Besucherinteraktion auf Veranstaltungen mit der OpenBeacon-Technologie

Abschlussarbeit

zur Erlangung des akademischen Grades

Bachelor of Science (B.Sc.)

an der

Hochschule für Technik und Wirtschaft Berlin
Fachbereich Wirtschaftswissenschaften II
Studiengang Angewandte Informatik

1. Prüfer: Prof. Dr. Jürgen Sieck
2. Prüfer: Dr.-Ing. Michael A. Herzog

eingereicht von: Stephan Bergemann
Matrikelnummer: 521034

Datum: 10. Juni 2010

Danksagung

Diese Bachelorarbeit wäre ohne einige Menschen in meinem näheren und weiteren Umfeld nicht zu dem geworden, was sie ist.
Daher möchte ich meinen Dank zunächst an Herrn Prof. Dr. Jürgen Sieck, sowie Herrn Prof. Dr. Michael A. Herzog richten, die mir mit wertvollen Ratschlägen über die gesamte Zeit hilfreich zur Seite standen. Darüber hinaus möchte ich Herrn Michael Quade vom BureauQ Berlin danken, ohne den das Projekt ROBERTA nicht zu realisieren gewesen wäre.
Des weiteren bin ich einigen Menschen für ihre tatkräftige persönliche Unterstützung, stets konstruktive Kritik und Aufmunterung sehr dankbar:

- Elly
- Monika Maria
- Björn
- Lars
- den aktiven Kommilitonen aus dem IRC-Channel #aiws2007

Auch meinen Eltern, Großeltern, meiner Schwester und insbesondere meiner Freundin Franziska möchte ich für die Unterstützung danken und dafür, dass sie mir während der Anfertigung dieser Arbeit den Rücken frei hielten.
Und schließlich bin ich den Teilnehmern des Feldtests zu großem Dank verpflichtet - ohne sie hätte ich keine repräsentativen Daten für eine Auswertung der im Rahmen dieser Arbeit geschaffenen Lösung gehabt.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Auf Veranstaltungen geht es oft darum, Kontakte zu knüpfen und Daten für die weitere Kommunikation auszutauschen. Das passiert bis dato in Form von Visitenkarten auf denen Informationen zu den Personen formfrei und daher höchst individuell strukturiert sind. Am Ende findet sich der Besucher mit einer Menge von Visitenkarten wieder und die Zuordnung von Gesichtern, Begegnungen und Karten aus dem Gedächtnis wird zur Herausforderung. Man erinnert sich eher an den Ort oder an Begleitumstände (wie etwa die ungefähre Zeit) interessanter Begegnungen, als an die Namen der Gesprächspartner. Denn man stellt sich am Anfang des Gespräches einmal vor, überreicht sich am Ende Visitenkarten oder vergleichbares Präsentationsmaterial und geht auseinander. Der Name des Gegenübers kommt dabei gerade zweimal zur Sprache. Dadurch sinkt den Wert der Kontaktdaten erheblich - oder verschwindet sogar vollständig - falls die Zuordnung nicht mehr erinnerlich ist.
Das führt zur Fragestellung, wie man die spätere Zuordnung von Gesprächspartnern schon während der Veranstaltung mit technischen Hilfsmitteln erleichtern und so den Wert solcher Begegnungsdaten stabilisieren, oder sogar erhöhen kann.

Vernetzung ist durch zunehmende Digitalisierung inzwischen ein so essentieller Bestandteil des Alltags, sowohl im Privat- als auch im Berufsleben geworden, dass Visitenkarten in Papierform gar nicht mehr alle relevanten Kontaktdaten abbilden können. Mittlerweile müssten darauf nicht mehr nur Personennamen, sondern auch Identitäten in sozialen

Netzwerken abgebildet werden. Auch das einfache „Hinzufügen“ neuer Bekanntschaften zu digitalen Freundschafts- und Kontaktnetzen ist auf Papierbasis einfach nicht mehr machbar. Die Daten müssen zuvor immer erst digitalisiert und in die entsprechende Plattform eingetragen werden, was oft mit erheblichem Aufwand verbunden ist.

Veranstalter haben darüber hinaus ein großes Interesse daran, ihre Veranstaltung so beeindruckend wie möglich zu gestalten. Damit Besucher sich noch lange angenehm an die Veranstaltung erinnern, sollen ihnen außergewöhnliche Services geboten werden.

Diese Bachelorarbeit befasst sich mit der automatisierten Erfassung und Verarbeitung von Kontaktinformationen für Besucher von Veranstaltungen. Dabei wird besonders auf Möglichkeiten eingegangen, diese Daten auch direkt auf der Veranstaltung für Besucherinteraktion zu nutzen.

1.2 Zielsetzung

Ziel dieser Arbeit sind die Konzeption und Umsetzung einer modularen Softwarearchitektur, mithilfe derer Kontakte von Veranstaltungsbesuchern registriert und berechnet werden können. Dabei sollen die Informationen gefiltert und für die weitere Verarbeitung sowie zur Nachbereitung der Veranstaltung geeignet aufbereitet und persistiert werden. Die zu entwickelnden Module kommunizieren über ein eigens dafür entworfenes Protokoll miteinander.

Im Rahmen dieser Arbeit soll vor allem die Nutzung der Daten für die Schaffung von Interaktionsmöglichkeiten zwischen Besucher und der Veranstaltungtechnik im Vordergrund stehen. Dazu werden sowohl ortsbezogene, als auch ortsunabhängige Formen von Interaktion evaluiert und in Form eines Softwareprototypen umgesetzt.

Die dabei entwickelte Softwarearchitektur soll nicht ausschließlich auf herkömmlichen PCs laufen: Es soll auch erprobt werden, wie sich weitere Hardware einbinden lässt, um die zu entwickelnde Lösung noch vielfältiger nutzen zu können.

1.3 Aufbau der Arbeit

Das Kapitel 2 setzt sich mit den technischen Grundlagen und Begrifflichkeiten auseinander. Es wird auf bereits existierende Lösungen für verschiedene Anforderungen eingegangen. Im sich anschließenden Kapitel 3 wird eine Analyse der Anforderungen an das System vorgenommen. Dies geschieht, indem das Umfeld der Anwendung näher spezifiziert und Nutzungsszenarien vorgestellt werden.

Im Kapitel 4 wird der Entwurf des Systems beschrieben, die dazugehörigen Daten und Funktionalitäten modelliert und die Umsetzung der Architektur in verschiedenen Komponenten erläutert.

Basierend auf den definierten Anforderungen und dem Systementwurf folgend, wird die technische Realisierung des Systems im Kapitel 5 beschrieben. Hier werden insbesondere Auszüge zentraler Schritte aus dem Quelltext angeführt, anhand derer die Umsetzung erklärt wird.

Anschließend wird im Kapitel 6 das System anhand vorgenommener Feldtests auf die Erfüllung der Anforderungen geprüft. Zunächst werden die Testumgebungen beschrieben, um dann die Funktionalität zu demonstrieren. Exemplarisch werden hier Bilder der verschiedenen Komponenten des Systems angeführt.

Nach der Auswertung der Feldtests beschreibt Kapitel 7 die geschaffene Lösung und beschließt diese Bachelorarbeit mit einem Ausblick auf potentielle Erweiterungen des vorgestellten Systems.

2 Grundlagen

Dieses Kapitel vermittelt die wichtigsten Technologien und führt in Begrifflichkeiten ein, welche die Grundsteine für die weitere Arbeit bilden.
Zunächst wird der Interaktivitätsbegriff erläutert und eingegrenzt. Es folgt eine Einführung in die OpenBeacon-Technologie, sowie RFID im Allgemeinen. Anschließend wird im Besonderen auf das Thema RFID und Ortung eingegangen.
Darauf folgend wird mit Arduino eine Hardwareplattform genauer betrachtet. Abgeschlossen wird das Kapitel durch einem Einstieg zu Veranstaltungstechnik mit besonderem Augenmerk auf DMX als verwendetes Steuerungsprotokoll.

2.1 Interaktion und Interaktivität

Interaktion und Interaktivität sind Dreh- und Angelpunkt dieser Arbeit. Es geht darum, eine Form von Mensch-Maschine- und ebenso Mensch-Maschine-Mensch-Interaktion zu ermöglichen.

> „Interaktivität ist das Schlüsselwort der neuen Informations- und Kommunikationtechnologien, das ihre spezifische Differenz und den Vorsprung gegenüber den „alten“ Print-, Ton und Bildmedien markieren soll.“[l_BL04, S. 7]

Diese Worte nehmen bereits einen der wichtigsten Grundgedanken vorweg: Interaktivität wird hier als Begriff zur Abgrenzung der „neuen“ digitalen Medien gegenüber den „alten“ Medien genutzt.

Es geht dabei im Wesentlichen darum, dass Medien einen neuen Rückkanal bekommen, durch den Interaktionen ermöglicht werden sollen.
In Zeitungen und Zeitschriften gab und gibt es mit Leserbriefen bereits eine rudimentäre Form eines solchen Rückkanals, Radiosender haben im Programm Talkshows, bei denen man anrufen und so „mitmachen“ kann. Auch das Fernsehen hat mit Talkshows und ähnlichen Formaten eine Form von Rückkanal. Keines dieser Medien ist jedoch von Anfang an auf diesen Kanal ausgelegt. Das Internet hingegen bietet hier jedem Raum, seinen eigenen Kanal zu definieren - im Internet kann jeder seine eigene Zeitung, seinen eigenen Radio- bzw. Fersehsender haben, in dem er selbst über den Inhalt entscheidet. Doch darüber hinaus bietet das Internet Menschen die Möglichkeit, „quasi-direkt“ (weil über den Computer) miteinander zu kommunizieren - beispielsweise über Foren, Chats, Mailinglists und Newsgroups.
In der letzten Zeit entstanden mit diversen sozialen Netzwerken (Facebook©, mySpace©™, LinkedIn®©, XING© und vielen weiteren), sowie Blogs und Microblogs (z.B. Twitter©™) immer mehr neue Plattformen zur Interaktion und Kommunikation.
All diese Plattformen haben allerdings eins gemeinsam: sie dienen lediglich der Kommunikation von Menschen über Maschinen - diese stehen immer als Medium zwischen den kommunizierenden Menschen.

Diese Bachelorarbeit befasst sich hingegen mit einer Möglichkeit, Kontakte ohne direkt sichtbare Maschinen zu erkennen und zu nutzen.
Dadurch wird die „alte“ Medienwelt mit der „neuen“ digitalen Welt auf eine neue Weise verknüpft: Man tauscht über physischen Kontakt oder räumliche Nähe digitale Kontaktinformationen aus und interagiert mit der technischen Umgebung.

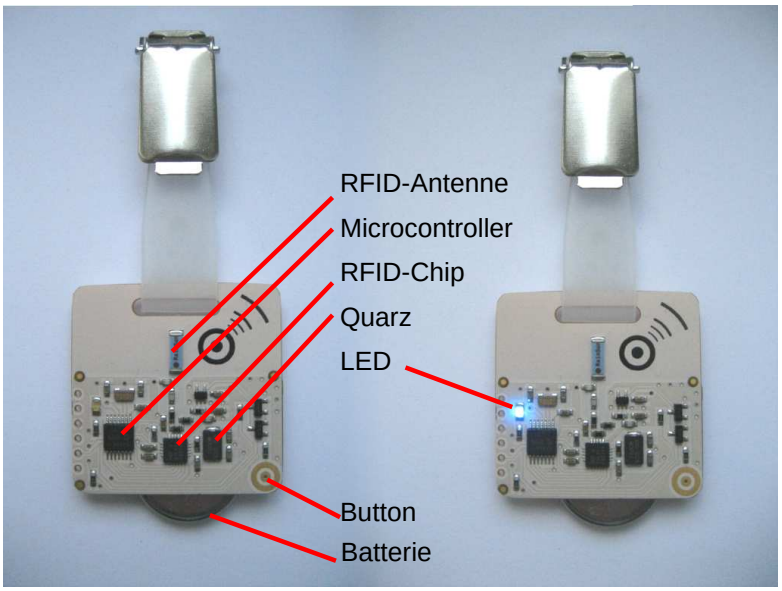


Abbildung 2.1: OpenBeacon-RFID-Tag (ProximityTag) der Firma Bitmanufaktur

2.2 OpenBeacon-Technologie

Die Berliner Firma Bitmanufaktur[1] rief 2006 das Open Source Projekt *OpenBeacon* ins Leben, dessen Ziel es war und ist, eine quelloffene Hard- und Softwareplattform unter GPL-Lizenzierung[2] für den drahtlosen Austausch von Informationen im international lizenzfrei nutzbarem 2.4GHz ISM Frequenzband zu entwerfen und umzusetzen.
Bei der hier entwickelten Lösung handelt es sich bisher um RFID-Tags unter dem Namen „Sputnik“ und dazugehörige Empfänger: „OpenBeacon EasyReader POE[3]“, „OpenBeacon USB-Node“ und seit 2009 „OpenBeacon WLAN“.

RFID[4] ist eine Technologie, mittels derer Gegenstände oder Körper kontaktlos identifiziert werden können.
Dazu wird an dem Gegenstand oder Körper ein sogenanntes RFID-Tag angebracht - ein Transponder. Im Frequenzbereich von ca. 100 kHZ bis etwa 30 MHz funktioniert RFID

[1] http://www.bitmanufaktur.de
[2] GPL: General Public License
[3] POE: Power over Ethernet
[4] RFID: Radio Frequency Identification (*„Identifizierung mit Hilfe elektromagnetischer Wellen“*)

physikalisch durch induktive Kopplung (siehe [l_Fin08, S. 28]). Das bedeutet, dass die Transponder das vom Lesegerät empfangene elektromagnetische Feld beispielsweise über Lastmodulation verändern. Durch Resonanz wird dabei im Tag ein Strom induziert, welcher seiner Ursache entgegen wirkt und so das wirkende Feld beeinflusst. Dies wird vom Transmitter (Lesegerät) in Form von leichten Spannungsabfällen in der Generatorspule registriert und der Transponder, sowie die Daten, die er sendet, werden erkannt.

In den Frequenzbereichen 2,4 GHz und 5,8 GHz werden elektromagnetische Felder zur Kopplung genutzt.

Es gibt dabei zwei unterschiedliche Arten von Transpondern:

Passive Transponder beziehen die Energie zum Auslesen des gespeicherten Wertes und entsprechender Modulation des Feldes des Lesegeräts aus dem Feld des Lesegeräts selbst - entziehen diesem Feld also Energie.

Aktive Transponder besitzen eine eigene Energieversorgung zum Auslesen des Speichers und zur Modulation des Feldes.

Sowohl aktive, als auch passive Transponder erzeugen zunächst selbst keine Felder - sie verändern nur das Feld des Lesegeräts.
Da passive Transponder keine eigene Energieversorgung besitzen und somit auf eine hinreichende Menge Energie aus dem Feld angewiesen sind, ist ihre Reichweite[5] wesentlich eingeschränkter, als die aktiver Tags, welche keine Energie des Feldes benötigen, um ihre Daten zu versenden.
Darüber hinaus ermöglicht eine eigene Energieversorgung des Tags auch das Aufbringen weiterer Chips und einer komplexeren Struktur integrierter Schaltungen zum Beispiel mit weiteren Eingabe- und Ausgabemöglichkeiten, sowie einem größeren Speicher auf aktiven Transpondern.

[5] mit Reichweite ist hier die maximale Distanz zwischen Transmitter und Transponder gemeint, bei der der Transponder das Feld des Transmitters noch ausmachen und modulieren kann

Das OpenBeacon-Projekt befasst sich momentan nur mit aktiver RFID-Technologie und so auch diese Arbeit. Eine der neusten Entwicklungen im OpenBeacon-Projekt sind sogenannte *Proximitytags*. Diese kommunizieren nicht nur mit den in Reichweite befindlichen Transmittern, sondern auch untereinander - sie werden selbst zu Transmittern und lernen sich somit „kennen und sehen“. Diese Eigenschaft lässt sich hervorragend für eine interaktive Nutzung des Systems nutzen und bildet eine unverzichtbare Grundlage dieser Arbeit. Erst mit dieser Technologie kann man verlässliche Aussagen darüber treffen, welches RFID-Tag gerade mit welchen anderen RFID-Tags Kontakt hat. Man kann also feststellen, welche Person gerade welchen Personen begegnet oder beispielsweise auch, welche Person gerade vor welchem Exponat steht.
Dazu müssen sowohl Person, als auch Exponat mit jeweils einem solchen Proximitytag ausgestattet sein und sich in Reichweite von mindestens einer Empfangsantenne befinden.

2.3 Proximity und Ortung

Im vorherigen Abschnitt (2.2) wurde das Prinzip der Kontakterkennung genannt, worüber eine gewisse Art der Ortung bereits möglich ist - man kann Aussagen darüber treffen, welche Personen sich in räumlicher Nähe zueinander befinden. Wenn man sich einen Raum vorstellt, der mit fest montierten Proximitytags gitterförmig überzogen ist (siehe Abbildung 2.2), und man nun einen Menschen mit einem weiterem Proximitytag sich in diesem Raum bewegen lässt, kann man durch die registrierten Kontakte mit den festen RFID-Tags eine rudimentäre Form von Positionserkennung realisieren. Das zu ortende Tag muss sich dabei immer zwischen den Tags befinden, mit denen es Kontakt hat.

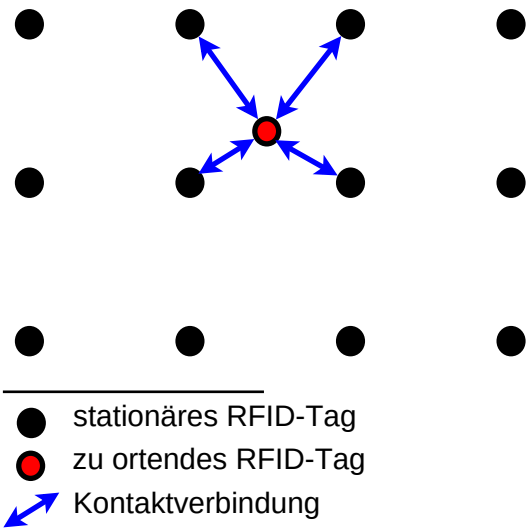


Abbildung 2.2: Ortung über Kontakte mit anderen RFID-Tags

Ortung ist ein zentraler Aspekt in mobilen Netzen: Es besteht ein großes Interesse daran zu wissen oder erkennen zu können, wo sich jemand befindet, wenn er bestimmte Aktionen ausführt oder Daten abruft. Man will über mobile Netze kontextsensitive Informationen vermitteln und der Ort ist einer der wichtigsten Kontexte in vielen mobilen Anwendungen. Ortung ist allerdings auch gleichzeitig eine große Herausforderung, vor allem wenn es um die Ortung in Gebäuden geht. Im eingangs erwähnten Beispiel ist der Installationsaufwand recht hoch (es werden viele Tags benötigt, die alle einzeln fest installiert werden müssen).

Eine andere weit verbreite Vorgehensweise zur Positionsbestimmung ist Trilateration. Über Signalstärken zu mindestens drei Empfangsstationen (RSSI – *Received Signal Strength Indication*) kann mittels Trilateration errechnet werden, wo sich beispielsweise ein RFID-Tag befindet. Dazu wird der Schnittpunkt dreier Kreise um die, durch die Position der Transmitter in der Ebene gegebenen, Punkte berechnet. Der jeweilige Radius des Kreises ergibt sich aus den empfangenen Signalstärken. Im Hinblick auf RFID ist hier die Berechnung der Radien die größte Herausforderung, da die empfangenen Signalstärken

stark durch Abschirmung beeinflusst werden. Weitere Informationen zu RSSI-basierter Ortung finden sich in [l_GSHD08].

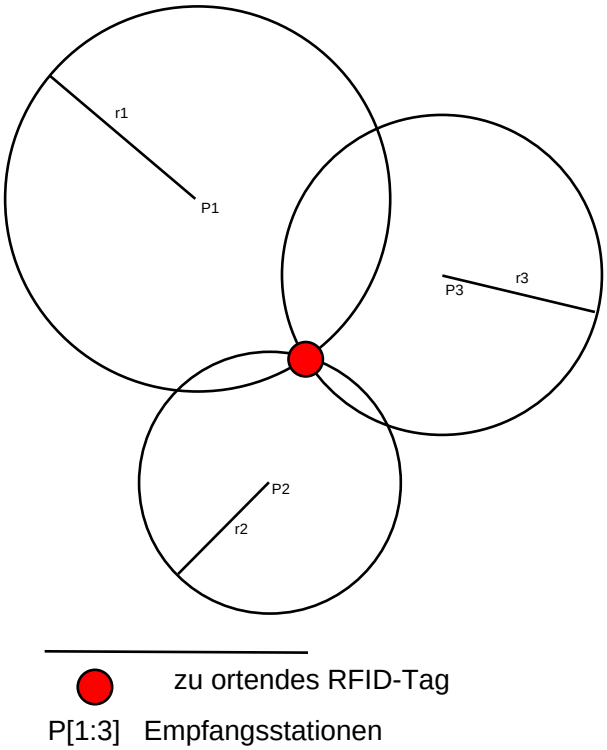


Abbildung 2.3: Ortung mittels RSSI per Trilateration

Aufgrund der durch verschiedene physikalische Faktoren (siehe [l_Dal06, Abschnitt 2.2.2]) verursachten Signalverfälschungen und Summierung der dadurch verursachten Fehler ist Ortung in Gebäuden meist nur raumbasiert zufriedenstellend lösbar.
Bei der aktiven OpenBeacon-Technologie sind die zu ortenden Personen selbst die größten Störfaktoren, denn *„Gebäude und andere Hindernisse wirken als gute Reflektoren und dämpfen eine elektromagnetische Welle bei Transmission (Durchgang) sehr stark“* (Quelle: [l_Fin08, S. 180]). Da die RFID-Tags üblicherweise vor dem Körper und nicht auf dem Kopf getragen werden (wo sie einen nahezu ungehinderten Rundumempfang hätten), sind sie also schon nach hinten sehr stark abgeschirmt. Daher ist auch die sogenannte

Fingerprintingmethode[6], bei der ein Raum im Vorfeld eingemessen wird (ein „Fingerabdruck“ der Signalstärken an verschiedenen Positionen im Raum genommen wird) und anschließend durch Vergleich aktuell empfangener Daten mit den Fingerprintmessungen die Position bestimmt wird, nicht ratsam. Denn durch die Anzahl und Bewegung der Hindernisse verändert sich ständig die physikalische Umgebung, in der gemessen wird. Trotzdem ist eine möglichst genaue Ortung auch für das zu entwickelnde System langfristig zu implementieren, um diesen wichtigen Kontext in die Anwendung einbeziehen zu können.

2.4 ähnliche Systeme

Wie in der Einleitung (Abschnitt 1.1) erwähnt, besteht auf vielen Veranstaltungen Interesse daran, Kontaktdaten auszutauschen. Dieser Austausch soll dabei so intuitiv und einfach, wie nur irgend möglich sein. Es gab in der Vergangenheit bereits einige Bemühungen, diesen Vorgang zu digitalisieren, beziehungsweise andere Entwicklungen, die sich zu diesem Zweck nutzen lassen. Drei Beispiele solcher Entwicklungen werden in diesem Abschnitt näher beleuchtet.

2.4.1 QR-Codes

QR-Codes wurden 1994 von der japanischen Firma Denso Wave eigentlich für die Markierung von Komponenten und Baugruppen in der Logistik und Automobilproduktion entwickelt. Sie erlangten jedoch vor allem in Japan schnell auch in vielen weiteren Bereichen des alltäglichen Lebens große Beliebtheit. Mittels QR-Codes können Informationen zweidimensional codiert werden.

[6] oft auch als Radiomap-Verfahren, bezeichnet (Quelle: [l_Dal06])

Abbildung 2.5: Beispiel eines QR-Codes [b_Sch08]

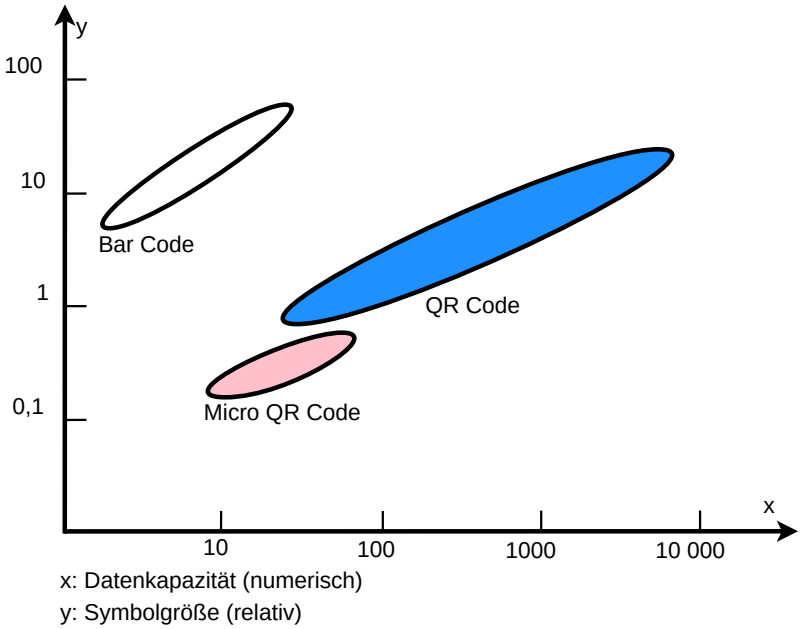


Abbildung 2.4: Datenkapazität relativ zur Größe von QR-Codes im Vergleich nach: [b_den10]

Dabei ist die Datenkapazität im Vergleich zu Barcodes aufgrund der Verwendung der zweiten Dimension wesentlich höher, wie Abbildung 2.4 zeigt. Darüber hinaus sind QR-Codes bei bis zu 30% (günstiger) Zerstörung immer noch maschinell lesbar.

Im Bezug auf Kontaktverwaltung auf Veranstaltungen könnte nun jeder Besucher einen solchen QR-Code mit sich tragen, in dem all seine Kontaktdaten codiert sind. Ausge-

tauscht werden die Informationen, indem man vom Gesprächspartner den QR-Code mit einem entsprechendem Lesegerät (beispielsweise ein Mobiltelefon) scannt.

Der offensichtliche Nachteil an diesem System ist, dass jeder Teilnehmer einen solchen Code braucht und dazu noch ein entsprechendes Lesegerät. Inzwischen können zwar viele Mobiltelefone bereits QR-Codes lesen (auf jedem Java-fähigem Mobiltelefon kann eine entsprechende Applikation installiert werden), jedoch bei weitem noch nicht alle (zusätzlich ist zur JVM[7] auch noch eine integrierte Kamera notwendig).

2.4.2 Hoccer

Eine im Vergleich zu QR-Codes noch recht neue Entwicklung entstammt der Berliner Firma ART+COM[8]. Hoccer ist ein Programm für das Android (und inzwischen auch für das iPhone) Smartphonebetriebssystem, mit dem man sich Daten „zuwerfen“ kann. Über den Beschleunigungssensor wird ermittelt, wann sich das Gerät in welche Richtung ruckartig bewegt.

Mittels dieser *Gesten* können Daten in die entsprechende Richtung geworfen werden. Befindet sich ein weiteres Android- oder iPhone-Smartphone in der Nähe und führt dessen Benutzer eine fangende Geste aus, werden die „geworfenen“ Daten hier gefangen.

Die angesprochene räumliche Nähe wird dabei über GPS-Koordinaten[9] ermittelt. Inzwischen (Stand: Juni 2010) wird auch ein auf beiden Geräten verfügbares WLAN als Indikator für räumliche Nähe ausgewertet.
In einem kurzen Zeitfenster wird nun geprüft, ob in Nähe zu dem „werfenden“ Smartphone ein anderes Smartphone eine Fangbewegung ausführt.
Es ist ebenso möglich, Daten an mehrere Smartphones mit einer Geste zu übertragen

[7] JVM: Java Virtual Machine - ein Programm, welches Java Bytecode ausführen kann
[8] `http://www.artcom.de/`
[9] GPS: Das Global-Positioning-System ist ein globales Navigationssystem, welches über Satelliten funktioniert, die ständig die genaue Uhrzeit ausstrahlen. Über die Signallaufzeitdifferenz der empfangenen Signale von mindestens drei Satelliten kann ein Empfänger seine aktuelle Position bestimmen.

- diese müssen dann in dem Zeitfenster alle die Fangbewegung in Nähe des „Werfenden“ ausführen (siehe [l_QM10]).
Die Daten werden über Server von ART+COM zwischen den Smartphones ausgetauscht - eine direkte Kommunikation zwischen den mobilen Geräten findet derzeit nicht statt. Darüber hinaus ist es mit Hoccer möglich, Daten fest an bestimmte Positionen zu setzen. Diese können dort anschließend immer wieder „gefangen“ werden. Somit eignet sich Hoccer nicht nur zum Austausch von (Kontakt-)Daten, sondern auch als Geoinformationssystem.

2.4.3 Facebook's f8 Konferenz

Sehr ähnlich dem im Rahmen dieser Arbeit zu entwickelndem System ist ein von Facebook initiiertes System. Auf der f8-Konferenz in San Francisco hat Facebook ein System mit passiven RFID-Tags vorgestellt und verwendet. Hierbei bekam jeder Besucher der Konferenz ein passives RFID-Tag und konnte an diversen Stationen „einchecken“. So konnte man sich beispielsweise an Fotoboxen mit dem RFID-Tag identifizieren und ein Foto von sich machen lassen, welches dann automatisch auf das Facebook-Profil des Besuchers geladen wurde (weitere Informationen dazu unter: [l_O'N10]).

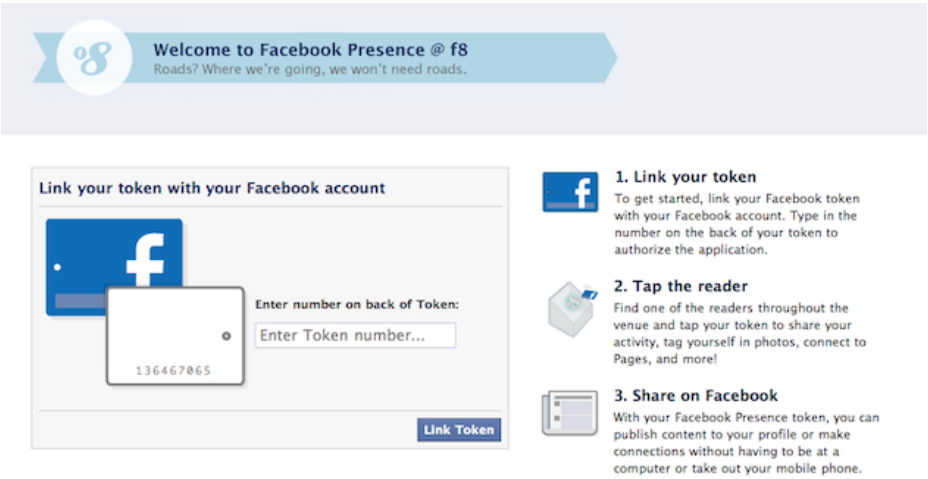


Abbildung 2.6: Webseite für die Nutzung der RFID-Tags auf der f8 [b_fac10]

Abbildung 2.6 zeigt die Webseite, auf der man sich einloggen musste, um das RFID-Tag zu seinem Facebook-Account hinzuzufügen. Danach konnte man die Stationen auf der Veranstaltung nutzen.

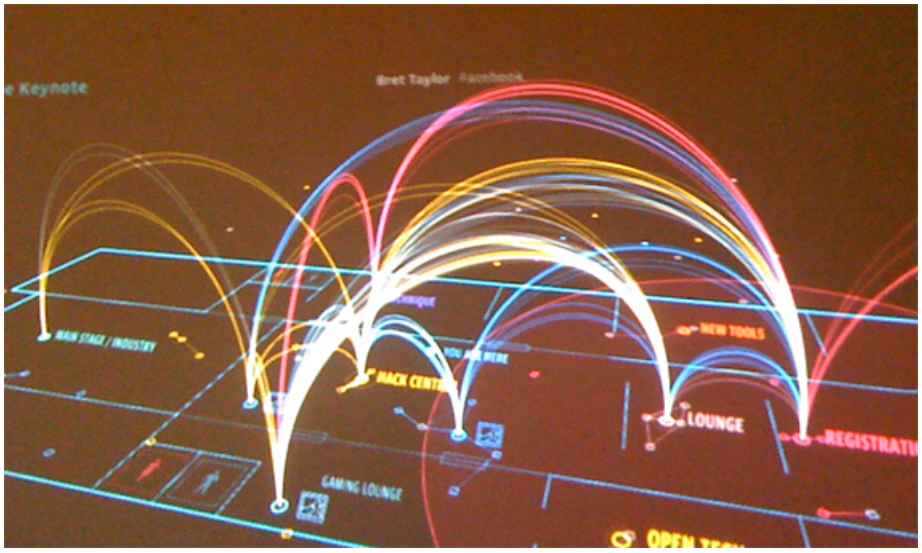


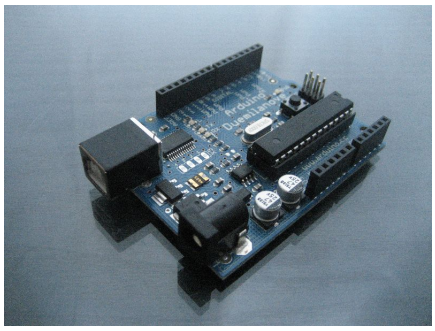
Abbildung 2.7: Visualisierung der Daten auf der f8 [b_O'N10]

Die Informationen darüber, welche Benutzer wo welche Stationen genutzt haben, wurden dabei live visualisiert, wie Abbildung 2.7 zeigt.

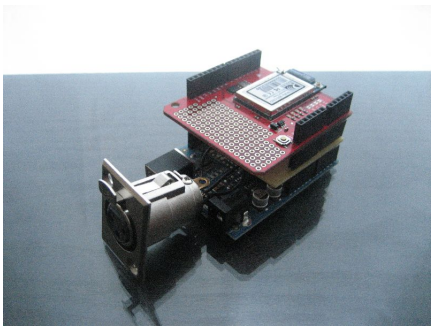
2.5 Physical Computing und Arduino

In Abschnitt 1.1 wurde bereits von zunehmender Digitalisierung gesprochen: Das alltägliche Leben wird immer mehr durch technische Geräte bestimmt. Viele Gegenstände des täglichen Gebrauchs enthalten kleine eigenständige Computer oder Mikrocontroller [l_OFW09]. Diese steuern nicht mehr nur Abläufe, wie beispielsweise in einer Waschmaschine das Waschprogramm, sondern reagieren und kommunizieren inzwischen über physikalische Schnittstellen mit ihrem Umfeld und dem Benutzer. Über Sensoren werden Veränderungen in der Umgebung erfasst und über Aktoren wird Einfluss auf die Umwelt genommen [l_Gün10].

Arduino ist ein System, bestehend aus einer Entwicklungsumgebung und einem damit

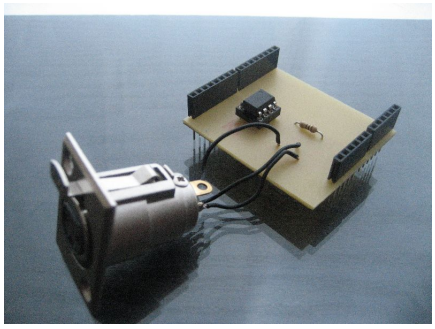


(a) das Arduino Entwicklungsboard

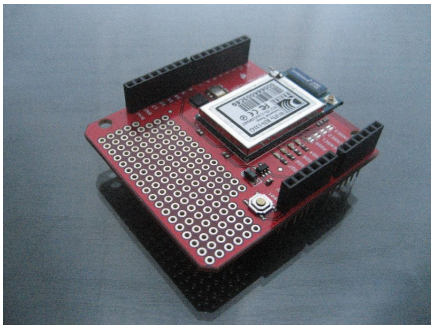


(b) gestapelte Shields auf dem Arduinoboard

Abbildung 2.8: Arduino Hardware



(c) ein DMX-Shield



(d) das WiFly WLAN-Shield

Abbildung 2.8: Arduino-Shields

programmierbaren Entwicklungsboard (siehe 2.8a), welches über verschiedene elektrische Ein- und Ausgänge für Aktoren und Sensoren verfügt. Die Programmiersprache für das Arduino wurde ausgehend von der Entwicklungsumgebung Processing[10] entworfen und basiert auf Wiring[11], was wiederum C ähnlich ist. Sowohl die Entwicklungsumgebung als auch das Boarddesign sind im Sinne von Open Source frei verfügbar [w_Arda]. Das Board kann über sogenannte Shields erweitert werden, die passgenau auf die Pin-Anordnung abgestimmt sind und auf die Boardpins aufgesteckt werden. Verschiedene Kombinationen von Shields (z.B. WLAN-Shield (Abbildung 2.8d) und DMX-Shield (Abbildung 2.8c)) ermöglichen so die Entwicklung komplexerer Systeme (Abbildung 2.8b).

[10] `http://processing.org/`
[11] `http://www.wiring.org.co/`

2.6 serielle Kommunikation

Serielle Kommunikation zwischen Computerperipherie verliert auch in der letzten Zeit keinesfalls an Bedeutung. Vor allem in eingebetteten Systemen, auf den untersten Abstraktionsebenen von vielen anderen Geräten und schließlich auch bei vielen Formen der Kommunikation mit Peripherie ist serielle neben der parallelen Datenübertragung weit verbreitet. Das liegt vor allem daran, dass serielle Datenströme oft schneller getaktet werden können, als parallele Datenströme, da hier der sogenannte Clock-Skew (Zeitversatz aufgrund von Leitungsparametern) nicht berücksichtigt werden muss - es gibt nur eine Leitung. Auch ist eine serielle Schnittstelle oft einfacher und kostengünstiger zu implementieren, da weniger Pins benötigt werden. USB (Universal Serial Bus) ist eines der momentan populärsten seriellen Bussysteme, die am Markt vertreten sind. Es dient zur Kommunikation von „klassischer“ Computerperipherie (Drucker, Scanner, Maus, Tastatur, Massenspeicher) mit dem Computer.

In diesem Kapitel werden im folgenden zwei andere serielle Bussysteme näher beleuchtet, die für die Implementierung eine zentrale Rolle spielen werden:

SPI ist ein serielles Bussystem für die Kommunikation zwischen digitalen Schaltungen.

DMX ist ein serielles Bussystem, dass meist in der Veranstaltungstechnik Anwendung findet und zur Steuerung selbiger benutzt wird.

2.6.1 SPI - Serial Peripheral Interface

Das Serial Peripheral Interface (kurz SPI) dient der *„synchrone serielle Kommunikation von Hostprozessor und Peripheriebausteinen“* [l_Sch00].
Dieser Standard wurde von Motorola entwickelt und beschreibt lediglich die Hardwarefunktionsweise und definiert kein Softwareprotokoll dazu. Obwohl es kein offiziellen vollständigen Standard oder eine Norm gibt, erlangte SPI wohl wegen der einfachen Implementierung und vor allem weil es keinen Patenten unterworfen wurde, große Bedeutung.

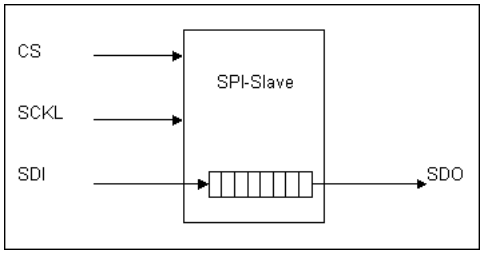


Abbildung 2.9: SPI-Slave Aufbau [b_Sch00]

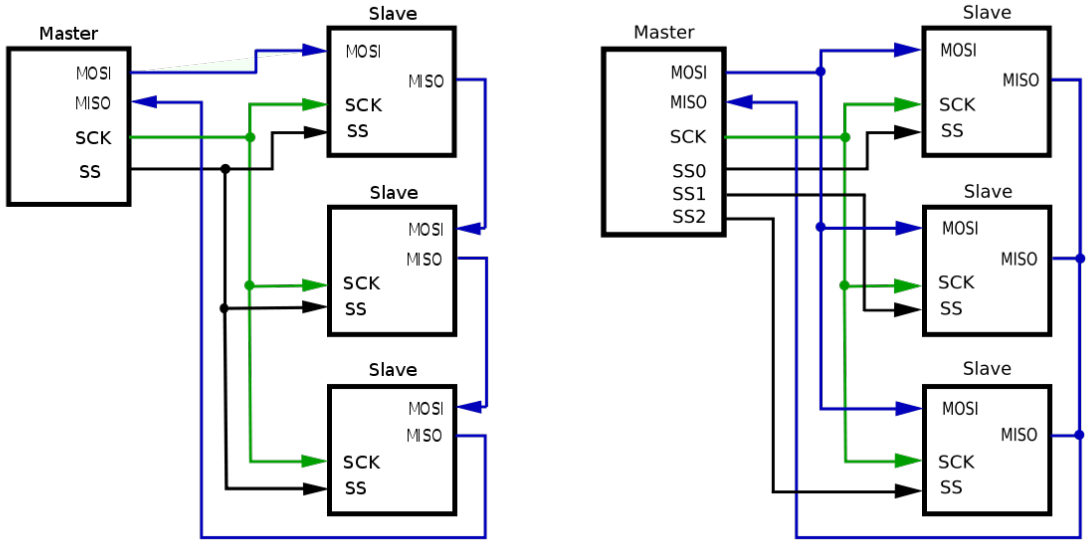
Ein anderes sehr populäres Bussystem ist das von Philips entwickelte und seit dem 1. Oktober 2006[12] lizenzkostenfrei nutzbare I²C-Protokoll. Die Unterschiede zwischen SPI und I²C sind in [l_Ran02] tabellarisch gegenübergestellt.

Bei SPI handelt es sich um ein Master-Slave Bussystem. Der Master gibt mit Clock (SCKL) das Taktsignal vor und aktiviert über Chip-Select (CS) oder auch Slave-Select (SS) den Peripheriebaustein, mit dem er kommunizieren will bzw. an den der Befehl geht. Die weitere Kommunikation findet über die MOSI (Master Out Slave In) und MISO (Master In Slave Out) Leitungen statt.

Da es immer nur ein Taktsignal geben kann, ist die Anzahl der Masterbausteine auf einen und die der Slavebausteine auf die Anzahl der Chip-Select-Leitungen in dem Masterbaustein begrenzt. Zur Verbindung der einzelnen Komponenten sieht SPI eine Kaskadierung der Slaves oder eine Sterntopologie vor (Abbildung 2.10) [l_Sch00].

SPI eignet sich - wie eingangs erwähnt - vor allem zur Kommunikation von Peripheriebausteinen mit dem dazugehörigem Hostprozessor. *„Da es keinerlei Störschutz gegen externe Signale gibt“* [l_Büc06], ist eine Anbindung externer Komponenten über SPI nicht ratsam und wird auch in der Regel in der Praxis nicht vorgenommen (eine Ausnahme bilden MMC-Speicherkarten, obwohl selbst diese im Prinzip direkt (ohne Kabel) angebunden werden). Man findet SPI daher vor allem direkt auf Platinen wieder. So auch beispielsweise auf den Proximitytags des OpenBeacon-Projekts, deren Antenne[13] an den Mikroprozessor per SPI angebunden ist.

[12] Quelle: `http://de.wikipedia.org/wiki/I2c`
[13] Datenblatt: `http://www.nordicsemi.no/files/Prod_brief_RFSilicon_nRF24L01.pdf`



(a) Kaskadierung [b_wika] (b) Stern [b_wikb]

Abbildung 2.10: Topologien von SPI

2.6.2 DMX und Veranstaltungstechnik

Das zu entwickelnde System soll bei größeren Veranstaltungen verwendet werden und hier eine Form der Besucherinteraktion auch mit der Veranstaltungstechnik ermöglichen. Dazu ist es notwendig, einige Grundlagen der Veranstaltungs- und insbesondere Lichttechnik zu erläutern. Dabei soll herausgearbeitet werden, welche Schnittstellen es gibt und wie man diese im Rahmen dieser Arbeit nutzen kann.

Früher wurde auf Veranstaltungen jede Lampe beziehungsweise auch einzelne Komponente einer Lampe (z. B. Motoren) mit einer eigenen Litze versehen, verkabelt und über diese gesteuert. Das führt bei größeren Veranstaltungen und umfangreicher werdenden Funktionalitäten der verwendeten Lampen (Farbwechsel, Dimmer, Motor etc.) zu immer mehr Kabeln, die zu einer Lampe gehen und somit zu immer mehr Fehlerquellen. Daher ersann man bald ein serielles Protokoll zur Steuerung mehrere Geräte über einen Bus, auf dem verschiedene Geräte an verschiedenen Adressen oder Adressbereichen auf Werte lauschen - DMX512. Damit wird es möglich, an ein Kabel hintereinander (Bus) bis zu 32 DMX-Geräte mit insgesamt bis zu 512 einzeln kontrollierbaren Parametern (Kanä-

len) zu installieren. Um mehr als 32 Geräte anzuschließen müssen sogenannte Splitter oder Booster zugeschaltet werden, da ansonsten der Spannungsabfall an jedem einzelnen Gerät des Busses zu hoch wäre.
DMX 512 wurde 1986 vom USITT (United States Institute for Theatre Technology) erstmals genormt und basiert auf EIA-485 (auch als RS-485 bezeichnet) [l_Ber08].
Seit 2000 gibt es auch eine deutsche Norm für DMX: DIN 56930[14].
RS-485 bezeichnet einen Schnittstellen-Standard für digitale, leitungssgebundene, differenzielle und serielle Datenübertragung.[15] Differenziell bedeutet hierbei, dass für die Datenübertragung zwei Leitungen benötigt werden: an einer liegt der invertierte und an der anderen der nicht invertierte Pegel des 1-Bit Datensignals, welches übertragen wird.

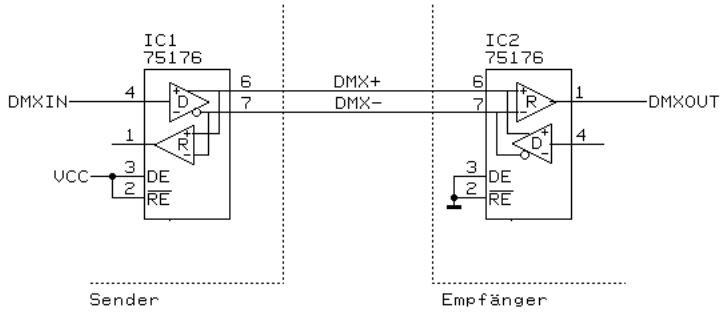


Abbildung 2.11: Schema der Übertragungseinheiten für DMX-Signale [b_soub]

Aus der Differenz wird vom Empfänger das ursprüngliche Signal rekonstruiert. Damit wird das generelle Problem von Leitungsstörungen bei der Übertragung elektrischer Signale umschifft, denn Störungen wirken auf beide Adern gleichermaßen und heben sich bei der Differenzbildung wieder auf. Dadurch und durch die Verdopplung des Ausgangssignals durch diese Differenzbildung, eignet sich diese Art der Übertragung hervorragend auch für große Übertragungsstrecken über einen Kilometer hinweg. Weitere Details zu RS-485 sind in [l_WB09, S. 253 ff.] näher erläutert. Darüber hinaus wird in [l_SGD09, S. 156] erwähnt, dass die Pegel von SPI (vgl. Unterabschnitt 2.6.1) auf RS-485 gewandelt werden können.

[14] DIN 56930-1: http://www.beuth.de/langanzeige/DIN+56930-1/de/10645640.html
DIN 56930-2: http://www.beuth.de/langanzeige/DIN+56930-2/de/10645679.html
DIN 56930-3 (seit 2010): http://www.beuth.de/langanzeige/DIN+56930-3/de/122789926.html
[15] Quelle: http://de.wikipedia.org/wiki/RS-485

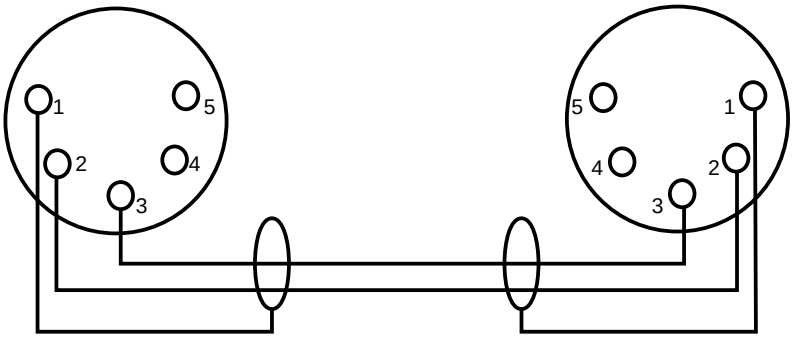


Abbildung 2.12: Verbindungsschema zweier DMX-Komponenten nach: [b_ard]

Laut Spezifikation hat ein DMX-Anschluss bzw. Kabel fünf Pole: Zwei Kanäle zu den Geräten (invertiert und nicht invertiert), zwei Kanäle von den Geräten zum Master (invertiert und nicht invertiert) und eine Masseleitung. Da die Rückkanäle jedoch meist nicht genutzt werden und es mit XLR-Kabeln bereits eine günstige dreiadrige Variante von Kabeln gab, die ebenso ein invertiertes und ein nicht invertiertes Signal neben der Masse transportieren, sind heute in DMX-Installationen meist nur noch dreiadrige Kabel zu finden (siehe auch Abbildung 2.12).
RS-485 ist eigentlich multimasterfähig. Diese Fähigkeit wird bei DMX nicht genutzt - es gibt immer nur ein Kontrollgerät für alle Slaves.
Das Protokoll, über das der DMX-Master mit den DMX-Slaves kommuniziert baut sich wie folgt auf:

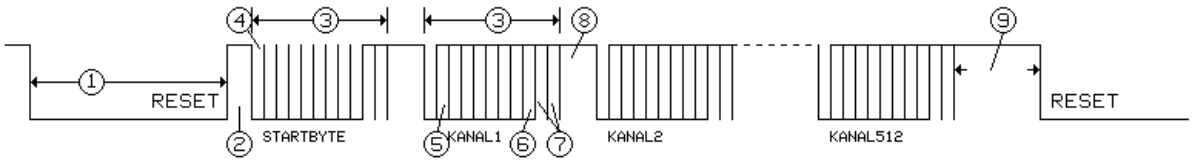


Abbildung 2.13: Das DMX-512 Protokoll [b_soua]

Nr.	Signalname	Min.	Typ.	Max.	Einheit
1	RESET	88	88		μs
2	MARK zw. RESET und Startbyte	8	-	1s	μs
3	Frame-Zeit	43,12	44,0	44,48	μs
4	Startbit	3,92	4,0	4,08	μs
5	LSB (niederwertigstes Datenbit)	3,92	4,0	4,08	μs
6	MSB (höchstwertigstes Datenbit)	3,92	4,0	4,08	μs
7	Stoppbit	3,92	4,0	4,08	μs
8	MARK zwischen Frames (Interdigit)	0	0	1,00	s
9	MARK zwischen Paketen	0	0	1,00	s

Tabelle 2.1: Erklärung der Werte in Abbildung 2.13 [w_sou99]

Dabei werden in einem Zyklus alle 512 Werte nacheinander mit einer Geschwindigkeit von 250 kBit/s übertragen. Das bedeutet, dass ein Bit $4\mu s$ benötigt und somit $44\mu s$ pro Datenwort für die Übertragung benötigt werden. Daraus ergibt sich eine Gesamtübertragungsdauer von $88 + 8 + 44 + 512 * 44 = 22668\mu s$ für einen Zyklus und somit eine Wiederholungsrate von maximal $44,1Hz$ [l_Ack06]. Ein akkurates Timing ist dabei von essenzieller Bedeutung, da innerhalb eines Zyklus nicht anders als über die Zeit auseinander gehalten werden kann, welches Byte zu welchem der 512 Kanäle gehört. Ein Zyklus wird immer mit einem Startbyte begonnen und mit einem Reset beendet. Danach ist maximal eine Sekunde Pause, bis der Zyklus erneut durchlaufen wird. Dabei kann ein Zyklus an beliebiger Stelle durch Reset unterbrochen und nach beliebiger Zeit (minimal jedoch $88\mu s$) neu gestartet werden. Da jedes Gerät im Bus alle 512 Werte bekommt, können auch mehrere Geräte an denselben Kanälen auf Werte lauschen.

DMX als Standard erfuhr in den letzten Jahren keine große Weiterentwicklung - es ist inzwischen möglich, andere Startbytes, als 0x00h zu senden und $4\mu s$ MARK zwischen RESET und Startbyte reichen für einige Geräte offiziell aus. Aber mit Art-Net gibt es dennoch eine größere Neuerung rund um DMX. Art-Net ist ein IP-basiertes Protokoll

auf Basis von UDP, mit dem bis zu 40 Universen (jeweils 512 DMX-Kanäle pro Universum) bedient werden können [l_Lic10].

Dabei werden die Nachrichten aus dem Art-Net-Protokoll anschließend durch *„so genannte Art-Net-Knoten, die die IP-Informationen in Standard DMX-512-Signale zurückverwandeln “* (Quelle: [w_dmx]) ausgewertet und können so normale DMX-Geräte ansteuern.

2.7 Zusammenfassung

Es gibt bereits eine Reihe von Plattformen und Technologien, die für das Austauschen von Kontaktinformationen genutzt werden können. Sowohl bei Hoccer, den QR-Codes, als auch auf der f8-Konferenz und einigen weiteren hier nicht näher betrachteten Anwendungen ist dafür allerdings immer eine Aktion des Benutzers erforderlich, um diese Daten auszutauschen.
Mittels RFID-Technologie kann man heutzutage Kontakte zwischen Tags erkennen. Die Abschirmung der RFID-Tags durch den menschlichen Körper kann dabei dazu genutzt werden, nur vis-à-vis-Kontakte zu erkennen. Somit ist der Grundstein für die automatisierte Kontakterkennung hier bereits gelegt.
Für Veranstalter ist nicht nur der direkte Mehrwert für die Gäste interessant - sie wollen dem Besucher besondere Erinnerungen verschaffen. Dies kann durch interaktive Elemente auf der Veranstaltung passieren, indem mit DMX Veranstaltungstechnik in Abhängigkeit von Besucheraktionen gesteuert wird.

3 Anforderungsanalyse

Nach einer kurzen Einführung in die Zielsetzungen des Projektes ROBERTA und Beschreibung der Anwendungsumgebung werden Anwendungsfälle aufgeführt. Abgeschlossen wird das Kapitel mit einem Abschnitt zum Thema Datenschutz.

3.1 ROBERTA-Projekt

ROBERTA steht für *„Revaluation of OpenBeacon for Business Event RFID Technology Applications“* und ist ein Projekt, dass die BureauQ GmbH Berlin[1] zusammen mit der Forschungsgruppe INKA[2] der HTW Berlin durchführt. Ziel des Projektes ist eine Machbarkeitsstudie zum Einsatz von aktivem RFID im Kontext von öffentlichem Veranstaltungsmanagement. Es sollen Bewegungen und Aktionen von Besuchern auf Veranstaltungen erfasst werden, um Kontakte zu rekapitulieren, Verbindungen in sozialen Netzwerken sichtbar zu machen und zu knüpfen. Die BureauQ GmbH Berlin ist eine Eventmanagement-Agentur, die unter anderem Firmenveranstaltungen für die Deutsche Bank, Bertelsmann oder auch Volkswagen ausrichtet. Mit der aktiven RFID-Technologie der Berliner Firma Bitmanufaktur, mit der die Forschungsgruppe INKA bereits Erfahrungen sammeln konnte, sollen im Projekt ROBERTA Veranstaltungen interaktiver gestaltet werden.

[1] `http://www.bureau-q.com/`
[2] `http://inka.htw-berlin.de`

3.2 Anwendungsumgebung

Gäste von Veranstaltungen wie beispielsweise Tagungen tragen oft Namensschilder und kommunizieren miteinander. Diese Namensschilder werden durch aktive RFID-Tags erweitert, die erkennen können, welche anderen RFID-Tags sich in ihrer näheren Umgebung befinden. Die so gewonnenen Kontaktinformationen sollen für verschiedene Zwecke aufbereitet werden und zur Weiterverarbeitung durch weitere Komponenten des Systems zur Verfügung gestellt werden.

3.3 Szenarien

Im Rahmen des Projektes sollen verschiedene Anwendungsszenarien durchgespielt werden, die im Folgenden näher erläutert werden.

3.3.1 Visitenkarten ersetzen

Auf Tagungen und Konferenzen werden für gewöhnlich unter den Besuchern Visitenkarten getauscht, um eine nachhaltige Kommunikation zu ermöglichen, bzw. sein persönliches Kontaktnetzwerk zu erweitern. Diese Visitenkarten in reiner Papierform sind jedoch inzwischen nicht mehr zeitgemäß (vgl. Abschnitt 1.1). So ist es zwar prinzipiell möglich Daten der digitalen Welt auf eine Visitenkarte zu drucken (wenngleich hier schon die Größe der Visitenkarte Grenzen setzt), es ist jedoch meist ein erheblicher Aufwand, all diese Informationen wieder zu digitalisieren. Um Daten in digitaler Form auszutauschen, gibt es verschiedene Ansätze, wie in Kapitel 2 erläutert wurden.
Durch die Möglichkeit, Kontakte ohne das aktive Zutun der Gäste zu erkennen, kann dieser Austausch von Kontaktinformationen automatisiert werden, indem man Kontakte zwischen RFID-Tags speichert und in einer Datenbank eine eindeutige Zuordnung zwischen RFID-Tag und Person schafft.
Besucher können dann beispielsweise auf einer Webseite ihre Kontaktinformationen eintragen und sehen, mit wem sie sich wann gegenüber standen. Zu diesen Personen können

sie nun über deren hinterlegte Daten in weitere Korrespondenz treten.

3.3.2 Ortung von Personen in Innenräumen

Es ist zu erwarten, dass die RFID-Tags nicht immer zuverlässig Kontakte auf eine bestimmte Distanz zwischen Personen registrieren. In Abhängigkeit von den Umgebungsbedingungen (zum Beispiel Anzahl der Personen im Raum und die räumliche Aufteilung und daraus resultierende Signalverfälschungen) ist damit zu rechnen, dass auch Kontakte vom System registriert werden, die eigentlich gar keine waren. Um diese Fehler einzudämmen und gleichzeitig eine breite Palette von neuen Interaktionsmöglichkeiten zu eröffnen, ist eine möglichst genaue Ortung der RFID-Tags im Raum wünschenswert.

Wie in Kapitel 2.3 bereits erwähnt ist die Ortung eine große Herausforderung. Es lassen sich allerdings zahllose Szenarien beschreiben, in denen eine Ortung von Personen zur Steigerung von Interaktivität auf Veranstaltungen beitragen kann. So kann man beispielsweise über einem bestimmten Punkt im Raum ein Licht in einer bestimmten Farbe angehen lassen, wenn darunter gerade eine bestimmte Anzahl von Personen Kontakt hat. Man kann Musik auf der Tanzfläche automatisch lauter werden lassen, wenn diese sich füllt. Dazu kann man automatisch eine Lichtshow starten. Man kann die Stimmung im Raum durch Beleuchtung so verändern, dass wechselnde Orte den Mittelpunkt bilden - abhängig von den Personen, die sich dort befinden.
Es sind hier allein durch Vorstellungskraft und dem Erfindungsreichtum des Veranstalters Grenzen gesetzt. Nicht zuletzt können über eine Ortung und deren Kombination mit anderen Umgebungsinformationen wie Kontakten und dem Gedrücktsein von Knöpfen auch Spielideen entwickelt werden.

Im Rahmen dieser Arbeit können aufgrund des zu erwartenden Umfangs bei Entwurf und Implementierung allerdings nur die Grundsteine für eine Ortung gelegt werden.

3.3.3 Veranstaltungstechnik steuern

Im vorherigen Abschnitt wurde die Möglichkeit, Veranstaltungstechnik über ortsabhängige Informationen zu steuern, kurz erwähnt. Natürlich lässt sich Veranstaltungstechnik auch ohne Ortung steuern. Allein die Informationen über Kontakte, die stattfinden reicht für viele Szenarien aus. Eine weitere Anwendung wäre beispielsweise eine Art Füllstandsanzeige für Veranstaltungsräume: In Abhängigkeit von der Menge von Personen, die über einen bestimmten Empfänger oder eine Anzahl von Empfängern registriert werden, kann am Eingang des Raums eine andere Farbe leuchten.

3.4 Use-Cases

Allgemein lassen sich die oben genannten Szenarien durch die in Abbildung 3.1 aufgeführten Anwendungsfälle umsetzen. Es ist wichtig, zu erkennen, wann ein Gast das Areal betritt und verlässt, um Aussagen darüber zu treffen, wie viele Tags sich gerade in einem Areal befinden. Für das Visitenkartenszenario ist es unabdingbar, Kontakte zu erkennen und zu speichern - dazu muss erkannt werden, wann und wie lang sich jeweils zwei Besucher vis-à-vis gegenüber stehen. Da die RFID-Tags auch einen Knopf zur Verfügung stellen und Nachrichten darüber verschicken, wenn der Knopf gedrückt wird, ist auch dieser Anwendungsfall von dem Softwaresystem zu berücksichtigen.

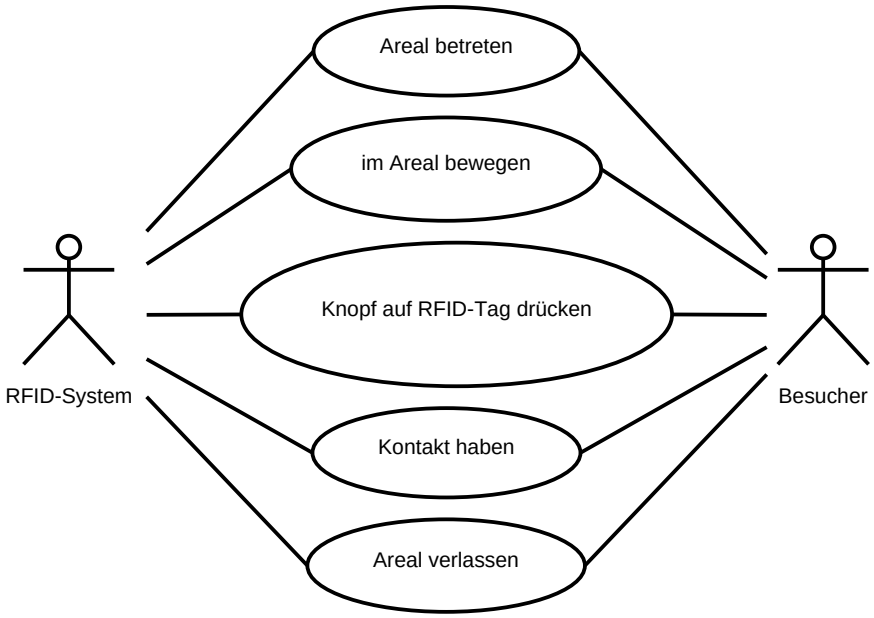


Abbildung 3.1: Anwendungsfälle zwischen dem Besucher und dem System

3.5 Kriterien

An das gesamte System werden in vielerlei Hinsicht Anforderungen an die Qualität gestellt. Diese sind in Tabelle 3.1 aufgeführt. Hierbei werden verschiedene Kriterien in Gruppen wie Funktionalität, Zuverlässigkeit, Benutzbarkeit, Effizienz, sowie Änderbarkeit und Übertragbarkeit gegliedert. Es wurde zum Beispiel schon beschrieben, dass das System um verschiedensten Anforderungen durch die wechselnden Veranstaltungshintergründe zu genügen sehr einfach erweiterbar sein muss und insgesamt sehr stabil laufen soll. Es ist nicht denkbar, dass das System auf der Veranstaltung ausfällt und Kontakte nicht mehr registriert werden.

Produktqualität	**sehr gut**	**gut**	**normal**	**nicht relevant**
Funktionalität				
Angemessenheit		x		
Richtigkeit			x	
Interoperabilität		x		
Ordnungsmäßigkeit			x	
Sicherheit		x		
Zuverlässigkeit				
Reife		x		
Fehlertoleranz		x		
Wiederherstellbarkeit		x		
Benutzbarkeit				
Verständlichkeit		x		
Erlernbarkeit			x	
Bedienbarkeit		x		
Effizienz				
Zeitverhalten		x		
Verbrauchsverhalten		x		
Änderbarkeit				
Analysierbarkeit		x		
Modifizierbarkeit		x		
Erweiterbarkeit	x			
Stabilität	x			
Prüfbarkeit			x	
Übertragbarkeit				
Anpassbarkeit		x		
Installierbarkeit		x	x	
Konformität			x	
Austauschbarkeit		x		

Tabelle 3.1: Qualitätskriterien des zu entwickelnden Systems

Dennoch gibt es Differenzierungen, die aus dieser Tabelle nicht hervorgehen. So ist beispielsweise die Bedienbarkeit ein Punkt, der zweigeteilt betrachtet werden muss: Den Besuchern soll die Bedienung des Systems so intuitiv vorkommen, als würden sie das System gar nicht bedienen - es soll selbstverständlich sein, dass die Kontakte gesammelt werden. Hingegen die Bedienbarkeit aus Sicht des Administrators des Systems, der es zum Laufen bringt und am Laufen hält, muss nur „gut“ sein. Ein Administrator, der einmal den Umgang mit dem System kennt, muss es auch nicht mehr erlernen. Dennoch sollte das System für den Benutzer (in dem Fall die Besucher der Veranstaltung) verständlich und weitestgehend selbsterklärend sein.
Wichtige Kriterien für den Administrator sind die Zuverlässigkeit und Effizienz des Systems. Dem Entwickler zusätzlicher Anpassungen ist entsprechend die Änderbarkeit und Übertragbarkeit sehr wichtig, wohingegen dem Besucher der Veranstaltung letztendlich die Benutzbarkeit und Funktionalität des Systems am wichtigsten ist.

3.6 Datenschutz und Datensicherheit

Wenn man Besuchern auf Veranstaltungen RFID-Tags gibt, kann sehr schnell der Gedanke an Überwachung aufkommen. Es geht den Besuchern dabei darum, ihre personenbezogene Daten vor dem Missbrauch durch Dritte zu schützen. In der angestrebten Anwendung geht es dabei um Kontaktdaten von Benutzern, die höchst sensible Informationen darstellen.
Im zu entwickelndem System ist daher insbesondere darauf zu achten, dass personenbezogene Daten nur da verwendet werden, wo sie unabdingbar für die Funktionalität sind. Datensparsamkeit steht im System an erster Stelle.

Alle Nachrichten der RFID-Tags werden bereits nach dem XXTEA-Verfahren[3] vor dem Senden verschlüsselt. Das zu entwickelnde System soll ebenfalls eine Möglichkeit bieten,

[3]XXTEA ist eine Weiterentwicklung von XTEA, was wiederum eine Weiterentwicklung von TEA darstellt. Es ist ein symmetrisches Verschlüsselungsverfahren, welches sich durch die Verwendung einfacher Grundoperationen auszeichnet und somit vor allem im Bereich der eingebetteten Systeme Anwendung findet

den gesamten Nachrichtenverkehr zu verschlüsseln. Dabei kommen sowohl symmetrische, als auch asymmetrische Verschlüsselungsmethoden infrage. Bei einer verteilten Anwendung mit Clients und einem Server ist es zum Beispiel denkbar, dass sich jeder Client zunächst beim Server authentifiziert, um dann die gewünschten Informationen zu bekommen.

Es ist darüber hinaus sicherzustellen, dass Besucher von Veranstaltungen selbst entscheiden, ob sie ihre Kontakte durch das System sammeln lassen wollen und mit der Technik interagieren wollen, oder nicht. Dies kann beispielsweise durch die Batterie geschehen, die benötigt wird, um das RFID-Tag in Betrieb zu nehmen. Indem man diese Batterie nicht zusammen mit dem RFID-Tag ausgibt, sondern den Besucher dazu auffordert, sie sich bei Teilnahmewunsch geben zu lassen, ist die Freiwilligkeit der Teilnahme gewahrt. Ein Teilnahmezwang darf nicht entstehen.

Es muss den Benutzern möglich sein, sämtliche eigenen Kontaktdaten hinzuzufügen, zu ändern und vor allem zu löschen. Im Rahmen des zu entwickelnden Systems kann dies beispielsweise über eine Weboberfläche geschehen, zu der der Besucher sich verbinden und sich mit Benutzername und einem Kennwort einwählt, um anschließend seine Daten zu editieren. Damit wird dem Recht des Benutzers auf informationelle Selbstbestimmung Rechnung getragen.

4 Systementwurf

Das zu entwickelnde System wird modular aufgebaut, da es schnell und einfach an verschiedene Anforderungen einer Veranstaltung anpassbar sein soll. So kann für eine Veranstaltung die Information über laufende Kontakte allein ausreichen, auf einer anderen jedoch soll der Button auf den verwendeten RFID-Tags eine ganz besondere Rolle einnehmen.

In diesem Kapitel wird beschrieben, wie dieser Gedanke eines modularen Systems umgesetzt werden soll.
Zunächst wird dargelegt, welchen Teil der Realität das System abbilden soll. Anschließend werden die abzubildenden Daten strukturiert und in ein Modell gegossen.
Schließlich wird ein Protokoll entworfen, um die einzelnen Module des Systems miteinander kommunizieren zu lassen.

4.1 Architektur

Um verschiedensten Anforderungen gleichzeitig gerecht werden zu können und dabei nicht ständig alle Komponenten des Systems neu implementieren oder überarbeiten zu müssen, bietet es sich an, das System verteilt zu gestalten.

Es gibt dabei zwei wesentliche verschiedene Möglichkeiten, ein System zu verteilen. Die klassische Herangehensweise ist ein Client-Server-Modell. Der Server bietet hierbei seinen Clients Daten und Dienste an und verwaltet alle relevanten Informationen zentral.

Eine weitere Herangehensweise ist das Peer-to-Peer-Modell. Hierbei gibt es keinen zentralen Server, der alle relevanten Informationen bereitstellt, sondern verschiedene Rechner oder Prozesse, die jeweils unterschiedliche Informationen bereitstellen und direkt miteinander kommunizieren. Dabei stellen beide Kommunikationspartner sowohl Client als auch Server dar.

Für die Anwendung im Veranstaltungskontext wird das Client-Server-Modell umgesetzt, da die RFID-Tags ihre Nachrichten zwar an alle Empfänger in Reichweite schicken, diese aber wiederum alle Daten an einen bestimmten Rechner weiterleiten. Auf diesem Rechner läuft ein Serverprozess, der die Daten entgegennimmt und für die Clients aufbereitet.
Bei der Kommunikation zwischen Server und Client gibt es nun wieder zwei Herangehensweisen, wie der Server seine Dienste und Daten anbietet. Bei einem sogenannten Pull-Server fragen die Clients bei Bedarf Daten und Dienste ab, während ihnen bei einem Push-Server von diesem immer wieder Daten zugeschickt werden.
Da es bei dem zu entwickelnden System darauf ankommt, dass möglichst alle Clients auf demselben zeitlichen Informationsstand sind, wird auf das Push-Server-Modell zurückgegriffen, bei dem die Nachrichten vom Server an eine Liste von Clients in der Art und Weise verschickt werden, wie in Abbildung 4.1 ersichtlich.
So wird auch ein eventuell zusätzlicher Synchronisationsaufwand vermieden.

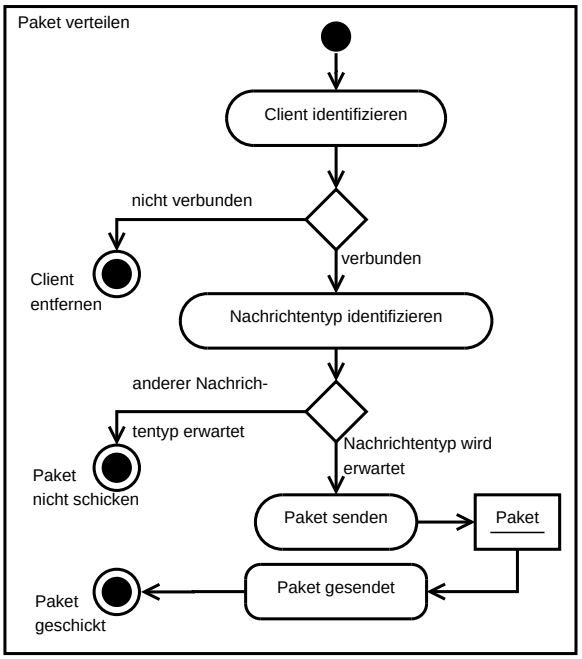


Abbildung 4.1: Aktivitätsdiagramm zum Verschicken von Nachrichten durch einen Push-Server

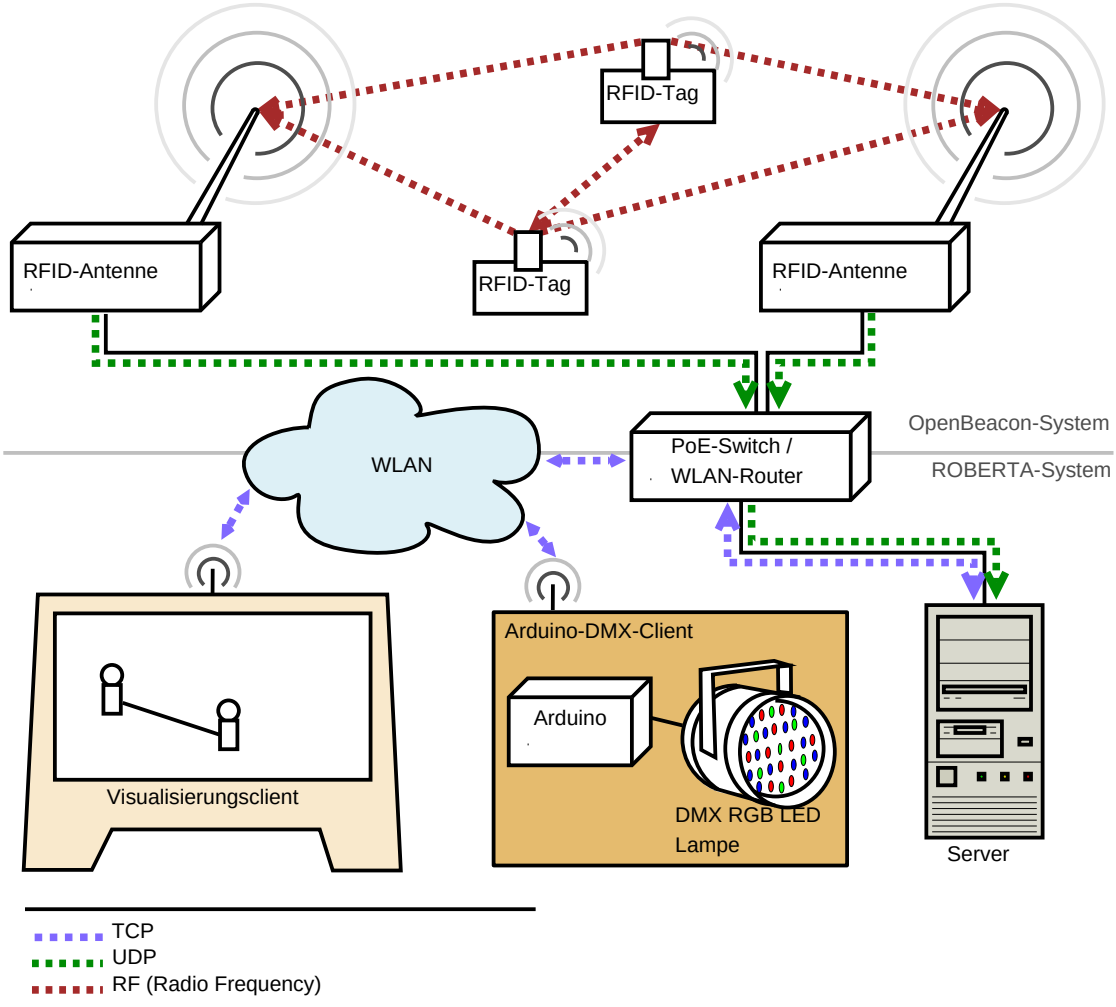


Abbildung 4.2: Aufbauskizze der Anwendung

In der Abbildung 4.2 ist das angestrebte Gesamtsystem detailliert skizziert. Es basiert auf einer klaren Trennung der OpenBeacon- von der ROBERTA-Infrastruktur - einzig das gemeinsam genutzte WLAN bzw. der gemeinsam genutzte Switch verbindet beide Systeme.
Die RFID-Tags kommunizieren untereinander und mit den RFID-Empfängern (RFID-Antennen).
Diese senden ihre Nachrichten per UDP weiter an den Server, zu dem hier exemplarisch zwei Clients per WLAN verbunden sind (ein Visualisierungsclient, wie in [l_Bre10] entwickelt und ein DMX-Client, wie er im Rahmen dieser Arbeit entwickelt wird). Der Server wandelt die Nachrichten vom OpenBeacon-System in Nachrichten des ROBERTA-Systems um. Die Kommunikation zwischen den Clients und dem Server soll über TCP ablaufen, um Flusskontrolle und Fehlertoleranz nicht auch hier noch weiter implementieren zu müssen. Es fällt weiterhin auf, dass die Kommunikation zwischen den Clients und dem Server bidirektional geplant ist, während sie zwischen dem OpenBeacon-System und dem ROBERTA-System unidirektional abläuft. Damit wird erreicht, dass Clients währen sie schon mit dem Server verbunden sind, ihre Wünsche an zu empfangenden Paketen ändern können.

Die Implementierung der einzelnen Komponenten des ROBERTA-Systems folgt darüber hinaus dem *Model-View-Controller*-Prinzip , um eine leichtere Wart- und Erweiterbarkeit der Komponenten selbst zu gewährleisten.

4.2 Funktionalitäten

Abbildung 4.3 zeigt den typischen Ablauf im System. Der RFID-Server bekommt dabei von einem Client eine Nachricht, welchen Nachrichtentyp dieser Client vom Server empfangen will. Der Server prüft diesen, trägt den Client in die Liste zu beliefernder Clients ein und akzeptiert den Client.

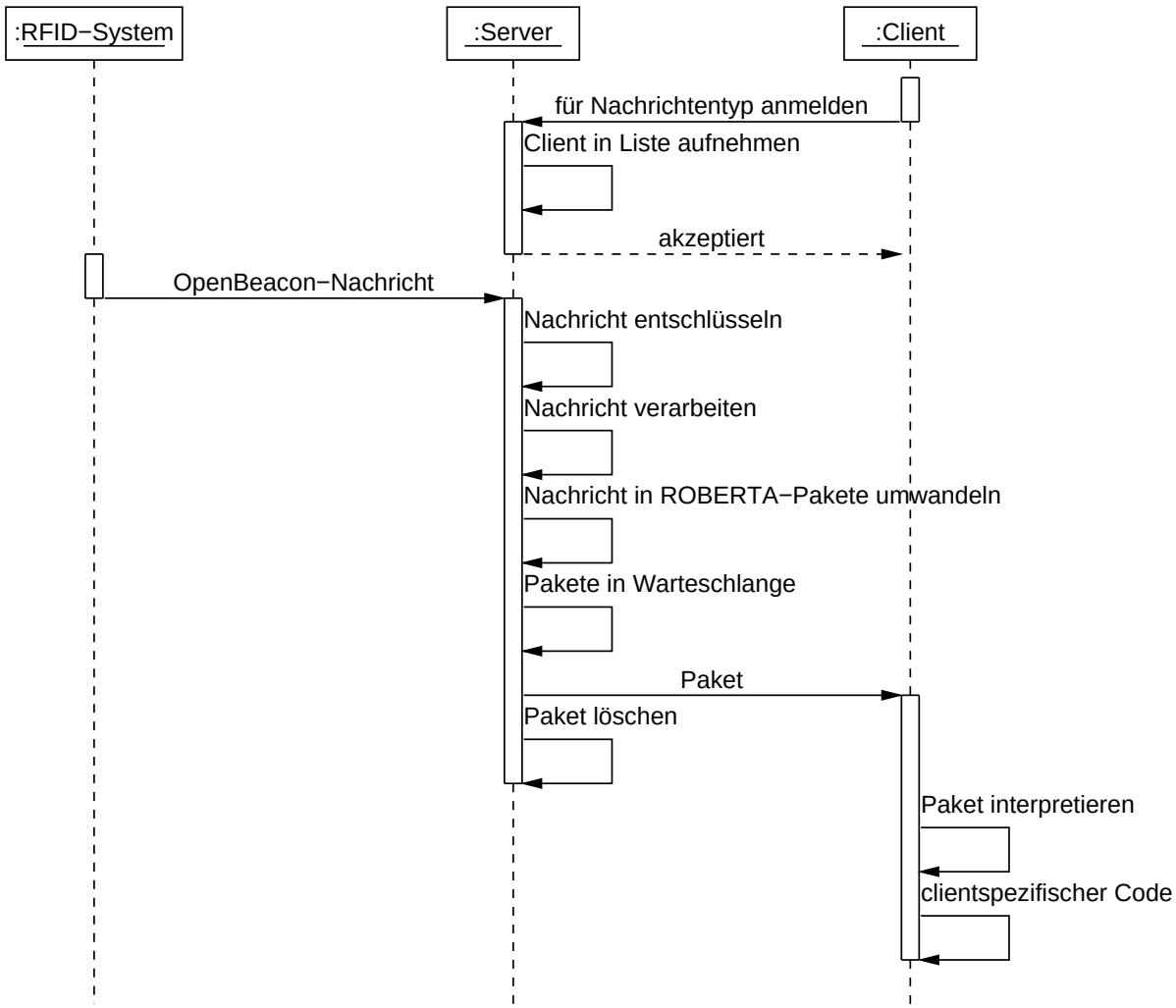


Abbildung 4.3: Sequenzdiagramm des wesentlichen Ablaufes im System

Daraufhin erhält der Server vom RFID-System immer wieder OpenBeacon-Nachrichten, die er entschlüsselt, verarbeitet und aus ihnen Nachrichten des ROBERTA-Protokolls erstellt. Diese werden in die Warteschlange zur Verteilung an die Clients gelegt, um anschließend nach der in Abbildung 4.1 abgebildeten Methode an die Clients zu versenden. Jeder Client verarbeitet nun die empfangenen Pakete und führt die entsprechenden clientspezifischen Funktionen aus aus. Im Server werden derweil Pakete, welche bereits an alle für den Nachrichtentyp des Pakets angemeldeten Clients gesendet wurden, gelöscht.

Das RFID-System steht im Sequenzdiagramm stellvertretend für die serielle Konsole, eine Logdatei, oder die RFID-Installation in einem Netzwerk.

Das Erkennen von Kontakten, ob Buttons gedrückt wurden und wo sich welche Person wann befindet, ist Teil der Verarbeitung der OpenBeacon-Nachricht im Server (*Nachricht verarbeiten* in Abbildung 4.3).

4.3 Datenmodellierung

Jede Softwareanwendung bildet immer nur einen Ausschnitt der Realität ab. In dieser Arbeit sind das die Kontakte zwischen Personen auf Veranstaltungen - genauer die Kontakte zwischen den RFID-Tags.
Dabei werden folgende Daten benötigt, die die RFID-Tags liefern.

TagID Die TagID ermöglicht eine 1:1-Zuordnung eines Tags zu einer Person.

Signalstärke Die Signalstärke wird zur Ortung und zur Erkennung der Nähe von Tags benötigt. Die Tags senden ihre Nachrichten nicht nur auf vier verschiedenen Signalstärken an die Empfänger, sondern erkennen auch untereinander in zwei Abstufungen, wie nah sie sich sind.

Sequenznummer Die Sequenznummer eines Paketes gibt an, an welcher Stelle im Nachrichtenstrom dieses Paket gesendet wurde. Somit kann festgestellt werden, ob Pakete zwischen zwei Nachrichten fehlen. Die Sequenznummer ist also bei der Flusskontrolle hilfreich.

Zusatzinformationen Eine Zusatzinformation, die die von uns verwendeten Tags senden ist zum Beispiel, ob der auf ihnen befindliche Button Abbildung 2.1 gerade gedrückt wurde.

Kontaktinformationen Die wichtigsten Informationen im Rahmen dieser Arbeit sind die Kontaktinformationen, die die Tags senden. Sie bestehen aus bis zu vier TagIDs und entsprechenden Signalstärken anderer Tags.

Mit diesen Daten berechnet der Server, wie lang Kontakte andauern, oder Buttons gedrückt werden. Diese Informationen bereitet er zu Nachrichten eines eigenen Protokolls auf, welches im Folgenden Abschnitt beschrieben wird.

Der Server selbst hat keine Verbindung zur Datenbank, die für die Webseite gefüllt wird, über die die Nutzer den Nachhaltigen Mehrwert der Anwendung erfahren.
Allein in dieser Datenbank besteht die oben angesprochene 1:1-Zuordnung eines Tags (anhand seiner TagID) zu einer Person. Dabei folgt die Datenbank dem Schema, welches (ebenso, wie die Webseite) von Hannes Breul entwickelt wurde und in Abbildung 4.4 dargestellt wird. Der SQL-Client hingegen, der die Datenbank mit den Kontaktdaten füllt, wurde vom Autor dieser Arbeit programmiert.

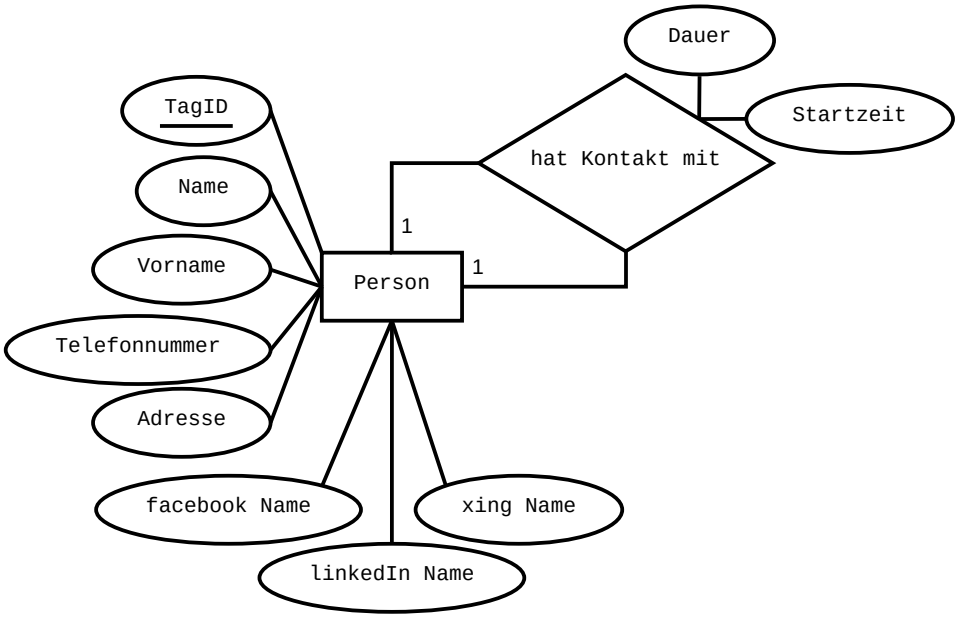


Abbildung 4.4: Entity-Relationship-Model der Datenbank für die Webseite

4.4 Protokollentwurf

OpenBeacon-Protokoll

Die Proximity-RFID-Tags senden ihre Daten strukturiert in Form eines Protokolls (fortan OpenBeacon-Protokoll genannt) an die Empfänger und diese wiederum an den Server. Pakete diesen Protokolls haben die Form, wie in Abbildung 4.5 ersichtlich.

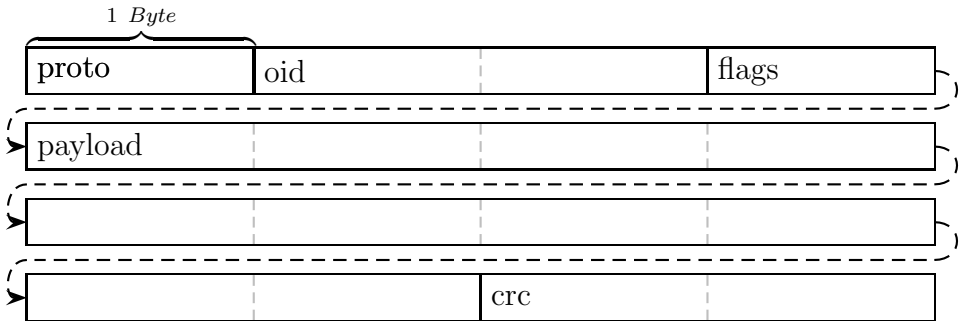


Abbildung 4.5: Paketform im OpenBeacon-Protokoll

In der vorliegenden Implementierungen kann `payload` zwei verschiedene Formen annehmen. Es gibt einmal die Trackernachrichten, die allein zur Ortung der Tags dienen und Auskunft darüber geben, wie oft das Tag schon „gestartet“ wurde - das heißt die Stromversorgung hergestellt wurde (`powerup_count`).
Diese Pakete sehen wie folgt aus:

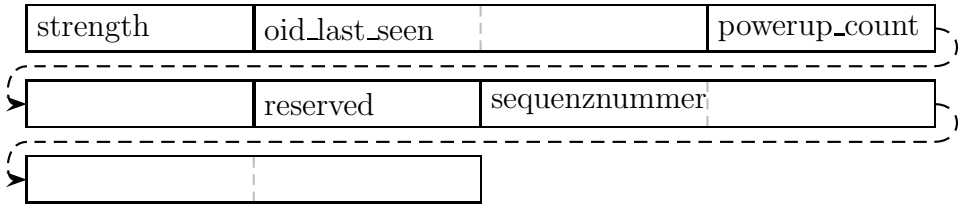


Abbildung 4.6: Aufteilung eines Trackerpakets

Darüber hinaus senden die Tags noch Proximityreporte. Diese Nachrichten enthalten bis zu vier IDs von Tags, die gerade von dem sendenden Tag „gesehen“ werden. Das Protokoll ist an dieser Stelle etwas unsauber, da nicht die vollständigen zwei Byte jeweils für eine TagID genutzt werden. Die letzten zwei Bit sind jeweils immer noch die Signalstärke der Tags untereinander. Dadurch wird jedoch erreicht, dass die Gesamtlänge der Nachricht 32 Byte bleibt und dennoch bis zu vier TagIDs mitgeschickt werden können, zu denen ein Kontakt besteht.

Diese Pakete sehen wie folgt aus:

Abbildung 4.7: Aufteilung eines Proximityreports

Der Server empfängt diese Daten und arbeitet sie nun neu auf.
Darüber hinaus bietet der Server die Möglichkeit, alle empfangenen Nachrichten zusammen mit einem Zeitstempel (in Form der Zeitdifferenz in Millisekunden zur letzten Nachricht in 64 Bit) in einer lokalen Datei zu speichern. Das ermöglicht ein späteres „Abspielen der Veranstaltung“ und somit der Rekonstruktion der Ereignisse.

Proximity-Tag-Network-Protocol

Der Server berechnet anhand dieser Daten beispielsweise, wann ein Kontakt besteht und endet. Er liest aus den Nachrichten des OpenBeacon-Systems, ob auf einem RFID-Tag der Button gedrückt wurde und errechnet gegebenenfalls die Dauer des Tastendrucks. Zukünftig soll der Server die RFID-Tags orten können und den Clients die Position der einzelnen Tags zur Verfügung stellen.
Diese Informationen sind dann nicht mehr mit dem bestehendem Datenprotokoll der Tags abzubilden, weshalb ein neues Protokoll für die Kommunikation zwischen dem Server und seinen Clients erdacht wurde.
Laut Anforderungen muss es möglich sein, verschiedene Granularitäten von Daten zu erzeugen - unterschiedliche Nachrichtentypen müssen gesendet werden.
Folgende Nachrichtentypen werden dabei vom Server erzeugt:

- **alle Antennen** – Nachricht mit allen AntennenIDs
- **Knopfnachrichten** – Nachrichten, wenn der Button auf den Tags gedrückt wurde
- **Trackernachricht** – Nachrichten über die empfangenen Signalstärken
- **abgeschlossener Kontakt** – Nachrichten, wenn Kontakte abgeschlossen wurden
- **laufender Kontakt** – Nachrichten über aktuell laufende Kontakte
- **Positionsnachrichten** – Nachrichten über die aktuelle Position des jeweiligen Tags

Sie werden durch das im Rahmen dieser Arbeit implementierte Proximity-Tag-Network-Protocol (PTNP) wie folgt definiert:

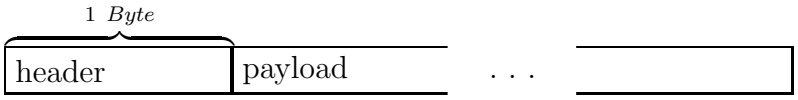


Abbildung 4.8: Struktur eines PTNP-Paketes

Im `header` steht jedes Bit für einen anderen Nachrichtentyp. Das hat zur Folge, dass nur maximal acht Nachrichtentypen existieren können und den Vorteil, dass jeder Client seine gewünschten Nachrichtentypen durch Verknüpfung der einzelnen Nachrichtentypen mit einer ODER-Operation zu seinem Nachrichtenbyte zusammenstellen kann, mit dem er sich dann beim Server meldet. Der `payload` sieht je nach Paketart unterschiedlich aus. Die Größe des Payloads hängt von der Art des Paketes ab und ist fest oder variabel (siehe *alle Antennen*).

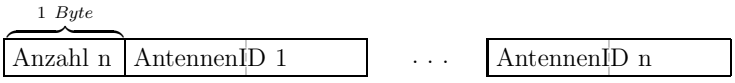


Abbildung 4.9: Struktur des Payloads mit allen AntennenIDs

Für die Antennen wird nach dem Header zunächst die Anzahl und dann hintereinander alle AntennenIDs in einer Auflösung von 16 Bit pro Antenne geschickt.

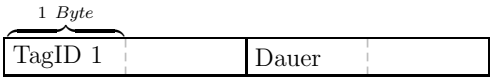


Abbildung 4.10: Struktur des Payloads für ein abgeschlossenes Button-Event

Die Nachrichten über das Abgeschlossene Button-Event (der Button wurde schon losgelassen) setzt sich aus der TagID und der Dauer des Button-Events in Sekunden zusammen.

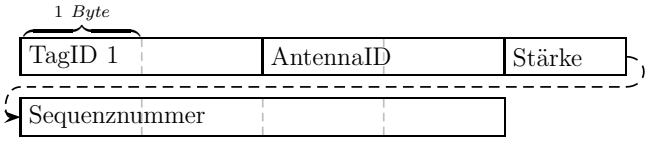


Abbildung 4.11: Struktur des alle Antennen-Payloads

Jede emfpangene Beacon-Message aus dem OpenBeacon-System wird in eine Tracker-Nachricht des ROBERTA-Systems überbführt. Aufgrund der besonderen Beschaffenheit des Proximity-Reports (er wird auf der vierten Signalstärke gesendet), wird für diesen ebenfalls eine Tracker-Nachricht erzeugt.

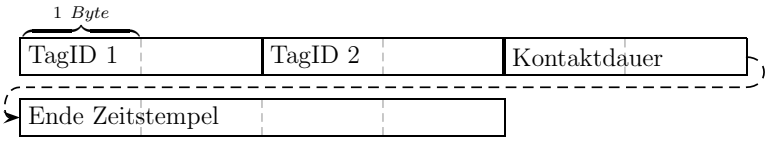


Abbildung 4.12: Struktur des Payloads von Paketen für abgeschlossene Kontakte

Die Pakete für abgeschlossene Kontakte setzen sich aus den beiden IDs der Tags zusammen, die den Kontakt hatten. Gefolgt wird das von der Dauer des Kontaktes in Sekunden

auf 16 Bit Breite und schließlich zur Eindeutigen Identifikation des Startpunktes wird der Endzeitstempel des Kontaktes mitgesendet. Anhand dessen kann zurückgerechnet werden, wann der Kontakt zustande kam.

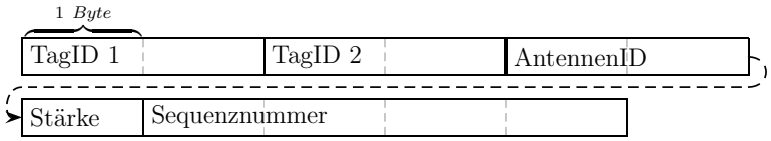


Abbildung 4.13: Struktur des Payloads für laufende Kontaktnachrichten

Pakete über laufende Kontaktnachrichten werden jedes mal generiert, wenn eines der beiden Tags einen Proximity-Report schickt. Die Stärke bezeichnet dabei die Signalstärke zwischen den beiden Kontakt habenden Tags und die Sequenznummer ist die des ersten Tags.

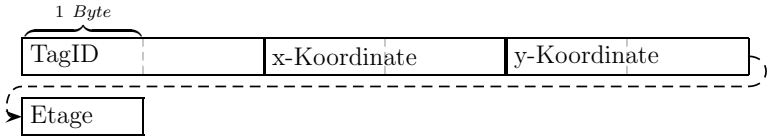


Abbildung 4.14: Struktur des Payloads für die Position des Tags auf dem Gelände

Pakete Art werden versendet, wenn sich die Position des Tags im Raum verändert. Dabei werden alle Längenangaben in Zentimetern übertragen (x- und y-Koordinate).

4.5 Benutzerschnittstelle

Im MVC-Modell ist neben der Datenhaltung und -verarbeitung noch ein sogenannter View definiert. Dieser bildet die Schnittstelle zum Benutzer. Alle Ein- und Ausgaben werden über diese Schnittstelle vorgenommen. Im entwickelten System gibt es pro Komponente unterschiedliche Benutzerschnittstellen:

Die Serveranwendung ist über Kommandozeile bedienbar, wobei sich diese Bedienung auf die Übergabeparameter beschränkt. Während der Laufzeit gibt die Serveranwendung Detailinformationen über empfangene Pakete, daraus generierte Nachrichten, sowie Clients, die sich an- und abmelden aus.
Hingegen die Clients bieten die unterschiedlichsten Benutzerschnittstellen:

- Der SQL-Client, der sämtliche Kontakte in die Datenbank der Webseite schreibt, hat ebenso wie der Server die Kommandozeile als einzige Benutzerschnittstelle, über die Informationen ausgegeben werden.

- Der im Rahmen dieser Arbeit entwickelte DMX-Client für Arduino bietet die gesteuerten DMX-Geräte als Ausgabemedien, sowie drei LEDs zum Visualisieren des momentanen Verbindungsstatus zum Server (eine grüne LED als Indikator für hergestellte Verbindungen, gelb für Verbindungsaufbau und rot für das Senden und Empfangen von Paketen über WLAN).

- Der Analyzer-Client hat eine grafische Benutzerschnittstelle, auf der alle Informationen dargestellt werden.

- Der Visualisierungsclient (siehe [l_Bre10]) verfügt über eine umfangreiche grafische Benutzeroberfläche.

5 Implementierung

In diesem Kapitel wird genauer dargelegt wie das System implementiert wird und worauf besonders geachtet wird. Zunächst wird die Programmierumgebung beschrieben, um anschließend die Umsetzung der Systemarchitektur näher zu beschreiben.

5.1 Programmierumgebung

Die geschickte Wahl der Programmierumgebung ist die wichtigste Voraussetzung für die erfolgreiche und effiziente Implementierung des ROBERTA-Systems. Sämtliche Firmware für die Tags und Reader des OpenBeacon-Projektes, sowie eine Software für PCs ist in der Programmiersprache C geschrieben. Da die Software für PC jedoch zu Beginn der Implementierung des ROBERTA-Projektes noch nicht ausgereift war und auch nicht hinreichend dokumentiert, wird die PC-Komponente für das ROBERTA-Projekt von Grund auf neu implementiert. Es können jedoch einige Algorithmen aus dem OpenBeacon-Projekt übernommen werden.
Ein wichtiges Kriterium bei der Wahl der Programmiersprache und -umgebung ist in diesem Projekt Plattformunabhängigkeit. Sowohl für Java als auch C++ ist dieses Kriterium nach dem objektorientiertem Programmierparadigma erfüllbar. Da ein großer Teil des Codes aus dem OpenBeacon-Projekt in der Programmiersprache C geschrieben wurde und C++ vollständig abwärtskompatibel zu C ist, wurden verstärkt Bibliotheken und Frameworks für C++ gesucht.
Mit Qt[1] gibt es ein mächtiges Framework für C++, mit dem auf vielen denkbaren Ziel-

[1] `http://qt.nokia.com/products/`

plattformen nativ lauffähige Anwendungen erzeugt werden können, die daher sehr performant sind. Somit kann bei der Implementierung Code aus dem OpenBeacon-Projekt nahezu 1:1 wiederverwendet werden kann (lediglich die Basisdatentypen müssen gegen das Qt-Äquivalent getauscht werden) - sämtliche Pointer-Arithmetik kann unangetastet bleiben (was mit Java beispielsweise nicht möglich wäre). Darüber hinaus gibt es mit dem Qt-Creator[2] eine dedizierte Entwicklungsumgebung für C++-Anwendungen, die Qt verwenden.

[2] http://qt.nokia.com/products/developer-tools/

5.2 Umsetzung der Systemarchitektur

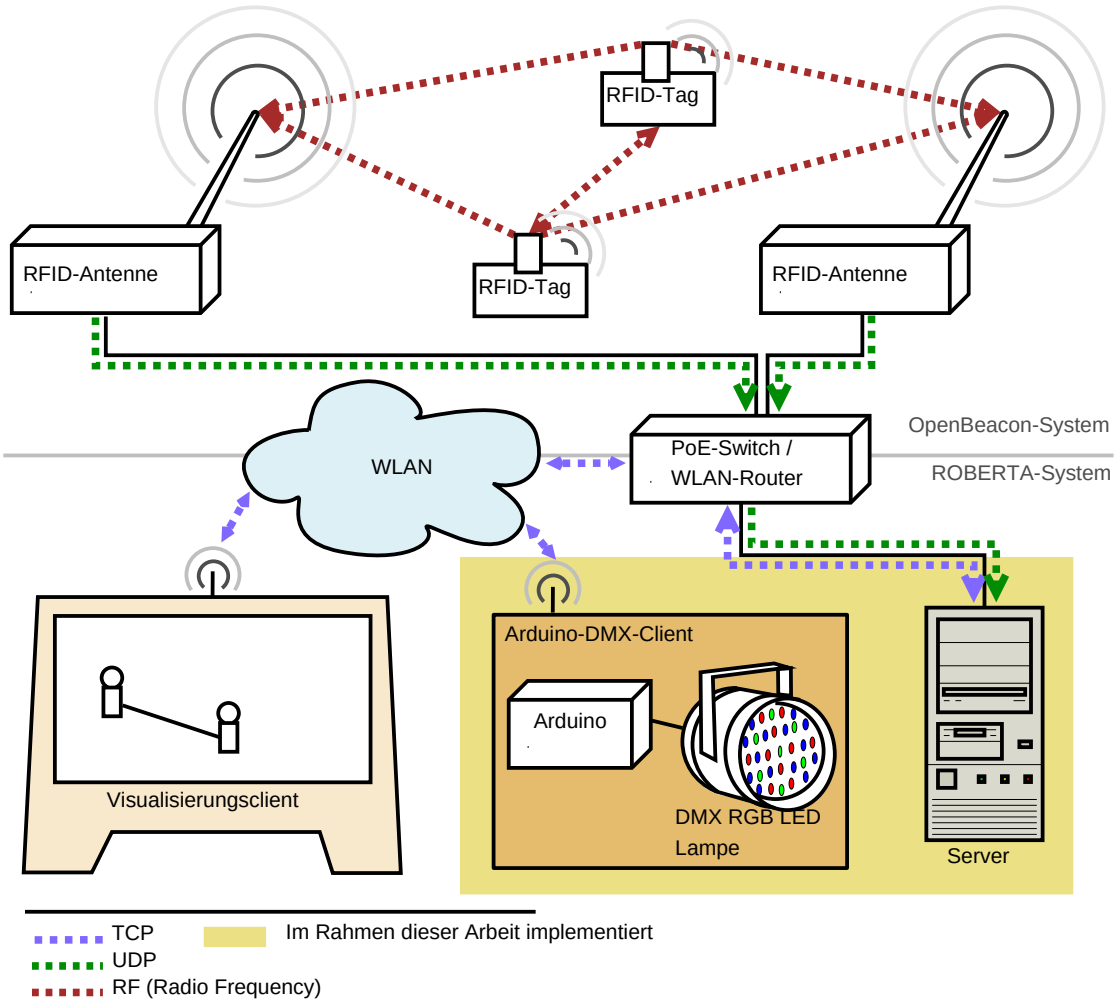


Abbildung 5.1: Aufbauskizze mit Hervorhebung des implementierten Teils

Im Rahmen dieser Arbeit wurden die Serveranwendung, sowie ein Client für die Arduino-Plattform umgesetzt. Zu der Serveranwendung gehört sowohl die Implementierung des ROBERTA-Protokolls, als auch die Implementierung einer Clientklasse, mit der das Programmieren neuer Clients für den Server erleichtert wird. Zusätzlich wurde die Serveranwendung so geschrieben, dass sie sowohl vom Netzwerk, als auch aus Logdateien und auch von der seriellen Schnittstelle Daten entgegen nehmen kann. Um diese Mög-

lichkeit zu nutzen, wurde eine Firmware für den OpenBeacon-USB-Node geschrieben, welche alle empfangenen Daten über die serielle Schnittstelle ausgibt.

5.3 Implementierung der Serverkomponente

In diesem Kapitel wird zunächst die Entwicklungsumgebung für die Implementierung näher betrachtet, um im Anschluss jeweils mit Codebeispielen auf die zentralen Aspekte bei der Umsetzung der einzelnen Komponenten und Funktionalitäten selbiger einzugehen.

5.3.1 plattformunabhängiges C++ mit Qt

Die meisten grafischen Toolkits verwenden Callback-Funktionen, um auf Ereignisse zum Beispiel von einer Benutzeroberfläche zu reagieren. Diese haben jedoch den Nachteil, dass sie nicht typensicher sind - *„man ist nie sicher, dass die gerade ausführende Funktion den Callback mit den richtigen Argumenten aufruft“* (Quelle: [l_Wol07, S. 27]). Des weiteren sind Callback und auszuführende Funktion fest verbunden, was zu weiteren Fehlern führen kann. Qt ist eine der größten Bibliotheken für C++, die anstatt von Callback-Funktionen ein Signal-Slot-Konzept umsetzt. Dazu erweitert Qt den Sprachumfang von C++ um die Schlüsselworte *signal, slot und emit* und führt den `moc` ein. Viele Objekte der Qt-Bibliothek emittieren Signale, die mit Slots (Methoden) verbunden werden können, welche beim Eintreffen des Signals ausgeführt werden. Darüber hinaus ist es möglich, Signale mit Signalen zu verbinden und sie damit weiter zu leiten. Gegenüber anderen C++-Bibliotheken, wie wxWidgets[3] oder auch gtkmm[4] ist Qt über die Zeit von einer Bibliothek für die GUI-Programmierung zu einem wesentlich größerem Framework herangewachsen, welches auch viele andere Dinge, wie Datenbankzugriffe, Netzwerkdatenaustausch und nicht zuletzt primitive Datentypen plattformunabhängig zur Verfügung stellt.

[3] `http://www.wxwidgets.org`
[4] `http://www.gtkmm.org`

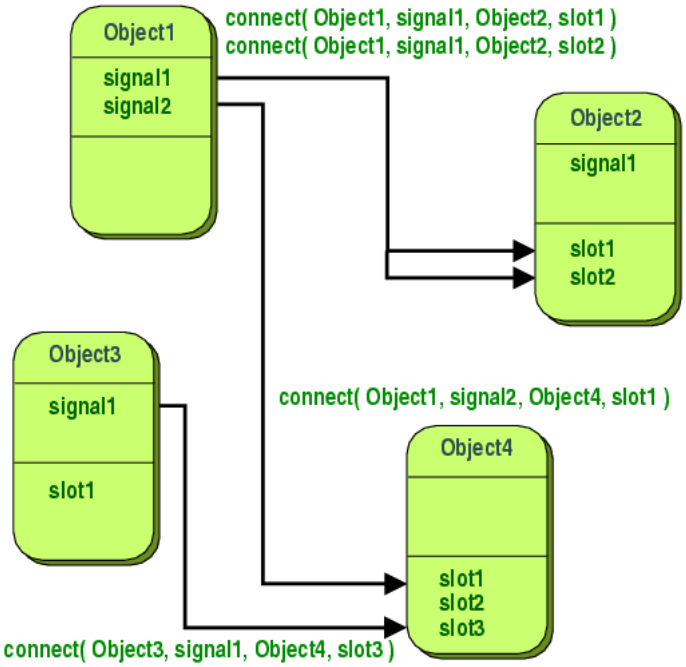


Abbildung 5.2: Signale und Slots in Qt [b_tro]

5.3.2 Abstraktionsebenen

Bei der Implementierung des Servers wurde auf einen modularen Aufbau geachtet - es gibt einen Datenempfänger, eine Datenhaltung und einen Datenverteiler.

Datenempfänger

Der Datenempfänger nimmt die OpenBeacon-Daten von verschiedenen Quellen entgegen:

- Netzwerk
- serielle Konsole
- Logdatei

Anschließend dekodiert und prüft er die empfangenen Daten und erzeugt Nachrichten in Form von Pakten des ROBERTA-Systems, wie sie in Abschnitt 4.3 beschrieben wurden. Diese werden in die Paketwarteschlange der Datenhaltung eingetragen und abschließend wird dem Datenverteiler signalisiert (Zeile 12 in Listing 5.1), dass neue Pakete zur Verteilung bereit stehen.

```
1  void ProximityDataReceiver::createTagToTagContactNewsPackage(PTNP::oid
       oid1, PTNP::oid oid2, PTNP::duration duration, PTNP::timestamp end) {
2      // Datensatz vorbereiten und einen Stream dafür
3      QByteArray *packageData = new QByteArray();
4      QDataStream packageDataStream(packageData, QIODevice::WriteOnly);
5
6      // alle Daten in den Datensatz schreiben
7      packageDataStream<<oid1;
8      packageDataStream<<oid2;
9      packageDataStream<<duration;
10     packageDataStream<<end;
11     _packageQueueMutex.lock();
12     // das Paket mit dem Datensatz an die Warteschlange anhängen
13     _queue.append(new PTNP::Package(PTNP::TAG_TO_TAG_CONTACT_NEWS,
           packageData));
14     _packageQueueMutex.unlock();
15     // das Signal emitieren, dass neue Daten vorhanden sind
16     emit receivedData();
17 }
```

Listing 5.1: Routine zur Erzeugung eines Paketes über einen laufenden Kontakt

Datenhaltung

Die Datenhaltung läuft direkt in der Serverklasse. Sie umfasst eine Warteschlange der Pakete, die gesendet werden sollen, eine Liste der verbundenen Clients und deren gewünschte Nachrichtentypen, und schließlich je einer Liste der Tags, Antennen und laufenden Kontakte.

Datenverteiler

Der Verteiler schickt alle über das Signal des Datenempfängers empfangenen Pakete an alle Clients der Liste, die sich für diesen Nachrichtentyp angemeldet haben.

```
1  // wenn Clients verbunden sind
2  if(!_clients.empty()) {
3      QTcpSocket *clientSocket;
4      ProximityServerClient* client;
5      QList<ProximityServerClient*>::iterator iter;
6      // ersten Client hernehmen
7      iter = _clients.begin();
8
9      // solang noch Clients in der Liste stehen
10     while (iter!=_clients.end()) {
11         clientSocket = (*iter)->serverSocket();
12         client = (*iter);
13    // wenn der Client verbunden ist
14         if(clientSocket->state() != QAbstractSocket::ClosingState &&
               clientSocket->state() != QAbstractSocket::UnconnectedState) {
15             // und der Client auf diese Nachrichten wartet
16             if(client->msgType() & paket->header()) {
17                 clientSocket->write(block); // Nachricht schicken
18             }
19         }
20         ++iter; // nächsten Client wählen
21     }
22 }
```

Listing 5.2: Routine zum Senden der Pakete an die Clients

Anschließend werden die Pakete verworfen.

Client

Für die komfortable Implementierung neuer Clients wird bereits im Zuge der Entwicklung der Serverkomponente eine passende Clientklasse entwickelt, die sämtliche Nachrichten des Servers empfängt und entsprechende Signale mit den Daten emittiert. Ein neuer Client verbindet diese Signale aus dieser Klasse mit Slots, die dann die clientspezifische Funktionalität bereitstellen. Listing 5.3 zeigt die Vorgehensweise bei der Implementierung eines Clients für Qt.

```
1  // Konstruktor des Analyzer-Clients
2  AnalyzerClient::AnalyzerClient()
3  {
4      QSettings settings;
5      // hier wird die Clientklasse ver-
6      // wendet. Alle Nachrichtentypen
7      // werden durch bitweises ODER verknüpft
8      _proximityClient = new ProximityClient(
9              settings.value("serverhost", "localhost").toString(),
10             settings.value("serverport", 3333).toUInt(),
11             PTNP::BEACON_MSG | PTNP::TAG_TO_TAG_CONTACT_NEWS | PTNP::
                   TAG_TO_TAG_CONTACT_RUNNING | PTNP::BUTTON_MSG);
12     // die einzelnen Signale der Nach-
13     // richtentypen werden mit eigenen
14     // Methoden verbunden
15     connect(_proximityClient, SIGNAL(beaconMsg(PTNP::beaconMsg)),
16             this, SLOT(handleBeaconMsg(PTNP::beaconMsg)));
17     connect(_proximityClient, SIGNAL(buttonMsg(PTNP::buttonMsg)),
18             this, SLOT(handleButtonMsg(PTNP::buttonMsg)));
19     connect(_proximityClient, SIGNAL(tagToTagContactNews(PTNP::
           tagToTagContactNews)),
20             this, SLOT(handleTagToTagContactNewsPackage(PTNP::
                   tagToTagContactNews)));
21     connect(_proximityClient, SIGNAL(tagToTagContactRunning(PTNP::
           tagToTagContactRunning)),
22             this, SLOT(handleTagToTagContactRunningPackage(PTNP::
                   tagToTagContactRunning)));
23 }
```

Listing 5.3: Beispielimplementierung zur Verwendung der Clientklasse

5.3.3 Kontakterkennung

Die Erkennung von Kontakten ist eine der wichtigsten Grundfunktionalitäten des Systems und wird im Server wie folgt implementiert. Vom OpenBeacon-System bekommt der Server immer wieder Nachrichten der Tags darüber, mit welchen anderen Tags sie Kontakt haben. Der Server pflegt nun eine Kontaktliste von Tagpaaren, die gerade Kontakt haben. Jedes dieser Kontaktpaare hat ein Ablaufintervall, nach dem es als beendet angesehen wird. Dieses Intervall ist am Anfang eines Kontaktes (also bei der ersten Nachricht des OpenBeacon-Systems, dass die zwei Tags Kontakt haben) sechs Sekunden. Die Tags senden alle zwei Sekunden, mit wem sie Kontakt haben und es kann sein, dass sie mit mehr als vier Tags gleichzeitig Kontakt haben. Dadurch kann es vorkommen, dass es länger als zwei Sekunden dauert, bis die Nachricht über den Kontakt wieder beim Server eintrifft und sechs Sekunden haben sich in Tests als sinnvolle Untergrenze für das Ablaufintervall erwiesen. Jedes mal, wenn ein Kontakt erneut registriert wird, wird das Intervall um eine Sekunde erhöht und neu gestartet (siehe Listing 5.5) - der Kontakt festigt sich.

```
1  // Benötigte Konstanten
2  #define MAX_CONTACT_TIMEOUT 20
3  #define MIN_CONTACT_TIMEOUT  6
4  #define TIMEOUT_UP_STEP 1
5
6  [...]
7
8  void OuttimingTagContact::seen(ProximityAntenna* const antenna, PTNP::
       signalStrength strength) {
9    // den momentanen timeout stoppen
10   this->stop();
11   // intervall hochzählen - Kontakt festigen
12   if(_timeout<MAX_CONTACT_TIMEOUT)
13       _timeout += TIMEOUT_UP_STEP;
14   _timestamp_LastView = QDateTime::currentDateTime().toTime_t();
```

```
15    _actualDuration = OuttimingObject::timeElapsedInSeconds();
16    this->setInterval(_timeout*MULTIPLIER);
17    // timeout neu starten
18    this->start();
19    [...]
20  }
21  [...]
```

Listing 5.4: Routine für das Halten eines Kontaktes

Wenn sich die Gesprächspartner nun für kurze Zeit anderen Dingen zuwenden und anschließend wieder zum Gespräch zurückkehren, wird das vom System nicht als neuer Kontakt, sondern als kurze Abwesenheit gewertet. Die maximale Intervalldauer beträgt 20 Sekunden. Das beutet, dass nach der letzten Nachricht über einen Kontakt weitere 20 Sekunden vergehen müssen, damit der Kontakt als abgeschlossen gilt und die Clients über diesen Kontakt in Kenntnis gesetzt werden. Erst dann wird dieser Kontakt aus der Kontaktliste des Servers entfernt. Gewertet wird bei der Länge eines Kontaktes allerdings die tatsächliche Länge zwischen erster und letzter Nachricht - ohne die 20 Sekunden Ablaufzeit.

5.4 Hardware und Hardwareentwicklung - ein Client

In diesem Abschnitt wird die Implementierung der Firmware für den OpenBeacon-USB-Node, sowie der Software für das Arduino-Board beschrieben.

5.4.1 USB-Node

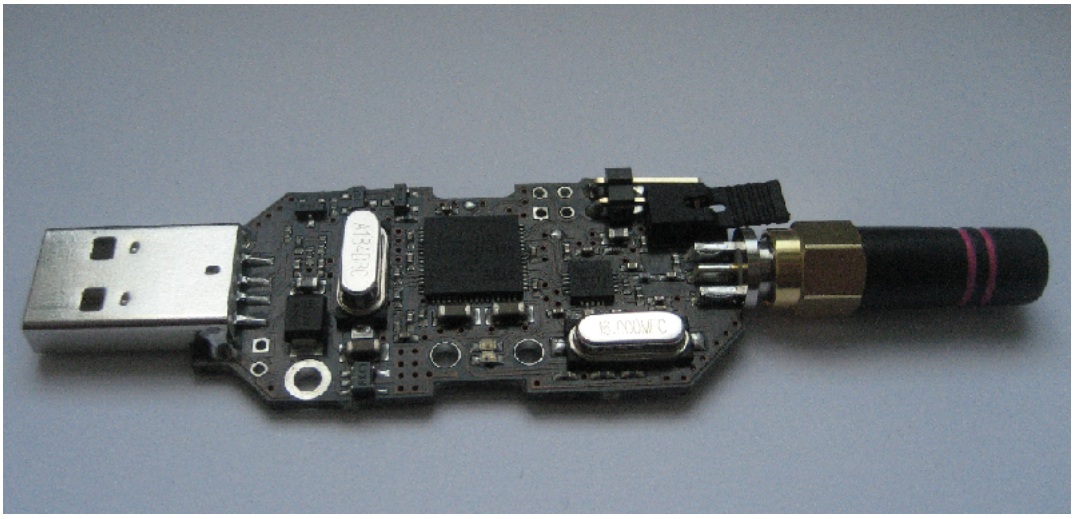


Abbildung 5.3: Der OpenBeacon USB-Node

Für den OpenBeacon-USB-Node wurde eine Firmware geschrieben, die auf einer bereits existierende Lösung aus dem Repository[5] basiert. Diese wurde soweit abgeändert, dass sie sämtliche Daten, die empfangen werden zusammen mit einer Absenderadresse auf die serielle Konsole schreibt.

Im Server gibt es dazu passend eine Komponente, die diese Nachrichten empfängt. Diese Komponente verwendet die SerialDevice-Bibliothek[6] für Qt.

Zum Kompilieren der Firmware und anschließendem Flashen[7] selbiger auf den OpenBeacon-USB-Node musste ein Cross-Compiling-Toolchain installiert werden.[8]

```
1 for (;;)
2   {
3     // wenn Daten vorhanden sind
4     if (nRFCMD_WaitRx (10))
5       {
```

[5] https://openbeacon.svn.sourceforge.net/svnroot/openbeacon/trunk/firmware/at91sam7/

[6] http://qt-apps.org/content/show.php/QSerialDevice?content=112039

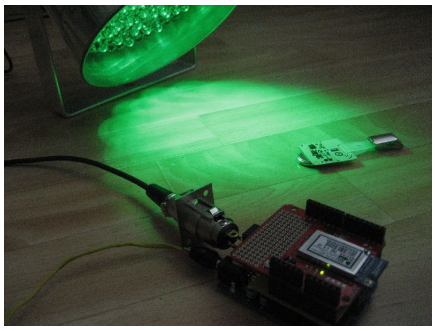
[7] Flashen bezeichnet in diesem Zusammenhang das Überspielen des Flash-Speichers vom OpenBeacon-USB-Node mit einer neuen Firmware.

[8] näheres zur Installation des Toolchains: http://wiki.openbeacon.org/wiki/OpenBeacon_USB

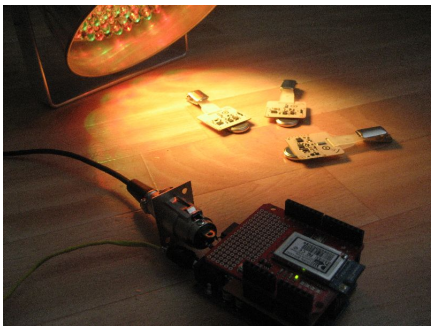
```
6         vLedSetRed (1); // Rote LED anschalten
7         do
8           {
9             // empfangene Daten vom Chip auslesen
10            nRFCMD_RegReadBuf (RD_RX_PLOAD, g_Beacon.byte,
11             sizeof (g_Beacon));
12            vLedSetGreen (0); // Grüne LED ausschalten
13            u_int8_t i=0;
14        // Absender schicken
15            vUSBSendByte(0x00);
16            vUSBSendByte(0xff);
17            vUSBSendByte(0x00);
18            vUSBSendByte(0xff);
19        // alle empfangenen Daten schicken
20            for(i=0; i<sizeof(g_Beacon); i++)
21            {
22              vUSBSendByte(g_Beacon.byte[i]);
23            }
24            vLedSetGreen (1); // Grüne LED anschalten
25          }
26        // solange die Warteschlange nicht leer ist
27        while ((nRFAPI_GetFifoStatus () & FIFO_RX_EMPTY) == 0);
28        vLedSetRed (0); // Rote LED ausschalten
29        }
30      nRFAPI_ClearIRQ (MASK_IRQ_FLAGS);
31    }
```

Listing 5.5: Routine zur Datenweiterleitung des USB-Nodes

In den Zeilen 12 bis 15 wird der Absender ausgegeben. Es handelt sich hierbei normalerweise um die IP des Senders (also in diesem Fall `0.255.0.255`). Nach der IP werden in den Zeilen 16 bis 19 sämtliche in Zeile 8 empfangenen Daten auf der seriellen Schnittstelle ausgegeben. Anschließend werden die LED's jeweils an- und ausgeschaltet, um dem Benutzer auch dann eine Rückmeldung zu geben, wenn auf der seriellen Schnittstelle nichts ankommt - so können Fehlerquellen schneller gefunden werden. Die gesamte in-



(a) Grün bei einem RFID-Tag



(b) Gelb bei zwei bis drei RFID-Tags

(c) Rot bei vier RFID-Tags

Abbildung 5.4: Der Arduinoclient in Aktion

nere Schleife (Zeile 6 bis 22) läuft solange, bis die Empfangswarteschlange leer ist (Zeile 22), um dann in der Endlosschleife immer wieder zu prüfen, ob neue Daten angekommen sind (Zeile 3).

5.4.2 Arduino

Das Arduino-Board wurde genutzt, um per WLAN Daten vom Server entgegenzunehmen, zu verarbeiten und dann per DMX eine Lampe zu steuern.

DMX-Shield und Aufbau

Da es für die Ansteuerung von DMX-Geräten noch kein fertiges Shield gab, wurde im Rahmen der Entwicklung ein eigenes Shield (basierend auf [w_Ardb]) mit der Software EAGLE[9] entworfen. Dieses wurde nach der Tonertransfermethode (beschrieben in

[9] `http://www.cadsoft.de/`

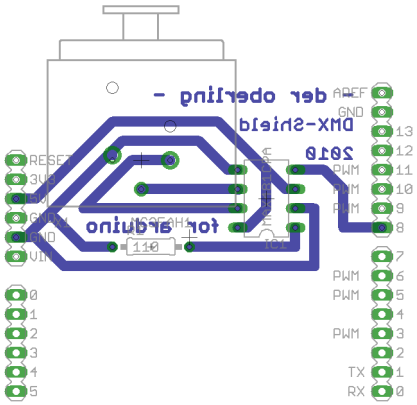


Abbildung 5.5: Platinenlayout des DMX-Shields

[w_Pfe10]) geätzt und anschließend wurden die Komponenten auf das Shield gelötet. Zusammen mit dem WiFly Shield[10](einem WLAN-Shield für Arduino) ergibt der in Abbildung 5.6 ersichtliche Schaltplan.

Funktionalität

Ein Programm für Arduino setzt sich immer aus einer `setup()`- und einer `loop()`-Routine zusammen.
In der `setup()`-Routine werden alle nötigen Variablen und Komponenten initialisiert. Anschließend läuft die `loop()`-Routine bis das Arduino-Board neu gestartet wird, oder die Stromzufuhr unterbrochen wird. (Die Möglichkeit der eventgesteuerten Programmierung wurde im Rahmen dieser Arbeit nicht genutzt.)

In der implementierten Anwendung wechselt eine DMX-RGB-LED Lampe je nachdem, wie viele Tags gerade empfangen werden, die Farbe von Grün (bei 0 Tags) bis zu Rot (bei einer wählbaren Anzahl von Tags bis zu 200).

Eine besondere Herausforderung bei der Programmierung des Arduinos war die Tatsache, dass nur 2 KB RAM zur Verfügung stehen. Davon belegen die Werte für den DMX-

[10] `http://www.sparkfun.com/commerce/product_info.php?products_id=9367`

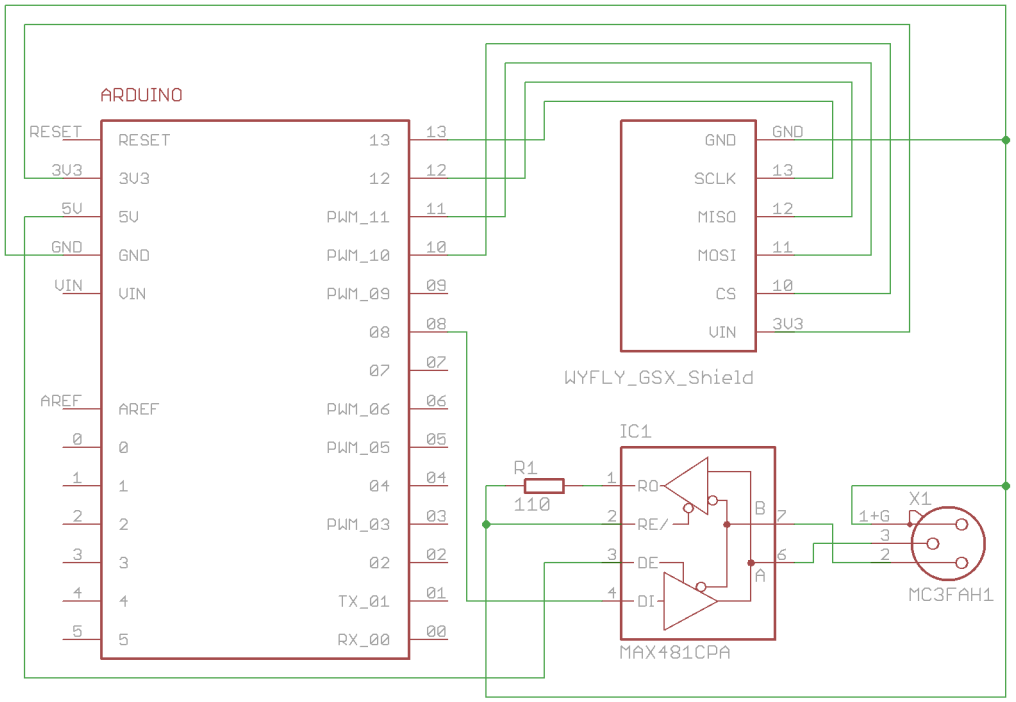


Abbildung 5.6: Schaltplan des Arduino-Clients

Controller bereits 512 Bytes (für jeden DMX-Kanal ein Byte) und schließlich noch $200 * 2Bytes = 400Bytes$ für die IDs der Tags. 912 von 2048 Bytes sind also schon allein mit den zur Laufzeit ständig notwendigen Variablen belegt. In den Rest müssen alle anderen Variablen des Systems. Da der WLAN-Controller per SPI-UART[11] über ASCII-Zeichenfolgen[12] angesprochen wird und diese alle in Variablen gespeichert sind, waren die 2048 Bytes schnell aufgebraucht.

Das Arduino-Board meldet sich beim Server als Client an, der sämtliche *BeaconMessages* bekommen möchte. Der Algorithmus, nach dem die empfangenen TagIDs behandelt werden, ist eine Art Keller-Algorithmus ohne Auslagerung [l_Tan03, S. 250 ff]. Dabei wird zunächst in den 200 reservierten Plätzen für TagIDs wird die gerade empfangene TagID gesucht. Anschließend wird die aktuelle TagID an die erste Position geschrieben, nachdem alle anderen TagIDs einen Platz nach hinten gerückt wurden, um den ersten

[11] UART: Universal Asynchronous Receiver Transmitter – ein Bauelement zur Realisierung einer seriellen Schnittstelle

[12] American Standard Code for Information Interchange

Platz frei zu machen. Im Unterschied zum Keller-Algorithmus wird jedoch der Platz, den die TagID vorher einnahm nicht durch eine andere gefüllt, sondern mit einer 0 überschrieben, die nun ebenfalls mit nach hinten rückt. Anschließend werden alle besetzten Plätze des Feldes gezählt. Das Ergebnis ist die aktuelle Zahl der Tags. Sollte ein Tag das Areal verlassen, rutscht es durch das ständige Empfangen von Nachrichten anderer Tags soweit nach hinten, bis es schließlich aus dem Feld entfernt wird - die Anzahl der Tags, die das Arduino nun zählt, hat sich um 1 verringert.

Der Algorithmus wurde dahingehend optimiert, das nicht immer alle 200 Plätze um einen nach rechts verschoben werden, sondern nur soviele, wie vorher Tags registriert waren. Notwendig ist diese Optimierung auch, weil es sonst zu lange dauern würde, bis eine TagID aus dem Feld entfernt wird - sie müsste dann immer erst alle 200 Plätze nach hinten rutschen, bevor sie gelöscht wird.

```
1  // shift Array
2  uint8_t shiftLimit = (tagCount + 10) % 255;
3  for(a=shiftLimit; a>0; a--) {
4    // TagID um einen Platz nach hinten schieben
5    // die letzte ID wird überschrieben
6    tag[a] = tag[a-1];
7  }
8
9  // input TagID
10 uint8_t tagPos = getTagPositionInArray(b.tag1);
11 // alte Position (wenn noch nicht vorhanden = 0)
12 // wird mit 0 überschrieben
13 tag[tagPos] = 0;
14 tag[0] = b.tag1;
15
16 // count tags
17 tagCount = 0;
18 for(a=0; a<MAXTAGS; a++) {
```

```
19    // wenn 0, dann ist die Position leer
20    if(tag[a]!=0) {
21      tagCount++;
22    }
23  }
```

Listing 5.6: Speicherung der TagIDs auf dem Arduino

In Listing 5.6 zeigt sich die Implementierung des Algorithmus. Das `shiftLimit` (siehe Zeile 2) wird benötigt, da nicht davon auszugehen ist, dass die Tags in immer derselben Reihenfolgen ihre Nachrichten senden. So kann es sein, dass ein oder mehrere Tags bereits zweimal eine Nachricht sandten, bevor ein anderes seine zweite schickte. Bei strikter Befolgung des oben beschrieben Algorithmus müsste diese TagID dann schon aus der Datenstruktur entfernt worden sein sein. Diese (hier zehn) Plätze sind also eine Art Auslagerungsspeicher (siehe Keller-Algorithmus), da die TagIDs, die sich hier befinden, zwar etwas länger nicht mehr gesichtet wurden, dies allerdings noch nicht bedeutet, dass sie wahrscheinlich in der nächsten Zeit auch nicht mehr gesichtet werden.

DMX-Implementierung

Für das DMX-Shield wurde im Programmcode des Arduino ein Bytefeld mit 512 Bytes definiert, in dem jedes Byte ein Wert für einen DMX-Kanal ist. Alle 512 Bytes werden pro loop()-Durchlauf einmal auf dem DMX-Kanal ausgegeben.
Die Pausen zwischen den einzelnen Bytes und schließlich dem gesamten Zyklus werden hart mit Assembler-Codes (`nop`) realisiert, wodurch ein akkurates Timing möglich ist.
Der Farbwechsel wird realisiert, wie in Listing 5.7 ersichtlich und in Abbildung 5.7 veranschaulicht.

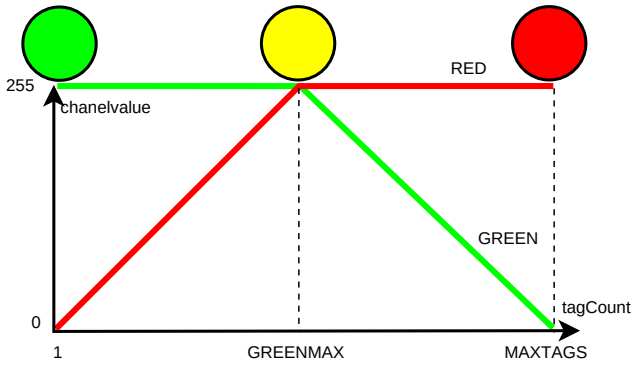


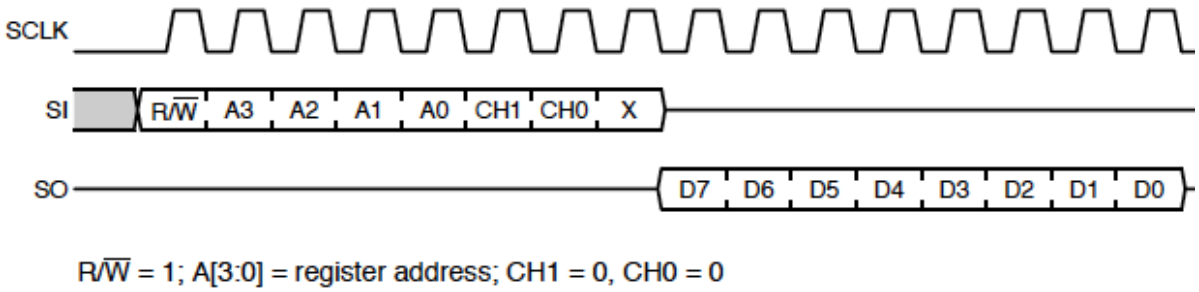
Abbildung 5.7: Verlauf der Farbfunktionen für die DMX-RGB-LED-Lampe

```
1  // gibt an, wie viele Tags die Lampe noch
2  // auf voller Intensität GRÜN leuchten lassen
3  // (bei tagCount = 40 leuchtet die Lampe gelb!)
4  #define GREENMAX  100
5  #define MAXTAGS       200
6  [...]
7
8  if(tagCount>GREENMAX) {
9    // bei mehr als GREENMAX Tags in Reichweite, nimmt der
10   // Grünanteil langsam ab - aus Gelb wird Rot
11   channelvalue[GREEN] = map(tagCount,GREENMAX+1,MAXTAGS, 255, 0);
12   channelvalue[RED] = 255;
13 } else {
14   // der Grünanteil ist maximal und Rot wird anteilig
15   // hinzugemischt - Gelb entsteht
16   channelvalue[GREEN] = 255;
17   channelvalue[RED] = map(tagCount, 1, GREENMAX, 0, 255);
18 }
19 [...]
```

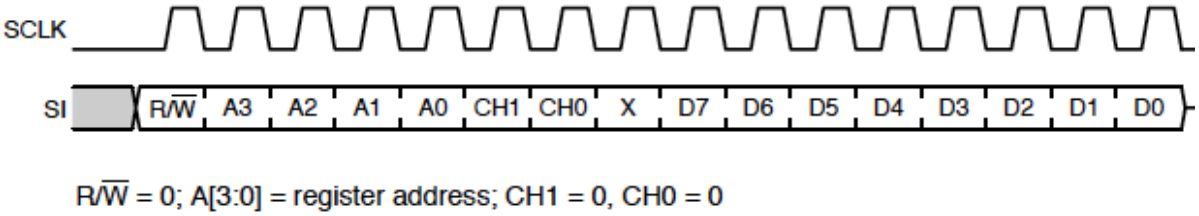
Listing 5.7: Farbwechsel in Abhängigkeit von der Anzahl an RFID-Tags

Kommunikation per SPI

Das WiFly-Shield enthält neben dem *RN-131G*[13] WLAN-Modul von Roving Network, welches per serieller Konsole bedient wird, den *SC16IS750 I2C/SPI-to-UART IC*[14]. Dieser Chip wird hier zur Kommunikation mit dem RN-131G WLAN-Modul über SPI eingesetzt, da das RN-131G noch über kein SPI verfügt (siehe [l_RN09, S. 3]). Sämtliche Nachrichten, die per WLAN empfangen oder gesendet werden, werden durch den SC16IS750 behandelt.



(a) Lesen aus Registern



(b) Schreiben in Register

Abbildung 5.8: SPI-Transfermethoden für den *SC16IS750*

Die Ansteuerung des SC16IS750 funktioniert dabei, wie aus Abbildung 5.8 ersichtlich: Zunächst wird der Chip über ChipSelect angewählt. Anschließend wird ein Byte mit Steuerungsinformationen übertragen. Die relevanten Bits sind hier $R/\overline{W}$, sowie $A[3:0]$. Über $R/\overline{W}$ wird festgelegt, ob in das Register, welches durch $A[3:0]$ referenziert wird, geschrieben oder aus ihm gelesen werden soll. Abhängig davon erfolgt beim Lesen die weitere Kommunikation über die (MI)SO- und beim Schreiben über die (MO)SI-Leitung.

[13]Datenblatt: `http://www.sparkfun.com/datasheets/Wireless/WiFi/rn-131G-ds.pdf`
[14]Datenblatt: `http://www.sparkfun.com/datasheets/Components/SMD/sc16is750.pdf`

```
1  namespace wifly{
2    [...]
3    void select(void) {
4      // ChannelSelect auf LOW
5      digitalWrite(CS,LOW);
6    }
7
8    // lese- / schreibroutine
9    char spiTransfer(volatile char data) {
10     SPDR = data; // Daten in SPI-DatenRegister schreiben
11     while (!(SPSR & (1<<SPIF))) {
12       // warten, bis SPIF-Flag im SPI-StatusRegister gesetzt ist
13     };
14     return SPDR; // den Inhalt des Datenregisters zurückgeben (bei Lesen
           wird das beschrieben)
15   }
16
17   void deselect(void) {
18     // ChannelSelect auf HIGH
19     digitalWrite(CS,HIGH);
20   }
21   [...]
22 }
23
24 [...]
25 wifly::select();
26 wifly::spiTransfer(THR); // in Schreibmodus gehen
27 wifly::spiTransfer(myMsgType); // Nachricht senden
28 wifly::deselect();
29 [...]
```

Listing 5.8: Routine zum Verschicken von Daten über SPI

In Listing 5.8 wird der Ausschnitt des Codes vom Arduino gezeigt, der den gewünschten Nachrichtentyp an den Server sendet. Die Adresse (`THR` - siehe Zeile 26) ist dabei das *„Transmit Holding Register“*, welches ein 64-Byte FIFO auf dem SC16IS750 ist (vergleiche: [l_NXP08]).

6 Test und Auswertung

Im Rahmen zweier Tagungen (*Medienproduktion 2010*[1] und *Kultur und Informatik 2010*[2]) wurde das System bereits getestet und die Daten stehen zur Auswertung zur Verfügung. In diesem Kapitel wird beschrieben, in welcher Umgebung getestet wurde, wer teilnahm und was besondere Herausforderungen waren.

6.1 Testkriterien

Das wichtigste Testkriterium bei einem System, dass die Benutzer nur relativ kurz nutzen, ist die intuitive Nutzbarkeit. Wenn das System nicht in wenigen, kurzen Sätzen erklärbar ist, verliert es schnell seinen Reiz für potentielle Benutzer. Getestet werden soll ebenso die Akzeptanz der Besucher gegenüber dem System sowie die Stabilität der Software. Für die Akzeptanz ist es von essenzieller Bedeutung (siehe Abschnitt 3.6), dass teilnehmende Gäste jederzeit die Möglichkeit haben, ihre personenbezogenen Daten selbst zu pflegen.

Die Testläufe sollen die Grundlage für weitere Entwicklungen der Software und Hardwarekopplung an die Software bilden. Es geht also darum, herauszufinden, wo die Grenzen des Systems und Anknüpfungspunkte für weitere Verbesserungen sind und was noch überarbeitet werden muss.

[1] `http://www.medienproduktion.org`
[2] `http://inka.htw-berlin.de/kui/`

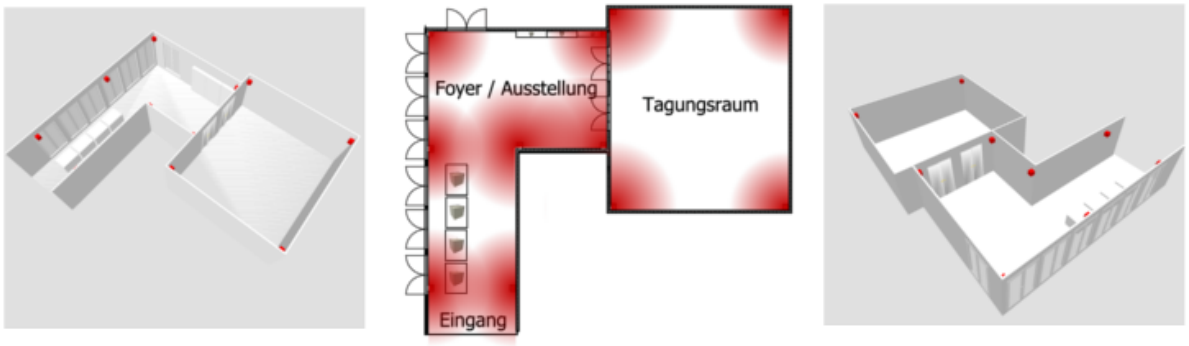


Abbildung 6.1: Aufteilung der OpenBeacon WLAN-Stationen bei der `Kultur und Informatik` (nicht maßstabsgetreu)

6.2 Demonstration der Funktionalität

In diesem Abschnitt wird der technische Aufbau des Systems auf den Tagungen beschrieben sowie die Durchführung der Feldtests. Dabei wird vor allem auf das Zusammenspiel der Komponenten und den gewünschten Mehrwert für den Tagungsgast eingegangen.

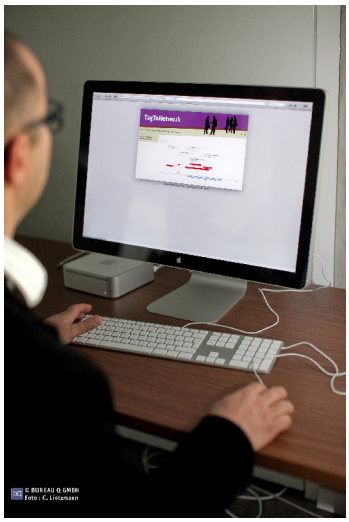
6.2.1 Aufbau

Auf den Tagungsgeländen wurden im Vorfeld OpenBeacon-WLAN-Stationen an strategisch günstigen Plätzen (meist Ecken) verteilt. Raumecken sind günstig, da sie das Signal der Antenne gleichmäßig in den Raum hinein leiten (siehe Abbildung 6.1). Diese meldeten sich bei einem WLAN-Accesspoint an und schickten alle Nachrichten weiter an den ebenfalls zu diesem Accesspoint verbundenem Serverrechner. Auch der SQL-Client, der sämtliche Kontaktinformationen aus den verarbeiteten Daten des Servers in die Datenbank schrieb, lief auf diesem Rechner. Darüber hinaus wurden auf beiden Tagungen ein Touchscreen und ein weiterer Rechner für die Webseitenpräsentation aufgestellt. Auf dem Touchscreen lief der in [l_Bre10] entwickelte Visualisierungsclient.

(a) RFID-Tagausgabe

(b) Batterieausgabe

(c) Namensschild



(d) Webseitenpräsentation

Abbildung 6.2: Anlaufstellen auf der `Medienproduktion 2010`

6.2.2 Durchführung

Jeder Tagungsgast hatte die Möglichkeit, sich eine Batterie geben zu lassen und somit am Versuch teilzunehmen - ohne Batterie (welche von vornherein nicht im RFID-Tag war) nahm der Besucher nicht teil. Dies war Teil des Datenschutzkonzeptes.
Während der Tagungen bewegten sich die Teilnehmer des Feldtests mit den RFID-Tags auf dem Tagungsgelände und wurden vom System erfasst. Schon während der Tagung konnten sich die Gäste über den Visualisierungsclient ansehen, mit wem sie bereits Kontakt hatten und wer gerade Kontakt mit wem hat bzw. überhaupt auf der Tagung ist. Über eine Webseite waren alle Kontaktinformationen der Personen einsehbar, mit denen man vis-à-vis Kontakt hatte. Diese wurde von Hannes Breul entwickelt, während die

Datenbank von einem Client gefüllt wird, der vom Autor dieser Arbeit implementiert wurde.

Abbildung 6.3: Der Login-Screen der Webseite

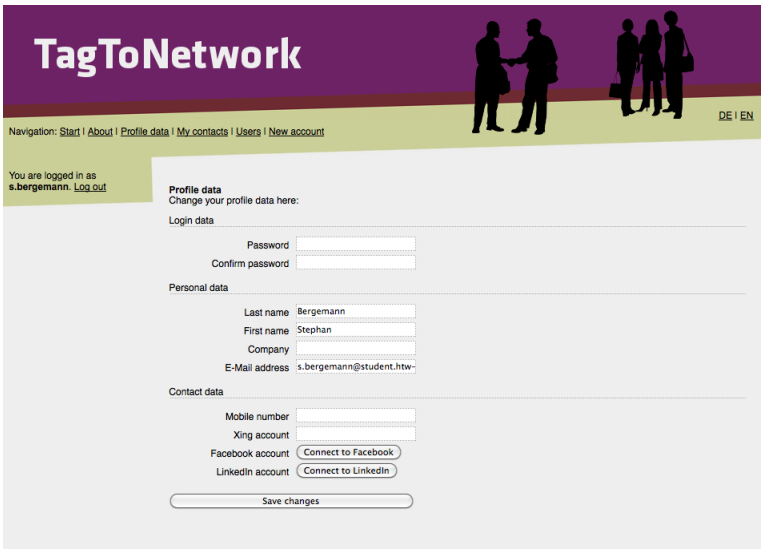


Abbildung 6.4: Das eigene Profil kann vollständig editiert werden

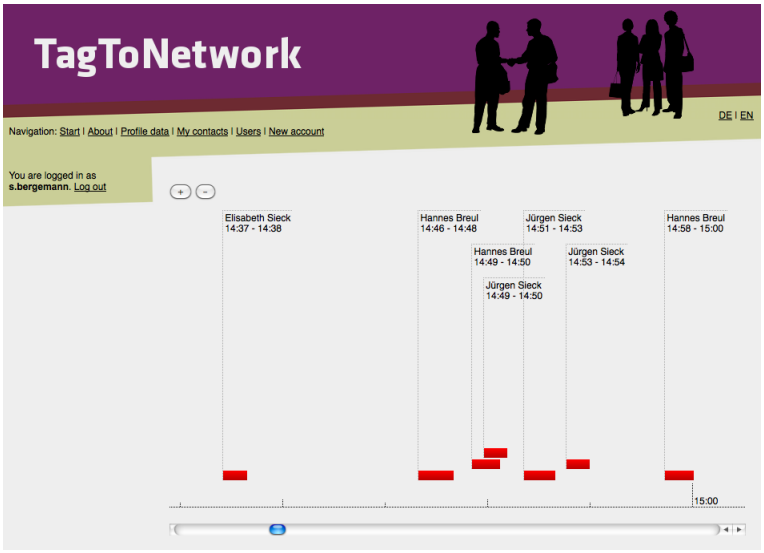


Abbildung 6.5: Die gesammelten Kontakte werden in einer Zeitleistenansicht zugänglich gemacht

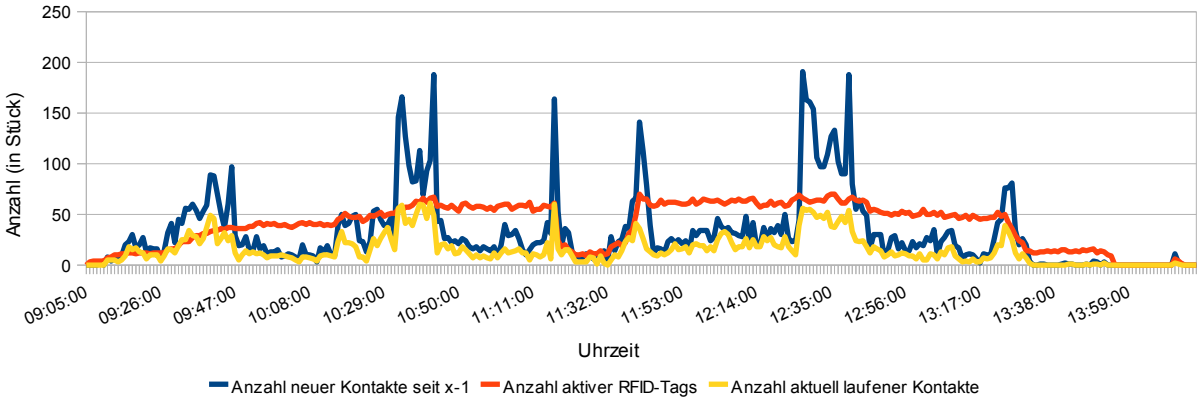
Alle Nachrichten der Tags wurden für eine adäquate Auswertung geloggt und können jederzeit wieder „abgespielt“ werden. Dabei wurden sämtliche Rohdaten der RFID-Tags, wie der Server sie empfing, binär mit der Zeitdifferenz zum vorherigen Signal abgespeichert.

6.3 Auswertung

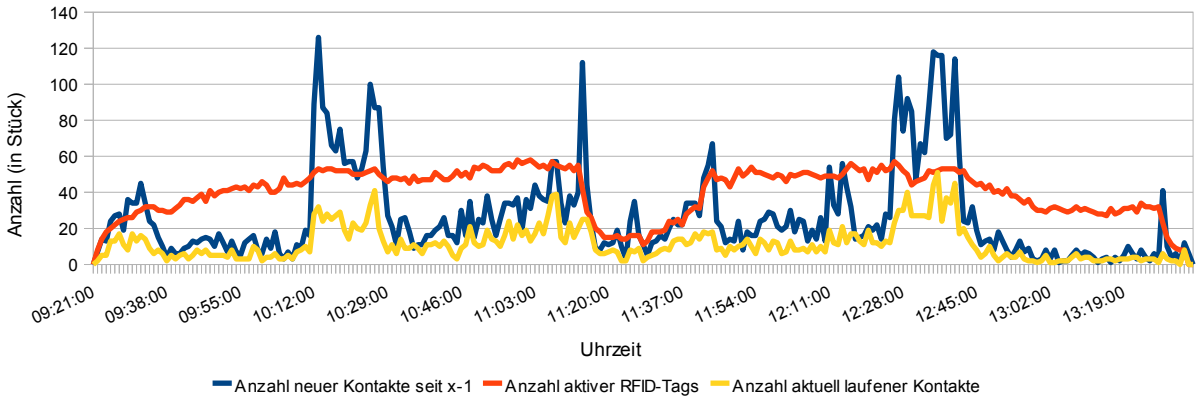
Zur Auswertung der gewonnenen Daten wurde ein separater Client für das System implementiert, der alle Nachrichten über laufende Kontakte sowie die Nachrichten in den verschiedenen Signalstärken der Tags abwartet (siehe auch Listing 5.3). Aus diesen Nachrichten berechnet der Client die Anzahl aktiver Tags sowie laufender Kontakte. Pro Minuten wird in eine csv-Datei[3] ein Eintrag vorgenommen, wie viele neue Kontakte es in dem Intervall gab, wie viele RFID-Tags aktiv sind und wie viele Kontakte gerade laufen. Die geloggten Daten ermöglichen eine grafische Aufarbeitung der Informationen, wie in Abbildung 6.6 ersichtlich.

[3]csv: comma separated value oder auch character separated value

Dabei ist *neue Kontakte seit x-1* die Anzahl von neu geknüpften Kontakten in der letzten Minute, während *aktuelle Kontakte* die Anzahl der Kontakte jeweils zum Zeitpunkt des log-Eintrags repräsentiert. Den Verlauf der beiden Kontakt-Graphen vergleichend lässt sich erkennen, dass es zwischen zwei Messungen oft eine ganze Menge kurzer Kontakte gibt.



(a) Kultur und Informatik Tag 1



(b) Kultur und Informatik Tag 2

Abbildung 6.6: grafische Auswertung der geloggten Nachrichten auf der Tagung *Kultur und Informatik 2010*

Es ist in den beiden Grafiken auch zu beobachten, dass die Anzahl der Kontakte zwischen den einzelnen Sessions immer wieder plötzlich stark steigt. Die Gäste befanden sich zu dieser Zeit überwiegend im Foyer und unterhielten sich. Das zeigt also, dass das System funktionierte und hier die Kontakte der Gespräche erkannte.
Darüber hinaus wird ersichtlich, dass in diesen Pausen die Anzahl der Tags leicht steigt,

was wohl dem Umstand zu verdanken ist, dass durch das Gespräch mit anderen Teilnehmern ein Interesse auch bei denjenigen geweckt wurde, die anfangs Skepsis hatten. Die maximale Anzahl von 70 gleichzeitigen teilnehmenden Besuchern wurde am 1. Tag der Tagung um 12:30 Uhr (kurz nach der Mittagspause) erreicht. Zur Mittagszeit ist an beiden Tagen ein massiver Einbruch der Anzahl aktiver Tags zu verzeichnen, was daran liegt, dass im Restaurant keine Antennen angebracht waren - die Gäste befanden sich somit außerhalb des abgedecktem Geländes.

Es lässt sich zusammenfassen, dass das System sich auf beiden Tagungen bewährt hat. Es wurde von den Besuchern (wenn auch meist mit Verweis auf den universitären Kontext der Veranstaltungen) angenommen.

7 Zusammenfassung und Ausblick

In diesem Kapitel werden die im Rahmen dieser Bachelorarbeit geschaffenen Lösungen zusammengefasst und deren Vor- und Nachteile erläutert. Schließlich werden Potentiale und Erweiterungsmöglichkeiten des Systems aufgezeigt.

7.1 geschaffene Lösung

Ziel der Arbeit war es, ein System zu kreieren, welches Besucherinteraktion auf Veranstaltungen nutzt, um dem Besucher Mehrwert und dem Veranstalter neue Gestaltungsmöglichkeiten zu geben. Dabei war es notwendig, das System modular aufzubauen, damit es einfach anpassbar und erweiterbar ist. Durch die verteilte Architektur wird genau diese Modularität erreicht. Es gibt einen Server, der für viele Anwendungsfälle die richtigen Nachrichtentypen zur Verfügung stellt. Für jede Anwendung wird nun ein Client geschrieben, der sich aus dem Repertoire des Servers die Nachrichten schicken lässt, die für seinen Zweck notwendig sind. Durch die Struktur des Quellcodes ist die Implementierung eines neuen Clients sehr einfach - eine entsprechende Client-Klasse, die sämtlichen Netzwerkverkehr abstrahiert existiert bereits und muss nur eingebunden werden. Diese bildet die Schnittstelle zum Server.
Der Server kann durch die Fähigkeit, Nachrichten von verschiedenen Quellen entgegennehmen zu können, auch zur Promotion des Systems bei potentiellen Kunden des Veranstalters eingesetzt werden. Zu diesem Zweck wird auf einem mobilen Rechner mit USB-Schnittstelle die Serveranwendung ausgeführt. Darüber hinaus ist der OpenBeacon-USB-Node an dem Rechner angeschlossen, der den Server mit den empfangenen Nach-

richten der RFID-Tags versorgt. Einzelne Clients können nun ebenso gestartet werden, wie im Produktivsystem mit den WLAN-Stationen. Dadurch ist ein elegantes Vorführen der prinzipiellen Funktionen des Systems ohne großen Aufwand (WLAN-Router und WLAN-Stationen installieren) möglich und gleichzeitig ein mobiles Präsentationssystem geschaffen. Diese Lösung wurde zwar ursprünglich für die Kundenakquise des BureauQ entworfen, eignet sich aber auch für diverse andere Einsatzgebiete (beispielsweise einer Anmeldungssoftware für die Besucher der Veranstaltung).

Gespeicherte Abläufe von Veranstaltungen können im Nachhinein abgespielt werden und bieten so dem Veranstalter eine detaillierte Auswertungsmöglichkeit. Er kann mit den entsprechenden Clients visualisieren, wann sich wo Gruppen gebildet haben und daraus Rückschlüsse darauf ziehen, welche Räumlichkeiten, Vorträge oder ähnliches sehr interessant waren und welche nicht (im musealen Kontext die Besucherforschung).
Es wurde darüber hinaus in dieser Arbeit ein Client für die Hardwareplattform Arduino geschaffen, der drahtlos (per WLAN) Daten vom Server empfängt, verarbeitet und schließlich per DMX Veranstaltungstechnik steuert. Somit ist eine einfache Integration des geschaffenen Systems in vorhandene Veranstaltungsinfrastruktur möglich.

7.2 Potentiale und Erweiterungen

Das System der verschiedenen Nachrichtentypen erzeugt eine breite Auswahlmöglichkeit an Kombinationen aus Nachrichten, die ein Client verarbeiten kann. Der Server kann dabei um weitere Nachrichtentypen erweitert werden, ohne dass alte Clients deswegen neu programmiert werden müssen. Die Abwärtskompatibilität muss einzig in der Clientklasse hergestellt werden, die die Schnittstelle zum Server bildet.

Das Gesamtsystem lässt sich einfach durch entsprechende Clients aus dem Veranstaltungskontext lösen und für andere Szenarien verwenden (beispielsweise Ausstellungen).

Im Rahmen dieser Bachelorarbeit wurde keine Ortung implementiert. Um das System also um einen weiteren großen Raum an Möglichkeiten zu erweitern, wäre eine möglichst genaue Ortung sehr wünschenswert. Daher wurden diverse Grundlagen für die Ortung bereits im Quellcode gelegt - so gibt es Objekte, die nach einer gewissen Zeit „ablaufen“ und bereits den definierten Nachrichtentyp `Positionsnachricht`.
Ebenso wenig, wie die Ortung wurde die Verschlüsselung bisher implementiert.
Eine mögliche Erweiterung des DMX-Clients für Arduino besteht darin, dem Client Art-Net „beizubringen“ und somit mehrere DMX-Universen gleichzeitig steuern zu können.

Auf die Fragestellung aus Abschnitt 1.1 zurückkommend lässt sich zusammenfassen: aktive RFID-Technologie bietet eine Möglichkeit, Bewegungsdaten stabil zu übermitteln und die Zuordnung von Kontaktereignissen zu erleichtern.
Damit bietet das ROBERTA-System schon jetzt innovativen Mehrwert für Veranstalter und ihre Gäste.
Aufgrund der Softwarearchitektur ist das System erweiterungsfähig und bietet noch zu erschließende Potentiale.

Anhänge

Glossar

C

Cross-Compiler Mit einem Cross-Compiler ist es möglich, Software auf einem System (Hostsystem) für ein anderes System (Zielsystem) zu kompilieren. Diese Zielsysteme können sich vom Hostsystem durch das Betriebssystem, andere Hardware (beispielsweise bei der CPU: 32-Bit, 62-Bit, Little-, oder Big-Endian, ARM, Intel, AMD), oder auch beidem unterscheiden

D

DMX Digital Multiplex Signal - ist ein Protokoll zur Steuerung von Bühnen- und Veranstaltungstechnik.

M

moc Der Meta-Object-Compiler (kurz: moc) ist ein für Qt entwickelter Compiler, der C++-Headerdateien einliest, dort nach `Q_OBJECT` sucht und bei erfolgreicher Suche den Quellcode für ein meta-Objekt dieser Klasse erstellt. In dieser Klasse werden unter anderem Signale und Slots C++-konform umgesetzt.

Model-View-Controller-Prinzip Beim Model-View-Controller-Prinzip (MVC) werden die Funktionalitäten der Anwendung in die Datenhaltung (Modell), die Präsentation (View) und schließlich Steuerung (Controller) unterteilt. Dadurch soll leichtere Wart- und Erweiterbarkeit von Software erreicht werden.

Q

Qt Qt ist ein plattformunabhängiges C++-Framework, welches den C++-Standard um die Schlüsselworte signal, slot und emit erweitert, die die Programmierung erleichtern sollen. Um dennoch C++-kompatibel zu bleiben, wird Qt mit dem `moc` ausgeliefert. Es bringt dabei Bibliotheken für diverse Anwendungen von GUI-Design, über Datenbank- und Dateisystemanbindung, bis hin zur Animation von Elementen und Medienwiedergabe mit.

R

RFID Radio Frequency Identification (*„Identifizierung mit Hilfe elektromagnetischer Wellen“*) ist eine Technologie, mittels derer Gegenstände oder Körper kontaktlos identifiziert werden können.

T

Transponder Ein Transponder ist ein Gerät, dass Signale entgegennimmt und automatisch beantwortet beziehungsweise weiterleitet.

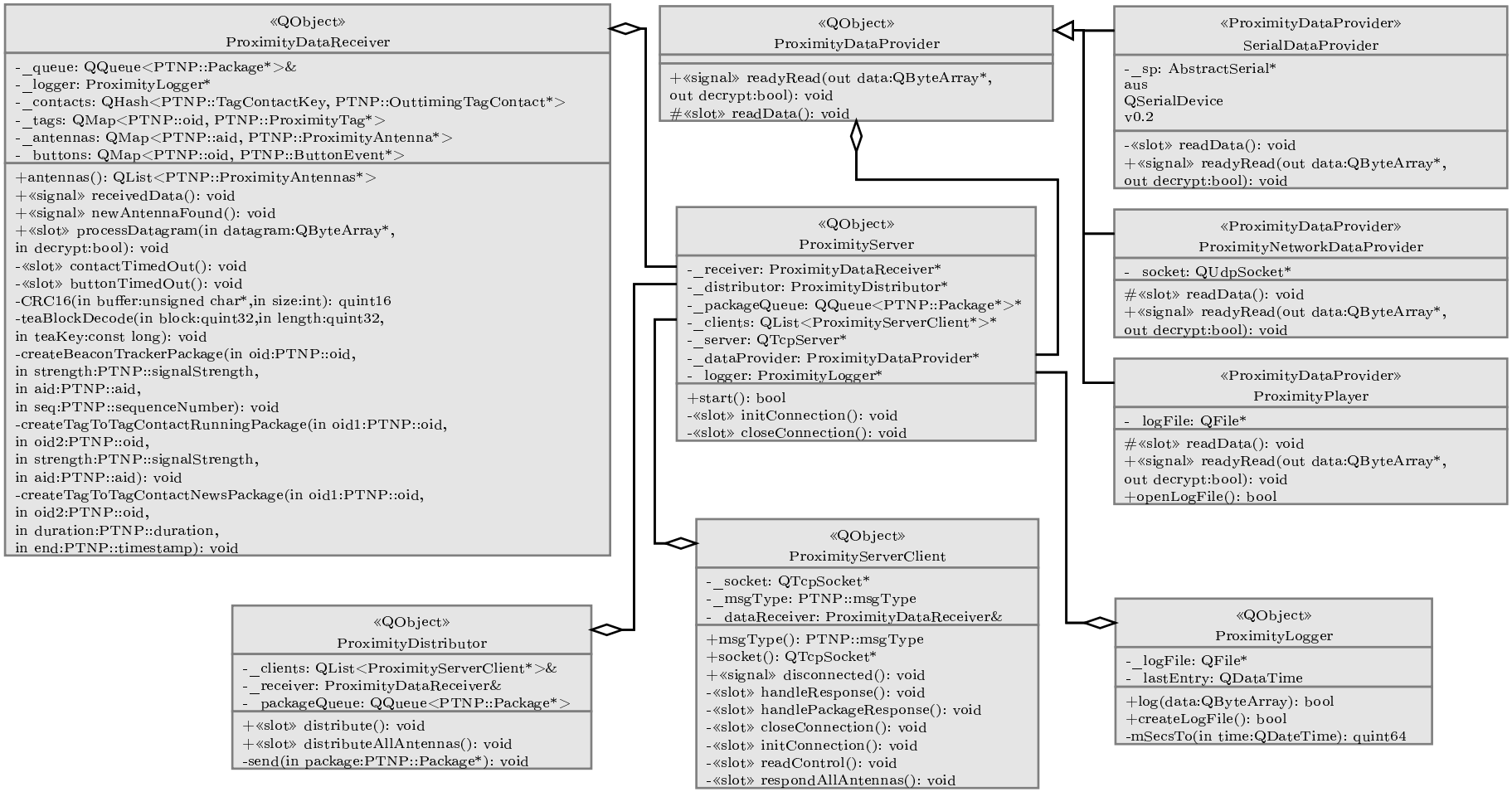


Abbildung .1: Klassendiagramm der RFID-Server-Komponente

Literaturverzeichnis

[l_Ack06] Ackermann, Norbert: *Lichttechnik: Systeme der Bühnen- und Studiobeleuchtung rationell planen und projekieren.* Zweite Aufl. Oldenburg Industrieverlag, 2006

[l_Ber08] Bernstädt, Herbert: *Digitale Steuersignale der Lichttechnik.* `http://www.hbernstaedt.de/KnowHow/Steuersignale/Steuersignale_02/Steuersignale02.htm`. Version: 2008. – [Online; abgerufen am 23. April 2010]

[l_BL04] Bieber, Christoph ; Leggewie, Claus: Interaktivität - Soziale Emergenzen im Cyberspace? In: *Interaktivität: Ein transdisziplinärer Schlüsselbegriff (Interaktiva, Schriftenreihe des Zentrums für Medien und Interaktivität)*, 2004, S. 7 – 14

[l_Bre10] Breul, Hannes: *Dynamische Visualisierung von Bewegungs- und Kontaktdaten aus RFID-basierten Lokalisierungsaufzeichnungen.* voraussichtlich 2010

[l_Büc06] Büch, Christian: *SPI – Serial Peripheral Interface.* `http://www.uni-koblenz.de/~physik/informatik/MCU/SPI.pdf`. Version: 2006. – [Online; abgerufen am 29. April 2010]

[l_Dal06] Dalhaus, Matthias: *WLAN Ortung innerhalb von Gebäuden mittels Signalstärkelinien.* `http://www.ikaros-projekt.de/index.php?name=CmodsDownload&file=index&req=getit&lid=64`. Version: 2006. – [Online; abgerufen am 21. April 2010]

[l_Fin08] FINKENZELLER, Klaus: *RFID-Handbuch. Grundlagen und praktische Anwendungen von Transpondern, kontaktlosen Chipkarten und NFC.* Fünfte akt. und erw. Aufl. Hanser Fachbuch, 2008

[l_GSHD08] GROSSMAN, Uwe ; SCHAUCH, Markus ; HAKOBYAN, Syuzanna ; DALHAUS, Matthias: RSSI-basierte WLAN-Ortung mit PDA (Personal Digital Assistant) innerhalb von Gebäuden. In: *Wireless Communication and Information: New Technologies and Applications*, 2008, S. 223 – 238

[l_Gün10] GÜNTHER, Andreas: *Konzeption und Entwicklung einer Smartphone-gestützten Schiffskontrolle zur Kollisionsvermeidung bei Schiffsmodellen.* 2010

[l_Lic10] LICENCE, Artistic: *Specification for the Art-Net II Ethernet Communication Standard.* `http://www.artisticlicence.com/WebSiteMaster/User%20Guides/art-net.pdf`. Version: 2010. – [Online; abgerufen am 06. Juni 2010]

[l_NXP08] NXP: *SC16IS740/750/760 - Single UART with I2C-bus/SPI interface, 64 bytes of transmit and receive FIFOs, IrDA SIR built-in support.* `http://www.sparkfun.com/datasheets/Components/SMD/sc16is750.pdf`. Version: 2008. – [Online; abgerufen am 05. Juni 2010]

[l_OFW09] ODENDAHL, Manuel ; FINN, Julian ; WENGER, Alex: *Arduino - Physical Computing für Bastler, Designer & Geeks.* Erste Aufl. O'Reilly, 2009

[l_O'N10] O'NEILL, Nick: *Facebook Tests Location Through RFID AT f8.* `http://www.allfacebook.com/2010/04/facebook-tests-location-through-rfid-at-f8/`. Version: 2010. – [Online; abgerufen am 30. Mai 2010]

[l_QM10] QUANTZ, Joachim ; MAYER, Pavel: Interaktive Installationen: Multitouch-Tische und Mobile Apps. In: *Kultur und Informatik: Interaktive Systeme*, 2010, S. 222 – 232

[l_Ran02] RANDT, Michael: *Bussysteme im Automobil.* http://www.carbussystems.com/pdfs/SPI_I2CTab.pdf. Version: 2002. – [Online; abgerufen am 01. Mai 2010]

[l_RN09] ROVING NETWORKS, Inc.: *WIFLY GSX RN-131G & RN-134 802.11 b/g wireless LAN Module - User Manual and Command Reference Version 2.08.* http://www.sparkfun.com/datasheets/Wireless/WiFi/WiFlyGSX-um.pdf. Version: 2009. – [Online; abgerufen am 05. Juni 2010]

[l_Sch00] SCHWERDTFEGER, Martin: *SPI - Serial Peripheral Interface.* http://www.mct.de/faq/spi.html. Version: 2000. – [Online; abgerufen am 29. April 2010]

[l_SGD09] SCHRÖDER, Joachim ; GOCKEL, Tilo ; DILLMANN, Rüdiger: *Embedded Linux: Das Praxisbuch.* Erste Aufl. Springer, Berlin, 2009

[l_Tan03] TANENBAUM, Andrew S.: *Moderne Betriebssysteme.* Zweite überarb. Aufl. Pearson Studium, 2003

[l_WB09] WÖRN, Heinz ; BRINKSCHULTE, Uwe: *Echtzeitsysteme: Grundlagen, Funktionsweisen, Anwendungen.* Erste Aufl. Springer, Berlin, 2009

[l_Wol07] WOLF, Jürgen: *Qt 4 - GUI-Entwicklung mit C++: Das umfassende Handbuch.* Erste Aufl. Galileo Press, 2007

weitere Internetquellen

[w_Arda] Arduino: *Arduino*. http://arduino.cc/. – [Online; abgerufen am 02. Juni 2010]

[w_Ardb] Arduino: *Arduino playground - DMXShield*. http://www.arduino.cc/playground/DMX/DMXShield. – [Online; abgerufen am 02. Juni 2010]

[w_dmx] dmxcontrol.de: *Art-Net - DMXCWiki*. http://www.dmxcontrol.de/wiki/Art-Net. – [Online; abgerufen am 06. Juni 2010]

[w_Pfe10] Pfeifer, Thomas: *Platinen ätzen mit der Direkt-Toner-Methode, Leiterplatten selber herstellen, Tonertransfermethode*. http://thomaspfeifer.net/platinen_aetzen.htm. Version: 2010. – [Online; abgerufen am 02. Juni 2010]

[w_sou99] soundlight.de: *SLH TechTips: DMX - was ist das?* http://www.soundlight.de/techtips/dmx512/dmx512.htm. Version: 1999. – [Online; abgerufen am 05. Juni 2010]

Bildquellen

[b_ard] ARDUINO.CC: *5-polige DMX-Verbindung*. http://www.arduino.cc/playground/uploads/DMX/5_pol_xlr.jpg. – [Online; abgerufen am 05. Juni 2010]

[b_den10] DENSO WAVE: *Data Capacity and Symbol Size*. http://www.denso-wave.com/qrcode/images-e/kcmqr.gif. Version: 2010. – [Online; abgerufen am 14. Mai 2010]

[b_fac10] FACEBOOK: *Welcome to Facebook Presence @ f8*. http://tctechcrunch.files.wordpress.com/2010/04/f8p.png. Version: 2010. – [Online; abgerufen am 30. Mai 2010]

[b_O'N10] O'NEILL, Nick: *Facebook Presence f8 Visualization*. http://www.allfacebook.com/wordpress/wp-content/uploads/2010/04/presence-visualization.jpg. Version: 2010. – [Online; abgerufen am 30. Mai 2010]

[b_Sch00] SCHWERDTFEGER, Martin: *SPI-Slave*. http://www.mct.de/faq/spi1.gif. Version: 2000. – [Online; abgerufen am 12. Mai 2010]

[b_Sch08] SCHLOSSAREK, Erich: *QR-Code die Zukunft der Kommunikation?* http://www.escha54.de/wp-content/uploads/2008/12/linux_bs-300x300.png. Version: 2008. – [Online; abgerufen am 15. Mai 2010]

[b_soua] SOUNDLIGHT: *Das DMX-512 Protokoll*. http://www.soundlight.de/techtips/dmx512/vpdmxti.gif. – [Online; abgerufen am 18. Mai 2010]

[b_soub] SOUNDLIGHT: *Schema der DMX-Übertragung für Sender und Empfänger.* http://www.soundlight.de/techtips/dmx512/vpdmxs.gif. – [Online; abgerufen am 03. Juni 2010]

[b_tro] TROLLTECH: *Signals and Slots.* http://doc.trolltech.com/4.6/images/abstract-connections.png. – [Online; abgerufen am 02. Juni 2010]

[b_wika] WIKIPEDIA: *SPI - Kaskadierung.* http://upload.wikimedia.org/wikipedia/commons/thumb/5/50/SPI_Kaskadierung.svg/600px-SPI_Kaskadierung.svg.png. – [Online; abgerufen am 01. Juni 2010

[b_wikb] WIKIPEDIA: *SPI - Stern.* http://upload.wikimedia.org/wikipedia/commons/thumb/b/b0/SPI_Stern.svg/600px-SPI_Stern.svg.png. – [Online; abgerufen am 01. Juni 2010

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Ort, Datum	Unterschrift