Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Diplomarbeit

Martin Sukale

Konstruktion eines Netzwerkes eingebetteter Systeme für interaktives Design

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Martin Sukale

Konstruktion eines Netzwerkes eingebetteter Systeme für interaktives Design

Diplomarbeit eingereicht im Rahmen der Diplomprüfung

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

Betreuender Prüfer: Prof. Dr. Kai v. Luck
Zweitgutachter: Prof. Dr. Gunter Klemke

Abgegeben am 18.08.2008

Martin Sukale

Thema der Diplomarbeit
Konstruktion eines Netzwerkes eingebetteter Systeme für interaktives Design
Stichworte
Physical Computing, Ambient Intelligence, Electronic Art, interaktives Design, verteilte eingebettete Systeme, Sensornetzwerke
Kurzzusammenfassung
Ausgehend von einer Analyse vorhandener Open Source Lösungen für den Einsatz in Physical Computing Projekten und zur Konstruktion interaktiver, multimedialer Installationen, wird ein System vorgestellt, welches neuere Technologien zusammenführt und an geeigneten Punkten erweitert und verbessert. Ziel ist ein offener modularer Aufbau, der auch von Anwendern ohne computertechnisches Fachwissen eingesetzt werden kann. Exemplarisch werden sog. Light-Motes und Sensor-Motes konstruiert zur Verwendung in Ambient Intelligence-Szenarien.

Martin Sukale

Title of the paper
Embedded system networks for interactive design
Keywords
physical computing, ambient intelligence, electronic art, interaction design, distributed embedded systems, sensor networks
Abstract
Analyzing existing open source solutions for physical computing and construction of interactive multimedia installations a system is introduced, that assembles new technologies and improves and extends them where necessary. Aiming at an open modular structure which can be used in multidisciplinary projects from users without deep knowledge of computer science. As an example so called embedded light-motes and sensor-motes are implemented for use in ambient intelligence networks.

Inhaltsverzeichnis

1 Einleitung

An den Schnittstellen von Kunst und Computertechnik ist eine Spielwiese für „Kreative Köpfe“ entstanden, auf der - abseits der industriellen Einsatzgebiete von Technischer Informatik - Experimente mit neuen Technologien stattfinden können, ohne zwangsläufig auf Nützlichkeit und kommerziellen Erfolg zu achten. Studenten und Professoren unterschiedlicher Disziplinen können hier fächerübergreifend Projekte entwickeln und sich gegenseitig ergänzen. Für Studenten bieten sich neue Orientierungsmöglichkeiten: Während von einem Kunststudenten beispielsweise nicht zu erwarten ist, dass er oder sie sich fundierte Kenntnisse in Programmiertechniken aneignet, um künstlerische Konzepte zu realisieren, so ist auf der anderen Seite von Studierenden der Informatik nicht unbedingt zu erwarten, dass diese besonders ästhetische oder gestalterisch ansprechende Zielsetzungen einer pragmatischen Lösung vorziehen. Durch eine universelle, interdisziplinäre Lehre und Forschung kann ein kreativer und evtl. kritischer Umgang mit der - in vielen Lebensbereichen dominierenden - modernen Computertechnologie gefördert werden, der letztlich auch unter wirtschaftlichen Gesichtspunkten („Gaming-Industrie“) Berechtigung erlangt.

Ich möchte mich mit dieser Arbeit auf das Terrain von interaktiven Kunstinstallationen begeben und habe mich hierzu im Vorfeld im Rahmen einer Studienarbeit mit den neueren technologischen Entwicklungen auf diesem Gebiet befasst (Sukale (2008)). Dabei habe ich erfahren, dass die Zusammenarbeit von Künstlern und Naturwissenschaftlern eine lange Tradition hat, dass es an vielen renommierten Universitäten Einrichtungen gibt, die sich mit *Electronic Art* beschäftigen, dass Professoren und Studenten der Künste großes Interesse an Informatik haben, und dass es glücklicherweise auch schon viele interessante und ausgereifte Lösungen auf diesem Gebiet gibt. Bei der Recherche zu dieser Arbeit ist mir vielmehr sogar aufgefallen, dass es eigentlich *nichts gibt, was es noch nicht gibt*. Während des Schreibens bin ich immer wieder über Projekte gestolpert, die Ansätze und Ideen von mir in der Zwischenzeit bereits mit großem Engagement realisiert haben. Bewusst habe ich den Kontakt zu Kunstschaffenden gesucht (u.A. bei der Durchführung und Betreuung von Workshops an der *HBK-Saar* und dem *Department Design* der *HAW-Hamburg*) und mir die Fragen gestellt: An welchen Punkten gibt es noch Entwicklungsbedarf, um die zahlreichen kreativen Ideen zu realisieren? Welche Projekte können noch „weitergesponnen“ werden, wo lohnt es sich nach Verbesserungen zu suchen?

Einen Überblick über vorhandene (Open Source) Software und Hardware zur Gestaltung und Komposition von Kunstwerken habe ich in meiner Studienarbeit gegeben und ich möchte nun Lösungen vorstellen, wo und wie diese konkret zum Einsatz kommen können.

Eine Gemeinsamkeit der konkreten Projekte ist die Verwendung von Elektrolumineszenz-Technik in Form von *Light Emitting Diodes* (LED), welche in den vorangegangenen Jahren immer mehr an praktischer Bedeutung gewonnen hat: Aktuell ist davon auszugehen, dass die Effizienzsteigerung und immer kostengünstigere Produktion von LEDs in absehbarer Zeit

die klassische Glühbirne aber auch deren Nachfahren wie z.B. die Halogen-Leuchtmittel und Leuchtstoffröhren vom Markt drängen werden, und wir in vielen Bereichen LEDs einsetzen werden. Nicht zuletzt unter den Gesichtspunkten Umwelt- und Klimaschutz wird die LED gerade auch auf dem Theaterlicht- und Veranstaltungstechniksektor weiter an Bedeutung gewinnen. Das Thema Lichttechnik und Bühnentechnik ist in anderen Arbeiten (vgl. Stickel (2004)) ausführlich behandelt worden und soll hier nicht vertieft werden. Ich möchte innovative Verwendung von Lichttechnik aufzeigen.

1.1 „Light-Motes“, „Smart-Pixel“

Abbildung 1: Pixi-LED Smart-Pixel, Quelle: Artistic Licence (UK) Ltd.

LEDs gibt es mittlerweile in allen Farben und Größen. Kleinste LEDs sind derzeit im SMD Package 0402 erhältlich (1.0mm x 0.5mm Grundfläche). Sie lassen sich überall kostengünstig und stromsparend integrieren. Ebenso wird bei *SoCs* [1] und Mikroprozessoren das Verhältnis von Größe und Preis zur Leistungsfähigkeit und Stromaufnahme fortlaufend besser. Im Zuge dessen werden konventionelle Leuchtmittel in der Bühnentechnik durch LED-Cluster ersetzt und mit einem Controller zur Steuerung der Farbe und Intensität ausgerüstet. Vorteile sind die geringere Leistungsaufnahme, geringere Wärmeentwicklung und die lange Lebensdauer. LED-Cluster gibt es in vielen Varianten: als „Dot-Matrix“, „Stripes“ oder flexible Bänder mit SMD-LED Bestückung. Monochrom oder als RGB[2]-Variante. Die meisten dieser Produkte besitzen selber - außer evtl. einer aktiven Stromregelung - keine weitere „Intelligenz“. Daher werden sie mit entsprechenden Vorschaltgeräten (Dimmer, DMX-Receiver) kombiniert. In dieser Arbeit möchte ich einen Ausblick entwerfen, was mit dieser Technik in Zukunft noch möglich sein könnte. Hier beziehe ich mich auf bereits umgesetzte und entworfene Visionen und übertrage diese auf den hier vorgestellten Anwendungsbereich interaktiver Kunstinstallationen.

[1] System on a Chip
[2] (R)ed, (G)reen, (B)lue

Hierzu möchte ich im Weiteren die Begriffe *Light-Motes* in Anlehnung an die sog. *Berkeley-Motes* und *Smart-Pixel* in Anlehnung an den sog. *Smart Dust* einführen.

Stanislaw Lem beschreibt in seinen Science-Fiction Erzählungen (z.B. Lem (1983)) die Evolution von Waffensystemen, die sich zu unbesiegbaren Schwärmen kleiner Kampfroboter organisieren („Schwarze Wolke“). Diese *mikroelektromechanischen Systeme* (MEMS) werden heute auch *Smart Dust*[3] genannt.

Von Seiten des Militärs und der Rüstungsindustrie wird aktuell an dieser Thematik geforscht: Konstruktion minituarisierter intelligenter (*smarter*) Einheiten z.B. zur Überwachung von feindlichem Terrain oder Konstruktion intelligenter Streumunition (vgl. z.B. BEA (2006)). So wurden z.B. in Berkeley[4] in Zusammenarbeit mit der DARPA [5] sog. *Berkeley-Motes* entwickelt: eingebettete Systeme mit kabelloser Kommunikationsschnittstelle zum Aufbau von Sensornetzwerken bzw. allgemein von Multiagenten-Systemen (vgl. Jakubowsky (2005)). In der Natur haben diese Systeme z.B. Insektenschwärme als

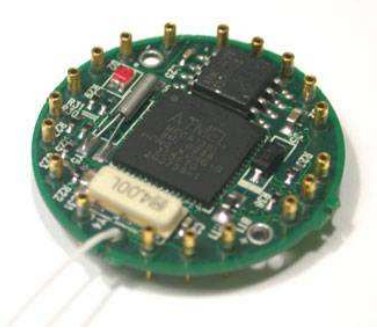

Abbildung 2: Berkeley Mote, MICA2Dot Implementation, Quelle: Crossbow Technology, Inc

Vorbild: Lem spricht von *Synsekten*[6]. An diesem Punkt knüpfe ich an und möchte von der (Spionage-)Wanze zum „sympathischen Glühwürmchen“ überleiten: es sollen für den Einsatz in Kunstinstallationen Light-Motes konstruiert werden, die sich ebenfalls in Schwärmen organisieren lassen. Jeder dieser lichtemittierenden Knoten, wird so in einem Glühwürmchen-Schwarm zu einem Lichtpunkt bzw. Pixel. Gepaart mit der nötigen Intelligenz wünsche ich mir Smart-Pixel, die ich individuell ansprechen kann, die auf ihre

[3] intelligenter Staub

[4] genauer: University of California, Berkeley, USA

[5] The Defense Advanced Research Projects Agency, USA

[6] „Deshalb bildeten nicht menschenähnliche Automaten die Armeen neuen Typs, sondern synthetische Insekten (Synsekten), keramische Mikrokrustentiere, Regenwürmer aus Titan und fliegende Pseudoinsekten [..] Die fliegenden Synsekten wurden zu einer Art Verschmelzung von Flugzeug, Pilot und Geschoß zu einer Miniatureinheit. Die Mikroarmee wurde zu einer operativen Einheit, die nur als Ganzes die von ihr geforderte Kampfkraft [..] besaß (wie - analog - nur der ganze Bienenschwarm eine selbständige Überlebenseinheit [..] ist)“

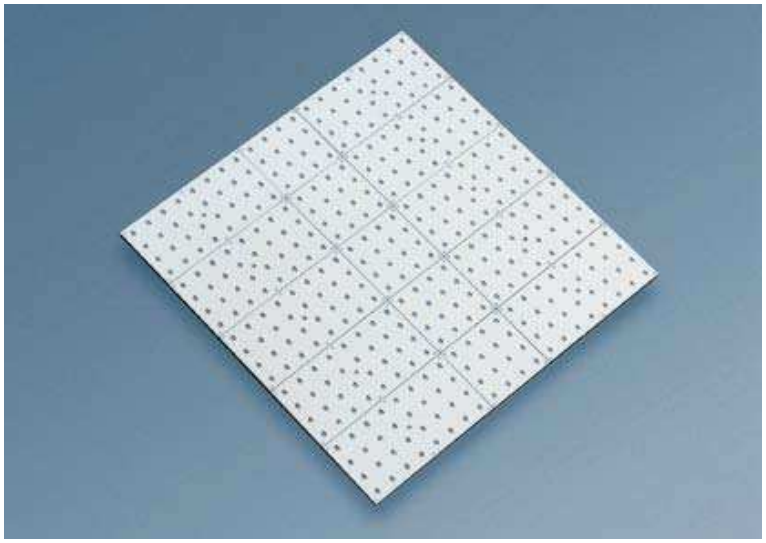

Abbildung 3: LED-Paneel mit 400 einzeln ansteuerbaren LED-Pixeln, Abdruck mit freundlicher Genehmigung der Schnick-Schnack-Systems GmbH.

Umgebung reagieren können und ein Eigenleben führen (*Stand-Alone*-Mode).

Die Entwicklungen auf diesem Gebiet sind weitreichend: von intelligenten Etiketten im Supermarkt bis zu fliegenden Kampfrobotern und medizinischen Sonden. Ich möchte mich von diesem weiten Feld, vor allem von militärischen Anwendungsbereichen abgrenzen und den Fokus ausschließlich auf gestalterische Aspekte und das Ausgeben von Farbe und Licht beschränken. Ähnliche Ansätze gibt es bereits: z.B. *Dot-Matrix-Displays* (3) bestehend aus vielen einzeln ansteuerbaren LED-Pixeln, einzelne Smart-Pixel: *BlinkM*-LED-Dots (4) und *Fireflys* (5) Entsprechende objektorientierte Programmiersprachen zum Erzeugen

Abbildung 4: BlinkM Smart-Pixel, Quelle: Blinkm Datasheet v20080130a, ThingM Corporation

virtueller Partikelschwärme gibt es bereits (z.B. *Processing*, vgl. Reas und Fry (2007)). Die neueren technischen Entwicklungen machen es möglich, diese softwareseitige Ob-

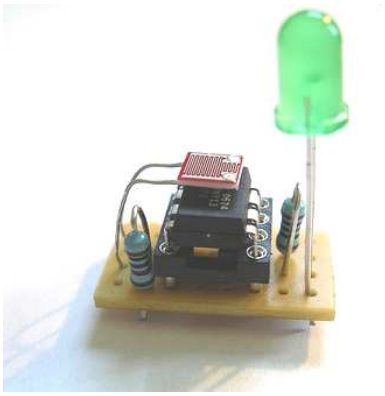

Abbildung 5: Firefly Smart-Pixel, Abdruck mit freundlicher Genehmigung von Alex Weber

jektinstanzierung auf Hardwareobjekte auszuweiten und Hardware und Software einfach zu verknüpfen (Stichwort: *Physical Computing*). Die Tendenz geht wieder dahin, „echte“ Dinge zu kreieren („Making Things“).

1.2 Interaktives Design

Interaktives Design, das Einbinden von interaktiven Elementen in die Gestaltung unterschiedlicher Medien soll hier thematisiert werden. Kunstinstallationen und Designprojekte müssen über eine interaktive Oberfläche komponiert bzw. programmiert werden können und ebenso sollen die Installationen selber interaktive Eingriffsmöglichkeiten für das Publikum bieten oder zumindest einfach ein- und auszuschalten sein. *Interaction Design* und *Usability* sind weite Themenfelder und es sollen hier nur Interaktionsmöglichenkeiten aufgezeigt und zur Verfügung gestellt werden, ohne en detail auf die speziellen Anforderungen und Gestaltungstechniken von interaktiven Systemen einzugehen. Die Möglichkeiten sind vielfältig: Die Unterhaltungsindustrie bietet uns allerlei „Spielzeug“: Touchscreens, Game-Controller, LPS [7], Bodymonitoring-Sensoren, Beschleunigungs-, Neigungs- und Winkel-sensoren (z.B. Nintendos „Wiimote“). Auf der Softwareseite haben wir ausgefeilte Oberflächen-Gestaltungs-Werkzeuge und Plattformen sowie *Icon*-, *Widget*- und *Theme*-Bibliotheken.

1.3 Überblick

Im Folgenden werden exemplarisch Szenarien für interaktive Kunstinstallationen mit dem Schwerpunkt Lichtkunst vorgestellt. Das erste Beispiel ist ein typischen Szenario für *Electronic Art* im professionellen Ausstellungsbetrieb. Das zweite Beispiel ist typisch für studentische Projekte an der Schnittstelle von Kunst und Computertechnik. Das dritte Beispiel

[7] Local Positioning Systeme

ist ein typisches „Weekend-Project“, eine lustige „Bricolage“. Eingehend auf diese Beispiele werden die gemeinsamen Anforderungen extrahiert und daraus ein System, welches diesen entspricht entworfen. Das vorgestellte System wird in seine Hauptmodule unterteilt: *User-Interface, Controller, Motes* und die verbindenden Kommunikationsprotokolle. Diese werden in eigenen Kapiteln behandelt, in denen jeweils Beispielimplementationen und Prototypen für den Einsatz in den Beispielszenarien vorgestellt werden. Abschließend gebe ich einen Ausblick und versuche diese Arbeit in den stetig wachsenden Kosmos von ähnlichen Projekten einzuordnen.

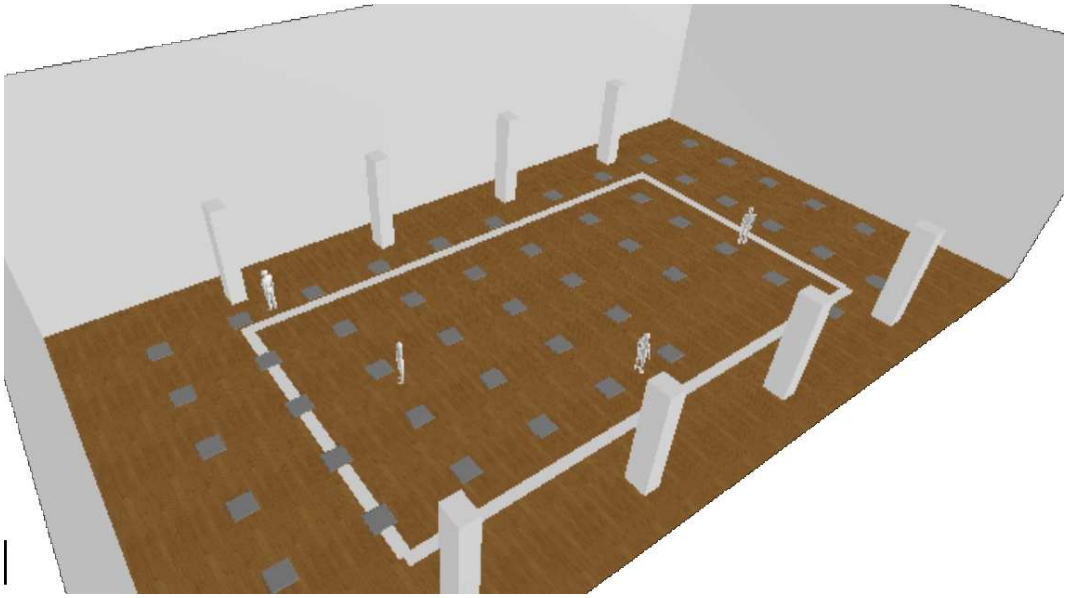

Abbildung 6: 3D-Simulation Leuchtobjekt-Feld, Quelle: Daniel Hausig

2 Beispielszenarien

2.1 Beispiel 1: Heidenheim Kunstinstallation

Künstlerisches Konzept

Ein altes, zur Kunsthalle umfunktioniertes Schwimmbad soll „bespielt“ werden. Hierfür sollen ca. 50 Glasplatten-Leuchtobjekte am Boden angeordnet werden. Der Raum soll zusätzlich beschallt werden. Anknüpfend am Thema Schwimmbad, werden Wassertropfenähnliche, künstliche oder verfremdete natürliche Geräusche zu hören sein, welche korrespondierende Farbverläufe und Bewegungen auf den Leuchtobjekten auslösen. Ein imaginärer Flüssigkeitstropfen fällt zu Boden, trifft mit entsprechendem Geräusch auf und löst hier einen Leuchtimpuls auf einem Glasobjekt aus, der sich durch den Raum über weitere Objekte fortpflanzt. Die Objekte sind lose als Matrix angeordnet und sollen in vorkomponierten Animationsmustern und Farbverläufen aufleuchten.

Technisches Konzept

Die große Anzahl der Leuchtobjekte macht einen Versuchsaufbau im Atelier aus Platzgründen schwierig. Daher wird eine spezielle Software zur 3D-Simulation eingesetzt. Leuchtobjekte werden als 3D-Modelle angelegt und im virtuellen Raum verteilt. Geräusche werden aufgezeichnet und mit Klangbearbeitungs-Software manipuliert, in MIDI-Noten umgewandelt und zerstückelt. Geräusche und MIDI-Daten werden in Musik-Kompositions-Software auf einer Zeitachse arrangiert. MIDI-Events triggern weitere Klänge und steuern gleichzei-

tig Farblicht-Kanäle einer Theaterlicht-Software. Diese wiederum gibt Daten an DMX512-Schnittstellen-Hardware weiter, welche die einzelnen Leuchtobjekte anspricht. Diese Hardware wird im Rahmen dieser Arbeit auf Basis einer offenen Hardware-Plattform entwickelt. Zur Komposition, Wiedergabe in Echtzeit und Simulation wird ein sehr leistungsfähiger Rechner benötigt. Für den Ausstellungsbetrieb wird dann die Komposition auf einen reinen Wiedergabe-Rechner übertragen, der nur noch die aufgezeichneten Audio- und Lichtsteuer-Daten abspielt. Weiterführend kann ein interaktives Element (z.B. ein PIR-Sensor, zur Erfassung von menschlicher Infrarotstrahlung bzw. deren Bewegung im Raum) hinzugefügt werden. Der Wiedergabe-Rechner soll über geeignete Schnittstellen und grafische Oberflächen konfigurierbar sein.

2.2 Beispiel 2: Ambient Intelligence Tent

Künstlerisches Konzept

Als interdisziplinäres Studentenprojekt des Departments Design und des Departments Informatik soll eine sog. *Ambient Intelligence* Installation aufgebaut werden. Es soll ein Zelt entstehen mit semitransparenter Aussenhülle, welches nach aussen und innen Farblicht-Informationen abstrahlen kann in Reaktion auf das Ausstellungspublikum. Publikumsbewegungen und ggf. individuelle Daten wie z.B. Pulsfrequenz, Körpertemperatur von Besuchern sollen erfasst werden und mit dem umgebenden Raum interagieren. Es sind narrative Elemente möglich und das Erzeugen von künstlichen Emotionen. Beispielsweise reagiert die Zelthülle mit rotem, warmen Licht, wenn Besucher mit niederiger Körpertemperatur oder langsamen Puls in die Nähe kommen, bzw. mit blauem Lichtspiel wenn „erhitzte Gemüter“ lokalisiert werden. Die gestalterischen Möglichkeiten von Light-Motes sollen erforscht werden. Interaktionskonzepte sollen entwickelt werden.

Technisches Konzept

Für die Wiedergabe von Farbinformationen kann hier wie in Heidenheim DMX512-Hardware bzw. *Light-Motes* eingesetzt werden. Diese dimmen LED-Cluster, die z.B. in die Zeltwandung integriert werden können oder z.B. auch als *Wearables* in Kleidung eingenäht werden können, um den Gemütszustand einzelner Personen anzuzeigen. Ebenfalls in die Kleidung integrierbar sind - ähnlich zu den Light-Motes aufgebaute - Mikroprozessor-basierte Systeme zur Erfassung von Sensorwerten (Körpertemperatur, Puls, etc.), sog. *Bodymonitoring* (vgl. Tetzlaff (2008)). Zusätzliche Informationen zur Publikumsbewegung im Austellungsraum können von *Local Positioning Systemen* (LPS) erfasst werden. Gesammelte Positions- und Sensor-Daten sollten zentral von einem Steuerungs-Rechner erfasst werden, und hier vorkomponierte, dem künstlerischen Konzept entsprechende Ausgabeinformationen an die Ausgabegeräte verteilt werden. Zudem wird ebenfalls eine Konfigurations und Kompositions-Ebene benötigt, über die einfach Modifikationen am System vorgenommen werden können.

Abbildung 7: LED-”POV”-Ventilator CL-F005 von Shenzhen Cailiang Electronics Co.,Ltd.

2.3 Beispiel 3: „Persistence Of Vision“

Künstlerisches Konzept

Eine Unzulänglichkeit der menschlichen Wahrnehmung ermöglicht es technischen Apparaten wie z.B. der *Laterna Magica* [8] durch das schnelle Hintereinanderschalten von Einzelbildern dem Auge bewegte Bilder „vorzugaukeln“. Das menschliche Gehirn kombiniert diese zu einem optischen Gesamteindruck. Ab einer gewissen Schaltfrequenz wird die Trägheit der menschlichen Wahrnehmung zu groß, um einzelne Zustände zu erkennen und die Bilder fangen an zu „laufen“. Gängige Werte bei analoger Filmwiedergabe und digitaler Videotechnik sind z.B. 24 Bilder bzw. Frames pro Sekunde. In der LED-Technik wird dieses ”Defizit” ebenfalls genutzt: LED-Dimmer schalten ihre PWM-Ausgänge mit Frequenzen von typischerweise 250Hz. Das menschlichen Gehirn kann so nur noch eine gedimmte Lichtintensität wahrnehmen. Dieser Effekt wird im Englischen gerne mit *Persistence Of Vision* (in etwa ”Beständigkeit der Wahrnehmung”) umschrieben und es sind unter diesem Schlagwort Kunstobjekte und Spielzeug-Produkte entstanden, die reziprok zu den bewegten Bildern aus bewegten und blinkenden Lichtpunkten wieder ein Standbild erzeugen (Stroboskop-Effekt). In einem von mir betreuten Workshop zum Thema Lichttechnik tauchte eine Studentin mit einem kleinem Ventilator auf, der auf einem Flügel 5 LEDs montiert hatte, welche bei einer bestimmten Drehgeschwindigkeit einen scheinbar stehenden Text ausgab. Mit nur 5 LEDs kann so ein Vielfaches an virtuellen Pixeln ausgegeben werden (siehe 7). Im selben Kurs

[8] Die Laterna Magica (”Zauberlaterne”) ist einer der Vorläufer aktueller Filmprojektoren: Sie projeziert auf Glas gemalte Bilder. Bildwechsel sind z.B. durch drehen einer Scheibe mit mehreren Bildern möglich, so dass auch bewegte Bilder gezeigt werden können

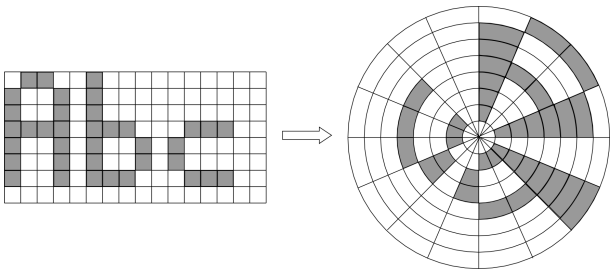

Abbildung 8: Dot-Matrix-Display Beispiel: Textausgabe

wurde häufiger das Interesse an *Dot-Matrix-Displays* geäussert[9], so dass sich hier ein Anknüpfungspunkt für Projekte ergeben hat: Alternativ zum Aufbau einer Anzeige mit vielen im Raster angeordneten LEDs (oder z.B. dem Einsatz von Projektoren und Flachbildschirmen), soll ein Gerät entwickelt werden mit einer Reihe weniger, sich bewegender LEDs, die mit richtiger Ansteuerung und hinreichender Bewegungsgeschwindigkeit relativ zum Betrachter ein Dot-Matrix-Display bilden. Einbau in Räder von Autos und Fahrrädern[10] oder in sich drehende Trommeln und Springseile bietet sich an und schafft Raum für weitere kreative Konzepte.

Technisches Konzept

Auf der Technischen Seite muss ein mechanischer Antrieb entwickelt werden: im einfachen Fall ein Ventilator oder ein umgebautes Fahrrad. Für akkurate Darstellung muss ein Drehfrequenz-Sensor eingebaut werden (oder die Drehgeschwindigkeit konstant gehalten werden). Kern bildet dann eine Reihe von individuell steuerbaren LEDs, die von einem Controller zum Richtigen Zeitpunkt geschaltet werden (z.B. Shiftregister und LED-Treiber Bausteine). Es soll entweder ein kommerziell verfügbares Spielzeug umgebaut werden oder ein ähnliches Gerät mit einfachen Mitteln (Arduino, LED-Cluster-Platine) nachgebaut werden. Diese Geräte sollen zusätzlich mit einer offenen, standardisierten Schnittstelle (DMX512, OSC) ausgerüstet werden, so dass Pixel der virtuellen Matrix individuell angesprochen werden können. Jeder Pixel erhält eine Addresse. Anbindung an vorhande Kompositionssoftware ist möglich.

[9] vgl. http://blinkenlights.de/
[10] vgl. http://ladyada.net/make/spokepov/

3 System Entwurf

Die vorgestellten Szenarien lassen sich alle mit einem Use-Case-Diagramm umschreiben (siehe 9). Der Küstler oder die Künstlerin gestaltet einen interaktiven Raum oder ein „intelligentes“ Produkt, welche eine bestimmte Anzahl von Ausgabemöglichkeiten (Licht und Klang) und Eingabemöglichkeiten (manuelle Eingaben, Umweltdaten) benötigen. Für diese muss es entsprechende Hardwareknoten geben zur Ausgabe und Erfassung von Daten (*Light-Motes*, *Audio-Motes* und *Sensor-Motes*). Ebenso wird ein bestimmtes Verhalten komponiert (sense->act), welches in geeigneter Weise programmiert und ausgeführt werden soll. Hierfür wird ein *User-Interface* zum Komponieren und Konfigurieren benötigt sowie ein entsprechender *Controller*, der die Datenverarbeitung nach Wünschen des Künstlers, entsprechend der Komposition übernimmt. Im laufenden Betrieb muss das Verhalten überwacht und angepasst werden können. Personal im Austellungsraum muss die Anlage einfach ein- und ausschalten können. Für diese Funktionen ist minimal ein Eingabegerät mit User-Interface erforderlich, ein *Stand-Alone-Controller* für den laufenden Betrieb sowie ein Knoten zur Ausgabe und ein Knoten zur Erfassung von Daten. Einfachstes Szenario ist ein Lichtpunkt, der über das User-Interface mit einem Sensoreingang verbunden wird und Sensorwerte via Lichtintensität darstellt (17).

3.1 Einteilung in Funktionseinheiten

Das zu konstruierende System lässt sich in drei grundlegende Funktionseinheiten aufteilen:

- **User-Interface** Methoden zur Komposition, Konfiguration und Monitoring der laufenden Installation
- **Controller** implementieren das Verhalten, stellen Daten zur Verfügung, wickeln die Kommunikation zwischen den Komponenten ab, laufen *stand-alone*
- **Motes** Ausgabe von Licht und Klang, Erfassung von Sensorwerten

3.2 Allgemeine Anforderungen

Analog zu industriellen Projekten tauchen die folgenden, allgemeingültigen Anforderungen auf:

- **Flexibilität** der künstlerische Entwurf soll so frei wie möglich, bestenfalls ohne technische Vorgaben entstehen können
- **Wiederverwendbarkeit** bei temporären Austellungen soll die Hardware im Folgeprojekt weiterhin einsetzbar sein
- **Skalierbarkeit** um gestalterische Konzepte fortspinnen, verbessern zu können sollen spontan beliebig Komponenten hinzugefügt werden können

Abbildung 9: Typischer Use Case interaktiver, multimedialer Installationen

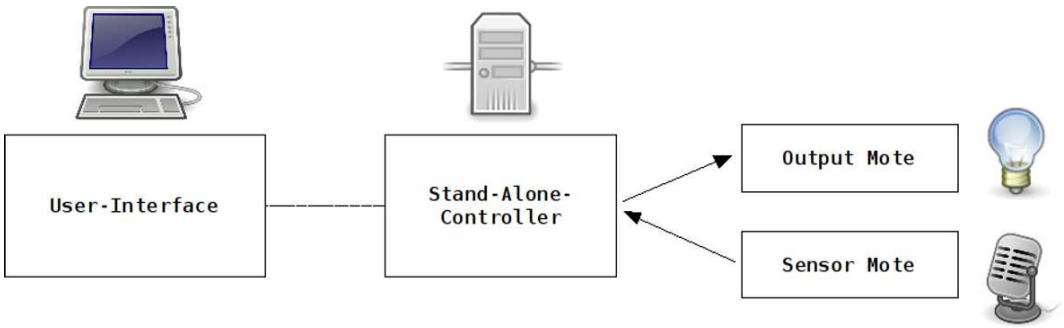

Abbildung 10: System Überblick

- **Bedienbarkeit** die Projektteilnehmer haben evtl. wenig technisches Vorwissen, es muss möglich sein, eine Installation intuitiv und einfach „zusammenzustecken“, *Plug-And-Play*
- **Integrierbarkeit** die Komponenten müssen evtl. im Austellungskontext unsichtbar werden, oder z.B. tragbar sein
- **Autonomität** die Komponenten müssen ohne Anwesenheit eines Technikers ”Stand-Alone” lauffähig sein, ohne User-Interface eingaben starten - z.B. bei Installationen im öffentlichen Raum und über längere Zeit
- **Zuverlässigkeit** Ausfälle der Technik beim Aufbau, Eröffnung und Betrieb von Kunstaustellungen sind teuer und schlecht für die Reputation
- **Low-Cost** kreative Ideen sollen unabhängig von einem großem finanziellen Budget z.B. im Zusammenhang von studentischen Projekten realisierbar sein, vorgestellte Lösungen werden sich nur durchsetzen, wenn diese auch kostengünstig einsetzbar sind

3.3 Umsetzung

Wie in Sukale (2008) erläutert, sollte dieses System auf offenen Schnittstellen, Standards und freier Hard- und Software basieren, um grösstmögliche Flexibilität und Wiederverwendbarkeit zu gewährleisten. Sicher ist es möglich, Systeme zu kaufen, die das gewünschte Verhalten mehr oder weniger gut mit kommerzieller Veranstaltungs- und Unterhaltungtechnik realisieren (z.B. moderne Lichtstellanlagen mit Unterstützung für unterschiedliche Multimediaformate). Dies mag für ein professionelles Vorhaben wie das erste Beispiel (2.1) evtl. noch finanzierbar sein - und hier wird z.B. auch kommerzielle 3D-Bühensimulation verwendet - bei einem studentisches Projekt wie dem zweiten Szenario (2.2 ist das finanzielle Budget evtl. nicht ausreichend, um kommerzielle Lösungen einzukaufen. Fraglich ist ebenfalls, ob es überhaupt geeignete kommerzielle Lösungen gibt. Das dritte als Beispiel vorgestellte Produkt ist zwar käuflich zu erwerben, doch die erhältlichen Varianten sind

von vorneherein nicht flexibel genug, um kreative Ideen damit zu verwirklichen. Eine universellere Lösung wird im folgenden entworfen, sie ist konsequenterweise an folgendende non-funktionale Anforderungen gebunden:

- **offene Standards** es sollen keine proprietären Lösungen verwendet werden, diese sind meist teuer und unflexibel
- **Open Source Software** Software muss quelloffen sein, um sie modifizieren zu können, Lizensierung unter GPL[11] o.Ä. ist notwendig
- **Open Hardware** analog zu Open Source Software sollen ebenso die verwendeten elektronischen Schaltungen offen liegen und ggf. angepasst werden können
- **portierbar und embeddable** Software soll auf unterschiedlichen Plattformen (z.B. tragbaren Geräten) lauffähig sein, es soll nicht für jedes Projekt neue eigene Hardware angeschafft werden müssen
- **„keep it simple“** die einfache Lösung wird der aufwendigeren vorgezogen, dies ist didaktisch sinnvoll bei universitären Projekten und verhindert, dass die Technik die Gestaltung dominiert

3.3.1 Model View Control

Wie in der objektorientierten Programmierung üblich, soll sich auch bei diesem Systementwurf verteilter Soft- und Hardware-Objekte grob an das allgemein bekannte „Model View Control“-Paradigma gehalten werden. Dieses beinhaltet die Trennung von Datenmodell, Präsentation und Steuerung, was hier u.A. schon durch die hardwareseitige Verteilung der einzelnen Komponenten gewährleistet ist. Das Anzeigen der Daten wird hier vom User-Interface übernommen, ist aber durch die Wahl geeigneter Datenformate und Schnittstellen (OSC, DMX512 s.u.) vom Datenmodell entkoppelt und durch andere „Viewer“ ersetzbar.

[11] GNU General Public License - Allgemeine Öffentliche GNU-Lizenz - ist eine freie Copyleft-Lizenz für Software und andere Arten von Werken

4 Kommunikation

4.1 Anforderungen

Es ergibt sich folgende Problemstellung:
Wir haben einen Raum mit einer „Wolke“ von - aus technischer Sicht - zufällig angeordneten Leuchtpunkten und Sensoren. Diese sollen mit einem oder mehreren Controllern verbunden werden, welche wiederum an ein User-Interface angeschlossen werden müssen. Aus Anwendersicht betrachtet: Ein User (Künstler) möchte von einem User-Interface an irgendeinem ihm genehmen Ort auf eine beliebige Anzahl von Sensoren und Leuchtpunkten zugreifen. Dieser Zugriff muss über verschiedene Controller abgewickelt werden, die das System auch nach dem Ausloggen des Users am Laufen halten. Zwischen Sensoren und Ausgabeknoten soll ein komplexes Zusammenspiel stattfinden können, das System soll sich gemäß entsprechender Algorithmen einer Komposition für den Ausstellungsraum verhalten (vgl. The McGill Digital Orchestra Malloch (2008)). Dabei muss berücksichtigt werden, dass evtl. große räumliche Distanzen überbrückt werden müssen und evtl. „unsichtbare“ kabellose Kommunikation notwendig ist.

4.1.1 Echtzeitanforderungen

Kontinuierliche Datenströme (z.B. Farbverläufe, Audiodaten) müssen in Echtzeit übertragen werden können. Usereingaben müssen in Echtzeit Reaktionen auslösen können. Sensoren sollten über die Ausgabeknoten spürbares Feedback geben können, um die Interaktion für das Publikum erfahrbar zu machen. Daraus ergeben sich kritische Reaktionszeitvorgaben. Für bewegte Bilder, Farbverläufe in Dot-Matrix-Anordnungen der Light-Motes sind min. 16Hz notwendig bzw. ein Refresh-Zyklus von 62.5ms, typischerweise sogar 24Hz (Filmprojektor), ein kompletter Bildaufbau ca. alle 40ms[12]. Zwischen dem Drücken einer Taste eines Keyboards und dem hörbaren Ton sollten weniger als 10ms Latenz liegen, damit der User keine störende Verzögerung zwischen Aktion und Reaktion wahrnimmt (vgl. Sukale (2004)). Gleiches gilt für interaktive multimediale Ausstellungsobjekte mit Schaltflächen, Drucksensoren, etc.

4.1.2 „Plug and Play“

Um einen einfachen Aufbau des verteilten Systems zu gewährleisten, sollte das Kommunikationsnetzwerk Mechanismen zum Auffinden von Geräten und deren Diensten bereitstellen und automatisch das Hinzufügen und Entfernen von Komponenten organisieren. Der Main-Controller sollte sich über die angeschlossenen Controller, Motes und User-Interfaces informieren und deren Funktionen ggf. anderen Komponenten zur Verfügung stellen können. Das System sollte spontan (*on the fly*, *hotplug*) mit zusätzlichen Geräten erweiterbar sein, Adressen sollten dabei automatisch zugewiesen werden können.

[12] vgl. http://de.wikipedia.org/wiki/Stroboskopeffekt

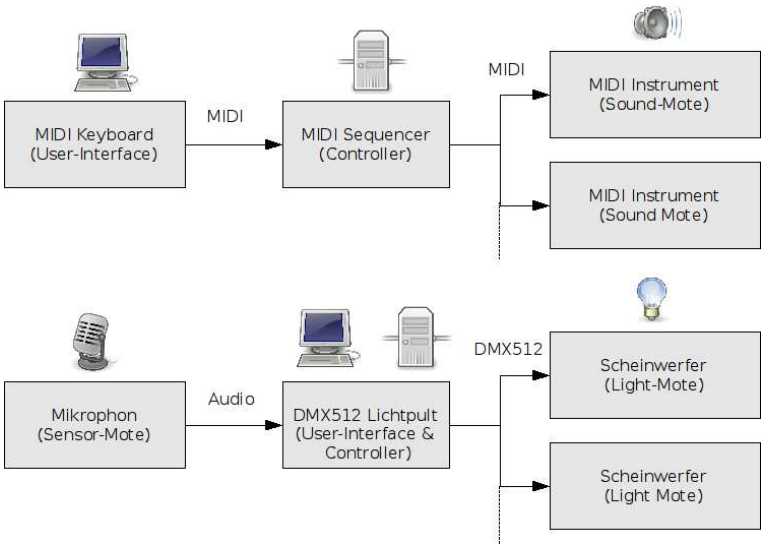

Abbildung 11: Klassische MIDI und DMX512 Netzwerke und Einordnung der Komponenten in das hier entworfene System (User-Interface, Controller, Motes).

4.2 Umsetzung

Klassischerweise würde man im Kontext von Licht- und Audio-Installationen die Übertragungsstandards *MIDI* und *DMX512* einsetzen (vgl. Sukale (2008)). Diese Standards sind offen, sehr robust, einfach zu bedienen und zu implementieren. Ein entscheidender Vorteil ist die weite Verbreitung durch die jahrzehntelange Hersteller-übergreifende Akzeptanz. Ein typischer Aufbau ist in 4.2 dargestellt.

Wie beschrieben können MIDI und DMX512 jedoch im Vergleich zu moderneren Standards als veraltet gelten: Die Protokolle sind relativ unflexibel und Interoperabilität ausserhalb der vorgesehen Einsatzgebiete ist nicht vorgesehen (vgl. Puckette1994). MIDI wurde für Audio und speziell für westliche, chromatisch notierte Musik entwickelt, DMX512 ausschließlich für Licht. Für multimediale interaktive Installationen sind Erweiterungen nötig, um Klänge mit Licht zu verknüpfen und evtl. noch unterschiedliche Sensorwerte einzubeziehen. Um diese Nachteile auszugleichen, wurden in der Vergangenheit viele (meist kommerzielle) Lösungen entwickelt, um die Domänen Audio und Licht über die beiden vorherrschenden Protokollgrenzen hinweg zu verknüpfen: z.B. Sound-to-Light, MIDI-to-DMX512, Controll-Voltage(CV)-to-MIDI, MIDI-to-CV. Diese Speziallösungen sollen hier aber zu Gunsten universellerer Ansätze nicht berücksichtigt werden.

Seit der Einführung von MIDI und DMX512 haben sich neue Netzwerkstandards und Bussysteme etabliert (z.B. Ethernet, WLAN, USB, „Bluetooth“, „Firewire“), die wesentliche Vorteile besitzen und ebenfalls weit verbreitet und günstig zu implementieren sind. Diese können gut in Multimedia-Komponenten integriert werden. So sind mittlerweile z.B. User-Interfaces zum Steuern von Audioanwendungen meist mit einer USB-Schnittstelle ausgerüstet (vgl. Behringer).

Abbildung 12: Behringer BFC2000 Controller mit USB und MIDI Schnittstellen

Hersteller von Veranstaltungs- und speziell Lichttechnik setzen zur Regelung von Scheinwerfern mittlerweile Ethernet-basierte Lösungen ein (vgl. Howell (2007)). Das Ersetzen von Netzwerken für spezielle Aufgaben durch universelle IP-basierte Netzwerke hat viele Vorteile:

- große Bandbreite
- breites Angebot an günstigen Produkten (Hubs, Router, etc.)
- Standard-Infrastruktur (evtl. schon voher am Einsatzort verfügbar, testbar)
- ausfallsicher (z.B. durch Stern-Topologie)
- physikalische Barrieren lassen sich einfach überwinden (WLAN, DSL, „WiMax“, Ethernet-over-Power Line)

es ergeben sich aber im Vergleich von Ethernet zum „low-level“-Protokoll DMX512 auch einige Nachteile:

- geringere Leitungslänge (100m für Ethernet gegenüber 300-1200m für EIA485)
- Stern-Topologie besitzt höheren Verkabelungsaufwand als Bus-Topologie
- Protokoll-Overhead (z.B. relevant bei Mikrocontroller-basierten Geräten)
- Echtzeitverhalten nicht garantiert (evtl. niedrige Verbindungsqualität, viele „Hops“)

Aufgrund der genannten Vorteile soll das hier vorgestellte System weitestgehend auf IP-Netzwerk-Technologien basieren. Gleichzeitig sollen die Vorteile von DMX512 weiterhin nutzbar sein, so dass dieser Standard in einer Nachfolge-Version mit der Erweiterung RDM4.2.5 eingesetzt wird. MIDI wird hier ersetzt und erweitert durch OSC4.2.3, ein Protokoll zur Vernetzung von Anwendungen und Geräten aus dem Multimedia Bereich (Software-Synthesizer, modulare Kompositionssysteme, HID-Controller). Beide basieren meinen Anforderungen nach auf liberalen Lizenzmodellen und sind quelloffen. Details und Spezifikationen zu diesen Standards werden im Folgenden dargestellt. OSC besitzt auf dem Sektor interaktiver, multimedialer Installationen großes Durchsetzungspotential als neuer

Standard. Viele weit entwickelte Softwareprojekte setzen ebenfalls auf OSC (Supercollider, PureData, MaxMSP, Processing).
Es stellt sich weiterhin die Frage, ob es optimal wäre, OSC für sämtliche Kommunikation im Netzwerk zu verwenden, Also zwischen User-Interface und Controller und Controller und Motes? OSC ist nicht abhängig von einer bestimmten physikalischen Netzwerkschicht oder Transport-Schicht (siehe 4.2) wird typischerweise aber via UDP[13] über Ethernet übertragen. User-Interface, Controller und Motes könnten mit Ethernet-Schnittstellen ausgerüstet werden, UDP/IP-Protokollstacks implementieren und so direkt via OSC-Paketen miteinander kommunizieren (Variante 1). Diese Variante bringt aber Nachteile mit sich. Zum einen bei der Kommunikation zwischen User-Interface und Controller:

- die zu verwendenden HTTP-Clients können nicht standardmässig OSC „sprechen“ und es sollen keine speziellen Plug-Ins oder Programme installiert werden müssen.
- OSC wird typischerweise über UDP transportiert, JavaScript als Standardtechnologie auf der Client-Seite kann nicht ohne Umwege direkt OSC-Nachrichten über UDP senden, ebensowenig proprietäre Lösungen wie Flash.

Zum Anderen gibt es Nachteile bei der Kommunikation zwischen Controller und Motes:

- Ethernetschnittstellen auf der Mote-Referenz-Hardware Arduino sind kostspieliger als die bereits vorhandene serielle Schnittstelle oder eine USB-Schnittstelle
- DMX512-Bussysteme zwischen den Motes sind einfacher aufzubauen und verlässlicher bzgl. Echtzeitkriterien
- DMX512 ist effizienter als OSC auf low-level Ebene bzgl. Rechenaufwand
- Stromversorgung der Motes über Ethernet (IEEE 802.3af) ist nicht sehr weit verbreitet und damit teuer und kompliziert z.B. im Gegensatz zu USB-Bussystemen mit integrierter Stromverteilung.

Diese Nachteile führen zu zwei weiteren Ansätzen die Datenübertragung im System aufzubauen (Variante 2 und 3): Weiterhin DMX512/RDM verwenden zur Vernetzung von Light-Motes und Sensor-Motes. HTTP-basierte Kommunikation zwischen User-Interface und Controller und nur die Kommunikation zwischen verteilten Controllern via OSC (siehe Abbildung).

Ich entscheide mich hier für eine Umsetzung der dritten Variante (siehe 14). Diese macht zwar die Entwicklung von *Gateway*-Lösungen auf Controller-Seite notwendig, (OSC-to-DMX512/RDM, HTTP-to-OSC) erlaubt aber den Einsatz von Standardtechnologien bei der Implementation des User-Interfaces. Die Verwendung von handelsüblichen DMX512-Geräten als Light-Motes, mit allen Vorteilen des schnellen und robusten

[13] User Datagram Protocol, Netzwerkprotokoll der Transportschicht, im Gegensatz zu TCP verbindungslos und ungesichert, d.h. Pakete können einfach und schnell gesendet werden, es gibt aber keine Mechanismen zur Empfangsbestätigung und Einhaltung der Paketreihenfolge

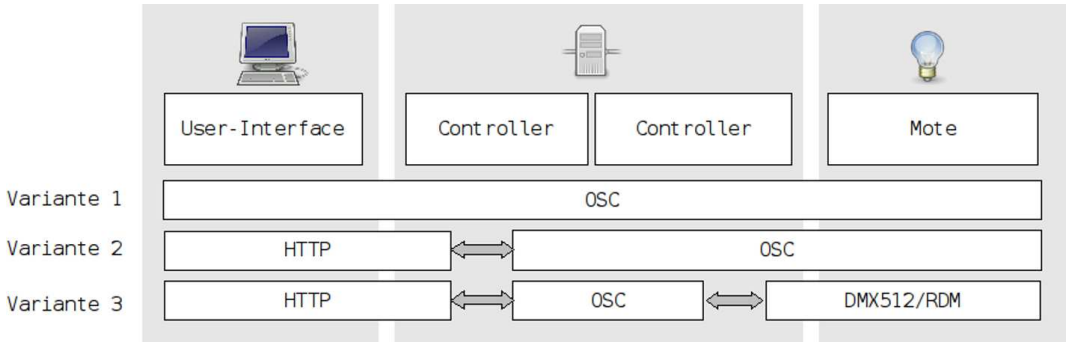

Abbildung 13: Verschiedene Varianten des Netzwerkaufbaus. Pfeile stellen Gateways zwischen den verschiedenen Protokollen dar.

OSI-Modell	DoD-Modell	Userinterface	Controller	Motes
Application	Application	HTTP	OSC	
Presentation				
Session				
Transport	Transport	TCP	UDP	
Network	Internet	IP		
Data Link	Network Access			DMX512/RDM
Physical		802.3 / 802.11 (Ethernet, WLAN)		EIA-485

Abbildung 14: Einordnung der Module in Netzwerk Schichtmodelle

DMX512-Bussystems bleibt möglich.

Es bleibt im Folgenden abzuwägen, in wie weit das OSC-Protokoll in andere Protokolle „gewrappt“ werden sollte, und in wie weit DMX512/RDM-Pakete in OSC-Nachrichten verpackt werden sollten. Aus OSC Perspektive sollten Motes und User-Interface wie OSC-Server und Clients angesehen werden. Beispielsweise sollten Light-Motes direkt via OSC vom Controller als OSC-Server angesprochen werden können, und OSC-Methoden zur Steuerung von Licht und Farbe zur Verfügung stellen. Sensor-Motes sollen als OSC-Server gepollt werden können. Das User-Interface tritt als OSC-Client auf, wenn Methoden aufgerufen werden.

4.2.1 Verteilung, Transparenz

Durch den modularen Aufbau, die Trennung von User-Interface, Controller und Motes sowie durch die Kommunikation via IP sind unterschiedliche räumliche Verteilungen des Systems möglich. Im einfachsten Fall handelt es sich um ein *Personal* oder *Desktop Area Network* z.B. als Aufbau im Atelier oder Arbeitsraum beim Entwurf und Test von Konzepten. Sämtliche Dienste des User-Interface und der Controller würden auf unterschiedlichen Ports auf einem lokalen Rechner angeboten werden. Die Übertragungsprotokolle dienen so-

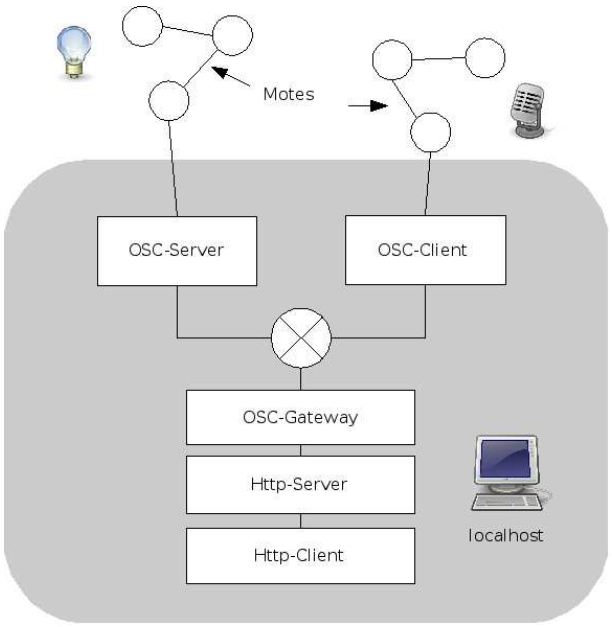

Abbildung 15: Netzwerkaufbau mit einem Entwicklungsrechner (localhost) und angeschlossenen Motes

mit eher als *IPC*[14]-Mechanismen (siehe 15). Mit zunehmender Komplexität der Installation können weitere Komponenten hinzugefügt werden, z.B. in einem großen Ausstellungsraum mit vielen Motes. Funktionen (z.B. Gateways zu Aktoren und Sensoren) können im *LAN*[15] auf verschiedene Controller verteilt werden. User-Interface und Controller können entkoppelt werden. Es ist dann im laufenden Betrieb nicht mehr notwendig, ein User-Interface anzuschließen (*Stand-Alone*-Modus). Dieses kann spontan ins Netzwerk eingefügt werden. Dabei bleibt das System für den Anwender transparent, d.h. es spielt keine Rolle, ob sich Ressourcen lokal oder auf entfernten Rechnern befinden (*Ortstransparenz* und *Persistenztransparenz*). Zudem sorgt die OSC-Spezifikation (4.2.3) für einen hohen Transparenzgrad im Netzwerk, da auf OSC-Ebene von Unterschieden in der lokalen Repräsentation der Daten abstrahiert wird (*Zugriffstransparenz*).

Auch eine Verteilung im *WAN*[16] ist denkbar (siehe 16): Teile des Systems können über Internet verbunden sein z.B. über feste öffentliche IP-Adressen oder Netzwerktunnel. So kann das User-Interface über HTTP von einem entfernten Rechner aus gestartet werden (z.B. zum Überwachen des Ausstellungsbetriebs). Global verteilte kollaborative Arbeiten sind möglich (z.B. Sensoren an einem entfernten Ort können Aktionen im lokalen Ausstellungsraum auslösen oder sog. „Online Jamming“).

[14] Inter Process Communication
[15] Local Area Network
[16] Wide Area Network

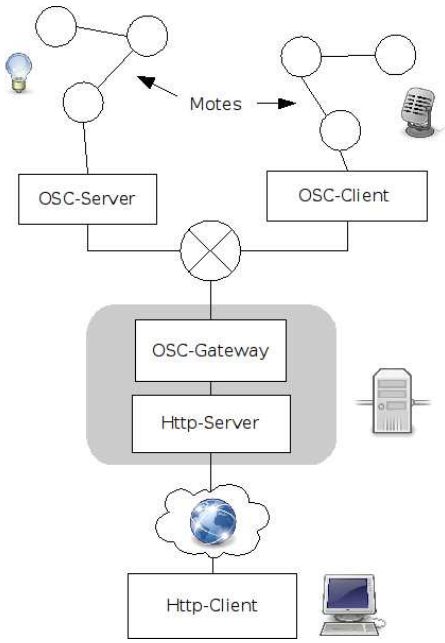

Abbildung 16: Verteiltes System im WAN, Netzwerkaufbau mit mehreren Controllern und externem User-Interface-Rechner sowie angeschlossenen Motes

4.2.2 Topologie

Es stellt sich die Frage, wie das Netzwerk aufgebaut werden sollte. Eine sternförmige zentralisierte Topologie oder ein *Peer-2-Peer* Aufbau? Bezüglich Echtzeitverhalten, Effizienz und Flexibilität stellt eine Peer-2-Peer-Variante sicher das Optimum dar: Motes und Controller können direkt miteinander kommunizieren: Ein Sensor-Mote taucht im Netzwerk auf, ein ästhetisch ausgeklügelter Algorithmus oder eine Komposition sieht vor, diesen aufgrund seiner besonderen Eigenschaften mit einem Ausgabeknoten zu verbinden. Der Sensor-Mote sendet seine gesammelten Werte z.B. direkt an ein Light-Mote, wo diese in Farbe oder Lichtintensität umgewandelt werden. Nur: wie machen sich die beiden Knoten bekannt, wo findet eine evtl. notwendige Datenverarbeitung (Skalierung, Konvertierung) statt? Wer wacht über das Mapping der Komponenten, wo wird das Verhalten der Teilnehmer festgelegt?

Letztere Fragen lassen eine zentralisierte Lösung als geeigneter erscheinen: es gibt einen *Main-Controller* bzw. *Router*, der die Nachrichten aufgrund einer *Mappingtabelle* routet, bei dem alle Motes und Controller sich anmelden und ihre Dienste bekannt geben. An diesem zentralen Punkt kann sich ein User-Interface-Client einloggen und Modifikationen am System vornehmen oder Status-Informationen abfragen. Der Main-Controller kann sich ebenfalls um die Vorverarbeitung von Daten kümmern.

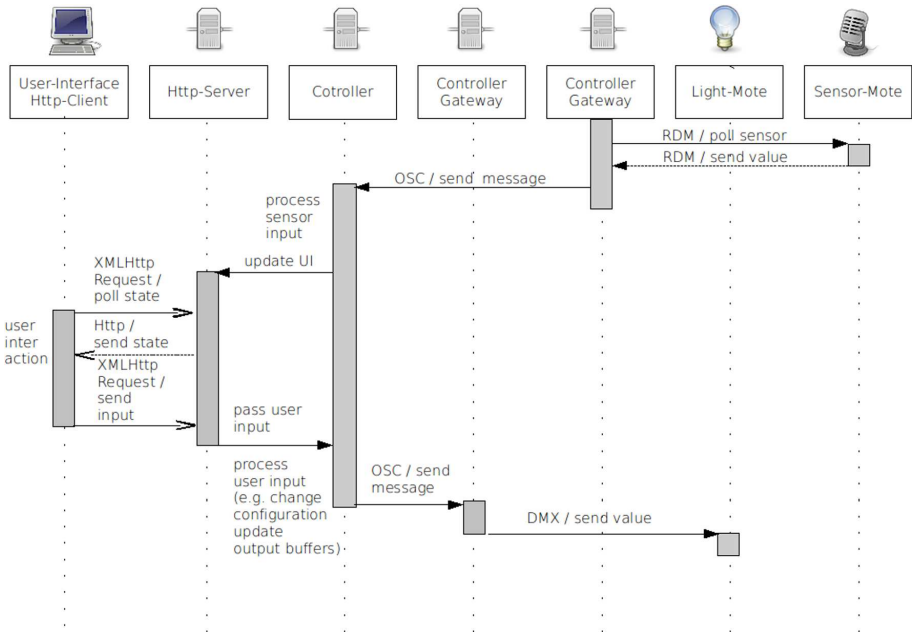

Abbildung 17: Sequenzdiagramm einer typischen interaktiven Installation

Eine kanadische Forschungsgruppe um das *McGill-Digital-Orchestra* stand vor einem ähnlichen Problem und schlägt ein Netzwerkaufbau vor, der beide Varianten (18, 19) ermöglicht, je nachdem, ob ein zentraler oder mehrere Router eingesetzt werden (siehe auch 4.2.7).

4.2.3 OSC

Open Sound Control (OSC) ist ein Kommunikationsprotokoll für Computer, Klangerzeuger, *Gestural Controller*[17] und andere Multimediageräte, optimiert für den Einsatz in modernen IP-Netzwerken. Der Fokus liegt auf Echtzeitverarbeitung von Signalen in interaktiven multimedialen Kontexten. OSC ist bereits weit verbreitet und wird in neueren Eingabegeräten für musikalische Aufführungen (z.B. *Lemur* [18], *monome* [19]), beim Musizieren über verteilte Systeme in lokalen und globalen Netzwerken sowie zur Interprozesskommunikation innerhalb von Anwendungen (*Supercollider*) eingesetzt. OSC wurde 1997 am CNMAT[20] von Matthew Wright und Adrian Freed als ein potentieller Nachfolger des MIDI-Standards vorgestellt und liegt derzeit in der Version 1.0 vor (siehe Wright (2002)).

[17] Geräte zur Erfassung von Gesten, Begriff wird im Zusammenhang von Eingabegeräten für elektronische Musikinstrumente verwendet

[18] http://www.jazzmutant.com/

[19] http://monome.org/

[20] Center for New Music and Audio, Univ. California, Berkeley, USA

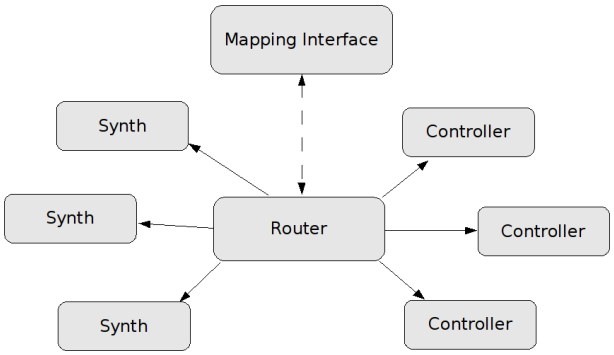

Abbildung 18: McGill Digital Orchestra: sternförmige Topologie. Hier werden Controller und Synthesizer über einen zentralen Router verbunden. Verknüpfungen werden über ein Mapping-Interface eingegeben. Im Kontext dieser Arbeit wären Synths Light-Motes, Controller Sensor-Motes und das Mapping Interface allgemein ein User-Interface.

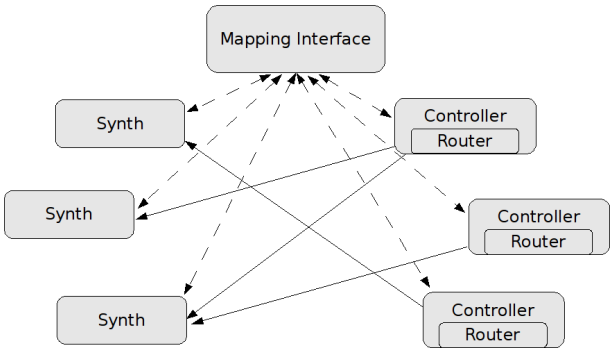

Abbildung 19: McGill Digital Orchestra: dezentrale Topologie. Durch das Einbetten von Routern in die einzelnen Controller (bzw. Motes) ist ein echtes Peer-2-Peer Netzwerk möglich. Das Mapping Interface kommuniziert über einen sog. ”Admin Bus” via Multicast UDP Nachrichten mit den Netzteilnehmern

OSC ist quelloffen und wurde bereits auf zahlreiche Architekturen portiert. Es existieren Plug-Ins und Bibliotheken u.A. für C (*liblo*),C++ (*WOscLib*), Java, für MAX/MSP, PureData, Reaktor[21], SuperCollider, Adobe Flash (flosc), Processing etc. (vgl. Wright (2003) und Schmeder (2008a)) OSC ist unabhängig von einer speziellen Transport-Schicht (obwohl hauptsächlich UDP verwendet wird).

Paketaufbau OSC ist Paket-orientiert: die Übertragungseinheit bei OSC ist ein *OSC-Paket*, bestehend aus einem kontinuierlichen Block Binärdaten und entsprechender Längenangabe in 8-Bit Bytes. Die Länge eines Pakets ist dabei immer ein Vielfaches von 4. Empfänger von OSC-Paketen werden laut Spezifikation als *OSC-Server*, Sender von OSC-Paketen werden als *OSC-Client* bezeichnet. [22]

Ein OSC-Paket besteht aus *OSC-Nachrichten* oder *OSC-Bundles*, wobei das erste Byte eines Pakets angibt, worum es sich handelt.

OSC-Nachrichten bestehen aus einem *Adressmuster* und einem *Type-Tag-String* (zur Angabe der Typen der folgenden Argumente) gefolgt von einer Reihe kontinuierlicher Daten (der binären Repräsentation der zu übertragenden Argumente).

Nachrichten können in sog. *OSC-Bundles* gepackt werden, beginnend mit der Zeichenkette `#bundle` und einem *OSC-Time-Tag* gefolgt von den eigentlichen Inhalten. Dabei können OSC-Bundles rekursiv auch weitere OSC-Bundles beinhalten. Bundles werden verwendet um mehrere durch OSC-Nachrichten gesteuerte Ereignisse zeitgleich auszulösen.

Jeder OSC-Server stellt einen Satz von *OSC-Methoden* zur Verfügung, die über entsprechend adressierte OSC-Nachrichten aufgerufen werden können (analog zu RPC [23]).

Adressierung Die Methoden eines OSC-Servers sind in einer hierarchischen Baumstruktur angeordnet, dem *OSC-Adressraum*. Endknoten bzw. „Blätter“ stellen dabei die Methoden dar, während Zweige als sog. *OSC-Container* bezeichnet werden. Die Struktur ist dynamisch und kann über die Zeit verändert werden. Knoten im Netz können vollkommen frei und intuitiv Namen zugewiesen werden. Somit sind im Gegensatz zu MIDI selbstbeschreibende und menschenlesbare Namen möglich. Im Gegensatz zu MIDI und DMX512 ist dieser Namensraum nicht auf eine maximale Anzahl von Teilnehmern begrenzt. Die Adressierung basiert auf einer „slash“-Notation ähnlich zu URLs im Internet, Xpath oder Unix-Filesystemen:

```
/container/container/methode
```

[21] erfolgreiche, kommerzielle Software zur virtuellen Klangsynthese

[22] Wobei die Begriffe OSC-Server und OSC-Client nicht korrekt definiert werden, da der Server dem Client auf Anfrage anwortet

[23] Remote Procedure Call

OSC verwendet dabei ebenfalls einen Pattern-Matching-Algorithmus bei der Auflösung von Adressen: Eine Adressierung mit regulären Ausdrücken ist möglich, so dass ein ganzer Bereich des Namensraumes mit einer Nachricht angesprochen werden kann. Dieser basiert auf den UNIX Kommandozeilen Pattern Matching Strategien:

```
/container/*/methode
```

Typisierung OSC Daten setzen sich zusammen aus einer Auswahl atomarer Datentypen mit einer Datenlänge von 32 Bit oder einem Vielfachen davon.

- **int32** 32-Bit Big-Endian Zweierkomplement Integer
- **float32** 32-Bit Big-Endian Fliesskomma-Zahl nach IEEE 754
- **OSC-string** Eine ASCII-Zeichenkette ohne Null, mit terminierender Null gefolgt von evtl. notwendigen „Zero-Padding“-Zeichen um auf ein Vielfaches von 32-Bit zu kommen.
- **OSC-timetag** 64bit Festkomma-Zahl nach dem Internet NTP Zeitstempel Standard (siehe Mills (1992))
- **OSC-blob** int32 Längeangabe gefolgt von einer entsprechenden Anzahl beliebiger 8-Bit Bytes plus zusätzlichen 0-3 Zero-Padding Bytes um auf ein Vielfaches von 32-Bit zu kommen.

Zur Deklaration von Typen stellt OSC Type Tags zur Verfügung:

OSC Type Tag	Typ
i	int32
f	float32
s	OSC-string
t	OSC-timetag
b	OSC-blob

Diese werden der Adresse nachgestellt, z.B.:

```
/container/methode ,if 255 0.0016
```

Zusätzlich zu den grundlegenden Typen und Type-Tags bestehen weitere nicht standardisierte Typen, z.B. für in OSC verpackte MIDI-Nachrichten, RGBA-Farbwerte, boolean Typen und Arrays.

Einschränkungen

- Der OSC-Adressraum ist nicht standardisiert (obwohl es Versuche gibt dieses zu tun: z.B. über Publikation in http://opensoundcontrol.org/schema, in Ehrentraud (2007), oder als *GDIF* Jensenius (2006)). Anwendungen unterschiedlicher Hersteller benutzen zwar das gleiche Adressierungsschema, aber es bleibt unklar, was ein Endknoten für Methoden zur Verfügung stellt. In komplexeren Installationen wird dies zu einem Problem (vgl. Place (2008)).
- Derzeit gibt es kein standardisiertes Query System für OSC-Netze (s.u.)
- Die Kommunikations-Bandbreite von OSC z.B. im Vergleich zu MIDI ist höher (mehr Overhead)
- Verarbeitungs-Bandbreite von OSC Nachrichten ist durch das String-basierte Design (z.B. im Gegensatz zu DMX512) wesentlich höher (relativ aufwendige Parser müssen eingesetzt werden).
- OSC verwendet keine Mapping-Strategien von dem menschenlesbaren Adressierungsschema auf computerlesbarere Formate (wie z.B. IP oder ARP [24]), die Verarbeitung durch Maschinen ist daher weniger effizient (vgl. Fraietta (2008)).

Embedded OSC Es ist aufgrund der o.g. Einschränkungen umstritten, ob OSC geeignet ist für eingebette Geräte. Dennoch gibt es zumindest zwei bekannte „Proof Of Concepts“ für den Einsatz von OSC auf Mikrocontroller-basierten Systemen. Zum einen werden die Methoden des ARM-basierten Make-Controllers über OSC angesprochen [25], zum Anderen gibt es uOSC:

uOSC - OSC auf eingebetten Systemen Für eingebette System z.B. auf *Microchip-PIC*-Basis besteht eine Referenz-Implementation des OSC-Protokolls (siehe Schmeder (2008b)). Diese wird derzeit aktiv am CNMAT entwickelt und soll dort auch auf die hier verwendete Arduino-Plattform portiert werden. Die Kommunikation zwischen den Geräten wird nicht wie üblich über UDP abgewickelt, sondern über USB bzw. eine serielle Schnittstelle und *SLIP* [26]. uOSC würde es möglich machen, Motes direkt über OSC anzusprechen, so dass keine Umsetzung von OSC zu DMX512 mehr notwendig ist und somit Schwierigkeiten bei der Protokollübersetzung vermieden werden. Nachteil ist allerdings, dass Standard-Geräte mit DMX-Schnittstelle nicht mehr kombinierbar sind mit den neuen Geräten und dass ein neues Bussystem (auch wenn dieses z.B. einfach mit USB-Hubs aufzubauen wäre) konstruiert werden müsste. Eine Lösung wäre z.B. OSC über DMX512/EIA-485 zu versenden, dadurch würde jedoch Overhead enstehen, der die Vorteile von DMX512 (einfach, schnell) zu nichte machen würde.

[24] Address Resolution Protocol

[25] http://www.makingthings.com/documentation/tutorial/osc/overview-and-concepts

[26] Serial Line Internet Protocol

Abbildung 20: monome 40h von Brian Crabtree und Kelli Cain.

Namensraum Beispiel Exemplarisch soll hier ein Blick auf den OSC-Namensraum des *monome*-Hardware-Controllers geworfen werden. Die Anwendung ist ähnlich zu den einleitenden Beispielszenarien und der Zusammenhang zwischen Hardware- und Softwareobjekten ist hier in eleganter Weise moduliert. Der monome-Namespace benennt explizit die Features des Moduls, das System wird so selbstdokumentierend und transparent in seiner Funktionalität. Der monome 40h besteht aus einer 8x8-Button-Matrix mit individuell schaltbarer LED-Hintergrundbeleuchtung (siehe 20). Der monome wird wie die Arduino-Boards via USB-to-seriell-Konverter (von FTDI) mit einem PC verbunden. Hier wird das serielle Signal mit spezieller Gateway-Software an ein OSC-Netzwerk angebunden. Als root-Container dient das Gerät selbst:

```
/40h
```

Die LEDs der Schalter sind einzeln ansprechbar:

```
/40h/led
```

oder spalten- und zeilenweise:

```
/40h/led_col
/40h/led_row
```

oder als komplette matrix:

```
/40h/frame
```

Einstellungen am System werden an

```
/sys
```

adressiert. Beispielsweise:

```
/sys/type
returns the device type (40h, 64, 256, etc.)
```

OSC-Methoden Aufrufe zum ein- oder ausschalten von LEDs werden z.B. wie folgt definiert:

```
/40h/led [x] [y] [state]
state: 1 (on) or 0 (off)
```

Beispiel: LED an Stelle [0,5] einschalten

```
/40h/led 0 5 1
```

monome verzichtet in dieser Version auf die explizite Angabe von Type Tags, korrekterweise sollte es also `/40h/led ,iii 0 5 1` lauten.

Namensraum Entwurf Auch hier sollen alle Einheiten des Systems über OSC angesprochen werden können. Das bedeutet jeder Controller, der User-Interface-Server und sämtliche Motes müssen über einen eindeutigen OSC-Pfad erreichbar sein. Dabei muss berücksichtigt werden, dass evtl. mehrere Komponenten auf einem Rechner unter der gleichen IP-Adresse laufen und dass das User-Interface evtl. ebenfalls auf einem Rechner läuft, der auch gleichzeitig Controller ist. Die Kollisionsvermeidung wird hier aber an das darunterliegende Zeroconf- Protokoll deligiert (siehe 6.2.3), so dass die einzelnen Container in ihrer physikalischen Ausprägung auch im Pfad moduliert werden können. Es gibt demnach:

```
/mote
/control
/ui
```

Es stellt sich die Frage, ob schon auf dieser root-Ebene direkt die Ausprägungen der einzelnen Motes moduliert werden sollten:

```
/lightmote
/sensormote
/standalonecontroller
/pixel
/smartpixel
/RGBpixel
/dimmer
```

Ich finde diesen Ansatz zwar „selbstbeschreibender“ aber unstrukturiert. Zudem muss berücksichtig werden, dass sehr viele Motes im System existieren, die individuell angesprochen werden müssen. Alternativ wären folgende Lösungen für ein Lightmote möglich:

```
/lightmote/1
/lightmote1/
/mote1/light
/mote/1/light
```

Leider gibt es hier keine Konvention, wie flach ein solcher Baum sein sollte, und wie mehrere Ausprägungen eines Containers differenziert werden sollten. Es lohnt der Blick auf das Adressierungsschema der OSC Referenz-Implementation uOSC (siehe 4.2.3). Die verwendete Hardwareplattform bietet ähnliche Funktionen wie die Arduino-Plattform auf der die Lightmotes und Sensormotes basieren. Hier gibt es ebenfalls PWM-Ausgänge die folgendermaßen angesprochen werden:

```
/pwm/0 ,i [val]
/pwm/1 ,i [val]
```

Prinzipiell wird von Seiten der OSC-Entwickler ein Namensraum Design in Anlehnung an die REST-Architektur [27] vorgeschlagen. Verallgemeinert ergibt sich in etwa folgende Organisation:

```
/devices/{device}/{properties,methods}/{...}
```

Konkret ist beispielsweise der Namensraum des ARM-basierten Make-Controllers [28] in dieser Weise gestaltet:

```
/subsystem/device/property argument
```

Nach diesen Vorbildern ergibt sich folgender Namensraum für das hier vorgestellte System: Für die Light-Motes:

```
/motes/{mote_id}/rgb ,iii [r] [g] [b]
/motes/{mote_id}/pwm/{pwm_chn} ,i [val]
/motes/{mote_id}/intensity ,i [val]
```

Für die Sensor-Motes:

```
/motes/{mote_id}/analogin ,i [pin]
/motes/{mote_id}/digitalin ,i [pin]
```

Auch wenn dieser Ansatz in Absprache mit OSC-Entwicklern und im Vergleich mit Namensräumen anderer Projekte als „Best Practice“ angesehen werden kann, halte ich eine einfachere „flachere“ Lösung für besser:

[27] REpresentational State Transfer, Architektur Stil der z.B. auch den Aufbau des WWW beschreibt, Hauptmerkmal ist die Trennung von Ressourcen und deren Repräsentation, wobei Ressourcen typischerweise über URIs dargestellt werden und Repräsentationen in Form von HTML oder XML Dokumenten vorliegen. Ein Aspekt von REST ist die Zustandslosigkeit: Anfragen des Clients an den Server sind nicht von vorangegangenen Anfragen oder zwischengespeicherten Daten abhängig

[28] http://www.makingthings.com/documentation/tutorial/osc/overview-and-concepts

```
/sensor/{id}/{methods}
/pixel/{id}/{methods}
```

Hier ist gleich zu erkennen, um was für ein Mote-Typ es sich handelt. IDs können z.B. in der Discovery-Phase des Systems vom Controller oder von der Mote-Hardware vorgegeben werden (z.B. RDM-Device-ID). Implementiert ein Endknoten mehrere Funktionalitäten (z.B. zwei RGB-Pixel-Ausgänge und ein Sensor-Eingang), so werden diese als eigenständige OSC-Container dargestellt:

```
/pixel/0/{methods}
/pixel/1/{methods}
/sensor/0/{methods}
```

In eingebetteten System würde das Parsen von OSC-Adressen einfacher sein, wenn diese aus weniger Zeichen aufgebaut wären und Namen die gleiche Zeichenlänge hätten. Man könnte also die Konvention einführen, dass Namen aus nur drei Zeichen bestehen dürfen. Dadurch verliert man jedoch den Vorteil von OSC der Menschenlesbarkeit und der Selbstbeschreibung:

```
/pix/000/rgb
/pix/001/rgb
/sns/000/val
```

Für Folgeprojekte unter Verwendung von Arduinos (siehe 7.1) empfehle ich einen universelleren Namesraum zum ansprechen der kompletten Hardware:

```
/arduino/digitalout/[1-n]
/arduino/digitalin/[1-n]
/arduino/analogout/[1-6]
/arduino/analogin/[1-8]
```

Namesraum Abfrage Um den Namensraum eines OSC-Pfades oder Unterpfades zu entdecken, sollen OSC-Server zusätzlich auf Nachrichten antworten, die mit einem / enden und den darunterliegenden Namensraum zurückgeben. Dies entspricht in etwa der üblichen Vorgehensweise anderer Projekte, und wird sich evtl. als Standard durchsetzen. Ein Controller würde z.B. auf die Nachricht

```
/
```

mit dem kompletten Adressraum antworten:

```
/pixel
/pixel/0
/pixel/0/rgb
/pixel/1
```

```
/pixel/1/rgb
/sensor
/sensor/0
/sensor/0/analogin
[..]
```

Ein Smart-Pixel würde z.B. auf die Nachricht

```
/pixel/0/
```

mit dem Teilpfad

```
/rgb
```

antworten.

4.2.4 DMX512

DMX512 ist der de facto Kommunikationsstandard im Lichttechnik-Bereich (vgl. Sukale (2008)) und soll auch hier aufgrund der o.g. Vorteile für die Kommunikation zwischen Controllern und Motes verwendet werden.

DMX512 ist ein einfaches serielles Datenprotokoll:

- 250kbps
- 8-Bit Daten
- keine Parität
- 2 Stopbits
- asynchron

Adressierung Mit DMX512 können 512 statische Adressen angesprochen werden. Als Erweiterung gibt es 16 DMX-Universen mit je 512-Adressen.

Device Discovery DMX512 ist ein *unidirektionales* Protokoll. Es gibt keine Möglichkeit Geräte zu lokalisieren. Geräte und deren Adressen und Funktionen müssen im Voraus bekannt sein.

Physikalische Schicht DMX512 wird über EIA-485-Schnittstellen und Leitungen übertragen (wie z.B. CAN, USB). Der EIA-485 Standard beinhaltet *symmetrische Datenübertragung*, ist somit störungsunempfindlich und erlaubt bis zu 1200m lange Leitungen. Maximal 32 DMX512-Geräte können an einen Bus angehängt werden.

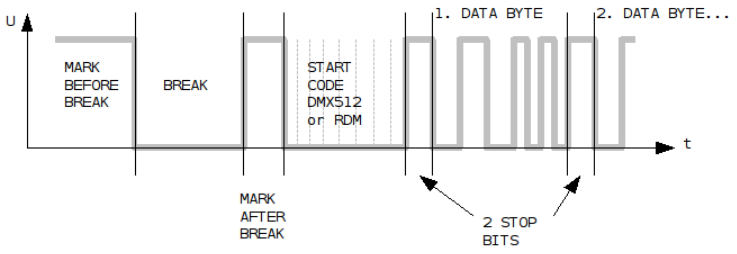

Abbildung 21: DMX512 / RDM Frame

Datenformat Nach einem *Break* und einem DMX512-*Startcode* werden die Nutzdaten in Serie übertragen. Jeder Empfänger liest die nacheinander eintreffenden Bytes eines Frames und erhöht pro Byte einen internen Zähler (siehe 47). Stimmen Zählerwert und Startadresse des Empfängers überein, „pickt“ sich dieser die folgenden Zeichen zur weiteren Verarbeitung heraus. Es werden maximal 512 Bytes übertragen, danach warten sämtliche Empfänger auf ein neues Break.

Echtzeitverhalten DMX512 ist geeignet für den Einsatz in zeitkritischen Systemen. Durch die festgelegte maximale Framelänge ist eine Aktualisierung eines kompletten DMX-Universums innerhalb von maximal 40ms garantiert. Dadurch sind z.B. Dot-Matrix-Anwendungen mit laufenden Bildfolgen, Videowände möglich, wobei eine Bildfrequenz von 25Hz erzielbar ist (entsprechend der üblichen Framerate gängiger Videocodecs). Die Framerate kann noch verbessert werden, wenn nur Werte bis zum letzten zu aktualisierenden Kanal übertragen werden, und der Frame danach abgebrochen wird. So kann die Framelänge verkürzt und somit die Wiederholfrequenz gesteigert werden.

Einschränkungen DMX512 sieht keinerlei Möglichkeiten vor zur automatischen oder dynamischen Adressierung von Empfängern. Ebenso sind keine Device Discovery oder Service Discovery Methoden möglich. Die Datenbreite ist beschränkt auf 8-Bit (ausser man verwendet zwei Kanäle für doppelte Auflösung). Es können nur 512 Empfänger angesprochen werden, was bei hoher Granularität des Systems z.B. bei einer Dot-Matrix-Anzeige aus einzelnen DMX512-Geräten bzw. Light-Motes schnell an die Grenzen des Protkolls stösst. Es gibt zudem keine Daten-Sicherungsschicht.

4.2.5 RDM

Das *Remote Device Management* (RDM) Protokoll wurde 2006 von der ESTA [29] als ANSI Norm ausgegeben (vgl. ESTA (2006)) und stellt eine Erweiterung des DMX512 Protokolls dar, welches zwar einfach zu implementieren ist, aber auch mit zunehmender „Intelligenz“

[29] Entertainment Services and Technology Association, New York, USA

von Beleuchtungstechnik viele Unzulänglichkeiten aufweist (s.o.). RDM ermöglicht halbduplex bidirektionale Kommunikation und macht so folgende Funktionen möglich:

- **Device Discovery** (”Gerätefindung”)
- **„In-System“-Konfiguration** (z.B. Geräteprofile auswählen)
- **Monitoring** (z.B. Temperatursensoren auslesen)
- **Verwaltung** (z.B. dynamische Adressierung)
- **Broadcast Nachrichten**

RDM ist abwärtskompatibel zu DMX512, d.h. Geräte die nicht RDM unterstützen, können an einem RDM-Bus wie normale DMX512-Geräte betrieben werden. Betrieb von RDM-Geräten über DMX512 Infrastruktur ist ebenfalls möglich, wenn Splitter, Repeater und Router (”Inline-Devices”) RDM kompatibel sind. Timingspezifikationen sind so gewählt, dass sie nicht mit dem DMX512 Standard kollidieren.

Adressierung Während bei DMX512 nur 512 feste Adressen pro ”Universum” vergeben werden können, stellt RDM eine eindeutige 48bit-breite ID zur Verfügung, unterteilt in 16bit Hersteller-ID und 32bit Geräte-ID. Diese sollte (ähnlich wie MAC-Adressen) für jedes hergestellte Geräte eindeutig sein, damit niemals zwei gleiche IDs in einem System auftauchen. Die Hersteller-IDs werden von der *Control Protocols Working Group* der ESTA vergeben. Für die in dieser Arbeit vorgestellten Motes habe ich die ID `0x5555` beantragt und zugesprochen bekommen, so dass ab jetzt die Geräte als „echte“ RDM-Geräte freigegeben werden können.

Nachrichtenformat Ein RDM-Datenframe baut auf dem DMX512-Format auf, nur dass hier ein anderer Startcode (`0xCC` + Substartcode: `0x01`) verwendet wird und nachfolgend RDM-Daten im Paketform und nicht seriell Kanalwerte übertragen werden. Ein RDM Paket ist wie abgebildet aufgebaut (siehe 22)

Device Discovery Zu Beginn einer Session werden alle RDM-Geräte (*Responder*) im Bus vom RDM-Controller über eine Discovery-Funktion ermittelt: Dazu verschickt der Controller eine Discovery-Nachricht an alle Busteilnehmer (Broadcast) mit der Unter- und Obergrenze des UID-Adressbereiches als Nutzdaten. Liegt die UID eines Responders innerhalb dieses Bereiches, so antwortet er mit seiner verschlüsselten UID. Jeder gefundene Responder wird vom Controller anschließend gemuted. Als Algorithmus wird von der ESTA eine Binärbaumsuche vorgeschlagen (Hölscher (2008)). Diese kann hier aber vereinfacht werden, wenn davon ausgegangen wird, dass nur Motes mit meiner Hersteller-ID und einer maximalen Geräte-ID eingesetzt werden.

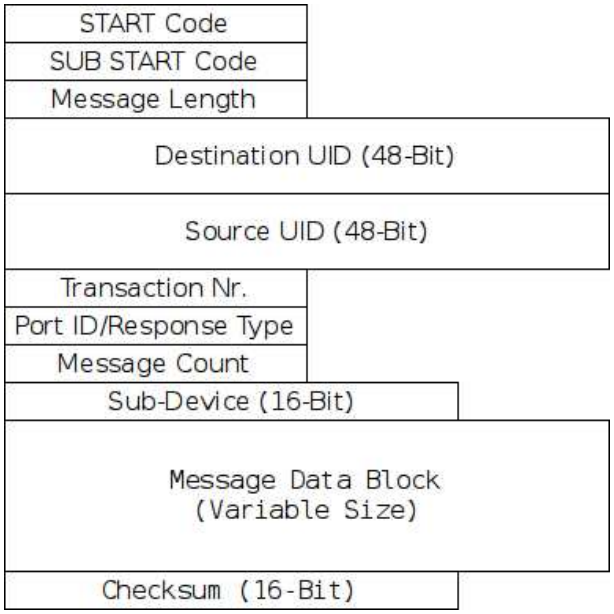

Abbildung 22: RDM Paketformat

Minimale RDM Implementation in Motes Die Firma Litespeed Design publiziert[30] zur freien Verwendung ein Header-File zum erstellen von RDM-Paketen. Weiterhin wird von Hendrik Hölscher [31] eine DMX512 und RDM Bibliothek für Atmel-Mikrocontroller unter der GPL vertrieben, die zur Umsetzung in Open Source Projekten verwendet werden kann. Beide definieren nicht den kompletten Code-Umfang der RDM Spezifikation. Diese Quellcodes sind aber geeignet, um daraus einen minimalen Funktionumfang zu extrahieren und auf die im weiteren vorgestellten offene Plattformen zu portieren. Neben den allgemein als notwendig eingestuften Codes (*Start-Codes, Command Classes, Network Management IDs, Versionsnummer,* etc.) verwende ich spezifische Codes für (LED-)Dimmer und Sensoren. Dies sind Produkt-Kategorie Definitionen, Sensor Typen und DMX512-Slot Definitionen bzw. Beschreibungen der ansprechbaren Kanäle. RDM offenbart hier seinen starren Charakter gegenüber OSC: es werden nur konventionelle Bühnentechnikgeräte definiert.

Für die Motes benötige ich folgende Auswahl an Codes, die ich den quelloffenen Header-Files hinzufüge (komplette Quellcodes im Anhang oder über meine Internetseite):

```
/*****************************************************/
/* Table A-5: Product Category Defines               */
/*****************************************************/

/* Intensity Control (specifically Dimming equipment) */
#define E120_PRODUCT_CATEGORY_DIMMER   0x0500 /* No Fine Category */

/* Miscellaneous */
#define E120_PRODUCT_CATEGORY_OTHER 0x7FFF /* For devices that aren't described within this table. */
```

[30] http://www.rdmprotocol.org

[31] http://www.hoelscher-hi.de/hendrik/light/profile.htm

```
/*****************************************************/
/* Table A-12: Sensor Type Defines                   */
/*****************************************************/
#define E120_SENS_VOLTAGE                             0x01
#define E120_SENS_POSITION_X                          0x12   /* E.g.: Lamp position on Truss */
#define E120_SENS_POSITION_Y                          0x13
#define E120_SENS_POSITION_Z                          0x14
#define E120_SENS_CONTACTS                            0x1C   /* digital input */

/*****************************************************/
/* Table C-1: Slot ID Type Value Description         */
/*****************************************************/

#define E120_ST_PRIMARY   0x00  /* Slot directly controls parameter (represents Coarse for 16-bit parameters) */

/*****************************************************/
/* Table C-2: Slot ID Definitions                    */
/*****************************************************/

#define E120_SD_COLOR_ADD_RED    0x0205 /* Additive Color Mixer - Red */
#define E120_SD_COLOR_ADD_GREEN  0x0206 /* Additive Color Mixer - Green */
#define E120_SD_COLOR_ADD_BLUE   0x0207 /* Additive Color Mixer - Blue */
```

Ich verwende hier u.A. auch die Codes für `SENS_POSITION` um eine spätere Anbindung an ein LPS[32] zu ermöglichen.

Fazit RDM wird allgemein wenig Bedeutung zugemessen („dead on arrival“), da es recht kompliziert zu implementieren ist, da viele Firmen bereits ähnliche proprietäre Lösungen entwickelt haben und da z.B. Ethernet-basierte Lösungen (*Artnet, Sand-Net, ACN*) bereits viele Funktionen (Adressierung, Routing, Verwaltung, Full-Duplex) ”out-of-the-box” integriert haben. Für RDM gibt es nur wenige freie Bibliotheken, die nur sehr unvollständig sind. Es hat aber einen entscheidenden Vorteil: DMX512 Verkabelung kann benutzt werden, vorhande DMX512-Geräte können (sofern die Schnittstellenbausteine für bidirektionale Kommunikation konfiguriert sind) rein softwareseitig auf RDM umgerüstet werden. Dies macht es interessant für Sensormote und Lightmote Netzwerke: Ich halte es momentan nicht sinnvoll, jeden Smart-Pixel mit einer Ethernet- oder WLAN-Schnittstelle und entsprechenden Protokollstacks auszurüsten. DMX512 soll zum Ansprechen von Lightmotes als Bussystem weiter eingesetzt werden. Die Möglichkeit Sensormotes via RDM in diese Bussysteme einfügen zu können, erscheint vorteilhaft. Ein einziger Gateway-Controller würde ausreichen, um Sensormotes und Lightmotes über RDM anzusprechen. RDM erweitert dabei die Lightmotes um Device-Discovery-Funktionalität und dynamische Adressierung. Hierfür müsste nur ein Ausschnitt aus dem umfangreichen RDM-Protokoll implementiert werden (gemäß der Devise: „keep it simple“). RDM ist ein offener Standard, d.h. es bleibt zumindest die Hoffnung, dass die Motes auch mit Software anderer Hersteller funktionieren würden. Sicher gibt es Alternativen zu RDM (nicht zu letzt uOSC 4.2.3).

4.2.6 Synchronisation

Um zeitgleich Ereignisse auszulösen, müssen die Knoten im Netzwerk synchronisiert werden. Hierfür bietet sich - wie in den OSC-Spezifikationen vorgegeben - das *Network Time Protocol* (NTP) an (vgl. Mills (1992)). Eine quelloffene NTP-Server Variante ist z.B. unter

[32] Local Positioning System

Ubuntu-Linux ohne aufwendige Konfiguration nutzbar [33]. Ein Controller im System kann somit einfach als NTP-Server konfiguriert werden. Angeschlossene Controller synchronisieren sich als NTP-Clients mit diesem. Besteht eine Verbindung ins Internet, kann auch ein öffentlicher NTP-Server [34] verwendet werden. Die 64bit-langen Zeitstempel von NTP sind sehr genau (Picosekunden-Bereich) und erlauben so, dass gefühlt zeitgleiche Aufrufen von OSC-Methoden im Netzwerk via time-tagged OSC-Bundles. NTP-Clients sind auch auf eingebetteten Plattformen z.B. als Bestandteil der OpenWRT-Distribution für WLAN-Router verfügbar.

4.2.7 Service Discovery

Um *Plug-And-Play* Funktionalität zu gewährleisten, ist der Einsatz von geeigneten Strategien zum Auffinden von Geräten und den zur Verfügung gestellten Diensten notwendig (*Device Discovery* und *Service Discovery*). Für die bislang in interaktiven multimedialen Installationen eingesetzten Kommunikationsprotokolle (inkl. OSC) gibt es hierfür keine standardisierten Lösungen. MIDI löst das Problem mit statisch festgelegten Adressen und vordefinierten Nachrichten für spezielle Funktionen (MIDI Implementation Chart). Bei DMX512 ist es ähnlich: statische Adressen, vordefiniertes Kanalbelegung (z.B. erster Kanal = rot), keine unterschiedlichen Typen sind möglich (nur 8-Bit Byte). OSC ist flexibler im Hinblick auf die Adressierung, aber es ist damit auch schwieriger Plug-And-Play zu realisieren, da die Bennung und Typisierung von OSC-Methoden und Containern völlig frei wählbar ist. In kollaborativen verteilten Multimedia-Systemen ist ein *Query System* notwendig: Spontan in das Netz eingefügte Motes, sollen automatisch den Controllern bekannt werden, und diese sollen wiederum die eingefügten Geräte dem User-Interface bekannt machen. Ein solches Query System für OSC wurde von unterschiedlichen Forschungseinrichtungen vorgeschlagen, aber bis jetzt noch nicht in die Spezifikation aufgenommen. Schmeder und Wright schlugen bereits 2004 ein OSC Query System vor (Schmeder2004 (2004)) für den Einsatz in Szenarien mit anwendungsübergreifenden Steuerungen. Ebenso wurde von JazzMutant[35] - Hersteller eines OSC-Multitouch-Controllers - ein Namespace-Query System vorgeschlagen, welches in die OSC 2.0 Spezifikation aufgenommen werden sollte. Weiterhin entstand „QSCQS“ [36] ohne sich als Standard durchgesetzt zu haben.

Derzeit gibt es keinen Konsens in der OSC Entwicklergemeinde für ein reines OSC Query System, zumindest aber für das Auffinden von OSC-Servern scheint sich eine Standardlösung herauszukristallisieren: *Multicast UDP* Nachrichten. Für diese Discovery Methode gibt es bereits einen anerkannten Standard: *Zeroconf*. U.A. bereits 2006 implementiert als „OSCBonjour“ von Remy Müller am IRCAM [37].

[33] http://www.openntpd.org/
[34] z.B. de.pool.ntp.org
[35] http://jazzmutant.com
[36] Schema for OpenSound Control Query System version 0.0.1, http://liboscqs.sourceforge.net/
[37] Institut de Recherche et Coordination Acoustique/Musique, Paris

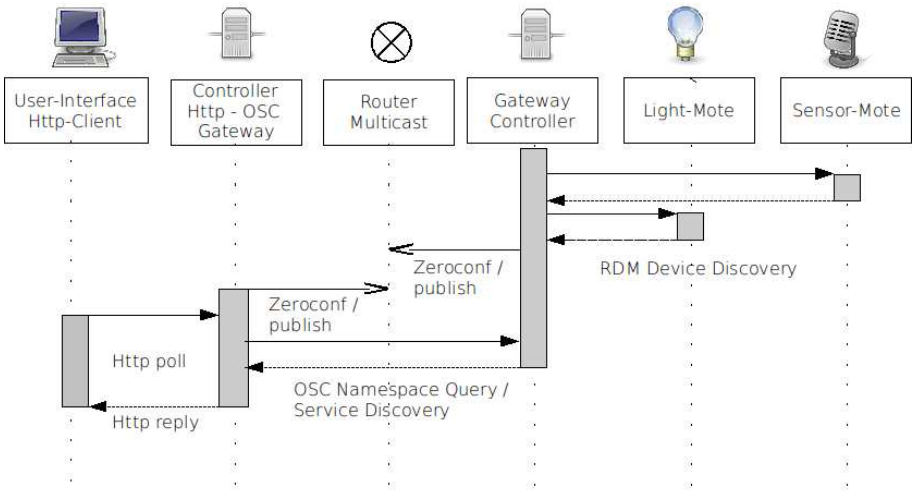

Abbildung 23: Sequenzdiagramm einer vereinfachten Discovery Phase

Zeroconf Zeroconf (auch als *TIB/Rendezvous* oder *Apple-Bonjour* bekannt) ist ein Protokoll für die dynamische Erkennung und konfigurationsfreie Vernetzung von Rechnern im LAN. Es baut auf den Protokollen *Multicast DNS*[38] (mDNS) und *DNS Service Discovery* (DNS-SD) auf und ermöglicht so das Auffinden von Diensten ohne zentralen Directory-, DNS- oder DHCP[39]-Server, wobei diese evtl. vorhandenen Netzwerkdienste nicht beeinträchtigt werden.
Zeroconf baut auf offenen Standards auf und es sind quelloffene Implementation verfügbar. Unter den gängigen Linux-Distributionen (inkl. embedded Linux Varianten) wird aktuell *avahi* [40] verwendet (siehe 6.2.3). Für Java gibt es die freie Implementation „jmDNS“ [41] und unter Windows und MAC OS ist Zeroconf via Bonjour möglich. Das macht Zeroconf interessant, um OSC mit Service Discovery Funktionalität zu erweitern (vgl. Ramakrishnan (2004)). Zeroconf konkurriert dabei mit verwandten Projekten wie *Jini* von Sun [42] und *UPnP* von Microsoft [43]. Ein Vergleich dieser Systeme soll hier aber nicht angestellt werden, da es hier nur um eine sehr einfache Implementation von Service Discovery geht, die mit jeder der verfügbaren Technologien zufriedenstellend umgesetzt werden kann.

Dienste werden bei Zeroconf nach einem speziellen eindeutigen Schema veröffentlicht. Für OSC-Dienste werden folgende Typen registriert (obwohl diese noch nicht in die offiziell publizierte Liste von DNS-Servicetypen [44] aufgenommen wurden):

[38] Domain Name System
[39] Dynamic Host Configuration Protocol
[40] http://avahi.org
[41] http://jmdns.sourceforge.net/
[42] http://jini.org
[43] Universal Plug and Play
[44] http://www.dns-sd.org/ServiceTypes.html

```
_osc._udp

_osc._tcp
```

Zusätzlich zu Servicename, Port und Protokollinformationen können beschreibende Text-Argumente veröffentlicht werden. Andrew Schmeder, Entwickler von OSC schlägt hierfür die Felder

```
version=   /* der OSC Spezifikation */
framing=   /* z.B. SLIP */
schema=    /* z.B. http://sukale.com/osccontroller.xml */
```

vor. Jeder Netzwerkknoten kann Dienste „abonnieren“. Über einen Listener oder eine Callback-Funktion kann auf Veränderungen im Netz reagiert werden. Ein Daemon[45] filtert dabei eingehende Multicast-Nachrichten und wickelt ausgehende Nachrichten ab (vgl. Tanenbaum und Steen (2003)).

4.3 Kabellose Kommunikation

Im vorgestellten Netzwerk kann sämtiche IP-basierte Kommunikation auch über Funk mit Standard WLAN-Komponenten stattfinden, unabhängig von den eingesetzten Protokollen auf höheren Schichten des Netzwerk-Modells. User-Interface-Clients könnten sich via WLAN mit dem Http-Server verbinden, OSC-Pakete bzw. UDP-Pakete könnten könnten auch drahtlos versendet werden. Nur die Lightmotes sind hier noch kabelgebunden. Als Alternative zu DMX512 könnte (wie auch bei den Berkeley Motes erfolgreich umgesetzt) *ZigBee* verwendet werden, das bei einer Frequenz von 2.4GHz ebenfalls Datenraten von 250kbps erreicht und über Kanal 255 ebenfalls Broadcast Nachrichten an alle Endknoten senden könnte (ähnlich zu einem DMX512-Transmitter). Problematisch bei der Verwendung von Funktechniken im ISM-Frequenz-Band sind jedoch Störungen durch andere konkurrierende Technologien wie Bluetooth und WLAN. Im Ausstellungsbetrieb könnte dies beispielsweise bedeuten, dass die Installationen durch regen Handygebrauch des Publikums im Betrieb gefährdet ist, weswegen in den meisten professionellen Veranstaltungsräumen auf eine kabellose Kommunikation in diesem freien Frequenzbereich verzichtet wird. Entsprechende Untersuchungen zur Leistungsfähigkeit von ZigBee sind im UbiComp-Labor an der HAW-Hamburg bereits von Christian Fischer angestellt worden (vgl. Fischer (2005)). Hier wird zusätzlich zu den Eigenheiten des ZigBee Protokolls auch allgemein festgestellt, dass kabellose „ubiquitous Devices“ immer auch das Problem der kabellosen Stromversorgung lösen müssen: Um sie nicht aufgrund der Stromversorgung wieder ans Kabel zu binden, werden sie per Batterie bzw. Akkumulator betrieben und müssen dementsprechend stromeffizient arbeiten. Dieses Problem kann hier nicht abschließend geklärt werden, so dass im Weiteren die Kommunikation und Stromversorgung der Motes über Kabel erfolgt, ausblickend aber ein mit Solarenergie betriebener Smart-Pixel skizziert wird.

[45] in etwa Hintergrundprozess

5 User Interface

Klassischerweise wurden in multimedialen Installationen Hardware-User-Interfaces verwendet, bei denen Bedienelemente fest mit Methoden eines fixen Objektes verknüpft waren: Ein Regler eines Lichtpultes mit einem Scheinwerfer-Dimmer-Kanal, ein Regler eines Audio-Mischpultes zur Lautstärke-Regelung eines Audiokanals. Verbindungen unterschiedlicher Objekte wurden mit Kabeln verdrahtet, direkt von Gerät zu Gerät oder über sog. *„Patchbays“*.
Mit zunehmender Virtualisierung in Software kamen komplexere Verschaltungen hinzu: Ein Kanal einer Audiosoftware konnte dynamisch einem oder mehreren von insg. 16 MIDI-Kanälen zugeordnet werden. Ein Bedienelement einer Lichtsteuersoftware konnte einem oder mehreren DMX512-Kanälen zugeordnet werden. Es taucht in Anlehnung an die verwendeten Hardware-Patchbays der Begriff *„Patching“* auf, das Verschalten von Modulen untereinander.

Mit zunehmender Komplexität und dem Zusammenschmelzen multimedialer Software stehen bereits innerhalb eines nicht verteilten Systems eine Vielzahl von Objekten (z.B. Module eines virtuellen Synthesizers) zur Verfügung, die es in geeigneter Weise zu bedienen und zu verbinden gilt. Hardware-User-Interfaces bieten nicht mehr für jede Funktion aller Objekte Regler an, sondern ermöglichen dem Anweder eine übersichtliche Anzahl von Bedienelementen auf die gerade gewünschten Funktionen zu *„mappen“*. Wir haben eine Anzahl unterschiedlicher Bedienelemente und Dateneingänge auf der einen Seite und eine Vielzahl von Objekten mit unterschiedlichen Methoden und Datenausgänge auf der anderen Seite (Quellen und Senken).
Einfache Darstellungen dieses Szenarios sind zweispaltige Listen: links Quellen und rechts Senken. Ein entsprechendes User-Interface würde hier dem Anwender die Möglichkeit geben, Verbindungen (Kanten) zu erstellen durch anwählen von je einem Objekt einer Spalte. Ein Open Source Beispiel eines solchen User-Interfaces ist *„QJackControl“* [46] von Rui Nuno Capela. Es stellt Audio Ein- und Ausgänge eines Rechners in Listenform dar und erlaubt das Patchen von diesen untereinander.

QJackControl ist in diesem Kontext weiterhin interessant, da hier User-Interface und Controller getrennt sind: das eigentliche Mapping findet im *Jack-Audio-Server* statt (vgl. Sukale (2004)), während das User-Interface als Monitor nur zunächst den aktuellen Zustand abfragt und bei User-Input Befehle an den Server weiterleitet.
In einem verteilten System wie z.B. dem *McGill Digital Orchestra* finden wir das selbe Prinzip wieder:
ein User-Interface (hier Mapping-Interface genannt) bietet eine zweispaltige Liste an, über die Geräte (Synth's, Controller) des Netzwerkes miteinander verbunden werden können 19.
Allgemein betrachtet reiht sich dieses Konzept in die visuellen datenstrom-orientierten Programmiersprachen ein, nur dass hier kein zweispaltiges Layout mehr verwendet wird, son-

[46] http://qjackctl.sourceforge.net/

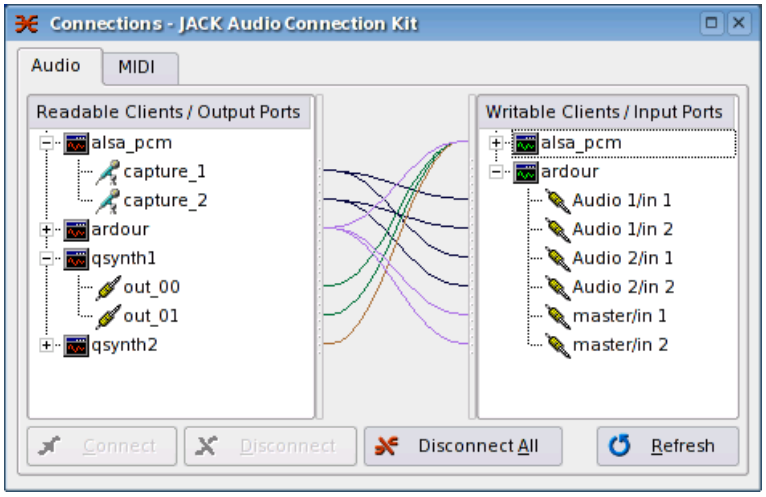

Abbildung 24: QJackCtl Screenshot, Quelle: http://qjackctl.sourceforge.net/

dern Objekte frei auf einer Seite angeordnet werden können und beliebig Graphen zwischen diesen gezogen werden können, so dass ein komplexes Netzwerk, ein Schaltplan des Systems entsteht. Die visuelle Programmierung macht es dabei nicht-Programmierern einfacher Funktionen miteinander zu verknüpfen (vgl. Sukale (2008)). User-Interfaces dieser Art sind in einer Vielzahl von Multimedia-Anwendungen und Sensor-Netzwerk-Systemen integriert. U.A. in *LabView* (25), in der LEGO-Spielzeug-Variante von LabView *NXT*(26), in *PureData* (27) und WebBrowser-basiert in *Lily* (28).

Als reine Mapping-Interfaces sind diese z.B. als *„Patchage“* [47] 29 (ein Nachfolger von QJackControl) unter Linux realisiert worden.

Signalketten und prozedurale Zusammenhänge lassen sich so sehr gut darstellen. Das Bildschirm-Layout entspricht z.B. dem tatsächlichen Hardware-Aufbau oder einem Flussdiagramm. Zunehmend nachteilig wirkt sich jedoch die Unübersichtlichkeit komplexer Netzwerke aus (*information overload*), gerade wenn (wie bei PureData oder MAX/MSP) Patches aus vielen atomaren Funktionen bestehen (mathematische Operatoren, if-Abfragen, etc.) verteilt über mehrere Seiten. Neuere Projekte auf diesem Gebiet versuchen höhere Abstraktionsgrade für mehr Lesbarkeit und Interoperabilität einzuführen (vgl. Place (2006)). Im Fall von *„Jamoma“* für MAX/MSP ist dies elegant gelöst, da hier versucht wird eine grundlegende Struktur für Patches einzuführen auf Basis (des hier ebenfalls verwendeten) OSC-Protokolls. Objekte die der Jamoma-Empfehlung folgen, implementieren Standard-Interfaces für ihre Methoden, so dass sie System-übergreifend verwendet werden können und z.B. auch mit standardisierten GUIs angesprochen werden können. In verteilten, interaktiven, multimedialen Installationen könnte dies aufgrund der nochmals gesteigerten Komplexität von Patches gegenüber rein lokalen Anwendungen vorteilhaft sein. Im Folgenden sollen allgemein die Anforderungen an das hier zu entwerfende User-Interface zusam-

[47] http://wiki.drobilla.net/Patchage

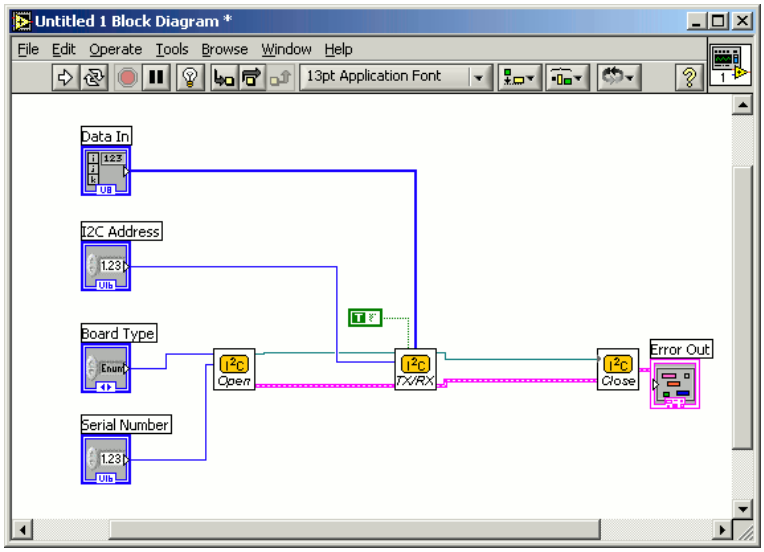

Abbildung 25: Labview, visuelle Programmierumgebung, Quelle: National Instruments Corporation Website

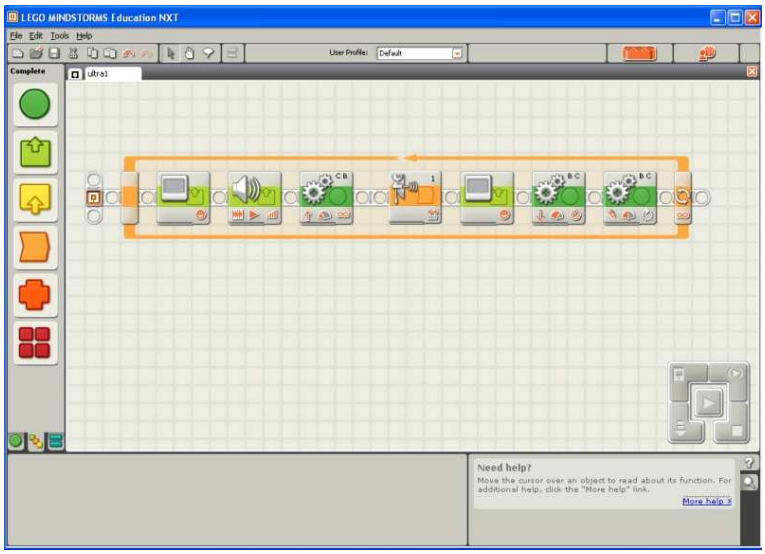

Abbildung 26: NXT Education Software für Lego ”Mindstorms”, Quelle: National Instruments Corporation Website

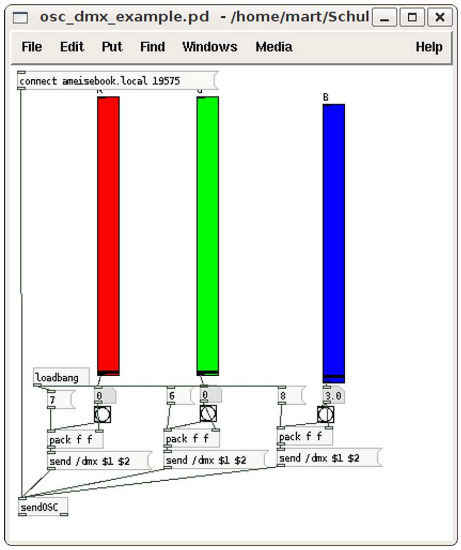

Abbildung 27: PureData, Open Source Programmiersprache und IDE für das visuelle Programmieren von interaktiven multimedialen Anwendungen, hier Beispielimplementation eines User-Interface zur Ansteuerung von Lightmotes via OSC und DMX512

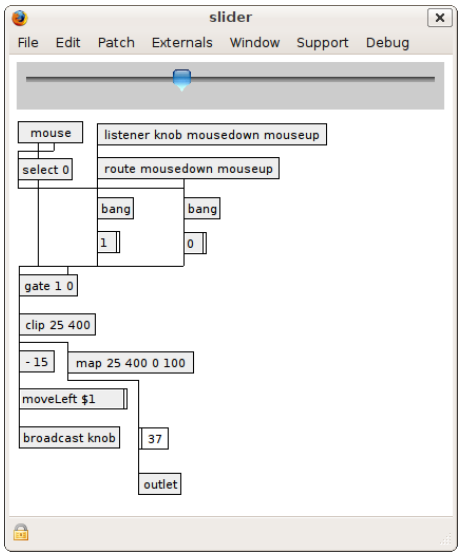

Abbildung 28: Lily, Open Source Programmiersprache und IDE für das visuelle Programmieren von interaktiven clientseitigen (Firefox-Add-Ons) Applikationen

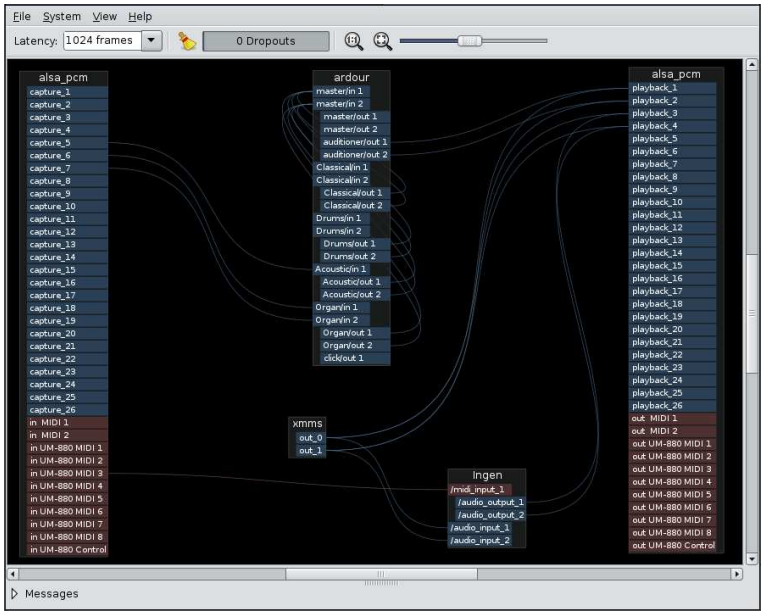

Abbildung 29: Patchage Screenshot, Quelle: http://wiki.drobilla.net/Patchage

mengefasst werden und unter Berücksichtigung der o.g. bewährten Konzepte eine mögliche Umsetzung vorgestellt werden.

5.1 Anforderungen

Funktionale Anforderungen

- verfügbare Objekte anzeigen
- verfügbare Methoden anzeigen
- Objekte verknüpfen (Mapping, Patching)
- Objekt-Methoden aufrufen
- Systemzustände anzeigen (Monitoring)

Im konkreten Fall hiesse dies beispielsweise: Motes und Controller im System anzeigen. Light-Motes-Methoden (Lichtintensität, Farbe regeln) anzeigen. Sensor-Motes-Methoden anzeigen (Wert auslesen). Sensor-Methoden auf Light-Mote-Methoden abbilden (z.B. Lichtsensor -> Lichtintensität). Direkt Light-Mote-Methoden aufrufen (Farbwerte einstellen). Aktuelle Parameter (Light-Mote Lichtintesität, Sensorwerte, Controller-Einstellugen, Mapping) darstellen. Reagiert z.B. ein Light-Mote auf einen Sensorwert, so soll der Zustand des Motes auch im GUI dargestellt werden, natürlich möglichst in Echtzeit

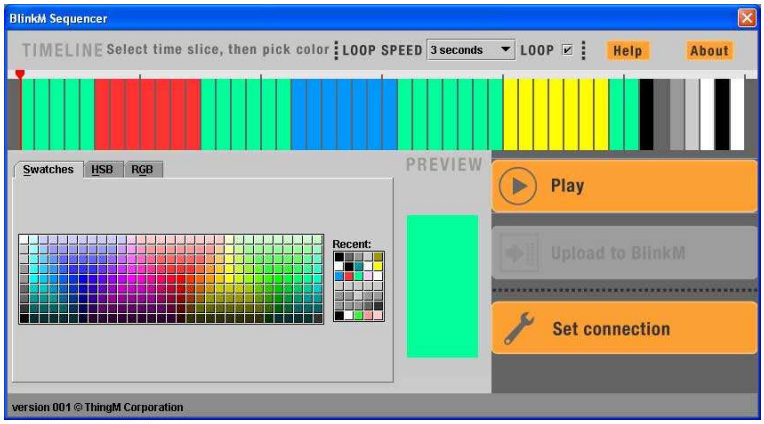

Abbildung 30: BlinkM RGB Sequencer Software, typische Anwendung aus dem Lichtkunst-Umfeld zur Programmierung von sog. RGB-Steps und Fades, Quelle: Blinkm Datasheet v20080130a, ThingM Corporation

(wobei das direkte Verhalten zeitkritischer ist als die „Spiegelung“ auf die GUI). Zusätzliche typische Anforderungen an das User-Interface von Light-Motes ist die Kompositions- und Darstellungsmöglichkeit von Farbverläufen über eine Zeitachse (siehe 30).

Non-funktionale Anforderungen Wie im übergreifenden Systementwurf dargestellt, soll das User-Interface ebenfalls folgenden non-funktionalen Anforderungen genügen:

- offenene Standards
- Open Source Software
- portierbar und embeddable (z.B. auf mobile Endgeräte)
- möglichst barrierefrei
- bedienungsfreundlich
- low-cost

5.2 Umsetzung

Keine der untersuchten visuellen Programmierumgebungen erfüllt die Anforderungen eines User-Interfaces für das hier vorgestellte System: PureData käme zwar als erstes in Frage aufgrund der Offenheit des Projektes, seiner weiten Verbreitung und der Unterstützung für OSC, ist in seiner jetzigen Version jedoch nicht eindeutig in Client/Server getrennt und erfahrungsgemäß nicht besonders intuitiv und einfach zu bedienen. MAX/MSP hat ebenfalls gute OSC-Unterstützung, ist aber wie LabView und Lego-NXT kommerziell und

nicht frei und quelloffen erhältlich.

Um eine möglichst portierbare, einfache und bedienungsfreundliche Lösung zu konstruieren, soll hier exemplarisch ein neues User-Interface in Form einer Rich-Client / WebBrowser-Applikationen vorgestellt werden. Ein vergleichbarer Ansatz ist bereits mit *„Lily“* [48] von Bill Orcutt realisiert worden. Idee und Funktionsumfang ist bereits sehr weit entwickelt. Leider ist Lily nicht barrierefrei und schwer portierbar, da es zur Zeit nur als Plug-In von *Firefox* bzw. *XULRunner* läuft. Zudem ist Lily wie PureData relativ schwierig zu bedienen.

Rich Internet Applikationen (RIA) erfreuen sich einiger Publicity aufgrund von erfolgreichen Anwendungen wie *Google-Mail* etc. Technologien wie client-seitige Skriptsprachen-Interpreter (*JavaScipt, Flash, Flex*), asynchrone Kommunikation zwischen Client und Server sowie die rasche Weiterentwicklung und kostenlose Verfügbarkeit von Webbrowsern wie Firefox haben viele (auch Open Source) Anwendungen hervorgebracht zum Erstellen von dynamischen interaktiven Inhalten auf Webseiten. Mehrere Veröffentlichungen des *UbiComp-Labs* an der HAW-Hamburg befassen sich mit Technologien um RIAs[49]. Fazit der angestellten Untersuchungen ist die Einsetzbarkeit in diesem konkreten Vorhaben unter Beachtung der o.g. Anforderungen. Konkrete Zusammenhänge und Implementierungen sind im Folgenden dargestellt.

5.2.1 Abbildung physikalischer Objekte und OSC-Namensraum auf DOM-Objekte

Eines der Hauptkriterien für die Wahl einer Rich-Client-Lösung auf HTML-Basis gegenüber einer eigenständigen, nicht Browser-basierten Lösung war, die Möglichkeit physikalische Objekte wie Light-Motes und Softwareobjekte auf (X)HTML-Elemente innerhalb geschachtelter DOM-Objekte abzubilden. Es bietet sich an, Container-Klassen wie `<div>` für die einzelnen Entitäten des Systems anzulegen, die wiederum untergeordnete Einheiten des Systems beinhalten. Da wie in 4 erläutert, OSC ebenfalls den Begriff des Containers einführt, ist ein eindeutiger Zusammenhang gegeben: Für jeden OSC-Container im System wird ein HTML-Container angelegt. Ein OSC-Pfad wie z.B.:

```
/controller/1/pixel/1/rgb
```

kann also dargestellt werden als:

```
<div id='controller1' class='osc_container'>
   <div id='pixel1' class='osc_container'>
      <div id='rgb' class='osc_method'>
```

[48] Lily ist eine Browser-basierte in ECMAScript geschriebene visuelle Programmierumgebung mit Unterstützung für multimediale Inhalte und OSC-Protokoll Anbindung via flosc, ähnlich zu PureData, http://www.lilyapp.org/

[49] http://users.informatik.haw-hamburg.de/ ubicomp/papers.html

```
            <!-- specific interface elements go here -->
          </div>
        </div>
      </div>
```

Ein entsprechender Parser ist einfach und kann z.B. clientseitig in JavaScript ausgeführt werden. In der Device- und Service-Discovery Phase kann das User-Interface also clientseitig dynamisch aufgebaut werden.
Für den Anwender ist die Schachtelung in Container sehr übersichtlich. Über *CSS*[50] und Bibliotheken zur sog. *DOM-Manipulation* können die Container-Elemente optisch frei gestaltet werden und mit Zusatzfunktionen (z.B. "Drag'n'Drop", Animationen) ausgestattet werden. Sollte HTML/CSS nicht ausreichen, um gewünschte Funktionen des User-Interfaces zu implementieren, kann dieser Ansatz durch die Kombination von SVG[51] und ECMA-Script erweitert werden, wobei dann grafische XML-Elemente die Entitäten des Systems repräsentieren und völlig frei gestaltet werden können (z.B. auch mit „Patch-Kabeln" bzw. Linien verbunden werden können wie in den gängigen Mapping-Interfaces). Canvas-APIs moderner Browser wären eine weitere Möglichkeit.

5.2.2 Clientseitige Javascript Bibliotheken für Rich Client Applikationen

Da Adobe Flash - obwohl es weitverbreitet ist und es bereits ein Projekt zur Kommunikation zwischen Flash-Programmen und OSC gibt - aufgrund seiner Konflikte mit der Anforderung nach quelloffenen Lösungen als Kandidat ausscheidet [52] bleibt hier als Standard für clientseitige Programmierung nur ECMAScript[53]. Wie erwähnt sind vor allem durch die RIAs der Firma Google einige ECMAScript Frameworks für Rich Content bekannt geworden:

- *Google Web Toolkit* (vgl. Carfagno (2007))
- *prototype.js* - MIT Lizenz, eines der ersten und verbreitesten Frameworks, eher spärlich dokumentiert
- *mootools* - prototype.js auf grafische Effekte spezialisiert
- *MochiKit* - MIT Lizenz, Fokus auf Tests und Dokumentation
- *Dojo Toolkit* - BSD Lizenz, Paketmanager , sehr umfangreich („Dijit", „Dojox"), gut dokumentiert (tutorials, „Dojo Book"), Server->Client Kommunikation („comet")

[50] Cascading Stylesheets

[51] Scalable Vector Graphics, XML-basiertes Vektor-Bildformat

[52] ebenso die OpenSource Projekte von Adobe wie z.B. „Flex", da diese in unkommerziellen Umgebungen algemein ebenfalls als zweifelhaft angesehen werden

[53] ECMAScript wird hier als genauere Bezeichnung für die JavaScript-Sprache verwendet, welche u.A. mit ActionScript und JScript in der ECMA-262 Spezifikation standardisiert wurde.

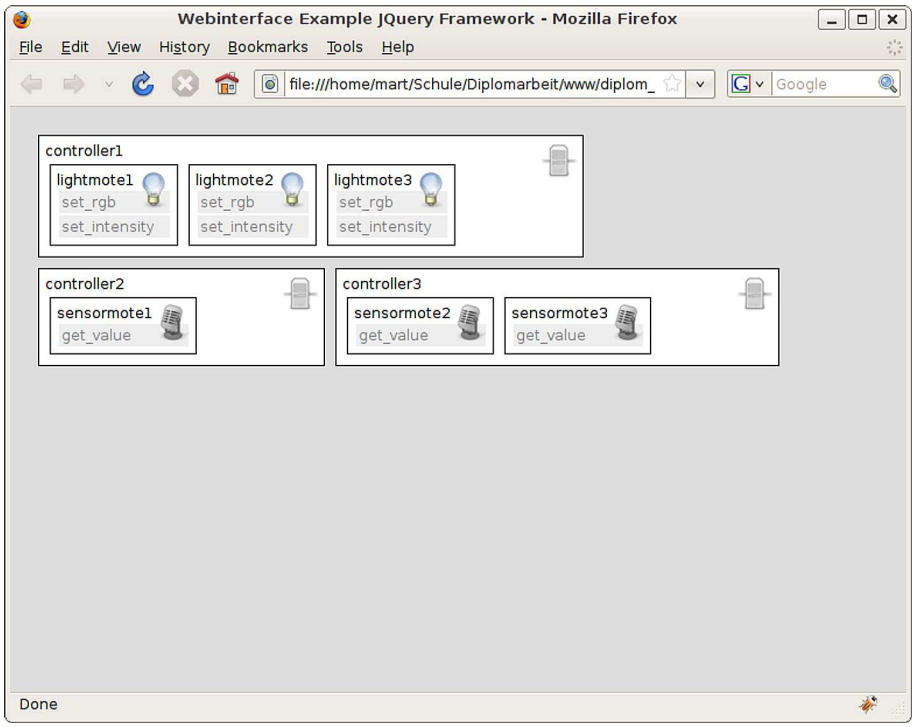

Abbildung 31: „Diplom“-User-Interface: OSC-Pfad abgebildet auf geschachtelte DOM-Objekte

- *jQuery* - MIT/GPL Lizenz, „objektorientierter“, Java-ähnlicher, unterstützt Xpath, erweiterbar durch Plug-Ins, Verkettung von Methoden eines Objektes (`return this;`)
- und viele mehr...

Allen gemeinsam sind schlanke Methoden zum Manipulieren von DOM-Objekten über kurze Deskriptoren (z.B. `$()`), die Erweiterung von JavaScript hin zu mehr objektorientierter Funktionsweise, das Bereitstellen von sog. *„Ajax“*-Frameworks und der allgemeine Fokus auf grafische Effekte über CSS-Manipulation (Animationen, Verstecken und Anzeigen von Ebenen, Widgets). Diese Frameworks machen es auch nicht versierten Programmierern möglich, in einer abstrakten Skriptsprache schnell und einfach Bedienoberflächen zu erzeugen für interaktive Installationen. Zudem ist es möglich durch Zugriff auf umfangreiche Widget-Bibliotheken Standard-Bedienelemente (z.B. „Slider“, „Buttons“, „Menus“) zu verwenden und evtl. eigene wiederverwendbare hinzuzufügen. Dabei können diese auch auf mehrere Server verteilt und soz. „on-demand“ in die Oberfläche eingebunden werden. Jeder Controller oder sogar einzelne Motes, können so Ihre eigenen Bedienelemente anbieten.

Seiten können mit Standard-Techniken zur Zugriffskontrolle entsprechend der User-Rolle gefiltert werden (Komponist, Techniker, Ausstellungspersonal) und spezielle *Views* angelegt werden. Das User-Interface ist ohne zusätzliche Software-Installation (in gewissem Sinne „barrierefrei“) mit jedem Webbrowser (W3C-Standard) nutzbar, und auch mit Standard-Werkzeugen validierbar. Das User-Interface ist portierbar auf z.B. Mobile-Clients mit entsprechenden Browsern. Das User-Interface ist intuitiv und einfach zu bedienen, da Anwender den Umgang mit ähnlichen Anwendungen durch die Verbreitung im Internet gewöhnt sind. Design und Funktion sind standardmässig getrennt (CSS, HTML, ECMA-Script), das Interface kann bzgl. Übersichtlichkeit im Nachhinein sehr flexibel angepasst werden (im Gegensatz zu relativ starren Layouts bei PureData oder MAX/MSP).

Beispielimplementation mit JQuery Exemplarisch sei hier die Implementation eines User-Interface mit dem von John Resig entwickelten Framework *JQuery* vorgestellt (siehe 31):
Jquery eignet sich besonders gut für das Mappen von physkalischen Objekten und ihren Methoden auf DOM-Objekte, da diese als Objekte auch in JavaScript ansprechbar sind. Zudem ist es möglich diese JQuery-Objekte um eigene Funktionen und Methoden zu erweitern. Jquery ist quelloffen und wird derzeit relativ aktiv entwickelt. Es gibt mehrere Plug-In Bibliotheken zur Erweiterung der Core-Biliothek. Im *interface.js* Plug-In findet sich z.B. ein Slider-Objekt, das geeignet ist Werte wir Lichtintensität, Farbe, Lautstärke intuitiv ansprechbar zu machen.

5.2.3 JSON

Um OSC-Server - wie hier vorgestellt - via ECMAScript Webinterface zu steuern und zu überwachen, könnte man direkt vom User-Interface-Client OSC-Nachrichten über UDP versenden. Derzeit fehlt jedoch eine einfache Lösung, um aus ECMAScript heraus UDP-Pakete zu versenden. Ebenso gibt es zwar eine Java-Bibliothek um OSC Nachrichten zu generieren, für JavaScript allerdings noch nicht. Hier müsste also Entwicklungsarbeit geleistet werden: Eine ECMA-Skript-OSC-Bibliothek müsste geschrieben werden, die eine Serialisierung ins OSC-Byteformat übernimmt. Es stellte sich jedoch die Frage, ob es sinnvoll ist OSC Nachrichten direkt vom Client aus zu verschicken oder ob es besser wäre, für die Kommunikation zwischen User-Interface-Client und Controller auf bereits ausgereifte Standards zuzugreifen und vorhandene offene Methoden zum Datenaustausch zu verwenden?

Zwei Standards haben sich für die Übertragung von Daten zwischen Client und Server in RIAs basierend auf ECMAScript durchgesetzt: *XML*[54] und *JSON*[55]. Letztere erscheint derzeit als die einfachere Variante und wird hier eingesetzt: OSC-Nachrichten bzw. Methodenaufrufe sowie Daten zur Aktualisierung der grafischen Oberfläche werden in das JSON-Format gekapselt. Serverseitig werden diese dann von einem JSON-to-OSC bzw. HTTP-to-OSC Gateway (wie z.B. bei *flosc*) weitergeleitet (siehe 6.2.6).

5.2.4 Fazit

Webbrowser-basierte User-Interfaces scheinen in diesem Fall die geeignete Wahl gegenüber speziellen Desktop-Anwendungen. User können sich an einem beliebigen Punkt des Netzwerkes in das System einloggen und Veränderungen vornehmen, danach läuft das System wieder Stand-Alone und „unsichtbar“. Spezielle Hardware und Softwarelösungen für das User-Interface werden nicht benötigt. Anwender können auf wahrscheinlich bereits vorhandene Geräte zurückgreifen. Benötigte Software ist auch für mobile Geräte verfügbar und wird sehr aktiv entwickelt. Obwohl die Client-Server-Technik bereits lange bekannt und ausgereift ist, so gibt es bzgl. OpenSource RIAs derzeit noch einige Schwierigkeiten: nicht alle Anwendungen laufen auf allen Browsern (z.B. *Lily* nur auf Firefox, *Flex* nur mit zusätzlicher kommerzieller Software). Probleme wie Server-initierte Kommunikation (Stichworte: *„comet“,„server push“, „pushlets“, „long-polling“*) sind zwar lösbar (*„Persistent Communication Pattern“* Gross (2006)) aber gehören noch nicht zum Standard.

[54] Extensible Markup Language
[55] JavaScript Object Notation

6 Controller

Mit *Controller* wird hier allgemein ein Steuergerät bezeichnet gemäß dem Netzwerk-Aufbau im Abschnitt „Kommunikation“ (4). Eine interaktive Installation wird grundsätzlich ausgehend von diesen Controllern aufgebaut, von denen einer oder mehrere - je nach Umfang des gestalterischen Konzeptes - eingesetzt werden kann. Controller sind notwendig, da die Motes in der hier umgesetzten Form noch nicht „intelligent“ genug sind, um selber als eigenständige Controller zu fungieren. Ein *Smart-Pixel* ist z.B. noch nicht in der Lage, sein eigenes User-Interface in Form von HTML-Seiten und HTTP-Server mitzubringen. Ebensowenig können Motes alle weiteren gewünschten Funktionalitäten des Systems (Parameter-Mapping, Routing, Service-Discovery) implementieren, so dass diese an entsprechende Controller bzw. Controller-Netzwerke deligiert werden. Auch soll gemäß MVC-Muster eine strenge Trennung von User-Interface und Controllern beibehalten werden.

Im klassischen Sinne betrachtet, übernehmen die Controller hier die Rollen eines MIDI-Sequencers, Lichtsteuerpults und Medienservers. Ausgehend von einem Controller werden im Laufe des kreativen Prozesses Komponenten zu diesem hinzugefügt, analog zu einem Dirigenten eines Orchesters, der für eine Aufführung Musiker mit unterschiedlichen Instrumenten um sich herum versammelt.

6.1 Anforderungen

Funktionale Anforderungen

- **Routing** von Datenpaketen an Motes, Controller und User-Interface
- **Mapping** von Parametern und Funktionen („Patching“)
- **OSC-Sender/Empfänger**
- **Gateway-Server** (z.B. OSC->USB->seriell->DMX512)
- **Zeit-Server** (siehe 4.2.6)

Non-funktionale Anforderungen

- Open Hardware
- Open Source Software
- plattformunabhängig
- embeddable
- automatisierbar, eingenständig (*Stand-Alone*-Betrieb)

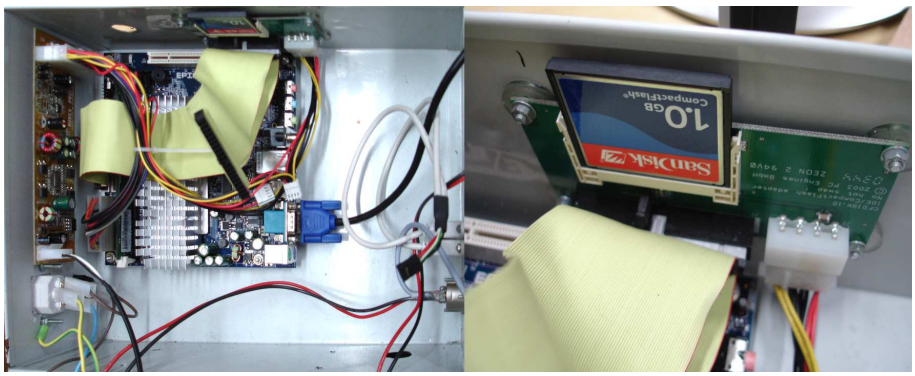

Abbildung 32: Stand-Alone Controller: Versuchsaufbau mit lüfterlosem VIA Epia Mini-ITX-Board und Linux-Betriebssystem auf ”Compact Flash” Karte

6.2 Umsetzung

Als Hardware Plattformen bieten sich neuere *x86*-basierte Systeme an, die mittlerweile relativ klein und stromsparend aufgebaut werden können (32). Installation eines quelloffenen Betriebsystems wie *Linux* stellt keine Probleme dar. Ebenso wären *ARM*- oder *MIPSEL*-basierte Systeme geeignet, auf denen spezialisierte Linux-Derivate (z.B. μLinux) installiert werden können (33). Linux als Betriebssystem erfüllt hierbei die Anforderungen: quelloffen, plattformunabhängig, embeddable, automatisierbar (z.B. über Shell-Skripte).

Um die geforderte Gateway-Funktionalität umzusetzen, werden zusätzlich noch Hardware-Schnittstellen mit entsprechenden Linux-Treibern benötigt (wobei davon ausgegangen werden kann, dass Ethernet oder WLAN Schnittstellen serienmässig vorhanden sind). Softwareseitig werden ein *HTTP-Server* und *OSC-Server/Client* benötigt. Die hierfür eingesetzten Komponenten werden im Folgenden vorgestellt.

6.2.1 DMX512 / RDM Schnittstellen

Durch die weite Verbreitung von USB-Schnittstellen und die einfache Implementierungsmöglichkeit bietet es sich an, DMX512 Nachrichten über *USB-to-DMX512*-Interfaces zu senden. Im einfachsten Fall bestehen diese lediglich aus Schnittstellenkonvertern (z.B. USB->TTL-seriell, seriell->EIA-485, siehe 34), wobei die DMX512 Nachrichten im Controller zusammengesetzt werden und zeichenweise über die USB-Schnittstelle bzw. die virtuelle serielle Schnittstelle (Linux: `/dev/ttyUSB`) ausgegeben werden. Nachteile hierbei sind die größere Last auf der Controller-Seite zur Ansteuerung der seriellen Schnittstelle und der Verlust der Verbindung bei Leitungsunterbrechung oder Ausfall des Controllers. Vorteilhaft ist der simple und kostengünstige Aufbau. Es gibt bereits quelloffene Treiber für diese Art von Geräten u.A. von der Firma Enttec, die das - als Open Hardware herausgegebene - *OpenDMX*-Interface vertreibt. OpenDMX oder der von Hendrik Hölscher empfohlene Auf-

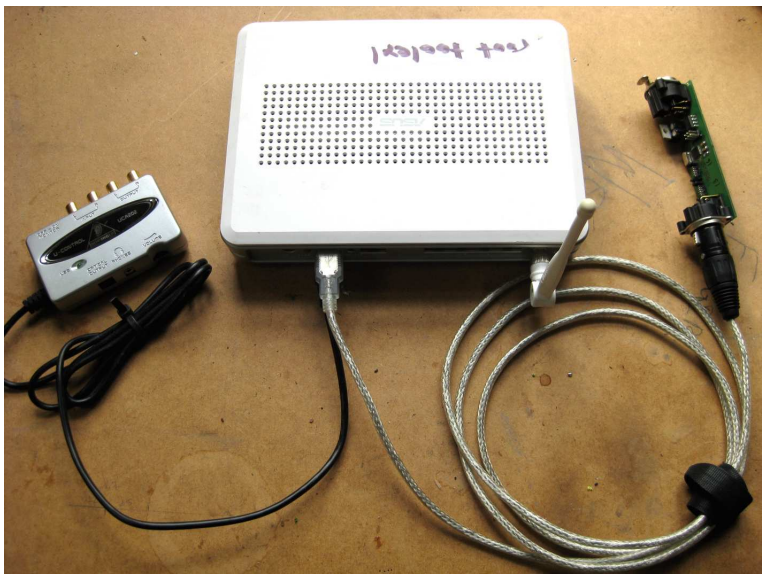

Abbildung 33: Stand Alone Controller: Versuchsaufbau mit Asus WLAN-Router, Behringer USB Soundkarte, Rodin USB-DMX512-Interface und angeschlossenem DMX512-Receiver

bau als *OpenRDM*-Interface[56] basieren auf USB-seriell-Konvertern der Firma FTDI. Auch die in dieser Arbeit vorgestellte Experimentierplattform *Arduino* verwendet diese Bausteine und kann daher - erweitert um EIA-485 Pegelkonverter - als OpenDMX-Interface genutzt werden (vgl. Ness (2006)). OpenRDM-Interfaces entsprechen den o.g. Anforderungen und wurden bei meinen Versuchsaufbauten eingesetzt. Es sei jedoch angemerkt, dass eine gepufferte Lösung, mit Zwischenspeicherung der Ausgabebefehle in einem Microcontroller langfristig sinnvoller wäre[57]. Durch den universellen Aufbau (bidirektionale Kommunikation) des hier vorgestellten DMX512-Receivers (siehe 7.1), liesse sich dieser durch entsprechende Software als ein solches Gerät verwenden.

6.2.2 OSC Server/Client

Für OSC-Server und Client empfiehlt sich der Einsatz der C-Bibliothek *liblo*. Liblo ist die am weitesten entwickelte freie Implementation von OSC. Die Bibliothek ist unter allen gängigen Linux-Distributionen vorhanden. Das Projekt wird auf Sourceforge[58] gehostet und stützt sich so bereits auf geeignete Werkzeuge zur Versionverwaltung, Dokumentation, Projektmanagemet und Distribution. Der Einsatz der Bilbiothek ist sehr einfach.
Über den Aufruf

```
lo_server server = lo_server_new(NULL, error);
```

[56] http://sourceforge.net/projects/openrdm/

[57] Für Versuchsaufbauten verwende ich ebenfalls das von Peperoni Lightning vertriebene gepufferte Rodin USB2DMX Interface mit Treibern für Linux

[58] http://www.sourceforge.net

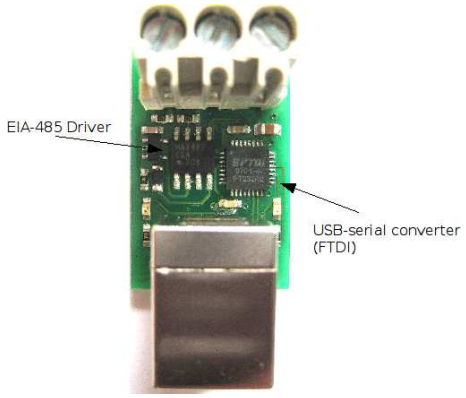

Abbildung 34: USB-EIA485 Konverter von Devantech, einsetzbar als ungepuffertes OpenDMX / OpenRDM Interface

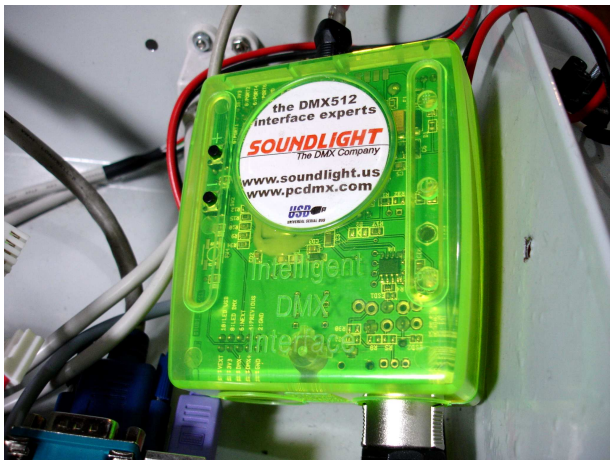

Abbildung 35: USB-DMX512 Interface von Soundlight auf Basis von 8-bit Atmel Microcontroller, Beispiel für kommerzielles ”closed-source” Gerät, Quelle: Daniel Hausig, HBK-Saar

wird ein neuer OSC Server gestartet, der OSC-Pakete via UDP auf einem bestimmten Port empfängt. In der aktuellen Version (0.25) wird hier noch nicht automatisch der Dienst über Zeroconf publiziert, so dass das hier noch über einen zweiten Aufruf und die Bibliothek *avahi* gemacht werden muss (siehe 6.2.3).
Läuft der Server, können über

```
lo_server_add_method(s, "/pixel/rgb", "fff", pixel_handler, NULL);
```

Methoden (hier für Light-Motes) registriert werden. Bei Empfang einer passenden Nachricht wird die Funktion `pixel_handler` aufgerufen, die in diesem Fall z.B. die übergebenden Argumente in das DMX512-Format umsetzt und an die serielle Schnittstelle weiterleitet (siehe mitgelieferten Quelltext `osc2dmx.c`). Entsprechende Methode und Handler muss für Sensor-Motes registriert werden:

```
lo_server_add_method(s, "/sensor/state", NULL, sensor_handler, NULL);
```

s Eine Handler-Funktion ist dabei wie folgt aufgebaut:

```
int
pixel_handler( const char *path,
               const char *types,
               lo_arg **argv,
               int argc,
               void *data,
               void *user_data
             )
```

Sie kann also über Arrays auf sämtliche Informationen einer OSC-Nachricht zugreifen.

Alternativ zu liblo gibt es für Java z.B. das *JavaOSC*-Paket [59], bei dem ein entsprechender Aufruf wie folgt aussieht:

```
receiver = new OSCPortIn(7770);
receiver.addListener("/pixel", listener);
receiver.startListening();
```

Kollisionsvermeidung Bei der Initialisierung von OSC-Servern können Kollisionen auftreten, wenn versucht wird, einen bereits geöffneten IP Port ein zweites Mal zu verwenden. Da es keine Rolle spielt, welche Portnummer OSC verwendet und auch in der Spezifikation kein Default-Port angegeben wird, kann hier eine zufällige Vergabe von Portnummern angewendet werden, solange die gewählte Portnummer größer als 1023 ist [60]. Sollte der zufällig gewählte Port bereits vergeben sein, so wird einfach ein weiterer Versuch gestartet. Entsprechende Alghorithmen sind in der liblo-0.25 Bibliothek implementiert. Da OSC-Dienste im weiteren Verlauf der Initialisierungsphase via Zeroconf veröffentlicht werden, können diese auch ohne festgelegte Portnummer gefunden und angesprochen werden.

[59] http://www.illposed.com/software/javaosc.html
[60] vgl. http://www.iana.org/assignments/port-numbers

6.2.3 Avahi - Zeroconf Implementation

Wie in 4 beschrieben, soll ein *Zeroconf* Netzwerk aufgebaut werden, um die *Service* und *Device Discovery* Funktionalität zu realisieren. Unter Linux verwende ich *avahi* von Lennart Poettering (Universität Hamburg).

Veröffentlichung Um die OSC-Controller zu registrieren, muss der *avahi-daemon* gestartet werden und folgende Veröffentlichungs-Prozedur durchlaufen werden:

- einen neuen *avahi-client* starten, der Zustandsänderungen abonniert (via Callback-Funktion).
- läuft der Server (`AVAHI_SERVER_RUNNING`) wird eine neue `entry group` erstellt
- Services werden hinzugefügt (`avahi_entry_group_add_service()`)
- und die `entry group` wird endgültig übergeben (`avahi_entry_group_commit`)
- Änderungen an den Einträgen werden abonniert

Alternativ zu der Verwendung der avahi-Bibliotheken können Services auch mit den mitgelieferten Kommandozeilen-Werkzeugen veröffentlicht werden:

```
avahi-publish [options] -s <name> <type> <port> [<txt ...>]
```

Im konkreten Fall für einen OSC-Server:

```
avahi-publish -s oscserver _osc._udp 7770
```

Kollisionsvermeidung Bei dieser Vorgehensweise können Kollisionen auftreten, wenn zwei Controller versuchen, Services mit gleichem Namen zu veröffentlichen. Für diesen Fall (`AVAHI_SERVER_COLLISION`) bietet die avahi-Bibliothek eine automatische Namensänderung an (dem Service wird eine fortlaufende Nummer angehängt) die hier auch verwendet wird.

Discovery Um Services im Netzwerk aufzufinden, kann dann wie folgt vorgegangen werden:

- eine Liste von Suchdomänen kann angefordert werden (`avahi_domain_browser_new()`)
- spezielle Dienste in einer Suchdomäne lassen sich auffinden mit `avahi_service_browser_new()`

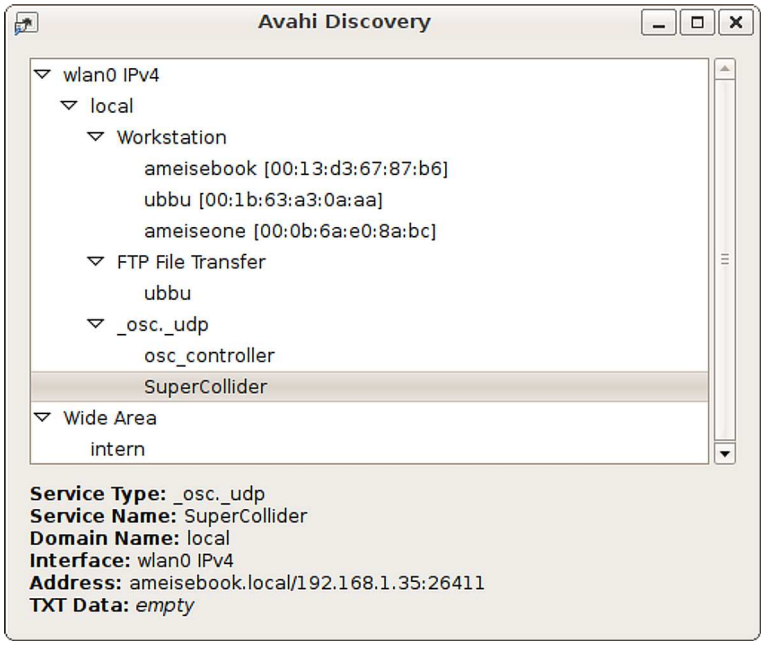

Abbildung 36: Avahi Discovery Screenshot im lokalen Netzwerk, es sind drei Workstations, ein FTP-Server und zwei OSC-Anwedungen sichtbar

- Daten gefundener Dienste können angefordert werden (`avahi_service_resolver_new()`)

Als zusäzliche Tools gibt es ebenfalls Kommandozeilen-Werkzeuge z.B. zur Verwendung in Shell-Skripten und grafische Oberflächen zum Anzeigen von Diensten (siehe 6.2.3) Auch andere OpenSource Anwendungen lassen sich so mit avahi auffinden. So gibt z.B. auch der SuperCollider-Server seinen Dienst, wie in der Abbildung zu sehen im Zeroconf-Netzwerk bekannt.

Teile dieser Initialisierung können in regelmässigen Abständen wiederholt werden um den Controller auf einem aktuellen Stand zu halten (Synchronisation, Discovery).

6.2.4 OSC-DMX512 Gateway

Vorrangehend wurde skizziert, wie ein geeigneter OSC-Server mit Open Source Bibliotheken umgesetzt werden kann. Im speziellen Fall, eines Controllers für Light-Motes mit DMX512 Schnittstelle muss dieser OSC Nachrichten an die DMX512 Schnittstellen-Hardware (z.B. USB-2-EIA485 Konverter) weiterleiten, also die Funktionalität eines OSC-to-DMX512 Gateways übernehmen. Ein ähnliches Programm gibt es z.B. für Apples CoreMIDI Soundkarten Treiber-API: „occam“ [61]. Dieses wandelt OSC-Nachrichten in MIDI-

[61] http://www.illposed.com/software/occam.html

Befehle um und leitet sie an die darunterliegende Hardware-Abstraktions-Schicht weiter. Der zugehörige OSC-Namensraum wurde dabei von den Entwicklern wie folgt gewählt:

```
/osc/midi/out/noteOn   [channel] [key] [velocity]
/osc/midi/out/noteOff  [channel] [key] [velocity]
```

für DMX512 ist folgender Namensraum geeignet (wobei DMX512 Eingangsignale zunächst keine Rolle spielen):

```
   /dmx/out ,ii [chn] [val]

TX:   /dmx/in
RX:   /dmx/in ,ii [chn] [val]
```

Diese Methoden müssen serverseitig an die USB-to-DMX512 Hardware weitergeleitet werden. Hier bietet sich der Einsatz der *dmx4linux*-Treiber-Suite[62] an, ein Projekt, welches von den HAW-Hamburg Absolventen Dirk Jagdmann und Micheal Stickel geleitet wird und u.A. quelloffene Treiber für OpenDMX (also auch Arduinos) Geräte anbietet. Dmx4linux verwendet die Linux Geräteverwaltung *udev* über die beim Einstecken („hotplug“) eines DMX512-Konverters automatisch eine virtuelle Gerätedatei (verb+/dev/dmx+) angelegt wird, welche - wie unter Unix üblich - über File-Deskriptoren aus Anwendungen heraus angesprochen werden kann. DMX512 Daten können so, unabhängig von der tatsächlich verwendeten Schnittstellen-Hardware (Hardwareabstraktionsschicht) ausgegeben werden. Das OSC-Gateway Programm öffnet also einen Netzwerk Socket für eingehende OSC-Nachrichten und gleichzeitig eine Gerätedatei an die die entsprechenden Parameter weitergeleitet werden.

Schwierigkeiten Es zeigte sich, dass dmx4linux derzeit noch kein RDM unterstützt und es weiterhin noch Probleme mit der OpenDMX bzw. OpenRDM Hardware gibt. Auch alternative Linux-Treiber für diese Hardware[63] konnten zwar erfolgreich in einer Testumgebung zum Senden jedoch nicht zum Empfang von DMX512 Paketen eingesetzt werden. Eine Quelltextrezension ergab, dass beide auf dem gleichen Code aufbauen, welcher von den offiziell im Linux-Kernel enthaltenen FTDI-USB-to-serial Treibermodulen abstammt. Diese Kernel-Module stellen via `/dev/ttyUSB/` eine virtuelle serielle Schnitstelle zur Verfügung. Theoretisch könnte diese auch direkt für DMX512 und RDM Kommunikation konfiguriert werden - und quelloffene openRDM-Implementationen (vgl. Hölscher (2008)) unter Windows funktionieren auch so - doch sind leider nicht alle notwendigen Funktionen ansprechbar. Zur Lösung dieses Problems kann eine weitere, quelloffene und z.B. unter Ubuntu-Linux über offizielle Paketquellen verfügbare Bibliothek herangezogen werden: *libftdi*[64]. Libftdi hat eine eigene Codebasis (wenn auch nicht so gut strukturiert wie die offziellen Kernel-Treiber) und kommuniziert mit der Hardware aus dem *Userspace* über die

[62] http://llg.cubic.org/dmx4linux/

[63] http://www.opendmx.net, http://www.erwinrol.com/index.php?opensource/dmxusb.php

[64] http://www.intra2net.com/de/produkte/opensource/ftdi/

offizielle *libusb*-Bibliothek. Dies hat u.A. den Vorteil, dass kein spezielles Wissen über Kernelprogrammierung notwendig ist und sämtliche Aufrufe mit User-Rechten ausgeführt werden können. Um libftdi für DMX512-Kommunikation einzusetzen, musste eine zusätzliche Funktion implementiert werden, welche der Hardware explizit den Befehl gibt den `BREAK`-Zustand (Sendeleitung auf LOW-Pegel) einzunehmen (siehe A). Funktionen zum Schreiben und Lesen von Zeichen und zum Einstellen der entsprechenden Baudrate etc. konnten übernommen werden. Eine Empfehlung an die libftdi Entwickler erwirkte die Aufnahme dieser Methode in die nächste Version.

6.2.5 HTTP-Server

Für die Anbindung an ein User-Interface-Client wird ein HTTP-Server benötigt, der in der Lage ist, dynamische Inhalte zu transportieren. Mehrere quelloffene Lösungen sind verfügbar, wobei der weit verbreitete *Apache* Server in diesem Fall zu umfangreich und nicht ohne Probleme embeddable ist. Alternativ stelle ich hier zwei Pakete vor, die ich getestet habe und die geeignet sind:

Lighttpd *Apache*, der beliebteste Open Source Web Server gilt nicht als besonders schlank. Daher sind zahlreiche Open Source Projekte gestartet worden, um Alternativen anzubieten. Darunter hat es *Lighttpd* derzeit an die 4. Position der *Netcraft Webserver Statistik* geschafft[65], kann also als ausgereift, robust und getestet gelten. Lighttpd ist als HTTP-Server für Controller geeignet und ist für die meisten Open Source Betriebssysteme und Linux-Distributionen bereits als vorkonfiguriertes Paket erhältlich und einfach zu installieren. Darunter auch *OpenWRT*, als Beispiel für ein Betriebsystem für eingebettete Systeme. Als lokale Testinstallation für das Deployment von User-Interface-Beispielimplementationen war lighttpd hilfreich.

Eine Qualität von lighttpd ist die Performanz: vor allem das „faster“ *Fast-CGI* Modul [66], erlaubt einfache Client->Server Kommunikation (z.B. mit in C-geschriebenen OSC-Gateways). Dabei wird nicht - wie bei ordinären CGI-Aufrufen - für jede Anfrage ein neuer Prozess gestartet, sondern ein einzelner Prozess nimmt alle Anfragen entgegen. Dadurch werden fork-Aufrufe und Speicher-Allozierung vermieden und so Rechenzeit eingespart. Dies ist bzgl. Echtzeitverhalten in einer interaktiven Ausstellung ein wichtiger Aspekt, um die Latenz des Systems niedrig zu halten. Realisiert wird dies bei Fast-CGI durch die Einbindung einer `fcgi_stdio` Bibliothek, die die Standard `stdio` Funktionen überschreibt. In einem „Super-Loop“ werden dann kontinuierlich Anfragen des HTTP-Clients ausgewertet (via `getenv()`) und Daten (via Ausgabe an `stdio` mit `printf()`) zurückgegeben.

[65] http://survey.netcraft.com
[66] http://www.fastcgi.com/

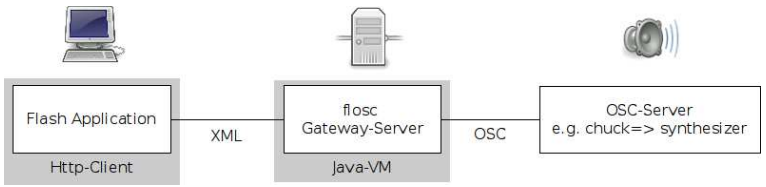

Abbildung 37: flosc Client-Gateway-Server Kommunikations-Schema

Jetty *Jetty*[67] ist ein Open Source[68] HTTP-Server und Servlet-Container für Java. Jetty lässt sich in vorhandene Software integrieren, um diese mit Webserver-Funktionalität zu erweitern, kann aber auch stand-alone betrieben werden. Der Fokus der Jetty Entwickler liegt dabei auf Einfachheit, Effizienz, Skalierbarkeit und Erweiterbarkeit. Für das hier vorgestellte System ist Jetty im Gegensatz zu anderen Java-basierten HTTP-Servern speziell geeignet, da es auch auf eingebetteten Geräten läuft (via J2ME bzw. CVM[69]), einfach in ein JavaOSC-Server integrierbar ist und zusätzlich sehr gut asynchrone Kommunikation zwischen Client und Server unterstützt. Hier z.B. auch in Zusammenarbeit mit dem ECMA-Script Framework *Dojo* und der Erweiterung *cometd*, welche die sog. *Server push*-Technologie via *asynchronous long polling* (vgl. Gross (2006) ermöglicht. Das User-Interface kann so ausgehend von dem Controller aktualisiert werden, wenn ein Zustand sich ändert. Java im Allgemeinen hat hier den Vorteil der Portierbarkeit und des ausgereiften Paket-Managements.

6.2.6 HTTP-OSC Gateway

Neben den zahlreichen Implementation von OSC ist *flosc* [70] für die hier vorgestellte Anwendung interessant: Flosc ist ein Java Gateway Server, der vom Web-Browser via TCP gesendete XML-Nachrichten in OSC Pakete umsetzt und über UDP an entsprechende OSC-Server im Netzwerk weiterleitet und vice versa. Flosc kommuniziert über die *Adobe Flash XMLSocket* Schnittstelle. Flosc akzeptiert mehrere simultane Eingangs- und Ausgangsverbidungen sowie Multicast Nachrichten an mehrere Flash-Clients.

Das Verpacken von OSC-Paketen in XML-Notation geschieht dabei auf Basis folgender DTD [71]:

```
<!ELEMENT OSCPACKET (MESSAGE+)>
<!ATTLIST OSCPACKET
            ADDRESS CDATA #REQUIRED
            PORT CDATA #REQUIRED
            TIME CDATA #REQUIRED
```

[67] http://www.mortbay.org/jetty-6/
[68] Apache 2.0 Lizenz, freigegeben für kommerziellen Gebrauch und Vertrieb
[69] Java Virtual Machine und SDK für mobile Endgeräte
[70] http://www.benchun.net/flosc/
[71] Document Type Definition

```
>

<!ELEMENT MESSAGE (ARRAY | ARGUMENT)* >
<!ATTLIST MESSAGE
            NAME CDATA #IMPLIED
>

<!ELEMENT ARRAY (ARRAY | ARGUMENT)*>
<!ELEMENT ARGUMENT EMPTY>
<!ATTLIST ARGUMENT
            TYPE (i|f|h|d|s|T|F|N|I) "i"
            VALUE CDATA #IMPLIED
>
```

Flosc wird hier als Schnittstelle zwischen User-Interface (HTTP) und Controller (OSC) verwendet mit der Option, die XML-Repräsentation der Daten auf das einfachere *JSON* umzustellen (siehe 5.2.3).

6.2.7 Mapping

Um Komponenten des Systems untereinander zu verschalten, sind Mapping-Controller notwendig. In anderen Zusammenhängen in der Netzwerktechnik auch als *Router* bezeichnet. Diese halten sog. *Routing-Tabellen* vor, die angeben, welche Datenquellen mit welchen Senken verbunden sind, d.h. zu welchen Knoten Datenpakete gesendet werden sollen. Aufgrund der angestrebten *Peer-to-Peer* Topologie (siehe 19), werden diese dezentral abgelegt: jeder Controller speichert eigene Maps bzw. Tabellen für seine Quellen. Im einfach Fall wird z.B. der Ausgang eines Sensor-Motes auf den Eingang eines Light-Motes gemapped: gemessene Werte, werden durch Lichtintensität oder Farbe eines Pixels dargestellt. Durch Einsatz vieler Knotenpunkte und zusätzlicher Module zur Skalierung und Filterung von Werten sind so z.B. Dot-Matrix-Mappings denkbar oder Verknüpfung mit Audio-Motes. Hierfür stellt jeder OSC-Server im System die Methode

```
/map ,ssss [src_adr] [src_path] [dest_adr] [dest_path]
```

zur Verfügung.

`[src_adr]` = Zeroconf Bezeichner der Quelle
`[dest_adr]` = Zeroconf Bezeichner der Senke
`[src_path]` = OSC Pfad Quelle
`[dest_path]` = OSC Pfad Senke

Über das User-Interface könnte so beispielsweise die Methode:

```
/map ,ssss oscserver#1.local /sensor/1/state oscserver#2.local /pixel/1/r
```

via Broadcast gesendet werden, woraufhin der per Zeroconf als erste OSC-Server konfigurierte und ansprechbare Controller als Ziel für Werte seines ersten Sensoreingangs den zweiten OSC-Server und dort den roten Kanal des ersten Pixels einträgt. Analog dazu der Aufruf

```
/unmap ,ssss [src_adr] [src_path] [dest_adr] [dest_path]
```

zum löschen einer Route.

Dabei können Routing-Tabellen auch redundant gespeichert werden, um höhere Ausfallsicherheit zu gewährleisten. Hierzu ist lediglich ein eingenständiger OSC-Server notwendig, der nur auf Mapping-Nachrichten reagiert und diese auf Anfrage, rekonstruiert also erneut sendet.

6.2.8 Initialisierungs Phase

Wird ein Controller in das System eingefügt, durchläuft er eine Initialisierungsphase:

- Betriebssystem starten
- Hardwareerkennung
- Kernel-Module (Hardwaretreiber) laden
- Zeroconf-Daemon starten
- Device Discovery
- interne Dienste starten (Mapper, Router, Gateways, OSC-Module)
- Dienste veröffentlichen (HTTP, OSC)
- Zeit synchronisieren bzw. eigenen Zeitserver starten (siehe 4.2.6)

6.2.9 Alternativen

Alternativ könnte auch ein Controller eingesetzt werden, auf dem *PureData* oder *MAX/MSP* läuft. Diese Programme könnten ebenfalls automatisch gestartet werden und implementieren bereits teilweise einige Funktionen wie z.B. Mapping und Routing von OSC-Nachrichten. Es bestehen bereits einige Gateways zwischen diesen Programmen und z.B. seriellen Schnitstellen, die für das Umsetzen von OSC-Nachrichten in DMX512-Nachrichten verwendet werden könnten. Als User-Interface könnten die bestehenden grafischen Oberflächen benutzt werden und z.B. mit „*X-Forwarding*“ über SSH oder *Remote Desktop* im System verteilt werden. Diese Lösung ist jedoch erfahrungsgemäß zu komplex für Anwender ohne computertechnischen „Background„.

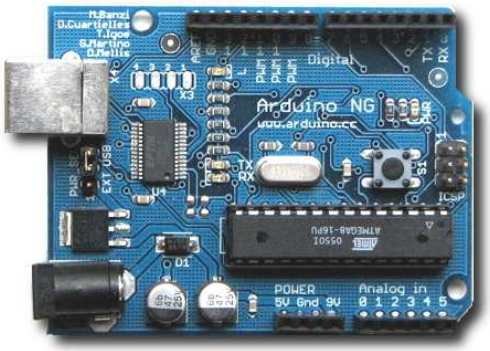

Abbildung 38: ArduinoNG Experimentierplattform auf Atmega168 Basis

7 Motes

Bei den hier vorgestellten *Motes* handelt es sich um Ausgabeknoten für Licht und Klang und Eingabeknoten für analoge und digitale Sensoren. Der Schwerpunkt liegt hier auf den *Light-Motes* und *Smart-Pixel*. In Installationen und Produkten stellen sie die wahrnehmbaren Elemente dar, die dem Publikum zugänglich sind. Sie liefern das visuelle Feedback in einer interaktiven Umgebung. Sie stellen grundlegende Methoden zur Verfügung, aus denen durch komplexes Verschalten bzw. *Patchen* die künstlerische Idee umgesetzt wird. Derzeit gibt es de facto eine Standard-Hardwareplattform im Bereich *Physical Computing* und interaktiver elektronischer Kunst, die für diese Anwendung prädestiniert ist und im folgenden vorgestellt wird.

7.1 Arduino - Hardware Plattform

Aufbauend auf der *Processing* Entwicklungsumgebung (vgl. Sukale (2008)) konstruierte Hernando Barragán 2003 am *Interaction Design Institute Ivrea*, Italien zum Erproben von interaktiven elektronischen Designs ein relativ kleines Atmel-MCU-basiertes IO-Board *Wiring*. Zur dieser Familie gehörend und auf der Wiring-Software basierend entwickelte sich daraus das internationale Projekt *Arduino* [72] . Arduinos sind kleine Mikroprozessorsysteme bestehend aus 8-Bit Atmega-Controller (Atmega88 oder Atmega168) und entsprechender Software zum einfachen Entwurf und zur Programmierung von Firmware (sog. *Sketches*). Der Referenz-Schaltplan ist unter der GPL veröffentlicht und die IDE ist ebenfalls quelloffen. Arduinos und den Vorläufer Wiring gibt es in verschiedensten Varianten: als *Mini* oder *Nano*, mit integrierter USB-Schnittstelle, kabellos mit Bluetooth-Modul, als Do-It-Yourself-Steckbrett-Projekt und in der aktuellen Standard-Version *Diecimila*. Arduinos werden z.B. am Department Design der HAW-Hamburg für die Konstruktion textiler Interfaces (siehe 2.2)eingesetzt und im UbiComp-Lab für sog. *Smart Shirts*. Überall dort, wo Studenten und

[72] http://arduino.cc

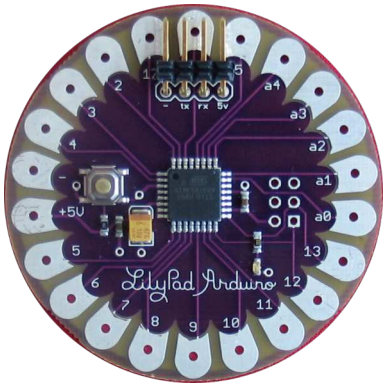

Abbildung 39: Lilypad Arduino: integrierbar in Textilien, von Leah Buechley, Department of Computer Science, University of Colorado

Kunstschaffende ohne tiefer gehende Programmierkenntnisse schnell und einfach *Physical Computing* Projekte realisieren wollen, erfreut sich die Arduino-Plattform großer Beliebtheit. Um meine Lösung in diesem Kontext zu etablieren, soll die von mir entwickelte Hardware ebenfalls Arduino-kompatibel sein und die Software zur weiteren Verwendung als Bibliothek für Arduino zur Verfügung gestellt werden. Mein Projekt kann dadurch von der Offenheit und großen Beliebtheit des Arduino-Projektes profitieren, obwohl angemerkt sei, dass andere Hardware-Designs evtl. leistungsfähiger und kostengünstiger wären (Controller mit integriertem DSP, 16 oder 32-Bit Controller). Arduinos entsprechen meinen Anforderungen:

- **Open Source Software** Entwicklungsumgebung und Firmware unter GPL2
- **Open Hardware** Referenzdesign, Schaltpläne und Layout sind offen
- **embeddable** SMD-Gehäuse für die verwendeten Bauteile sind erhältlich
- **portierbar** Atmega-basierte Hardware sollte grundsätzlich mit Arduino-IDE und firmware laufen, auch andere Designs sind möglich: z.B. mini-Arduinos, Arduinos auf flexiblen Leiterplatten, Arduinos integriert in DMX-Dimmer Hardware
- **automatisierbar** Atmel Controller starten eigenständig nach einem Reset, kein Betriebsystem muss „hochgefahren“ werden, erfahrungsgemäß ein großer Vorteil gegenüber ARM oder x86-basierten Designs bei der Konstruktion robuster Geräte
- **simple** einfach zu bedienen, günstig zu beschaffen
- **erweiterbar** durch eigene Bibliotheken und Hardware-Add-Ons

Aufbau und Größe unterscheidet sich nicht wesentlich von den *Berkeley Motes*, von denen einige Referenzdesigns ebenfalls auf Atmel 8-Bit Prozessoren (Atmega128) aufbauen.

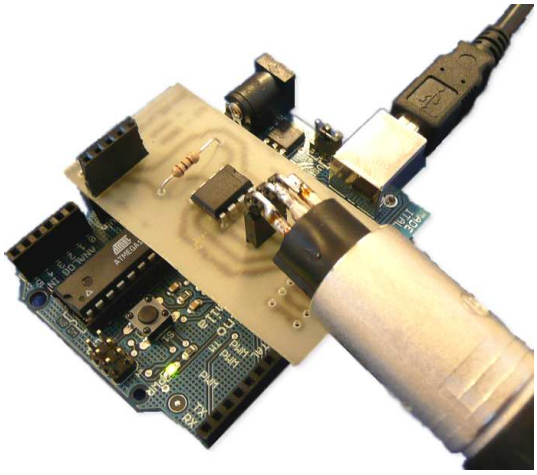

Abbildung 40: Arduino DMX Shield, Quelle: HC Gilje, Bergen National Academy of the Arts

Es liesse sich auch das Berkeley-Mote Betriebsystem TinyOS auf Arduino portieren, was hier aber nicht notwendig ist.

Softwareseitig lässt sich die Arduino Firmware und Entwicklungsumgebung einfach mit (in C++ geschriebenen) eigenen Bibliotheken erweitern.

Hardwareseitig ist es möglich, durch das offene Design und die, bei den meisten Varianten vorgesehenen Platinensteckverbinder sog. *Shields* zu entwickeln, die den Arduino um zusätzliche Funktionen erweitern (z.B. externer Speicher, Schnittstellen-Bausteine 40, Treiber ICs, Funk-Module, GPS-Module, etc.).

7.2 Light-Motes

Wie einleitend erwähnt, soll hier Stanislaw Lem's Vision von *Synsekten* aufgegriffen werden und deren Umsetzung als sog. *Light-Motes* bzw. synthetische Glühwürmchen erörtert werden[73]. Ausblickend stelle ich mir diese in miniturisierter Form als *Smart-Pixel* vor, die bestenfalls - bis auf Licht und Farbe - „unsichtbar" und autonom im (Austellungs-)Raum umherschwirren und dabei individuell angesprochen werden können.

7.2.1 Anforderungen

Es müssen folgende Aufgaben erfüllt werden:

[73] Mote sollte hier übrigens nicht mit dem deutschen Wort „Motte" verwechselt werden, sondern ist eine Abkürzung von „remote", ein entferntes System

- Ausgänge für mehrere LEDs (beispielsweise rot, grün, blau) oder ganze LED-Cluster zur Verfügung stellen
- LEDs dimmen zur Regelung der Lichtintensiät und zur Farbsynthese
- Schnittstellen zur Kommunikation mit einem Controller zur Verfügung stellen
- Stand-Alone Farb- und Helligkeitsverläufe generieren oder zumindest letzten Zustand halten können
- evtl. Firmware-Aktualisierungen ermöglichen (z.B. zur Wiederverwendung in einer anderen Austellung)

Es sollen - je nach gestalterischem Konzept - beliebig viele Motes miteinander verschaltet werden können (*Skalierbarkeit*).

Idealerweise wäre die *Granularität* des Light-Mote Systems sehr hoch, um eine optimale Auflösung der imaginären Dot-Matrix zu erzielen, z.B. bei einem Aufbau wie in 2.1: Ein Feld aus Farb-Licht-Objekten. D.h. es muss hier speziell auf Kosteneffizienz geachtet werden, um z.B. große Pixel Felder oder Wolken zu realisieren. Nach Möglichkeit sollten sie einfach aufgebaut werden können, um auch in studentischen Projekten eingesetzt werden zu können (gemäß der Devise: „keep it simple“). Ähnliche Projekte im Kontext von *Street-Art* [74] haben relativ „unintelligente“ LED-Pixel-Aufbauten entwickelt (sog. *LED-Throwies*, siehe 41) bei denen der Preis eines Pixels mittlerweile bei nur 0.25 USD liegt, und so eine sehr hohe Granularität möglich wird.

Für die hier erwähnten Projekte reicht eine gröbere Auflösung von z.B. 50 Pixeln (2.1). Es muss aber zusätzlich möglich sein, die Lichtstärke eines Pixels (z.B. in Form einer Leuchtplatte aus Glas) durch das Zusammensschalten und gleichzeitige Ansteuern mehrerer LEDs zu erhöhen, entsprechende Ausgangstreiber sind nötig.

7.2.2 Umsetzung

Mit LEDs und aktuellen Mikrocontrollern ist eine Umsetzung der o.g. Vision annähernd möglich. Viele kommerzielle Lösungen (RGB-LED-Dimmer) sind bereits erhältlich. Im Veranstaltungstechnik-Bereich zudem meistens mit DMX512-Schnittstelle zum Ansprechen über Lichtsteuerpulte. Die meisten kommerziellen Geräte entsprechen jedoch nicht meinen Anforderungen (nicht quelloffen, unflexibel, zu teuer für studentische Projekte etc.). Die Hardware ist aber zumeist sehr simpel und fast baugleich mit Arduinos. Dem Arduino fehlen nur die DMX512 (bzw. EIA-485) Schnittstelle und die Leistungsschalter an den

[74] Streetart: Straßenkunst, allgemein die Bezeichnung für künstlerische Aktivitäten auf offener Straße (Brockhaus)

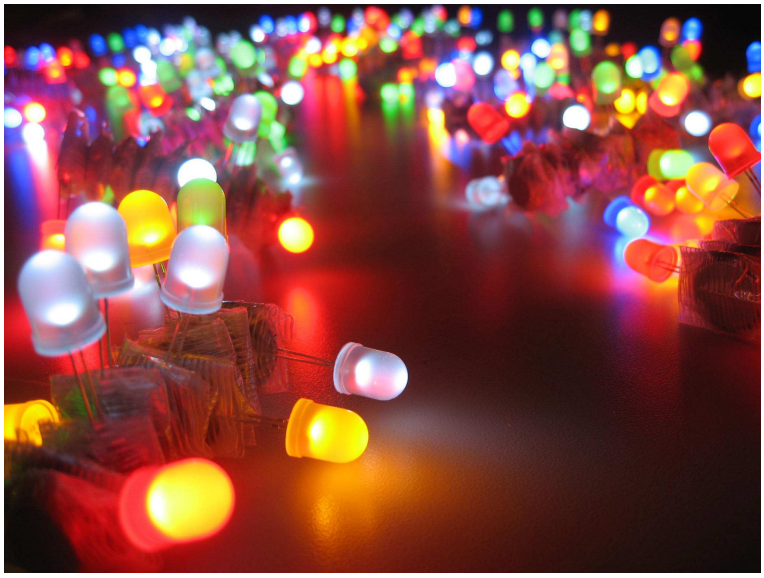

Abbildung 41: LED Throwies, Beispiel einer Pixel Wolke mit relativ hoher Granularität, Quelle: grafittyresearchlab.com, Eyebeam R&D OpenLab, USA, Bild veröffentlicht unter Creative Commons Lizenz

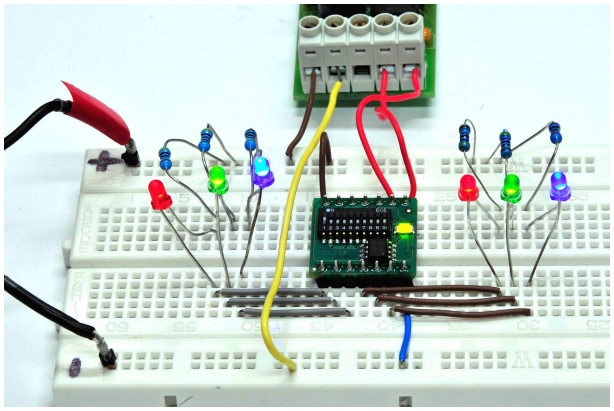

Abbildung 42: Beispiel für Light-Motes: ”DMX4All Baby Dimmer” auf Steckboard mit sechs PWM-Ausgängen und RGB-LED, Quelle: Daniel Hausig, HBK-Saar

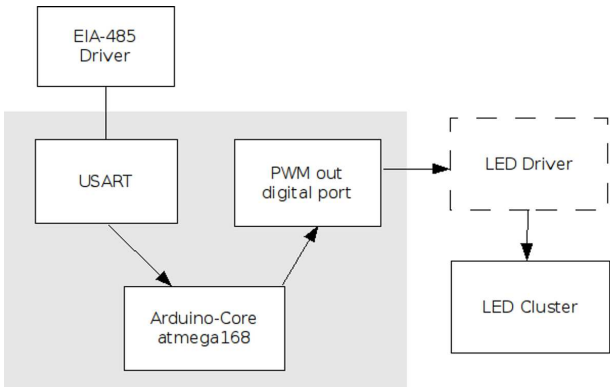

Abbildung 43: Exemplarischer Aufbau Light-Mote auf Basis des Arduino Referenzdesigns und mit DMX512 (EIA-485) Schnittstelle

Ausgängen. Hier könnte man ein Arduino-Shield (siehe 40) verwenden mit dem zusätzlichen EIA-485 Treiberbaustein und MOSFET-Ausgängen auf einem Steckbrett (vgl. Ness (2006)). Zur besseren Integrierbarkeit stelle ich hier eine Lösung vor ohne Steckbrett oder Shield, bei der die notwendigen Komponenten auf einer einzigen industriell gefertigten Platine untergebracht sind.

Technische Daten

- 8-Bit Atmega168 Prozessor (Arduino kompatibel)
- bidirektionale DMX512/RDM Schnittstelle
- sechs leistungsstarke dimmbare Ausgänge
- ISP-Header für „in-circuit“ Programmierung
- großer Eingangsspannungsbereich (5V-30V) durch leistungsstarke aktive Regelung
- kleine Abmessungen, SMD-Technik (unsichtbar, integrierbar)
- Standard-Verbinder (5-pol XLR, Pinbelegung nach DIN56930-2, MICA-6 Flachbandkabel Steckverbinder)

Diese Light-Motes lassen sich mit der Arduino Version des Atmel-STK500 Bootloader vorprogrammieren, sind also über die serielle Schnittstelle bzw. ein USB-seriell Konverter ohne zusätzliche Programmiergeräte via Arduino-Entwicklungsumgebung und mitgeliefertem avrdude[75] beschreibbar.

[75] Open Source Programmiersoftware für Atmel Mikrocontroller http://www.nongnu.org/avrdude/

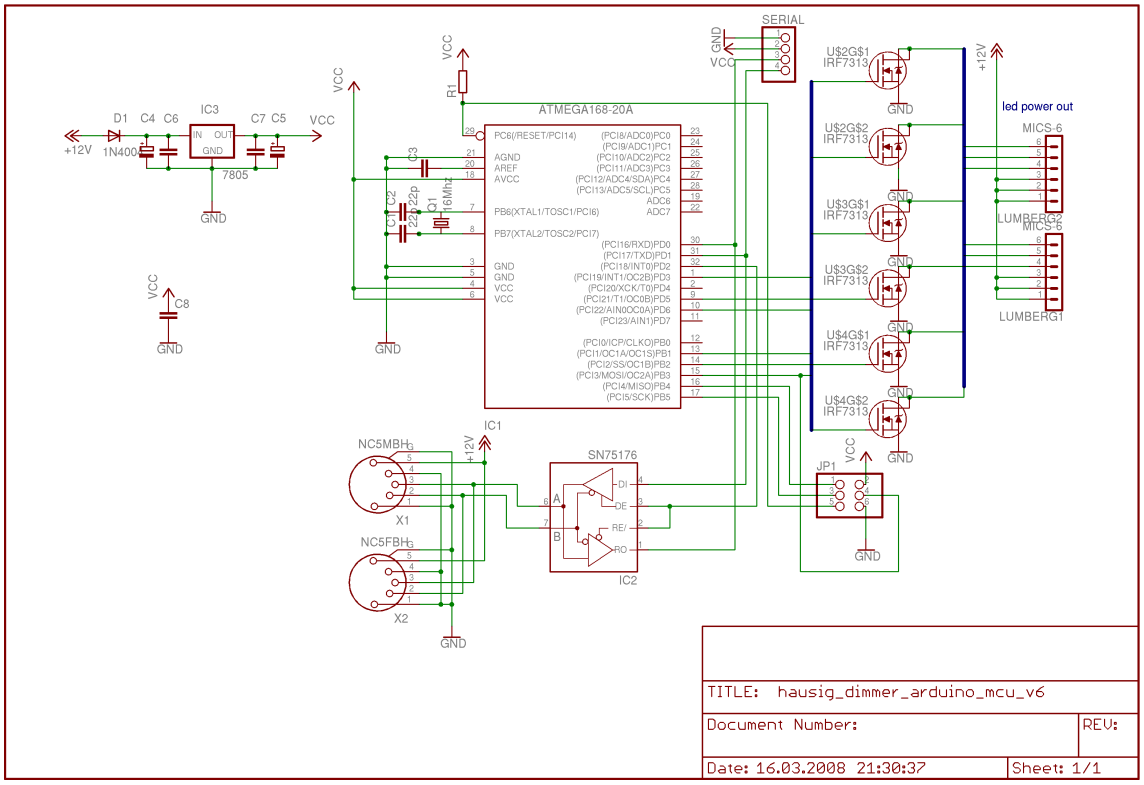

Abbildung 44: Arduino-Light-Mote Schaltplan

Aufbau Der Dimmer kommuniziert über DMX512/RDM. Ein EIA-485 Pegelkonverter (SN75176) übernimmt „on-board“ die Anbindung an die serielle Schnittstelle des Controllers. Dieser Baustein ist so verschaltet, dass die MCU diesen auch auf „transmit“ umschalten kann. Das Gerät kann also auch als Basis für DMX512-Sender bzw. RDM-Responder dienen.

Sechs SMD-MOSFETs bzw. drei Dual-HEXFETs (IRF7313) verstärken die Schaltleistung an den Ausgängen des Controllers auf 2.5A Dauerstrombelastung bei max. 30V. Diese können direkt an der Ausgangsspannung des Atmegas betrieben werden (R[Drain-Source] = 0,050 Ohm bei V[Gate-Source] = 4,5V). Eine handelsübliche LED (z.B. OSRAM TopLED) benötigt eine Stromstärke von ca. 20mA, es könnten im Idealfall 750 solcher LEDs an dem Dimmer betrieben werden. Dies ist ausreichend, um relativ lange LED-Stripes oder andere große LED-Cluster zu betreiben.

Firmware Die gewünschten Funktionen sind in C umgesetzt und wurden mit *avr-gcc* kompiliert. Eine einfache *State-Machine* übernimmt innerhalb der *Interrupt Service Routine* für Empfang serieller Daten die Auflösung des DMX512 Protokolls. Für jeden Abschnitt im DMX512-Frame (siehe 21) wird ein Zustand definiert, welcher bei Empfang eines

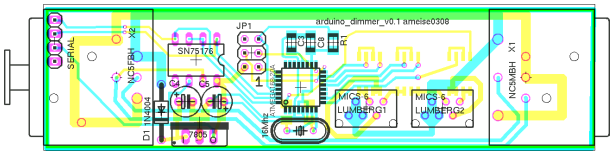

Abbildung 45: Arduino-Light-Mote Platinenlayout

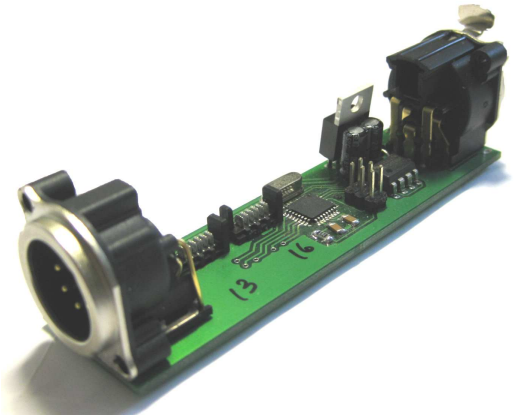

Abbildung 46: Arduino-Light-Mote Prototyp mit typischen XLR-Steckverbindern

gültigen Zeichens in den nächsten bzw. bei Empfang eines ungültigen Zeichens in den Ausgangs-Zustand (`IDLE`) überleitet.

Nach Empfang des gültigen DMX512-Startcodes (`0x00`), werden die DMX-Werte für jeden Kanal einzeln nacheinander als Bytes empfangen. Wird an dieser Stelle in der erweiterten Firmware-Version ein RDM-Startcode (`0xCC`) erkannt, springt das Programm in die State-Machine zur Auswertung von RDM-Pakten (siehe 4.2.5). Empfängt das Programm gültige DMX512-Werte, werden diese in einen vorallozierten Speicherbereich kopiert und ein globales Flag gesetzt (`EVAL_DMX`). Im Main-Loop kann dann auf neu eingetroffene Daten reagiert werden und z.B. die entsprechenden Register der sechs PWM-Generatoren des Atmega168 aktualisiert werden, woraufhin die LEDs mit entsprechender Intensität leuchten. Die DMX512-Startadresse kann dabei beliebig gewählt werden und wird im nicht flüchtigen EEPROM-Bereich des Speichers abgelegt. Über das RDM-Kommando `E120_DMX_START_ADDRESS 0x00F0` kann diese vom Controller aus konfiguriert werden.

7.2.3 Modulationstechniken zur Steuerung der Leuchtintensität von LEDs

LEDs können nicht, wie Glühbirnen über Variation der Strom- oder Spannungszufuhr gedimmt werden. Es würden hierbei unerwünschte Wellenlängenverschiebungen auftreten und zudem wäre das Dimmen nur in einem schmalen Bereich möglich. Alternativ werden

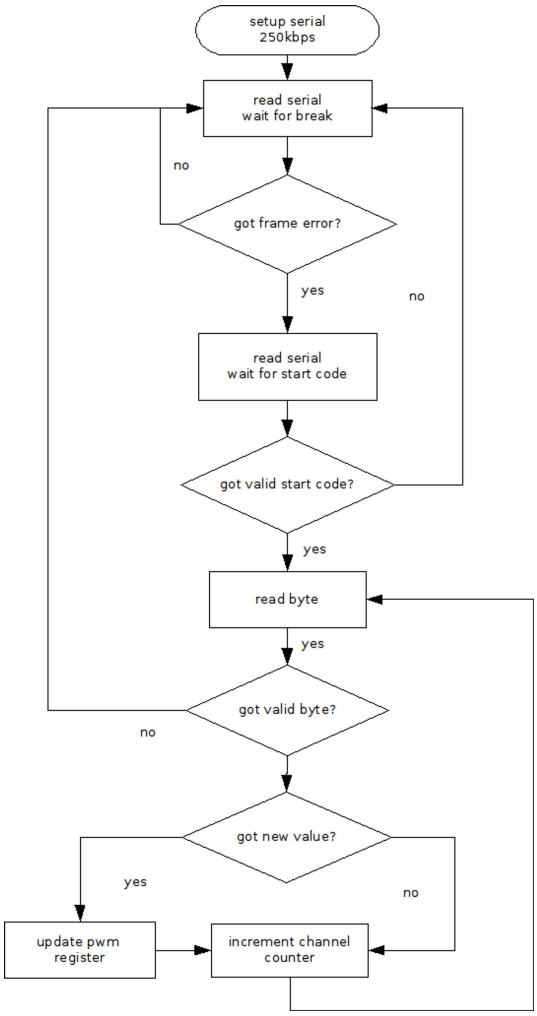

Abbildung 47: DMX512 Receiver Flowchart, Prinzip der (vereinfachten) DMX512 Empfangsroutine

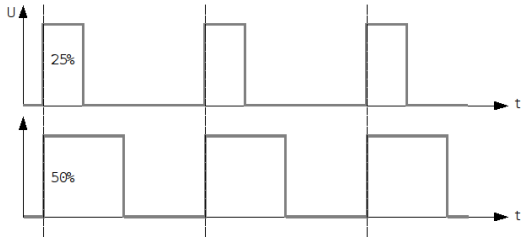

Abbildung 48: Prinzip der Pulsweitenmodulation

LEDs an einer Konstantstromquelle betrieben, die im eingeschalteten Zustand, auf optimale Leuchtkraft der LED ausgelegt ist. Durch schnelles Schalten, lässt sich über das Verhältnis von Einschaltzeit zu Periodendauer (*Tastverhältnis*) die Helligkeit abschwächen, da das menschliche Auge bei ausreichender Schaltfrequenz keine einzelnen Zustände mehr erkennen kann (siehe 2.3).

Pulsweitenmodulation *Pulsweitenmodulation* (PWM) ist die verbreiteste und einfachste Technik zum Dimmen von LEDs. Bei der vorgestellten Firmware für das Arduino-Board wird das direkt von der Hardware übernommen, die über sechs interne Timer, jeweils einen PWM-Ausgang schaltet. Ein Light-Mote ist so auf sechs Ausgänge mit einer Auflösung von 8-bit beschränkt: Maximal zwei unabhängige RGB-Module bzw. Smart-Pixel können betrieben werden. Die Ausgänge werden dabei pro festgelegtem Zyklus (z.B. alle 4 μs) für eine über das Tastverhältnis vorgegebene Periode auf „HIGH“ geschaltet.

Ein Problem dabei ist, die lineare Ansteuerung der Ausgänge zwischen den Werten 0 und 255: Die menschliche Wahrnehmung von Lautstärke und Lichtintensität verhält sich aufgrund des sog. *Weber-Fechner*-Gesetzes in etwa logarithmisch zu der zugeführten Energie [76] . Im unteren Intensitäts-Bereich kommt es so bei diesem Verfahren zu einer Art Treppeneffekt: einzelne Zustände eines Helligkeitsverlauf sind als Flackern erkennbar. Eine Differenzierung im oberen Intensitäts-Bereich ist kaum möglich, hier wird also Auflösung verschwendet.

Eine Lösung wäre mit 12-Bit oder 16-Bit Timern zu arbeiten und über Tabellen den 8-Bit Eingangswertebereich auf eine annähernd exponentielle Ausgangskurve abzubilden. Hierfür stehen aber bei den low-cost Mikrocontrollern von Atmel derzeit nicht ausreichend Ressourcen zur Verfügung. Weiterhin werden bei dieser Technik alle Lasten (zumindest eines Mikrocontrollers) zur gleichen Zeit eingeschaltet, was zu asymmetrischer Belastung der Stromversorgung und ungewollten elektromagnetischen Störungen führen kann. PWM wird daher mittlerweile als wenig geeignet angesehen für die Ansteuerung von LEDs. Im folgenden seien drei Alternativen vorgestellt, die in Light-Motes eingesetzt werden könnten.

[76] Diese zuerst von E.H. Weber formulierte Annahme besagt, „je intensiver der Reiz, desto stärker muss der Zuwachs sein, um eine Unterschiedswahrnehmung zu bewirken.“ (vgl. Meyers Lexikon)

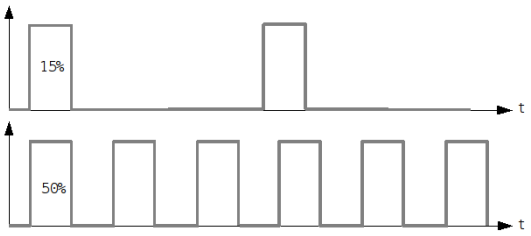

Abbildung 49: Prinzip der Frequenzmodulation

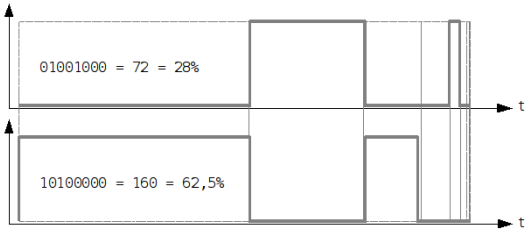

Abbildung 50: Prinzip der Bit Angle Modulation

Frequenzmodulation Bei der *Frequenz Modulation* (FM) ist das Einschaltsegment ein kurzer Puls fester Länge, der sich mit variabler Trägerfrequenz wiederholt. Dabei entsteht eine „Lücke“ zwischen zwei Einschaltsegmenten, die umgekehrt proportional zur vorgegebenen Lichtintensität ist (je länger, desto dunkler). Frequenzmodulation hat dabei gegenüber PWM den Vorteil des nichtlinearen Verhaltens bei der Ansteurung von LEDs und des nicht zeitgleichen Einschaltens der angehängten Lasten (durch die Phasenverschiebung der Kanäle zueinander).

Bit Angle Modulation Bei der *Bit Angle Modulation* gibt es pro Zyklus mehrere Schaltsegmente unterschiedlicher Länge. Die Anzahl dieser Blöcke entspricht dabei der Bitbreite des Eingangswertes (bei 8-Bit, acht Blöcke). Schaltsegmente können jeweils einen Schaltzustand (LOW oder HIGH) annehmen. Dabei ist der jeweils folgende Block doppelt so lang wie sein Vorgänger. Dies entspricht einer Gewichtung nach dem Binärsystem:

Bit Position	**relative Einschaltdauer**
8	128
7	64
6	32
5	16
4	8
3	4
2	2
1	1

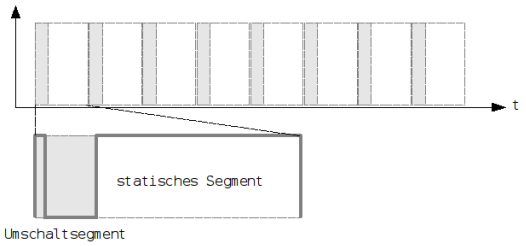

Abbildung 51: Prinzip der Lehmann Modulation

Bit Angle Modulation wird von der Firma Artistic Licence als optimale Modulationstechnik zur Ansteurung von LEDs vorgeschlagen (vgl.Howell (2002)): da die Eingangswerte direkt in das Ausgangs-Schaltmuster umgesetzt werden können, wird nur relativ wenig Rechenleistung benötig. Mit nur einem Hardware-Timer der MCU können so sämtliche verfügbaren Ports als Dimmerausgänge verwendet werden. Eine simple State-Machine könnte eingesetzt werden: Pro Block würde der Timer einen Interrupt auslösen und in der entsprechenden ISR würde das *Compare-Match*-Register des Timers für den nächsten Block initialisiert sowie die Zustände sämtlicher Ausgangs-Pins via Bit-Schiebe-Operation aktualisiert werden. Im Gegensatz zur normalen beispielsweise bei Arduinos eingesetzten PWM-Modulation, die pro Ausgang einen Timer verwendet, wesentlich effizienter.

Lehmann Modulation Um die Unzulänglichkeiten bzgl. Rechenleistung und linearer Ansteurung klassischer Modulationstechniken zu überwinden, sind verschiedene Abwandlungen und Kombinationen von den vorgestellten Verfahren erfunden worden. U.A. die sog. *Lehmann Modulation* Lehmann (2004) zum „weichen“ Dimmen von LEDs. Gegenstand der Patentschrift ist ein Verfahren, das bei geringer Prozessorauslastung eine hohe Auflösung ermöglicht.

Hier wird die Periode ebenfalls in mehrere Blöcke unterteilt, die wiederum in *Umschaltsegmente* und ein *statisches Segment* unterteilt sind. Während des statischen Segments, kann der Schaltzustand nicht verändert werden (wie bei der Bit Angle Modulation), so dass während dieser Zeit kaum Rechenleistung benötigt wird. Zur Verteilung der einstellbaren Einschaltdauer auf die Periodendauer sind jedem der 8-Bit breiten Eingangswerte über eine Tabelle Stellgrössen zugeordnet, aus denen das Programm ermittelt, welche der Blöcke einer Periode zu welchen Zeitpunkten innerhalb des Umschaltsegments eingeschaltet bzw. ausgeschaltet werden (siehe 7.2.3).

Leider ist die Lehmann Modulation durch ein Patent geschützt, kann also nicht in Open Source Projekten eingesetzt werden. Sie ist aber auch nicht die einzige Methode sanftes Dimmen von LEDs mit nur 8-Bit Datenbreite zu realisieren. Da es wohl kaum möglich sein dürfte, allgemein bekannte Verfahren wie PWM, FM und BAM zu patentieren, könnte man

z.B. auf Grundlage der Bit Angle Modulation eine Erweiterung entwerfen, die in den unteren 10 % des Regelbereiches exponentielles Verhalten aufweist, indem den acht statischen Blöcken ein einzelner Block variabler Länge nach- oder zwischengeschaltet wird (also quasi eine Kombination aus BAM und FM). Es bleibt abzuwarten, ob Techniken wie die Lehmann Modulation auch noch eingesetzt werden, wenn in den Light-Motes bzw. LED-Dimmern 16bit oder 32bit-Architekturen verwendet werden und somit auch ausreichend 16bit Timer verwendet werden können. Arduinos beispielsweise werden wohl früher oder später von ihren großen Brüdern den *„ARMuinos“* [77] überholt werden, die derzeit allerdings noch viel teurer sind und mehr Leistung verbrauchen. Eine weitere Alternative wären die von Atmel angebotenen 8-Bit MCUs mit spezialisierten PWM-Modulen zum Einsatz in der Lichtindustrie. Die AT90PWM-Familie kann z.B. PWM mit 12-Bit Genauigkeit generieren und bietet zusätzlich noch ein Verfahren zur Erhöhung der Auflösung an („Flank Width Modulation“). Letztere sind aber wiederum nicht universell einsetzbar, schwieriger zu beschaffen und nicht als Arduino-Experimentierplattform erhältlich.

7.2.4 OSC Namensraum

Light-Motes in der vorgestellten Ausführung mit zwei RGB-Ausgängen werden im OSC-Netzwerk (siehe 4.2.3)) durch zwei Pixel repräsentiert:

```
/pixel/{id}
/pixel/{id+1}
```

Er stellt im einfachsten Fall folgende Methoden zur Verfügung:

```
/pixel/{id}/rgb ,fff [r] [g] [b]
```

Setzt die Farbe bzw. das Tastverhältnis der PWM-Kanäle auf den entsprechenden Wert. Wobei der Wertebereich normalisiert ist auf [0.0-1.0] und vom Contoller in den entsprechenden DMX512-Wertebereich skaliert wird [0-255].

Sendet man `rgb` ohne Argumente, wird der aktuell eingestellte Farbwert zurückgegeben:

```
TX: /pixel/{id}/rgb
RX: /pixel/{id]/rgb ,fff [r] [g] [b]
```

Weiterhin können die Farb- und DMX512-Kanäle auch direkt abgesprochen werden:

```
/pixel/{id}/r ,f [r]
/pixel/{id}/g ,f [g]
/pixel/{id}/b ,f [b]

/dmx/{adr}/val ,i [0..255]
```

[77] ARMunio = ARM-RISC-CPU basierte 32-Bit Open Hardware Systeme wie z.B. der Make Controller: http://www.makezine.com/controller/

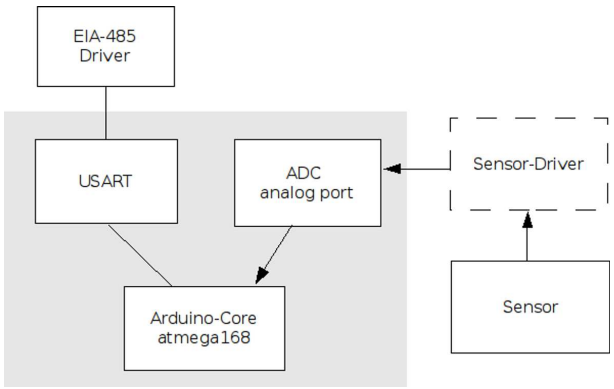

Abbildung 52: Exemplarischer Aufbau Sensor-Mote auf Basis des Arduino Referenzdesigns und mit RDM (EIA-485) Schnittstelle

7.3 Sensor-Motes

Anknüpfend an das breite Forschungsfeld der Sensornetzwerke, sei hier eine einfache Ausprägung eines solchen vorgestellt. Glücklicherweise muss im Kontext einer Kunstaustellung, kein großer Wert auf Sicherheit gelegt werden, wie z.B. bei lebensrettenden oder auch lebensbedrohenden Szenarien der militärischen Forschung oder beim Einsatz im Fahrzeugbau, wo sich ähnliche Konzepte wiederfinden.

7.3.1 Anforderungen

Hier sind zum Einen Hardware-User-Interfaces zu nennen, über die z.B. Tasten bzw. *digitale Sensoren* abgefragt werden sollen. Zum Anderen sind bei den meisten Projekten auch immer wieder *analoge Sensoren* vorgesehen, die verschiedenste dynamische Qualitäten des Austellungsraumes oder des Publikums erfassen können. Im einfachen Fall sollte z.B. der ”Stand-Alone”-betriebene Controller auf das Auftauchen von Publikum reagieren können, typischerweise mit dem Aktivieren des Austellungsobjektes. Er sollte von einem Knoten mit *PIR-Sensor* [78] einen Wert erhalten und bei überschreiten eines Schwellwerts entsprechend reagieren können. Im komplexeren Fall sollte der Controller am Besucher getragene Sensoren zur Erfassung von Parametern des menschlichen Organismus (Puls, Temperatur, Stichwort: *Body Monitoring*) auswerten können (siehe 2.2). Hierfür sind eingebettete tragbare Systeme zu konstruieren zur Sensordatenerfassung, Auswertung und Weiterleitung. Auch die Erfassung der Rotationsgeschwindigkeit beim dritten vorgestellten Szenario(2.3 ist eine Sensor-Anwendung.

[78] Passive Infrared Sensor, als „Bewegungsmelder“ bekannt

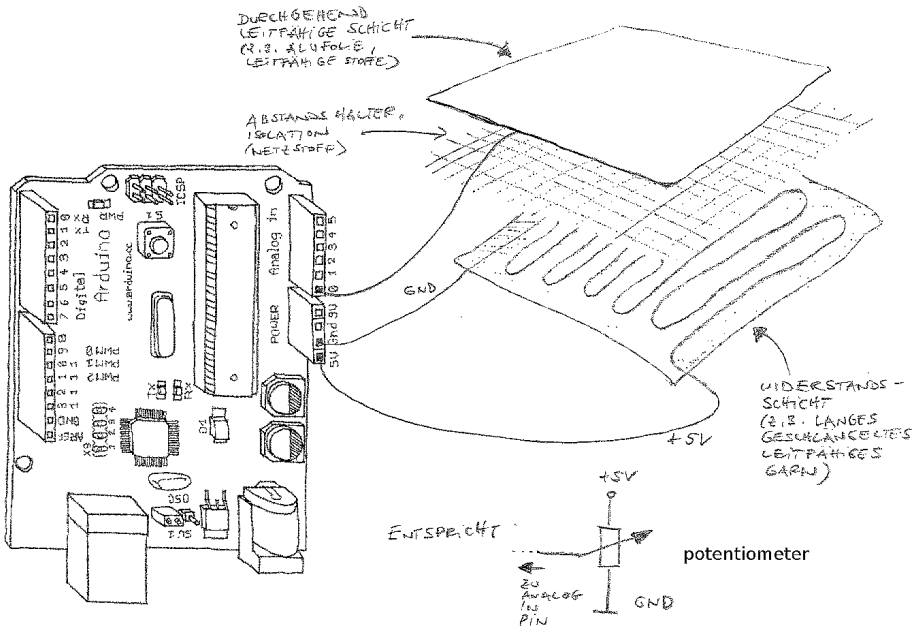

Abbildung 53: Sensor-Mote Beispiel aus einem Workshop an der HAW-Hamburg, Textiles Poti, „Slider“ mit Anleitung zum Anschluss an ein Arduino Experimentier-Board

7.3.2 Umsetzung

Sensor-Motes basieren wie die Light-Motes auch auf Arduino-Hardware und werden in der hier vorgestellten Form auch in anderen Projekten (vgl. Tetzlaff (2008), Hübler und Abbett (2002)) eingesetzt. In der neuesten *Nano*-Version haben diese beispielsweise acht analoge Eingänge mit 10-Bit Wandlern und 14 mögliche digitale Eingänge. In den üblichen Beispielen aus den Arduino-Anleitungen werden Werte dieser Eingänge zumeist unspezifiziert über die serielle Schnittstelle an einen Host-Rechner weitergeleitet und dort verarbeitet. Selten wird mehr als ein Arduino verwendet. Als Bus-System wird evtl. noch *I2C* empfohlen, jedoch kein standardisiertes Protokoll zur Sensordatenerfassung und Übertragung von Werten. Interoperabilität ist so nicht möglich, stellt hier jedoch eine wichtige Anforderung an das System dar. Daher soll hier RDM (siehe 4.2.5) als standardisiertes Übertragungsprotokoll eingesetzt werden, so dass viele Sensoren an einem Bus möglich sind, deren Daten über ein definiertes Format ausgelesen werden können.

Firmware Empfang von Daten (Kommandos zur Abfrage von Sensorwerten) geschieht analog zu oben vorgestellter DMX512 Empfangsroutine.

Zusätzlich ist noch eine Interrupt-Service-Routine zum Senden von Antworten (*RDM-Responses*) notwendig. Ein (nach DIN56930) mindestens 88us langes `BREAK` leitet dabei einen neuen DMX512-Frame ein (siehe 21). Um dieses mit dem USART eines Atmega168

senden zu können, wird kurzfristig die Baudrate herunter gesetzt (da das „normale“ Break zu kurz wäre). Danach können (wieder mit 250kbps) die Nutzdaten gesendet werden. Diese Vorgehensweise entspricht einem DMX512-Sender. Zusätzlich dazu werden über ein im Header-File definiertes `struct` und entsprechende Zeigervariable RDM-Pakete erzeugt. Folgende RDM-Codes sind dabei relevant (siehe auch 4.2.5):

- `E120_[DISCOVERY_]COMMAND_RESPONSE` u.ä. Codes zum reagieren auf Device-Discovery-Nachrichten des Controllers
- `E120_SENSOR_DEFINITION 0x0200, E120_SENSOR_VALUE 0x0201` zum Auslesen von Sensortyp und Sensorwerten.
- `E120_SENS_VOLTAGE 0x01, E120_SENS_CONTACTS 0x1C` zum Senden des Sensortyps (analog, digital), evtl. noch zusätzlich die Codes für Positions-Sensoren.

Empfängt ein Sensor-Mote ein RDM-Start und RDM-Substart-Code und erkennt sich über die eindeutige ID als Adressat der Nachricht, so prüft er über einen einfachen Zustandsautomaten den Kommandotyp, dreht den Bus um und sendet ein entsprechendes `COMMAND_RESPONSE` Paket. Anschließend wird der Bus wieder frei gegeben und auf Nachrichten „gelauscht“. Die Datenrichtung wird dabei, wie im Schaltplan der Light-Motes zu sehen, über eine digitale Leitung zwischen Atmega168 und EIA-485 Treiber bestimmt. Normale DMX512 Nachrichten werden von Sensor-Motes ignoriert, so dass diese mit Light-Motes an einem Bus zusammen eingesetzt werden können.

7.3.3 OSC Namensraum

Je nach Anzahl verwendeter Sensoreingänge, stellt der Controller nach der Discovery-Phase diese im OSC-Adressraum dar. Für jeden Eingang wird ein

```
/sensor/{id}
```

angelegt. Mit folgenden Methoden:

```
TX: /sensor/{id}/state
RX: /sensor/{id}/state ,f [value]
```

Dabei wird der über RDM abgefragte Wert normalisiert auf den Bereich [0.0-1.0]. Zur Vereinfachung sollen sowohl analoge als auch digitale Sensorwerte durch float-Typen repräsentiert werden, wobei digitale Werte eben nur 0.0 oder 1.0 annehmen können.

7.4 Audio-Motes

Audio-Motes gehören nicht zum Fokus dieser Arbeit. Anwendungen zur modularen synthetischen Klangerzeugung oder einfach nur zum Abspielen von Soundsamples mit OSC-Schnittstelle gibt es bereits in vielen Varianten (u.A. *chuck, PureData, MAX/MSP,*

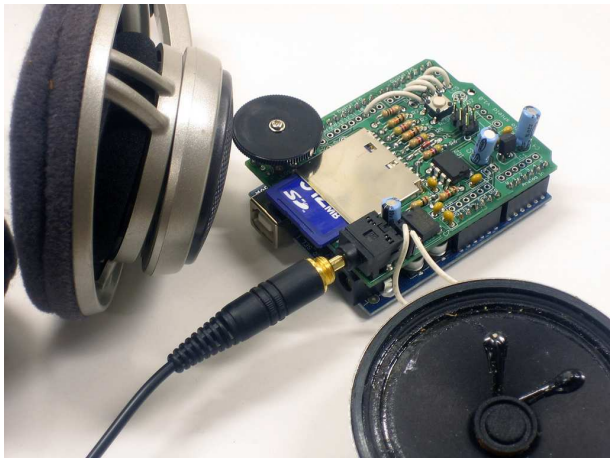

Abbildung 54: Audio-Mote auf Basis eines Arduinos und aufsteckbarem “Waves-hield,, mit DAC, SD-Kartenleser und Audio-Operationsverstärker, Quelle: Limor Fried, http://ladyada.net

SuperCollider). Darunter einige die auch auf eingebetteten Systemen lauffähig sind. Ein Audio-Mote könnte z.B. ein portabler Mediaplayer mit Ethernet oder serieller Schnittstelle und OpenSource Betriebssystem sein (vgl. Sukale (2008), *IPOD-Linux*). Mindestens müsste ein einfacher OSC-Server und Zeroconf-Daemon lauffähig sein. Es müssten dann mindestens die Methoden „play“ und „stop“ implementiert werden.

Als Versuchsaufbau verwende ich einen OpenWRT-Router mit USB-Schnittstelle und USB-Soundkarte (siehe 33). Das „usb-audio“-Format ist offen und wird unter Linux sowie unter μLinux für eingebette Geräte von den mit dem Kernel ausgelieferten *ALSA*-Treibern unterstützt. Die freie Zeroconf Implementation Avahi ist ebenfalls als OpenWRT-Paket erhältlich. Zukünftig wird man jedoch verstärkt spezielle ARM-basierte oder x86-basiertes Plattformen (z.B. *Make-Controller-Kit*) mit integrierten Audio-Ausgängen oder „on-board,, Audio-Codecs verwenden. Es gibt bereits einige quelloffene Designs, die online vertrieben werden und ebenfalls immer preiswerter werden. Arduinos selber können nur gut mit sog. Piezo-Lautsprechern (“Beepern,,) umgehen, mit denen man über Digital-Ausgang mit wenig Aufwand Rechteck-Signale ausgeben kann. Einfache monophone Melodien sind damit möglich, komplexere Klänge nur mit Zusatzhardware (siehe 54) oder hohem Programmieraufwand (z.B. *Chiptunes*: Wellenformgeneration über Lookup-Tabelle).

8 Fazit

Es wurde eine Vorgehensweise gezeigt und erprobt, wie und aus welchen Komponenten in zukünftigen Projekten interaktive multimediale Installationen grundsätzlich aufgebaut werden sollten. Speziell der Einsatz von *OSC* und *RDM* als Kommunikationsprotokolle ermöglicht zukünftig das effiziente Einbinden von Aktoren und Sensoren in verteilten Netzwerken mit hoher Granularität.

Als Teilergebnis dieser Arbeit wurde die derzeit aktuelle Entwicklungsplattform *Arduino* um die hardware- und softwareseitigen Fähigkeiten zum Senden und Empfangen entsprechender Nachrichten erfolgreich erweitert. Beispielhaft wurden *Light-Motes* realisiert, die nun in konkreten Projekten zur Anwendungen kommen und im Anschluß an diese Arbeit in Kleinserie produziert werden. Im Zuge dessen wird die entwickelte Software der OpenSource-Gemeinde zur Verfügung gestellt werden und ermöglicht so auch in anderen Zusammenhängen (*Ambient Intelligence*, *Bodymonitoring*, *Sensornetzwerke*) den einfachen und robusten Aufbau von spontanen Arduino-Netzwerken. Um diese Netzwerke bedienbar zu machen, wurden Gateway-Lösungen zu Rich-Client-Frameworks geschaffen und so die Trennung von Model, View und Control im Gegensatz zu bestehenden Softwarelösungen weiter vorangetrieben. Flexiblere, wiederverwendbare User-Interfaces können auf den dargestellten Prinzipien entworfen werden. Es wurde besonderer Wert auf *Plug-And-Play*-Funktionalität gelegt und diese unter Verwendung von Software nach aktuellem Stand der Technik umgesetzt.

Dennoch handelt es sich bei dem vorgestellten System nur um *eine* mögliche Lösungsvariante. Im großen Kontext der Forschung in den Bereichen interaktiver Kunst, verteiltes Musizieren, virtuelle Welten, Veranstaltungstechnik etc. betrachtet, stellen meine Empfehlungen nur kleine Schritte dar. Sie beweisen die Durchführbarkeit derartiger Projekte unter der doch recht strengen Vorraussetzung der *hundertprozentigen Quelloffenheit* der verwendeten Technologien. Erfolgreiche Etablierung und damit einhergehende projektübegreifende Interoperabilität der dargestellten Verfahren wäre jedoch nur möglich, durch gezielte, institutionalisierte Vermarktung.

9 Ausblick

Die Vision vom *Smart-Pixel*, der tatsächlich ein unabhängiges „Leben“ führt und sich in Austellungen zu einem ästhetischen Ganzen selbstorganisiert sollte zumindest technisch in absehbarer Zeit realisierbar sein.

Ich stelle mir einen Leuchtpunkt vor, der sanft dimmbar viele Farben aussenden kann. Der unabhängig von einer Stromversorgung und kabellos ansprechbar ist. Der sich im Netzwerk selbstorganisiert, adressiert und seinen Dienst zur Verfügung stellt oder ein ein-

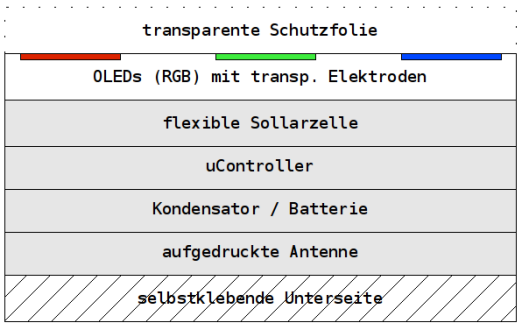

Abbildung 55: Vision eines Smart Pixel Etiketts

programmiertes eiegenständiges (Leucht-)Verhalten aufweist (Stanislaw Lem spricht hier von *künstlichem Instinkt*).

Interessant wäre ein Produkt, welches günstig in Großserie z.B. als Aufkleber hergestellt werden könnte.

Prinzipiell wäre ein Aufbau ähnlich zu einem RFID-Etikett denkbar: Integrierte aufgedruckte Antenne und Microcontroller, flexbile Solarzelle, organische aufgedruckte LEDs in rot, grün und blau. Kondensatorschicht mit ausreichender Kapazität um tagsüber Energie zu speichern und nachts wieder an die Elektronik und die LEDs abzugeben. Derartige Produkte könnten wie beschrieben mit spezieller Software zur sanften und leistungsschonenden Regelung der Lichtintensität ausgestattet werden. Organische leitfähige transparente Materialen für die Herstellung von Solarzellen und organischen LEDs sind auf dem Markt erhältlich und werden mit zunehmender Verbreitung immer günstiger werden. Antennen, Leiterbahnen und vollflächige Elektroden-Schichten könnten im *Rapid-Prototyping*-Verfahren z.B. einfach ausgedruckt werden. Ebenso könnten lichtemittierende Bereiche gedruckt werden. Als Träger kann wie bei den RFID-Etiketten ein Aufklebermaterial oder eine Folie verwendet werden.

Ein Schritt in diese Richtung, wäre die Entwicklung eines mit RGB-SMD-LEDs bestückten Arduinoboards auf Folienplatinen unter der Verwendung der hier vorgestellten Schaltpläne und Firmware. Ein anderer Meilenstein wäre die Entwicklung von Arduinoboards auf Basis von SoCs mit integriertem Funkmodul (z.B. aktuell Texas Instruments CC2431, Stichwort: Aktive RFIDs), unter Beibehaltung der erfolgreichen und einfach bedienbaren Arduino-Entwicklungsplattform.

A Quellcodes

Komplette Quellcodes zu dieser Arbeit können unter http://sukale.com/src heruntergeladen werden.

FTDI set break from userspace

```
/**************************************************************/
/* SET FTDI TO BREAK                                          */
/**************************************************************/
int ftdi_break(struct ftdi_context *ftdi, int break_state) {
    unsigned short value = 8;  /* 8 bit data */
    value |= (0x00 << 8); /* no parity */
    value |= (0x02 << 11); /* 2 stop bits */

    if (break_state) value |= (0x01 << 14); /* break on */
    if (usb_control_msg(ftdi->usb_dev,
                        0x40, 0x04,
                        value,
                        ftdi->index,
                        NULL, 0,
     ftdi->usb_write_timeout) != 0) {
        fprintf(stderr, "Setting ftdi break failed!");
        return -1;
    }
    return 0;
}
```

Abbildungsverzeichnis

Literatur

[Puckette1994] Is there life after MIDI?

[Schmeder2004 2004] : *A Query System for Open Sound Control*. Berkeley, CA : CNMAT, 30/07/2004 2004. – URL `http://cnmat.berkeley.edu/publications/query_system_open_sound_control`

[BEA 2006] BEA: 155 BONUS - Strike and destroy up to 35 km. 2006. – Forschungsbericht

[Behringer] BEHRINGER: *BFC2000 Technical Specifications*. – URL `http://www.behringerdownload.de/BCF2000/BCF2000_ENG_Rev_A_Specs.pdf`

[Carfagno 2007] CARFAGNO, Virginio: *Evaluation des Google Web Toolkits durch Entwicklung einer ajaxbasierten Mind-Mapping-Anwendung*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2007

[Ehrentraud 2007] EHRENTRAUD, Fabian: OSC 'SYN' namespace. 2007. – Forschungsbericht

[ESTA 2006] ESTA: American National Standard, E1.20 - 2006, Entertainment Technology, RDM Remote Device Management Over DMX512 Networks. 2006. – Forschungsbericht

[Fischer 2005] FISCHER, Christian: *Entwicklung von ZigBee-Modulen für spontane Funknetzwerke*, Hochschule für Angewandte Wissenschaften Hamburg, Bachelorarbeit, 2005

[Fraietta 2008] FRAIETTA, Angelo: Open Sound Control: Constraints and Limitations / Smart Controller Pty Ltd. 2008. – Forschungsbericht

[Gross 2006] GROSS, Christian: *Ajax, Design Patterns and Best Practices*. Apress L.P., Berkeley, USA, 2006

[Howell 2002] HOWELL: An overview of the electronic drive techniques for intensity control and colour mixing of low voltage light sources such as LEDs and LEPs. 2002. – Forschungsbericht

[Howell 2007] HOWELL, Wayne: Artistic Licence Application Note 13 What is Art-Net? (2007)

[Hölscher 2008] HÖLSCHER, Hendrik: AN017: Remote Device Management. (2008)

[Hübler und Abbett 2002] HÜBLER ; ABBETT: designing desire, wearables / Interaction Design Institute Ivrea, Italien. 2002. – Forschungsbericht

[Jakubowsky 2005] JAKUBOWSKY: *MOTEs - Evaluation und exemplarische Anwendung einer neuen Technologie*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2005

[Jensenius 2006] JENSENIUS: Towards a Gesture Description Interchange Format / Musical Gestures Group, Department of Musicology, University of Oslo. 2006. – Forschungsbericht

[Lehmann 2004] LEHMANN, E.: *Verfahren zur Steuerung der Energiezufuhr von einer Stromquelle an einen Stromverbraucher*. 2004. – Deutsche Patentschrift Aktenzeichen-DE: 10 2004 044 001.8

[Lem 1983] LEM, Stanislaw: *Waffensysteme des 21. Jahrhunderts*. Suhrkamp Taschenbuch, Frankfurt, 1983

[Malloch 2008] MALLOCH, Wanderley: From Controller to Sound: Tools for Collaborative Development of Digital Musical Instruments. 2008. – Forschungsbericht

[Mills 1992] MILLS: *Network Time Protocol (Version 3) Specification, Implementation and Analysis*. 1992. – URL `http://tools.ietf.org/html/rfc1305`

[Ness 2006] NESS, Tomek: Arduino becomes a OpenDmx USB interface. 2006. – Forschungsbericht

[Place 2008] PLACE, Jensenius Peters B.: Adressing Classes by Differentiating Values and Properties in OSC. 2008. – Forschungsbericht

[Place 2006] PLACE, Lossius: Jamoma: A modular standard for structuring patches in Max. 2006. – Forschungsbericht

[Ramakrishnan 2004] RAMAKRISHNAN, Chandrasekar: Discovering OSC services with ZeroConf. In: *OSC Conference 2004*, 30/07/2004 2004

[Reas und Fry 2007] REAS, Casey ; FRY, Ben: *Processing: a programming handbook for visual designers and artists*. The MIT Press, Cambridge, Massachusetts, 2007

[Schmeder 2008a] SCHMEDER, Andy: Everything you ever wanted to know about Open Sound Control. 03/2008 2008. – Forschungsbericht

[Schmeder 2008b] SCHMEDER, Freed: uOSC: The Open Sound Control Reference Platform for Embedded Devices. 2008. – Forschungsbericht

[Stickel 2004] STICKEL: *Konstruktion eines mobilen Systems zur Steuerung von Bühnenbeleuchtungsanlagen*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2004

[Sukale 2004] SUKALE, Martin: Projekt Musikproduktion mit GNU/Linux. 2004. – Forschungsbericht

[Sukale 2008] SUKALE, Martin: *Computergestützte Kunstprojekte - Neuere technologische Entwicklungen*, Hochschule für Angewandte Wissenschaften Hamburg, Studienarbeit, 2008

[Tanenbaum und Steen 2003] TANENBAUM ; STEEN: *Verteilte Systeme - Grundlagen und Paradigmen*. Pearson Studium, 2003

[Tetzlaff 2008] TETZLAFF: *Bodymonitoring: Entwicklung eines Prototypen für intelligente Kleidung*, Hochschule für Angewandte Wissenschaften Hamburg, Masterthesis, work in progress, 2008

[Wright 2002] WRIGHT, Matt: *OpenSound Control Specification Version 1.0*. 2002. – URL `http://www.opensoundcontrol.org`

[Wright 2003] WRIGHT, Matthew: OpenSound Control: State of the Art 2003. 2003. – Forschungsbericht

Eingesetzte Werkzeuge, Bibliotheken

Die hier abgebildeten Diagramme sind mit OpenOffice erstellt worden. Symbole sind der Open Source Icon Bibliothek ”Tango” entnommen. Die Arbeit wurde mit ”kile” und LaTeX auf ”Ubuntu”-Linux geschrieben und gesetzt. Bilder wurden mit ”gimp” bearbeitet. Schaltpläne und Platinenlayouts mit Eagle-CAD (Freeware-Version). Programme wurden mit gcc bzw. avr-gcc compiliert.

Danke

Daniel Hausig für die langjährige Unterstützung.