Generalized Interfaces for Interacting with Media Façades

Master's Thesis

zur Erlangung des akademischen Grades
Master of Science (M. Sc.)

an der
Hochschule für Technik und Wirtschaft Berlin
HTW Berlin

Fachbereich: Wirtschaftswissenschaften II
Studienfach: Angewandte Informatik

Erstbetreuer: Prof. Dr. Jürgen Sieck
Zweitbetreuer: Prof. Dr. Volodymir Brovkov

Autor: Stephan Bergemann, B. Sc.
Matrikelnummer: 521034

Datum: 5. September 2012

Inhaltsverzeichnis

Danksagung

Diese Masterarbeit wäre ohne einige Menschen in meinem näheren und weiteren Umfeld nicht zu dem geworden, was sie ist.

Daher möchte ich meinen Dank zunächst an Herrn Prof. Dr. Jürgen Sieck, sowie Herrn Prof. Dr. Volodymir Brovkov richten, die mir mit wertvollen Ratschlägen über die gesamte Zeit hilfreich zur Seite standen.

Des weiteren bin ich Andreas für seine tatkräftige persönliche Unterstützung, stets konstruktive Kritik und Aufmunterung sehr herzlich danken.

Ganz besonderen Dank möchte ich gegenüber meiner Freundin Franziska zum Ausdruck bringen, die mir während der gesamten Bearbeitungszeit stets den Rücken frei hielt und mich intensiv unterstützte.

Auch meinen Eltern, Großeltern und meiner Schwester möchte ich für die Unterstützung danken und dafür, dass sie während der Anfertigung dieser Arbeit Verständnis für meine „Abwesenheit“ hatten.

1. Einleitung

Die Silhouette einer Stadt ist vor allem von den Formen und Farben der Gebäude abhängig, die die Stadt prägen. Die Fassaden der Häuser, sowie die Konturen von Denkmälern und Ahnlichem tragen wesentlich zum Eindruck bei, den Menschen von Städten in Erinnerung behalten.

Über die Zeit verändert sich ein solches Bild von einer Stadt: Häuser verschwinden, neue entstehen. In den letzten Jahren hat sich dabei ein neues Stilmittel für Architekten und Städteplaner herausgebildet, welches mehr und mehr Einsatz findet: wandelbare Fassaden – sogenannte Medienfassaden – halten Einzug in die Architektur. Um uns herum entstehen immer mehr Gebäude, die nicht mehr nur ein festes Aussehen haben, sondern teils drastisch in ihrer Erscheinung beeinflusst bzw. verändert werden können. Durch neue Technologien ist es möglich, die Farbe und Form einer Fassade sich kontinuierlich ändern zu lassen. Dabei ist es nicht nur möglich, feste Abläufe zu bestimmen und damit Kunst, Information und multimedialen Inhalt zu vermitteln. Vielmehr bietet sich durch die Verwendung verschiedenster Eingabegeräte inzwischen auch die Möglichkeit, interaktive Installationen zu realisieren, die das architektonische Werk zu einem besonderen Unikat machen können.

Der Oberbegriff dieser neuen Bewegung in der Architektur ist *Media Architecture*. In den vergangen Jahren gab es bereits zahlreiche Konferenzen und Veranstaltungen dazu und international erfreut sich dieser Trend hin zu bespielbaren Elementen in der Architektur wachsender Beliebtheit. *Media Architecture* ist dabei sehr breit gefächert. Der Fokus dieser Arbeit liegt auf Medienfassaden.

Der Begriff Medienfassade beschreibt dabei Fassaden, die auf verschiedene Arten verändert werden können – sei es über computergesteuerte Lichteffekte oder auch beispielsweise über computergesteuerte bewegte Elemente der Fassade selbst.

1.1. Hintergrund und Motivation

Für das Forschungs- und Weiterbildungszentrum für Kultur und Informatik (FKI) an der HTW Berlin, welches sich momentan im Bau befindet und voraussichtlich im Dezember 2012 fertig gestellt werden soll, ist die Installation einer Medienfassade geplant, welche neben einer Photovoltaikanlage integraler Bestandteil des Gebäudes sein soll.

Die Medienfassade ist zum Zeitpunkt des Schreibens dieser Arbeit noch in der Detailplanung und nicht fertig gestellt. Daher sind einige Schlüsselfragen noch offen und auch im Rahmen dieser Arbeit zu diskutieren. Die Fassade als solche soll nicht nur mit multimedialen Inhalten bespielt werden soll, sondern auch Möglichkeiten zur interaktiven Steuerung von Inhalten bieten. Es stellen sich dabei beispielsweise die Fragen, was auf ihr dargestellt werden soll, wer mit der Fassade in welcher Form interagieren wird und wie das gesamte System generell anzusprechen ist.

Vor allem die letzte Frage ist insofern interessant, als dass die geplante Fassade nicht wie sonst sehr oft nur eine einzige Technologie zur Visualisierung nutzt, die über die gesamte Fassade angewandt werden soll, sondern durch geschickte Kombination verschiedener Technologien ein flexibel bespielbares Gesamtwerk entstehen soll. Dabei soll schließlich die gesamte Nordfassade des Gebäudes illuminieren werden. Durch diese Kombination von Technologien mit unterschiedlichen Eigenschaften haben einerseits Künstler und Studenten die Möglichkeit diese jeweils auszuprobieren und auch die Option, ihre Fertigkeiten in der Bespielung und Programmierung der Kombination der verschiedenen Systeme unter Beweis zu stellen.

1.2. Zielstellung

Neben dieser Besonderheit, verschiedene Technologien auf einer Fassade zu vereinen, ergeben sich aus dem Wunsch eine interaktiv nutzbare Fassade zu schaffen weitere Fragestellungen, die in dieser Arbeit näher beleuchtet werden sollen. Interkative Systeme werden immer populärer und sind inzwischen fast schon allgegenwärtig: mit Web 2.0, interaktiven Filmen, immersive *Virtual Reality* und einigem mehr ist man ständig umgeben von interaktiv bedienbaren Systemen. Daher ist es nicht verwunderlich, dass auch im Bereich der *Media Architecture* Interaktivität schnell eine größer werdende Rolle

spielt. Neue Eingabegeräte, wie die Wii™[1], die Kinect™[2] und Ähnliche zeigen auf, dass auch immer größere Potentiale in diesem Gebiet stecken. Dennoch ist es auf Gebieten, in denen schnelle Entwicklung stattfindet meist auch so, dass es an Standards mangelt und letztlich jeder seine eigene Form interaktiver Steuerung in seiner eigenen Software implementiert. Interessant ist es daher Unterschiede und Gemeinsamkeiten solcher Eingabegeräte zu erarbeiten und anschließend eine generalisierte Möglichkeit zu schaffen, interaktive Anwendungen für Medienfassaden zu implementieren.

Konkretes Ziel dieser Arbeit ist es also, eine Infrastruktur und ein Rahmenwerk für Anwendungsentwickler zu konzipieren. Mit diesen Komponenten einher geht die Definition einer abstrahierten Schnittstelle für verschiedene Eingabedaten, die von allen Anwendungen genutzt werden kann. Dadurch können Eingabegeräte getauscht werden, ohne dass Anwendungen immer wieder neu implementiert werden müssen.

1.3. Struktur der Arbeit

Nach der Einleitung werden in Kapitel 2 die technischen Grundlagen und Begrifflichkeiten eingeführt, die für das weitere Verständnis der Ausführungen essenziell sind. Es wird auf bereits existierende Lösungen sowohl aus dem Bereich der Schnittstellen, als auch dem Bereich der Medienfassaden und insbesondere interaktiver Anwendungen auf Medienfassaden eingegangen. Daraus ergibt sich eine Analyse von Anforderungen, die an eine solche Schnittstelle gestellt werden können. Diese Anforderungen werden unter Erklärung verschiedener Nutzungszenarien ausgebreitet. In Kapitel 4 wird der Entwurf eines Systems beschrieben, welches eben jene Anforderungen erfüllt und anhand von Beispielanwendungen soll gezeigt werden können, wie flexibel es einsetzbar ist.

Die Anforderungen aus Kapitel 3 und dem Entwurf aus Kapitel 4 folgend wird in Kapitel 5 die technische Realisierung des Systems und seiner einzelnen Komponenten beschrieben. Dabei wird anhand von Quelltexten auf die Umsetzung des Systems eingegangen.

In Kapitel 6 werden die einzelnen entwickelten Komponenten und insbesondere die geschaffene generalisierte Schnittstelle evaluiert und anhand wichtiger Kriterien getestet.

[1] `http://www.nintendo.de/NOE/de_DE/wii_54.html`(besucht am 31. August 2012)
[2] `http://www.xbox.com/de-DE/Kinect`(besucht am 31. August 2012)

Es werden exemplarisch Bilder der entwickelten Komponenten aufgeführt, die das System im Einsatz zeigen.

Mit Kapitel 7 wird noch einmal rekapituliert, was im Rahmen dieser Arbeit geschaffen wurde und welche einzelnen Komponenten das System ausmachen. Es wird noch einmal herausgehoben, was das Entworfene System besonders macht und welche Tests welche Ergebnisse hatten.

Abschließend wird in Kapitel 8 darauf eingegangen, was mit diesem System an Anwendungen umsetzbar ist und auch abseits von Medienfassaden angeregt in welche Richtung weitere Enwticklungen gehen könnten, die die Vorteile des Systems weiter zur Geltung bringen können.

2. Grundlagen

In diesem Kapitel werden Typen von Medienfassaden anhand genutzter Technologien beschrieben. Neben einer Einführung in die verschiedenen Visualisierungstechniken wird ebenso ein Überblick über die dabei verwendeten Schnittstellen gegeben. Nach einem Vergleich dieser Schnittstellen wird auf Schnittstellen zur Interaktion im Allgemeinen eingegangen, wobei zunächst der Begriff der Interaktion für diese Arbeit definiert wird.

2.1. Medienfassaden – Typen und Technologien

Nach Häusler [Häu09] können Medienfassaden in verschiedene Kategorien eingeteilt werden: zunächst können sie in mechanische und elektronische Fassaden unterteilt werden. Anschließend kann nach der Art und Weise, wie Inhalt dargestellt werden kann unterschieden werden. So unterscheidet Häusler weiter in mechanische, Projektions-, Illuminations- und Displayfassaden. Die letzten drei dieser Unterteilungen können weiter anhand der verwandten Technologie unterteilt werden. In Bergemann und Schlegel [BS12] werden die wesentlichen heute verwendeten Technologien sowie dabei verwendeten Visualisierungsschnittstellen wie DMX und DVI näher beschrieben. In der vorliegenden Arbeit geht es allerdings weniger um die Techniken zur Visualisierung, sondern vielmehr um die Techniken zu Interaktion, die verwedeten Schnittstellen auf Hard- und Softwareebene, welche Ansteuerungsarten sinnvoll sind und welche nicht.

2.2. Interaktion mit Medienfassaden

Interaktion beschreibt eine Wechselwirkung zwischen Akteuren und Systemen. Diese Wechselwirkung wird oft in Form von Kommunikation beschrieben - ein Akteur kommuniziert mit einem System und umgekehrt. In Bergemann und Schlegel [BS12] werden

wichtige grundlegende Interaktionsformen mit Medienfassaden vorgestellt. Es wird dabei herausgearbeitet, dass verschiedene Formen von Eingabegeräten zur Interaktion genutzt werden können:

Geräte der Fassade gehören dediziert zur Medienfassade und werden entsprechend nur dort zur Verfügung gestellt (z.B. Eingabeterminals oder extra Controller).

Eigene Eingabegeräte werden vom Nutzer der Medienfassade selbst mitgeführt (z.B. Do it yourself (DIY) Hardware oder auch Smartphones.

Nicht haptische Eingabegeräte sind im Wesentlichen all solche Geräte, die zwar an der Fassade verbaut sind, aber weitestgehend berührungslos bedient werden (z.B. Kameras oder Mikrofone).

2.2.1. Sensoren

Sämtliche Eingabegeräte, wie sie in Abschnitt 2.2 aufgeführt werden beschreiben letztlich Sensoren. Sensoren gibt es für nahezu jede physikalisch messbare Eigenschaft bzw. Größe. Sie liefern für diese Größe diskretisierte Werte. „Werte liefern“ bedeutet, dass durch physikalische oder chemische Effekte bzw. Reaktionen physikalische Größen gemessen und in meist elektrische Signale umgewandelt werden. Generell gibt es aktive und passive Sensoren. Aktive Sensoren erzeugen selbst Strom, während passive Sensoren den Stromfluss einer Leitung beeinflussen. Unterschieden wird weiterhin in binäre, digitale und analoge Sensoren. Binär, digital und analog bezieht sich dabei allerdings nur auf die Ausgangssignale der Sensoren. Binäre Sensoren liefern genau entweder eine bestimmte Spannung, oder eine andere (bei Schaltern: Strom fließt oder Strom fließt nicht). Digitale Sensoren liefern Werte in schon kodierter Form über eine entsprechende Schnittstelle - bilden also genau genommen eher eine Zwischenform zwischen analogen oder binären Sensoren und einem Mikrocontroller, da sie einen gemessenen Wert bereits kodieren. Analoge Sensoren liefern eine dem gemessenen Wert entsprechende Spannung üblicherweise zwischen einer Referenzspannung und der Erdung. Mittels eines Analog-Digital-Umsetzers können daraus am Mikrocontroller diskrete Werte erzeugt werden.

Es gibt eine Vielzahl von Sensoren auch für die Messung ähnlicher und gleicher physikalischer Größen. Prinzipiell sollte unterschieden werden in Sensoren, die ständig Daten liefern und solche, die nur gelegentlich eine Wertänderung registrieren. Zu Ersteren

gehören viele analogen Sensoren, wie solche beispielsweise für Licht (Photowiderstände und Kameras), Beschleunigung (Lage und Geschwindigkeit) und Druck (Mikrofone, Luftdrucksensoren). Von Menschenhand bedienbare Sensoren wie Taster, Drehregler, Schieberegler und Ähnliche werden meist an Stellen genutzt, an denen Wertänderungen nur gelegentlich vorkommen.

2.3. Soft- und Hardwareschnittstellen

Die Daten von Sensoren müssen über definierte Schnittstellen an eine verarbeitende Infrastruktur geleitet werden, um beispielsweise für Software nutzbar zu werden. Da unzählige Schnittstellen bereits existieren, werden in diesem Abschnitt einige von ihnen exemplarisch und anhand ihrer kennzeichnenden Eigenschaften vorgestellt. Anschließend wird darauf aufbauend eine Abwägung getroffen, welche Schnittstelle für die Interaktion mit Medienfassaden sinnvoll nutzbar ist, welche Anpassungen vorgenommen werden müssen oder was gegebenenfalls von Grund auf neu implementiert werden muss.

2.3.1. RS-232 und RS-485

Eine der ältesten Schnittstellen für Computer überhaupt ist die serielle Schnittstelle nach EIA-RS-232. Sie wurde bereits in den 1960er Jaren eingeführt. Während zunächst 25-polige Stecker vorgesehen waren, wurden vor allem beim PC zunehmend 9-polige Verbindungselemente verbaut. Die Übertragung von Informationen erfolgt in Wörtern. Dabei ist ein Wort zwischen 5 und 9 Bit lang und von Start-Bit, optionalem Paritätsbit und einem Stopp-Bit umschlossen. Die Synchronisation des Taktes erfolgt über die Zeitdifferenz zwischen Start- und Stop-Bit und einer bei jedem Teilnehmer festgelegten Baudrate, die nur um wenige Prozent von der tatsächlichen Baudrate abweichen sollte. Um dem Empfänger Zeit zur Synchronisation einzuräumen kann das Stop-Bit $1,5$ bis 2 mal so lang sein, wie ein normales Bit. Dies muss allerdings bei der Initialisierung der seriellen Schnittstelle mit angegeben werden.

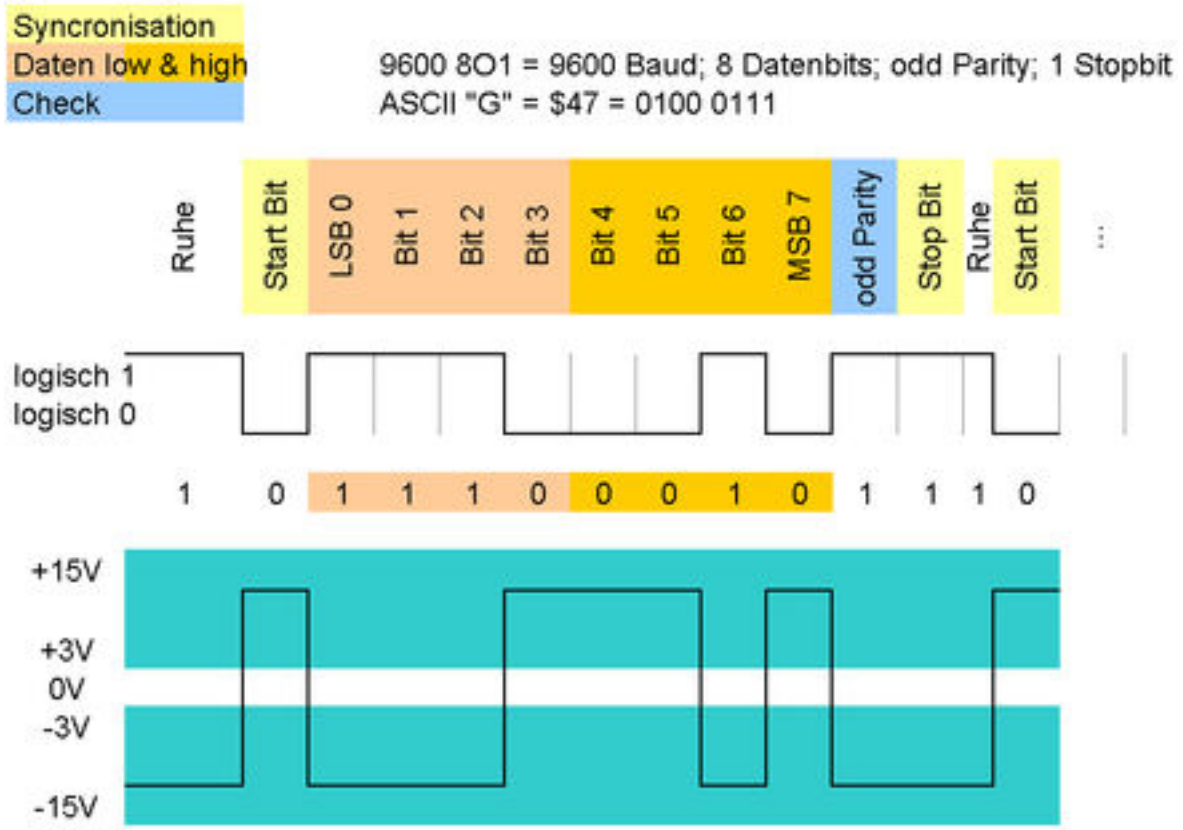


Abbildung 2.1.: Das Schema der Datenübertragung nach EIA-232 (RS-232) [wik]

Wie in Abbildung 2.1 dargestellt erfolgt die Datenübertragung in RS-232 mit Pegelwechseln zwischen $-15\,\mathrm{V}$ und $15\,\mathrm{V}$, wobei $-3\,\mathrm{V}$ bis $3\,\mathrm{V}$ nie erreicht werden sollten. Jede Spannung ab $-3\,\mathrm{V}$ bis $-15\,\mathrm{V}$ wird als 1 interpretiert und jede Spannung zwischen $3\,\mathrm{V}$ und $15\,\mathrm{V}$ als 0. Auch ersichtlich wird, dass RS-232 Least Siginificant Bit (LSB) als Bitorder vorgibt.

Sämtliche weitere betrachteten Schnittstellen sind ebenfalls serielle Schnittstellen und für das anvisierte Anwendungsgebiet wohl die vielversprechendsten bzw. gebräuchlichsten. Eine Erweiterung der EIA-232 Schnittstelle ist die EIA-485 bzw. RS-485 Schnittstelle, welche im Gegensatz zur RS-232 Schnittstelle auf differenzieller Datenübertragung basiert. Es wird immer ein invertierter und ein nicht invertierter Pegel übertragen. Da beide Pegel den selben Störungen ausgesetzt werden, kann der Empfänger nun immernoch aus der Differenz der beiden Signale das Ausgangssignal detektieren. Daher sind mittels RS-485 wesentlich höhere Distanzen überbrückbar. Weiterhin ist zu bemerken, dass Verbindungen nach RS-485 multipoint-fähig sind (mehrere Teilnehmer können an

einem Bus und Daten senden und empfangen), wohingegen RS-232 für Punkt-zu-Punkt-Verbindungen ausgelegt ist.

2.3.2. MIDI

Musical Instrument Digital Interface (MIDI) steht für einen Standard, dessen Version 1.0 1982 eingeführt wurde. Dieser Standard beschreibt die Beschaffenheit von Hardware und das zur Datenübertragung genutzte Protokoll für den Austausch von Steuerungsinformationen vor allem im Bereich elektronischer Musikinstrumente. Für den Bereich Musik konzipiert und dort auch heute noch de facto Standard hat es auch in anderen kreativen Bereichen Einzug gehalten. Aufgrund seiner Flexibilität und einfacher Beschaffenheit, welche im Folgenden erläutert werden soll, ist MIDI in vielfältigen Anwendungsgebieten einsetzbar.

Hardwareschnittstelle

MIDI wurde als unidirektionale serielle Schnittstelle mit einer Geschwindigkeit von 31 250 Baud entworfen, auf der Daten in Form von Bytes übertragen werden.

Über MIDI werden Daten über gedrückte und losgelassene Tasten und die Intensität des Drucks auf Tasten eines Keyboards übertragen. Synthesizer (elektronische Klangerzeuger) können anschließend die entsprechende Note in der entsprechenden Intensität (Lautstärke) spielen. Neben diesen Informationen können weitere Datenpakete zur allgemeinen Steuerung von Synthesizern übertragen werden (beispielsweise Werte bzw. Wertänderungen von Parametern).

Als Grundlage für die MIDI Hardware dienen DIN 5/180° Buchsen und Stecker. Die Pins 4 und 5 werden zur Datenübertragung genutzt, wobei Pin 4 an 5 V und Pin 5 an 0 V jeweils über einen 220 Ω Widerstand liegen. Die Hardware funktioniert nach dem Schema eines Busses, auf dem mehrere Geräte unabhängig schreiben und lesen können. MIDI-Geräte verfügen normalerweise über drei DIN 5/180° Buchsen: IN, OUT und THRU bezeichnet. An IN werden die Nachrichten aus dem MIDI-System empfangen, an OUT die Nachrichten dieses Controllers ausgegeben und über THRU die Nachrichten wie sie an IN ankommen weitergegeben. Geräte werden entsprechend immer durch Kabel zwischen OUT oder THRU nach IN verbunden.

Datenformat

MIDI-Steuernachrichten werden an Kanäle verschickt und jedes zu steuernde Gerät lauscht auf mindestens einem bestimmten Kanal (schickt aber alle Nachrichten nichtsdestotrotz weiter über THRU). Vom zuerst gesandten Statusbyte sind für den Kanal ein Nibble (4 Bit) reserviert und für die Ankündigung des Nachrichtentyps wiederum ein Nibble. Aus den 4 Bit für den Kanal ergibt sich auch die Beschränkung von 16 Kanälen. Ist das Most Siginificant Bit (MSB) in einem übertragenen Byte gesetzt, ist das entsprechende Byte das Erste einer Nachricht, ansonsten ein zweites oder drittes je nach Nachrichtentyp.

Es gibt im Wesentlichen drei Klassen von MIDI Nachrichten, welche in Britz und Löhle [BL] beschrieben werden:

Channel Voice Messages betreffen immer einen bestimmten MIDI Kanal und kodieren Informationen über gedrückte Tasten, Anschlagdynamik und Ähnliches.

System Common Messages betreffen das gesamte System und dienen beispielsweise dazu, einen Song auszuwählen, den Songpointer an eine bestimmte Position zu setzen, das Zeitformat festzulegen und weiteres.

System Realtime Messages dienen der Steuerung des MIDI Systems. Darunter zählen Signale wie Start und Stop, wie auch die MIDI Clock (ein Signal, dass zur Synchronisierung mehrer MIDI-Geräte genutzt werden kann).

Zu den System Common Messages gehören auch die sogenannten System Exclusive Messages. Dies sind vom Hersteller festgelegte individuelle Nachrichten, welche meist keinen weiteren Standards folgen und daher einerseits die generischsten MIDI Nachrichten darstellen, andererseits aber durch ihre fehlende Standardisierung meist nur von ganz bestimmten Geräten oder Softwareprodukten verstanden werden.

MIDI über USB

Serielle Schnittstellen in ihrer ursprünglichen Form sind heute an Rechnern kaum noch zu finden. Dafür hielt der Universal Serial Bus (USB) Einzug und mit ihm das Konzept von Device-Deskriptoren. Diese Deskriptoren geben an, was für ein Gerät am Bus hängt, damit der Host die empfangenen Daten entsprechend verarbeiten und das Gerät entsprechend ansprechen kann. Seit 1999 gibt es auch einen MIDI USB-Deskriptor

Abbildung 2.2.: Aufbau eines MIDI-Pakets über USB

und entsprechende Definitionen für MIDI-Nachrichten über USB, welcher in Ashour u. a. [Ash+99] beschrieben wird. Dabei sind einige Merkmale besonders hervorzuheben:

Höhere Geschwindigkeit ermöglicht das gleichzeitige Übertragen von wesentlich mehr MIDI-Nachrichten.

Virtuelle Kabel ermöglichen es, mehr als 16 Geräte zu adressieren.

Feste Breite von Nachrichten (4 Bytes) wird definiert, um aufwändiges Parsen der Nachrichten zu verhindern.

Durch die wesentliche höhere Geschwindigkeit wird es möglich, mehr Daten zu übertragen und sinnvoll mehr als 16 Geräte gleichzeitig ansprechen zu können, da selbst 16 Geräte diese Menge an MIDI-Daten gar nicht erzeugen können. Um allerdings MIDI nicht allzu sehr abwandeln zu müssen, werden hier virtuelle Kabel eingeführt. Jedes Kabel für sich kann wiederum nur 16 Geräte ansprechen. Die Information über das Kabel wird hier in einem weiteren Byte vor jeder MIDI-Nachricht kodiert. Dadurch werden die längsten MIDI-Nachrichten 4 Byte lang. Dieser Maximalwert wird auch gleichzeitig als feste Größe von Nachrichten definiert. Neben der Nummer des virtuellen Kabels, welche in 4 Bit gespeichert wird, enthält das zusätzliche Headerbyte noch 4 Bit, die die nachfolgende MIDI-Nachricht beschreiben.

2.3.3. DMX

Ein etwas jüngeres Protokoll als MIDI ist das zur Steuerung von Veranstaltungstechnik entworfene DMX 512 bzw. DMX, dessen Erarbeitung 1986 begann und 1990 zur Standardisierung führte. Wie bereits in Bergemann [Ber10] beschrieben, wurde DMX als Protokoll nebst Hardwareschnittstelle zur Steuerung von beispielsweise Licht, Nebelmaschinen und ähnlichen Effektgeräten entworfen. Es basiert auf der Idee, zyklisch 512 Werte (bei DMX „Kanäle“ genannt) über eine serielle Schnittstelle basierend auf dem RS-485 Standard zu verschicken. Jeweils ein oder mehrere Geräte sind entsprechend

konfiguriert, den Werten auf bestimmten Kanälen folgend Aktionen auszuführen (beispielsweise ein Licht in einer bestimmten Farbe aufleuchten zu lassen oder Ähnliches). Es wurde in Bergemann [Ber10] ebenso herausgearbeitet, dass basierend auf den Informationen aus Ackermann [Ack06] das ursprüngliche DMX mit seiner Geschwindigkeit von 250 kbit/s eine Wiederholrate der 512 Werte von 44,1 Hz hat.

Aufgrund seiner beschränkten Beschaffenheit, was Geschwindigkeit und Anzahl der Kanäle anbelangt und andererseits seiner weiten Verbreitung wurden Systeme um DMX herum geschaffen, die auf größere Installationen anwendbar sind. Dazu werden 512 Kanäle und die darüber gesteuerten Geräte zu einem Universum zusammengeschlossen. Mehrere Universen können nun über andere Kommunikationskanäle (beispielsweise Ethernet) angesprochen und gesteuert werden – dazu wird nur eine Umsetzung von Ethernet zu den einzelnen Universen benötigt. Ab diesem Umsetzer ist das Protokoll für die angeschlossenen Geräte wieder DMX mit seinen Vor- und Nachteilen.

Zu DMX noch zu bemerken ist, dass es sowohl für Visualisierung, als auch zur Interaktion genutzt werden kann. Es bildet ein Protokoll, bei dem 512 Werte beliebig geändert werden können. Was diese Werte letztlich beschreiben richtet sich nach den Geräten, die am Bus die Werte in Aktionen umsetzen.

2.3.4. OSC

Open Sound Control (OSC) ist im herkömmlichen Sinne kein Protokoll, sondern eine Spezifikation eines Nachrichtenformats. Es gibt keine Hinweise oder Vorgaben über die Übertragungsgeschwindigkeit oder Beschaffenheit zu verwendender Übertragungshardware – es soll im Gegenteil von Anfang an kein Medium ausgeschlossen sein. Auch ist bei der Kommunikation keinerlei Reihenfolge bestimmter Nachrichten zu beachten (auch kein Sessionmanagement oder Ähnliches). Daher ist OSC am ehesten noch vergleichbar mit XML[W3C], WDDX[Sim], JSON[Cro06] oder YAML[BEN09], welche auch alle letztlich je ein Datenformat beschreiben. Auf der anderen Seite wird OSC auch im OSI-Modell auf Ebene 6 eingeordnet [SFW10]. Gedacht war OSC bei seiner Einführung als potentieller Nachfolger von MIDI und soll dessen Einschränkungen (vor allem in Sachen Geschwindigkeit und Auflösung der übertragenen Daten) überwinden. Mittels verschiedener vordefinierter Datentypen und einer Adressierung von Parametern über (menschenlesbare) Pfadstrukturen (z. B. `/pong/hintergrund/farbe`) wurde eine

generische Basis zur Entwicklung von Steuerungscontrollern unterschiedlicher Art geschaffen.

Als nachrichtenbasiertes Protokoll ist es nicht für Streaming von multimedialen Inhalten geeignet. Allerdings lassen sich mit OSC einzelne oder Kombinationen verschiedener Datentypen an entweder ein und dieselbe Adresse oder auch wildcard-basierte Adressen senden. Da OSC-Nachrichten beliebig lang werden können und verschiedene Protokolle und Schnittstellen streambasiert sind (Transmission Control Protocol (TCP) beispielsweise), gibt es zwei Möglichkeiten ein nachrichtenbasiertes Protokoll über solche Schnittstellen zu übertragen:

- Die Länge des Pakets zuerst senden
- Mit Start- und Endmarkierungen arbeiten

Der Vorteil einer zuerst gesendeten Länge ist, dass anschließend genau die passende Größe an Speicher reserviert werden kann. Der Nachteil hingegen ist, dass bei seriellem Datenstrom nicht gewährleistet ist, dass tatsächlich das erste gelesene Byte auch die Länge des Pakets angibt. Wenn beispielsweise Daten verloren gehen, oder ein Empfänger erst später zwischen geschaltet wird, ist es allein mit einer Längenangabe ohne deren eindeutige Kennzeichnung nicht möglich, sie vom Rest der übertragenen Daten zu unterscheiden.

Bei der Verwendung von Start- und Endmarkierungen verhält es sich genau umgekehrt: Es kann keine passende Größe an Speicher alloziert werden – hier wir daher meist eine maximalen Puffergröße voreingestellt. Andererseits kann bei einem seriellen Datenstrom mit Start- und Endmarkierungen genau festgestellt werden, wann ein Paket anfängt und wann es endet. Per Spezifikation von 1.1 von OSC in Freed und Schmeder [FS09] wurde mit SLIP, welches in Romkey [Rom88] beschrieben wird, auf zweitere Option zurückgegriffen. Das ist allerdings nicht in allen Implementierungen umgesetzt.

OSC wird meist per User Datagram Protocol (UDP) übertragen und ermöglicht – wie auch schon MIDI – die Arbeit simultan mit verschiedenen Controllern, die auch dieselben Parameter ansteuern können. Somit wird eine Kolaboration von Musikern ermöglicht. OSC ist dabei unicast, multicast und auch peer-to-peer-fähig.

Daten in OSC werden immer mit einer Länge eines Vielfachen von 4 Byte verschickt [Wri]. Ist die Adresse beispielsweise nicht ein Vielfaches von 4 Bytes lang, wird sie mit Null-Bytes aufgefüllt (padding).

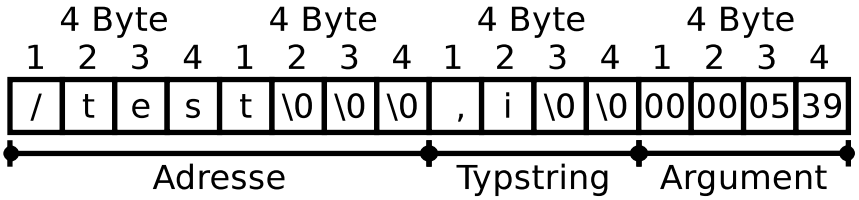


Abbildung 2.3.: Byteaufteilung eines OSC-Pakets

Die Basisstruktur der Nachrichten enthält immer eine Adresse als nullterminierte Zeichenkette, eine Zeichenkette für die folgenden Datentypen (Typetag) (wiederum nullterminert und mit „,“ beginnend) und einer durch die Kombination der in der Typetag Zeichenkette vermerkten Datentypen bestimmten Menge an Binärdaten.

Am Beispiel von Abbildung 2.3 werden je von der Adresse und von der Typtag Zeichenkette die Bereiche bis zur 4 Bytegrenze mit `\0` aufgefüllt. In diesem Beispiel ist mit `,i` festgelegt worden, dass ein 32 Bit Integer Wert folgt, welcher in den letzten 4 Byte der Nachricht (hexadezimal) kodiert ist.

Folgende Datentypen existieren laut Open Sound Control 1.0 und sind big-endian kodiert zu übertragen (die Werte in Klammern sind die Buchstaben, die im Typetag korrespondierend vermerkt werden):

int32 ('i') beschreibt eine 32 Bit vorzeichenbehaftete Ganzzahl

OSC-timetag ('t') ist ein 64 Bit langer Zeitstempel dessen Semantik zwar mit Version 1.0 genau festgelegt war, mit Version 1.1 allerdings sehr aufgeweicht wurde

float32 ('f') wird definiert als eine 32 Bit breite Fließkommazahl nach IEEE 754

OSC-string ('s') meint eine Zeichenkette, welche nullterminiert und ebenso wie die Adresse gegebenenfalls mit Null-Bytes aufgefüllt wird, um ein vielfaches von 4 Bytes als Länge zu erhalten

OSC-blob ('b') schließlich ist ein generischer Datentyp, welcher sich aus 4 Byte Anzahl folgender Bytes, folgenden <Anzahl> Bytes und wiederum gegebenenfalls 0 bis 3 Null-Bytes zusammensetzt

Mit OSC 1.1 kommen folgende Datentypen hinzu:

TRUE ('T') für den Wahrheitswert wahr - hier werden keine Daten übermittelt, sondern nur die Adresse einmal angesprochen.

FALSE ('F') für den Wahrheitswert falsch - auch hier werden keine Daten übermittelt, sondern nur die Adresse einmal angesprochen.

IMPULSE ('I') für einen Impuls, der ein Ereignis auslöst (in der Robotik sehr nützlich)

Nachrichten könen auch zusammengesetzt in einem sogenannten Bundle verschickt werden, welches als Adresse das Wort „#bundle“ enthält, gefolgt von einem OSC-timetag und einer Menge von OSC-Nachrichten oder weiteren Bundles. Diese Nachrichten werden beim Empfänger so behandelt, als wären sie gleichzeitig angekommen.

Da die Adressen im OSC-Protokoll nicht vorgegeben sind, sondern im Gegenteil von jedem Hersteller bzw. Programmierer selbst frei definiert werden können, kann das zu unübersichtlichen Adressbäumen führen kann. Wenn hinreichen viele Geräte beteiligt sind, implementiert OSC reguläre Ausdrücke (Wildcards) zur Adressierung und verteilt Nachrichten an solche regulären Adressen an die entsprechenden Adressen.

Weiterhin implementiert OSC eine Art Request-Response-System, mit dem Controller verfügbare Adressen sukkzessive erfragen können. Dies geschieht, indem an die Adresse ein „/“ angehängt wird. Solche Nachrichten beantwortet das System mit einer Nachricht über verfügbare Subadressen unter der gefragten Adresse. Für ein Request-Response-System muss jeder Teilnehmer identifizierbar sein. Da OSC allerdings unabhängig vom Übertragungsmedium sein soll, wird in der Spezifikation nicht weiter darauf eingegangen, wodurch ein Teilnehmer identifiziert wird. Es wird lediglich beschrieben, dass es eine Identifizierung möglich sein muss. Bei Netzwerkimplementierungen wird daher meist die IP-Adresse genutzt.

Pro Nachricht in einem OSC System können verschiedene Argumente übergeben werden (beispielsweise erst eine Ganzzahl, dann eine Fließkommazahl und anschließend eine Zeichenkette). Der Sender, der an diese Adresse senden möchte, muss darüber Kenntnis haben, in welcher Reihenfolge wie viele Argumente geschickt werden müssen. Um dies zu erleichtern wird der Adresszusatz „/type-signature“ definiert, welcher als Antwort die erwarteten Argumenttypen in der vorgegebenen Reihenfolge zurückliefert. Zusätzlich gibt es auch noch das vordefinierte Adressanhängsel „/documentation“, welches zu jeder Adresse eine menschenlesbare Dokumentation der angefragten Adresse liefern soll. Schlussendlich die wohl wichtigste Anfrage, ist die nach „/current-value“, welche die aktuellen Werte der Adresse zurückliefert. [WF97]

Aus verschiedenen Gründen ist OSC heute noch nicht so weit verbreitet, wie MIDI:

- Das Verbauen von Netzwerkhardware ist momentan teurer, als eine einfache serielle Schnittstelle wie bei MIDI.
- OSC ist nicht bzw. nur umständlich abwärtskompatibel zu MIDI - OSC und MIDI Hardware kann nicht beliebig kombiniert werden.
- Einem Großteil von Musikern genügt MIDI weiterhin für ihre Anwendungen

Dennoch bietet OSC einige Vorteile gegenüber MIDI:

- wesentlich höhere Auflösung der Daten
- menschenlesbare Adressierung - einfachere Konfiguration
- per Referenzimplementierung bereits netzwerkfähig

OSC Bibliotheken

OSC kann aus vielen gängigen Programmumgebungen wie PureData[1], MaxMSP[2], SuperCollider[3], Processing [4] und weiteren heraus genutzt werden. Auch im mobilen Bereich ist es mit TouchOSC[5] für iOS[6] und Android[7] und GoOSC[8] für MeeGo[9] bereits in Form von sehr gut konfigurierbaren Anwendungen einsetzbar.

Auch für die Softwareentwicklung stehen viele Bibliotheken für die verschiedensten Programmiersprachen zur Verfügung. Drei davon sollen kurz vorgestellt werden.

WOscLib

Die WOscLib[10] basiert auf dem OSC-Kit[11], welches von Matt Wright (einem der Erfinder von OSC) geschrieben wurde. Da das OSC-Kit seit 2004 nicht mehr weiter entwickelt wird und einige Limitationen mit sich bringt, wurde WOscLib als Neuimplementierung

[1] `http://www.puredata.info`
[2] `http://cycling74.com/products/max/`
[3] `http://supercollider.sourceforge.net/`
[4] `http://processing.org/`
[5] `http://hexler.net/docs/touchosc`
[6] `http://de.wikipedia.org/wiki/Apple_iOS`
[7] `http://de.wikipedia.org/wiki/Android_%28Betriebssystem%29`
[8] `http://goosc.sourceforge.net/`
[9] `http://de.wikipedia.org/wiki/Meego`
[10] `http://wosclib.sourceforge.net/`
[11] `http://archive.cnmat.berkeley.edu/OpenSoundControl/Kit/`

des OSC-Kit gestartet und statt in C in C++ geschrieben. Dadurch ist die WOscLib modularer und flexibler nutzbar als das OSC-Kit.

Oscpack

Oscpack[12] ist eine weitere in C++ programmierte Bibliothek zum Erstellen, Packen und Entpacken von OSC-Nachrichten. Darüber hinaus stellt es einfache Klassen für die Socket-Kommunikation via UDP zur Verfügung.

liblo

Liblo[13] bezeichnet eine „leightweight OSC implementation“ (also eine schlanke OSC Implementation) in C. Sie unterstützt sowohl das Erstellen und Dekodieren, wie auch das Senden und Empfangen von Nachrichten und implementiert darüber hinaus entsprechende Callbackmechanismen, um die Nachrichten an die vom Programmierer anzugebenden Methoden weiter zu leiten, die die empfangenen Daten auswerten. Außerdem unterstützt die liblo auch Wildcards.

Mit pyliblo[14] gibt es zur liblo auch ein sogenanntes Python-Binding.

Vergleich

Alle getesteten Bibliotheken unterstützen alle gängigen Betriebsysteme (Linux, MacOS und Windows). Die WOscLib unterstützt zwar Wildcards und Bundles, ist aber sonst von den Datentypen her recht beschränkt. Abgesehen davon ist das Matching von Wildcards auf Methoden insofern unelegant gelöst, als dass jede Methode als eigene Klasse realisiert wird, die nur eine Methode (`Method()`) enthält. Dies zwingt den Programmierer zum Erzeugen einer Klasse je Parameter, der gesteuert werden soll.

Oscpack unterstützt zwar ebenso viele Datentypen wie liblo (und einige mehr als in Tabelle 2.1 aufgelistet, die allerdings liblo auch unterstützt) und ist nach Ermessen des Autors einfacher zu bedienen als WOscLib und auch liblo, besitzt aber keine Methoden zum Matching von Wildcards oder Adressen generell. Stattdessen werden auch in den

[12] `http://code.google.com/p/oscpack/`
[13] `http://liblo.sourceforge.net/`
[14] `http://das.nasophon.de/pyliblo/`

	WOscLib	oscpack	liblo
Datentypen			
i	x	x	x
b	x	x	x
s	x	x	x
f	x	x	x
t	x	x	x
T		x	x
F		x	x
I		x	x
d (double)		x	x
S (Symbol)		x	x
N (Null)		x	x
m (MIDI)		x	x
weiteres			
Bundles	x	x	x
Wildcards	x		x

Tabelle 2.1.: Vergleich der OSC Bibliotheken

Beispielen zu dieser Bibliothek nur Zeichenkettenvergleiche angewandt. Zwar muss an der Stelle keine Klasse pro Parameter erzeugt werden (wie bei der WOscLib), aber dafür müssen die entsprechenden Klassen zum Adress- und Wildcard-matching selbst implementiert werden.

2.3.5. RTP

Das Real-Time Transport Protocol (RTP) ist ein Protokoll für den Austausch von Live-Daten und die hier zusammengetragenen Informationen stammen im Wesentlichen aus dem RFC 3550 [Sch+03]. Es ist entwickelt worden, um Konferenzsysteme zur Übertragung von Audio- und Videodaten zu ermöglichen und entsprechenden Problemen mit Bandbreitenbeschränkungen, Verzögerungen, Datenkompression und Synchronität zu begegnen. Dabei bildet RTP prinzipiell zunächst nur die protokollseitige Basis mit vorgefertigten Pakettypen, Prüfsummen, Zeitstempeln, Sequenznummern und der Möglichkeit, die Zustellung von Paketen zu überprüfen. Wie auch OSC wird RTP zumeist über UDP genutzt, obwohl auch hier andere Protokolle möglich sind. Es gibt im UDP keine Zustellgarantie - somit wurde auch beim Design von RTP nicht davon ausgegangen, dass jedes Paket generell ankommt und auch nicht davon, dass ankommende Pakete in

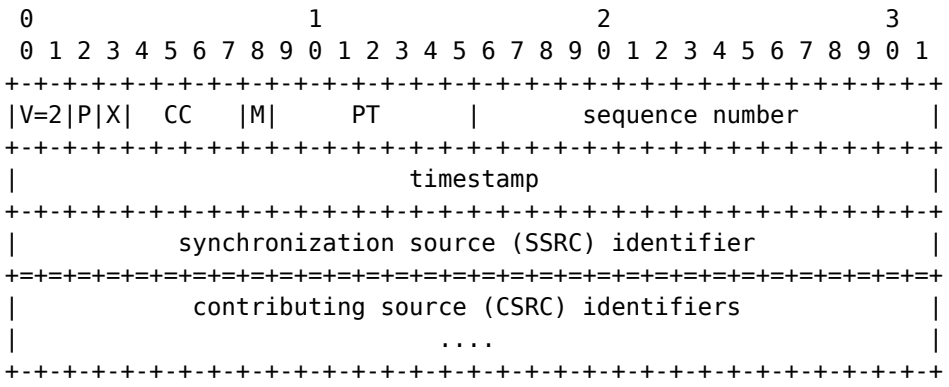


Abbildung 2.4.: Aufbau des RTP-Headers [Sch+03]

der richtigen Reihenfolge ankommen. Durch Sequenznummern kann jederzeit ermittelt werden, wie viele Pakete fehlen und in welcher Reihenfolge ankommende Pakete abgearbeitet werden müssen. Trotzdem implementiert RTP keinen Mechanismus zum erneuten Anfragen verlorener Pakete. In vielen Anwendungsfällen, wie beispielsweise Audio- oder Videoübertragung, ist dies auch nicht zielführend, da letztlich eine erneute Anfrage eines verlorenen Pakets nur unnötige Latenz in das System bringen würde. RTP besteht im Wesentlichen aus zwei Komponenten: dem eigentlichen Transport Protocol (RTP), sowie einem Steuerungsprotokoll (RTP Control Protocol (RTCP)). Während Ersteres zur Übertragung der eigentlichen Daten dient, wurde das Zweite erdacht um die Qualität der Übertragung zu überwachen und Informationen über die Teilnehmer der Sitzung zu übermitteln.

Datenformat

Die Übertragung der eigentlichen Daten findet in Form von RTP Paketen statt, welche im Wesentlichen aus einem Header und dem Payload bestehen. In diesem Header sind verschiedene Informationen und vor allem PT - Payloadtype, Sequenznummer und Zeitstempel kodiert. Darüber hinaus enthält jedes Paket eine eindeutige SSRC Nummer, welche die Quelle des Pakets angibt. Sie ist unabhängig von der IP-Adresse des Senders und eindeutig pro RTP-Session. Verschiedene Payloadtypes für Video- und Audioformate werden im RFC 3551 [SC03] definiert.

Das RTCP Protokoll wird genutzt, um periodisch Kontrollpakete an alle Teilnehmer einer RTP Session zu senden.

Eine Sitzung wird normalerweise immer für genau ein zu übertragendes Medium eröffnet:

Bei Übertragung von Audio und Video wird je ein Port für RTP als auch RTCP geöffnet (eine RTP Session), über den die jeweiligen Medien bzw. die zugehörigen Steuerinformationen übertragen werden. So kann gewährleistet werden, dass verschiedene Teilnehmer, von denen manche nur das Audio- und manche nur das Videosignal empfangen wollen, auch nur dieses empfangen. Mittels der übertragenen Timing-Informationen aus den RTCP-Paketen kann anschließend eine synchrone Wiedergabe der beiden Streams erreicht werden.

Erweiterbarkeit und MIDI über RTP

Wie bereits oben erwähnt wurde RTP durch definierbare Payload Types (PT in Abbildung 2.4) erweiterbar entworfen. Es ist nicht auf einen bestimmten zu übertragenden Datentyp (RTP media types) festgelegt. Durch diese Eigenschaft kann auch ein Datenformat definiert werden, das Audio- und Videosignale in einem RTP media type koppelt und somit nur eine RTP Session eröffnet werden muss (um weniger Netzwerkports zu benötigen), was allerdings auch bei oben genanntem Szenario zum Nachteil des Zwangs für jeden Client führen kann, nun an Audio- und Videokonferenz teilzunehmen und damit womöglich an die Grenzen der Bandbreite zu stoßen. Gerade zur Bekämpfung des Bandbreitenproblems gibt es mit RTCP die Möglichkeit, Qualitätskriterien des Dienstes festzulegen.

Darüber hinaus implementiert RTP weitere Mechanismen wie Mixer, Monitore und Translator, welche hier nicht näher diskutiert werden sollen, die allerdings weitere Flexibilitäten eröffnen. Eine Ausführliche Beschreibung findet sich in [Sch+03].

In RFC 6295 [LW11] wird ein Payload Type für MIDI-Nachrichten für RTP definiert. Dieser wurde zuvor in [LW01] bereits vorgestellt und ebnet MIDI so nicht nur den Weg zur Netzwerktauglichkeit sondern auch für das kollaborative Arbeiten mit MIDI über Netzwerk. Da der Verlust eines MIDI-Befehls um einiges schlimmer für das Gesamtsystem sein kann und RTP aber keine Mechanismen für das Anfragen verloren gegangener Pakete bietet, wird neben der Umsetzung der MIDI-Befehle in RTP-Paketen auch ein ausgereiftes Journaling-System definiert. Mit diesem wird es möglich, verloren gegangene Pakete beim Transport von RTP beispielsweise über UDP zu rekonstruieren. Es wird so etwa verhindert, dass durch den Verlust eines NoteOff-Events eine Note unendlich lang gespielt wird.

RTP Bibliotheken

RTP wurde in den letzten Jahren vor allem für Voice over IP (VoIP) und ähnliche Konferenzsysteme genutzt. Neben einigen Bibliotheken, die auf bestimmte Anwendungsfälle optimiert sind: beispielsweise das Robust Audio Tool (RAT)[15], wie auch die JVOIPLIB[16] und weitere für die Nutzung im Bereich VoIP, oder die live555[17] für die Nutzung als Bibliothek für RTSP-Applikationen (RTSP wird vor allem genutzt, um verschiedene Streams zur Verfügung zu stellen - nicht direkt für Konferenzsysteme - nutzt aber auch RTP Datenformate).

Andererseits gibt es auch speziell Bibliotheken zur Programmierung eigener RTP Anwendungen. Im Folgenden sollen drei Bibliotheken kurz vorgestellt und anschließend verglichen werden.

ccRTP

ccRTP[18] ist eine Implementierung von RTP nach den Request for Comments (RFC) Standards 3550, 3551 und 3555. Dabei bildet ccRTP ein C++ Framework für die Programmierung von RTP-Anwendungen. Es bietet neben fest definierten Datenformaten, für die allerdings keine Encoding oder Decoding Funktionalitäten zur Verfügung gestellt werden (lediglich die Payload Types sind definiert), seit einiger Zeit auch in einer separaten Zusatzbibliotek ZRTP. Diese ist eine spezielle Erweiterung für den Aufbau von RTP Sessions, bei der Schlüssel nach dem Diffie-Hellman-Verfahren ausgetauscht werden, um anschließend eine verschlüsselte SRTP Session zu initiieren. Näheres dazu ist auch in Zimmermann, Johnston und Callas [ZJC11] beschrieben.

rtplib

Die ursprüngliche RTP Bibliothek wurde unter anderem von einem der Autoren (Henning Schulzrinne) des RFCs zu RTP entwickelt. Sie liegt seit 2006 als Version 1.0a1 vor, während die Dokumentation seit 1998 nicht weiter gepflegt wurde. Sie ist in C geschrieben und bietet sowohl eine highlevel, als auch eine lowlevel Schnittstelle an.

[15] http://www-mice.cs.ucl.ac.uk/multimedia/software/rat/
[16] http://research.edm.uhasselt.be/~jori/page/index.php?n=CS.Jvoiplib
[17] http://live555.com/
[18] http://www.gnu.org/software/ccrtp/

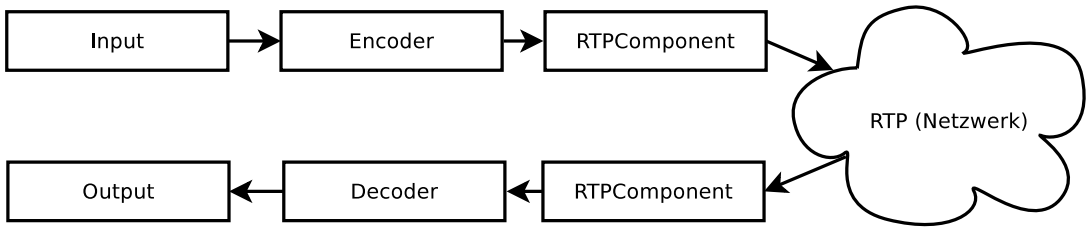


Abbildung 2.5.: Schematische Darstellung des Chainingkonzepts der EMIPLIB

JRTPLIB und EMIPLIB

Die JRTPLIB ist eine in C++ geschriebene RTP Bibliothek. Sie abstrahiert sämtliche Kommunikation in verschiedenen Klassen und behandelt RTCP Kommunikation komplett intern.

Eine Erweiterung zur JRTPLIB ist die EDM Media over IP library (EMIPLIB)[19]. Die JRTPLIB bietet an sich nur Klassen und Methoden zum Senden und Empfangen von RTP-Paketen. Es werden aber keinerlei Payload Formate vorgegeben oder entsprechende Codecs eingebunden.

Durch die EMIPLIB wird ein Chainingkonzept umgesetzt (vgl. Abbildung 2.5), welches verschiedene Codecs für Audio und Video unterstützt. Chaining (also zu deutsch *Verketten*) bedeutet dabei, dass verschiedene Komponenten miteinander verbunden werden und Signale an einem Ende eingegeben werden, von dieser Kette umgewandelt und anschließend wieder ausgegeben werden.

Es ist mit der EMIPLIB nicht nur möglich, einen Chain zu erstellen, der lokal Daten von einer Quelle aufnimmt, sie mittels verschiedener aneinandergereihter Komponenten umwandelt und dann wiedergibt, sondern mit der RTPComponent ist es ebenso möglich, die generierten Nachrichten über RTP zu verschicken. Quellen können dabei unter anderem Dateien auf der Festplatte, oder auch aufgenommene Daten von einer Kamera oder Soundkarte sein.

Dabei übernimmt die RTPComponent sowohl um das Senden, als auch das Empfangen von Paketen per RTP, sowie sämtliche RTCP Kommunikation zwischen den Teilnehmern. Diese RTPComponent ist gleichzeitig das Bindeglied zwischen EMIPLIB und JRTPLIB. Der Signalweg in Abbildung 2.5 kann sowohl auf einem Rechner allein aufgebaut und genutzt werden, als auch auf mehreren Rechnern gleichzeitig.

[19] `http://research.edm.uhasselt.be/emiplib/emiplib.html`

Vergleich

Während sowohl die ccRTP, wie auch JRTPLIB und die ursprüngliche rtplib zunächst nur Funktionalität zum Senden und Empfangen von RTP Paketen zur Verfügung stellen, bietet die JRTPLIB mit ihrer Erweiterung EMIPLIB die wohl umfangreichsten Möglichkeiten, RTP gleich mit unterschiedlichen Codecs zu nutzen und ggf. um weitere zu erweitern.

2.4. Stand der Technik

In verschiedenen Installationen wurden bereits Medienfassaden in interaktive Medienfassaden verwandelt, indem ein oder mehrere Sensoren verbaut wurden.

Tower of Winds Beim Tower of Winds in Japan (1986) werden Umweltparameter genutzt (in dem Fall Windgeschwindigkeit und Richtung, sowie Lautstärkepegel der Umwelt und Lichtintensität) um die Illumination des Gebäudes zu steuern. [Häu09, S. 69]

Blinkenlights Bei der Installation anlässlich des 20. Jahrestages der Gründung des Chaos Computer Clubs in Berlin wurde das Berliner Haus des Lehrers mittels Fensterrasteranimation in eine interaktiv über Handy nutzbare Medienfassade verwandelt.[bli]

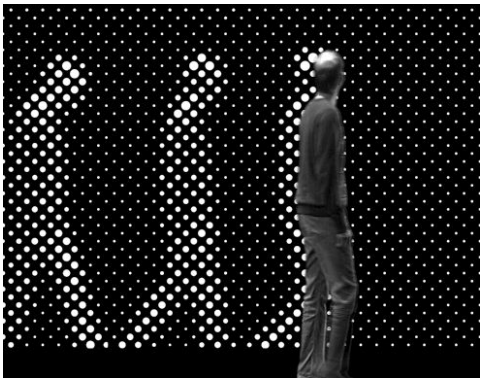
Aperture TheGreenEyl aus Berlin haben ein System entworfen, bei dem sich irisförmig von hinten beleuchtet runde Löcher in der Fassade entweder animiert oder anhand von Schattenwürfen auf die Oberfläche (also auch in Abhängigkeit der Lichtintensität) öffnen und schließen.[The]

ARS Electronica Center Am Linzer ARS Electronica Center wurde eine Installation geschaffen, an der mehrere Personen über ihre Smartphones auf die Fassade „malen“ können.[Bor+11]

Spread.Gun und SMSlingshot Die in Fischer, Zöllner und Hornecker [FZH10] und Fischer und Hornecker [FH12] vorgestellen Lösungen sind Applikationen, mit denen Textnachrichten von einem mobilen Endgerät aus auf Medienfassaden geschossen werden können.

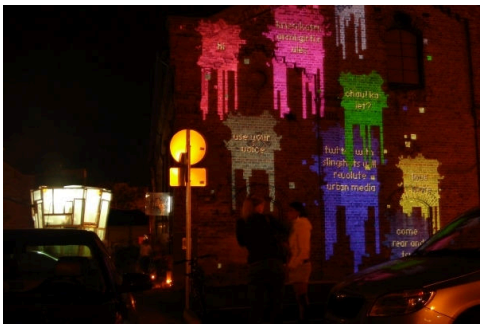
(a) Tower of Winds [Oha]

(b) Blinkenlights [GH]

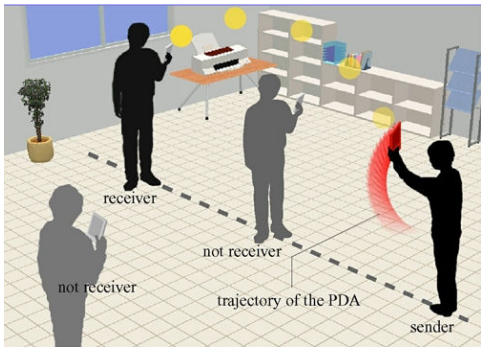


(c) Aperture [The]

(d) Ars Electronica Center [FL]



(e) SMSlingshot [Bie]



(f) Toss-It [Sug]

(g) BBC Big Screen [OGR08]

Abbildung 2.6.: verschiedene interaktive Installationen

Toss-It Die Anwendung Toss-It aus Yatani u. a. [Yat+05] ermöglicht es mittels Gesten Daten zu übertragen. Dieser Anwendung in ihrer Grundfunktionalität ist auch Hoccer[20] sehr ähnlich. Prinzpiell ist es damit auch sinnvoll und möglich, Daten an Medienfassaden zu übertragen.

BBC Big Screen Im Rahmen dieses Projektes wurden in vielen britischen Großstädten Großbildschrime aufgestellt, an denen auch interaktive Installationen vorgenommen wurden, wie in O'Hara, Glancy und Robertshaw [OGR08] beschrieben ist.

Die angeführten Systeme sind in Abbildung 2.6 jeweils dargestellt.

Es werden dabei bereits verschiedene Eingabeschnittstellen deutlich: beim Tower of Winds ist es die Umgebung, welche gemessen wird. Beim Blinkenlights-Projekt war es eine Website, über die Animationen erstellt werden konnten, die anschließend per SMS vom Handy bzw. Anruf für interkative Anwendungen aktiviert und gesteuert werden können. Bei Aperture ist es wieder die Umgebung und beim ARS Electronica Center ist es ein Smartphone, welches per Augmented Reality und Distant Pointing (wie in Vogel und Balakrishnan [VB05] und Bateman u. a. [Bat+11] beschrieben) Punkte auf der Fassade bemalt. Es wird deutlich, dass jeweils sehr unterschiedliche Eingabemethoden und Schnittstellen genutzt werden. Dies lässt sich hier vor allem mit verschiedenen Hintergründen der Applikationen, verschiedenen Daten, die übertragen werden und auch der verschiedenen Zeit der Entstehung der Anwendungen erklären. Die Applikation am Tower of Winds wurde zur „Verschönerung" eines Ablufturms für ein Hochhaus entworfen, Blinkenlights war eine temporäre Installation mit klarem Computerbezug, Aperture ist eine Designstudie und die Installation am ARS Electronica Center entstand aus HCI[21] Projekten.

Wie in Fischer, Zöllner und Hornecker [FZH10] auch noch einmal herausgestellt wird, arbeitet im Grunde jedes System mit einer anderen speziell für diesen Zweck entworfenen Hardware- und Softwareschnittstelle.

2.5. Diskusion vorhandener Schnittstellen

Für Medienfassaden können verschiedenste Schnittstellen von den oben genannten sinnvoll eingesetzt werden. Jede in ihrer Form bildet eine Möglichkeit, Daten unterschied-

[20] `http://www.hoccer.com`
[21] HCI: Human Computer Interaction

lichster Art zu übertragen und ist daher für unterschiedliche Einsatzzwecke mehr oder weniger geeignet.

Es kommt entscheidend auf den Inhalt der Medienfassade an und darauf, welche Formen der Interaktion zugelassen sein sollen. Will man eine generalisierte Schnittstelle schaffen sollen allerdings möglichst viele Anwendungsszenarien abdeckbar sein. Einfache Steuerprotokolle und -schnittstellen wie MIDI, DMX oder OSC allein bieten sich beispielsweise nicht an, um live-Daten wie Audio oder Video zu übertragen. Hingegen sind USB und RTP für einige Einsatzgebiete im Embedded-Bereich zu aufwändig.

Das MIDI Protokoll ist per Definition sehr stark auf Ansteuerung von Audiohardware spezialisiert und mit 31 250 Baud sehr langsam. Dennoch ist es über System Exclusive Messages nahezu beliebig erweiterbar, was jedoch eigentlich nicht im Sinne der MIDI Spezifikation ist.

DMX ist im Vergleich zu MIDI auch für andere als den vorgesehenen Zweck der Steuerung von Veranstaltungstechnik sehr gut einsetzbar, da es letztlich nicht entscheidend ist, welche Geräte welche Aktionen für die Werte ausführen, die übertragen werden (es gibt beispeislweise keine definierten NoteOn oder NoteOff-Events, Play oder Program Change).

OSC hat gegenüber MIDI und DMX den großen Vorteil, dass es für seinen Einsatzzweck als Steuerungsprotokoll sehr flexibel einsetzbar ist. Durch die beliebige Adressierbarkeit von Parametern in menschenlesbarer Form eignet es sich hervorragend zur erweiterbaren Konfiguration. Diese beliebige Adressierbarkeit bringt allerdings auch Nachteile mit sich: Einerseits wird dadurch ein im Vergleich zu MIDI, wo ein Nibble für die Kodierung der Adresse genutzt wird, und DMX, wo letztlich allein der zeitliche Versatz der Nachrichten zur Adressierung genutzt wird, ein Overhead erzeugt. Andererseits führt diese Beliebigkeit in der Wahl der Adressen und Parameterzusammensetzung zu einer großen Vielfalt unterschiedlicher Adressräume, die sich nicht mehr leicht von einem Controller abdecken lassen und vor allem von Gerät zu Gerät bzw. Anwendung zu Anwendung, die gesteuert werden soll, unterschiedlich sein können. Dem Problem der Adressvielfalt kann mit Wildcards entgegen gewirkt werden. Damit OSC mit seinen Vorteilen seine Nachteile überwiegen kann, ist ein hinreichend schnelles Übertragungsmedium notwendig.

RTP bietet vielseitige Möglichkeiten zum konferenzartigen Austausch von Informationen. Da es nicht auf einen Datentyp festgelegt ist, sondern sich prinzipiell beliebige Daten darüber übertragen lassen, ohne Standards zu verletzen, eignet es sich nicht nur

für die Übertragung von Audio- und Videostreams, sondern auch beispielsweise von MIDI-Daten. Darüber hinaus gibt es verschiedene Erweiterungen für RTP, die einige Nachteile auflösen können, wie beispielsweise SRTP für verschlüsselte RTP-Kommunikation und SIP, RTSP und SDP für Session Management.

3. Anforderungsanalyse – Forschungsaspekt

Aufbauend auf den Grundlagen aus dem vorherigen Kapitel werden Anforderungen an das Gesamtsystem hergeleitet und spezifiziert. Darüber hinaus soll erläutert werden, wo die Neuerung gegenüber bisherigen Ansätzen liegt und was das System letztlich ausmacht.

3.1. Umgebung – Hintergrundinformationen

Ziel dieser Arbeit ist es, eine generalisierte Schnittstelle zu schaffen, durch die mit Medienfassaden und Anwendungen auf diesen interagiert werden kann. Dabei geht es vorrangig darum, eine Schnittstelle zu finden, oder zu definieren, die möglichst nicht nur auf eine Medienfassade, einen Anwendungsfall oder sogar nur auf eine Anwendung spezialisiert ist. Als konkrete Umgebung für diese Arbeit wird die Medienfassade am Forschungs- und Weiterbildungszentrum für Kultur und Informatik genutzt. Diese entsteht momentan am Spreecampus (Wilhelminenhofcampus) der HTW Berlin. Sie wird aus LED-Grid Modulen und Rückprojektionen hinter den Fenstern sowie einzeln ansteuerbaren LEDs am Rest der Nordfassade bestehen (vergleiche Abbildung 3.1). Zum Zeitpunkt des Schreibens dieser Arbeit ist die Fassade noch nicht fertiggestellt. Dennoch sind einige Komponenten (LED-Grid Module und Rückprojektion), welche von Anfang an in der Planung waren, in einem Testlabor aufgebaut worden und erste prototypische Anwendungen können getestet werden (siehe dazu Abbildung 3.2 und Abbildung 3.3). Im Testlabor wird zwar eine Rückprojektionsfolie genutzt, es wird jedoch aus Platzgründen mit dem Beamer darauf eine Aufprojektion vorgenommen (vgl. Abbildung 3.3a).

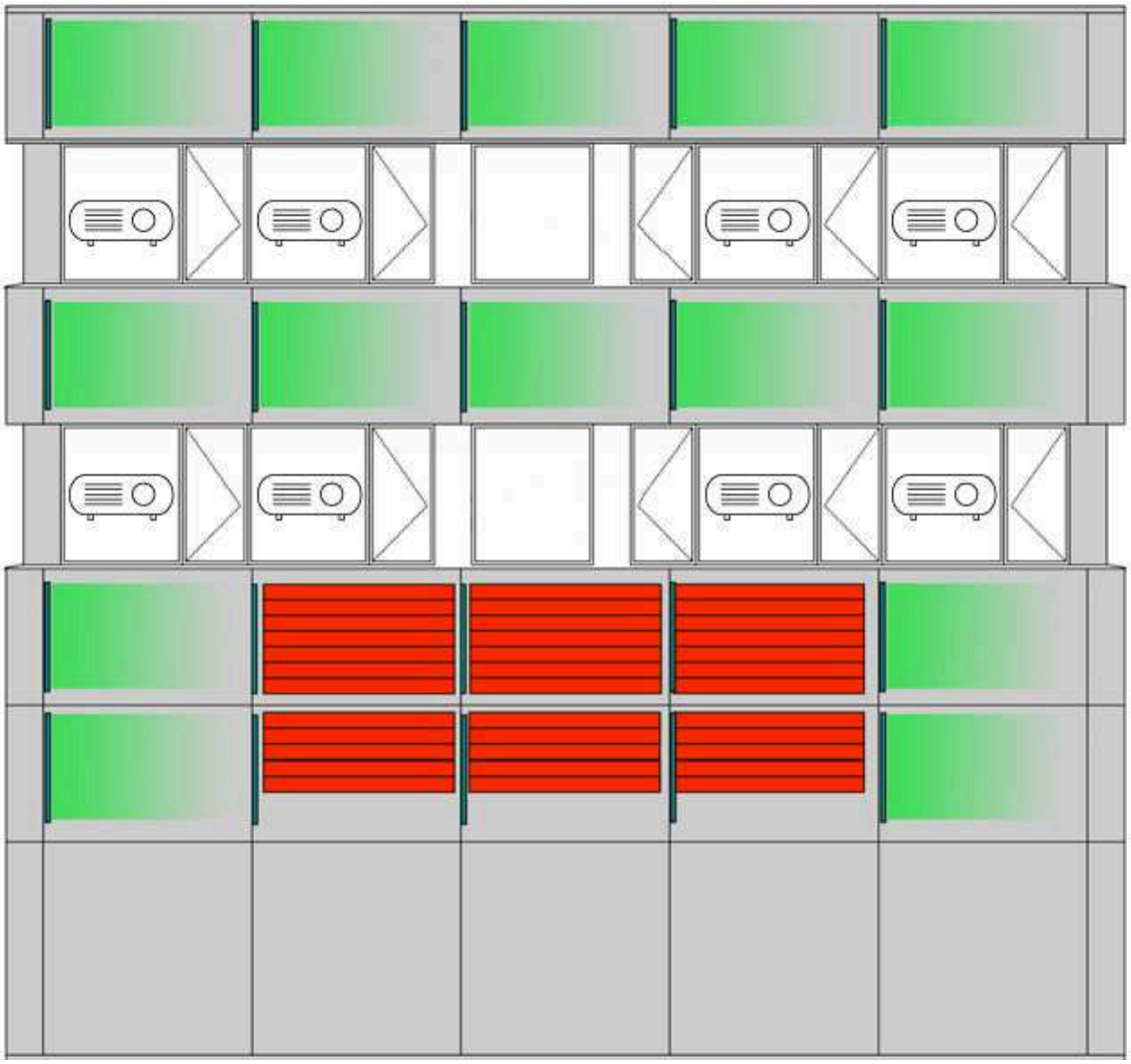


Abbildung 3.1.: Planungsskizze der Medienfassade mit acht Beamern für die Rückprojektion hinter den Fenstern, Ambient-Light-artige senkrechte LED-Ketten (grün gekennzeichnet und nach rechts scheinend), sowie den LED-Grid-Modulen (Rot gekennzeichnet)

Softwareseitig wurde bereits in Bergemann und Schlegel [BS12] evaluiert, welche Möglichkeiten insbesondere für die Darstellung von Inhalten auf mehreren Ausgabemedien unterschiedlicher Auflösung gleichzeitig existieren. Dabei wurde herausgestellt, dass ClusterGL[1] momentan eine sehr gute Lösung darstellt, um OpenGL[2] Anwendungen verteilt auf verschiedenen Rechnern zu rendern und darzustellen.

[1] `http://clustergl.org/`
[2] `http://www.opengl.org/`

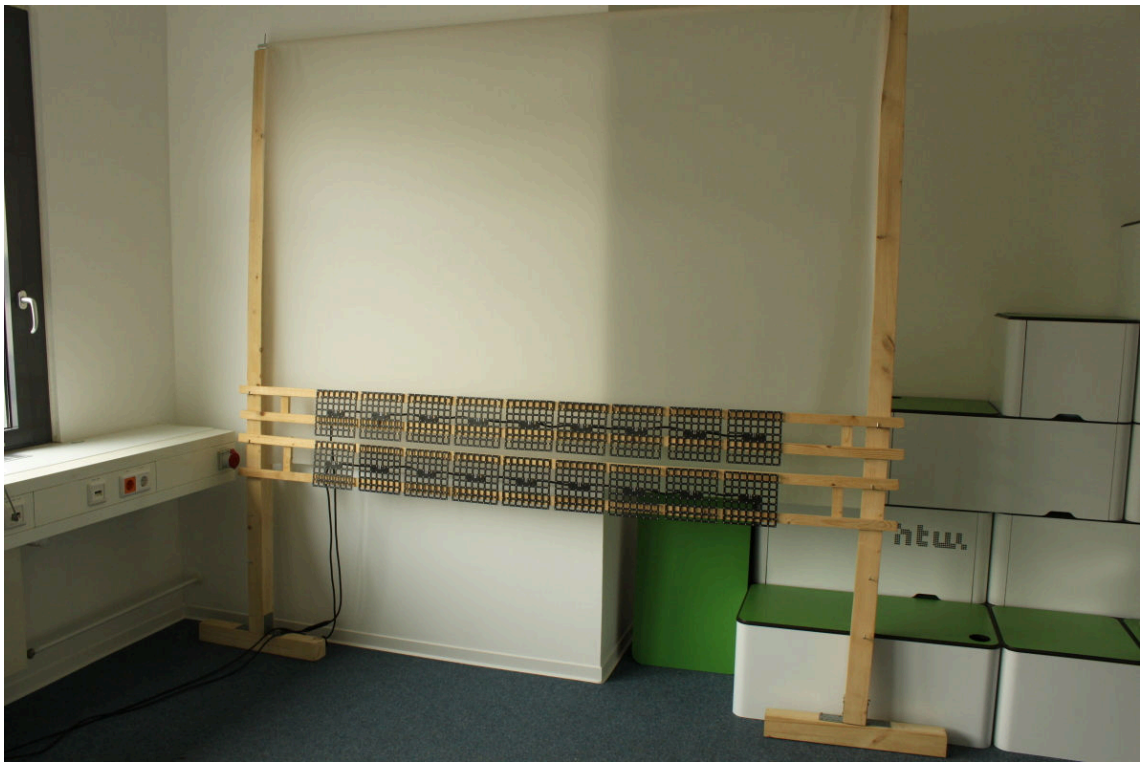
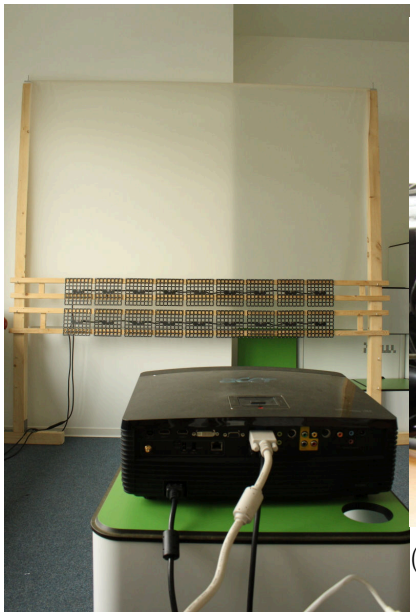


Abbildung 3.2.: Testaufbau mit Rückprojektionsfolie und LED-Modulen



(a) Testaufbau mit Beamer

(b) LED Steuerungsmodule des Testaufbaus (Kontrollmonitor oben (angeschnitten), DVI-Controller darunter, LED-Controller ganz unten)

Abbildung 3.3.: weitere Komponenten des Testaufbaus

(a) Testaufbau in Betrieb (b) Monitor bei Testbetrieb

Abbildung 3.4.: Testaufbau mit Testapplikation

Eine Testanwendung eines rotierenden Würfels, demonstriert in Abbildung 3.4, dass es mit ClusterGL möglich ist, auf mehreren unterschiedlich auflösenden Ausgabemedien eine OpenGL-Anwendung zu rendern. In Abbildung 3.4b ist zu sehen, wie das Bild vom DVI-Controller (aus Abbildung 3.3b) auf die LED-Module umgerechnet wird: es wird ein Ausschnitt des Gesamtbildes eines 1024×768 Pixel großen Monitors auf ihnen abgebildet (in Abbildung 3.4b durch eine kleine rote Fläche nahe der linken oberen Ecke des Monitor gekennzeichnet).

In Vorbereitung dieser Arbeit wurde auch das in Bergemann und Schlegel [BS12] beschriebene Speicherloch[3] gefunden und behoben. Somit läuft ClusterGL auf der im Testlabor installierten Konstruktion nun ohne allozierten Speicher nie wieder freizugeben. Dies würde bei längerer Ausführung eines Programmes grundsätzlich dazu führen, dass der Arbeitsspeicher des Rechners irgendwann voll liefe und das System bis zur Unbrauchbarkeit ausgebremst würde.

OpenGL, welches die Grundlage für ClusterGL darstellt, bietet zunächst keine allgemeine Schnittstelle zur Interaktion mit der Anwendung. Maus- und Tastatureingaben können über entsprechende Erweiterungsbibliotheken zwar genutzt werden, aber Smartphones und weitere alternative Eingabegeräte sind kaum mit der Standardfunktionalität zu integrieren. Daher bringt auch CusterGL keine solche Schnittstelle mit sich. Es dient ausschließlich dem über Netzwerk verteilten Rendern von OpenGL-Applikationen auf

[3]Speicherlöcher sind Teile im Programm, die (schlimmstenfalls immer wieder) dynamisch Speicher vom Betriebssystem angefordern, dessen Speicheradresse im weiteren Verlauf überschrieben wird, ohne dass der Speicher selbst wieder freigegeben wird. Der Speicherbereich ist nun bis zum Beenden des Programms weder für das Betriebssystem, noch für das Programm nutzbar.

unterschiedlichen Rechnern, an die jeweils Ausgabemedien angeschlossen sind. Noch dazu können diese Anwendungen beim momentanen Stand der Entwicklung nur X[4] oder die Klassen der Simple DirectMedia Layer (SDL)[5] als Fensterklassen nutzen, um das Programm darin darzustellen. Dies ist beim Entwurf der Testanwendungen zu bedenken.

Ein Vorteil der Verwendung von ClusterGL für den Entwickler einer Applikation für die Medienfassade liegt darin, dass die Programmierung komplett transparent ist: der Entwickler implementiert eine OpenGL-Anwendung lokal, als würde er sie auch nur lokal laufen lassen, kann sie lokal testen und optimieren, die Interaktionsmöglichkeiten implementieren und die Anwendung komplett ohne eine Medienfassade fertig implementieren. Anschließend wird die Anwendung nur in der ClusterGL-Umgebung gestartet. Dies geschieht indem statt der OpenGL-Bibliothek zu nutzen die ClusterGL-Bibliothek genutzt wird, die exakt dieselben Funktionssignaturen zur Verfügung stellt, aber nicht direkt die Funktionen von OpenGL ausführt, sondern die Verteilung der Befehle auf die verschiedenen Ausgabemedien der Fassade übernimmt. Auf diesen Rechnern laufen sogenannte Renderer, welche auch von ClusterGL zur Verfügung gestellt werden, die die Befehle entgegennehmen und die entsprechende OpenGL-Funktion ausführen.

Dennoch ist bei der Kombinatoin von Technologien zu beachten, dass die Anwendung auf der Fassade in verschiedenen Bereichen unterschiedlich hochauflösend dargestellt werden kann.

3.2. Systemanforderungen

Die Medienfassade am FKI soll nicht nur einem Publikum Unterhaltung bieten, sondern auch Studenten, Entwicklern und Künstlern die Möglichkeit geben, ihre Ideen zu verwirklichen und auszuprobieren. Es wird also nicht nur eine Anwendung geben, die ständig auf der Medienfassade läuft. Sowohl auf Entwickler- als auch auf Anwenderseite ist es daher erforderlich, dass sämtliche Schnittstellen verständlich und einfach zu bedienen bzw. einzubinden sind. Darüber hinaus bringt die Anforderung, dass mehrere Entwickler bzw. Künstler die Medienfassade nutzen sollen zusätzlich die Anforderung mit sich, dass die Schnittstellen so gehalten werden, dass sie auf möglichst viele Anwendungsszenarien adaptierbar bzw. für neue Anwendungsszenarien einfach zu erweitern

[4] `http://www.x.org/`
[5] `http://www.libsdl.org`

sein müssen. Grundlegend ist daher zu beachten, dass viele mögliche Eingabemethoden und -geräte unterstützt werden.

Bisher wird bei der Entwicklung von Applikationen für Medienfassaden der Fokus vor allem auf eine bestimmte Form und je auch nur ein bestimmtes Eingabegerät zur Interaktion gelegt – sei es durch das Mobiltelefon mit Tasten, ein Touchscreen, einen Wii-Controller, eine Kinect oder ein Arduino[6]. Es existieren daher viele Insellösungen, die so zwar für die entsprechende Anwendung auf der entsprechenden Fassade funktionieren, eine Adaption der Lösung von Medienfassade A auf Medienfassade B würde allerdings in den meisten Fällen eine komplette Neuimplementierung bedeuten. Auch das Austauschen von Eingabegeräten oder die Nutzung verschiedener Eingabegeräte für verschiedene Applikationen würde einen hohen Aufwand mit sich bringen und in den meisten Fällen eine Neuimplementierung der Anwendung erfordern.

Für die Applikationen der Fassade mag das noch hinnehmbar sein – wenn sie so gut mit dem Gebäude harmonieren wie nur irgend möglich, werden sie vermutlich auf einem anderen Gebäude gar nicht entsprechend wirken. Für die Nutzerschnittstelle ist es allerdings nicht hinnehmbar, dass Benutzer pro Fassade eine andere Applikation starten müssen, ein anderes Protokoll sprechen müssen, sich anders zur Medienfassade verbinden und womöglich authentifizieren müssen. Daher ist Grundziel dieser Arbeit, solche Schnittstellen zu Medienfassaden zu vereinheitlichen und somit einem Nutzer, der mit einer Medienfassade interagieren möchte, die Möglichkeit zu geben, auch mit anderen Medienfassaden zu interagieren, ohne selbst viel neu erlernen, oder neu installieren zu müssen.

Ist diese Benutzerinteraktion abstrahiert, wird ist es auch möglich, dass Medienfassaden untereinander kommunizieren – dass also beispielsweise ein Element einer Medienfassade nahtlos auf eine andere Medienfassade erweitert werden kann oder Ähnliches. Weiterhin ist damit gegeben, dass auch fest installierte Eingabegeräte eingebunden werden können – sie müssen lediglich dieselbe Schnittstelle bedienen und sind im Gegensatz zu einem temporären Benutzer ständig vorhanden. Dies kann vor allem in Fällen interessant sein, wo Nutzer der Applikationen erst gar keine Eingabegeräte mitbringen sollen, weil beispielsweise eine Kamera an der Fassade installiert ist, die über Tracking von Personen die Steuerung der Anwendung(en) übernimmt.

[6] `http://www.arduino.cc`

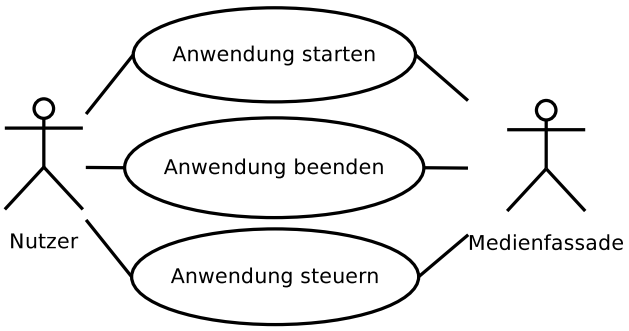


Abbildung 3.5.: Generelle Anwendungsfälle für eine interaktive Medienfassade

Aus Abbildung 3.5 gehen grundlegende Funktionsanforderungen an interaktive Anwendungen einer Medienfassade hervor.

Es soll dem Nutzer möglich sein, mit einer Applikation (sei es auf seinem selbst gebauten Hardware Controller oder auf dem Smartphone) Parameter der laufenden Software auf der Medienfassade zu beeinflussen. Solche Parameter können unter anderem die Farbe, die Position oder Größe von Elementen, oder auch die Geschwindigkeit einer Animation sein. Es könnte aber genauso gut ein Text sein, der an einer bestimmten Stelle visualisiert werden soll. Parameter stellen letztlich je eine Variable im ausgeführten Programm dar, die von außen direkt oder indirekt (z. B. nach Plausibilitätsprüfungen) manipuliert werden kann.

Darüber hinaus soll es aber auch möglich sein, Medien (Audio und Video) an die Fassade zu übertragen und möglichst auch live zu streamen. Sollen mehrere Medienfassaden bzw. deren Applikationen mit derselben Schnittstelle ausgestattet interaktiv nutzbar werden, erfordert dies eine zu entwerfende Infrastruktur im Hintergrund. Es muss abrufbar sein, welche Parameter welcher Anwendung an welcher Fassade wie angesprochen werden müssen. Anwendungen, die auf der jeweiligen Medienfassade vefügbar sind, müssen start-, steuer- und beendbar sein. Darüber hinaus müssen entsprechende Vorkehrungen zumindest vorgedacht werden, einem Missbrauch von laufenden Anwendungen oder Medienfassaden generell vorzubeugen. Zu diesen Missbrauchsszenarien zählen neben inhaltlichem Missbrauch für Gesetzesübertretungen auch schon solche Dinge wie das ungewollte „Übernehmen“ eines Parameters, dass eigentlich von einem anderen Teilnehmer gesteuert werden soll.

Wenn mehrere Medienfassaden mit derselben Schnittselle ausgestattet sind und somit die Entwicklung von sowohl Anwendungen für die Fassade, als auch für mobile Endgeräte vereinheitlicht werden kann, bietet sich eine zentrale Anlaufstelle für Medienfassadenbetreiber und Nutzer der Anwendung an. Bei dieser zentralen Anlaufstelle können alle

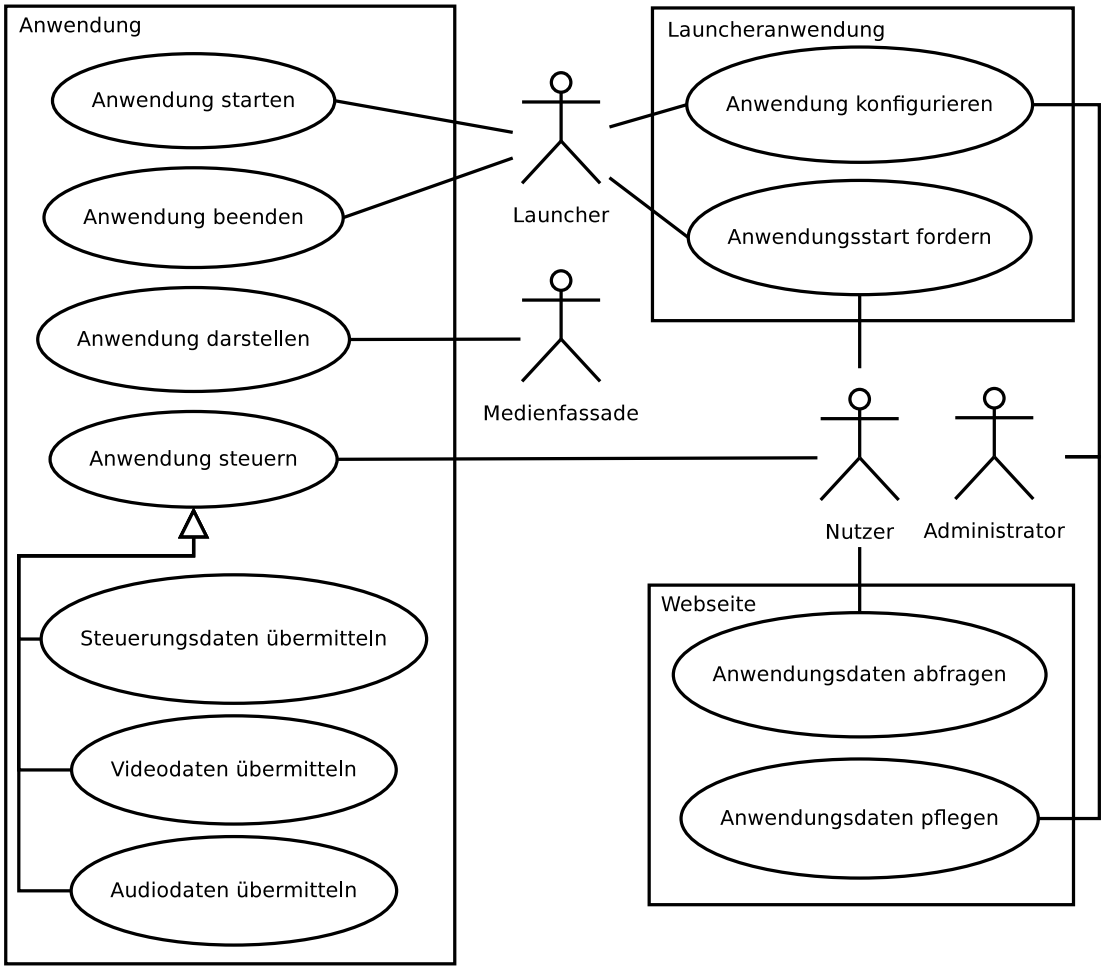


Abbildung 3.6.: Anwendungsfalldiagramm für eine generalisierte Schnittstelle zu Medienfassaden

Fassaden und Applikationen, die auf ihnen verfügbar sind, aufgelistet werden und je Anwendung deren Parameter, die sie zur Interaktion bereitstellt.

All diese zusätzlichen Eigenschaften führen zu einer größeren Menge an Anwendungsfällen, welche in Abbildung 3.6 aufgeführt sind. Darüber hinaus wurde in diese Abbildung bereits der sogenannte „Launcher“ eingebunden. Dieser dient als eine Art Daemon[7] zum Starten und Beenden von Medienfassadenapplikationen. Er ist von außen ansprechbar und von einem Administrator konfigurierbar. Ebenso wird bereits eine Website erwähnt, welche als zentrale Anlaufstelle für Entwickler von Steuerungs- und Medienfassadenapplikationen dient. Steuerungsapplikationen bezeichnen dabei Anwendungen, die beispielsweise auf mobilen Endgeräten vom Benutzer zur Interaktion mit der Medienfassadenapplikation genutzt werden.

[7]Daemon bezeichnet ein im Hintergrund ablaufendes Programm, welches Dienste zur Verfügung stellt

3.3. Zusammenfassung der Anforderungen

Zusammenfassend ergeben sich folgende Anforderungen:

- Medienfassadenanwendungen
  - Eine Medienfassade soll mehrere Applikationen zur Verfügung stellen können.
  - Jede Applikation soll vor Ort gestartet werden können.
  - Jede Anwendung läuft nur für eine begrenzte Zeit.
  - Es kann nur eine Applikation gleichzeitig auf einer Fassade laufen.
  - Jede Applikation soll für sich allein auch auf anderen Rechnern ausführbar sein (zum Beispiel für Entwicklung und Test).
  - Parameter einer Applikation sollen über eine einheitliche Schnittstelle manipulierbar sein.
- Parameter
  - Jede Applikation kann eine beliebige aber feste Anzahl von verschiedenen Parametern zur Verüfung stellen.
  - Parameter können in Form von Zahlenwerten, Buchstaben und Trigger-Befehlen dargestellt werden.
  - Jeder Parameter ist eindeutig adressierbar.
  - Mehrere Benutzer können gleichzeitig verschiedene Parameter beeinflussen.
- Schnittstelle
  - Über die Schnittstelle sollen alle Parameter einer Anwendung gesteuert werden können.
  - Über die Schnittstelle sollen ferner Multimediadaten an die Fassade übertragen werden können.
  - Die Schnittstelle ist nicht rein unidirektional zu wählen.
- Über eine zentrale Anlaufstelle sind Medienfassaden, dort verfügbare Anwendungen und zu den Anwendungen steuerbare Parameter einsehbar.
- Jeder Nutzer einer Anwendung muss während der Nutzung eindeutig identifizierbar sein.

3.4. Beispielanwendungen

Da generalisierte Schnittstellen nicht sinnvoll anhand eines einzigen Anwendungsbeispiels evaluiert werden können, werden im Folgenden zumindest auf theoretischer Ebene mehrere Beispielanwendungen vorgestellt, die ein möglichst breites Spektrum der Anforderungen an eine generalisierte Schnittstelle abdecken sollen.

Medienfassaden stellen eine mediale Projektionsfläche dar und können visuelle Inhalte transportieren. Solche Inhalte können vorgerendert (Videos oder Bilder) oder derart interaktiv sein , dass man bestimmte Parameter der Visualisierung ändern kann, die grundsätzlich visualisierten Inhalte aber unverändert bleiben (Spiele und ähnliche interaktive Anwendungen gehören hierzu). Wenn der Inhalt der Projektion selbst geändert werden soll, wird eine aufwändigere Kommuniktion notwendig: es müssen Inhalte zur Fassade übertragen werden, die dann dargestellt werden können. Zwei Anwendungsarten können also zunächst grundsätzlich unterschieden werden, wobei sie auch gemischt auftreten können:

Parametrisierte Visualisierung bezeichnet eine Form von Anwendungen bei der bestimmte Parameter einer Visualisierung beeinflusst werden. Dazu zählen wie oben erwähnt Positionen, Größen, Texte, Farben und Ähnliches.

Freie Visualisierung Bei dieser Form von Anwendungen können direkt Inhalte (Audio oder Video) übertragen werden, die dann dargestellt werden, oder zur Manipulation der Darstellung verwendet werden.

Ein Kernthema dieser Ausarbeitung ist also die Schaffung einer gemeinsamen Schnittstelle, über die sinnvoll nicht nur Kontrolldaten (für parametrisierte Visualisierung) sondern auch Streamingdaten (für die freie Visualisierung) übertragen werden können.

3.4.1. Pong

Eine recht einfach zu steuernde interaktive Anwendungen, an der dennoch bereits einige Dinge erläutert werden können, ist der Spieleklassiker Pong, bei dem zwei Spieler sich auf einem rechteckigen Spielfeld mit je einem Schläger gegenüberstehen. Ein Ball wird zwischen ihnen hin- und hergespielt und die Spieler müssen ihre Schläger jeweils so positionieren, dass der Ball an ihnen abprallt. Schaffen sie es nicht und der Ball landet im Aus (hinter den jeweiligen Schlägern), bekommt der Gegenspieler einen Punkt.

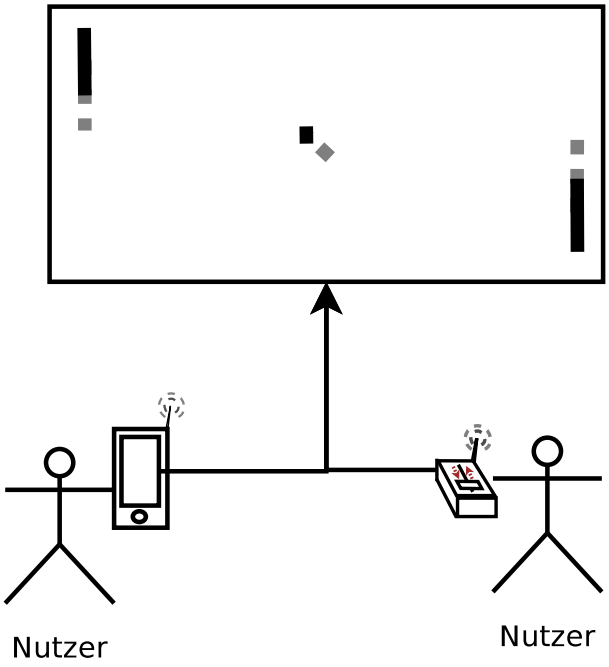


Abbildung 3.7.: Pong mit zwei Spielern

Pong – wie die meisten Spiele – ist ein Programm in dem bestimmte Parameter direkt manipuliert werden können (nach obiger Einteilung also eine parametrisierte Visualisierung). Im Fall von Pong sind diese Parameter die Positionen der Schläger. Zusätzlich könnte hier aber auch die Farbe des Spielfeldes, die Farben der Schläger, die Anzahl der Bälle und Ähnliches als Parameter zur Steuerung zur Verfügung gestellt werden.

In diesem Spiel spielen bereits zwei Spieler. Eine Interaktion eines Spielers bzw. einer Person mit der Medienfassade ist hiermit also abgedeckt, ebenso das Mehrspielerszenario. Es ergeben sich bereits bei diesem einfachen Spiel einige grundsätzliche zu klärende Fragen:

- Wie werden Spieler erkannt?
- Wie lange geht ein Spiel - was passiert, wenn ein Spieler zwischendurch das Spiel verlässt?
- Wie wird verhindert, dass ein Spieler schummelt (z. B. den Schläger des anderen Spielers übernimmt)?
- Welche Möglichkeiten zur Authentifizierung soll es geben?
- Wird es Möglichkeiten zur nachhaltigen Nutzung geben (Highscore)?

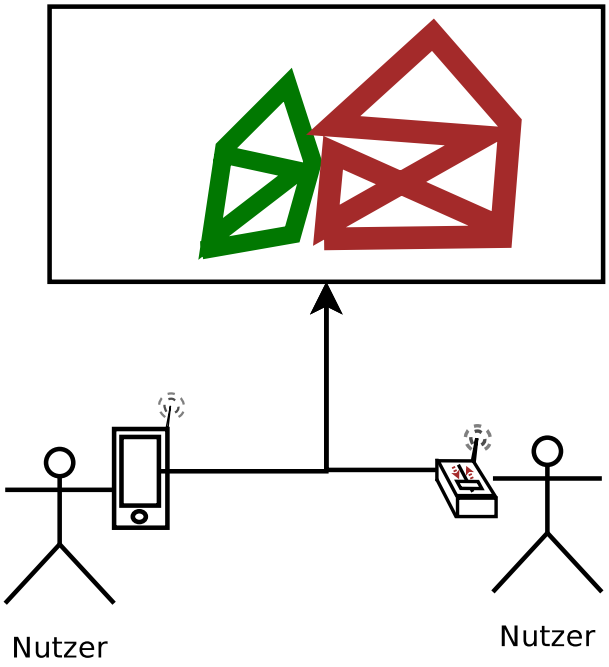


Abbildung 3.8.: gemeinsames Zeichnen auf der Medienfassade

Neben diesen Fragen ist ebenso vor allem die Frage zu klären, auf welche Weise welche Befehle von welchem Endgerät der Applikation signalisieren, dass sich der Schläger eines Spielers bewegt.

Andere abzudeckende Szenarien werden allerdings von Pong noch nicht berührt: Insbesondere die Möglichkeit, multimediale Daten (Video und Sound) an die Fassade zu übertragen, die visualisiert werden sollen bzw. die Visualisierung beeinflussen, ist hier nicht sinnvoll zu demonstrieren.

3.4.2. Kollaboratives Malen

Mit dieser Anwendung, bei der mehrere Personen gleichzeitig zur Fassade verbunden auf dieser virtuell malen können, soll demonstriert werden, wie existierende Anwendungen mit der vorgeschlagenen Schnittstelle umsetzbar sind.

Im Vergleich zu Pong werden hier mehrere Nutzer gleichzeitig kollaborativ an einem gemeinsamen Bild arbeiten. Dabei werden jedoch wie auch bei Pong allein Steuerungsdaten übertragen.

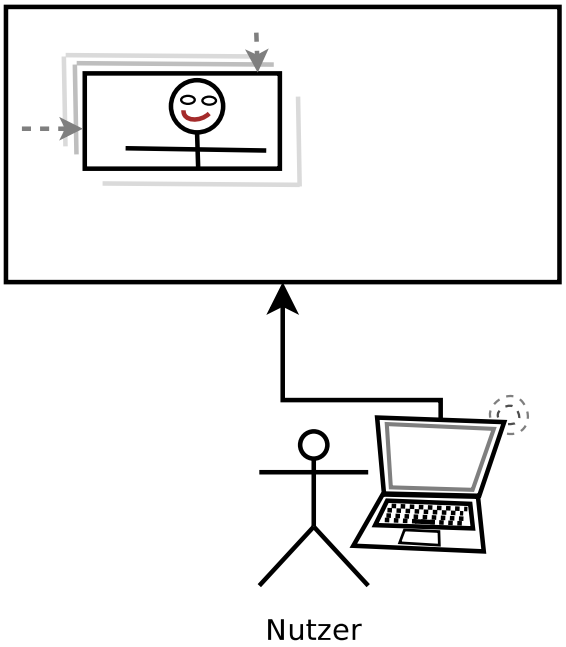


Abbildung 3.9.: Interaktive Medien auf der Medienfassade

Konkret wird diese Anwendung von der Funktionalität her sehr ähnlich der in Boring u. a. [Bor+11] beschriebenen Anwendung für das Ars Electronica Center in Linz: Die Fassade kann von einem mobilen Endgerät aus bemalt werden. Allerdings geschieht dies im Rahmen dieser Arbeit ohne die Übertragung eines live Kamera Videos, sondern lediglich durch die Übertragung der Touchevents.

Mit dieser Anwendung wird eine Art Zwischenschritt eröffnet zwischen parametrisierter und freier Visualisierung: es könnte sowohl das gemalte Bild an die Fassade übertragen werden, als auch Befehle zum Zeichnen eines Bildes.

3.4.3. Mediascreen

Diese Anwendung soll demonstrieren, dass neben Steuerungsdaten auch Multimediadaten über die Schnittstelle an die Fassade übertragen werden können. Dabei soll das Bild einer Kamera (beispielsweise einer Webcam an einem Laptop) an die Fassade übertragen und dort live visualisiert werden. Darüber hinaus soll gleichzeitig ein Audiodatenstrom, sowie ein Steuerungsdatenstrom übertragen werden. Die steuerbaren Parameter werden dabei die Position und Größe der Darstellung des übertragenen Kamerabilds auf der Medienfassade sein.

Nach obiger Kategorisierung handelt es sich zunächst um eine freie Visualisierung, weil hier die multimedialen Inhalte direkt übertragen werden. Da gleichzeitig allerdings auch Steuerungsdaten übertragen werden, ist diese Anwendung als Mischform zu betrachten.

Im Gegensatz zu Pong und ebenso wie beim kollaborativen Malen sind hier keine konkurrierenden Teilnehmer zu erwarten.

4. Systementwurf

In diesem Kapitel wird aufbauend auf den Anforderungen an das System ein Entwurf vorgenommen und diskutiert.

4.1. Schnittstellenentwurf

Hardwareseitig wird aufgrund hoher verfügbarer Bandbreite, weiter Verbreitung und der Tatsache, dass sowohl Steuerungsdaten, als auch Streamingdaten darüber bereits erfolgreich übertragen wurden, die Netzwerkschnittstelle vorgeschlagen und auch im Prototypen für die Implementierung genutzt. Dabei ist es prinzipiell zunächst unerheblich, ob diese Netzwerkschnittstelle drahtlos oder drahtgebunden ist: sowohl W-LAN als auch Ethernet können UDP übertragen. Es bietet sich jedoch aufgrund der niedrigeren Hemmschwelle W-LAN zur Nutzung an, da für Ethernet ein Kabel mitgeführt werden müsste bzw. Smartphones beispielsweise gar keinen Ethernetanschluss besitzen.

Als Interaktionsschnittstelle wird daher eine Kombination aus OSC und RTP vorgeschlagen. Diese Kombination wird vor allem deswegen vorgeschlagen, weil sich RTP besonders gut für die Übertragung von Streamingdaten und OSC für die Übertragung von Steuerungsdaten für Parameter von Applikationen eignet. Da jedoch RTP vor allem für Embeddedanwendungen wie beispielsweise DIY-Hardware auf Basis von Arduino wenn überhaupt dann nicht ohne erheblichen Aufwand zu implementieren ist, wird neben der Lösung OSC über RTP zu verschicken parallel dazu OSC einzeln vorgeschlagen.

Von interaktiven Applikationen wird also mindestens OSC zur Verfügung gestellt, um Parameter zu steuern. Sollen nicht nur Parameter gesteuert werden, sondern auch Livemedien an die Fassade übertragen werden, wird zusätzlich RTP implementiert. OSC einzeln soll dabei nur noch als Fall Back Lösung oder für Embedded Geräte zur Verfügung stehen. Dieselben Parameter, die sich per OSC steuern lassen, sollen sich auch per OSC über RTP steuern lassen.

Aus dieser Kombination aus OSC und OSC über RTP ergeben sich verschiedene Vorteile gegenüber einer separaten Nutzung von OSC und RTP:

- Clientseitige Anwendungen müssen nur eine Schnittstelle bedienen. Entweder sie verfügen über eine RTP Implementierung – dann können sie auch ihre OSC Daten darüber verschicken oder sie verfügen nicht pber eine RTP-Implementierung – dann können sie immerhin die OSC Schnittstelle noch nutzen.
- OSC Daten, die parallel zu Audio- und Videodaten übertragen werden, werden auch in der entsprechenden Reihenfolge bzw. gleichzeitig abgearbeitet.

4.2. Protokollentwurf

Da OSC über RTP noch nicht verschickt wurde, existieren dazu bislang keinerlei Implementierungen und Unterlagen. Daher wird ein eigener Payload Type definiert und neben einer Audio- und/oder Video- auch eine OSC-Session über RTP eröffnet. Dies ermöglicht nun nicht nur die Interaktion mehrere Benutzer mit einer Fassade im Sinne einer unidirektionalen Kommunikation zwischen jeweils einem Benutzer und der Fassade, sondern auch die Kommunikation zwischen den Benutzern und – so sie über Netzwerk verbunden sind – auch zwischen Medienfassaden. Die Kommunikation zwischen Medienfassade und Benutzer ist in beiden Fällen (nur über OSC oder mittels RTP) bidirektional.

Protokollseitig wird für die Schnittstelle nicht viel mehr festgelegt, als es schon mit OSC und RTP wurde. Die Daten im Payload eines OSC-Paketes für RTP entsprechen dem OSC-Standard und können somit von derselben OSC-Komponente verarbeitet werden, die auch die restlichen OSC-Nachrichten verarbeitet, die nicht über RTP empfangen wurden.

Somit wird weder festgelegt, dass Applikationen von Medienfassaden bestimmte Standardparameter zur Verfügung stellen müssen, noch welche Form die OSC-Adressen haben müssen, die für Parameter vergeben werden. Dadurch wird dem Programmierer einer Anwendung jegliche Freiheit zur Umsetzung seiner Anwendung gewährt und dennoch festgelegt, dass OSC für Steuerungsdaten verwendet werden soll. Da wie oben beschrieben viele verschiedene Datentypen und letztlich mit `blob` ein generischer binärer Datentyp zur Verfügung stehen, sollte diese Forderung nach der Verwendung von OSC auch keine Einschränkung darstellen. Damit wird allerdings auch die Website ein wichtiger Bestandteil des Gesamtsystems: wenn alle Anwendungen verschiedene Parameter

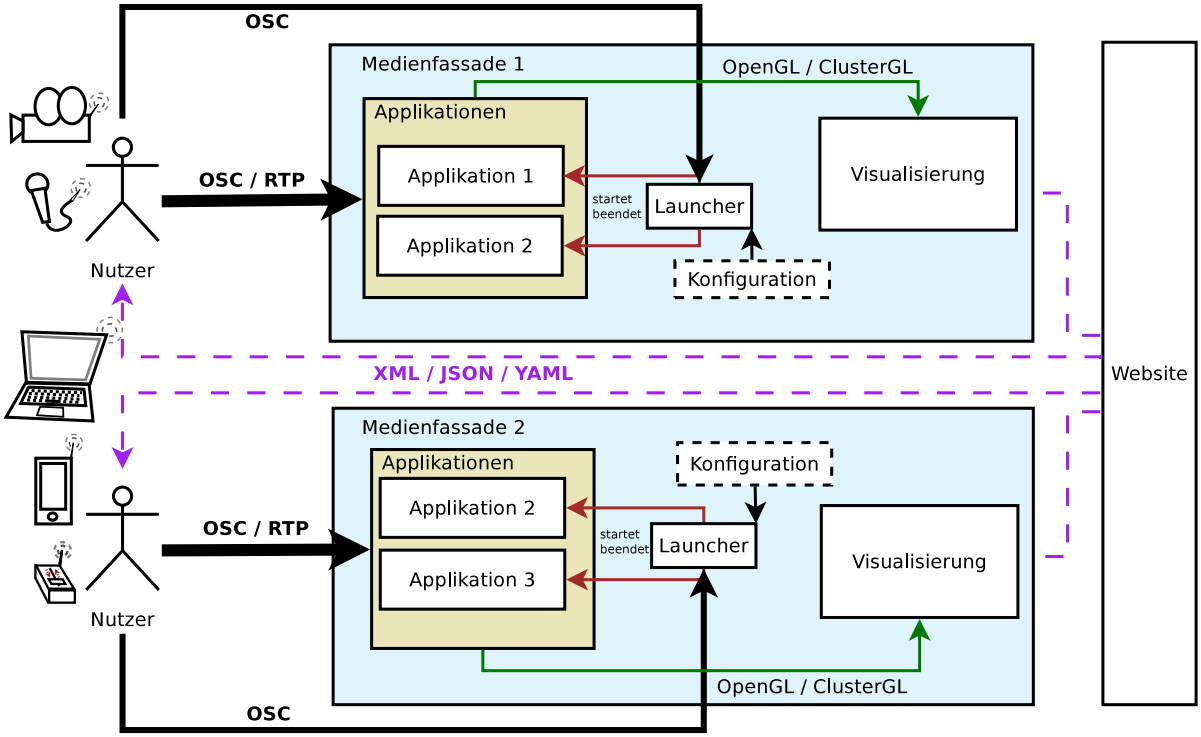


Abbildung 4.1.: Schematische Darstellung des Gesamtsystems mit zwei Medienfassaden

über verschiedene Adressen zur Verfügung stellen, muss das dokumentiert sein, damit Anwender oder Anwendungsentwickler, die darauf aufbauen wollen, die Anwendungen auch nutzen können.

4.3. Gesamtsystem

Wie in der Anforderungsanalyse ausgearbeitet, soll das zu entwickelnde System nicht auf nur einer Medienfassade funktionieren. Da jedoch verschiedene Medienfassaden verschieden aufgebaut sind und verschiedene Applikationen verschiedene Eingabemethoden und -geräte benötigen, bietet sich ein modularer Aufbau an, bei dem bestimmte Kernkomponenten immer gleich sind und entsprechend der generellen Schnittstelle bedient werden. Diese Modularität erstreckt sich dabei nicht nur über die ansteuerbare Hardware, sondern auch auf die verwendeten Softwaremodule. Sollen beispielsweise Videos oder Audiodaten übertragen werden, bietet es sich an, verschiedene Standards zu unterstützen bzw. eine um weitere Standards erweiterbare Infrastruktur zu schaffen. Zusätzlich wurde in Abschnitt 3.2 herausgearbeitet, dass auf einer Medienfassade auch mehrere Anwendungen installiert sein können, die entsprechend vom Nutzer gestartet werden können.

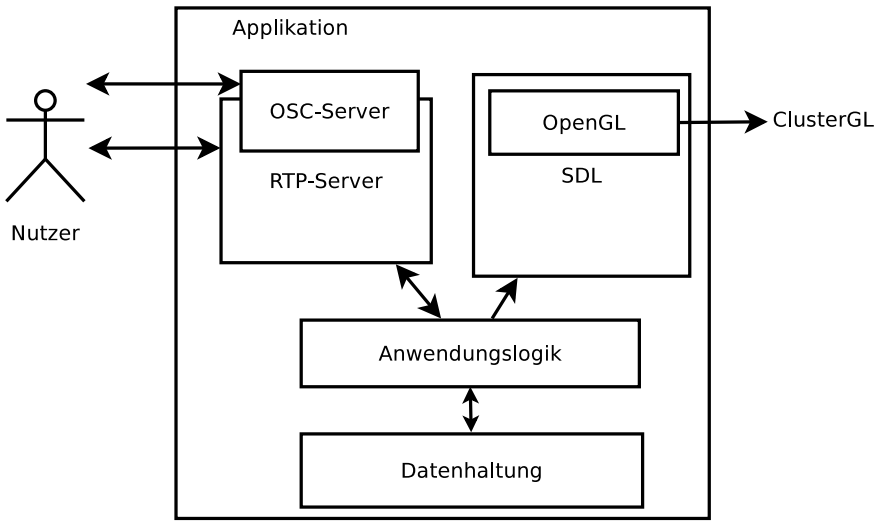


Abbildung 4.2.: Schematische Darstellung der Softwarearchitektur einer Applikation

Abbildung 4.1 stellt das Gesamtsystem, welches im Rahmen dieser Arbeit umgesetzt werden soll, dar. Es besteht im Wesentlichen aus einer Website und je Medienfassade aus Applikationen und einem konfigurierbaren Applikationslauncher. Die generalisierte Schnittstelle zur Steuerung der Anwendungen ist in Form von je einem Pfeil von einem Akteur „Nutzer“ zu den Applikationen, als auch vom Nutzer zum Launcher dargestellt. Der Nutzer muss dabei nicht unbedingt in beiden Fällen eine tatsächliche Person sein und die Aktionen müssen auch nicht direkt von einer Person initiiert werden. Es könnte auch ein Programm sein, dass beispielsweise durch Kameratracking Interaktionsmuster in entsprechende Befehle umsetzt und diese per OSC bzw. RTP an die Anwendung weiterleitet. Auch könnte ein Programm beispielsweise einem zeitlichen Ablauf folgend den Launcher veranlassen Programme zu starten.

Medienfassade 2 ist in der Darstellung nahezu identisch mit Medienfassade 1. Allerdings ist zu bemerken, das Applikation 2 nicht nur auf Medienfassade 1 lauffähig ist, sondern auch auf Medienfassade 2. Diese Applikation könnte nun auch, sofern sie auf beiden Medienfassaden gestartet ist, mit der jeweils anderen Instanz kommunizieren. Dies geschieht wieder über OSC bzw. RTP und stellt eine weitere Generalisierung des Konzepts dar.

4.3.1. Applikationen

Im Mittelpunkt des Gesamtsystems stehen die Applikationen. Der generalisierte Ansatz ist hier im wesentlichen auf zwei Komponenten bezogen: die Visualisierung und die Interaktion.

Die Visualisierungskomponente der Anwendung wird dabei in OpenGL implementiert und mittels SDL in eine Fensterumgebung eingebettet. So kann die Applikation an ClusterGL angebunden werden und entsprechend flexibel mittels verteiltem Rendering auch auf unterschiedlich auflösenden Displays dargestellt werden. Da die Anwendung jedoch nur OpenGL nutzt, kann sie lokal entwickelt und getestet werden und erst anschließend auf die Medienfassade gespielt werden.

Darüber hinaus implementieren Applikationen zur Interaktion eine OSC-Schnittstelle und wenn nötig eine RTP-Schnittstelle. Über die OSC-Schnittstelle können Steuerungsdaten übermittelt und so Parameter manipuliert werden und über die RTP-Schnittstelle können ebenso die OSC-Daten übertragen werden, aber auch Streamingdaten wie Audio und Video. OSC über RTP bietet dabei den Vorteil der zeitlichen Synchronität, die durch RTP gewährleistet wird. So können Video und Audiodaten synchron zu OSC-Daten übertragen werden.

Diese Dopplung geschieht vor allem deswegen, weil es so nicht notwendig ist, bei jedem Gerät, dass nur Parameter der Visualisierung auf der Fassade steuern soll und keine live-Daten übertragen soll, RTP komplett implementieren zu müssen. Das könnte sonst beispielsweise für Microcontroller mit sehr begrenztem Speicher und begrenzter Prozessorgeschwindigkeit ein Ausschlusskriterium sein und somit der Prämisse widersprechen, auch Hobbyelektroniker und Bastler zur Interaktion mit der Fassade einzuladen. OSC sollte auch auf solchen Microcontrollern noch mit vertretbarem Aufwand implementierbar sein. Genau diese Schnittstelle (OSC über RTP und OSC allein parallel dazu) vorzufertigen, sodass Anwendungen letztlich nur noch die Daten entgegennehmen müssen und verarbeiten müssen, ist das eigentliche Forschungsziel dieser Arbeit.

4.3.2. Launcher

Um Applikationen auf einer Medienfassade bei Bedarf zu starten und zu nutzen, benötigt es einer weiteren Komponente – hier Launcher genannt (vgl. Abbildung 4.3). Die Applikationen werden je nach Eingabe beim Launcher von diesem gestartet. Sie laufen

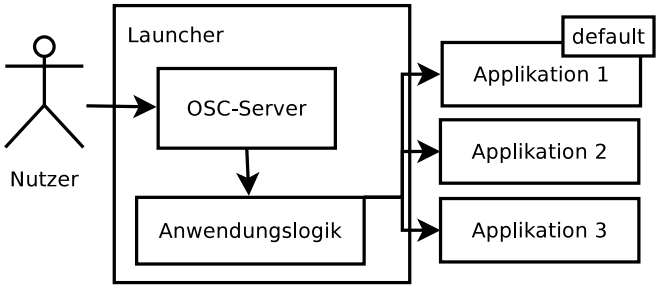


Abbildung 4.3.: Schematische Darstellung der Softwarearchitektur des Launchers

anschließend eine für eine festgelegte Maximalzeit und nach Beenden (entweder durch die Applikation selbst oder nach Ablauf der Zeit) wird wieder in eine Standardapplikation, welche festgelegt werden kann, gewechselt. Da für das mit dem Launcher abgedeckte Starten und Stoppen der Anwendung noch keine kollaborative Interaktion notwendig ist und auch keine Übertragung multimedialer Inhalte, scheint RTP an dieser Stelle nicht sinnvoll. Daher wird hier nur OSC genutzt um dem Launcher mitzuteilen, welche Anwendung nun gestartet werden soll. Der Launcher wird an einer bestimmten OSC-Adresse auf eine übertragene Zeichenkette warten und dieses Programm anschließend starten. Jedem Programm kann dabei eine maximale Laufzeit vorkonfiguriert werden. Darüber hinaus kann eine Anwendung als Standardanwendung der Fassade festgelegt werden, die immer läuft, wenn gerade keine andere Anwendung gestartet wurde. Damit ist gewährleistet, dass die Fassade immer bespielt wird, wenn sie eingeschaltet ist.

4.3.3. Website

Da nun die Applikationen für Medienfassaden in einheitlicher und übertragbarer Form definiert wurden, soll nun eine zentrale Verwaltungskomponente beschrieben werden. Diese kann in Form einer Website umgesetzt werden, welche die verschiedenen Medienfassaden, dort installierte Anwendungen sowie Interaktionsmöglichkeiten auflistet. Je Anwendung werden darüber hinaus die öffentlich zugänglichen Eingabeparameter gelistet, sodass Nutzer einsehen können, mit welcher Anwendung unter Verwendung welcher Adressen sie welche Effekte erzielen können. Wünschenswert wäre darüber hinaus die Möglichkeit für eventuell entwickelte mobile Anwendungen eine Art Preset-System zu entwerfen, dass je nach Anwendung eine vorkonfigurierte Oberflächer zur Bedienung mit einem mobilen Endgerät zur Verfügung stellt. Diese Presets könnten automatisch von der Website geladen werden, sobald sich ein Nutzer einer Fassade nähert.

Selbst wenn eine Medienfassade ausschließlich mit nicht-interaktiven Applikationen ausgestattet ist oder die Parameter der Applikationen nicht für mobile Endgeräte öffnet und daher eigentlich keine Dokumentation der steuerbaren Parameter der Anwendung auf der Website bräuchte, ist es dennoch im Sinne eines kreativen Austauschs, eine Art Medienfassadenarchiv zu gründen.

4.3.4. Eingabegeräte

Durch die Entkopplung von Eingabegeräten und Applikation sind Eingabegeräte für Applikationen austauschbar - ein live-Bild beispielsweise kann von einer stationären Kamera, einer mobilen Kamera oder (so ein Internetanschluss vorhanden ist) auch von einer Webcam kommen. Dies wird durch zusätzliche der Medienfassade eigenen Komponenten abstrahiert. In der Applikation werden letztlich an einer Stelle auf Bilder einer Quelle entgegen genommen. Durch diese Abstraktion wird auch erreicht, dass Betreiber von Medienfassaden selbst entscheiden können, ob sie eine Interaktion mit mobilen Endgeräten zulassen wollen, oder nicht. Dafür wäre ein W-LAN-Netz oder ähnliches notwendig, über das Daten an die Fassade übertragen werden können. Selbes gilt für Audio- und Steuerungsdaten.

4.4. Interaktionsworkflow

Durch die eben eingeführte Infrastruktur wird eine bestimmte Art der Bedienung nahegelegt, die im Folgenden kurz erläutert werden soll und in Abbildung 4.4 dargestellt wird: Zunächst wird durch den Launcher die Konfiguration der Applikationen der Fassade geladen. Hier ist eine default Anwendung definiert, welche immer dann ausgeführt wird, wenn gerade keine andere Anwendung läuft.

Der Nutzer einer Fassade bekommt die Möglichkeit, sich gegebenenfalls auf der Website der Medienfassade(n) zu erkundigen, wie die Applikation bedient werden kann – welche Parameter sie zur Steuerung zur Verfügung stellt. Er kann darüber hinaus durch eine Interaktion mit dem Launcher eine Anwendung auf der Fassade starten. Diese Interaktion kann entweder direkt zwischen Nutzer und Launcher stattfinden (über OSC), oder über eine Mittelsapplikation. So eine Mittelsapplikation kann beispielsweise eine solche sein, die von einem Near Field Communication (NFC) Kartenleser Karten einliest und

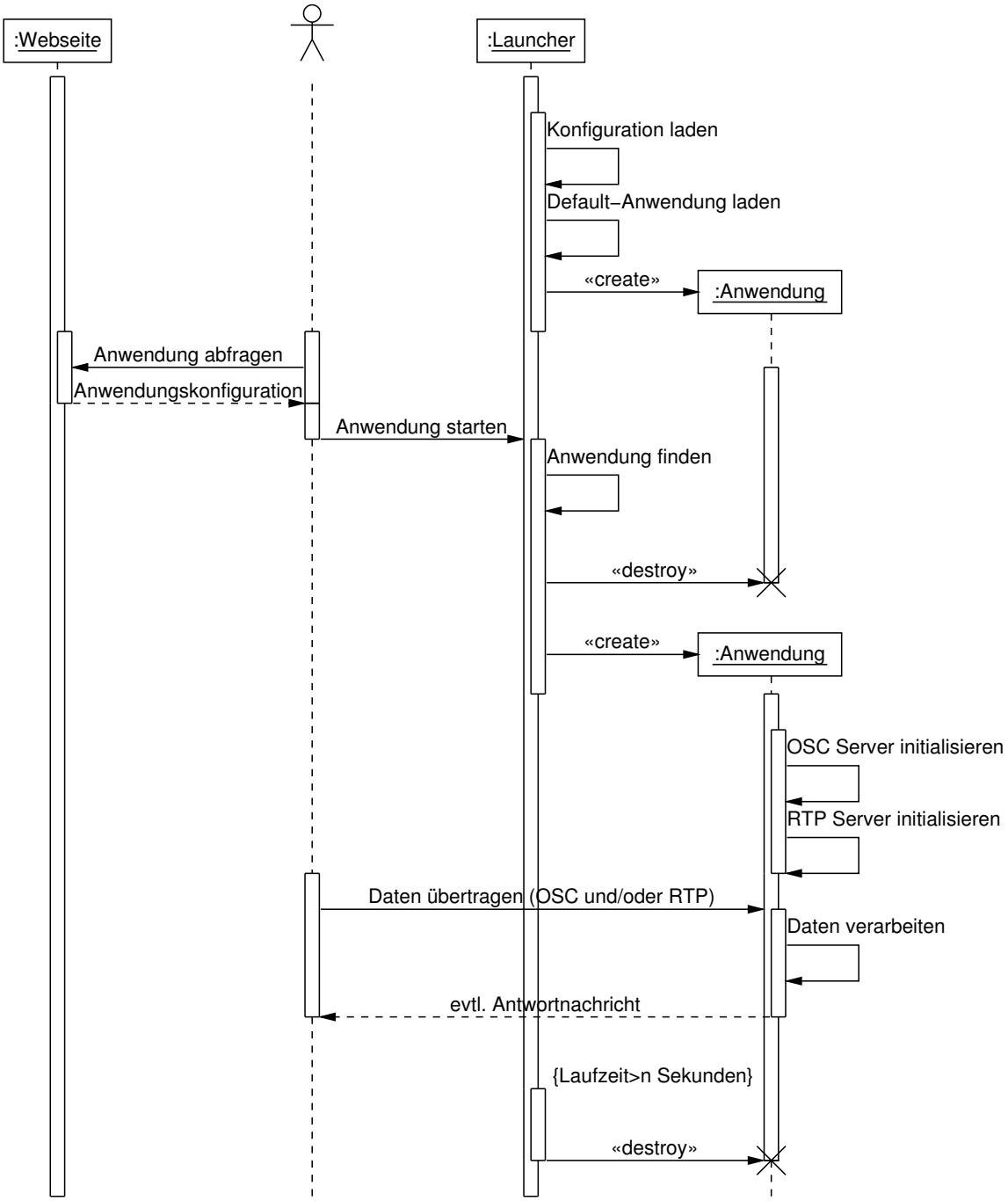


Abbildung 4.4.: Sequenzdiagramm der Bedienung des Gesamtsystems

anhand dessen den Launcher über OSC anweist, die entsprechende Applikation zu starten. Auch ein Touchterminal, welches Nutzereingaben in OSC-Befehle für den Launcher umwandelt ist hier denkbar. Mit dieser vom Launcher gestarteten Anwendung kann der Nutzer anschließend gegebenenfalls interagieren. Sie wird entweder vom Benutzer, durch sich selbst, oder vom Launcher nach einer bestimmten Maximalzeit gestoppt und es wird wieder in die Defaultanwendung gewechselt. Dies garantiert, dass nicht eine Anwendung ständig läuft und nur ein Benutzer die Fassade nutzen kann.

Die Unterscheidung von Benutzern, das Sessionmanagement und gegebenenfalls persistente Speicherung von Werten wird von der Anwendung selbst geregelt. Auf diese Weise wird eine weitestgehend von der Medienfassade unabhängige Entwicklung der Anwendung möglich, sowie eine flexible – dem Entwickler nach eigenen Vorlieben überlassene – Art der Anwendungsentwicklung. Lediglich die Interaktionsschnittstelle ist vorgegeben.

5. Implementierung

In diesem Kapitel wird es nach dem Systementwurf um die Beschreibung wichtiger Details und Designentscheidungen bei der Implementierung gehen. Sämtliche Erweiterungen und Änderungen, die dabei teilweise in umfangreicher Form an bestehenden Bibliotheken im Rahmen dieser Arbeit vorgenommen werden mussten, wurden in öffentlichen Git Repositories zur Verfügung gestellt. Im Fall von ClusterGL wurden diese Änderungen bereits vom Entwickler des Projektes in dessen Hauptzweig übernommen.

5.1. Entwicklungs- und Einsatzumgebung

Für die Entwicklung wurden die Programmiersprachen C und C++ gewählt. Diese Wahl basiert neben persönlichen Vorlieben vor allem auf der Tatsache, dass die verwendeten Bibliotheken in C bzw. C++ zur Verfügung stehen und auch in RFCs üblicherweise C als Sprache für Beispiele genutzt wird.

ClusterGL, welches als Visualisierungskomponente der Programme genutzt werden soll, ist in C++ implementiert, lässt sich aber auch aus C heraus nutzen. Darüber hinaus ist auch die SDL eine C- und C++-Bibliothek. Daher wäre für Applikationen, die liblo und ClusterGL und SDL nutzen, eine C-Anwendung bereits die beste Wahl.

In Abschnitt 2.3.4 wurde erläutert, dass liblo nach momentanen Recherchen und Entwicklungsstand eine der am besten geeigneten Bibliotheken für OSC-Anwendungen in C ist.

Da die verwendeten Bibliotheken ursprünglich für Linux entwickelt wurden, wurde als System- und Entwicklungsumgebung ebenfalls Linux gewählt. Sowohl EMIPLIB, als auch liblo sollen auf Mac OS X und Windows laufen. ClusterGL lief allerdings bis vor kurzem ausschließlich auf Linux. Inzwischen soll sie auch unter OS X ab 10.7 lauffähig sein. Alle unten beschriebenen Programme sind unter dem aktuellen Ubuntu 12.04.1

nach der Installation von sowohl liblo als auch EMIPLIB und den entsprechend notwendigen Systembibliotheken aus den Standard Repositories kompilier- und ausführbar. Entwicklungs und Einsatzumgebung gleichen sich hier in alle Softwarekomponenten: Das Testsystem für die Medienfassade am FKI ist ebenfalls unter Ubuntu 12.04 aufgesetzt und sämtliche Bibliotheken sind ebenso installiert, wie auf dem Entwicklungssystem. Alle verwendeten Bibliotheken sind dabei unter freien Lizenzen verfügbar, was eine Anpassung und Weiterentwicklung an den Stellen, wo es notwendig war, überhaupt erst ermöglichte.

Für die Implementierung der Website wurde nicht auf C bzw. C++ zurückgegriffen, sondern auf das weit verbreitete Python Webframework Django. Auch dieses ist quelloffen, musste jedoch im Rahmen dieser Arbeit nicht weiterentwickelt, sondern lediglich verwendet werden. Als Datenbank für die Website wurde MySQL genutzt.

5.2. Schnittstellenentwicklung – OSC über RTP

Ein wichtiger Kernpunkt bei der Entwicklung war neben der Umsetzung der einzelnen Komponenten des Gesamtsystems vor allem die Erweiterung von RTP um einen weiteren Payload Type für OSC. Da die angesprochenene EMIPLIB mit ihrem Chainingkonzept bereits eine Infrastruktur zur Verfügung stellt, stand die Überlegung nah, hier weitere Komponenten – ähnlich denen, die für Audio- und Videoübertragung bereits existieren – für diese Bibliothek zu implementieren.

Diese Infrastruktur der EMIPLIB ist so aufgebaut, dass sämtliche Elemente einer Chain von `MIPComponent` abgeleitet werden. In `MIPComponent` sind die Funktionen `push()` und `pull()` vorgegeben, welche von den abgeleiteten Klassen implementiert werden müssen. Die Funktion `push()` wird dabei beim Durchlaufen der Chain aufgerufen, wenn Daten vom vorherigen Element der Chain Daten per `pull()` von diesem Element emfpangen wurden. Es gibt dabei drei Arten von Elementen einer Chain

Eingabekomponenten bekommen am Anfang der Chain je Durchlauf immer eine Art Triggerbefehl von einem Timer. Diesen Triggerbefehl bekommen Eingabekomponenten über ihre `push()`-Funktion. Nun können sie entweder bereits in der `push()`-Funktion Daten generieren, die anschließend per `pull()` abgefragt werden, oder sie werden erst in der `pull()`-Funktion generiert.

Durchgangskomponenten wandeln Eingabedaten, die über `push()` an sie geliefert werden in Ausgabedaten um, die per `pull()` von der Chain anschließend wieder abgefragt werden.

Ausgabekomponenten werden nur per `push()` mit Daten versorgt – es werden keine Daten per `pull()` von ihnen erfragt.

Dem Konzept der EMIPLIB folgend wurden also neben einem neuen Datentyp (`OSCMessage`) folgende Komponenten entwickelt:

`OSCInput` nimmt OSC-Nachrichten entgegen und speichert sie zwischen.

`OSCEncoder` nimmt vom OSCInput OSC-Nachrichten und wandelt sie in `OSCMessages` um.

`OSCRTPEncoder` wandelt `OSCMessages` in `RTPMessages` um, welche von der `RTPComponent` der EMIPLIB verarbeitet werden können.

`OSCRTPDecoder` wandelt `RTPMessages` wieder zurück in `OSCMessages`.

`OSCDecoder` wandelt `OSCMessages` wieder zurück in OSC-Nachrichten.

`OSCOutput` speichert empfangene OSC-Nachrichten zwischen.

All diese Komponenten in einer Chain hintereinander mit einer `RTPComponent` in der Mitte (zwischen `OSCRTPEncoder` und `OSCRTPDecoder`) ergeben die Architektur zum Senden und Empfangen von OSC Nachrichten über RTP. Das System ist flexibel entworfen und erlaubt es weitere Komponenten hinzuzufügen, um beispielsweise die Nachrichten vor dem Versenden zu komprimieren oder durch ein Journaling System (ähnlich bei MIDI über RTP) Paketverluste zu vermeiden.

Die Entwicklungen an der EMIPLIB Bibliothek sind unter folgender URL öffentlich verfügbar: `https://github.com/oberling/emiplib`.

Da es für OSC Nachrichten wie in Abschnitt 2.3.4 beschrieben bereits einge Bibliotheken gibt, die etabliert sind und den Standard umsetzen, wurde davon Abstand genommen, selbst eine Bibliothek von Grund auf neu zu implementieren und stattdessen auf die oben angesprochene liblo zurückgegriffen. Da die liblo jedoch einen eigenen Server mitbringt, der ab dem Erstellen der Nachricht bis zum Auswerten der empfangenen Nachricht bereits alles intern umsetzt, waren hier einige Anpassungen notwendig. Denn letztlich sollte genau dieser Server zwar einerseits noch nutzbar sein (um auch reine OSC Nachrichten

von beispielsweise DIY-Hardware Controllern weiterhin verarbeiten zu können), andererseits sollte es auch möglich sein, Nachrichten extern über einen anderen Weg (über oben beschriebenen EMIPLIB Chain) zu verschicken. Letztlich wurden einige Methoden für die liblo neu entwickelt und bestehende öffentlich verfügbar (von außen erreichbar) gemacht. Es fiel dabei auf, dass in der liblo Bibliothek als Konzept zur Datenkapselung intensiv Typendefinitionen auf `void*`, dem allgemeinsten Datentyp in C, genutzt werden. Deshalb können aus den hinterlegten Daten extern keinerlei Informationen gewonnen werden. Lediglich über die von der Bibliothek zugänglich gemachten Methoden können Daten extrahiert und manipuliert werden. Die Methoden arbeiten dabei mit internen Datentypen, die letztlich die wirkliche Beschaffenheit hinterlegter Datenstrukturen offenbart.

Um nicht zwei komplett autarke Systeme (einmal OSC und einmal OSC über RTP) zu haben, was doppelten Implementierungs- und Pflegeaufwand für neue und bestehende Anwendungen bedeuten würde, wurde die liblo so erweitert, dass sie auch das Verarbeiten der Nachrichten externer Quellen übernimmt.

Die Entwicklungen an der liblo Bibliothek sind unter folgender URL öffentlich verfügbar: `https://gitorious.org/~oberling/liblo/oberlings-liblo`.

5.3. Einzelkomponenten

Im folgenden werden die einzelnen Komponenten, die im Rahmen dieser Arbeit umgesetzt wurden, näher beschrieben. Dabei werden sowohl Auszüge des Quelltextes, als auch textuelle Beschreibungen der Funktionalität gegeben.

5.3.1. Launcher

Der Launcher wurde als C++-Applikation unter Nutzung der liblo C-Bibliothek für den OSC-Server und die libconfuse für die Konfiguration implementiert.

Die Konfiguration der Anwendungen wird über eine Konfigurationsdatei wie beispielsweise Quelltext 5.1 vorgenommen. Sie folgt dem Schema einer Schlüssel-Wert-Definition und erlaubt die Gruppierung von Schlüsseln und Werten in Gruppen. Eine solche Gruppe ist hier `application`.

```
# Standardlaufzeit der Applikationen
defaultruntime=100

# Beispiel der Pong-Applikation
application {
  # der Name der auszuführenden Datei
  name = "pong"
  # der Pfad (relativ oder absolut)
  path = "./"
  # Argumente für den Aufruf
  arguments = ""
  # festgelegte Laufzeit diese Programms
  runtime = 300
}

application {
  name = "ping"
  arguments = "htw-berlin.de"
  runtime = 3000
  # Dies ist die Standardanwendung
  isdefault = true
}
```

Quelltext 5.1: Konfigurationsdatei des Launchers

Für jede Anwendung können folgende Einstellungen vorgenommen werden:

`name` Der Name der ausführbaren Datei

`path` Der Pfad zur ausführbaren Datei (entweder absolut oder auch relativ)

`arguments` Die Argumente, die dem Programm bei Aufruf übergeben werden sollen (per Leerzeichen getrennt)

`runtime` Die Anzahl an Sekunden, die diese Applikation – einmal gestartet – maximal laufen soll

`isdefault` Ist standardmäßig `false` und gibt an, ob diese Anwendung die Standardanwendung der Fassade sein soll

Bis auf den Namen der Anwendung sind alle weiteren Einstellungen optional. Global (außerhalb der `application`-Gruppen) ist die `defaultruntime` festgelegt, welche für Anwendungen, bei denen keine `runtime` definiert ist, angibt, wie lange sie laufen sollen.

Um Anwendungen zu starten stellt der Launcher einen OSC-Server zur Verfügung. Die verwendete Bibliothek liblo ist zwar in C geschrieben und lässt sich nicht allzu einfach nach C++ portieren (der Versuch wurde vom Autor unternommen), ist mit kleineren Einschränkungen aber letztlich doch für den Anwendungszweck sehr gut zu gebrauchen. Die Herausforderung besteht hier dabei, dass dem Pattern Matching in der liblo ein Callback Mechanismus angeschlossen ist, welcher gleich eine hinterlegte Methode anhand der Zieladresse der OSC-Nachricht ausführt.

Die Signatur dieser hinterlegten Methode muss allerdings eine C-konforme Methodensignatur sein. In den Klassen von C++ wird allerdings unter anderem für das Überladen von Methoden eine andere Form von Signatur notwendig. Weiterhin ist C++ laut Stroustrup [Str00, S. 8] auf einem Subset von C aufgebaut. Daher ist es nicht möglich, sämtliche Funktionalität von C-Programmen in derselben Form auch in C++ weiter zu verwenden. Mit statischen Klassenmethoden ist es dennoch möglich, Methoden mit C-konformen Signaturen zu erzeugen. Dadurch wird allerdings der objektorientierte Ansatz von C++ unterlaufen. Diese Einschränkung kann aber in Anbetracht der Tatsache, dass eine Anwendung nicht mehr als eine OSC-Umgebung zur Verfügung stellen können muss, in Kauf genommen werden. Eine OSC-Umgebung bezieht sich dabei auf eine Komponente, die empfangene OSC-Nachrichten auswertet und entsprechende Funktionen aufruft.

```
1 class OSCServer {
2 public:
3    OSCServer() {}
4    static int run(void* data);
5    string getApplicationName() { return string(_appname); }
6 private:
7    static void error(int num, const char* m, const char* ↩
       path) {}
8    static int start_application_handler(const char* path, ↩
        const char* types, lo_arg** argv, int argc, void* ↩
       data, void* user_data);
9    static char _appname[1024];
```

```
10      static bool _newSignal;
11      static lo_server_thread _server;
12  };
13
14  char OSCServer::_appname[1024] = "";
15  bool OSCServer::_newSignal = false;
16  lo_server_thread OSCServer::_server = lo_server_thread_new("7770", error);
17
18  int OSCServer::start_application_handler(const char* path, const char* types, lo_arg** argv, int argc, void* data, void* user_data) {
19      // kopiere die gesendete Zeichenkette in _appname
20      strcpy(_appname,(char*) argv[0]);
21      // setze Flag, dass eine neue Anwendung gestartet werden kann
22      _newSignal = true;
23      return 0;
24  }
25  void OSCServer::run(void* data) {
26      // registriere start_app mit einer Zeichenkette als argument mit Methode start_application_handler
27      lo_server_thread_add_method(_server, "/start_app", "s", start_application_handler, NULL);
28      lo_server_thread_start(_server);
29  }
```

Quelltext 5.2: OSC Server des Launchers

Die wesentlichen Bestandteile des OSC-Servers im Launcher sind in Quelltext 5.2 aufgeführt. Sämtliche Methoden und Variablen, die mit der liblo zusammenspielen (`run()`, `error()`, `start_application_handler()`, `_appname`, `_newsignal` und `_server`) sind statisch (`static`) deklariert – stehen also im kompilierten Programm nicht pro erzeugtem Objekt sondern für sämtliche Objekte der Klasse nur einmal zur Verfügung. Das Erzeugen mehrerer Server würde also fehlschlagen, da global nur einmal die Variable `_server` existiert.

```
1  // im Vaterprozess
2  int waittime = 0;
3  int childExitStatus;
4  do {
5     _waitpid = waitpid(pID, &childExitStatus, WNOHANG);
6     if(_waitpid == 0) { // Anwendung läuft
7        // Wenn timeout noch nicht erreicht
8        if(waittime<app->timeout()) {
9           // Anwendung laufen lassen
10          sleep(1);
11          waittime++;
12       } else {
13          // ansonsten anwendung beenden
14          kill(pID, SIGKILL);
15          sleep(1);
16       }
17       // neue Anwendung starten, wenn Befehl dazu erkannt
              wurde
18       if(_osc.newSignal()) {
19          _nextApp = _osc.getApplicationName();
20          _newNextApp = true;
21          kill(pID,SIGKILL);
22          sleep(1);
23       }
24    }
25 } while (_waitpid == 0 && waittime <= app->timeout());
26 // wenn keine neue Anwendung spezifiziert wurde,
     Defaultanwendung starten
27 if(!_newNextApp) {
28    _nextApp = _applications->at(0)->name();
29 }
30 _newNextApp = false;
```

Quelltext 5.3: Sleep-Loop des Launchers

Der OSC-Server im Launcher läuft in einem eigenen Thread. Der Hauptthread des Launchers übernimmt dabei das Starten und Stoppen der Anwendungen. Jede Anwendung wird als `fork` mit anschließendem `exec` vom Launcher gestartet. Anschließend durchläuft der Launcher für die in der Konfigurationsdatei eingestellte Maximallaufzeit der Anwendung einen sleep-loop, in dem nach je einsekündigem Schlafen ein Sekundenzähler inkrementiert wird und geprüft wird, ob die Maximallaufzeit erreicht ist oder das Programm beendet wurde. Ist die Maximallaufzeit des Programms erreicht und es läuft noch immer, wird es per `kill()` beendet. Nach Beenden der Anwendung initiiert der Launcher wieder die Standardanwendung und wartet auf weitere Befehle über OSC.

5.3.2. Website

Die Website ist unter Verwendung des Django Frameworks[1] für Python implementiert worden. Da die Website nicht im Fokus dieser Ausarbeitung steht, wurde nicht sämtliche Funktionalität implementiert – lediglich die Kernkomponenten sind umgesetzt worden. Dazu zählen das Erstellen von Medienfassaden durch registrierte Benutzer, Auflisten von Medienfassaden und deren Anwendungen. Darüber hinaus können pro Anwendung steuerbare Parameter angelegt und anschließend eingesehen werden. Die steuerbaren Parameter können darüber hinaus pro Anwendung als XML, JSON oder YAML Datei exportiert werden. Exmplarisch folgt die Ausgabe der Parameterdatei für die Applikation Pong in JSON.

```
1  [{
2     "pk": 1,
3     "model": "mediafacades.parameter",
4     "fields":{
5        "datatype": "i",
6        "application": 1,
7        "name": "Spieler␣1␣Position",
8        "address": "/player1/position"
9     }
10 },{
11    "pk": 2,
12    "model": "mediafacades.parameter",
```

[1] `https://www.djangoproject.com/`

```
13     "fields": {
14         "datatype": "i",
15         "application": 1,
16         "name": "Spieler␣2␣Position",
17         "address": "/player2/position"
18     }
19 }]
```

Quelltext 5.4: Parameterexport von Pong im JSON-Format

5.4. Beispielanwendungen

In diesem Abschnitt wird näher auf die Implementierung der Beispielanwendungen und deren Besonderheiten eingegangen.

5.4.1. Pong

Die Implementierung von Pong erfolgte zunächst in der Programmiersprache C, um einen Ausgangspunkt für weitere Entwicklungen zu stellen. Als erste Form der Interaktion wurde die Steuerung der Schläger mittels Tastatur implementiert, anschließend per Nachrichten von der seriellen Schnittstelle. Dazu wurde eine sehr simple Form der Übermittlung von Schlägerpositionen gewählt: auf serieller Schnittstelle werden zeilenweise die Werte des analogen Potentiometers von diesem ausgegeben und von der Anwendung eingelesen und entsprechend an die Spiellogik weitergereicht.

Als nächste Form wurde ein OSC-Server unter Verwendung der liblo implementiert, der ähnlich wie der OSC-Server im Launcher arbeitet. Dabei nimmt dieser OSC-Server allerdings Fließkommazahlen für die Schlägerpositionen entgegen. Sämtliche weitere Spiellogik bleibt bestehen – lediglich die Eingabekomponente wird ausgetauscht.

Als finale Version wurde Pong objektorientiert in C++ neu geschrieben, um auch die RTP-Komponente anbinden zu können. Da bereits vorher alles in einzelnen Modulen gekapselt war, bedeutete diese Neuimplementierung vor allem ein Umbenennen von Variablen und Funktionen.

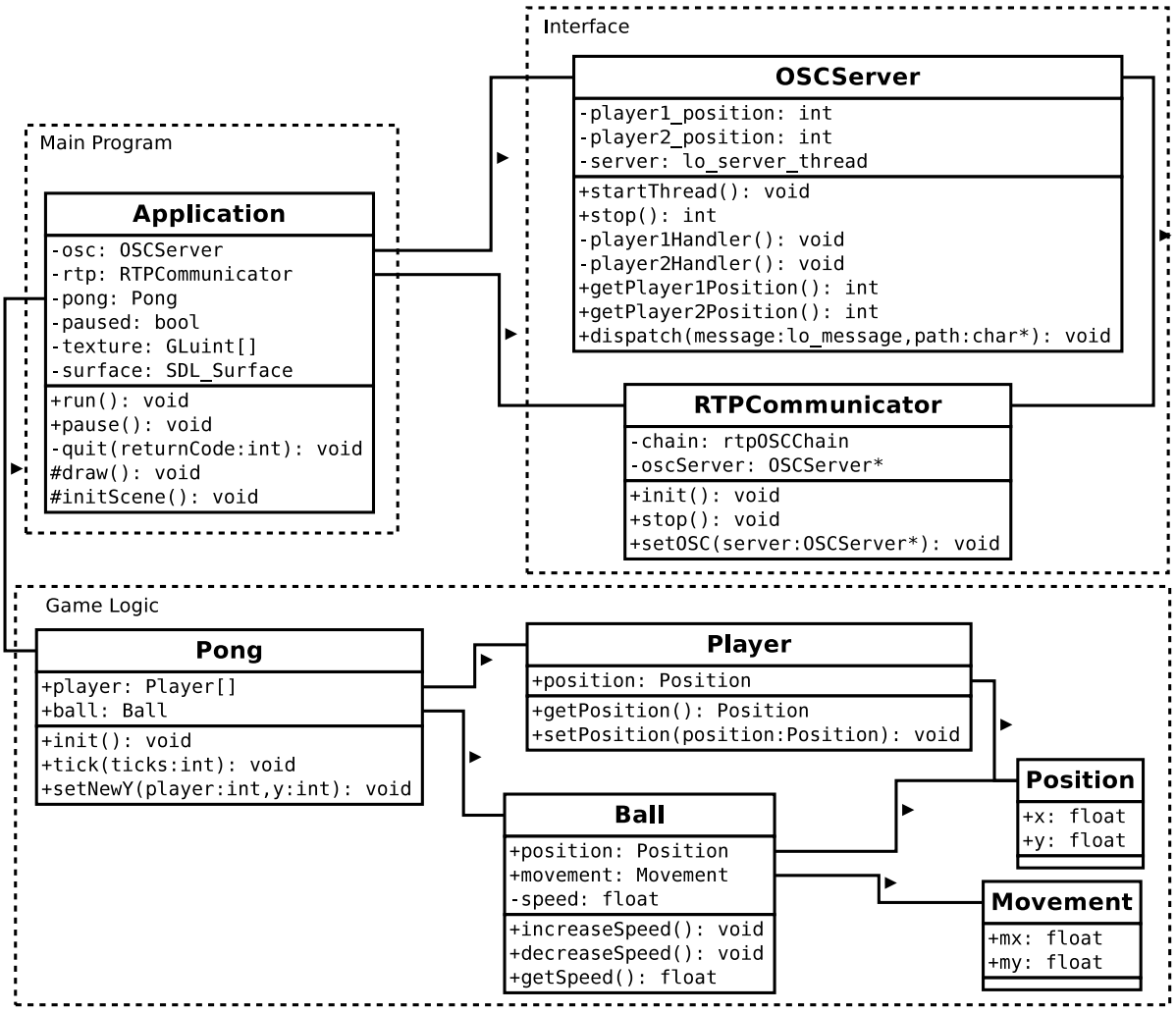


Abbildung 5.1.: Klassendiagramm von Pong für die Medienfassade

Die Grundstruktur des Programms und damit des Quelltexts eines jeden Medienfassadenprogramms umfasst immer eine Application-Klasse, die OSC-Server-Klasse, sowie bei Bedarf eine RTPCommunicator-Klasse. Darüber hinaus gehen die individuellen durch die Anwendungslogik bestimmten Klassen. All diese Komponenten finden sich in Abbildung 5.1 wieder.

Die Application-Klasse ist dabei zuständig für die Behandlung der Nachrichten der Schnittstelle, sowie die Darstellung des Programms in einem Fenster. Hier läuft auch die Hauptschleife des Prgramms in der `run()`-Methode. Sie ist dafür zuständig, dass Daten von der Schnittstelle an die Anwendungslogik weiter gereicht werden und die Visualisierung mit den neu berechneten Werten aus der Anwendungslogik durch die `draw()`-Funktion neu gerendert werden.

Die Schnittstelle abstrahiert sämtliche Interaktion mit dem Programm. Dennoch muss die Schnittstelle pro Programm um die entsprechenden Parameter erweitert werden, die gesteuert werden sollen – daher hat sie von Programm zu Programm unterschiedlich viele Methoden und Attribute.

Die Programmlogik findet in den individuellen Klassen für jede Anwendung statt. Im Fall von Pong ist das wiederum in einer Methode, die immer wieder aufgerufen wird: die `tick()`-Methode, die für Pong intern alle Werte (Spieler- und Ballpositionen) neu berechnet und die Kollisionsbehandlung für den Ball durchführt.

Die Erkennung von Spielern und damit eine Art Session Management passiert, indem jeweils die SSRC desjenigen angenommen wird, der zuerst einen bestimmten Spieler steuern will. Anschließend werden Nachrichten für diesen Spieler auch nur noch von dieser Quelle angenommen. Die SSRC des anderen Spielers muss dabei verschieden von der des ersten Spielers sein, was durch die Verwendung von RTP bereits gegeben ist. Somit wird garantiert, dass zumindest auf Schnittstellenseite kein Spieler den Schläger des anderen Spielers übernehmen kann. Er kann zwar Nachrichten an diese Adresse senden (denn sie ist prinzipiell offen für jeden Spieler), diese werden jedoch ignoriert. Auch der Versuch die Adresse durch intensives wiederholtes Senden von Nachrichten verlangsamt letztlich nur das gesamte Netzwerk und beeinträchtigt somit alle beteiligten Spieler gleichermaßen.

5.4.2. Kollaboratives Malen

Diese Anwendung wurde von Grund auf in C++ implementiert und viele Komponenten aus der Pong-Anwendung konnten übernommen werden bzw. bedurften nur leichter Anpassungen. Sämtliche Initialisierung von Bibliotheken blieb gleich, lediglich die Anwendungslogik und die entsprechende `draw()`-Funktion mussten neu implementiert werden. Auch die Schnittstelle über OSC war dank der generalisierten Schnittstelle, über die eben beliebige Daten gesendet werden können, komfortabel programmierbar. Jeder Maler bekommt hier dynamisch eine Ebene zugewiesen, auf der er anschließend malen kann. Im Gegensatz zu Pong ist an dieser Anwendung besonders, dass die OSC-Adresse für alle Maler dieselbe ist und zwei Fließkommazahlen (Position x und y des Punktes, der gezeichnet werden soll) als Argumente hat.

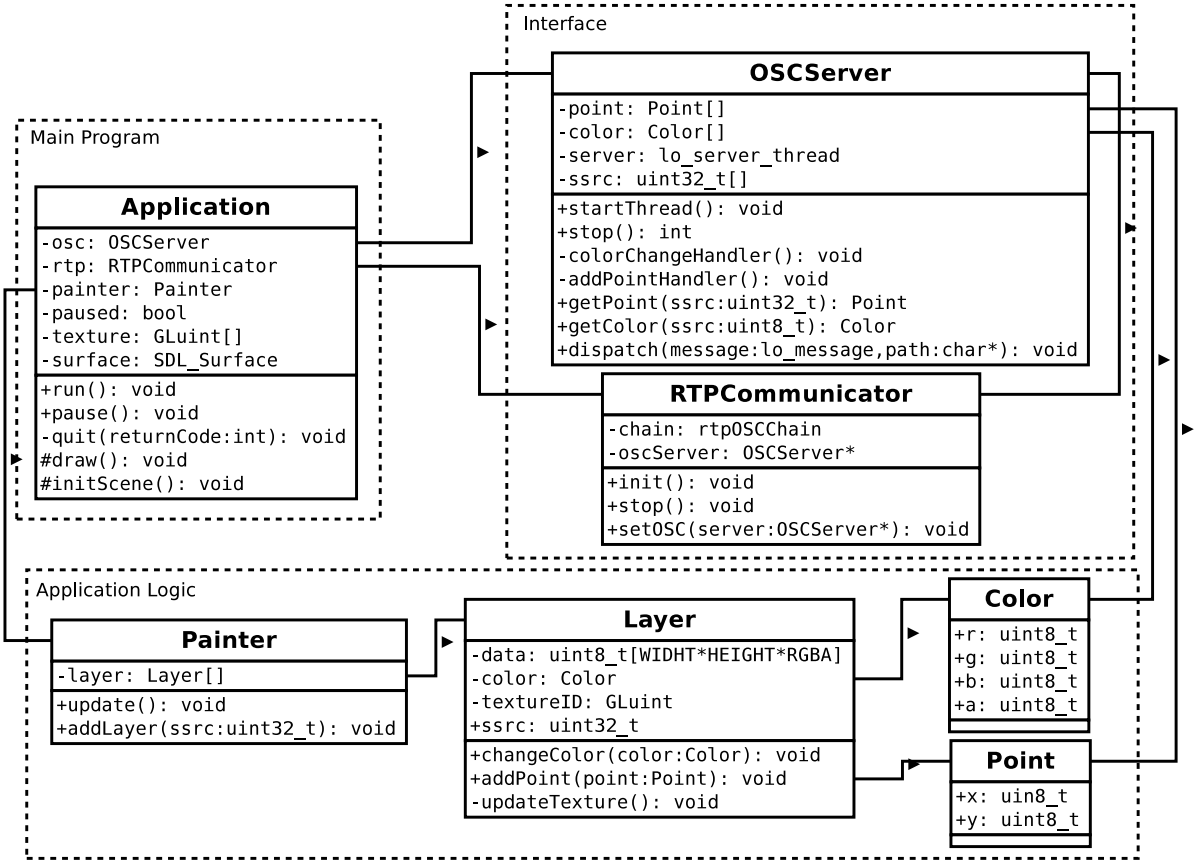


Abbildung 5.2.: Klassendiagramm von der Anwendung für kollaboratives Malen auf der Medienfassade

Die Struktur des Programms ist der von Pong sehr ähnlich. Wieder zeichnen sich die drei Komponenten Hauptprogramm, Schnittstelle und Anwendungslogik ab. Am Klassendiagram in Abbildung 5.2 ist darüber hinaus sehr gut zu beobachten, wie Anwendungslogik mit Schnittstellenlogik verzahnt ist: Schnittstellenlogik enthält einen Platzhalter für jeden steuerbaren Parameter. Werden neue Teilnehmer in der RTP-Session erkannt, wird eine neue Ebene erzeugt. Jede Ebene enthält die entsprechenden Pixeldaten (`data`), eine aktuelle Farbe und die ID der Textur im OpenGL-Umfeld. Auf diese Texture werden die Daten mittels der `updateTexture()`-Funktion intern von der Ebene selbst bei jeder Änderung (über `addPoint()`) neu geladen.

Pro eindeutiger SSRC wird eine Ebene erzeugt, auf der anschließend alle übertragenen Punkte von dieser SSRC gezeichnet werden.

5.4.3. Mediascreen

Die Anwendung Mediascreen unterscheidet sich im Wesentlichen von den vorangegangen Anwendungen dadurch, dass nicht nur OSC-Daten übertragen werden, sondern auch Audio- und Videodaten. Die Grundfunktionalität der Anwendung ist rein zu Demonstrationszwecken: Ein Videostream kann nebst Audiostream und OSC-Stream übertragen werden. Dabei stellt der Videostream einen Livestream von einer Kamera, der Audiostream die Wiedergabe einer Audiodatei von der Festplatte und der OSC-Stream die dynamische Änderung der Position des Videos auf der Medienfassade dar.

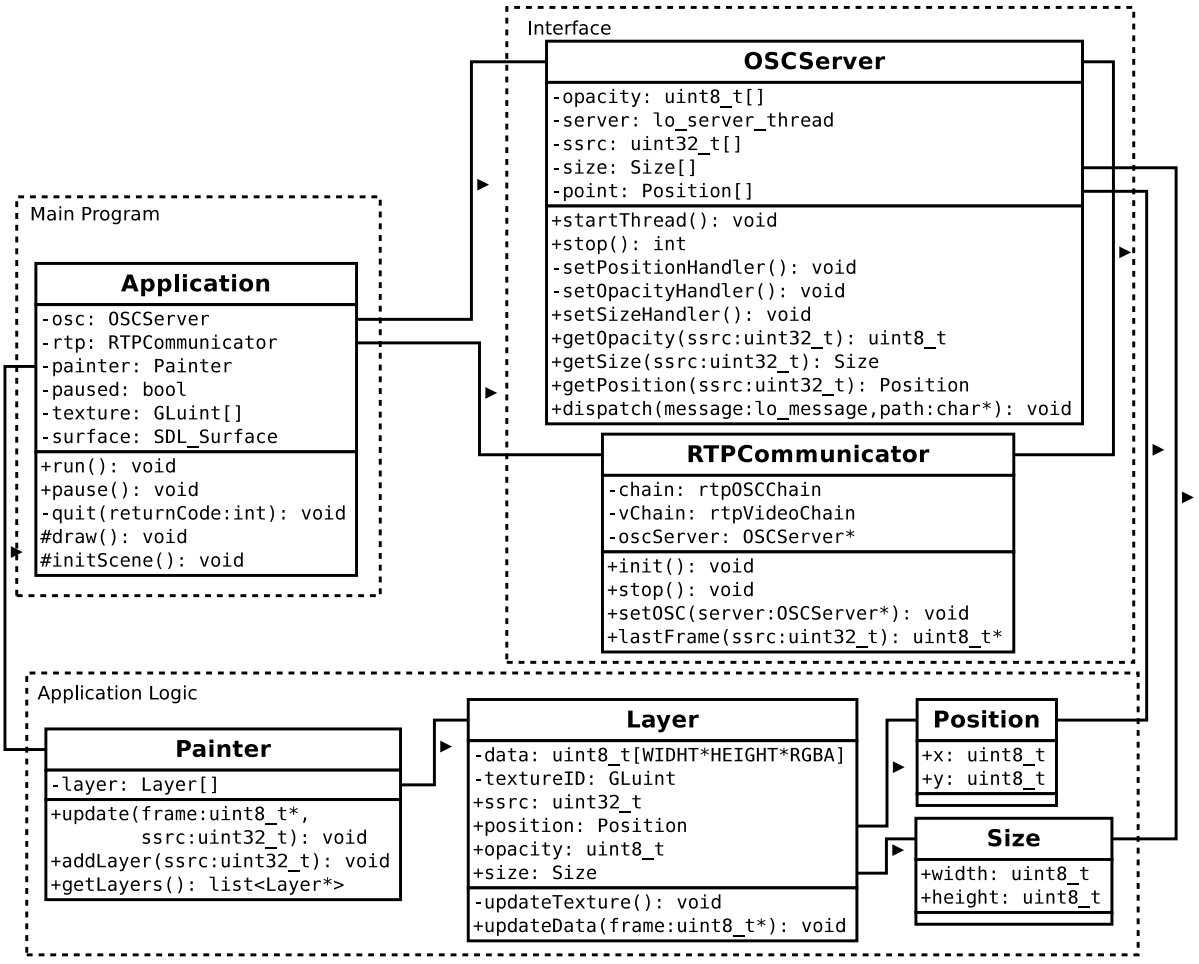


Abbildung 5.3.: Klassendiagramm der Mediascreen-Anwendung

An der grundlegenden Struktur des Programms hat sich auch für die Mediascreen-Anwendung gegenüber Pong und der Anwendung für kollaboratives Malen nichts geändert. Dennoch sind neue Dinge hinzu gekommen, die es erst ermöglichen, über RTP Vi-

deos zu empfangen: durch die `rtpVideoChain` werden nun neben den OSC-Nachrichten auch Videonachrichten entgegen genommen und weiter verarbeitet. Sämtliche OSC-Nachrichten werden wie gehabt vom `RTPCommunicator` an den entsprechenden OSC-Server, welcher wie auch bei den anderen Anwendungen über `setOSC()` festgelegt werden kann, bearbeitet und die Platzhalter der Variablen im OSC-Server gefüllt.

Aus der Anwendung für kollaboratives Malen konnten sogar weitere Klassen übernommen werden, da es letztlich für die Visualisierung keinen Unterschied macht, woher die Pixeldaten für die Ebenen kamen. So sind sowohl Painter, als auch Layer zwar erweitert, aber im Grunde übernommen worden. Neu ist, dass per `update()` im Painter und `updateData()` im Layer die Daten, welche aus dem `rtpVideoStream` empfangen wurden, in die entsprechende Textur der Ebene gezeichnet werden.

6. Evaluation

In diesem Kapitel werden die geschaffenen Lösungen unter verschiedenen Gesichtspunkten getestet und Vor- und Nachteile beschrieben. Dabei werden zunächst die geschaffene Schnittstelle, anschließend die weiteren Komponenten und schließlich die Beispielanwendungen ausgewertet.

6.1. Testaufbau

Für die Evaluation der Schnittstelle selbst wurden zwei Szenarien getestet, die auch im Produktivbetrieb der Fassade auftauchen können: Das Netzwerk kann dabei unterschiedlich aufgebaut sein: es kann beispielsweise das Internet mit einschließen oder ausschließlich lokal sein. Des Weiteren kann die Medienfassade entweder per Ethernet oder per W-LAN (wie schon die Clients) angeschlossen sein. Da die Medienfassade jedoch meist stationär ist und somit fest an eine Netzinfrastruktur angebunden werden kann (per Ethernet), wurden folgende zwei Szenarien getestet:

- Ein Client ist per W-LAN in ein Heimnetz eingebunden, welches per 1 Mbit/s Upstream und 5 Mbit/s Downstream an das Internet angebunden ist, und überträgt seine Daten an einen über Ethernet ans Internet angebundenen Rechner, der als Medienfassade fungiert.
- Ein Client ist per W-LAN zu einem Heimnetz verbunden, an welchem per Ethernet ein Rechner angebunden ist, der als Medienfassade fungiert.

Als W-LAN wurde in beiden Fällen ein 54 Mbit/s W-LAN-Netz verwendet. Die Ethernetverbindung zum Medienfassadenrechnern war jeweils eine Gigabit-Ethernet Verbindung.

Die Testumgebung der Beispielanwendungen wich insofern von der Testumgebung der Schnittstelle ab, als dass der Client nicht zu einem privaten W-LAN verbunden war,

sondern zum hochschulweiten Eduroam. Zu diesem Netz waren auch die Rechner der Medienfassade verbunden. Auf einem der Rechner der Medienfassade liefen dabei Anwendung (an ClusterGL angebunden), Launcher, sowie ein ClusterGL-Renderer und auf dem anderen Rechner ein weiterer ClusterGL-Renderer. Der eine ClusterGL-Renderer war für das Bespielen des Beamers mit dem oberen Teil des Bildes, der andere Renderer für das Bespielen der LED-Grid-Module zuständig. Die Anwendungsbefehle an OpenGL wurden von ClusterGL abgefangen und per Gigabit-Ethernet an den anderen Rechner bzw. an Localhost gesendet, dort je von einem ClusterGL-Renderer empfangen und wieder in OpenGL-Befehle umgewandelt, die letztlich zur Darstellung des Programms auf den zwei Teilen der Testfassade führten.

Die Interaktion mit den Beispielanwendungen fand dabei mit verschiedenen Eingabegeräten und -methoden statt, um die Tauglichkeit der Schnittstelle unter Beweis zu stellen.

6.2. Schnittstellenevaluation

Für die geschaffene Softwareschnittstelle kann in verschiedenen Konfigurationen des Netzwerks und der Schnittstelle der Paketverlust bei der Übertragung von Nachrichten gemessen werden.

Ein Paketverlust kann bei OSC ebenso wie bei MIDI je nach Inhalt des verlorenen Pakets verheerende Folgen haben: Bei MIDI sind es unter anderem Nachrichten über Tasten, die wieder losgelassen wurden, die bei Verlust schnell durch eine wesentlich länger als gewollt spielende Note erkennbar werden. Bei OSC können es beliebige Informationen sein, die verloren gehen – sei es ein Text, der dargestellt werden soll oder eine selten vorkommende, aber gravierende Parameteränderung (z. B. der Wechsel der Hintergrundfarbe bei Pong).

Für Video- und Audiostreams ist es hingegen meist nicht allzu gravierend, wenn Pakete verloren gehen: es fehlt schlimmstenfalls ein Bild bzw. eine kurze Audiosequenz, was sich als Flackern im Bild oder Ausetzern bei der Tonwiedergabe bemerkbar macht. Bei komprimierten Datenformaten, welche zur Dekompression die Historie empfangener Pakete verwendet, wird üblicherweise in festen Intervallen ein neuer Referenzwert geschickt (z. B. Keyframes bei Videos). Solch einen Mechanismus könnte man auch für OSC einführen. Der potenzielle Nachteil eines solchen Mechanismus liegt allerdings in der nicht

festgelegten Menge an Adressen, die eine OSC-Anwendung maximal unterstützen kann. Das würde bei vielen Parametern dazu führen, dass solch ein Referenzpaket mit allen Werten aller Adressen schnell sehr groß werden kann.

Anhand der in den folgenden Tests gemessenen Werte können Aussagen darüber getroffen werden, ob es notwendig oder sinnvoll ist, ähnliche Journaling Mechanismen, wie für MIDI über RTP bereits spezifiziert. auch für OSC über RTP zu definieren.

Durchführung und Ergebnisse

RTP ist so konzipiert, dass in regelmäßigen Abständen eine Menge von Paketen geschickt wird. Dieser zeitliche Abstand zwischen den Paketen ist variabel konfigurierbar. Gemessen wurde je Testreihe der Paketverlust bei 10 000 OSC-Nachrichten. Dieser Test wurde pro eingestelltem zeitlichen Abstand zwischen den Paketen 100 mal durchgeführt. Dabei wurde jeweils der Paketverlust beim Senden der Nachricht zur Fassade, als auch beim Senden der Nachricht von der Anwendung an sich selbst (also localhost) gemessen.

Die Boxplot Diagramme lesen sich derart, dass immer der schwarze waagerechte Strich in einer Box der Medianwert aller Messungen für das Intervall darstellt. Die Box beschreibt den Bereich, in dem 50 % der Anzahlen empfangener Pakete von maximal 10 000 aus den 100 Versuchen pro Intervall liegen. Die Antennen (durch gestrichelte senkrechte Linien gekennzeichnet, welche von der jeweiligen Box nach oben und unten abgehen und mit kurzen wagerechten Linien abgeschlossen werden) beschreiben den weiteren Wertebereich bis zur maximalen bzw. minimalen Anzahl empfangener Pakete. Sie werden allerdings per Definition auf maximal das dreifache der Länge der Box beschränkt. Alle weiteren Werte werden als Ausreißer betrachtet und als nicht gefüllte Kreise dargestellt.

Zunächst wurden die Tests über das lokale W-LAN-Netz mit der Medienfassade per Ethernet an den Router angeschlossen durchgeführt. Bei bereits 10 µs Abstand zwischen den generierten Paketen ist hier die Anzahl empfangener Pakete bei localhost im Mittel bei 9460 gewesen (vgl. Abbildung 6.1). Zwar liegt hier die untere Grenze der Antennen noch bei etwa 9000, was 90 % entspricht, aber diese ist bei den folgenden Versuchsreihen erkennbar nach oben gewandert (vgl. auch Abbildung 6.3) und lag bei 100 µs bereits bei über 9850. Gleichzeitig erreichen den Fassadenrechner von diesen 10 000 Paketen pro Test bei so geringen Paketabständen wie 10 µs bis 30 µs teilweise weniger als 75 % der Pakete (mit Ausreißern bis unter 60 %). Der Median lag aber auch hier von Beginn an

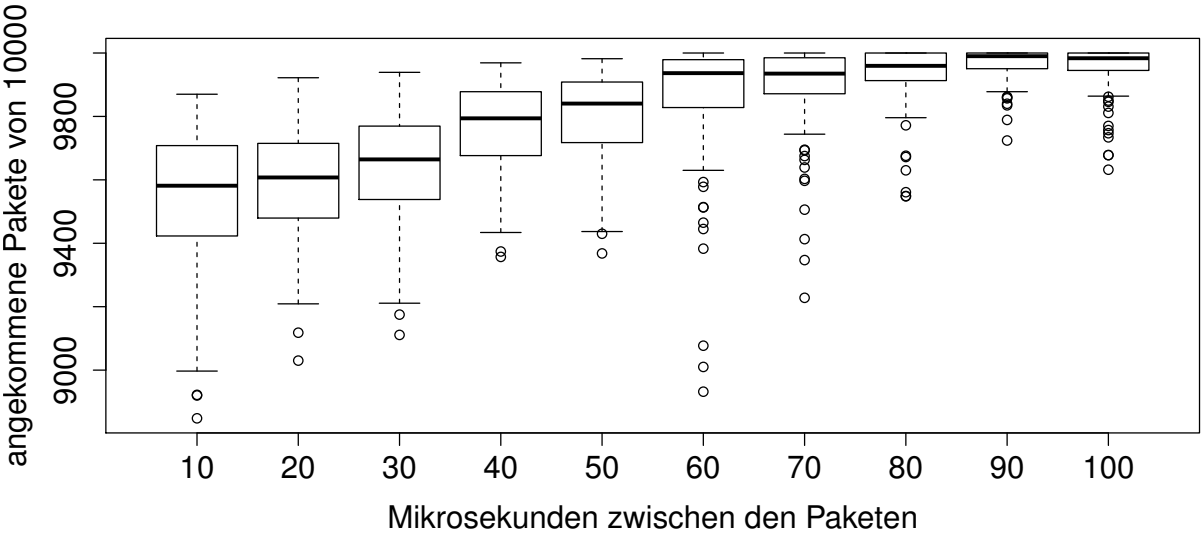


Abbildung 6.1.: Paketverlust bei localhost in der Testreihe über LAN

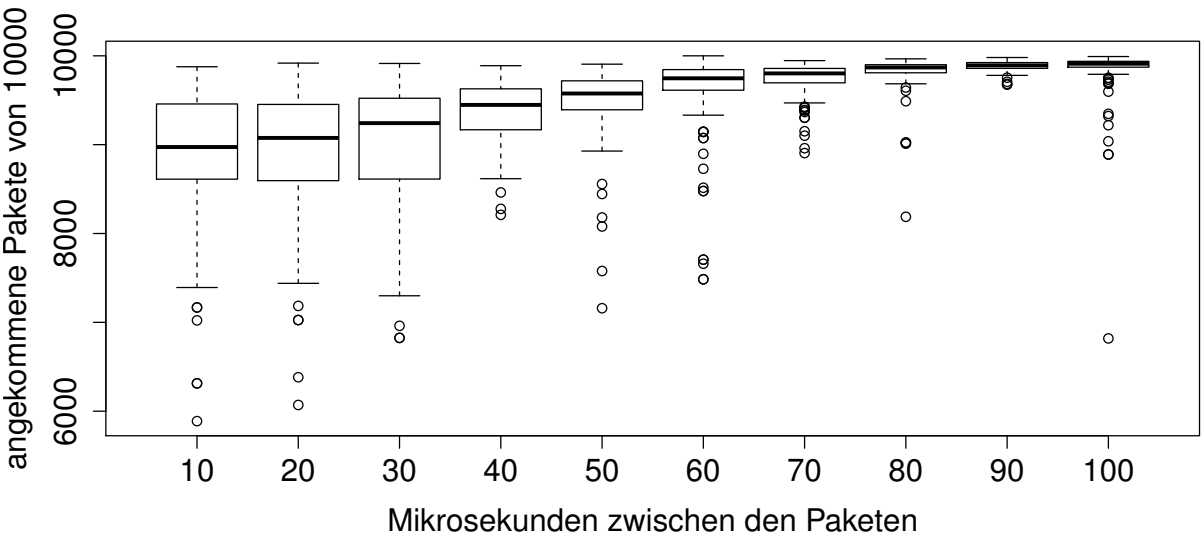


Abbildung 6.2.: Paketverlust beim Fassadenrechner in der Testreihe über LAN

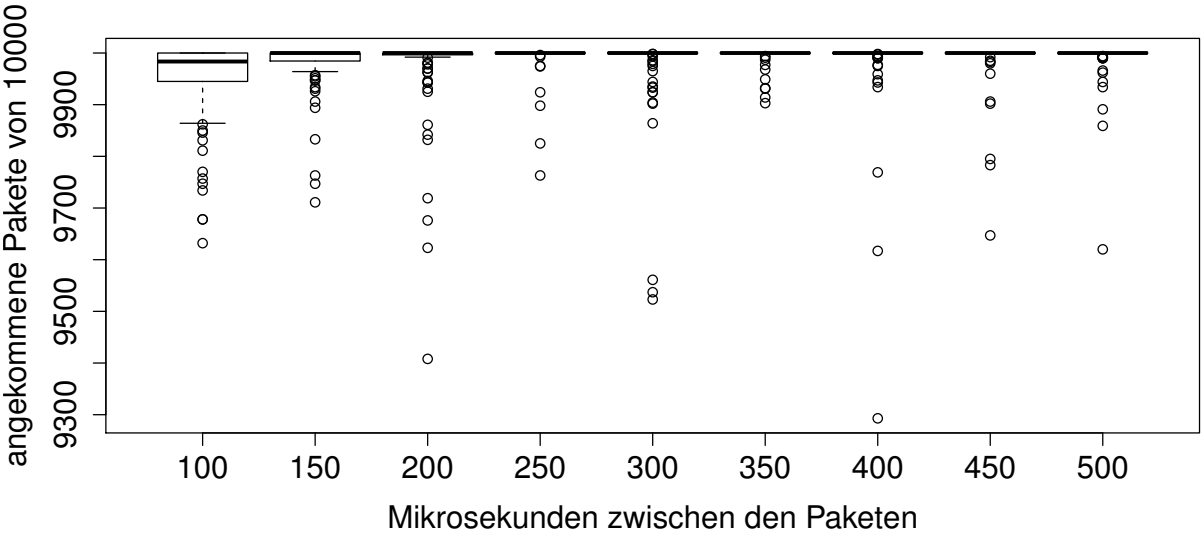


Abbildung 6.3.: Paketverlust bei localhost in der Testreihe über LAN (2)

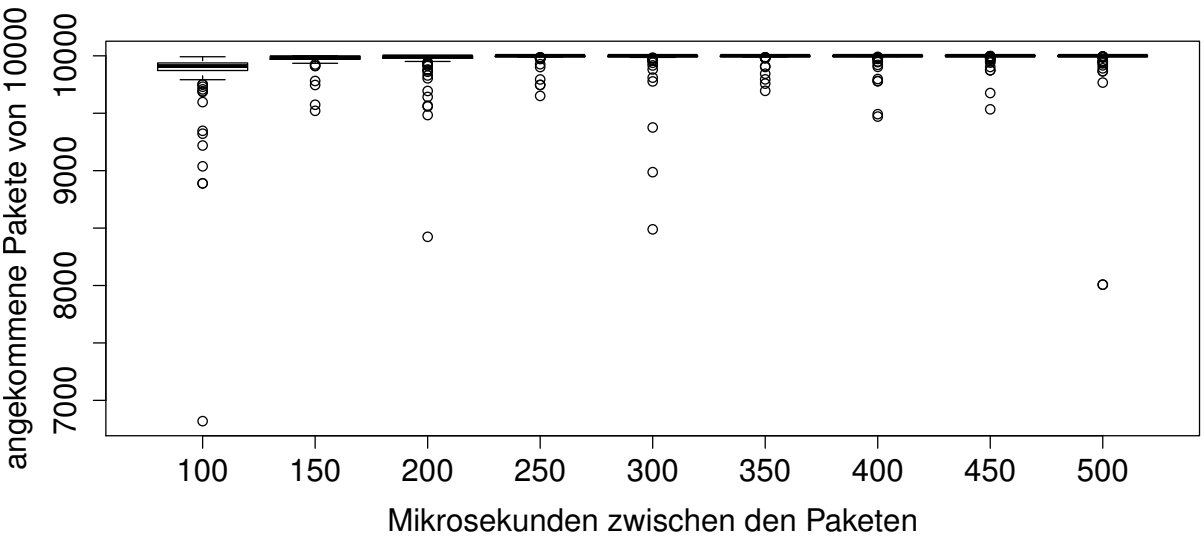


Abbildung 6.4.: Paketverlust beim Fassadenrechner in der Testreihe über LAN (2)

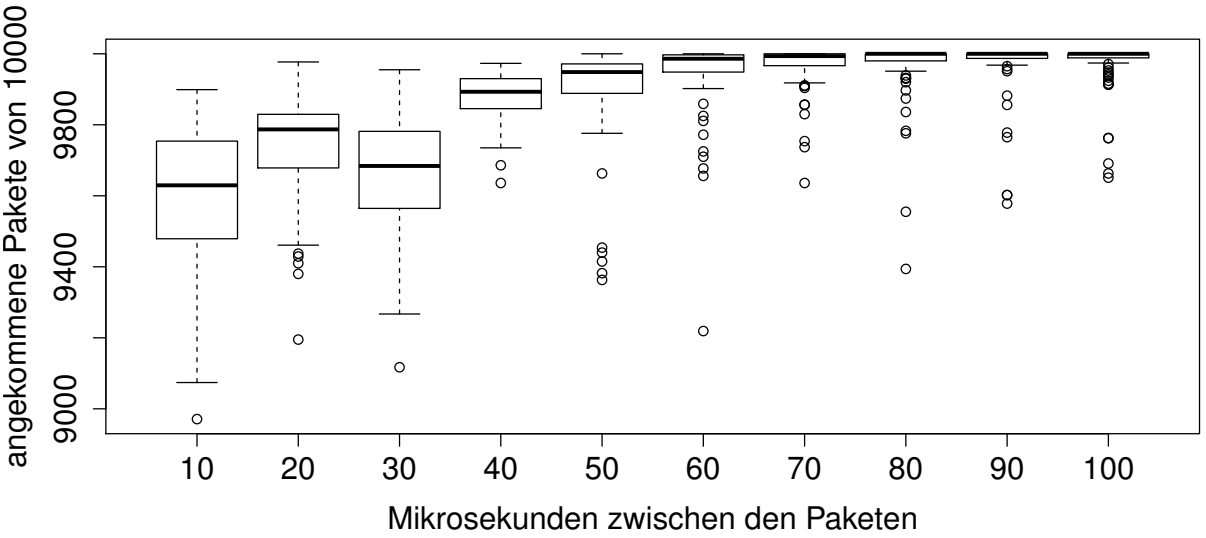


Abbildung 6.5.: Paketverlust bei localhost in der Testreihe über Internet

bei etwa 90 % mit klarer Tendenz nach oben. Bereits bei 90 µs ist die Box, in der 50 % der Werte liegen bei etwa 9900 Paketen so weit zusammen geschrumpft, dass sie kaum noch zu erkennen ist – hier kommen also nur noch sehr selten Pakete nicht am Ziel an. Der Trend setzt sich, wie in Abbildung 6.4 zu erkennen, bis auf einige teils massive Ausreißer fort. Diese Ausreißer sind unter anderem damit zu begründen, dass weiterer Netzwerkverkehr über TCP in dem Moment die Leitung ungewöhnlich stark belegte. Dieser Fall, dass das Netz gleichzeitig noch von anderen Diensten so intensiv genutzt wird, ist im Produktivbetrieb also unbedingt zu vermeiden.

Anschließend wurde derselbe Test ein weiteres mal mit dem einen Client per W-LAN in einem Heimnetz und über Internet zu einem per Ethernet angeschlossenen Medienfassadenrechner durchgeführt. Dabei fiel auf, dass bei 10 µs bis 30 µs nur etwa 70 % bis 80 % der Pakete ihr Ziel erreichen (vgl. Abbildung 6.6). Ab 40 µs steigt die mittlere Paketanzahl bei den 100 Tests allerdings stark an. Dies liegt daran, dass ab 40 µs die Upstream-Leitung des DSL Anschlusses nicht mehr voll ausgelastet war, wie das bei Paketabständen unter 40 µs der Fall war. Zu Erkennen ist auch in den Grafiken Abbildung 6.9 und Abbildung 6.10, dass sich der Trend weiter fortsetzt und letztlich bis auf einen starken Ausreißer bei 450 µs beim Fassadenrechner mit etwa 50 % Paketverlust in den meisten Fällen bei 100 % liegt. Auch dieser starke Ausreißer ist wieder auf intensiven TCP Verkehr parallel zum Test zurück zu führen.

Die durchgeführten Tests zeigen, dass ein Journaling System tatsächlich vermutlich nicht sinnvoll ist. Diese Überzeugung wächst aus zwei Gründen: zunächst ist bei einer Pake-

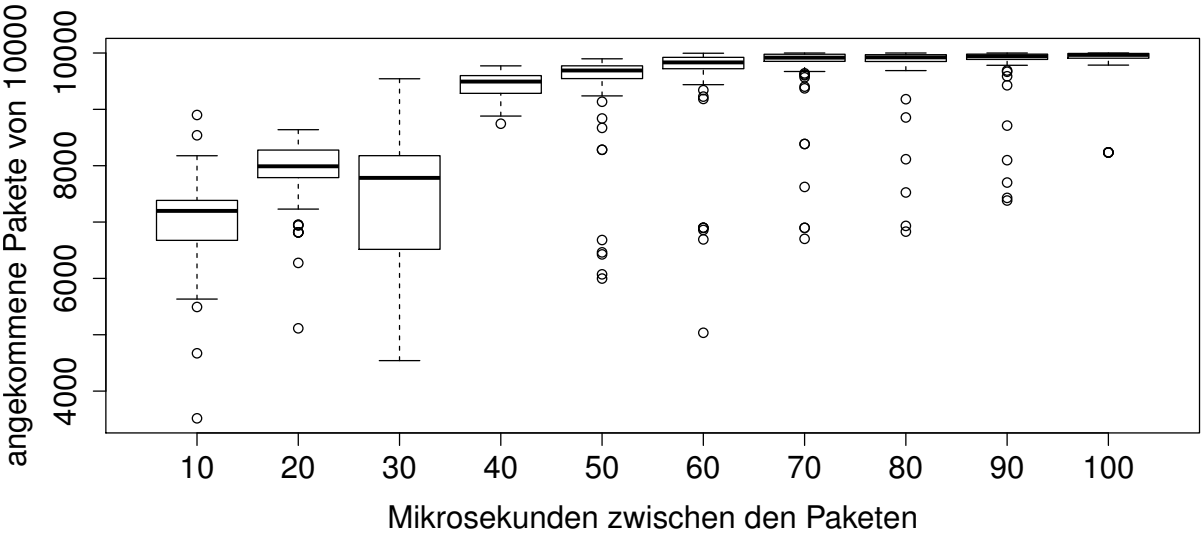


Abbildung 6.6.: Paketverlust beim Fassadenrechner in der Testreihe über Internet

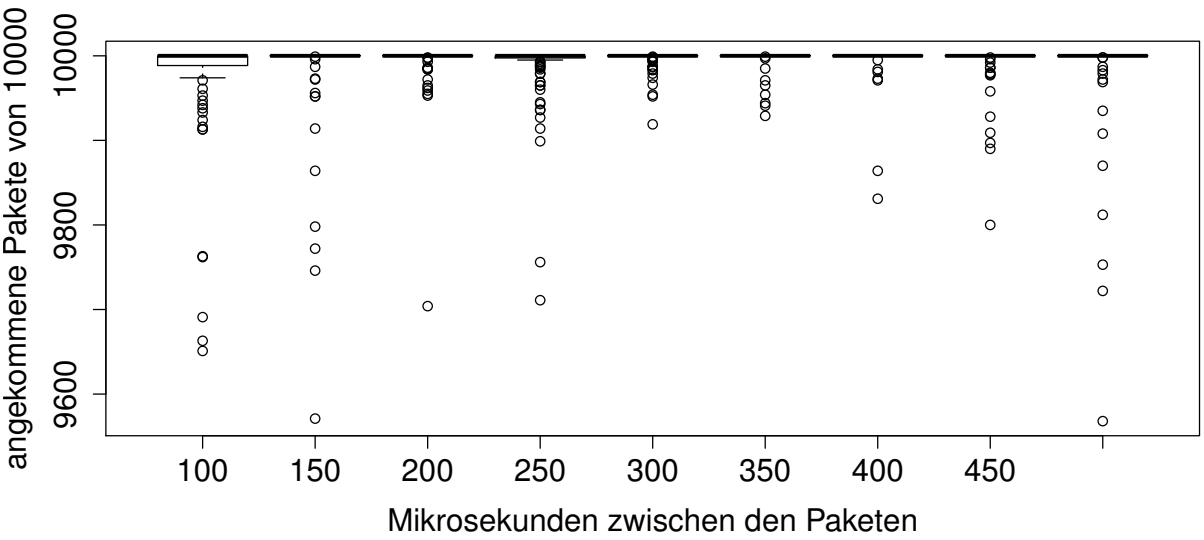


Abbildung 6.7.: Paketverlust bei localhost in der Testreihe über Internet (2)

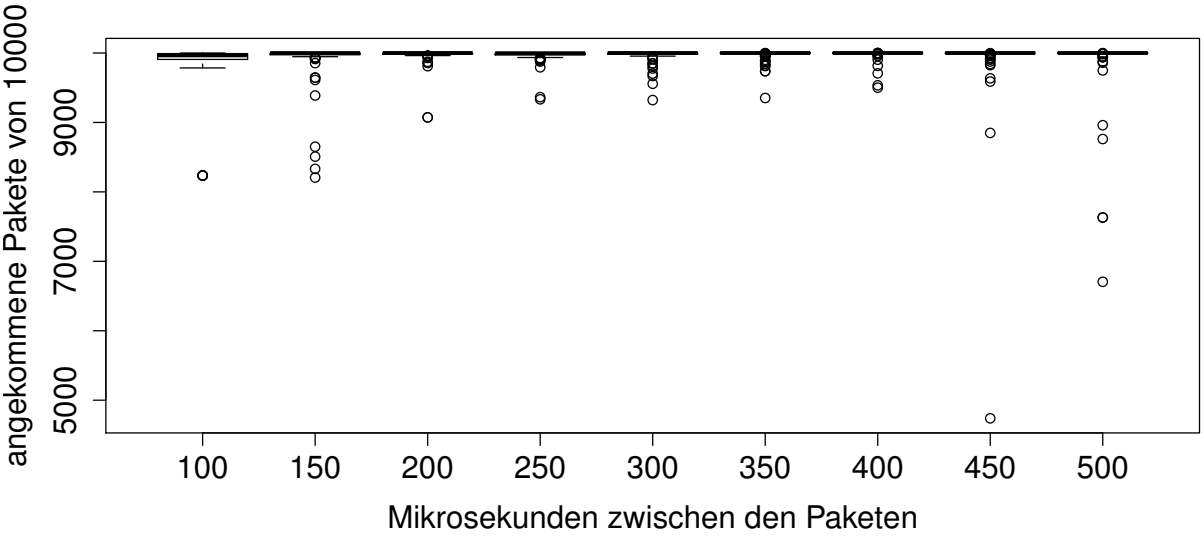


Abbildung 6.8.: Paketverlust beim Fassadenrechner in der Testreihe über Internet (2)

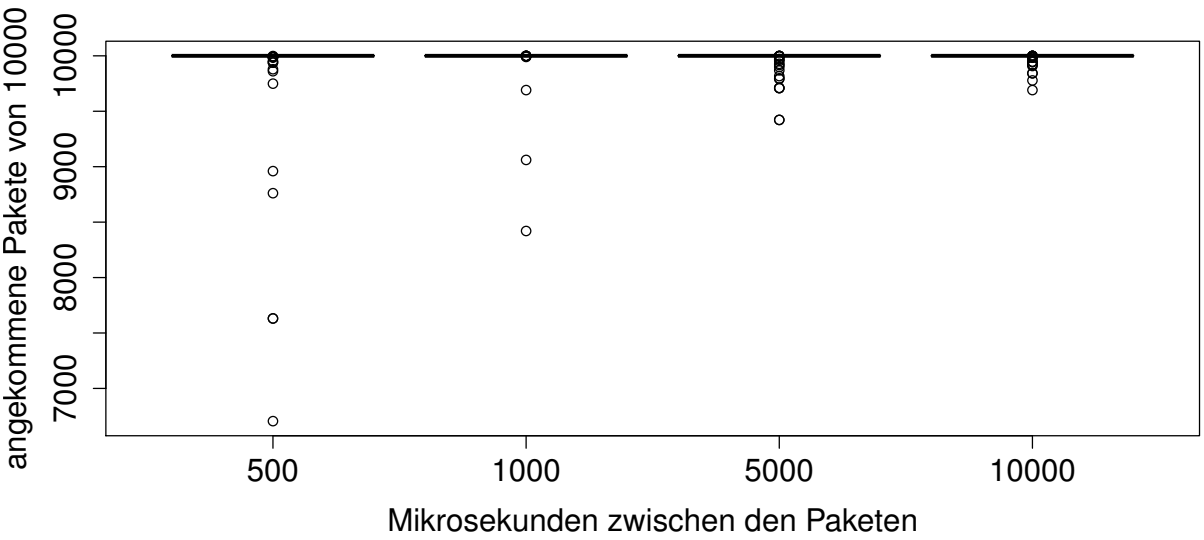


Abbildung 6.9.: Paketverlust bei localhost in der Testreihe über Internet (3)

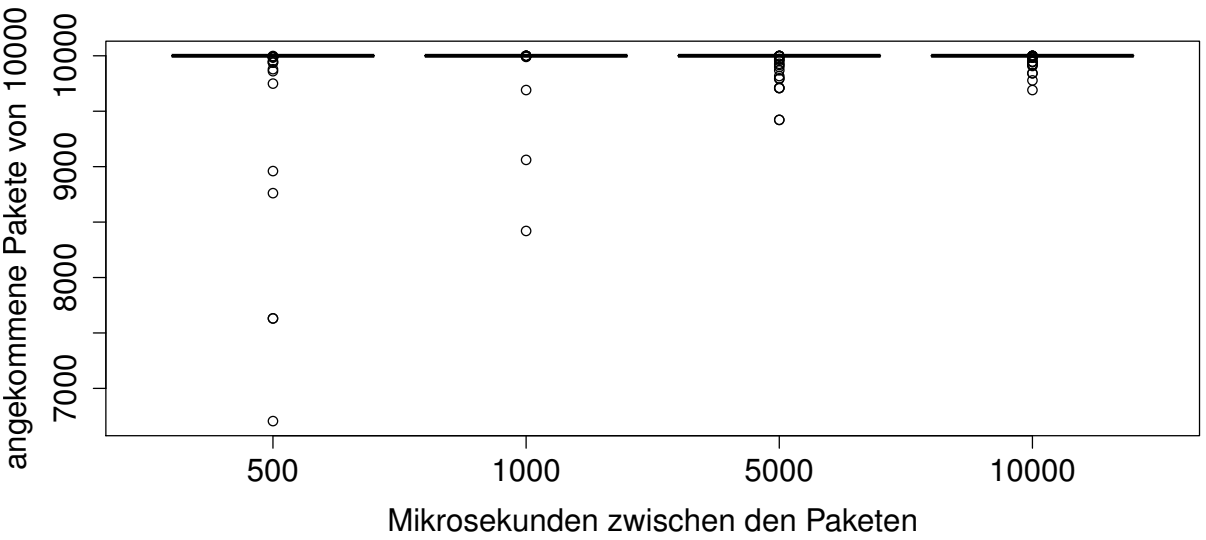


Abbildung 6.10.: Paketverlust beim Fassadenrechner in der Testreihe über Internet (3)

trate von 10 000 Paketen pro Sekunde (100 µs Abstand zwischen den Paketen) die Menge an Paketen, die nicht am Ziel ankommt bereits sehr minimal und ein Journaling System würde zwangsläufig mehr Daten pro Paket versenden müssen. Diese Daten müssen nicht nur gesendet, sondern auch generiert und wieder ausgewertet werden. OSC eignet sich durch seine flexible Adressstruktur und dadurch schnell größer werdende Pakete weiterhin nicht für ein Journaling mittels Paketdifferenzen, da aufeinanderfolgende Pakete sehr stark voneinander abweichen können.

Eine Möglichkeit, die Paketrate zu vergrößern, wäre eine wesentlich aufwändigere lokale Nachrichtenverarbeitung, bei der Pfade, an die bereits gesendet wurde, zwischengespeichert werden, dem Empfänger gleichzeitig für diesen Pfad ein eindeutiges Kürzel übermittelt wird. Der Empfänger baut aus diesen Kürzeln eine lokale Lookup Tabelle auf. Anschließend sendet der Sender nur noch unter Verwendung dieser Kürzel. Dadurch werden gesendete Nachrichten wesentlich kleiner und ein Journaling eventuell doch sinnvoll. Diese Variante hätte allerdings den gravierenden Nachteil, der sie ad absurdum führt, dass nie klar ist, welche Nachricht angekommen ist und welche nicht – welcher Teil der Lookup Tabelle beim Empfänger bekannt ist und welcher eventuell fehlt.

Eine andere Möglichkeit wäre die Kompression der Pakete mittels bekannter Kompressionsalgorithmen. Dies würde aber ein Prinzip von OSC unterlaufen, da die einfach lesbare Datenstruktur dadurch verloren ginge.

Insgesamt ist für diese Arbeit also der einzig sinnvolle Weg die Vergrößerung des Abstandes zwischen Paketen, bis eine akzeptable (<2 %) Paketverlustrate erreicht ist. Dabei ist

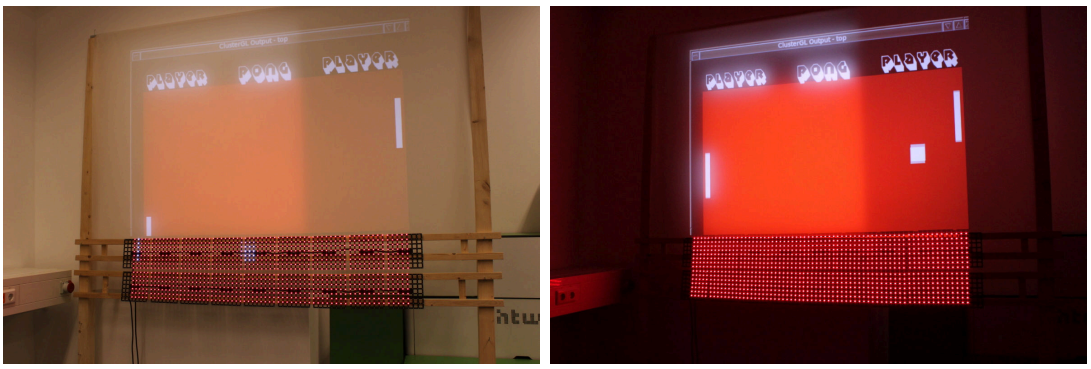
allerdings mit Nachdruck darauf hinzuweisen, dass die durchgeführten Test zwar mit Paketen unterschiedlichen Inhalts, aber doch sehr kleinen Paketen (etwa 60 Byte pro Paket) durchgeführt wurden. Es ist daher ratsam, zunächst noch einen größeren Abstand zwischen den Paketen zu wählen und bei der Implementierung darauf zu achten, dass die Adressen lesbar aber möglichst kurz sind.

6.3. Evaluation des Launchers

Der Launcher wurde im Rahmen der Arbeit vor allem auf sein Langzeitlaufverhalten hin geprüft und dabei konnten bis zuletzt keine Einschränkungen festgestellt werden. Er läuft stabil auch Tage und Wochen hindurch. Lediglich zu beachten ist, dass der von ihm zur Verfügung gestellte OSC-Server auf einem anderen Port läuft, als der jeweilige OSC-Server der laufenden Anwendung. Es ist nicht sinnvoll, diesen OSC-Server des Launchers so einzurichten, dass er automatisch alle ihm unbekannten Nachrichten an den OSC-Server der Anwendung sendet, da dies nur eine weitere Fehlerquelle im System einführen würde. Zudem würde diese Lösung eine weitere Komponente in die Verarbeitungskette der OSC-Nachrichten einführen, was Latenz in das System bringen kann. Sollte es aus bestimmten Gründen sinnvoll sein, doch die Nachrichten über nur einen Port zu empfangen, so ist per Firewall einzurichten, dass diese Nachrichten intern an den jeweiligen Applikationsport weiter geleitet wird. Es ergibt sich daraus, dass es sinnvoll ist, allen Applikationen denselben Port fest zuzuweisen, da sonst eine solche Firewall-Regel nur schwer zu realisieren wäre. Daher ist der Port für die Anwendung zwar im Quelltext, aber nicht per Konfiguration zur Laufzeit änderbar implementiert worden.

6.4. Evaluation der Beispielanwendungen

Auch die Beispielanwendungen wurden in der Testumgebung getestet. Dabei wurden grobe Fehler gefunden und behoben und die Anwendungen im Wesentlichen auf ihre Wirkung auf der größeren Umgebung hin untersucht.



(a) Pong auf der Testfassade in beleuchteter Umgebung

(b) Pong auf der Testfassade in dunkler Umgebung

Abbildung 6.11.: Pong auf der Testfassade

6.4.1. Pong

Die Anwendung Pong wurde bereits bei der Langen Nacht der Wissenschaften 2012 in ihrer damaligen Form (serielle Kommunikation direkt mit der Anwendung ohne OSC oder RTP) gezeigt. Seitdem wurde sie wie in Unterabschnitt 5.4.1 beschrieben von der Programmiersprache C in C++ überführt und mit der in dieser Arbeit beschriebenen generalisierten Schnittstelle ausgestattet.

Die Anwendungslogik (Kollisionserkennung und -behandlung, sowie das Zählen der Punkte) bliebt dabei gleich und in Abbildung 6.11 ist die Anwendung einmal in beleuchteter Umgebung und einmal in nicht beleuchteter Umgebung abgebildet.

Die Bedienelemente sind so groß, dass sie auch auf den LED-Modulen noch zu sehen sind und nicht etwa durch das herunterskalieren der Auflösung verloren gehen. Auch der Ball ist immer zu sehen. Die Schrift kann nur auf dem hochauflösenden Bild des Beamers lesbar dargestellt werden – auf den LED-Modulen wäre sie nicht zu erkennen.

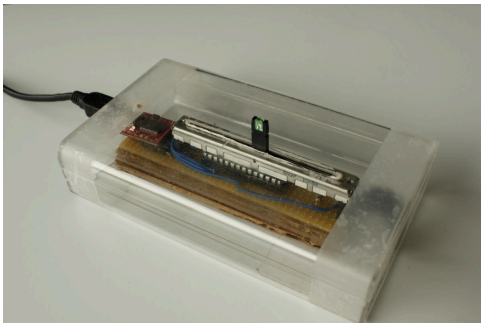


Abbildung 6.12.: Eingabegerät für Pong

(a) Anwendung zum kollaborativen Malen auf der Testfassade in beleuchteter Umgebung

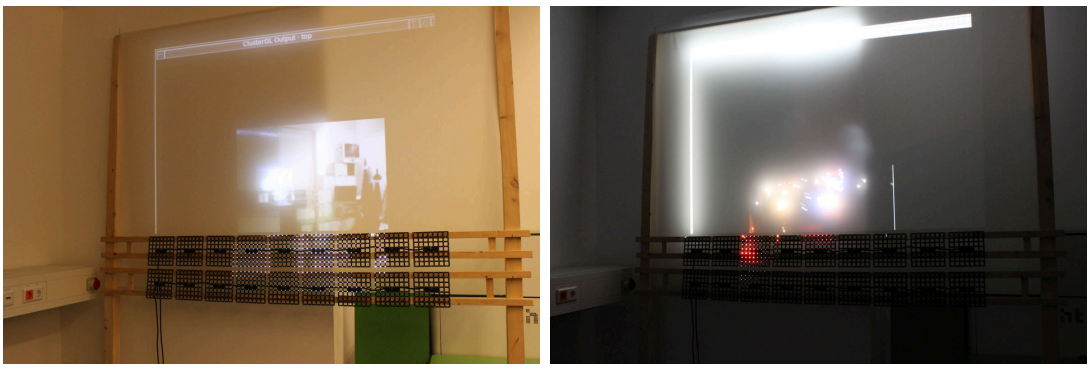
(b) Anwendung zum kollaborativen Malen auf der Testfassade in dunkler Umgebung

Abbildung 6.13.: Anwendung zum kollaborativen Malen auf der Testfassade

Bereits bei der langen Nacht der Wissenschaften wurden an der Bedienbarkeit des Systems keine Akzeptanzhürden oder Fehler festgestellt. Auch mit der neuen Schnittstelle, durch die nur die Art der Kommunikation des Eingabegeräts (siehe Abbildung 6.12) mit der Fassade geändert wurde, hat dort keine Änderungen mit sich gebracht. Das in Abbildung 6.12 dargestellte Eingabegerät für Pong wurde extra für diese Anwendung für die Lange Nacht der Wissenschaften entworfen, umgesetzt und programmiert und besteht im Wesentlichen aus einem 8-Bit Mikrocontroller (Atmega8), einem Schieberegler mit LED, sowie einem Wandler von serieller Schnittstelle auf USB. Da an diesem Gerät noch keine Netzwerkschnittstelle vorhanden war, wurde ein Programm geschrieben, dass die über serielle Schnittstelle empfangenen OSC-Nachrichten per Netzwerk weiter an die Fassade schickt. Generell hätte für die Integration dieses DIY-Controllers eine weitere Komponente hinzugefügt werden müssen, um ihn drahtlos zu nutzen. Daher stellt die Verwendung eines Notebooks für diese Zwecke keine unnötige Verkomplizierung dar. Für eine zukünftige Nutzung wäre es an der Stelle allerdings sinnvoll, eine W-LAN Schnittstelle direkt in den Controller zu integrieren. Dies war jedoch nicht der Fokus dieser Arbeit und wurde daher nicht umgesetzt.

6.4.2. Kollaboratives Malen

Für die Anwendung zum kollaborativen Malen wurde nicht nur auf eine bestehende Anwendungsidee zurückgegriffen, sondern auch zur Bedienung auf bestehende Anwendungen: Mit einem Nokia N9, auf dem das Betriebssystem MeeGo und die Anwendung



(a) Mediascreen-Anwendung auf der Testfassade in beleuchteter Umgebung

(b) Mediascreen-Anwendung auf der Testfassade in dunkler Umgebung

Abbildung 6.14.: Mediascreen-Anwendung auf der Testfassade

GoOSC installiert sind wurde per OSC auf der Fassade gemalt. Weiterhin wurde mit einem iPod per OSCTouch ein vergleichbares Layout angelegt um die Anwendung zu testen. Beide Geräte konnten gleichzeitig auf der Fassade malen und ein Beispielbild ist in Abbildung 6.13 sowohl in heller, als auch in dunkler Umgebung zu sehen. Beim Vergleich der beiden Abbildungen fällt auf, dass die rechte Abbildung stark überstrahlt erscheint. Das liegt an zwei Faktoren: die Belichtungszeit ist aufgrund der dunklen Umgebung länger und helle Punkte wirken dadurch noch heller und zum Zweiten ist hinter dem linken Teil der Testfassade eine Wand. Da die Rückprojektionsfolie lichtdurchlässig ist, wird von der Wand zusätzlich Licht reflektiert. Das gleiche Problem tritt auf, wenn die Medienfassade am FKI tagsüber betrieben wird: Wenn die Räume nicht abgedunkelt werden, ist dadurch von außen das auf der Rückprojektionsfolie Gezeigte nicht mehr deutlich zu erkennen.

Es fiel bei den Tests der Anwendung auf, dass erwartungsgemäß erst ab einer bestimmten Größe des Pinsels (in diesem Fall von 6x6 Pixeln) die Punkte auch auf den LED-Modulen sichtbar gezeichnet wurden. Bei einer Auflösung der Beispielanwendungen von 640x480 Pixel und einer Breite der LED-Module unter der Beamerfläche von 96x16 Pixeln ist das auch erklärbar, denn $640/96$ ergibt $6,666$. Hier kommt zum Tragen, dass die unterschiedlichen Auflösungen durch Skalierung eines großen Bildes auf kleine Flächen zu Informationsverlusten führt.

6.4.3. Mediascreen

In Abbildung 6.14 ist dritte Anwendung dargestellt: Per Webcam wird an einem Rechner ein live-Video aufgenommen, von der EMIPLIB encodiert, per W-LAN an den Fassadenrechner geschickt, von diesem dekodiert und von der Anwendung schließlich dargestellt.

Zusätzlich ist über dieselbe Schnittstelle durch das im Rahmen dieser Arbeit umgesetzte OSC über RTP die Position, Größe und Durchsichtigkeit der Darstellung des Videos auf der Fassade änderbar. Beim Test sind keine nennenswert höheren Verzögerungen in der Bildwiedergabe aufgetreten, als bei direktem Anschluss der Kamera und Nutzung des Bildes ohne die Übertragung über RTP.

Auch in diesen Darstellungen erkennt man gut, wie sich die Durchlässigkeit der Rückprojektionsfolie auf das dargestellte Bild auswirkt. Zusätzlich wird auch erstmals sichtbar, dass die LED-Module das Bild bei größerer Helligkeit nicht mehr deutlich wiedergeben können. Das liegt in diesem Fall allerdings daran, dass die LED-Module auf eine sehr geringe Helligkeitsstufe (30 von 256 möglichen Helligkeitsstufen) konfiguriert waren. Da die Medienfassade des FKI nach Norden ausgerichtet ist, wird es nie zu direkter Sonneneinstrahlung kommen und es ist daher nicht zu befürchten, dass die LED-Module tagsüber nicht sichtbar sein sollten.

7. Zusammenfassung

Im Rahmen dieser Arbeit wurde ein System zur interaktiven Nutzung von Medienfassaden geschaffen.

Ziel war es dabei, eine generalisierte Schnittstelle zu schaffen, mit der möglichst viele denkbare Szenarien und Geräte für interaktive Anwendungen abgedeckt werden können. Dabei wurden zunächst bestehende Systeme beschrieben, verwendete und weitere Technologien beschrieben und anschließend herausgestellt, was die Anforderungen an eine solche allgemeine Schnittstelle sind. Es wurde darauf eingegangen, dass letztlich durch solche Interaktionen immer bestimmte Parameter in der laufenden Anwendung geändert werden sollen. Diese können entweder Zahlen und Buchstaben, oder auch weitere abstraktere Dinge wie Farben, Positionen und Geschwindigkeit von Animationen sein. Darüber hinaus wurde klargestellt, dass nicht nur solche Steuerungsdaten über eine generalisierte Schnittstelle übertragen werden sollen, mit der einzelne Elemente der Anwendung gesteuert werden können, sondern es gerade für Medienfassaden interessant ist, Inhalte direkt an die Fassade zu übertragen. Da viele Protokolle entweder auf das Übertragen von Streamingdaten oder das Übertragen von Steuerungsdaten spezialisiert sind, wurden Wege aufgezeigt, wie diese Daten in einem gemeinsamen Protokoll gebündelt werden können. RTP ist dabei zur Kernkomponente des im Systementwurf beschriebenen Gesamtkonzept einer generalisierten Schnittstelle zur Interaktion mit den Anwendungen auf einer Medienfassade geworden. Da jedoch noch kein Payload Type für OSC Daten existierte, wurde im Rahmen dieser Arbeit ebendieser Payload Type geschaffen. In einer prototypischen Implementierung unter Verwendung freier Bibliotheken und Werkzeuge wurde so ein evaluierbares System geschaffen, mit dem sowohl Video- und Audiostreams, als auch Steuerungsdaten parallel über dieselbe Schnittstelle übertragen werden können.

Damit jedoch die auch anvisierte Zielgruppe von Hobby-Elektronikern und Bastlern mit begrenzten Hardware Resourcen eine Möglichkeit haben, mit ihrer selbst gebauten Hardware mit Applikationen auf der Medienfassade zu interagieren, wurde neben OSC

über RTP ebenso parallel OSC allein nutzbar gemacht. Damit ist es zwar nicht möglich, Livestreams von Audio- und Videodaten an die Fassade zu übertragen, dies ist aber mit der entsprechend in ihren Ressourcen begrenzten Hardware wie einem Arduino auch gar nicht sinnvoll machbar.

Neben der Schnittstelle wurden weitere Komponenten für ein flexibles Gesamtsystem entworfen. Dazu zählt ein Launcher, welcher zum Starten verschiedene für die Fassade konfigurierte Anwendungen zum Einsatz kommt. Dieser ist ebenso über OSC anzusprechen. Auch eine Website wurde unter Verwendung eines Webframeworks entwickelt und aufgesetzt. Mit dieser Website sind Medienfassaden, die über die im Rahmen dieser Arbeit implementierten Schnittstelle verfügen, für Administratoren konfigurierbar. Für Nutzer ist über die Website einsehbar, welche Applikation auf welcher Medienfassade läuft und welche Parameter zur Steuerung wie angesprochen werden.

Um die Fähigkeiten der Schnittstelle zur demonstrieren wurden drei Beispielanwendungen implementiert: ein Spiel (Pong), eine Anwendung, wie sie bereits auf anderen Medienfassaden zum Einsatz kommt (kollaboratives Malen) und eine weitere Anwendung, die besonders den Fokus auf die Kombination von Streaming- und Steuerungsdaten zur Interaktion mit Medienfassaden legt (Mediascreen).

Die geschaffene Schnittstelle wurde ausgiebig evaluiert und in Form von Testreihen auf Paketverlustraten hin geprüft. Dabei wurde das Ergebnis erlangt, dass ein spezieller Journaling Mechanismus für diese Art der Anwendung nicht notwendig ist.

8. Ausblick

Mit dem geschaffenen System ist eine Vielzahl von weiteren Anwendungen umsetzbar: von einer Anwendung, mit der man per Beschleunigungssensor und Tastatureingabe Nachrichten an die Fassade „werfen“ kann, über Diashow-Anwendungen (inklusive Videoübertragung), Visualisierung von Umweltdaten, Klangeindrücken und Wetterdaten, hin zu multimedialen Chatanwendungen, bei der die Medienfassade Medium für Menschen wird, mit der Umgebung der Fassade zu interagieren. Darüber hinaus ist eine weitere sehr interessante Form von Anwendungen möglich bei der Medienfassaden untereinander vernetzt miteinander kommunizieren. Diese Anwendungen wären zwar bereits ohne die entworfene Schnittstelle auch schon mit ClusterGL möglich, werden aber unter Verwendung von RTP wesentlich flexibler programmierbar. Mit ClusterGL wäre es möglich, die Visualisierung einer großen Anwendungsumgebung auf mehrere Medienfassaden zu verteilen. Dabei wären aber sämtliche OpenGL-Befehle möglicherweise sogar über das Internet zu übertragen, was zu erheblichen Anforderungen an die verfügbare Bandbreite führen würde. Mit RTP wäre hier nicht nur das dynamische „Zuschalten“ anderer Fassaden möglich, sondern prinzipiell durch die Beschaffenheit von RTP auch eine flexiblere Kontrolle der Datenmengen, die übertragen werden.

Die Lösung OSC über RTP zu übertragen ist weiterhin nicht auf den Anwendungsfall einer Medienfassade beschränkt. Zwar wird Hardware mit OSC-Unterstützung von den Herstellern von beispielsweise Musik-Controllern noch nicht in großer Menge gebaut, OSC gehört aber generell zu den Dingen, die auch in vielen Softwareprodukten der Kreativindustrie (vor allem der Musiksoftwareindustrie) immer stärkere Verbeitung findet. Da OSC allerdings keine Möglichkeiten bietet, Musikstreams zu übertragen, ist eine Kollaboration von Musikern über das Internet immer entweder auf die Übertragung von Steuerungsdaten allein, oder mittels Konferenzsystemen auf ausschließlich Audiodaten beschränkt. Im ersten Szenario müsste jeder Teilnehmer die exakt selben Synthesizer und weiteren Module zur Verüfung haben, damit er dasselbe hört, wie die anderen Teilnehmer. Im zweiten Szenario könnte ein einzelner Teilnehmer nicht mehr manipulieren, was

ein anderer Teilnehmer an seinen Klangerzeugern einstellt. Mittels OSC über RTP und den entsprechenden Audiostreams ebenfalls über RTP ist es nun möglich ein Szenario in der Mitte zu entwerfen: Die Klänge fehlende Instrumente bei einem Teilnehmer werden per Audiostream an ihn übertragen und die Steuerungsdaten für vorhandene Synthesizer und weitere Komponenten können per OSC bedient werden. Dabei geschieht die Übertragung der Daten durch RTP synchronisiert und es ist nur eine Schnittstelle notwendig.

A. Anhänge

A.1. weitere Diagramme und Schaubilder

A.1.1. Launcher

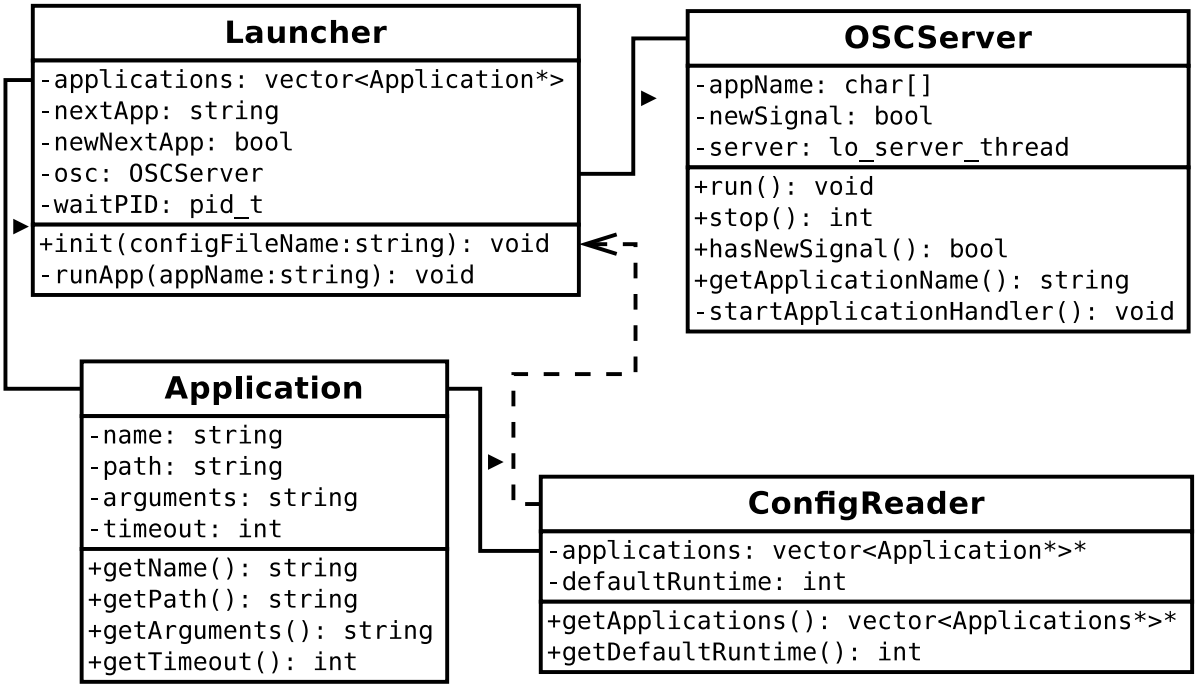


Abbildung A.1.: Klassendiagramm des Launchers für Anwendungen der Medienfassade

A.1.2. Website

ERM

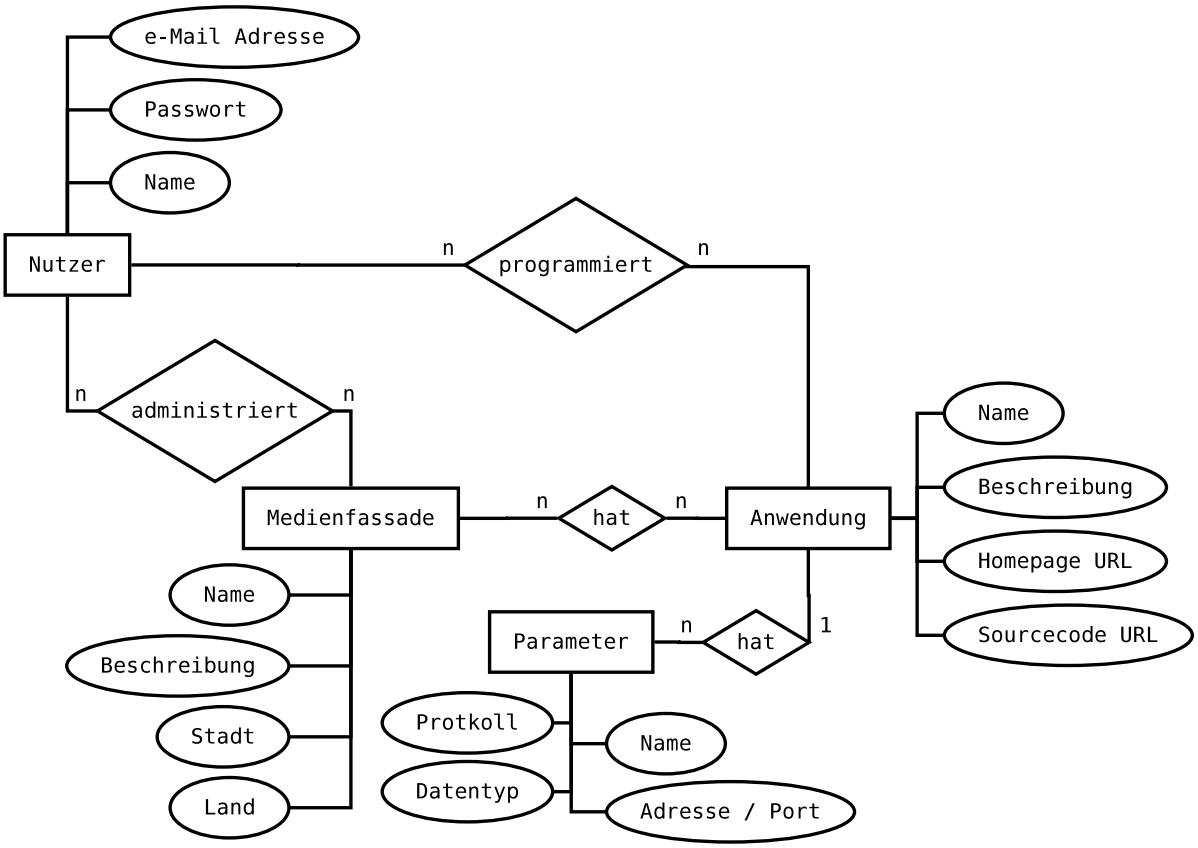


Abbildung A.2.: Entity-Relationship-Modell der Datenbank für die Website

Screenshots

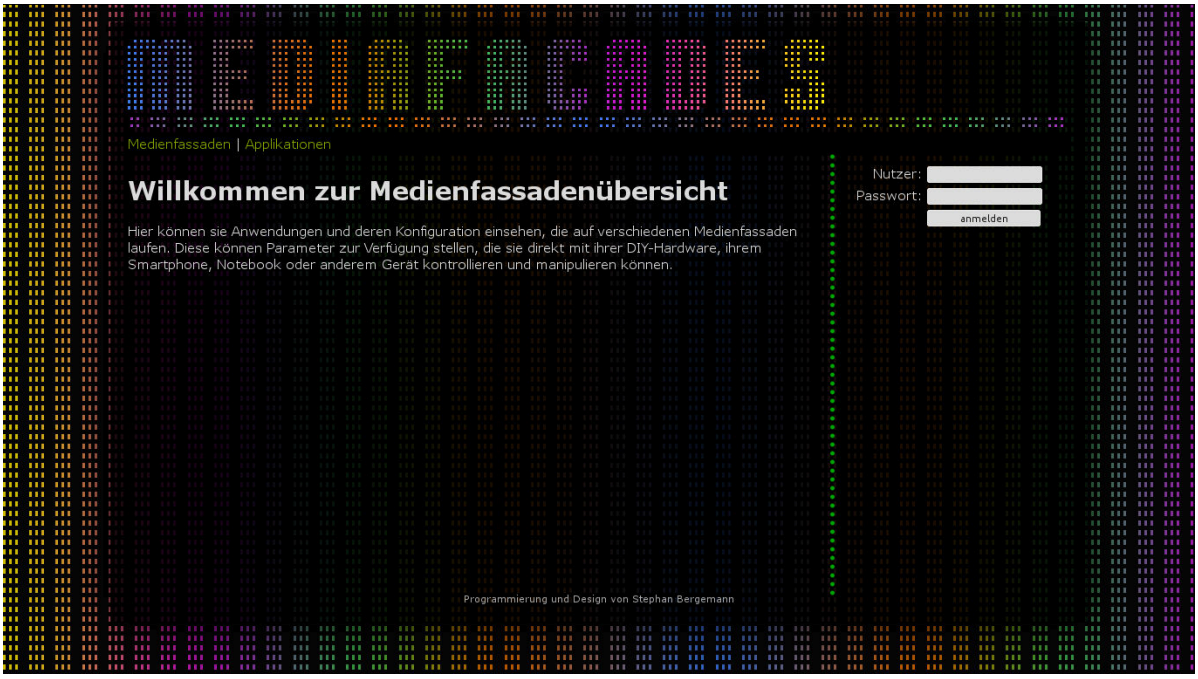


Abbildung A.3.: Startseite

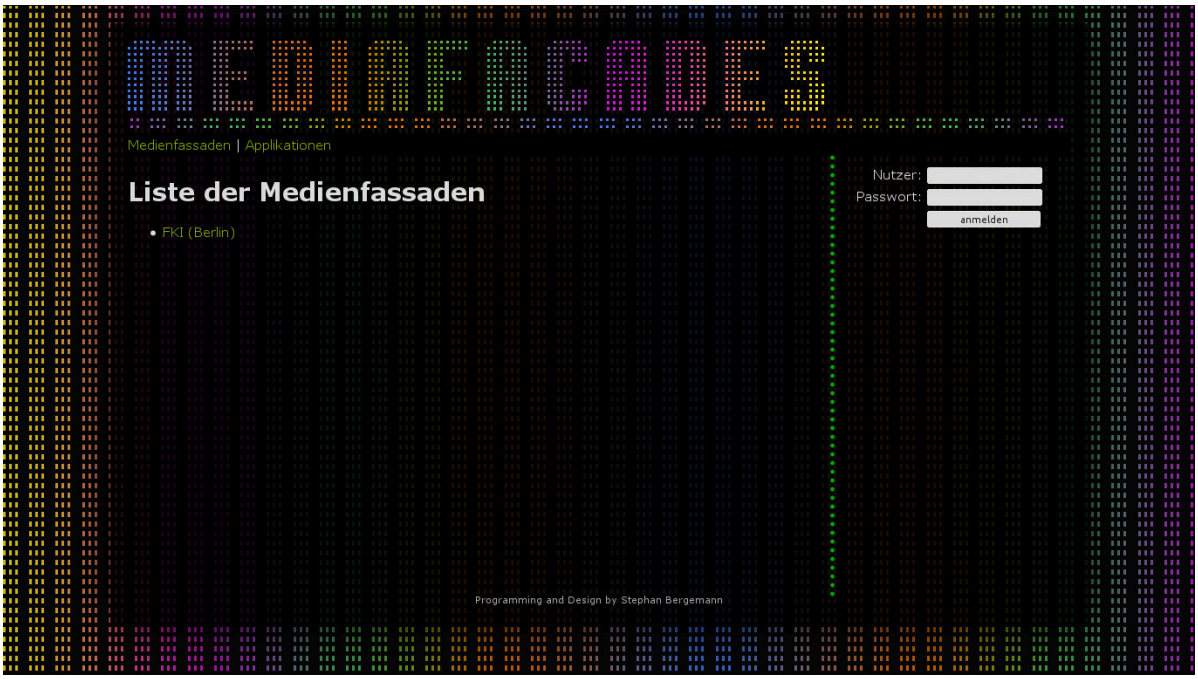


Abbildung A.4.: Liste der Medienfassaden

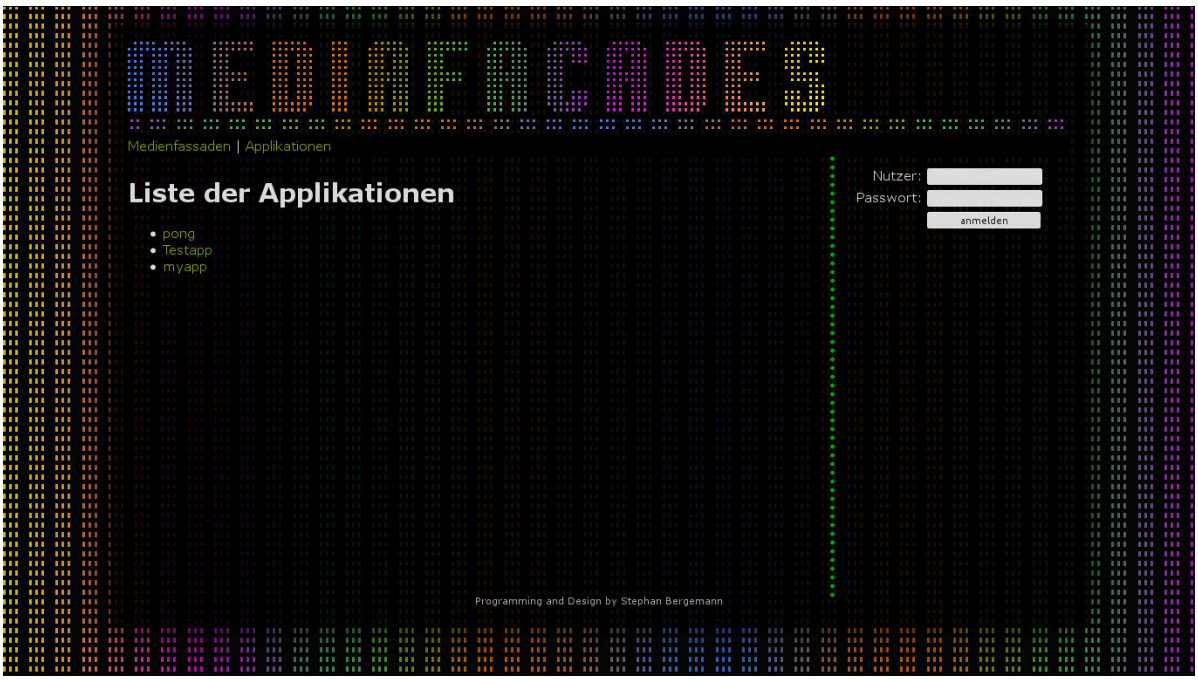


Abbildung A.5.: Liste der Applikationen

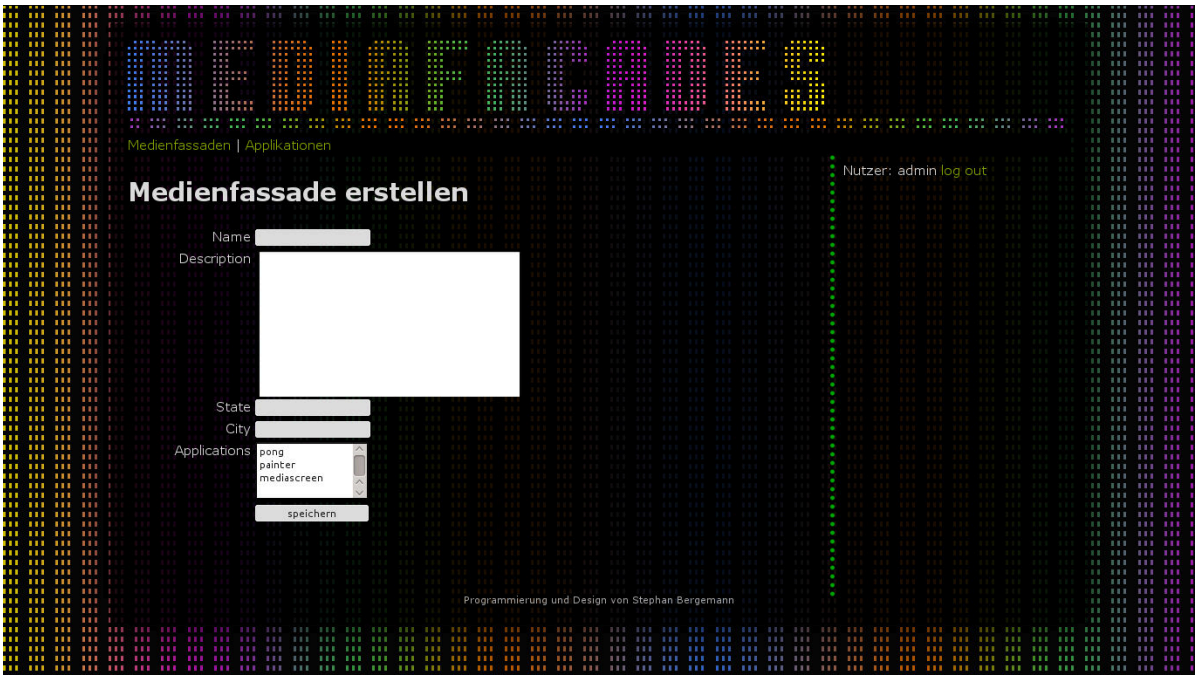


Abbildung A.6.: Formular zum Erstellen einer Medienfassaden

Abbildung A.7.: Formular zum Erstellen einer Applikation

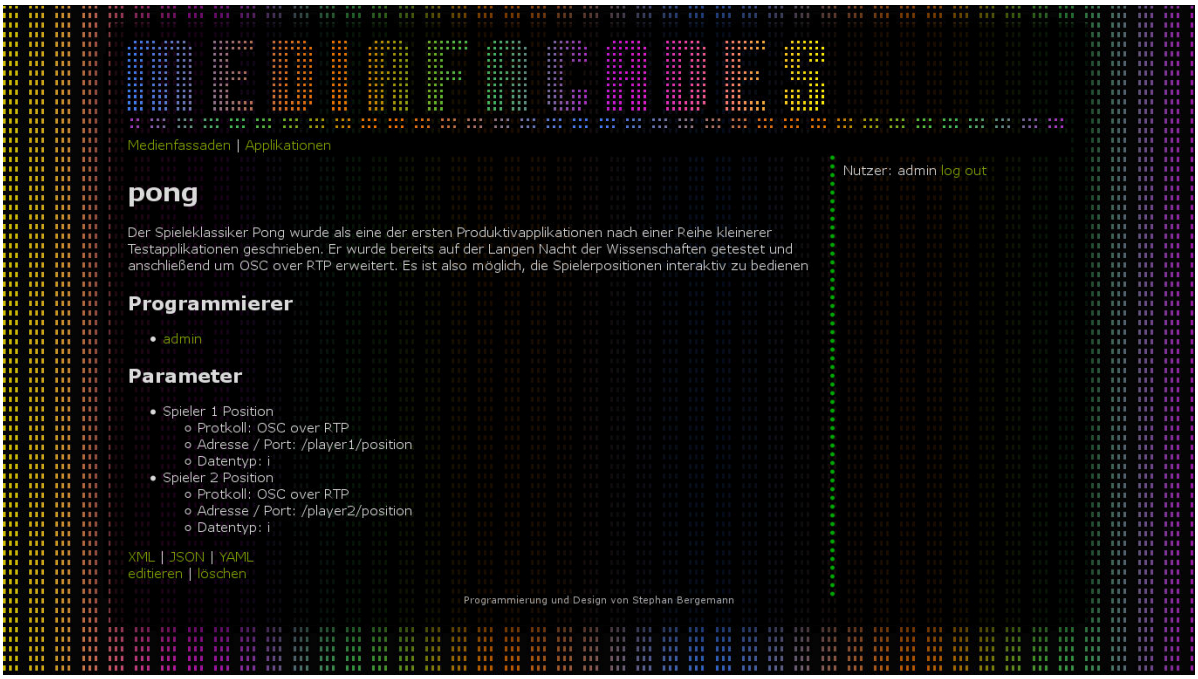


Abbildung A.8.: Übersicht zur Applikation Pong

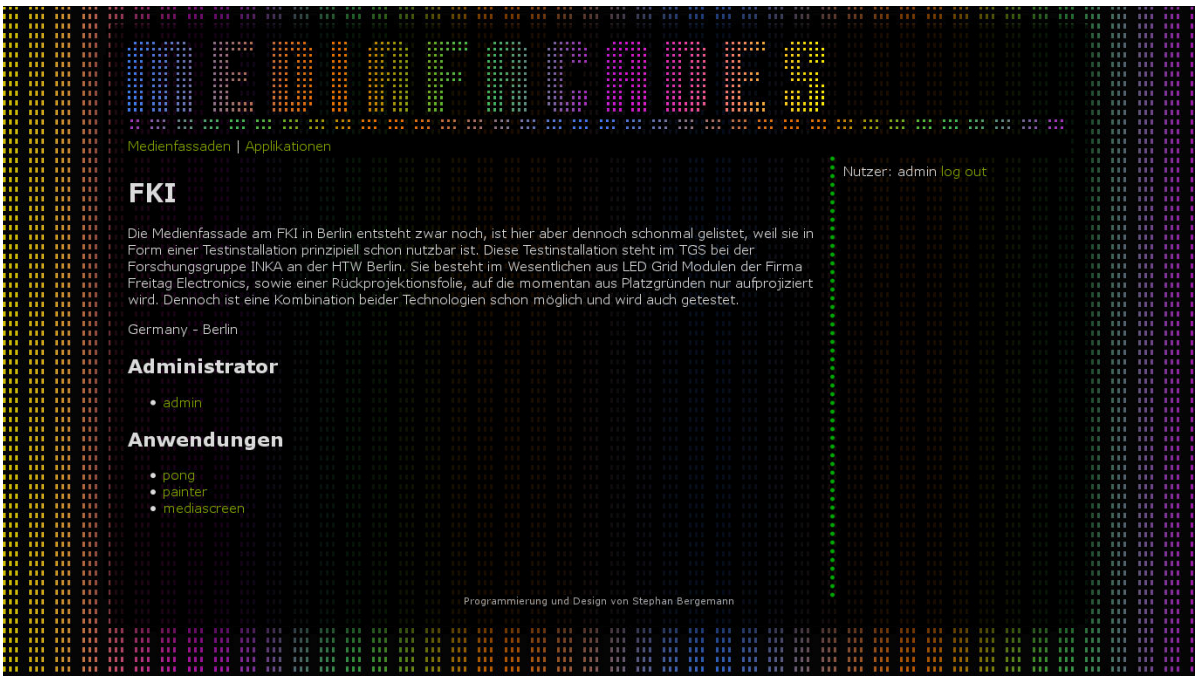


Abbildung A.9.: Übersicht zur Applikation Pong

A.2. Beispielanwendungen

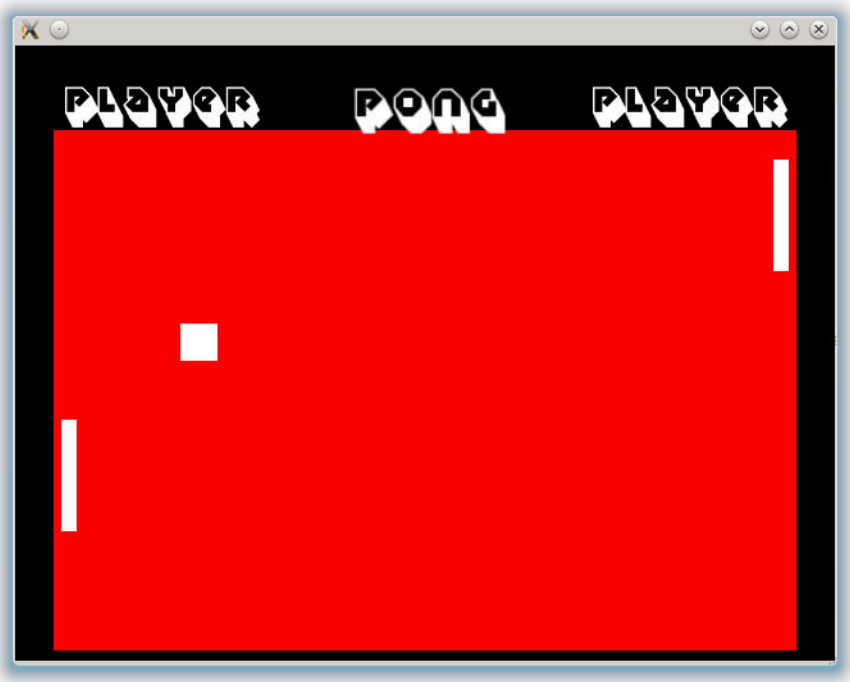


Abbildung A.10.: Screenshot von Pong

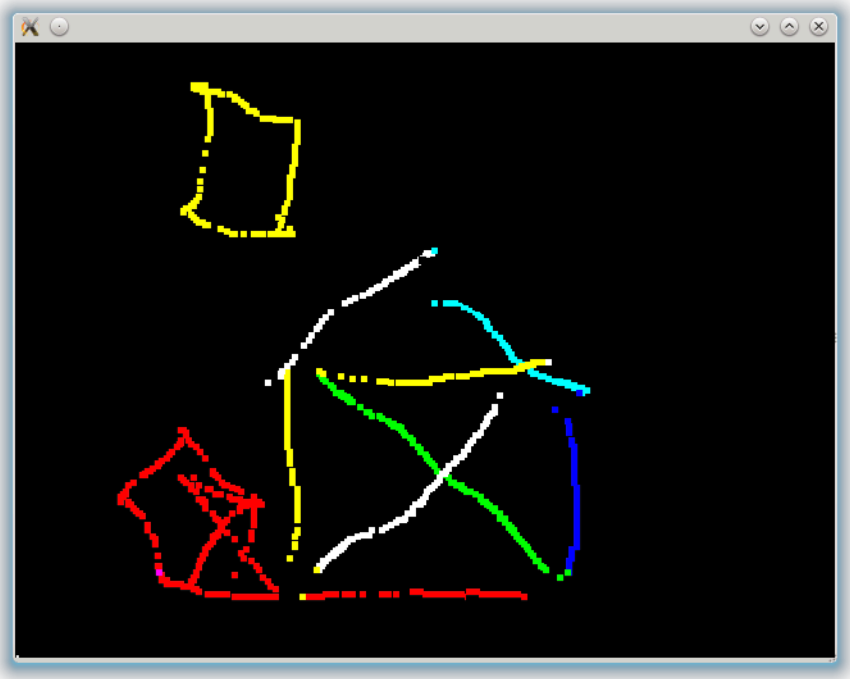


Abbildung A.11.: Screenshot der Anwendung zum kollaborativen Malen

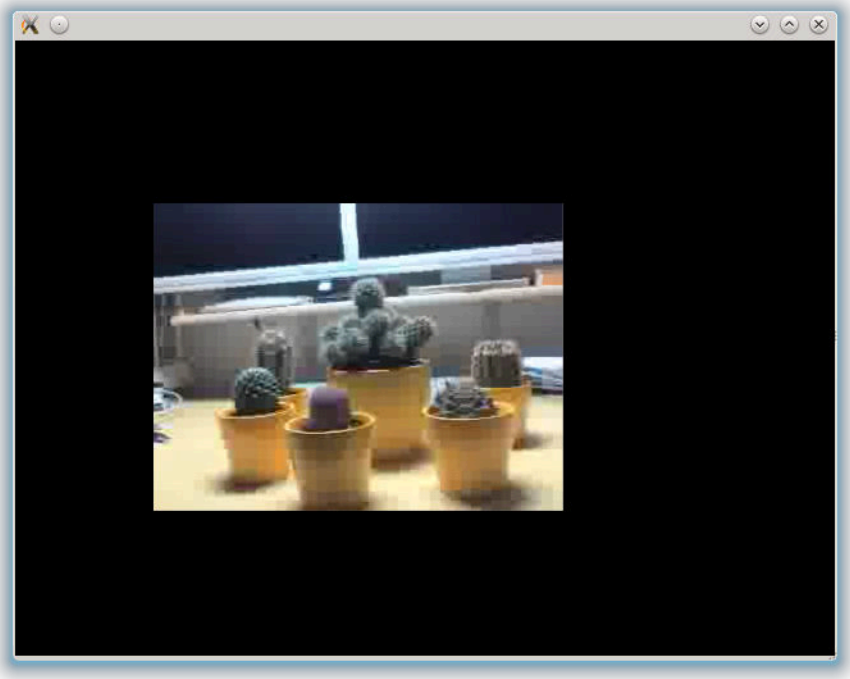


Abbildung A.12.: Screenshot der Mediascreen Anwendung

A.3. weitere Quelltexte

A.3.1. Website

Außer im oben gezeigten JSON-Format können die Parameter von Applikationen auch noch in YAML und XML exportiert werden. Folgend werden die von Django generierten Ausgaben der Parameter für die Ponanwendung in den Formaten YAML, XML und JSON dargestellt.

```
- fields: {address: /player1/position, application: 1, datatype: i, name: Spieler
    1 Position}
  model: mediafacades.parameter
  pk: 1
- fields: {address: /player2/position, application: 1, datatype: i, name: Spieler
    2 Position}
  model: mediafacades.parameter
  pk: 2
```

Quelltext A.1: Parameterexport von Pong im YAML-Format

```
<?xml version="1.0" encoding="utf-8"?>
<django-objects version="1.0">
   <object pk="1" model="mediafacades.parameter">
      <field to="mediafacades.application" name="application" rel="ManyToOneRel">1</field>
      <field type="CharField" name="name">Spieler 1 Position</field>
      <field type="CharField" name="address">/player1/position</field>
      <field type="CharField" name="datatype">i</field>
   </object>
   <object pk="2" model="mediafacades.parameter">
```

```
10        <field to="mediafacades.application" name="
              application" rel="ManyToOneRel">1</field>
11        <field type="CharField" name="name">Spieler 2
              Position</field>
12        <field type="CharField" name="address">/player2/
              position</field>
13        <field type="CharField" name="datatype">i</field>
14    </object>
15 </django-objects>
```

Quelltext A.2: Parameterexport von Pong im XML-Format

```
1  [{
2     "pk": 1,
3     "model": "mediafacades.parameter",
4     "fields":{
5        "datatype": "i",
6        "application": 1,
7        "name": "Spieler␣1␣Position",
8        "address": "/player1/position"
9     }
10 },{
11    "pk": 2,
12    "model": "mediafacades.parameter",
13    "fields": {
14       "datatype": "i",
15       "application": 1,
16       "name": "Spieler␣2␣Position",
17       "address": "/player2/position"
18    }
19 }]
```

Quelltext A.3: Parameterexport von Pong im JSON-Format

Literatur

[Ack06] Norbert Ackermann. *Lichttechnik: Systeme der Bühnen- und Studiobeleuchtung rationell planen und projektieren*. Oldenbourg, 2006. ISBN: 9783486270426.

[Ash+99] Gal Ashour u. a. *Universal Serial Bus Device Class Definition for MIDI Devices*. 1999.

[Bat+11] Scott Bateman u. a. „Target assistance for subtly balancing competitive play“. In: *Proceedings of the 2011 annual conference on Human factors in computing systems*. CHI '11. Vancouver, BC, Canada: ACM, 2011, S. 2355–2364. ISBN: 978-1-4503-0228-9. DOI: 10.1145/1978942.1979287. URL: http://doi.acm.org/10.1145/1978942.1979287.

[BEN09] Oren Ben-Kiki, Clark Evans und Ingy döt Net. *YAML Ain't Markup Language (YAML)*. 2009. URL: http://yaml.org/spec/1.2/spec.pdf (besucht am 22. 07. 2012).

[Ber10] Stephan Bergemann. „Besucherinteraktion auf Veranstaltungen mit der OpenBeacon-Technologie“. Bachelorarbeit. Hochschule für Technik und Wirtschaft Berlin (HTW), 2010.

[BL] Hans Georg Britz und Franz Martin Löhle. *MIDI Kompendium*. URL: http://www.zem-college.de/midi/index.htm (besucht am 27. 07. 2012).

[bli] blinkenlights.net. *Project Blinkenlights*. URL: http://blinkenlights.net (besucht am 15. 07. 2012).

[Bor+11] Sebastian Boring u. a. „Multi-user interaction on media facades through live video on mobile devices“. In: *Proceedings of the 2011 annual conference on Human factors in computing systems*. CHI '11. Vancouver, BC, Canada: ACM, 2011, S. 2721–2724. ISBN: 978-1-4503-0228-9. DOI: 10.1145/1978942.1979342. URL: http://doi.acm.org/10.1145/1978942.1979342.

[BS12] Stephan Bergemann und Robin Schlegel. *Medienfassade - Möglichkeiten von Visualisierung und Interaktion*. 2012. URL: http://inka.htw-berlin.de/Sieck/Stud_Projekte/bergemann_schlegel_fp2.pdf (besucht am 16. 07. 2012).

[Cro06] Douglas Crockford. *RFC 4627 - The application/json Media Type for JavaScript Object Notation (JSON)*. 2006. URL: http://tools.ietf.org/html/rfc4627 (besucht am 22. 07. 2012).

[FH12] Patrick Tobias Fischer und Eva Hornecker. „Urban HCI: spatial aspects in the design of shared encounters for media facades". In: *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*. CHI '12. Austin, Texas, USA: ACM, 2012, S. 307–316. ISBN: 978-1-4503-1015-4. DOI: 10.1145/2207676.2207719. URL: http://doi.acm.org/10.1145/2207676.2207719.

[FS09] Adrian Freed und Andy Schmeder. „Features and Future of Open Sound Control version 1.1 for NIME". In: *NIME*. 2009. URL: http://cnmat.berkeley.edu/node/7002.

[FZH10] Patrick Tobias Fischer, Christian Zöllner und Eva Hornecker. „VR/Urban: Spread.gun - design process and challenges in developing a shared encounter for media façades". In: *Proceedings of the 24th BCS Interaction Specialist Group Conference*. BCS '10. Dundee, United Kingdom: British Computer Society, 2010, S. 289–298. ISBN: 978-1-78017-130-2. URL: http://dl.acm.org/citation.cfm?id=2146303.2146346.

[Häu09] M. Hank. Häusler. *Media facades : history, technology, content*. English. avedition, [Ludwigsburg, Germany], 2009. ISBN: 9783899861075 3899861078.

[LW01] John Lazzaro und John Wawrzynek. „A case for network musical performance". In: *Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*. NOSSDAV '01. Port Jefferson, New York, United States: ACM, 2001, S. 157–166. ISBN: 1-58113-370-7. DOI: 10.1145/378344.378367. URL: http://doi.acm.org/10.1145/378344.378367.

[LW11] J. Lazzaro und J. Wawrzynek. *RTP Payload Format for MIDI*. RFC 6295 (Proposed Standard). Internet Engineering Task Force, Juni 2011. URL: http://www.ietf.org/rfc/rfc6295.txt.

[OGR08] Kenton O'Hara, Maxine Glancy und Simon Robertshaw. „Understanding collective play in an urban screen game". In: *Proceedings of the 2008 ACM conference on Computer supported cooperative work*. CSCW '08. San Diego, CA, USA: ACM, 2008, S. 67–76. ISBN: 978-1-60558-007-4. DOI: 10.1145/1460563.1460576. URL: http://doi.acm.org/10.1145/1460563.1460576.

[Rom88] J.L. Romkey. *Nonstandard for transmission of IP datagrams over serial lines: SLIP*. RFC 1055 (Standard). Internet Engineering Task Force, Juni 1988. URL: http://www.ietf.org/rfc/rfc1055.txt.

[SC03] H. Schulzrinne und S. Casner. *RTP Profile for Audio and Video Conferences with Minimal Control*. RFC 3551 (Standard). Updated by RFC 5761. Internet Engineering Task Force, 2003. URL: http://www.ietf.org/rfc/rfc3551.txt.

[Sch+03] H. Schulzrinne u. a. *RTP: A Transport Protocol for Real-Time Applications*. RFC 3550 (Standard). Updated by RFC 5506, 5761, 6051, 6222. Internet Engineering Task Force, 2003. URL: http://www.ietf.org/rfc/rfc3550.txt.

[SFW10] Andrew Schmeder, Adrian Freed und David Wessel. „Best Practices for Open Sound Control". In: *Linux Audio Conference*. Utrecht, NL, 2010.

[Sim] Simeon Simeonov. *WDDX: Distributed Data for the Web*. URL: http://www.infoloom.com/gcaconfs/WEB/chicago98/simeonov.HTM (besucht am 20. 07. 2012).

[Str00] Bjarne Stroustrup. *The C++ Programming Language*. 3rd. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000. ISBN: 0201700735.

[The] TheGreenEyl. *Aperture*. URL: http://www.thegreeneyl.com/aperture (besucht am 15. 07. 2012).

[VB05] Daniel Vogel und Ravin Balakrishnan. „Distant freehand pointing and clicking on very large, high resolution displays". In: *Proceedings of the 18th annual ACM symposium on User interface software and technology*. UIST '05. Seattle, WA, USA: ACM, 2005, S. 33–42. ISBN: 1-59593-271-2. DOI: 10.1145/1095034.1095041. URL: http://doi.acm.org/10.1145/1095034.1095041.

[W3C] W3C. *Extensible Markup Language (XML)*. URL: http://www.w3.org/XML/ (besucht am 22. 07. 2012).

[WF97] Matthew Wright und Adrian Freed. „Open Sound Control: A New Protocol for Communicating with Sound Synthesizers“. In: *International Computer Music Conference*. Thessaloniki, Hellas: International Computer Music Association, 1997, S. 101–104. URL: http://cnmat.berkeley.edu/publications/open_sound_control_new_protocol_communicating_sound_synthesizers.

[Wri] Matt Wright. *The Open Sound Control 1.0 Specification*. URL: http://opensoundcontrol.org/spec-1_0 (besucht am 26. 07. 2012).

[Yat+05] Koji Yatani u. a. „Toss-it: intuitive information transfer techniques for mobile devices“. In: *CHI '05 extended abstracts on Human factors in computing systems*. CHI EA '05. Portland, OR, USA: ACM, 2005, S. 1881–1884. ISBN: 1-59593-002-7. DOI: 10.1145/1056808.1057046. URL: http://doi.acm.org/10.1145/1056808.1057046.

[ZJC11] P. Zimmermann, A. Johnston und J. Callas. *ZRTP: Media Path Key Agreement for Unicast Secure RTP*. RFC 6189 (Informal). Internet Engineering Task Force, 2011. URL: http://www.ietf.org/rfc/rfc6189.txt.

Bilderquellen

[Bie] Alian Bieber. *VR/urban: SMSlingshot*. URL: http://rebelart.net/vrurban-smslingshot/005846/ (besucht am 28. 08. 2012).

[FL] Nicolas Ferrando und Lois Lammerhuber. *Ars Electronica Center*. URL: http://www.aec.at/press/2011/09/27/lange-nacht-der-museen-im-ars-electronica-center/ (besucht am 28. 08. 2012).

[GH] Dorit Günter und Nadja Hannaske. *Blinkenlights*. URL: http://blinkenlights.net/blinkenlights (besucht am 28. 08. 2012).

[Oha] Tomio Ohashi. *Tower of Winds, Japan*. URL: http://www.architecture.com/Awards/RoyalGoldMedal/RoyalGoldMedal2006/TowerofWinds.aspx (besucht am 28. 08. 2012).

[Sug] Masanori Sugimoto. *Intuitive Information Transfer with Gestural Input*. URL: http://www.k.u-tokyo.ac.jp/pros-e/person/masanori_sugimoto/masanori_sugimoto.htm (besucht am 28. 08. 2012).

[wik] wikipedia. *EIA-232-Datenrahmen*. URL: http://upload.wikimedia.org/wikipedia/commons/d/de/RS-232_timing.png (besucht am 06. 06. 2012).

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis

Abkürzungsverzeichnis

EMIPLIB EDM Media over IP library. 22, 23, 51–54, 79

FKI Forschungs- und Weiterbildungszentrum für Kultur und Informatik. 2, 32, 52, 78, 79

LSB Least Siginificant Bit. 8

MIDI Musical Instrument Digital Interface. 9–13, 15, 16, 20, 26, 27, 53, 67, 68, 98

MSB Most Siginificant Bit. 10

NFC Near Field Communication. 48

OSC Open Sound Control. 12–18, 26, 42, 43, 45–48, 50–54, 56, 57, 59, 60, 62, 64, 65, 67, 68, 74–77, 79–83, 101

RFC Request for Comments. 21, 51

RTCP RTP Control Protocol. 19, 20, 22

RTP Real-Time Transport Protocol. 18–23, 26, 27, 42, 43, 45–47, 52–54, 60, 62–64, 68, 76, 79–83

SDL Simple DirectMedia Layer. 32, 46, 51

TCP Transmission Control Protocol. 13, 71

UDP User Datagram Protocol. 13, 17, 18, 20

VoIP Voice over IP. 21

Eigenständigkeitserklärung

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

Ort, Datum	Unterschrift