3 Routing

Routing soll sicherstellen, dass Daten über mehrere Knoten hinweg geroutet werden können (sog. Multihopping Verfahren), und dass die Daten nicht in einer Sackgasse landen oder sich Schleifen bilden, in denen Pakete unendlich hin- und her gesendet werden. Es muss jeder Knoten mittels eines Routing-Protokolls zu den anderen Knoten den möglichst kürzesten bzw. effizientesten Weg finden. Für Sensornetzwerke gibt es spezielle Routing-Protokolle[25]. Die Protokolle aus anderen Netzwerken funktionieren zwar ebenfalls für den Spezialfall Sensornetzwerk, sie sind aber nicht optimal. Sensornetzwerke sind häufig nicht optimalen Umgebungen ausgesetzt. So können spontan durch äußere Einflüsse Knoten im Netzwerk nicht erreichbar sein und wieder hinzugefügt werden. Zu berücksichtigen ist, dass das Aktualhalten einer Routingtabelle auf jedem Knoten sowohl Speicherplatz als auch Rechenleistung und damit letztlich Energie kostet. Es gibt es prinzipiell zwei Ansätze, wann dieses Routing stattfindet.

Proaktives Routing Beim proaktiven Routing werden vor dem Versenden der eigentlichen Daten schon alle möglichen, durch Kontrollpakete in Erfahrung gebrachten Routen in den Knoten gespeichert. Dies verbraucht mehr Speicher und durch die hohe Anzahl an Kontrollpaketen auch

mehr Energie. Dafür ist es aber auch schneller, weil eine Route nicht erst gefunden werden muss. Ein Beispiel für dieses Verfahren liefert das *Optimized Link State Routing* (OLSR) der IETF (RFC 3626).

Reaktives Routing Wie der Name schon andeutet, wird bei diesem Routingverfahren der Pfad nur berechnet, wenn auch Nutzerdaten gesendet werden sollen. Dies führt zu sehr viel weniger Kontrollpaketen und hat somit einen positiven Einfluss auf den Energieverbrauch. Wenn eine Route einmal berechnet wurde, kann sie in einem Routenverzeichnis gecacht werden, wie es z.B. beim *Ad Hoc On-Demand Distance Vector* (AODV) Verfahren der Fall ist. Der Nachteil ist, dass es eine Verzögerung gibt, da die Route erst gesucht werden muss.

Hybride Verfahren Proaktives und reaktives Routing können auch verbunden werden. Dabei kann z.B. im lokalen Bereich proaktives Routing verwendet werden sowie reaktives Routing für weiter entfernte Knoten. Ein Beispiel hierfür ist das *Zone Routing Protocol* (ZRP), welches für *Intra-zone* Routing ein proaktives Verfahren benutzt und für *Inter-zone* Routing ein reaktives Verfahren.

Positionsbasierte Verfahren Neben den Verfahren, die auf der Topologie basieren, gibt es jedoch auch noch positionsbasierte Verfahren, bei denen die Knoten z.B. aufgrund ihrer eigenen GPS Koordinaten ein effizientes Routing berechnen können.

Protokolle

Beim Routing gibt es zwei grundsätzliche Protokollarten, die nach den folgenden Prinzipien funktionieren:

Übertrage deine Weltsicht Diese Verfahren werden Distanzvektorverfahren genannt. Jeder Knoten erstellt eine Kostenmatrix der Knoten, die er sehen kann, und schickt sie weiter. Er erwartet dann die Matrizen der Nachbarknoten, um seine Kostenmatrix ggf. anpassen zu können. Dazu müssen sich die Knoten vorher nicht alle kennen. Ggf. wird also das Paket nur an den optimalen Nachbarn weitergereicht und das Problem somit aufgeteilt, da kein Knoten Kenntnis über das gesamte Netzwerk hat.

Übertrage deine Nachbarn Diese Verfahren werden Link State Verfahren genannt. Im Gegensatz zu Distanzvektoralgorithmen ist auf jedem Knoten der komplette Aufbau des Netzes bekannt. Der Knoten kann also eigenhändig berechnen und entscheiden, wohin er die Daten

senden muss. Wenn es oft Änderungen in der Routingtabelle gibt, ist ein Routing-Protokoll mit einem Link State Verfahren zu empfehlen. Vor allem bei großen Netzwerken werden dann nur die Änderungen per Flooding an alle Nachbarknoten geschickt.

Bei Sensornetzwerken können beide Prinzipien angewendet werden. In der Praxis benutzt z.B. der schon erwähnte ZigBee Standard das AODV Protokoll, welches zu den Distanzvektorprotokollen gehört. Außerhalb der Forschung wird das nach dem Link State Verfahren funktionierende OLSR Protokoll z.B. in der Freifunk Community eingesetzt. Diese entwickelt das B.A.T.M.A.N. Protokoll, welches dazu eingesetzt wird, kostengünstige Heimrouter miteinander zu vernetzen, um z.B. Internetanschlüsse zwischen Hausnachbarn in Mietshäusern zu teilen. Eine der neusten Entwicklungen in diesem Bereich ist das von der IETF im RFC 6550 spezifizierte dynamische Routing-Protokoll *RPL (Routing Protocol for Low power and Lossy Networks)*, welches für IPv6 spezifiziert ist und eng mit dem *6LoWPAN (IPv6 over Low power Wireless Personal Area Network)* Protokoll verwandt ist[25][24]. Es basiert auf einem Distanzvektoralgorithmus, ist selbstorganisierend und einfach zu implementieren. Welches Routing-Protokoll bzw. welches Verfahren eingesetzt wird, ist unter anderem abhängig von den Netzwerktopologien, die im nächsten Kapitel vorgestellt werden.

3.3.4 Netztopologien

Die Art und Weise wie ein Netzwerk organisiert ist, nennt man Topologie. In selbstorganisierenden Netzen bauen Knoten selbstständig ein Netzwerk nach einer bestimmten Topologie auf. Es kann verschiedenartig aufgebaut sein, Abb. 3.3 zeigt einige Beispiele. Für drahtlose Sensornetzwerke bieten sich vor allem das schon erwähnte Mesh-Netzwerk, Stern-Netzwerk (Star), aber auch das Baum-Netzwerk (Tree) bzw. das Cluster Tree Netzwerk an (Abb. 3.4).

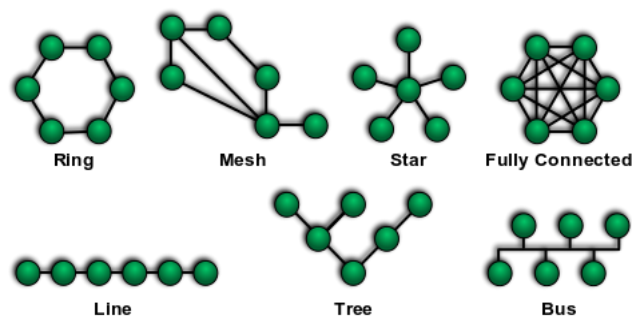


Abbildung 3.3: Verschiedene Netzwerktopologien [36]

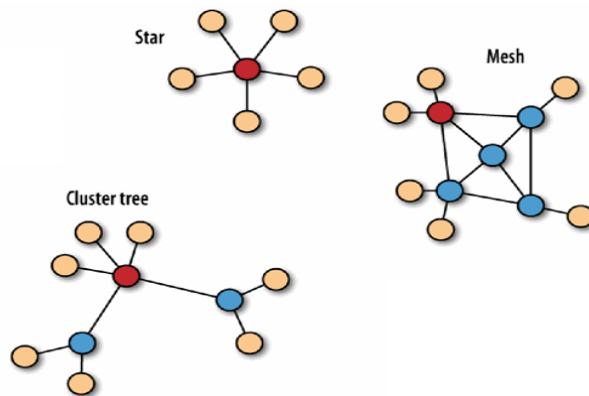


Abbildung 3.4: Verschiedene Netzwerktopologien für drahtlose Sensornetze [19]

Wie die verschiedenen Farben der Knoten in Abb. 3.4 andeuten, kann es in einem Sensornetzwerk mehrere verschiedene Funktionen der Knoten geben, aber dazu später mehr in Kapitel 5.3.3. Typischerweise gibt es einen zentralen Punkt, einen Koordinator, und zumindest eine Schnittstelle zum Benutzer, damit die Daten abgegriffen werden können.

Star Network Im Star Network kommunizieren die Knoten mittels eines zentralen Koordinators. Er kann entweder Daten empfangen und auswerten oder auch nur als Mittler zwischen den Knoten agieren. Der zentrale Punkt ist der Single Point of Failure, da jegliche Kommunikation zwischen den Knoten unmöglich wird, sobald er ausfällt. Auf die Konstruktion des zentralen Knotens ist deshalb besonders großer Wert zu legen.

Mesh Network In einem Mesh Network kommunizieren die Knoten untereinander direkt d.h. sie sind als gleichwertige Knoten definiert, die zwar andere Funktion erfüllen können, aber nicht in einer klassischen Server-Client Verbindung stehen. Zusätzlich leiten sie noch Daten ihrer Nachbarknoten weiter. Das Senden der Daten nach diesem Prinzip nennt man Multihopping, weil das Datenpaket auf dem Weg zur Destination mehrere Sprünge (Hops) machen kann. Das Mesh Network kann sich ad hoc selbst oder über eine Infrastruktur organisieren. Es setzt einen komplizierteren Aufbau als beim Star Network voraus, ermöglicht jedoch dank der Peer-to-Peer Architektur eine Kommunikation unabhängig von einer Zentrale. Es kann allerdings auch dazu führen, dass Knoten nicht miteinander kommunizieren können, weil sie sich im Netzwerk kurzzeitig nicht gegenseitig sehen bzw. keine Route vorhanden ist, da dieser Aufbau zu schwankenden Sende- und Empfangszeiten führt oder aufgrund des Hinzufügens oder Entfernens von Knoten eine unvorhergesehenen Neuorganisation stattfindet, s. auch Kap. 3.3.5.

Cluster Tree Das in einer Baumstruktur angeordnete Cluster Tree Network ist gewissermaßen eine Mischung aus Star und Mesh Network. Es kann mehrere Subnetze geben, die aber über einen gemeinsamen Parent-Knoten verbunden sind. Die einzelnen Knoten kommunizieren nicht über *einen* zentralen Punkt, sondern über die jeweiligen Parent-Knoten.

3.3.5 Kommunikation

In Sensor-Aktor-Netzwerken können Sensoren und Aktoren gezielt aufeinander abgestimmt sein, d.h. ein oder mehrere spezielle Sensoren sind für ein oder mehrere Aktoren zuständig. In diesem Fall schickt der Sensorknoten per Unicast seine Daten an eine oder mehrere spezielle Adressen. Es gibt jedoch auch Fälle, in denen es auf Aktorseite irrelevant ist, woher die Messdaten kommen. In diesem Beispiel könnten die Sensorknoten auch einen Broadcast senden, was zum Nachteil hätte, dass auch Knoten das Paket bekommen, für die es nicht gedacht war und sie so Rechenzeit aufwenden müssen, um eben das festzustellen. Ob sich beim Design für Broadcast oder Unicast entschieden wird, hängt auch davon ab, nach welchem Muster die Daten gesendet werden. Unabhängig von Unicast oder Broadcast lässt sich die Kommunikation auch noch nach den folgenden Kommunikationstypen klassifizieren:

Periodisches Senden Der Sensorknoten sendet periodisch pro Zeitabschnitt eine festgelegte Anzahl an Daten. Diese Art der Kommunikation kann das Netz stark belasten, ist dafür aber auch am besten vorhersagbar und bleibt auch in Ausnahmesituationen konstant.

Senden bei Änderung Hier entscheidet der Sensorknoten anhand vorher festgelegter Schwellenwerte, ob der gerade gemessene Wert gesendet werden soll. Bei einer Fehlfunktion oder in einer Ausnahmesituation und Kettenreaktionen kann es es zu sog. Alarmfluten kommen, die das Netzwerk stärker belasten als periodisches Senden.

Protokoll der Anwendungsschicht

Genau wie bei der Kommunikation über TCP/IP im Internet, gibt es eine Anwendungsschicht bzw. ein Anwendungsprotokoll, welches definiert, welche Nachrichten Sensoren und Aktoren untereinander austauschen. Dazu gehören Konventionen zur Identifikation der Knoten, Konstanten für z.B. Flags, um Einstellungen zu ändern, und das Wrappen von Rohdaten. Beachtet werden müssen hierbei insbesondere die Datentypen bzw. Endianess (Byte-Reihenfolge). Je nach Übertragungsstandard können auf der Anwendungsschicht noch weitere Features wie z.B. Verschlüsselung implementiert werden, falls dies nicht von den unteren Schichten (ausreichend) bereitgestellt wird. Da die meistens Standards eine eher kleine Payload (z.B. ZigBee ~100 Byte) erlauben, ist es außerdem möglich, die Daten zu fragmentieren und wie bei TCP über Sequenznummer in eine Reihenfolge zu bringen, um ggf. größere zusammenhängende Datenmengen zu übertragen.

3.3.6 Zuverlässigkeit

Sensor- und Aktornetzwerke können wie schon erwähnt Umgebungen ausgesetzt sein, die eine fehlerfreie Funktion unmöglich machen. Zum Beispiel könnte es passieren, dass elektromagnetische Interferenzen die Übertragung stören, Wolken die Energiegewinnung mit Solarpanelen erschweren oder dass die Witterung oder Naturgewalten Sensoren bzw. Aktoren von ihren ursprünglichen Plätzen verschieben. Deswegen muss schon beim Design darauf geachtet werden, dass Daten zuverlässig übermittelt werden können.

Die einfachste Form wäre, zu jedem gesendeten Paket ein Acknowledgment (ACK) Paket zu senden, wie es auch in TCP/IP Netzen praktiziert wird[26]. Auch kann der zuverlässige Versand in das Routing-Protokoll eingebaut sein. Ein Beispiel hierfür ist das Greedy Perimeter Stateless Routing in Wireless Networks (GPSR), ein Verfahren, bei dem Datenpakete direkt gesendet werden, ohne vorher die perfekte Route zu kalkulieren (Greedy Strategie). Sie umkreisen auch den Zielknoten, um sicherzustellen, dass Datenpakete nicht in Sackgassen enden, falls Knoten ausfallen[33].

3.3.7 Synchronisation

Gerade in zeitkritischen oder sicherheitsrelevanten Anwendungen ist es wichtig, dass sich die Knoten untereinander synchronisieren, also eine gemeinsame Zeit haben. Dies muss nicht der realen Zeit entsprechen, es reicht, wenn die Knoten untereinander über ein konsistentes System zur Synchronisation verfügen. Dies stellt in jedem verteilten System eines der Hauptprobleme dar[11]. Es gibt diverse Standardlösungen für dieses häufige Problem. Diese umfassen entweder

einen zentralen Zeitgeber (z.B. *Reference Broadcast Synchronisation* (RBS)[35]), selbstorganisierende hierarchische Algorithmen (z.B. *Timing-Sync Protocol for Sensor Networks* (TPSN)[22]) oder ganz einfach die Messung der Round Trip Time (RTT).

3.4 Zusammenfassung

In diesem Kapitel wurden die Grundlagen für Sensor- bzw. Sensor-Aktor-Netzwerke erläutert und Beispiele aus den Bereichen Forschung und Industrie vorgestellt, auch im Hinblick auf das Internet of Things. Weiterhin wurde auf die Grundlagen der Netzwerktheorie bzw. Netzwerkarchitektur eingegangen, welche später für das Design bzw. die Umsetzung in Kapitel 5 bzw. 6 benötigt werden.

4 Analyse

4.1 Ziel der Arbeit

Das Ziel dieser Arbeit ist die Entwicklung eines selbstorganisierenden Sensor- und Aktornetzwerkes für das Anwendungsgebiet interaktive Installationen, welches im Folgenden erläutert wird. Hierbei soll besonderen Wert darauf gelegt werden, dass Künstler ohne bzw. mit wenig Programmierausbildung Hard- und Software eigenständig oder mit wenig Hilfe in Betrieb nehmen können. Es soll ein System entstehen, mit dem Künstler interaktive Installationen aufgrund der drahtlosen Technologie über größere Entfernungen aufbauen können. Dies ermöglicht z.B., dass die Reaktion auf einen Sensorwert bei einem entfernten Aktor auftritt. Außerdem soll die Funktechnik dazu dienen, die Anzahl der Kabel zu reduzieren, so dass die Technik in den Hintergrund treten kann. Da es sich hierbei um Kunstprojekte handelt, gelten andere Voraussetzungen als bei z.B. sicherheitskritischen Anwendungsfeldern wie der Medizintechnik.

4.2 Anwendungsfeld interaktive Installationen

Eine interaktive Installation ist eine computerbasierte Kunstinstallation, die mit dem Benutzer interagiert. Dies kann realisiert werden, indem der Benutzer Teil der Installation wird oder seine Aktionen von außen die Installation beeinflussen. interaktive Installationen können z.B. eine Kunstaussstellung in einer Galerie oder auch webbasiert sein. Diese Art von Installation tauchte das erste Mal Ende der 1980er Jahre auf[54][10].

Eingesetzt werden sie als reine Kunstobjekte aber z.B. auch im Marketing, in der Werbung bzw. für PR Aktionen oder für soziale Experimente. Eines der bekanntesten Beispiele ist das *Projekt Blinkenlights* aus dem Chaos Computer Club Umfeld[41]. Es handelt sich dabei um eine Lichtinstallation an verschiedenen Gebäuden in Großstädten. Bei mehreren Ausstellungen wurden die Fassaden als Bildschirme genutzt, so dass ein Fenster ein Pixel darstellt. Für die ursprüngliche gleichnamige Installation im Jahre 2001 wurde das Haus des Lehrers in Berlin genutzt. Dieses zeigte von Nutzern per E-Mail eingesendete Animationen. Die Installation *Arcade* 2002 an der *Bibliothèque nationale de France* in Paris zeigte verschiedene spielbare Computerspiel-Klassiker wie Pong, Pac-Man oder Break Out[40].



Abbildung 4.1: Projekt Blinkenlights am Haus des Lehrers, Berlin [58]

Ein weiteres bekanntes Beispiel ist das *Piano Staircase* im Rahmen des *The Fun Theory Awards* [2], in dem in Stockholm untersucht wurde, ob Passanten lieber die mit Klaviertasten beklebten Treppe hochgehen oder die Rolltreppe nehmen, wenn das Treten auf einer Stufe einen Ton hervorruft. Dieses Projekt wurde von Volkswagen finanziert und auf die Videoplattform YouTube hochgeladen [62].



Abbildung 4.2: Piano Staircase von *The Fun Theory*, Stockholm[63]

Ein etwas älteres Projekt mit mehr als 10.000 Teilnehmern ist der Telegarden. Bei diesem Projekt handelt es sich um einen Roboterarm, um den herum ein Garten gestaltet wurde. Teilnehmer konnten diesen Arm über eine Website steuern und z.B. Blumen einpflanzen. Von 1995-2004 wurde aus dem ehemaligen Kunstprojekt eine Art soziales Netzwerk in dem eingebauten Chatraum. Es war nicht nur ein Kunstprojekt, sondern wurde auch sozialwissenschaftlich untersucht, da

sich im Laufe der Zeit interessante Muster im zwischenmenschlichen Verhalten - wie etwa das Territorialverhalten - bildeten[31].



Abbildung 4.3: Telegarden [23]

Technisch gesehen werden in den Installationen Sensoren und Aktoren verwendet, um die Umgebung bzw. den Benutzer zu erfassen und entsprechend zu reagieren.

4.3 Technische Anforderungen

Mit dem Wissen aus den Grundlagen und der Zielsetzung soll nun eine Anforderungsanalyse durchgeführt werden. Die verfügbaren Technologien werden dann im nächsten Kapitel aufgelistet und miteinander verglichen. Die Anwendungslogik braucht weiterhin ein Protokoll und Konventionen für Nachrichteninhalte, die ebenfalls in dieser Arbeit entworfen werden.

Kommunikation der Netzwerkknoten Es soll ein Netzwerk aufgebaut werden, das aus verschiedenen drahtlos miteinander kommunizierenden Knoten besteht. Das Netzwerk soll sich dabei ad hoc selbst organisieren. Es soll mit Standardhardware gearbeitet werden und diese fest oder ggf. mobil (batteriebetrieben) in eine interaktive Kunstinstallation verbaut werden können. Die Knoten sollen ihre gemessenen Daten untereinander austauschen können.

Benutzbarkeit durch Laien Da die Konzeption und der Aufbau interaktiver Kunstinstallationen nicht zum Fachbereich eines Informatikers gehört, soll der Aufbau dieses Netzes möglichst leicht verständlich und die Benutzung durch Nicht-Programmierer möglich sein. Grundkenntnisse in der Programmierung werden allerdings vorausgesetzt. Der Benutzer soll nur wenig Anwendungslogik bzw. lediglich die Implementierung der Sensoren und Aktoren übernehmen

und diese zusammen mit dem eigentlichen Quelltext über Namenskonventionen verbinden können. Eine Konfiguration des Netzwerkes an zentraler Stelle soll über Konfigurationsdateien möglich sein. Der komplette Quelltext soll dann mit Standardwerkzeugen auf die Knoten geladen werden können.

Skalierbarkeit Die Installation kann sich aufgrund der Beschaffenheit des Netzes über ein größeres Gebiet ausdehnen, z.B. über mehrere Räume. Dies soll dem Benutzer größtmögliche Freiheit beim Erstellen der interaktiven Installation geben.

Konfiguration der Kommunikation und Knoten In der Initialisierungsphase soll der Anwender komfortabel zentral festlegen können, welche Knoten miteinander kommunizieren. Dies soll auch vorher in einer Datei festgelegt werden können, die in der Initialisierungsphase eingelesen wird. Kommen neue Sensor- bzw. Aktortypen hinzu, soll lediglich eine Konfigurationsdatei bzw. eine Funktion geschrieben werden, um die Daten auszulesen bzw. umzusetzen.

Beschaffenheit der Knoten Sensoren an den Netzwerkknoten sollen Daten erfassen können. Aktoren an den Netzwerkknoten sollen auf Input reagieren können, unabhängig davon, ob dieser von einem Sensor oder Aktor kommt. Weiterhin soll eine eindeutige Identifikation der Knotenpunkte möglich sein. Sie können eine unterschiedliche Intelligenz aufweisen, das Netzwerk kann also heterogen sein.

Hybride Sensor- und Aktorknoten In dem Anwendungsgebiet interaktive Installationen kann es von Vorteil sein, wenn Sensoren und Aktoren auf einem Knoten installiert sind, obwohl laut den meisten Definitionen Sensoren kleine, kostengünstige, mobile Knoten mit eingeschränkter Leistungsfähigkeit sind und Aktoren mehr Rechenleistung haben und meistens an eine feste Energieversorgung angeschlossen sind. Vorgesehen ist, dass die Knoten in diesem Netzwerk meistens ohnehin an eine feste Energieversorgung angeschlossen sind und nicht über Batterie oder *energy harvesting* betrieben werden. Diese Art von Hybridknoten erfordert vom Netzwerkknoten mehr Rechenleistung als vielleicht bei einem reinen Sensorknoten erforderlich wäre. Aufgrund des Aufbaus interaktiver Installationen kann es, z.B. aus Platzgründen, notwendig sein, dass sich sowohl Sensor als auch Aktor auf demselben Knoten befinden.

Multihopping Nicht jeder Netzwerkknoten muss mit einer zentralen Stelle verbunden sein, da Daten über mehrere Knoten weitergeleitet werden.

Sonstiges Wie bei jedem Projekt sollen auch hier die Kosten minimal gehalten werden.

4.4 Fehlerbehandlung

In professionell betriebenen Netzwerken müsste sichergestellt werden, wie auf Fehler reagiert wird. Außerdem müssten die Knoten in einigen Szenarien bis zu einem bestimmten Grad redundant ausgelegt sein, um einen fehlerfreien Betrieb sicherzustellen. Diese Punkte werden in dieser Arbeit aufgrund der Komplexität außen vor gelassen. Das System darf allerdings nicht abstürzen. Fehler können zwar erkannt, aber nicht behandelt werden. Das dynamische Entfernen und Hinzufügen von Knoten während des laufenden Betriebs soll möglich sein, ohne dass das Netzwerk zusammenbricht oder die Software auf Knoten abstürzt. Eine Kommunikation wird jedoch nicht möglich sein, da wie schon erwähnt die Konfiguration in der Initialisierungsphase stattfindet. Wenn ein Knoten entfernt wird, senden die dazugehörigen Knoten weiterhin Daten.

4.5 Abgrenzungen und Einschränkungen

Da das Anwendungsgebiet interaktive Installationen nicht sicherheitskritisch ist, gelten andere Voraussetzungen als bei anderen Sensor-Aktor-Netzwerken. Im Folgenden wird diese Arbeit abgegrenzt und Unterschiede erläutert.

4.5.1 Aktualität der Daten

In Sensor-Aktor-Netzwerken muss normalerweise sichergestellt sein, dass die Daten zum Zeitpunkt des Eintreffens noch aktuell sind[52]. Dies wird aufgrund des großen Implementierungsaufwands und der nicht kritischen Anwendung ebenfalls vernachlässigt.

4.5.2 Zustellungsbestätigungen

Nach [14] sollte sichergestellt werden, dass Nachrichten angekommen sind. Da dies zu erhöhtem Kommunikationsaufwand sowie Berechnungsaufwand in den Knoten führen würde und die Anwendung als nicht sicherheitskritisch identifiziert wurde, werden die Daten asynchron, also ohne eine Rückmeldung (ACK) zu erwarten, verschickt.

4.5.3 Ortsbestimmung

Neuere Sensorknoten können häufig bestimmen, an welchem Ort sie sich befinden. Damit sind nicht nur die GPS-Koordinaten gemeint, sondern auch die Positionen relativ zu anderen Sensorknoten, wenn mindestens ein Sensorknoten seine genaue Position kennt. Dies wird entweder über Matrizen bzw. Graphen oder z.B. „Fingerabdrücke“ des Hintergrundrauschens umgesetzt.

Dies ist sehr aufwendig und würde über den Umfang dieser Arbeit hinausgehen. Auch wäre die Hardware für GPS zu teuer.

4.6 Vergleichbare Arbeiten

Die folgenden Arbeiten haben Gemeinsamkeiten mit dieser Bachelorarbeit. Sie werden im Folgenden vorgestellt und abgegrenzt. Die vorgestellten Arbeiten ähneln dieser Arbeit jedoch nur in technologischer Hinsicht. Zwar werden ähnliche Technologien in interaktiven Installationen häufig benutzt, aber zu einer Verwendung von Sensor-Aktor Netzwerken gab es bis zum Zeitpunkt der Fertigstellung dieser Arbeit noch keine vergleichbaren Arbeiten. Daher baut diese Arbeit nicht direkt auf den folgenden Arbeiten auf.

- Die Bachelorarbeit von Marco Schneider zum Thema „Entwicklung und Realisierung eines Sensornetzwerkes für das Living Place Hamburg“ an der HAW Hamburg.[51].
- Die Diplomarbeit von Joakim Eriksson zum Thema „Detailed Simulation of Heterogeneous Wireless Sensor Networks“ an der *Uppsala universitet* in Schweden.[17]
- Die Dissertation von Andreas Kuntz zum Thema „Dienstbasierte Kommunikation über unzuverlässige drahtlose Verbindungen für selbstorganisierende Sensor-Aktor-Netze“ am Karlsruhe Institute of Technology.[37]
- Die Masterarbeit von Alexander Pautz zum Thema „Kabellose, stromsparende Sensornetzwerke im Bereich Ambient Intelligence“ an der HAW Hamburg.[43]

Die Bachelorarbeit von Marco Schneider enthält die Analyse bzw. Entwicklung eines Sensornetzwerks für eine intelligente Wohnung (Smart Home). Jedoch werden Aktoren nur über die schon vorhandene Infrastruktur des Smart Home „Living Place“ angesprochen. Aus Sicht der Sensorknoten gibt es nur den Aktor *Message Queue*. In Rahmen der Arbeit wurden die Daten ausschließlich an diese Infrastruktur weitergegeben, eine direkte Einbindung von Aktoren fand nicht statt. Die verwendeten Technologien wie etwa Arduino und das ZigBee Protokoll sind der in dieser Arbeit verwendeten Technik jedoch sehr ähnlich.

Die Diplomarbeit von Joakim Erikson setzt sich nur mit dem Thema Sensornetzwerke auseinander, zeigt aber, wie man generell diese Art von Netzwerken simulieren kann und wo die Grenzen liegen. Der Schwerpunkt liegt hierbei auf der Simulation und keiner konkreten Implementierung. Die Dissertation von Andreas Kuntz befasst sich mit Sensor-Aktor-Netzwerken und geht dabei insbesondere auf den Aspekt der Unzuverlässigkeit der Verbindungen ein. Außerdem wird die

von ihm entwickelte dienstbasierte Architektur *ServiceCast* vorgestellt und erläutert. Aufgrund der Art der Arbeit ist sie viel umfangreicher, die Grundlagen sind jedoch die Gleichen.

Die Masterarbeit von Alexander Pautz ähnelt der Bachelorarbeit von Marco Schneider sehr, da sie beide für dieselbe intelligente Wohnung, das „Living Place“ der HAW Hamburg, geschrieben wurden. Auch Pautz baut ein Sensornetzwerk zur Überwachung ähnlicher Sensorwerte auf. Jedoch werden statt Arduinos eigens entwickelte Platinen verwendet. Auch hier werden die Daten an die Infrastruktur weitergegeben, worauf über einen Server eine Aktion veranlasst werden kann.

4.7 Zusammenfassung

Im ersten Abschnitt dieses Kapitels wird das Ziel dieser Arbeit klar formuliert. Da das zu entwickelnde System für das Anwendungsfeld *interaktive Installationen* konzipiert werden soll, wird letztgenanntes definiert und anhand von Beispielen erläutert. Weiterhin enthält dieses Kapitel alle technischen Forderungen, die im Kapitel Design erfüllt werden sollen. Dies umfasst auch die Fehlerbehandlung und Abgrenzungen zu Anforderungen von ähnlichen Systemen. Außerdem wird diese Arbeit noch mit anderen akademischen Arbeiten verglichen bzw. von diesen abgegrenzt.

5 Design

In diesem Kapitel werden konkrete Hardware-Plattformen und Funkprotokolle zur Umsetzung der im letzten Kapitel definierten Zielen vorgestellt und ausgewählt. Weiterhin wird ein eigenes Protokoll für das spezielle Anwendungsgebiet *interaktive Installationen* entwickelt. Dies soll auf dem Funkprotokoll aufsetzen. Außerdem werden bestimmte Konventionen zur Kommunikation vereinbart.

5.1 Konzept

In diesem Kapitel soll ein Sensor-Aktor-Netzwerk entworfen werden, das den Benutzern möglichst viel Komplexität bei der Programmierung bzw. der Konfiguration abnimmt. Das Netzwerk soll aus zwei Komponenten bestehen: Sensor-, Aktor- und Sensor-Aktor-Knoten, die Werte messen bzw. Geräte ansteuern und einem zentralen Knoten, über den dieses Netz konfiguriert werden kann. Die Knoten sollen drahtlos miteinander kommunizieren können. Durch die Protokollwahl soll möglichst viel Implementierungsaufwand und Konfigurationsaufwand gespart werden sowie der Aspekt der Selbstorganisation des Netzes ausgelagert werden.

Die Knoten sollen direkt miteinander kommunizieren können, was dem Benutzer jedoch verborgen bleiben kann. Es ist jedoch ein wichtiges Detail, da sie auch die Daten von anderen Knoten weiterreichen können, so dass der Benutzer die Möglichkeit hat, Installationen über einen größeren Raum zu verteilen.

Über den zentralen Knoten, die *Zentrale*, soll der Benutzer komfortabel festlegen, welcher Knoten an welchen Knoten Daten sendet und von welchem angeschlossenen Gerät diese Daten stammen. Dies ist die zentrale Steuerungsfunktion in diesem Projekt. Über eine leicht verständliche Konfigurationsdatei soll diese Funktionalität konfiguriert werden, so dass der Benutzer möglichst wenig mit den internen Details der ausgewählten Hard- und Software oder des Protokolls arbeiten muss. Ein minimales Maß an Verständnis der eingesetzten Technologie ist jedoch nötig. Durch die Wahl häufig bekannter Werkzeuge mit intuitiver grafischer Oberfläche kann aber auch hier den Benutzern entgegen gekommen werden.

Weiterhin lässt sich nicht vermeiden, dass die Benutzer für neue Kunst-Installationen mit neuer Hardware die Funktionalität dieser selbst implementieren müssen. Es soll jedoch durch bestimmte

Konventionen möglichst einfach möglich sein, neue Implementierung von Hardware in eine Vorlage zu integrieren. Auch welche Daten Sensoren und Aktoren senden, z.B. direkte Messdaten oder aufbereitete Daten in einem bestimmtem Spektrum und wie die Knoten darauf reagieren, muss der Benutzer selbst festlegen, da dies für jede Installation anders ist. Es soll jedoch ein festes Format existieren, an dem sich der Benutzer orientieren kann, so dass es zu keinen Missverständnissen wie z.B. bei der Byte-Reihenfolge (Endianess) oder ähnlichen internen Problemen kommt. Auch die Knoten brauchen ein minimales Wissen über die angeschlossenen Sensoren und Aktoren, welches in einer extra Konfigurationsdatei zusammen mit der Implementierung dieser festgelegt wird. So muss der Benutzer nicht alle mitgelieferten Dateien nach bestimmten Variablen und Funktionen durchsuchen, was aufgrund der Größe des Projektes schnell zu unübersichtlich werden würde.

Wenn das Netzwerk gestartet wird, teilt die Zentrale gemäß der vom Nutzer erstellten Konfigurationsdatei allen anderen Knoten mit, wohin sie welche Daten senden sollen. Die Knoten speichern diese empfangene Konfiguration in einem temporären Speicher, so dass sie jedes Mal neu gesendet werden muss. Sind alle diese Nachrichten verschickt, gibt die Zentrale den Knoten ein Signal zum Starten und die Knoten fangen an, entsprechend der gerade empfangenen Konfiguration Daten zu senden und zu empfangen. Sind alle Konfigurationsdateien geschrieben, ist keine weitere Konfiguration seitens des Benutzers mehr erforderlich. Das an die Zentrale angeschlossene Netzwerkmodul übernimmt nur noch interne Aufgaben des Protokolls und nimmt nicht aktiv am Sensor-Aktor-Netzwerk teil. Eine grafische Repräsentation findet sich in Abb. 5.1.

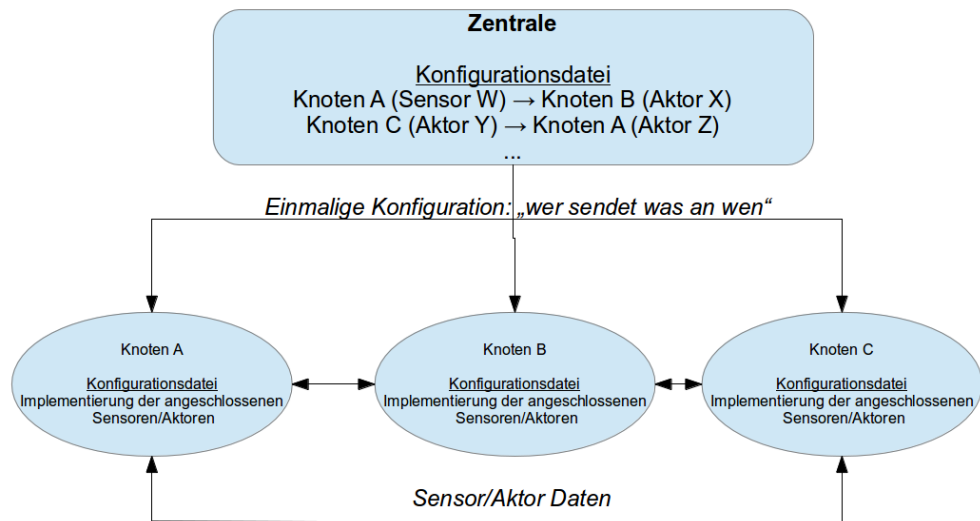


Abbildung 5.1: Konzept

5.2 Hardware

5.2.1 Arduino

Arduino ist eine quelloffene Physical-Computing-Plattform, das heißt, dass sie aus zwei Komponenten besteht: Die Hardware besteht aus einem I/O-Board mit digitalen und analogen Eingängen und (bis auf wenige Ausnahmen) einem Atmel AVR Mikrocontroller. Die Software ist eine Integrierte Entwicklungsumgebung (IDE), in der sich der Mikrocontroller programmieren lässt. Da die Hardware unter der *Creative Commons License* und die Software unter der *GNU Public License* steht, können Interessierte sowohl die Software als auch die Hardware erweitern, ohne dafür Lizenzgebühren zahlen zu müssen.

Die Plattform wurde 2005 am *Interaction Design Institute* in Ivrea, Italien entworfen und hat sich seitdem zu einer beliebten Plattform für Kunstinstallationen, gerade im Hochschulbereich, entwickelt. Dabei liegt der Fokus, auch dank der einfachen, an C/C++ angelehnten Programmiersprache *Processing* darauf, auch Nicht-Informatikern das Programmieren zu ermöglichen, ohne viele komplizierte Programmierkenntnisse erlangen zu müssen.

Dies zeigt sich zum Beispiel im einfachen Aufbau des Codes, in dem nur die zwei Funktionen `setup()` und `loop()` beschrieben werden müssen. Weitere Funktionen können definiert und von diesen beiden Funktionen aufgerufen werden. Beispielsweise ist die Konfiguration der Input bzw. Output Pins sehr einfach gehalten und für Laien gut lesbar.

Mit dem Befehl `pinMode(13, OUTPUT)` wird der 13. Pin als Output definiert. Im Gegensatz dazu ist es auf anderen Plattformen bzw. Mikrocontrollern aufgrund des Schreibens von vielen Registern zumindest für Laien nur schwer nachvollziehbar.

Ein weiterer Vorteil ist der eingebaute In-System-Programmierer (ISP), der einen externen Programmierer wie z.B. bei der PICmicro Mikrocontrollerfamilie überflüssig macht. Über den ISP wird ein minimales System gebootet, mit dem sich der Arduino über USB flashen lässt.

Bei der Spannungsversorgung bieten sich zwei Möglichkeiten. Über die USB-Schnittstelle lässt sich der Arduino nicht nur programmieren, sondern auch an die Spannungsversorgung anschließen. Zusätzlich gibt es noch eine standardisierte Buchse, mit dem sich der Arduino mit höheren Spannungen von 6-20V versorgen lässt. Allerdings ist es aufgrund der verbauten Hardware ratsam, nicht mehr als 9V zu verwenden, da der Arduino, je nach Umgebung, überhitzen kann. Eine 9V-Blockbatterie ist somit eine gute mobile Möglichkeit. Für die Steuerung von externer Hardware kann eine externe Spannungsversorgung in Verbindung mit einem Transistor verwendet werden.

Jeder Arduino besitzt eine festgelegte Anzahl an Pins, die analoge und digitale Signale empfangen bzw. ausgeben können. Dabei gibt es häufig auch Pins mit extra Funktionen z.B. eine Referenzspannung, ein Pin zum Durchschleifen der Versorgungsspannung oder Pulsweitenmodulation (PWM).

Arduino Boards

Im Folgenden werden drei Arduino Boards vorgestellt. Es existieren jedoch zahlreiche weitere Boards verschiedener Größen und Ausführungen.

Uno Der Arduino Uno ist eine der beliebtesten Ausführungen der Arduino Plattform und inzwischen schon in der dritten Revision erschienen. Er ist mit einem ATmega328P 16 MHz Prozessor, 32 kB Flash, 1 kB EEPROM und 2 kB SRAM ausgestattet. Weiterhin verfügt er über 14 digitale I/O Pins, von denen 6 PWM-fähig sind, und weiteren 6 analogen Input Pins.

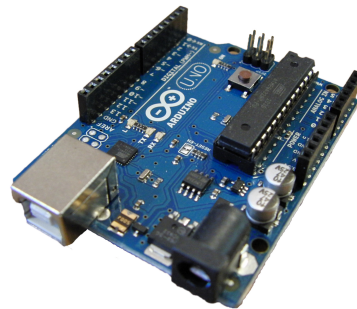


Abbildung 5.2: Arduino Uno [28]

Due Der Arduino Due ist insofern eine Besonderheit, als dass er als eine der wenigen Arduino Boards mit einem ARM-Prozessor (ARM Cortex M3) ausgestattet ist. Mit 84 MHz ist er weitaus schneller als alle anderen Modelle und besitzt mit 512 kB Flash und 96 kB SRAM auch wesentlich mehr Speicher. Auch die 54 (12 PWM-fähig) digitalen I/O Pins und die 12 analogen Eingänge übersteigt die Leistungsfähigkeit der anderen Modelle bei weitem.

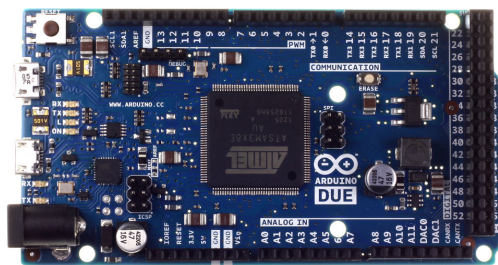


Abbildung 5.3: Arduino Due [3]

Mega Der Arduino Mega ist die der Name vermuten lässt eine der größten Ausführungen der ATmega basierten Arduinos. Zwar ist der ATmega2560 Prozessor mit 16 MHz ebenso schnell wie z.B. der Uno, allerdings hat er mit 256 kB Flash, 4 kB EEPROM und 8 kB SRAM wesentlich mehr Speicher. Um diesen sinnvoll füllen zu können, besitzt er außerdem 54 digitale Ein- und Ausgänge (14 PWM-fähig) sowie 16 analoge Eingänge.

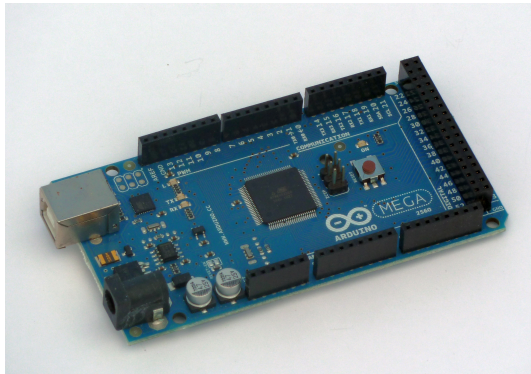


Abbildung 5.4: Arduino Mega [13]

Inoffizielle Klone Wie schon erwähnt, ist sowohl das Hardware Design als auch die Software unter freien Lizenzen veröffentlicht, was mehreren Herstellern ermöglichte, teilweise günstigere, Arduino-kompatible Hardware herzustellen. Diese kann hardwarekompatibel, softwarekompatibel oder beides sein. Der Begriff „Arduino“ ist jedoch als Marke eingetragen und so enden die geklonten Produkte häufig nur mit dem Suffix *-ino* oder etwas Vergleichbarem. Als bekanntester Arduino Ableger ist die Freeduino-Reihe erwähnenswert[21].

Shields

Zusätzlich zu den Arduinos selbst gibt es noch Aufsteckmodule, die die Funktionalität erweitern. Diese werden Shields genannt und sind im Grunde Breakout Boards, die mit den Pins direkt auf Arduinos passen. Damit wird vermieden, dass Komponenten über Kabel z.B. auf Steckplatinen verbunden werden müssen. Da es aufgrund der quelloffenen Natur der Plattform oft neue Komponenten gibt, werden hier nur beispielhaft einige Shields vorgestellt.

Ethernet Für viele Anwendungen kann es nützlich sein, nicht nur Daten zu sammeln und darauf zu reagieren, sondern auch weiterzugeben, vor allem ins lokale Netzwerk oder Internet (s. Kap 3.2.2 zu Internet of Things). Dafür gibt es das Ethernet Shield (Abb. 5.5), welches mit der Ethernet Bibliothek den Anschluss mit einem Netzkabel ermöglicht.

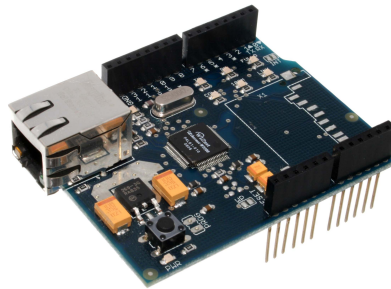


Abbildung 5.5: Arduino Ethernet Shield [6]

Motor Ein Motorshield sorgt für zusätzliche Anschlussmöglichkeiten und Treiber für z.B. Gleichstrommotoren oder Schrittmotoren. Da hier größere Ströme fließen können, ist es wichtig, den Mikrocontroller bzw. Motor bei Überhitzung automatisch abzuschalten.

Erweiterungen

Natürlich können Interessierte auch Shields bzw. eigene Hardware herstellen und an Arduinos anschließen, Shields sollen dies lediglich vereinfachen.

5.2.2 Weitere Hardware

Raspberry Pi

Der Raspberry Pi ist ein kreditkartengroßer single-board Computer. Er besitzt eine ARM CPU mit 700 MHz, 512 MB RAM, einen HDMI Ausgang und 2 USB Ports. Die CPU kann aber bis auf 1 GHz übertaktet werden. Das Betriebssystem wird über eine SD-Karte im internen SD-Karten Slot gestartet. Mit einem Verbrauch von 3,5W ist er sehr energieeffizient.

Mit einem angepeilten Preis von 25 US\$ sollte das Forschungsprojekt der Raspberry Pi Foundation vor allem finanziell schwache Familien und Schulen unterstützen, Kindern das Programmieren beizubringen. Er wird jedoch häufig in der Hausautomatisierung, zum Basteln, als Homeserver oder als Media-Center (z.B. mit XBMC[65]) eingesetzt. Vor allem in der Do it yourself Community im Internet hat er großen Anklang gefunden. So gibt es zahlreiche Blogs und Foren mit Anleitungen und Tipps. Man kann verschiedene Linux-Distributionen und sogar BSDs auf der SD-Karte installieren. Die bekannteste Linux Distribution heißt Raspbian und ist ein Derivat von Debian.

Der Raspberry Pi ist trotz seiner Größe ein vollwertiger Rechner. Bei Bedarf gibt es Plastikgehäuse in zahlreichen Formen und für unterschiedliche Zwecke. Durch die zwei USB Ports lässt sich ggf. fehlende Funktionalität meistens nachrüsten.



Abbildung 5.6: Raspberry Pi [30]

5.2.3 Auswahl

Arduino

Der Einfachheit halber soll hier mit Shields gearbeitet werden. Demzufolge stehen nur zwei Arduino Typen zur Verfügung: Uno und Mega. Der Mega verfügt zwar sowohl über mehr Speicher als auch über mehr I/O-Ports, ist aber für diese Aufgabe nicht zwingend nötig. Da unter anderem die Kosten berücksichtigt werden sollen, wird sich hier für den Uno entschieden. Eine spätere Umstellung auf den Mega, z.B. falls die Ports vom Arduino Uno nicht mehr ausreichen, ist unproblematisch.

Weitere Hardware

Für Knoten mit komplexeren Aufgaben, z.B. der in Kapitel 5.1 erwähnten Zentrale, soll ein vollwertiger Rechner und kein Mikrocontroller verwendet werden. Außerdem soll dieser zur Konfiguration über das Netzwerk erreichbar sein. Theoretisch wäre auch ein Laptop denkbar, dieser lässt sich allerdings schlechter in interaktive Installationen (s. Kap. 4.3) integrieren. Deshalb wird sich hier für den vorgeschlagenen Mini-Rechner Raspberry Pi entschieden. Über USB kann ggf. WLAN nachgerüstet werden. Es gibt einige Konkurrenzprodukte, die sich am Raspberry Pi orientieren, in dieser Arbeit wird aber die originale Version des Raspberry Pi genutzt, da es für diese diverse Anleitungen gibt und er auf jeden Fall unterstützt wird.

5.3 Drahtlose Standards

Wie in der Analyse gefordert, soll das Netzwerk drahtlos sein. Dazu werden verschiedene Netzwerkprotokolle miteinander verglichen und das Beste ausgewählt. Viele Standards basieren auf den Entwicklungen des *Institute of Electrical and Electronics Engineers*, IEEE, genauer gesagt der IEEE 802. Da es bei weitem zu viele Netzwerkstandards gibt, die man hier theoretisch benutzen könnte, wird nur auf die Wichtigsten eingegangen. Einen Vergleich der Standards bezüglich Reichweite, Kosten und Energieverbrauch ist in Abb. 5.7 dargestellt. Die mit roten Punkten markierten Standards werden im Folgenden besprochen.

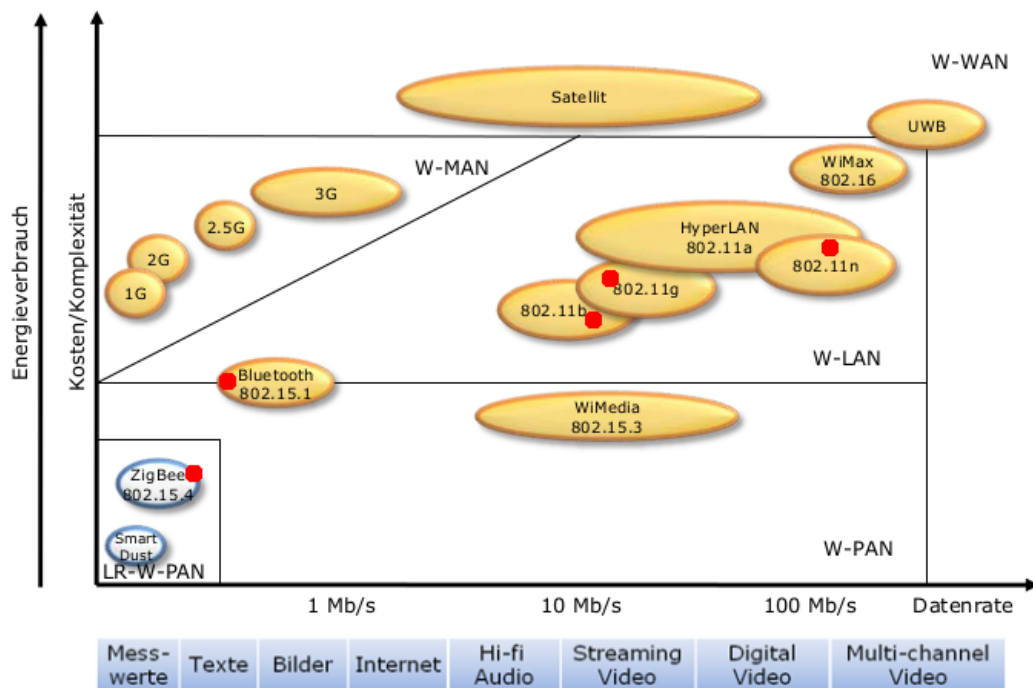


Abbildung 5.7: Energieverbrauch und Komplexität verschiedener Standards[46]

5.3.1 WLAN

Der WLAN Standard IEEE 802.11 wird häufig für die unterschiedlichsten Projekte verwendet. Meistens findet WLAN in größeren Geräten, wie etwa Laptops, Verwendung. Es gibt mehrere WLAN Standards, die jeweils mit einem kleinen Buchstaben bezeichnet werden, z.B. 802.11a für das 5 GHz Netz. Alle Standards laufen entweder im 2,4 GHz oder im 5 GHz Netz. Dabei wird das *Orthogonale Frequenzmultiplexverfahren* (OFDM) oder *Direct Sequence Spread Spectrum* (DSS)

benutzt, um die Signale zu übertragen.

WLAN wurde als drahtloser Ersatz für LAN für größere Umgebungen entwickelt. Im Gegensatz zu vergleichbaren Protokollen ist der Implementierungsaufwand relativ groß und das Protokoll sehr umfassend.

5.3.2 Bluetooth

Bluetooth ist der de-facto Standard für viele mobile Anwendungen. Ursprünglich wurde es 1994 von Ericsson als drahtlose Alternative zu RS232 entwickelt. Inzwischen wird das Protokoll von der *Bluetooth Special Interest Group* verwaltet und weiterentwickelt und ist in der siebten Version, Bluetooth 4.0 oder auch Bluetooth Low Energy (BLE) genannt. Bluetooth läuft mit ebenfalls 2,4GHz im lizenzfreien ISM-Band und benutzt das *Frequency Hopping Spread Spectrum* (FHSS) Verfahren.

Eine große Limitierung von Bluetooth ist die Anzahl möglicher Knoten. In einem Netzwerk darf es maximal 255 Knoten geben, wovon maximal acht gleichzeitig aktiv senden dürfen.

Häufige Anwendungen findet Bluetooth bei drahtlosen Headsets, Datenübertragung bei Mobiltelefonen, drahtlosen Mäusen und Tastaturen, Spielekonsolen (Playstation 3, Nintendo Wii) oder Dial-up Verbindungen, bei denen das mobile Gerät als Zugangspunkt benutzt wird.

5.3.3 ZigBee

ZigBee ist ein Standard für Funknetzwerke, der seit 1999 von der ZigBee Alliance – mehr als 230 Unternehmen – gegründet wurde. OSI Schicht 1 und 2 (PHY/MAC) basieren dabei auf dem IEEE 802.15.4 -Standard, welchen sog. *low-rate wireless personal area networks* (LR WPANs) bildet. Dieser Standard zeichnet sich besonders durch geringen Leistungsverbrauch und Komplexität, allerdings damit einhergehend auch geringere Datenübertragungsgeschwindigkeit und Reichweite aus. Das ZigBee Netzwerk wird selbstständig aufgebaut und übernimmt Funktionen wie Adressierung und Routing durch das *Ad-hoc On-demand Distance Vector* (AODV) Routing-Protokoll. Für den Anwender ist das Protokoll vergleichsweise schnell zu verstehen und gut konfigurierbar. Eine Verschlüsselung ist optional und ebenso leicht entschlüsselbar wie die WLAN Verschlüsselung WEP, da es ähnliche Fehler enthält. Eine Sicherheit der Daten kann ohne externe Verschlüsselung demnach nicht garantiert werden. Der schlanke Protokollaufbau, so fehlen z.B. Handshakes, kann sowohl ein Nachteil, als auch ein Vorteil sein. Die 64-Bit-Adressen erlauben mit 65536 möglichen Knoten ein sehr skalierbares Netzwerk.

In einem ZigBee Netzwerk gibt es die folgenden Gerätetypen:

Coordinator Der Coordinator baut das Netz auf und verwaltet die Adressen neuer Knoten. Es muss in jedem Netz genau einen Coordinator geben. Da ein ZigBee Netzwerk ohne diesen Coordinator nicht funktioniert, ist er der Single Point of Failure. Im IEEE 802.15.4 Standard heißt dieser Knoten *full functioning device* (FDD).

Router Der Router baut das Netzwerk mit auf, sendet Daten von anderen Knoten weiter und kann gleichzeitig als Endgerät auch selbst Daten senden. Es kann mehrere Router in einem Netzwerk geben. Fällt ein Router aus, werden die Daten über den nächsten Router geschickt. Im IEEE 802.15.4 Standard heißt dieser Knoten *reduced functioning device* (RDD).

Endgerät Das Endgerät ist immer an einem Router angemeldet und übernimmt sonst keine Aufgaben im Netzwerk. In bestimmten Situationen ist das Endgerät einem Router vorzuziehen. So braucht es aufgrund der reduzierten Komplexität und aufgrund eines möglichen Sleep Modus weniger Energie und wird z.B. bei batteriebetriebenen Netzwerkknoten angewendet. Sie müssen immer an einem Router oder dem Coordinator angemeldet sein, da diese die Adressvergabe und Routingfunktionen übernehmen.

5.3.4 Auswahl

WLAN ist für diese Zwecke völlig überdimensioniert und hat zu viel Overhead. Der WLAN Standard bietet entweder nur einen strikten Infrastrukturmodus mit einem zentralen Punkt oder eine strikte Punkt-zu-Punkt Verbindung an. Daher müsste auf der Applikationsschicht noch ein Mesh-Protokoll laufen, was allerdings mehr Implementierungsarbeit bedeutet. WLAN scheidet aus, da es Funkstandards gibt, die Meshnetzwerke direkt unterstützen.

Zwar mag es in vielen Situationen ausreichen, 255 Knotenpunkte zu haben, doch die Beschränkung von acht gleichzeitig sendenden Knoten ist ein Grund kein Bluetooth zu verwenden. In vielen komplexen Systemen wäre es möglich, die Grenze von 255 Geräten zu überschreiten, insbesondere ist es wahrscheinlich, dass acht Geräte gleichzeitig senden. Weiterhin ist Bluetooth z.B. aufgrund der vielen Profile ebenso wie WLAN überdimensioniert.

ZigBee hingegen ist mit 65536 möglichen Geräten sehr skalierbar. Die mangelnde Sicherheit ist kein Ausschlusskriterium, da man - wenn nötig - auch auf einer höheren Schicht noch Verschlüsselung implementieren könnte und sie für viele Anwendungen schlicht nicht nötig ist. Der einzige Nachteil ist, dass der Coordinator ein Single Point of Failure darstellt. ZigBee ist ein in der Praxis erprobtes Protokoll für Sensornetzwerke und soll nicht zuletzt deswegen hier eingesetzt werden.

5.4 Netzwerk

5.4.1 XBee

XBee sind Funkchips der Firma *Digi*. Für den ZigBee-Standard gibt es inzwischen drei verschiedene Versionen, zusätzlich zur ersten Serie. Die drei neueren Versionen unterstützen Mesh-Netzwerke. Es gibt XBee und XBee-Pro Chips, welche sich jedoch nur in der Ausgangsleistung unterscheiden. Bis auf wenige Chips, die auf 900 MHz funken, senden XBee Module im 2,4 GHz Bereich und sind damit im selben Frequenzbereich wie WLAN und Bluetooth.

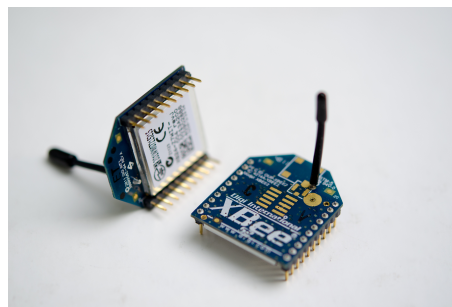


Abbildung 5.8: XBee Module [20]

Series 1

Diese XBee Module sind sehr einfach zu konfigurieren. Dafür beherrschen sie nur Punkt-zu-Punkt Kommunikation und können kein Mesh-Netzwerk bilden. Sie sind nicht kompatibel mit Series 2, unterstützen nicht das ZigBee Protokoll und werden hier nur der Vollständigkeit halber erwähnt.

Series 2

Der Begriff „Series 2“ ist häufig mehrmals besetzt. Man muss unterscheiden zwischen der ZNet 2.5, ZB und 2B Reihe. Die ZNet Reihe ist die alte Series 2 Reihe, allerdings hardwarekompatibel mit der aktuellen ZB-Reihe. Jedes ZNet Modul kann man somit einfach mit neuer Firmware bespielen. In den meisten aktuellen Publikationen wird die ZB Reihe einfach nur als „Series 2“ bezeichnet. Auf den neueren 2B-Modellen läuft zwar die ZB Firmware, die Hardware hingegen wurde verbessert[56].

5.4.2 Modulhardware

Die Zentrale braucht im Vergleich zu den restlichen Knotenpunkten mehr Rechenleistung und wird deswegen über USB und ein XBee Breakout Board an den Raspberry Pi angeschlossen,

vgl. Abb. 5.9. Dieses XBee Modul wird als dementsprechend als Coordinator konfiguriert. Die restlichen Knotenpunkte bestehen aus einem Arduino mit einem XBee Shield und Modul.



Abbildung 5.9: XBee USB Explorer[57]

5.4.3 Vorkonfiguration

XBee Module der Series 2 können in zwei Modi betrieben werden: AT und API. Im AT Modus können die Konfigurationsparameter der Module direkt über sog. AT-Commands aus dem *Hayes command set* eingestellt werden[19]. Dazu wird eine serielle Verbindung zu dem Modul aufgebaut oder ein Konfigurationsprogramm benutzt. Im API Modus kann das Modul über eine Software API angesteuert werden. Es gibt für viele Programmiersprachen und Plattformen quelloffene Bibliotheken, z.B. Java, Perl, Python, Arduino oder sogar JavaScript. Neue Module werden von der Firma *Digi* im AT Modus ausgeliefert.

AT Modus und API Modus

Im API Modus empfängt und verschickt das XBee Modul Pakete, im AT Modus hingegen werden einfach nur die Daten aus der seriellen Schnittstelle weitergeleitet. Der AT Modus wird deswegen auch „transparent Mode“ genannt. Dabei ist die Zieladresse über die AT Parameter DH und DL bestimmt, anstatt über eine frei definierbare Adresse. Des Weiteren wird über die Pakete eine Rückmeldung (ACK) geschickt, ob die Pakete angekommen sind, sowie eine Checksumme, um die Datenintegrität festzustellen. Falls AT Commands im API Modus gesendet werden müssen, gibt es die Möglichkeit sog. RemoteAT Commands zu senden [47].

Ansteuerung der XBee Module

Die XBee Module auf den Arduinos können mit der *xbee-arduino* [49] Bibliothek im API Modus angesteuert werden. Das zentrale XBee Modul am Raspberry Pi kann über die Java API *xbee-api* [48] angesteuert werden. Beide Bibliotheken sind unter der freien *GNU Public License v3* veröffentlicht.

5.4.4 Architektur

Es gibt einen zentralen Punkt („Zentrale“), von dem aus das Netzwerk initial konfiguriert wird. Hier wird festgelegt, welche Sensoren bzw. Aktoren miteinander verbunden werden. Die verarbeiteten Messdaten werden im Idealfall direkt von Knoten zu Knoten geschickt. Ein Aktor kann dabei auch Daten von mehreren Sensoren empfangen. Der Aktor sammelt die Daten und führt erst die gewünschte Aktion aus, wenn alle Daten vorhanden sind. Dies muss allerdings auf Anwenderseite implementiert werden. Der Rest der Netzwerkknoten sind Router bzw. Router, die als Endgeräte agieren. Ein Aktor kann auch an andere Aktoren Daten senden.

5.4.5 Konfiguration der Knoten

Es gibt eine globale Konfigurationsdatei pro Knoten, in der die folgenden Werte festgelegt werden:

1. Lokale 64-Bit-XBee-Adresse
2. Anzahl der Sensoren und Aktoren
3. Maximale Anzahl von Verbindungen von Sensoren (Source) und Aktoren (Source oder Sink), sog. Source-Sink Paare
4. Eine eindeutige ID sowie eine Typ ID für jeden Sensor und Aktor
5. Die Implementierung zum Ansteuern der angeschlossenen Sensoren und Aktoren inklusive der Ports am Arduino

Die Konfigurationsdatei muss von jedem Benutzer manuell bearbeitet werden, da sich diese Schritte nicht automatisieren lassen. Die Sensoren und Aktoren können frei ausgewählt werden, müssen aber von einem Arduino angesteuert werden können. Ein unerfahrener Benutzer muss lediglich den Sensor bzw. Aktor implementieren, sofern noch keine Beispielimplementierung vorhanden ist, diesen Code in die Konfigurationsdatei einfügen und auf die Arduinos flashen. Die eindeutige ID kann beliebig gewählt werden, für die Typ ID gibt es Vorgaben, damit sowohl in der Zentrale als auch auf den Knoten eine Typ ID auf denselben Typ verweist. Dies soll der einzige Punkt sein, an dem es nötig ist, dass Benutzer zur Konfiguration Code verändern müssen. Diese Funktionen haben einen festgelegten Namen, der im Hauptcode des Knotens referenziert wird. Ein Beispiel für eine vollständige Konfigurationsdatei befindet sich im Anhang [9.5.3](#).

Konfiguration in der Zentrale

JSON JSON (kurz für *JavaScript Object Notation*) ist ein effizientes Datenformat zum Speichern und Austauschen von Key-Value Paaren und wird vor allem in Netzwerken als eine Alternative zu

XML benutzt. Wie der Name vermuten lässt, kommt es zwar ursprünglich von der Skriptsprache JavaScript, jedoch existieren für praktisch alle Programmiersprachen Parser, so dass es sich gut plattform- und sprachübergreifend einsetzen lässt.

Nummerierung der Knoten Zur besseren Wiedererkennung werden die Knoten bzw. ihre XBee Adressen fortlaufend im JSON Format in einer Textdatei nummeriert. Die Adressen sind auf den XBee Modulen aufgeklebt. Dies muss während der gesamten Installation nur einmal gemacht werden. Diese Nummern werden später in der Zentrale mit den XBee Adressen angezeigt, so dass es einfach zu erkennen ist, um welchen Knoten es sich handelt. Ein Beispiel hierfür findet sich in Anhang [9.5.1](#).

Konfiguration der Source-Sink Paare Es besteht die Möglichkeit über eine Konfigurationsdatei die Source-Sink Paare vorher festzulegen oder sie manuell einzeln während der Initialisierungsphase einzugeben. Dies muss in der Zentrale beim Start als Parameter übergeben werden. Die manuelle Eingabe umfasst für jedes Paar eine Abfrage der eindeutigen ID aus Kap. [5.4.5](#) und der XBee Adresse des sendenden Knoten (Source) und der eindeutigen ID sowie der XBee Adresse des empfangenden Knotens (Sink). Alternativ sind diese Informationen schon im JSON Format in einer Konfigurationsdatei eingetragen. Ein Beispiel hierfür findet sich in Anhang [9.5.2](#).

5.4.6 Ablauf

Nach der Konfiguration ist das grundlegende Prinzip *if-this-then-that* oder auch *Reiz-Reaktion*; wenn ein Sensor einen bestimmten Wert misst, gibt der Knoten diesen Wert an einen oder mehrere Knoten weiter und die Aktoren reagieren entsprechend darauf und können die Daten ebenfalls an weitere Aktoren weiterleiten. Die folgenden Diagramme verdeutlichen den Ablauf von sowohl Zentrale als auch Knoten.

Es ist wichtig darauf zu achten, dass erst die Zentrale eingeschaltet wird und dann die Knoten, da die Module sich über das ZigBee Protokoll bei dem Coordinator Modul anmelden. Mehr zur Reihenfolge findet sich in Kapitel [6.4.3](#).

Ablaufdiagramm Zentrale

Die Zentrale wartet zu Beginn auf das erste Paket von einem Knoten. Sobald das erste Paket eingetroffen ist, wird ein Counter im 1 Sekunden Takt hochgezählt bis zu einem Maximum. Ist diese erreicht, wird davon ausgegangen, dass alle Pakete eingetroffen sind. Anschließend werden entweder manuell Source-Sink Paare erstellt oder eine Konfigurationsdatei eingelesen, aufgrund

der diese Paare erstellt werden. Die Source-Sink Paare werden den entsprechenden Knoten geschickt. Zum Schluss wird eine Konstante gesendet um anzudeuten, dass die Initialisierungsphase beendet ist und die Knoten mit dem Messen anfangen sollen.

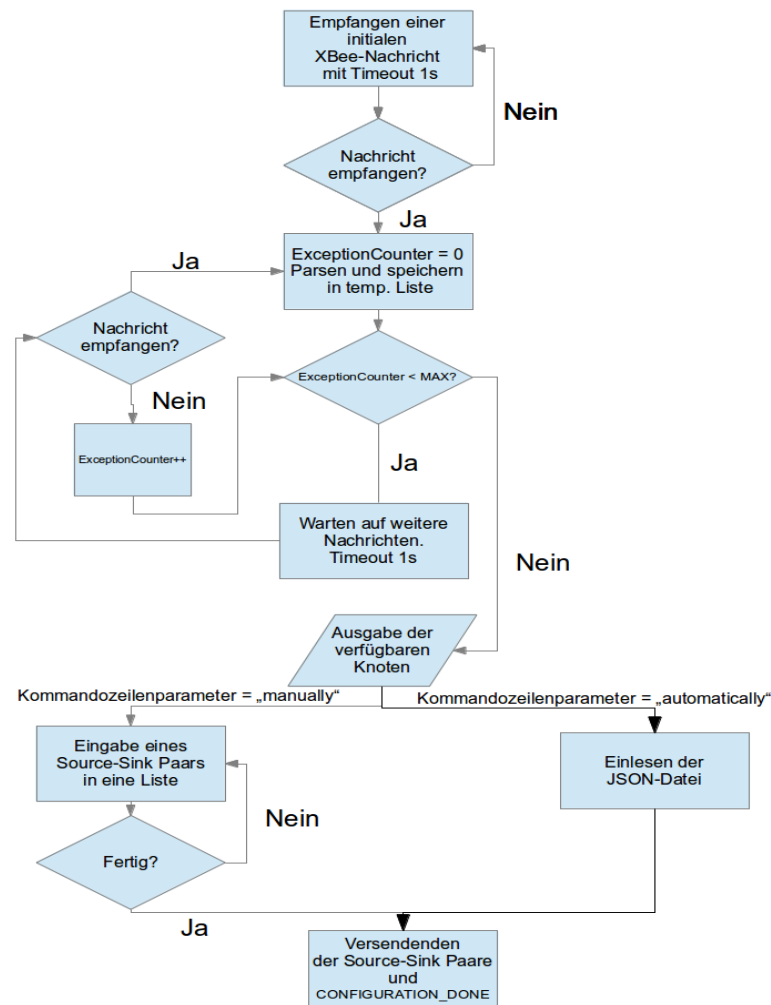


Abbildung 5.10: Ablaufdiagramm der Zentrale

FSM und Ablaufdiagramme der Knoten

Die Implementierung der Knoten soll als Endlicher Automat (engl. *Finite State Machine*, FSM) erfolgen. Dabei gibt es im Quelltext einen Sensorteil und einen Aktorteil, die auf jedem Knoten nacheinander ausgeführt werden.

Im Grunde haben beide nur zwei Phasen: Eine Aufbauphase (INIT bzw. NOCONFIGYET) und eine Betriebsphase (RUNNING). Nach dem einmaligen Aussenden der angeschlossenen Geräte an die Zentrale geht der Sensorteil automatisch in NOCONFIGYET über. Wenn ein bestimmtes Signal von der Zentrale empfangen wird, geht der Sensorteil in den Zustand RUNNING. Erst dann folgt der Aktorteil dem Sensorteil und geht ebenfalls in den RUNNING Zustand über. Ein genauer Ablauf wird in den dazugehörigen Ablaufdiagrammen in Abb. 5.13 und 5.14 beschrieben.

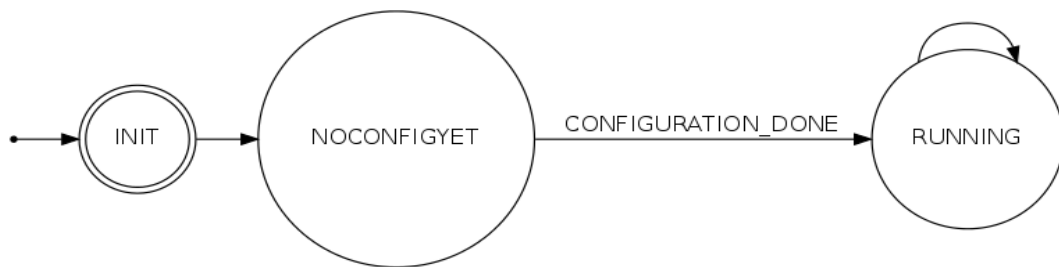


Abbildung 5.11: FSM Sensor



Abbildung 5.12: FSM Aktor

Der Sensorteil übernimmt die Initialisierung für den gesamten Knoten. Nach dem Aussenden der Initialisierungspakete wird im 1 Sekunden Takt auf neue Pakete gewartet. Die Pakete sind in Kapitel 5.4.7 ausführlich erklärt. Wird ein Paket empfangen, wird entweder eine LED zur Identifikation des Knotens eingeschaltet (s. Kapitel 5.4.7), ein neues Source-Sink Paar in die Konfiguration eingetragen oder der Zustand in RUNNING gewechselt. Sollte ein unerwartetes Paket empfangen werden, wird eine Fehlermeldung auf der seriellen Konsole ausgegeben. Im RUNNING Zustand wird ein Wert gemessen und eine Nachricht an den konfigurierten Knoten gesendet.

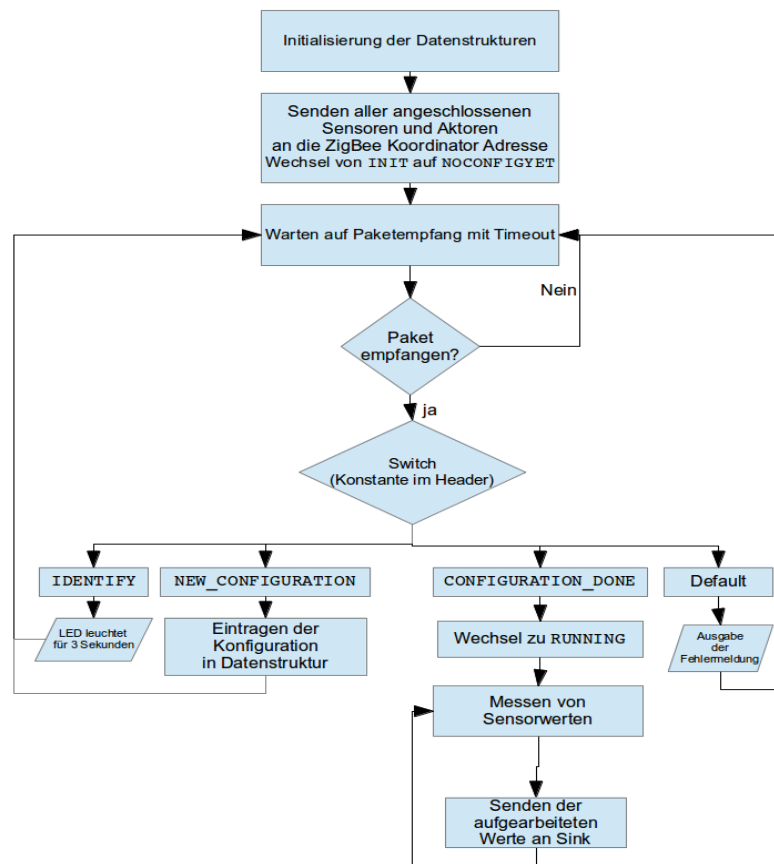


Abbildung 5.13: Ablaufdiagramm eines Knoten, Sensorteil

Da der Aktorzustand dem Sensorzustand folgt, wartet der Aktorteil so lange, bis der Sensorteil anfängt, Daten zu messen. Wird im RUNNING Zustand ein Paket empfangen, wird der entsprechende Aktor betätigt und ggf. die Daten weiter an andere Aktoren gesendet.

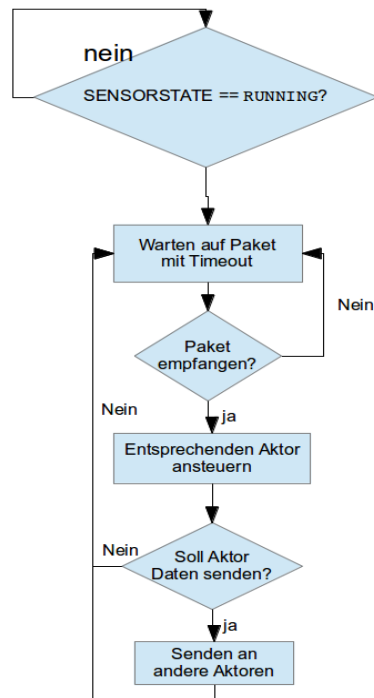


Abbildung 5.14: Ablaufdiagramm eines Knoten, Aktorteil

Ablauf in Worten

Die Konfiguration soll remote erfolgen.

1. Die Zentrale wird als Erstes angeschlossen und übernimmt die Coordinator Funktion im Netzwerk.
2. Die anderen Knotenpunkte werden angeschlossen und melden sich automatisch über die ZigBee-Coordinatoradresse (0x0000 0000 0000 0000) bei der Zentrale an. Dabei übertragen sie, welche Sensoren und Aktoren sie angeschlossen haben.

3. Die XBee Adressen werden aus den empfangenen Paketen extrahiert, mit den Sensoren-IDs bzw. Aktoren-IDs in der Zentrale in einer Liste gespeichert und an den Benutzer ausgegeben.
4. Zu den Aktoren werden entsprechende Sensoren bzw. Aktoren ausgewählt bzw. es wird je nach Einstellung eine Konfigurationsdatei mit den Source-Sink Paaren eingelesen. An jeden Knoten wird eine Nachricht mit der Zieladresse und Ziel-ID für die Sensor bzw. Aktordaten gesendet.
5. Sobald eine solche Nachricht vom Knoten empfangen wurde, wird geprüft, ob der Aktor lokal vorhanden ist. Die Zieladresse und Ziel-ID werden in eine Datenstruktur eingetragen und der Knoten beginnt mit dem Messen der Daten.
6. Wurde ein Wert gemessen, wird er an den entsprechenden Aktor gesendet. Es erfolgt also ein periodisches Senden, welches ggf. in der Implementierung der Sensoren bzw. Aktoren verzögert werden kann. Ist der Aktor lokal vorhanden, wird der Wert sofort an den Aktor weitergereicht.

5.4.7 Protokoll

API Paketaufbau

Das hier benutzte Protokoll ist ZigBee im API Modus. Die Übertragung der Sensor bzw. Aktorwerte erfolgt in der Payload. Die Pakete sind folgendermaßen aufgebaut[19].

Header Der Header ist folgendermaßen aufgebaut.

Name	Bedeutung	Offset
Start delimiter 0x7E	API Paket folgt	0
Length	Anzahl an Bytes zwischen length und der Checksumme	
Frame Type		3
Frame ID	Wenn 0, wird kein ACK gesendet	4
64-bit destination address	Ziel Adresse	5 (MSB)-12 (LSB)
16-bit destination network address	Ziel Adresse	13 (MSB)-14 (LSB)
Broadcast Radius	Maximale Anzahl an Sprüngen zwischen Knoten (0 für maximal mögliche Anzahl)	15
Options	Verschiedene Optionen z.B. ACK ausschalten, APS Verschlüsselung oder Timeout Länge	16

Payload Mit einem Offset von 17-24 darf die Payload maximal 84 Bytes betragen. Nur wenn Verschlüsselung (4 Byte) oder Source Routing (Standard ist AODV) aktiviert ist, kann sich diese Größe verändern.

Checksumme Die Checksumme (Offset 25) besteht aus einer 8 Byte Summe von Offset 3 zu diesem Byte.

Standard Einstellungen Alle Pakete werden ohne Verschlüsselung und mit deaktiviertem ACK verschickt.

FSM States

Nach Anforderung 4.3 sollen die Knoten identifiziert werden können. Wird das folgende Byte in den Zuständen NOCONFIGYET oder RUNNING empfangen, soll der Knoten sich angemessen für einen Zeitraum von 3 Sekunden identifizieren. Während dieser Zeit pausiert der Knoten.

Wert	Bedeutung	Konstantenname
0x0	Identifikation für 3 Sekunden	IDENTIFY

SENSORSTATE: INIT Der Knoten sendet Informationen über angeschlossene Sensoren und Aktoren an die Zentrale. Sie kann so eine Liste aller verfügbaren Sensoren und Aktoren anhand Absenderadressen und den IDs erstellen. Dabei werden alle TypIDs von 1 bis 127 Sensoren und alle TypIDs von 128 bis 255 Aktoren zugeordnet. Die ID 0 dient zur Fehlerbehandlung und wird deswegen nicht verwendet. Was für ein Sensortyp bzw. Aktortyp sich hinter der TypID verbirgt, wird in einer Konfigurationsdatei, die sowohl die Knoten als auch die Zentrale bekommt, in einem für Menschen lesbaren Format festgehalten. Dahinter kommt eine eindeutige ID, welche jedoch nur für den Knoten eindeutig ist. Als Maximum können damit 255 verschiedene Typen von Sensoren und Aktoren festgelegt werden sowie maximal verschiedene 255 Sensoren bzw. Aktoren pro Knoten. Dies lässt genug Raum für eventuelle Erweiterungen.

Sensor/Aktor TypID	Eindeutige ID
1 Byte	1 Byte

SENSORSTATE: NOCONFIGYET Die folgenden Bytes werden von der Zentrale gesendet.

Wert	Bedeutung	Konstantenname
0x0	Identifikation über LED	IDENTIFY
0x1	Neue Konfiguration	NEW_CONFIGURATION
0x2	Konfiguration von Knoten beendet	CONFIGURATION_DONE

Die Payload des Pakets ist folgendermaßen aufgebaut. Die 64-Bit-Adresse sowie die Source ID und Sink ID wird nur gesendet, wenn der Wert `NEW_CONFIGURATION` als Erstes empfangen wird.

Identifikation	[XBeeAddress64 Adresse	Source ID	Sink ID]
Neue Konfiguration oder Ende			
1 Byte	[8 Byte	1 Byte	1 Byte]

SENSORSTATE: RUNNING Die Payload des Pakets mit den Messdaten ist folgendermaßen aufgebaut. In den 4 Byte des Messwertes kann ein großer zusammenhängender Wert sein, z.B. 4 Byte einer `uint32_t` Zahl oder es wird nur 1 Byte für eine `uint_8t` Zahl benutzt. Dies gibt der zweite Wert an. Sollten wider Erwarten 32 Bit nicht genug sein, um Sensordaten zu übermitteln, können mehrere Pakete gesendet werden. Auf Empfängerseite muss allerdings die Anwendungslogik dafür implementiert werden.

Sink ID	Anzahl benutzten Bytes	Messwert(e)
1 Byte	1 Byte	4 Byte

ACTORSTATE: RUNNING Entsprechend den gesendeten Daten aus **SENSORSTATE: RUNNING** werden hier die folgenden Daten empfangen. Das Senden der Aktoren soll sich nicht vom Senden der Sensoren unterscheiden.

Sink ID	Anzahl benutzten Bytes	Messwert(e)
1 Byte	1 Byte	4 Byte

5.5 Zusammenfassung

In diesem Kapitel sind die Grundlagen für die Umsetzung gelegt worden. Nach einem umfassenden Konzept, welches die grundlegenden Ideen erläutert, werden die Hardware der Knoten (Arduinos) und der Zentrale (Raspberry Pi) sowie die drahtlosen Standards WLAN, Bluetooth und ZigBee vorgestellt und ausgewählt. Darauf basierend werden die zu benutzenden Java und Processing APIs vorgestellt. Ab Kapitel 5.4.5 werden die im Konzept genannten Konfigurationsdateien spezifiziert. Darauf folgend wird im Kapitel 5.4.6 beschrieben, wie die Programme auf Zentrale und Knoten ablaufen sollen, welches auch in Ablaufdiagrammen und mit Hilfe einer FSM dargestellt wird. Nach einer generellen Erklärung des API Modus der XBee Module wird ein Protokoll entwickelt, welches bestimmt, wann welche Nachrichten gesendet werden müssen, was sie bedeuten und welche Datentypen verwendet werden.

6 Umsetzung

Das Design aus dem letzten Kapitel wird umgesetzt. Dabei soll dargestellt werden, welche Teile des Designs umgesetzt worden sind und welche Teile später verändert wurden. Es folgen Schwierigkeiten bei der Hardware und Software. Anschließend wird das gesamte Projekt in Form eines Proof of Concept vorgestellt. Unter Hardware wird hier der Teil behandelt, der nicht direkt die Software bzw. Firmware der Knoten betrifft.

6.1 Hardware

Aufbau Skizze

Die Abb. 6.1 zeigt einen modellhaften Aufbau des Konzepts aus Abb. 5.1 zur besseren Verständlichkeit. Auf ihr sieht man den Raspberry Pi mit dem USB Modul. Im unteren Bildteil sieht man die Arduinos mit XBee Shields und Modulen. Dies können theoretisch bis zu 65535 Knoten sein.

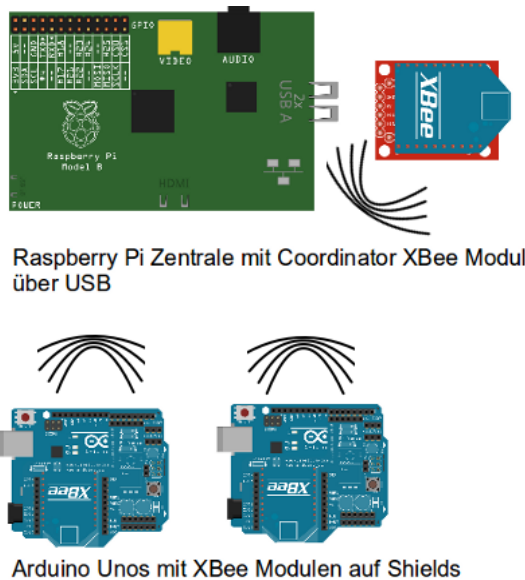


Abbildung 6.1: Modellhafter Aufbau im Fritzing Stil

6.1.1 Identifizierung über LED

Zum Debuggen kann an den Knoten an Pin 13 eine LED eingeschaltet und nach 3 Sekunden wieder ausgeschaltet werden. Es wird eine 5V LED mit eingebauten Widerstand benutzt, da dies den Einbau mit Vorwiderstand umgeht. Pin 13 wird gewählt, da er sich direkt neben GND befindet, s. Abb. 6.2. Der Code befindet sich in Anhang 9.2.

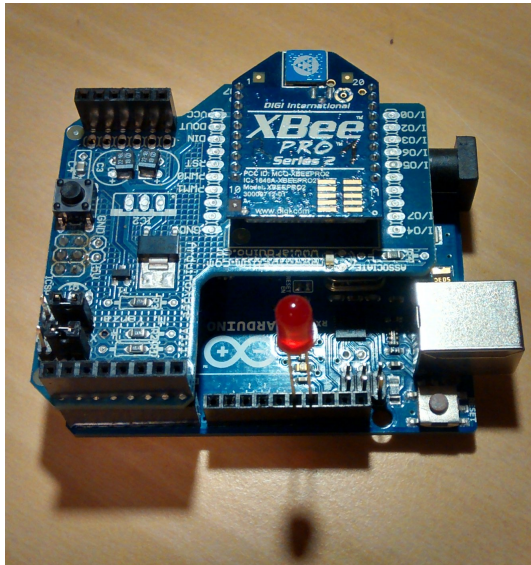


Abbildung 6.2: LED an Pin 13

6.2 Software

Wie im Design-Kapitel 5.4.3 festgelegt, wird die Software der Knoten in einer vereinfachten Version der Programmiersprache Processing mit Hilfe der *xbee-arduino* [49] Bibliothek geschrieben. Die Zentrale hingegen wird in Java mit Hilfe der API *xbee-api* [48] geschrieben.

6.2.1 Diagramme

Die folgenden Klassendiagramme erläutern den Aufbau der Zentrale und der Knoten. Sie wurden automatisch aus dem Quelltext mit dem Programm *ArgoUML* erzeugt.

Klassendiagramme der Zentrale

Die Klasse *AvailableNodes* enthält nur eine Map mit den nach der Initialisierung verfügbaren Knoten. In der Klasse *Helper* befinden sich statische Hilfsmethoden für andere Klassen. Die

Objekte, die sich in der Map in *AvailableNodes* befinden sind vom Typ *GeneralDevice*, da sich für die Zentrale Sensoren und Aktoren nicht unterscheiden. In der Klasse *SourceSink* befinden sich die Informationen, die die Zentrale nach der Konfiguration an alle Knoten sendet. Hierfür reicht die 64-Bit-XBee-Adresse zur Identifikation des Knoten und die *uniqueID* des Sensoren/Aktoren. In *Constants* befinden sich Konstanten die TypIDs mit ihrer Bedeutung. Die Klasse *Zentrale* ist der zentrale Teil der Anwendung und enthält die in Kapitel 5.4.6 beschriebene Anwendungslogik.

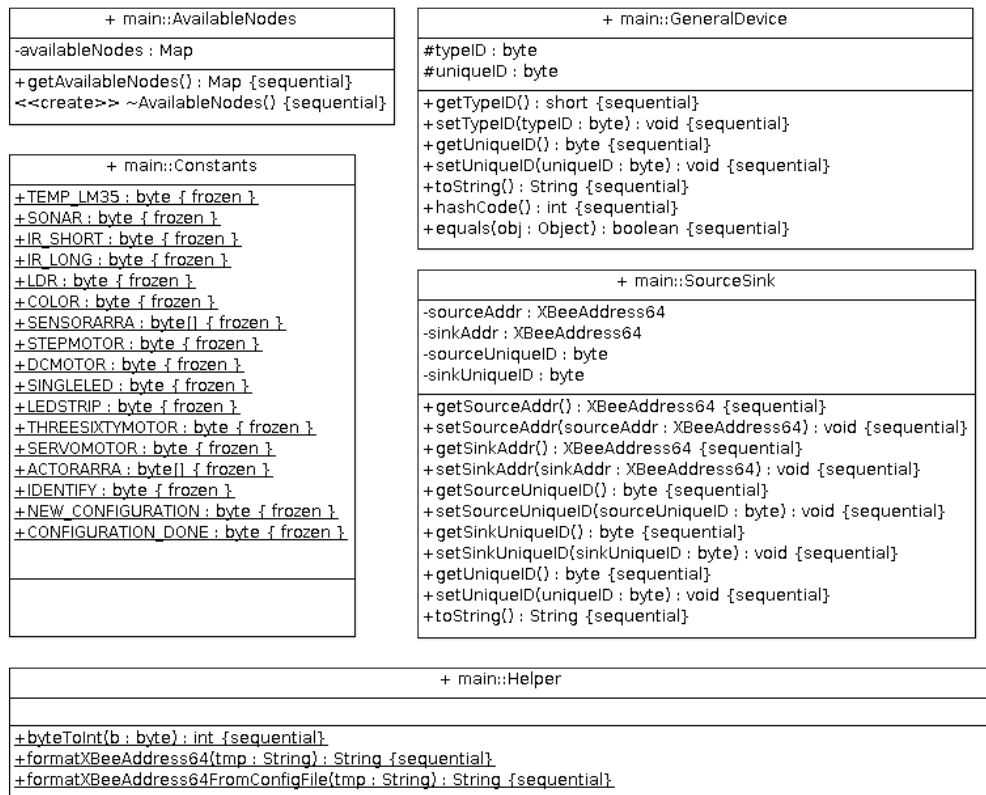


Abbildung 6.3: Klassendiagramm der Hilfsklassen der Zentrale

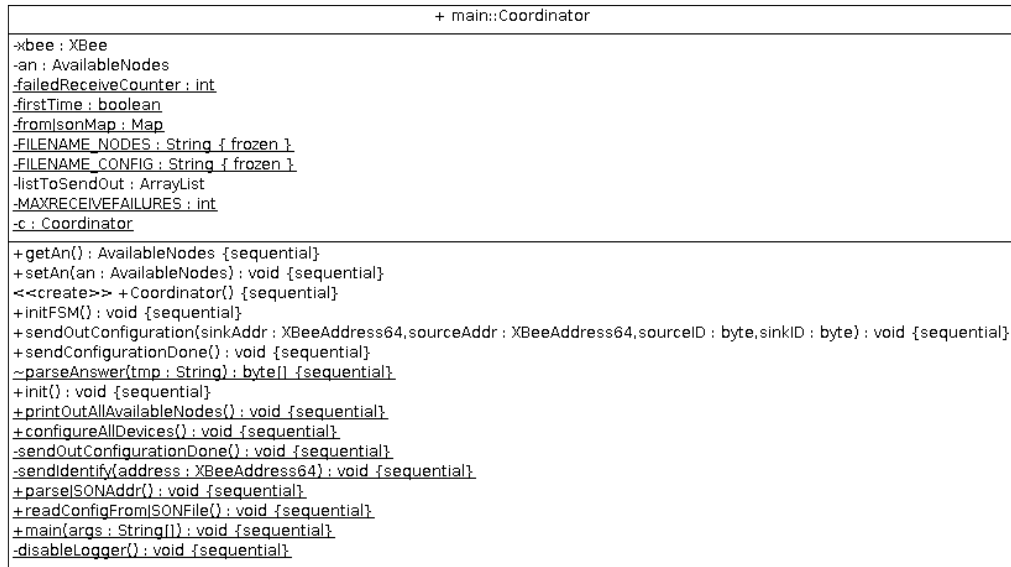


Abbildung 6.4: Klassendiagramm der Zentrale

Klassendiagramme für Arduino

Auf dem Arduino sind Sensoren und Aktoren entsprechend der Klassen *Sensor* und *Actor* aus Abb. 6.5 aufgebaut.

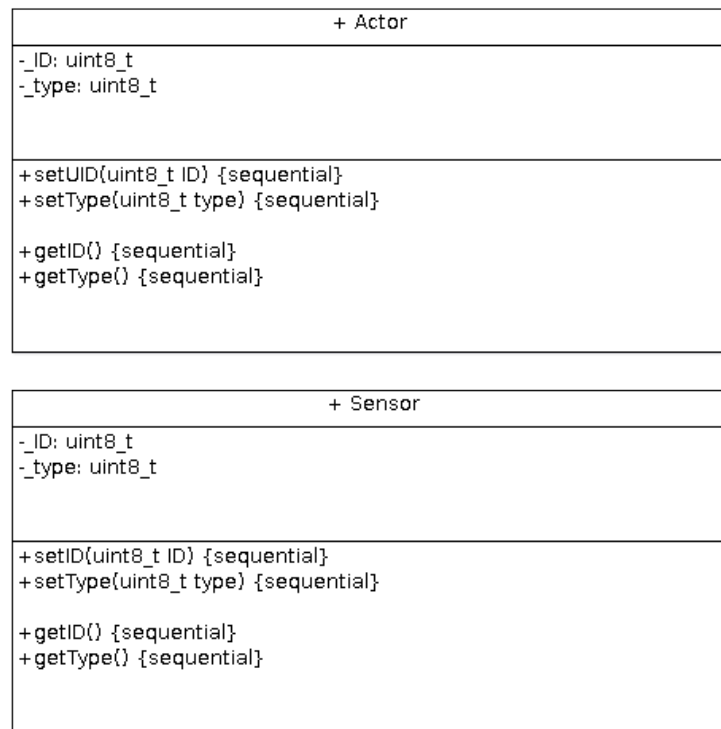


Abbildung 6.5: Klassendiagramm der Arduinos

Für beide Klassen werden Objekte in den Funktionen `initSensors()` bzw. `initActors()` in der Datei `userConfiguration.h` initialisiert. Diese Funktionen werden in der `setup()` Funktion des Hauptprogramms aufgerufen. Die Klassen unterscheiden sich inhaltlich nicht und sind nur der Übersichtlichkeit halber unterschiedlich benannt. Dies soll gemäß Kapitel 4.3 die Lesbarkeit des Codes für Nicht-Informatiker erhöhen.

6.2.2 Software auf den Knoten

Die Software für die Arduino Knoten ist in Processing geschrieben und besteht aus einer Templatdatei mit dem Hauptprogramm, einer Konfigurationsdatei mit Funktionsdeklarationen (`nodeTemplate_config.h/nodeTemplate_config.ino`), einer Datei in der Konstanten definiert werden (`sharedConfig.h`), die auch die Zentrale kennt und der schon erwähnten Konfigurationsdatei für den Benutzer (`userConfiguration.h`), s. Anhang 9.5.3. In der Templatdatei mit dem Hauptprogramm gibt es zwei größere Switch-Case Anweisungen (jeweils für Sensoren und Aktoren),

in der die Funktionsnamen für die vom Benutzer in *userConfiguration.h* zu implementierende Funktionen stehen.

6.2.3 GUI

Eine GUI belegt immer mehr Ressourcen auf einem System als eine textbasierte Oberfläche. Vor dem Erstellen des Programms war unklar, wie performant Java GUIs auf dem Raspberry Pi laufen. Das Programmieren einer GUI bedeutet unverhältnismäßig viel Mehraufwand. Dies ist einer der Gründe, nicht eine GUI, sondern eine textbasierte Oberfläche für diese Arbeit zu benutzen. Außerdem hätte eine GUI direkt auf dem Raspberry Pi zur Folge, dass zur Konfiguration der Installation immer ein Monitor angeschlossen werden müsste. So ließe sich die Zentrale nicht mit in die Installation integrieren.

Stattdessen wird die Zentrale über SSH gestartet und das Netzwerk ebenfalls über die Konsole konfiguriert.

6.2.4 Vergleichen von XBeeAddress64 Adressen auf dem Arduino

Wie bereits erwähnt wird in dieser Arbeit die *xbee-arduino* Bibliothek verwendet. Diese bietet jedoch keine Möglichkeit Objekte der Klasse *XBeeAddress64* zu vergleichen. Die Klassenreferenz gibt an, dass diese Klasse zwei 32-Bit Zahlen enthält: *LSB* und *MSB*. Wenn diese beiden Zahlen übereinstimmen, handelt es sich um denselben Knoten, da jede 64-Bit-Adresse vom Hersteller nur einmal vergeben wird. Um nicht jedes Mal *LSB* und *MSB* zu vergleichen, wird der Operator `==` bzw. `!=` in den Dateien *XBee.cpp* bzw. der dazugehörigen Header-Datei *XBee.h* überladen. Weiterhin ist dieser Code als *Issue 42* auf der offiziellen Webseite zur Verbesserung vorgeschlagen [1]. Der Code befindet sich in Anhang 9.3.

6.3 X-CTU und Arduino IDE

Das Software Tool X-CTU (Abb. 6.6) wird wie die XBee Module von der Firma *Digi* hergestellt. Hiermit ist es möglich, einfach die Parameter eines Moduls zu verändern. Es ist für Windows programmiert, läuft aber auch unter Linux oder Mac OS X. Initial müssen alle Module von *AT* auf *API* Modus umgestellt werden. Außerdem muss mindestens ein Modul als *Coordinator* konfiguriert werden.

Da die Module im *API* Modus angesprochen werden sollen, muss für alle XBee Module der *API* Modus als *Router* konfiguriert werden. Es wird darauf verzichtet, reine Endgeräte zu konfigurieren, da durch viele Router die Reichweite des Netzwerkes erhöht wird und alle Knoten vorerst an

einer festen Stromversorgung angeschlossen sind. Es ist weiterhin darauf zu achten, dass alle Module denselben Firmwarestand haben.

Es wird eine *PAN ID* festgelegt, die in alle Module eingetragen gleich wird. Zusätzlich können noch beliebige Namen als *Node Identifier* vergeben werden, um die Module im Verwechslungsfall wiederzuerkennen.

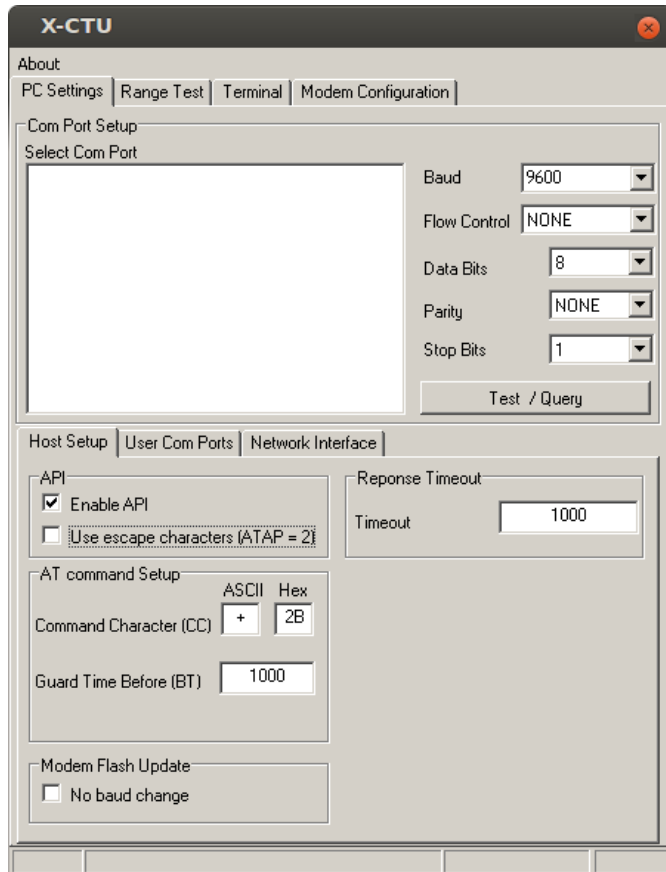
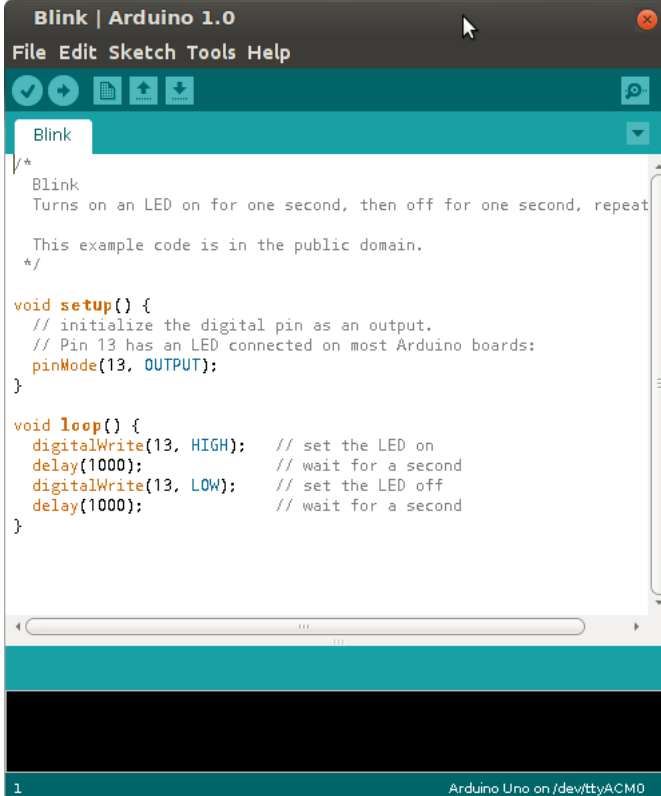


Abbildung 6.6: X-CTU

Die Arduino IDE ist eine quelloffene Software um Arduino Programme, sog. Sketches, zu schreiben und diese auf den Arduino zu übertragen. Dabei ist darauf zu achten, den richtigen Arduino Typ und Anschluss ausgewählt zu haben.

The image shows a screenshot of the Arduino IDE interface. The window title is "Blink | Arduino 1.0". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for opening files, saving, and uploading. The main editor area displays the code for the "Blink" sketch. The code includes a comment block explaining the sketch's purpose and a public domain notice. It defines a setup function to initialize pin 13 as an output and a loop function that turns the LED on for one second, then off for one second, repeating this cycle. The status bar at the bottom indicates "1" and "Arduino Uno on /dev/ttyACM0".

```
Blink | Arduino 1.0
File Edit Sketch Tools Help
Blink
/*
 * Blink
 * Turns on an LED on for one second, then off for one second, repeat
 *
 * This example code is in the public domain.
 */

void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH); // set the LED on
  delay(1000);           // wait for a second
  digitalWrite(13, LOW); // set the LED off
  delay(1000);          // wait for a second
}

1 Arduino Uno on /dev/ttyACM0
```

Abbildung 6.7: Arduino IDE

6.4 Mögliche Fehler

6.4.1 X-CTU

Bei der Konfiguration der XBee Module kann es unregelmäßig einer Fehlermeldung kommen. Das Problem kann umgangen werden, indem mit einem Wiring Kabel der GND Pin mit dem RST Pin kurz verbunden wird, wie in Abb. 6.8 dargestellt.

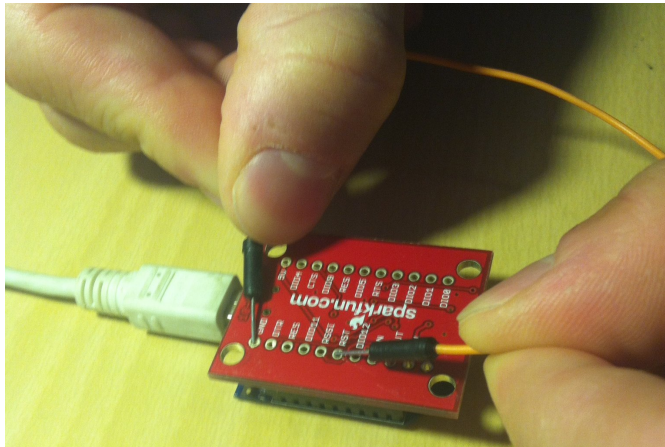


Abbildung 6.8: Reset des XBee Moduls auf dem XBee USB Explorer

6.4.2 Arduino IDE

Wichtig ist vor der Übertragung des Programms, das XBee Shield zu entfernen, da sonst die folgende Fehlermeldung auftaucht:

```
avrdude: stk500_getsync(): not in sync: resp=0x00.
```

6.4.3 Senden des ersten Pakets schlägt fehl

Wenn ein Arduino an die Spannungsversorgung oder über USB angeschlossen wird, geht das erste Paket und damit der erste konfigurierte Sensor oder Aktor verloren. Dies kann umgangen werden, indem erst die Knoten angeschlossen werden und dann die Zentrale gestartet wird, wobei das XBee Modul schon vorher über USB an den Raspberry Pi verbunden sein muss. Durch ein Betätigen des Reset-Schalters am Arduino bzw. am Shield (s. Abb 6.9) wird die FSM neugestartet und damit alle Pakete neu versendet.

Der Fehler liegt darin, dass die Software auf dem Knoten nicht wartet, bis sich das XBee Modul mit dem Coordinator verbunden hat, da dies über die API nicht abfragbar ist.

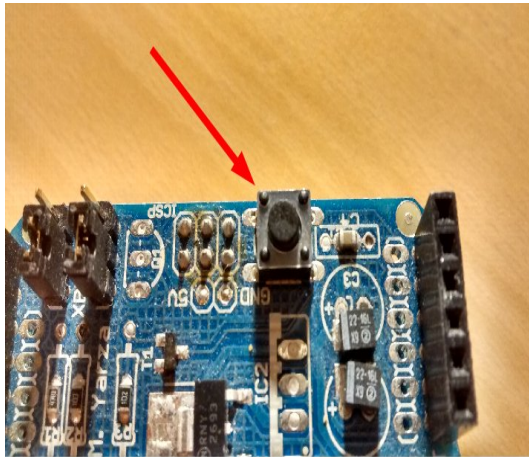


Abbildung 6.9: Reset-Schalter am Xbee Shield

6.4.4 Reset bei Verbinden der seriellen Schnittstelle

Durch das Öffnen der seriellen Konsole in der Arduino IDE wird der Arduino neu gestartet. Dies lässt sich umgehen, indem ein $10\mu F - 120\mu F$ Kondensator zwischen RESET und GND gesteckt wird. Dies zieht die Reset Verbindung permanent hoch und verhindert so einen Reset[45][15].

6.5 Proof of Concept

Die entwickelte Software wird aus Zeitgründen nicht an einer interaktiven Installation getestet. Stattdessen wird ein aussagekräftiges Proof of Concept erstellt. Dabei wird darauf geachtet, dass es einen reinen Sensorknoten, einen hybriden Sensor-Aktor Knoten und einen reinen Aktorknoten gibt. Denkbar wäre zum Testen auch, dass ein virtueller Sensor bzw. Aktor erstellt wird, der Zufallszahlen generiert und sendet. Der Aufbau besteht aus den folgenden Komponenten:

- Node 0: Photodiode (Sensor) + LED (Aktor).
- Node 1: Servomotor (Aktor), Abb. 6.10
- Node 2: Temperatursensor (Sensor)

Die Photodiode hat laut *sharedConfig.h* die ID 4 und den Konstantennamen *LDR*. Die einzelne LED hat die ID 130 und den Konstantennamen *SINGLELED*. Der Servomotor hat die ID 132 und den Konstantennamen *SERVOMOTOR*. Der Temperatursensor hat den Konstantennamen *TEMP_LM35* und die ID 0.

Folgender Ablauf soll simuliert werden:

- Wenn auf die Photodiode kein Licht mehr fällt, soll sich der Servomotor bewegen (Sensor-Aktor).
- Wenn sich der Servomotor bewegt, blinkt die LED einmal kurz auf (Aktor-Aktor).
- Wenn der Temperatur auf 25°C steigt, leuchtet die LED durchgehend (Sensor-Aktor).
- Wenn die Temperatur unter 25°C sinkt, erlischt die LED (Sensor-Aktor).

6.5.1 Aufbau

Die Abb. 6.10 zeigt den Aufbau des Servomotors an einen vollständig aufgebauten Arduino Knoten mit XBee Modul. Der Einfachheit halber wird für die LED im Proof of Concept die Identifizierungs-LED genutzt. Die Funktionen werden in die einzelnen Arduino Images geschrieben und wie oben beschrieben auf die Arduinos geflasht.

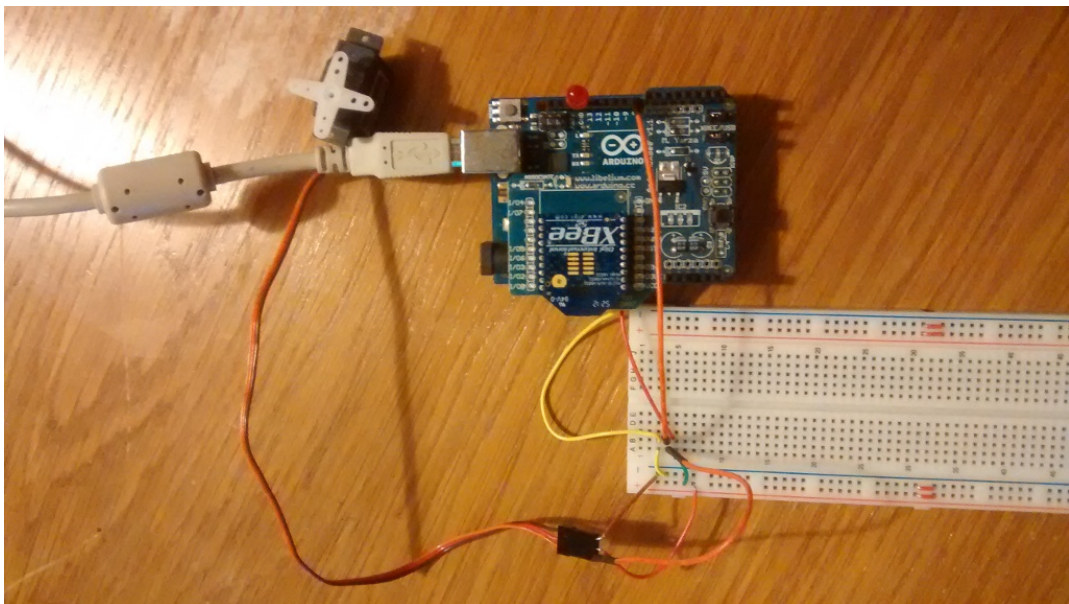


Abbildung 6.10: Node 1: Servomotor

Die Photodiode wird wie in Abb. 6.11 über einen 10k Ω Widerstand angeschlossen.

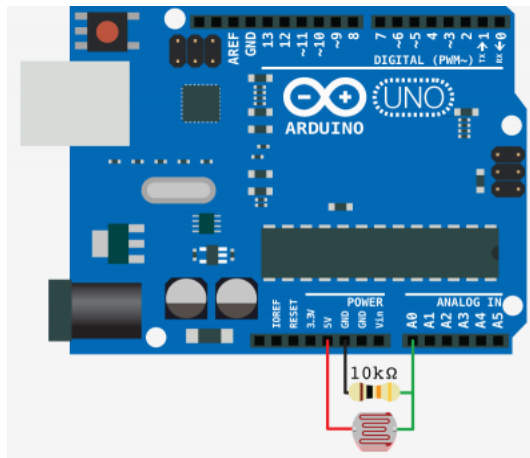


Abbildung 6.11: Node 0: LDR über $10k\Omega$ Widerstand [5]

6.5.2 Ablauf

Der Ablauf aus Kapitel 5.4.6 wird weitestgehend umgesetzt. Die Befehle zum Starten des Programms befinden sich im Anhang 9.4. Eine Beispieldatei für die Nummerierung von Knoten befindet sich ebenfalls im Anhang 9.5.1. Die Ein- und Ausgabe der Konfiguration erfolgt über SSH. Ein Beispiel ist in Abb. 6.12 dargestellt. Wird die Zentrale mit dem Parameter `manually` gestartet, erscheint ein Dialog zur Konfiguration. Wird `automatically` angegeben, wird automatisch die Datei `config.js` aus dem Homeverzeichnis des Users eingelesen. In dieser stehen Source-Sink Paare, s. Anhang 9.5.2. Die automatische Konfiguration ist die empfohlene Methode, da die manuelle Eingabe sehr mühsam ist.

6.5.3 Test

Zuerst wird überprüft, ob die in *userConfiguration.h* festgelegten Sensor- und Aktortypen in der Zentrale richtig dargestellt werden. Dazu werden die Konfigurationsdateien der drei Knoten entsprechend Kap. 6.5 erstellt. Geprüft wird hier der automatische Fall, in der die JSON Datei eingelesen wird.

Um sicherzustellen, dass die Konstante `CONFIGURATION_DONE` (0x2, s. Kap. 5.4.7) von jedem Knoten empfangen wird, wird zusätzlich in den Knoten nach dem vermeintlichen Empfang (s. Listing) die Funktion `identifyAndPause()` aus Anhang 9.2 aufgerufen, welche die LED für drei Sekunden aufleuchten lässt.

```
1 else if(response == CONFIGURATION_DONE ) {  
2     identifyAndPause();  
3     SENSORSTATE = RUNNING;  
4 }
```

Die Funktion des Beispielnetzes wird gemäß den Vorgaben aus 6.5 geprüft.

6.5.4 Auswertung

Ein Test mit Beispieldaten ist in Abb. 6.12 als Screenshot dargestellt.

```

Aktivitäten Terminal 02:07
Datei Bearbeiten Ansicht Suchen Terminal Hilfe

ID: 1
Type: 1
counter:1
Waiting for package...

ID: 2
Type: 2
counter:1
Waiting for package...

ID: 128
Type: 128
counter:1
Waiting for package...

ID: 129
Type: 129
counter:1
Waiting for package...

counter:2
Waiting for package...

counter:3
Waiting for package...

Received configuration from all nodes
1 : 0x00,0x13,0xa2,0x00,0x40,0xa8,0xb6,0x7d=[[uniqueID: 1 | typeID: 1], [uniqueID: 2 | typeID: 2], [uniqueID: 128 | typeID: 128], [uniqueID: 129 | typeID: 129]]
This Source-Sink Pair has been created:
SourceSink [sourceAddr=0x00,0x13,0xa2,0x00,0x40,0xa8,0xb6,0x7d, sinkAddr=0x00,0x13,0xa2,0x00,0x40,0xa8,0xb6,0x7d, sourceID=1, sinkID=42]
This Source-Sink Pair has been created:
SourceSink [sourceAddr=0x00,0x13,0xa2,0x00,0x40,0xa8,0xb6,0x7d, sinkAddr=0x00,0x13,0xa2,0x00,0x40,0xa8,0xb6,0x7d, sourceID=2, sinkID=23]
SourceSink [sourceAddr=0x00,0x13,0xa2,0x00,0x40,0xa8,0xb6,0x7d, sinkAddr=0x00,0x13,0xa2,0x00,0x40,0xa8,0xb6,0x7d, sourceID=3, sinkID=66]
Sent configurations
Sent CONFIGURATION_DONE(0x2) to all nodes.
Configuration done. Keep this active window for coordination.

```

Abbildung 6.12: Test mit Beispieldaten über SSH mit einem Knoten

Die Lampe leuchtet nach Empfang von CONFIGURATION_DONE drei Sekunden lang auf.

Der Einfachheit halber wird jeder Arduino an einen eigenen Laptop angeschlossen. Dort wird die serielle Konsole der Arduino IDE geöffnet. Das Netzwerk wird entsprechend der Anleitung aus Kapitel 5.4.6 aufgebaut. Über die serielle Konsole ist verifiziert worden, dass die Knoten die richtigen Daten empfangen.

6.6 Zusammenfassung

In Kapitel 6.1 wird der Aufbau der Arduino Knoten für Sensor-/Aktorknoten bzw. des Raspberry Pis für die Zentrale beschrieben. In Kapitel 6.2 befinden sich sowohl die Klassendiagramme von Zentrale und den Knoten als auch eine Erklärung der Erweiterungen der API und warum keine GUI programmiert wurde. In Kapitel 6.3 ist erläutert worden, wie die Entwicklungswerkzeuge X-CTU und Arduino IDE zum Bespielen der Knoten eingesetzt werden. Da es einige Probleme bei der Umsetzung gibt, sind diese in Kapitel 6.4 mit Lösungsansätzen genannt. Statt einer interaktiven Installation wird diese Arbeit in Kapitel 6.5 mit einem Proof of Concept getestet und ausgewertet.

7 Zusammenfassung

In diesem Kapitel soll das Projekt noch einmal zusammengefasst werden und ein Fazit gezogen werden. Dabei wird besonders auf die Umsetzung des Designs, eventuelle Verbesserungen und die gewonnenen Erkenntnisse eingegangen. Im Ausblick werden mögliche Funktionen für eine Weiterentwicklung genannt.

7.1 Ergebnisse

In dieser Arbeit wurde ein drahtloses, selbstorganisierendes Sensor- und Aktornetzwerk entworfen. Hierzu wurde eine Unterteilung in Vorbereitungskapitel (Grundlagen, Analyse) und Umsetzungskapitel (Design, Umsetzung) vorgenommen.

Das Kapitel Grundlagen gibt einen Überblick über die theoretischen Grundlagen, welche später für die Erstellung eines Sensor-Aktor-Netzwerkes benötigt werden. Dies umfasst Wissen über Sensor bzw. Sensor-Aktor-Netzwerke, Anwendungsgebiete sowie Netzwerktheorie. Mit diesem Wissen wurden in der Analyse die Anforderungen analysiert und konkretisiert, auch im Hinblick auf das Anwendungsgebiet interaktive Installationen mit seinen speziellen Anforderungen. Anschließend wurde diese Arbeit mit anderen Arbeiten verglichen und abgegrenzt.

Im Design Kapitel wurden ein Protokoll und Konventionen entworfen mit dem die Knoten untereinander kommunizieren sowie Hard- und Software vorgestellt. Auch wurde ZigBee als drahtloser Standard ausgewählt und erklärt. Mit dieser Protokollwahl wurde die Anforderung der Selbstorganisation auf das Protokoll verlagert. Der Ablauf von Zentrale und Knoten wurde anhand von Diagrammen und Fließtext erklärt. Dies bildet die Grundlage für das Kapitel 6 (Umsetzung). Das Design wurde überwiegend so umgesetzt, wie es im Kapitel *Design* besprochen wurde. Bis auf die angesprochenen Probleme funktioniert die Umsetzung wie ursprünglich geplant. Die Anwendung wurde anhand eines Proof of Concepts evaluiert. Es hat sich gezeigt, dass dies eine Grundlage ist, die es Nicht-Informatikern ermöglicht, interaktive Installationen mit vernetzen Objekten zu entwickeln. Usability Tests wären dennoch wünschenswert.

7.2 Ausblick

In der Analyse wird diese Arbeit aufgrund der Komplexität einiger Teilbereiche von Sensor- und Aktornetzwerken eingeschränkt. Dies beinhaltet die Fehlerbehandlung, Aktualität der Daten und Zustellbestätigungen. Für komplexere interaktive Installationen könnten diese Funktionen jedoch nützlich sein. Der Hauptgrund, sie nicht zu beachten, war der Zeitmangel bzw. die Komplexität.

Da diese Arbeit von Nicht-Informatikern verwendet werden soll, wäre eine grafische Oberfläche wünschenswert. Statt eine SSH-Verbindung in der Konsole aufzubauen, um die Zentrale zu konfigurieren, würde es möglich sein, ein Wrapper zu erstellen, der die Konsolen-Ausgaben in einer GUI darstellt. Alternativ könnte die Zentrale umgeschrieben werden und über eine Art API ansprechbar sein oder ein X-Fenster über einen SSH Tunnel weiterleiten. Weiterhin wäre denkbar, statt einer Java Zentrale die JavaScript API *xbee-api*[29] für node.js zu verwenden, um die Zentrale mit einem Webinterface zu reimplementieren.

In der Analyse wurde weiterhin argumentiert, dass es nützlich sei, hybride Knoten zu benutzen. Da ein Proof of Concept erstellt wurde, müsste diese Hypothese in der Praxis verifiziert werden. Gegebenenfalls könnten Sensoren- und Aktorknoten aufgetrennt werden, um Sensorknoten auch mit Batterien betreiben zu können. Auch könnte für die Aktoren ggf. ein Arduino Mega verwendet werden, da dieser mehr Anschlüsse besitzt. Der Code müsste hierfür nicht geändert werden, für Sensorknoten würden nur keine Aktoren und für Aktorknoten keine Sensoren implementiert werden.

Es hat sich gezeigt, dass die verwendete Technologie für Netzwerke in interaktiven Installationen ausreichend ist. Nichtsdestotrotz wäre es praktischer, ein Betriebssystem zu benutzen, das Nebenläufigkeit beherrscht. Beispiele für solche Betriebssysteme sind die in Kap. 3.2.1 erwähnten Projekte RIOT oder teilweise auch TinyOS[34] und Contiki. Eine derartige Programmierung allerdings erfordert mehr Wissen und ist somit für Künstler ohne Programmierhintergrund nur schwer umzusetzen.

7.3 Abschließendes Fazit

Zusammengefasst lässt sich sagen, dass das Design erfolgreich umgesetzt wurde und Nicht-Informatiker mit Hilfestellungen und Grundkenntnissen im Programmieren die Technologie bedienen könnten. Die Arduino Plattform ist bei vielen Künstlern bereits bekannt und die Programmiersprache *Processing* ist relativ einfach zu erlernen.

Die im Ausblick angesprochenen Verbesserungen, vor allem eine GUI, würden sicherlich noch weiter dazu beitragen, dem Anwender Komplexität in der Konfiguration eines solchen Netzes abzunehmen.

8 Glossar

Abkürzung	Bedeutung
API	Application Programming Interface
AODV	Ad-hoc On-demand Distance Vector
FSM	Finite State Machine, englisch für Endlicher Automat
GND	Ground, englisch für Erdung
GUI	Graphical User Interface
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
I/O	Input/Output
JSON	JavaScript Object Notation
LED	Licht-emittierende Diode
LR-WPAN	Low-Rate Wireless Personal Area Network
MAC	Media Access Control
OSI	Open Systems Interconnection
PHY	Physical Layer
PWM	Pulsweitenmodulation
RS232	Standard für serielle Schnittstelle
SSH	Secure Shell
WLAN	Wireless Local Area Network
WPAN	Wireless Personal Area Network(s. auch LR-WPAN)
WSAN	Wireless Sensor Actor Network
WSN	Wireless Sensor Network
XML	Extensible Markup Language

9 Anhänge

9.1 Kurzanleitung

1. Die in Kapitel 9.5.3 genannte Konfigurationsdatei gemäß den Kommentaren ausfüllen
2. Für jeden Knoten die Funktionen für Sensoren und Aktoren implementieren
3. Code auf Arduinos flashen
4. XBee USB Modul an den Raspberry Pi anschließen, Raspberry Pi booten und per SSH einloggen
5. Im Homeverzeichnis *config.js* und *nodes.js* erstellen (Kap. 9.5)
6. Knoten an die Spannungsversorgung anschließen
7. Zentrale gemäß Kap. 9.4 starten
8. Reset Knöpfe auf allen Knoten drücken, s. Abb 6.9

9.2 Identifikations-LED an Arduino

Diese LED dient als Identifikationshilfe zum Debuggen.

```
1 int led = 13;
2
3 [...]
4
5 void setup() {
6   pinMode(led, OUTPUT);
7   [...]
8 }
9
10 [...]
11
```

```
12 void identifyAndPause {
13     digitalWrite(led, HIGH);
14     delay(3000);
15     digitalWrite(led, LOW);
16 }
```

9.3 Vergleichen von XBee64Address

XBee.h

```
1 bool operator==(XBeeAddress64 addr);
2 bool operator!=(XBeeAddress64 addr);
```

XBee.cpp

```
1 bool XBeeAddress64::operator==(XBeeAddress64 addr) {
2     return ((_lsb == addr.getLsb()) && (_msb == addr.getMsb()));
3 }
4
5 bool XBeeAddress64::operator!=(XBeeAddress64 addr) {
6     return !(*this == addr);
7 }
```

9.4 Zentrale starten

Abfrage auf manuelle Eingabe der Source-Sink Paare:

```
1 java -jar zentrale.jar manually
```

Automatisch mit Einlesen der JSON Datei:

```
1 java -jar zentrale.jar automatically
```

9.5 Konfigurationsdateien

Beide Konfigurationsdateien müssen sich im Homeverzeichnis des Users befinden.

9.5.1 nodes.js

Der Schlüssel ist die ID des Knotens. Der Wert entspricht der 64 Bit XBee Adresse, die sich auf einem Aufkleber auf dem Modul befindet. Alternativ kann sie auch über das Programm X-CTU ermittelt werden. Es ist wichtig, dass die Adresse in diesem Format angegeben wird.

```
1 {
2     "0": "0x00,0x13,0xa2,0x00,0x40,0x31,0x20,0x3d",
3     "1": "0x00,0x13,0xa2,0x00,0x40,0xa8,0xbe,0x7d"
4 }
```

9.5.2 config.js

Diese JSON Datei ist als Array verschachtelt. Beginnend bei 0 werden die Source-Sink Paare inkrementiert. Jedes Paar enthält dabei zwei Einträge. Im ersten Eintrag stellt der Schlüssel die Source ID und der Wert die Source Adresse dar. Der zweite Eintrag ist dementsprechend die Sink ID bzw. Sink Adresse. Dies kann beliebig oft wiederholt werden. Es ist wichtig, dass die Adresse in diesem Format angegeben wird.

```
1 {
2     "0": {
3         "1" : "0x00,0x13,0xa2,0x00,0x40,0x31,0x20,0x3a",
4         "42": "0x00,0x13,0xa2,0x00,0x40,0x31,0x20,0x3b"
5     },
6     "1": {
7         "2" : "0x00,0x13,0xa2,0x00,0x40,0x31,0x20,0x3c",
8         "23": "0x00,0x13,0xa2,0x00,0x40,0x31,0x20,0x3d"
9     },
10    "2": {
11        "3" : "0x00,0x13,0xa2,0x00,0x40,0x31,0x20,0x3e",
12        "66": "0x00,0x13,0xa2,0x00,0x40,0x31,0x20,0x3f"
13    }
14 }
```

9.5.3 userConfiguration.h

In Zeile 28 und 37 sind beispielhaft Funktionen aufgeführt, die der Benutzer implementieren könnte. Diese sind in der Hauptdatei referenziert.

```
1 // local 64-Bit XBee address. Can be found via X-CTU
2 // or on the actual device. see nodes.js
3 XBeeAddress64 localXBeeAddress = XBeeAddress64(0x0000,0x0000);
4
5 // number of sensors and actors that are wired to this node
6 const int NUMBER_OF_SENSORS = 1;
7 const int NUMBER_OF_ACTORS = 1;
```

```
8
9 // maximum number of remote nodes that we send data to
10 const int MAX_SOURCES_SINKS = 2;
11
12 // For setup()
13 void initSensors() {
14     sensorArray[0].setID(...);
15     sensorArray[0].setType(...);
16 }
17
18 void initActors() {
19     actorArray[0].setID(...);
20     actorArray[0].setType(...);
21 }
22
23 // -----//
24 // implement these function for every actor and
25 // add more functions if needed.
26 // -----//
27
28 void actSingleLED(uint8_t actorID, uint32_t values) {
29     ...
30 }
31
32 // -----//
33 // implement these function for every sensor and
34 // add more functions if needed.
35 // -----//
36
37 uint32_t getLDRSensorValue() {
38     ...
39 }
```

9.6 CD

- Tex Dateien und Bilder der Bachelorarbeit
- Bachelorarbeit als PDF-Datei
- Java Quelltext der Zentrale

- Processing Quelltext eines Beispielknoten

Literatur

- [1] „repat123”. *Issue 42: Comparing operator== for XBeeAddress64*. (Letzter Zugriff: 17. April 2014). Dez. 2013. URL: <https://code.google.com/p/xbee-arduino/issues/detail?id=42>.
- [2] Volkswagen AG. *Piano Staircase - The Fun Theory*. (Letzter Zugriff: 17. April 2014). Sep. 2009. URL: <http://www.thefuntheory.com/piano-staircase>.
- [3] arduino.cc. *ArduinoBoardDue*. (Letzter Zugriff: 17. April 2014). 2013. URL: <http://arduino.cc/en/Main/ArduinoBoardDue>.
- [4] US Army. *Planting an T-UGS Sensor field at Fort Bliss (gemeinfrei)*. (Letzter Zugriff: 17. April 2014). 2008. URL: <https://commons.wikimedia.org/wiki/File:UGS-training-FtBliss1-full.jpg>.
- [5] bildr.org. *Simple Light Reading With LDR + Arduino (CC-BY-SA 3.0)*. (Letzter Zugriff: 17. April 2014). 2012. URL: <http://bildr.org/2012/11/photoresistor-arduino/>.
- [6] Bomazi. *Arduino Ethernet Shield (CC-BY-SA 2.0)*. (Letzter Zugriff: 17. April 2014). 2009. URL: https://commons.wikimedia.org/wiki/File:Arduino_Ethernet_Shield.jpg.
- [7] National Data Buoy Center. *NDBC DART Programm*. (Letzter Zugriff: 17. April 2014). Aug. 2013. URL: <http://www.ndbc.noaa.gov/dart.shtml>.
- [8] Chee-Yee Chong, Srikanta P. Kumar und (IEEE). *Sensor Networks: Evolution, Opportunities, and Challenges*. (Letzter Zugriff: 17. April 2014). Aug. 2003. URL: <http://www.ics.uci.edu/~dsm/ics280sensor/readings/intro/chong.pdf>.
- [9] Contiki. *Contiki Resources and Support*. (Letzter Zugriff: 17. April 2014). Apr. 2014. URL: <http://www.contiki-os.org/support.html>.
- [10] Edmond Couchot, Michel Bret und Marie-Hélène Tramus. *La Plume(franz.)* (Letzter Zugriff: 17. April 2014). Apr. 2014. URL: http://www.ciren.org/artifice/artifices_1/couchot.html.
- [11] George Coulouris, Jean Dollimore und Tim Kindberg. *Verteilte Systeme - Konzepte und Design*. Pearson Studium, 2002.

- [12] W. Dargie und C. Poellabauer. *Fundamentals of wireless sensor networks: theory and practice*. John Wiley und Sons, 2010.
- [13] Andy Dingley. *Arduino Mega2560*. (Letzter Zugriff: 17. April 2014). 2011. URL: http://commons.wikimedia.org/wiki/File:Arduino_Mega2560.jpg.
- [14] N.T. Dinh und Y. Kim. *ICN Wireless Sensor and Actor Network BaseLine Scenarios - Version 01*. Techn. Ber. IETF(draft-dinh-icn-sensor-scenarios-01), 2013.
- [15] „DrRossi“. *Arduino Uno can't disable serial reset*. (Letzter Zugriff: 17. April 2014). Jan. 2011. URL: <http://forum.arduino.cc/index.php/topic,28723.0.html>.
- [16] Bundesministerium für Wirtschaft und Energie. *Internet der Dinge - Vernetzte Lebens- und Arbeitswelten*. (Letzter Zugriff: 17. April 2014). Apr. 2011. URL: <http://www.nextgenerationmedia.de/>.
- [17] Joakim Eriksson. “Detailed Simulation of Heterogeneous Wireless Sensor Networks”. Diplomarbeit. Uppsala University, Schweden, 2010.
- [18] ZVEI e.V. *Intelligents Wohnen, Modernes Zuhause*. (Letzter Zugriff: 17. April 2014). Dez. 2013. URL: http://www.intelligenteswohnen.com/iw_de/.
- [19] Robert Fauldi. *Building Wireless Sensor Networks: with ZigBee, XBee, Arduino, and Processing*. O'Reilly, 2010.
- [20] Mark Fickett. *Pair of XBee Series 2s with Whip Antennas*. (Letzter Zugriff: 17. April 2014). 2012. URL: https://commons.wikimedia.org/wiki/File:Pair_of_XBee_Series_2s_with_Whip_Antennas.jpg.
- [21] Freeduino. *The World Famous Index of Arduino and Freeduino Knowledge*. (Letzter Zugriff: 17. April 2014). Juli 2010. URL: <http://www.freeduino.org/>.
- [22] Saurabh Ganeriwal, Ram Kumar und Mani B. Srivastava. *Timing-sync Protocol for Sensor Networks*. Techn. Ber. Networked und Embedded Systems Lab (NESL), University of California Los Angeles, 2003.
- [23] Ken Goldberg. *Telegarden*. (Letzter Zugriff: 17. April 2014). 2004. URL: <https://commons.wikimedia.org/wiki/File:Telegarden.jpg>.
- [24] IETF. *IPv6 over Low power WPAN (6lowpan)*. (Letzter Zugriff: 17. April 2014). Apr. 2014. URL: <http://datatracker.ietf.org/wg/6lowpan/charter/>.
- [25] IETF. *Routing Over Low power and Lossy networks (roll)*. (Letzter Zugriff: 17. April 2014). März 2014. URL: <http://datatracker.ietf.org/wg/roll/charter/>.

- [26] IETF. *TRANSMISSION CONTROL PROTOCOL*. (Letzter Zugriff: 17. April 2014). Sep. 1981. URL: <https://tools.ietf.org/html/rfc793>.
- [27] Fraunhofer IML. *Aktuelle Projekte*. (Letzter Zugriff: 17. April 2014). Apr. 2014. URL: http://www.internet-der-dinge.de/de/projekte0/aktuelle_projekte.html.
- [28] JotaCartas. *Arduino-uno-perspective-whitw*. (Letzter Zugriff: 17. April 2014). 2011. URL: <http://commons.wikimedia.org/wiki/File:Arduino-uno-perspective-whitw.jpg>.
- [29] „jouz“. *xbee-api auf GitHub*. (Letzter Zugriff: 04. Mai 2014). Mai 2014. URL: <https://github.com/jouz/xbee-api>.
- [30] Jwrodgers. *RaspberryPi*. (Letzter Zugriff: 17. April 2014). 2012. URL: <http://commons.wikimedia.org/wiki/File:RaspberryPi.jpg>.
- [31] P.H. Kahn u. a. *The distant gardener: what conversations in the Telegarden reveal about human-telebotonic interaction*. (Letzter Zugriff: 17. April 2014). Aug. 2005. URL: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1513749>.
- [32] Maryam Kamali u. a. *Self-Recovering Sensor-Actor Networks*. Techn. Ber. IT-Department Abo Akademi University Turku Finland, 2013.
- [33] B. Karp. “Geographic Routing for Wireless Network”. Diss. Harvard University in Cambridge, MA, 2000.
- [34] Kevin Klues u. a. *The TOSThreads Thread Library*. (Letzter Zugriff: 17. April 2014). Mai 2008. URL: <http://www.tinyos.net/tinyos-2.x/doc/html/tep134.html>.
- [35] Tobias Krauer. *Zeitsynchronisation in Sensornetzen*. Techn. Ber. ETH Zürich, 2003.
- [36] Maksim mit Änderungen von Krzysztof Zajączkowski. *Network Topologies*. (Letzter Zugriff: 17. April 2014). 2011. URL: <https://commons.wikimedia.org/wiki/File:NetworkTopologies.svg>.
- [37] Dr. Andreas Kuntz. “Dienstbasierte Kommunikation über unzuverlässige drahtlose Verbindungen für selbstorganisierende Sensor-Aktor-Netze”. Diss. KIT Karlsruhe, Deutschland, 2011.
- [38] Murata Manufacturing Co. Ltd. *Development of Sensor Network System for Smart Homes*. (Letzter Zugriff: 17. April 2014). Sep. 2011. URL: http://www.murata.com/new/news_release/2011/0930b/.
- [39] Willow Technologies Ltd. *Wireless*. (Letzter Zugriff: 17. April 2014). Dez. 2013. URL: <http://www.willow.co.uk/html/wireless.html>.

- [40] Projekt Blinkenlights/ Kaspar Metz. *Arcade*. (Letzter Zugriff: 17. April 2014). Jan. 2008. URL: <http://blinkenlights.net/arcade>.
- [41] Projekt Blinkenlights/ Kaspar Metz. *Blinkenlights*. (Letzter Zugriff: 17. April 2014). Jan. 2008. URL: <http://blinkenlights.net/blinkenlights>.
- [42] Alireza Moini. *Smart sensors*. (Letzter Zugriff: 17. April 2014). März 1997. URL: http://www.iee.et.tu-dresden.de/iee/analog/papers/mirror/visionchips/vision_chips/smart_sensors.html.
- [43] Alexander Pautz. "Kabelloses, stromsparendes Sensornetzwerk im Bereich Ambient Intelligence". Masterarbeit. HAW Hamburg, Deutschland, 2012.
- [44] Yeo Kian Pee u. a. *StickEar: Augmenting Objects and Places Wherever Whenever*. (Letzter Zugriff: 5. Mai 2014). Jan. 2013. URL: http://www.sutd.edu.sg/cmsresource/idc/papers/2013_StickEar_Augmenting_Objects_and_Places_Wherever_Whenever.pdf.
- [45] Arduion Playground. *DisablingAutoResetOnSerialConnection*. (Letzter Zugriff: 03. Mai 2014). Mai 2014. URL: <http://playground.arduino.cc/Main/DisablingAutoResetOnSerialConnection>.
- [46] Dr.-Ing. Joern Ploennigs. *Drahtlose und drahtgebundene Sensor-Aktor-Netzwerke*, Folie 28. Techn. Ber. TU Dresden, 2013.
- [47] Andrew Rapp. *Why API mode?* (Letzter Zugriff: 17. April 2014). Dez. 2010. URL: <https://code.google.com/p/xbee-api/wiki/WhyApiMode>.
- [48] Andrew Rapp. *XBee API*. (Letzter Zugriff: 17. April 2014). Nov. 2013. URL: <https://code.google.com/p/xbee-api/>.
- [49] Andrew Rapp. *XBee Arduino*. (Letzter Zugriff: 17. April 2014). Nov. 2013. URL: <http://xbee-arduino.googlecode.com/>.
- [50] RIOT. *RIOT - Three good reasons to think about a new OS for the IoT*. (Letzter Zugriff: 17. April 2014). Apr. 2014. URL: <http://www.riot-os.org/#features>.
- [51] Marco Schneider. "Entwicklung und Realisierung eines Sensornetzwerkes fuer das Living Place Hamburg". Bachelorarbeit. HAW Hamburg, Deutschland, 2010.
- [52] Ad Hoc Networks: Wireless sensor und actor networks: research challenges. *Ian F. Akyildiz and Ismail H. Kasimoglu*. ELSEVIER, 2004.
- [53] Ghalib A. Shah u. a. *Real-Time Coordination and Routing in Wireless Sensor and Actor Networks*. (Letzter Zugriff: 17. April 2014). Mai 2006. URL: <http://nwcl.ku.edu.tr/paper/C10.pdf>.

- [54] Jeffrey Shaw. *The Legible City* beim Zentrum für Kunst und Medientechnologie Karlsruhe. (Letzter Zugriff: 17. April 2014). Apr. 2014. URL: <http://on1.zkm.de/zkm/werke/TheLegibleCity>.
- [55] Javier Solobera. *Detecting Forest Fires using Wireless Sensor Networks*. (Letzter Zugriff: 17. April 2014). Sep. 2010. URL: http://www.libelium.com/wireless_sensor_networks_to_detec_forest_fires/.
- [56] Sparkfun. *XBee Buying Guide*. (Letzter Zugriff: 17. April 2014). Nov. 2013. URL: https://www.sparkfun.com/pages/XBee_Guide.
- [57] Sparkfun. *XBee Explorer USB (CC-BY-NC-SA 3.0)*. (Letzter Zugriff: 17. April 2014). 2013. URL: <https://dlnmh9ip6v2uc.cloudfront.net//images/products/8/6/8/7/08687-07-L.jpg>.
- [58] Marten marsu Suhr. *Projekt Blinkenlights, Berlin (CC-BY-SA 3.0)*. (Letzter Zugriff: 17. April 2014). 2005. URL: <https://commons.wikimedia.org/wiki/File:Hdl.jpg>.
- [59] Textron Defense Systems. *Unattended Ground Sensors - MicroObserver*. (Letzter Zugriff: 17. April 2014). Apr. 2014. URL: <http://www.textrondefense.com/products/isr/unattended-ground-sensors/microobserver>.
- [60] Augmented Senses Group der Singapore University of Technology und Design. *StickEar*. (Letzter Zugriff: 17. April 2014). März 2014. URL: <http://asg.sutd.edu.sg/project/stickears/>.
- [61] Augmented Sensor Group der Singapore University of Technology und Design. *StickEar*. (Letzter Zugriff: 17. April 2014). 2012. URL: http://asg.sutd.edu.sg/wp-content/uploads/2013/03/IMG_1225.jpg.
- [62] The Fun Theory. *Piano Staircase*. (Letzter Zugriff: 17. April 2014). Sep. 2009. URL: <http://www.youtube.com/watch?v=2lXh2n0aPyw>.
- [63] KJ Vogelius. *Piano Staircase - behind the scenes (CC-BY-SA 3.0)*. (Letzter Zugriff: 17. April 2014). 2009. URL: https://www.flickr.com/photos/kj_/3669721910/.
- [64] Mark Weiser. *The Computer for the 21st Century*. (Letzter Zugriff: 17. April 2014). Sep. 1991. URL: <http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html>.
- [65] Mitglieder bei wiki.xmbc.org. *Raspberry Pi*. (Letzter Zugriff: 17. April 2014). Apr. 2014. URL: http://wiki.xmbc.org/?title=Raspberry_Pi.

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

Hamburg, 12. Mai 2014

René Schultz