Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

# Bachelorarbeit

**Martin Hansen**

## Positionierung von Klangquellen einer Wellenfeldsynthese-Anlage mit Hilfe eines Audio-Plugins

*Fakultät Technik und Informatik*
*Studiendepartment Informatik*

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

Martin Hansen

# Positionierung von Klangquellen einer Wellenfeldsynthese-Anlage mit Hilfe eines Audio-Plugins

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

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

Betreuender Prüfer: Prof. Dr. Wolfgang Fohl
Zweitgutachter: Prof. Dr.-Ing. Andreas Meisel

Eingereicht am: 27. Juni 2014

**Martin Hansen**

**Thema der Arbeit**
Positionierung von Klangquellen einer Wellenfeldsynthese-Anlage mit Hilfe eines Audio-Plugins

**Stichworte**
Wellenfeldsynthese, Surround Sound, Audio-Plugin, VST, VST3, Audio Units, RTAS, OSC, Open Sound Control

**Kurzzusammenfassung**
Diese Arbeit beschreibt die Entwicklung eines Audio-Plugins zur Steuerung von virtuellen Klangquellen einer auf der Software WONDER basierenden Wellenfeldsynthese-Anlage. Darüber hinaus wird ein Überblick über verschiedene Software-Systeme und Audio-Plugins für die Wellfenfeldsynthese gegeben. Abschließend wird eine Reihe von Vorschlägen unterbreitet, wie das entwickelte Plugin weiter verbessert werden könnte und wie die in der Arbeit gewonnenen Erkenntnisse auch zur Weiterentwicklung anderer Komponenten des WONDER-Software-Systems genutzt werden könnten.

**Martin Hansen**

**Title of the paper**
Positioning of sound sources of a wave field system by the use of an audio-plugin

**Keywords**
Wave Field Synthesis, Surround Sound, Audio Plugin, VST, VST3, Audio Units, RTAS, OSC, Open Sound Control

**Abstract**
This thesis describes the development of an audio plugin that controls virtual sound sources of a wave field synthesis system that is based on the software WONDER. In addition an overview of various software systems and audio plugins for wave field synthesis is given. Concluding, several suggestions are made how the developed plugin could be improved further and how the findings made in this thesis could be used for further development of the WONDER software system.

# Inhaltsverzeichnis

# 1 Einleitung

Im Bereich der Video-Produktion und -Wiedergabe wurden in den vergangenen Jahren erfolgreich neue Verfahren zur räumlichen Darstellung von Inhalten zur Marktreife gebracht: „3D“-Kinos finden sich heute überall auf der Welt, entsprechende Fernsehgeräte werden von allen großen Herstellern vermarktet.

Im Audio-Bereich wird der Markt sowohl in Kinos als auch bei den Geräten für den Endkunden von sogenannten „Surround Sound“-Technologien beherrscht, die bereits in den 1980er Jahren etabliert wurden – obwohl diese Technologien deutliche Unzulänglichkeiten vorweisen [Mel10, S. 53]. Mit der Wellenfeldsynthese existiert ein Verfahren zur räumlichen Darstellung von Klangquellen, das der zur Zeit den Markt dominierenden Technik in vielen Aspekten weit überlegen ist. Dennoch findet dieses Verfahren in der Praxis zur Zeit noch kaum Anwendung.

Die Geschichte der „3D“-Filme und der entsprechenden Wiedergabetechnologien zeigt, dass es zur erfolgreichen Verbreitung derartiger neuer Technologien auch entsprechende Inhalte braucht, um die Vorzüge des neuen Verfahrens zur Geltung zu bringen. Um solche Inhalte erstellen zu können, braucht es wiederum entsprechende Werkzeuge.

Aus dieser Erkenntnis heraus soll diese Arbeit einen Beitrag dazu leisten, die Produktion von Audio-Inhalten für die Wellenfeldsynthese zu vereinfachen.

## 1.1 Thema dieser Arbeit

Im Rahmen dieser Bachelorarbeit wird eine Software-Erweiterung für bestehende Audio-Programme, ein sogenanntes Audio-Plugin, entwickelt. Dieses ermöglicht es, die virtuellen Klangquellen einer Wellenfeldsynthese-Anlage aus etablierten Audio-Produktions-Umgebungen heraus zu kontrollieren. Die Steuerung der virtuellen Klangquellen wird dadurch in eine Arbeitsumgebung integriert, wie sie weltweit von Tontechnikern zur Erstellung und Bearbeitung von Audio-Inhalten genutzt wird. Das entwickelte Plugin verbindet somit Wellenfeldsynthese-Anlage und Audio-Software zu einem System, das erfahrene Anwender der genutzten Audio-Software in die Lage versetzt, Inhalte für die Wellenfeldsynthese mit einem Werkzeug zu erstellen, dessen Bedienung sie bereits gewohnt sind.

Vorrangiges Ziel der Entwicklung ist der Einsatz des Plugins im Wellenfeldsynthese-Labor der HAW Hamburg. Die Software wird jedoch so konzipiert, dass sie auch auf anderen Wellenfeldsynthese-Anlagen zum Einsatz kommen könnte.

## 1.2 Ausgangssituation im Labor der HAW

An der HAW Hamburg ist seit 2011 eine Wellenfeldsynthese-Anlage in einem Laborraum installiert [Foh13]. Dort wird die Wellenfeldsynthese auf Basis der Open-Source-Software WONDER [Baa08, S. 50 ff.] realisiert. Die Erstellung und Wiedergabe von Inhalten für Präsentationen dieser Anlage gestaltete sich bisher relativ aufwändig. Um dort zum Beispiel ein Video zu präsentieren und gleichzeitig die entsprechenden Audiosignale mittels Wellenfeldsynthese im Raum verteilt wiederzugeben, mussten bisher drei getrennte Systeme synchron gestartet werden:

1. Ein Rechner spielt das Video auf der Powerwall – einer aus sechs einzelnen HD-Displays zusammengesetzten Video-Wiedergabefläche, ab.

2. Auf einem weiteren Rechner spielt eine Audio-Software (wahlweise Cubase oder Ardour) die einzelnen Tonspuren ab.

3. Parallel dazu läuft ein Skript in der Skript-Sprache SuperCollider[1], das die Positionierung der einzelnen Klangquellen über den zeitlichen Verlauf steuert, indem es entsprechende Nachrichten an das WONDER-System sendet.

Ein derartiges SuperCollider-Skript beinhaltet für jede Änderung am Zustand der Wellenfeldsynthese-Anlage mindestens eine Zeile Programmcode. Das Erstellen eines solchen Skriptes stellt bereits für studierte Informatiker eine mühsame Aufgabe dar – einem Tontechniker hingegen ist diese Arbeit kaum zuzumuten.

Ziel dieser Bachelorarbeit ist daher, ein Werkzeug zum Erstellen, Bearbeiten und Wiedergeben von Audio-Inhalten für das WONDER-System zu schaffen, dessen Nutzung keine Informatik-Fachkenntnisse erfordert.

## 1.3 Aufbau

In dieser Arbeit werden zunächst die fachlichen Grundlagen geschildert, auf denen diese Arbeit aufbaut. Erläutert werden dabei unter anderem das Verfahren der Wellenfeldsynthese,

[1] http://supercollider.sourceforge.net - Abruf: 2014-06-20

das Konzept der „Digital Audio Workstation“-Programme und die für diese existierenden Programmier-Schnittstellen sowie der Aufbau des Wellenfeldsynthese-Labors an der HAW und die dort genutzte Software WONDER. Zudem wird ein Überblick über verschiedene weitere Software-Systeme für die Wellenfeldsynthese und bereits existierende Audio-Plugins zur Steuerung solcher Systeme gegeben.

Daraufhin wird, ausgehend von einer genaueren Schilderung der Anforderungen an die im Rahmen dieser Arbeit entwickelten Software, der gesamte Entwicklungsprozess des entstandenen Audio-Plugins erläutert.

Abschließend werden das Ergebnis der Arbeit und die gewonnenen Erkenntnisse zusammengefasst sowie eine Reihe von Vorschlägen unterbreitet, wie das entwickelte Plugin weiter verbessert werden könnte und wie die in der Arbeit gewonnen Erkenntnisse auch über das Plugin hinaus genutzt werden könnten.

# 2 Grundlagen

In diesem Kapitel werden zunächst verschiedene Verfahren zur räumlichen Darstellung von Klangquellen und die Besonderheiten der Wellenfeldsynthese gegenüber anderen Verfahren geschildert. Daran anschließend wird erläutert, welche Software-Werkzeuge heutzutage für die Erstellung von Audio-Inhalten genutzt werden und welche Schnittstellen existieren, um diese Werkzeuge um zusätzliche Funktionalitäten erweitern zu können. Daraufhin folgt ein Überblick über verschiedene Software-Systeme für die Wellenfeldsynthese, bevor schließlich das Wellenfeldsynthese-Labor an der HAW sowie das dort verwendete Software-System genauer erklärt werden.

Dieses Kapitel erhebt dabei keinen Anspruch auf Vollständigkeit. An vielen Stellen werden nur diejenigen Aspekte behandelt, die für diese Arbeit relevant sind.

## 2.1 Verfahren zur räumlichen Darstellung von Klangquellen

Neben den „Surround Sound“-Verfahren, die die meisten Menschen heutzutage aus dem Kino kennen, existieren verschiedene weitere Verfahren zur räumlichen Darstellung von Klangquellen. Im Folgenden soll ein kurzer Überblick über die verschiedenen Technologien gegeben und erläutert werden, wie sich die einzelnen Verfahren unterscheiden.

### 2.1.1 Mehrkanal-Stereofonie

Das am weitesten verbreitete Verfahren zur räumlichen Darstellung von Klangquellen ist die Mehrkanal-Stereofonie. Bei diesem Verfahren werden mehrere Lautsprecher um den Hörer herum positioniert. Um eine räumliche Ortung zu ermöglichen, werden die Audiosignale auf einen oder mehrere dieser Lautsprecher verteilt und ihre Amplituden für die einzelnen Laustprecherkanäle entsprechend gewichtet.

Übliche Lautsprecher-Aufstellungen für dieses Verfahren umfassen heute fünf oder sieben separate Laustprecherkanäle. Diese werden durch einen separaten Kanal für tieffrequente Signale, die für die räumliche Ortung nicht relevant sind, ergänzt, weshalb man von „5.1“- oder „7.1 Surround Sound“ spricht.

Mehrkanal-Stereofonie kommt heutzutage nicht nur weltweit in Kinos zum Einsatz, es existieren auch entsprechende Systeme für Endkunden, welche die Nutzung des Verfahrens im privaten Haushalt ermöglichen. Über digitale Medien wie die DVD und ihre Nachfolge-Formate sowie über digitale Fernsehausstrahlung wird zudem eine große Vielfalt entsprechend produzierter audiovisueller Inhalte angeboten.

Die realitätsnahe räumliche Darstellung von Klangquellen ist bei diesem Verfahren allerdings auf verschiedene Weise begrenzt. So ist die optimale räumliche Wahrnehmung der reproduzierten Klangquellen ausschließlich im Zentrum der Lautsprecher-Aufstellung, dem sogenannten Sweet Spot, möglich. Zudem ist es notwendig, dass die Lautsprecher der Wiedergabesysteme exakt so positioniert sind, wie es für den jeweils wiedergegebenen Inhalt vorgegeben ist. Um dieser Anforderung gerecht zu werden, existieren präzise Empfehlungen für einheitliche Lautsprecher-Aufstellungen [itu12]. Darüber hinaus sind Klangquellen, die zwischen zwei Lautsprechern positioniert sind, für den Hörer deutlich schlechter zu lokalisieren als solche, die aus der Richtung eines Lautsprechers kommen. Bei den üblichen Wiedergabesystemen zur Wiedergabe von audiovisuellen Inhalten wird diesem Problem begegnet, indem Lautsprecher im Bereich der visuellen Wiedergabefläche (Leinwand oder Bildschirm) deutlich dichter beieinander positioniert werden. Die Lautsprecher seitlich und rückwärtig zum Zuschauer müssen dann mit entsprechend größeren Abständen aufgestellt werden, wodurch die Ortung von Klangquellen in diesem Bereich zusätzlich erschwert wird.

### 2.1.2 Ambisonics

Eine Alternative zur Mehrkanal-Stereofonie bietet Ambisonics. Ambisonics ist ein zweiteiliger Ansatz zur Kodierung von Schall-Amplitude und Schall-Richtung und deren Wiedergabe im dreidimensionalen Raum [MM95]. Dabei werden Audiosignale in dem sogenannten B-Format kodiert. Bei Ambisonics erster Ordnung beinhaltet dieses Format vier Kanäle: ein Kanal (W) beinhaltet die Schalldruck-Komponente, drei Kanäle (X, Y, Z) beinhalten die Schallschnelle-Komponenten für drei orthogonale Richtungen. Dieses Kodierungsverfahren entspricht der Aufnahme mit einem speziellen Mikrofon, dem „Soundfield-Mikrofon“, das die Schalldruck-Komponente mit kugelförmiger Richtcharakteristik aufnimmt, während die drei Schallschnelle-Komponenten mit entsprechend ausgerichteten Mikrofon-Kapseln mit achtförmiger Richtcharakteristik aufgenommen werden. Für eine bessere räumliche Auflösung wird bei Ambisonics höherer Ordnung der Schallschnelle-Anteil in eine höhere Anzahl von Einzelkomponenten unterteilt, wodurch das B-Format dann entsprechend mehr Kanäle umfasst.

Aus dem B-Format lassen sich die Wiedergabesignale für verschiedene Lautsprecher-Aufstellungen dekodieren, sodass Ambisonics bezüglich möglicher Lautsprecher-Konstellationen

deutlich flexibler ist als die Mehrkanal-Stereofonie. Dabei sind die Wiedergabemöglichkeiten dank des Z-Kanals nicht auf die horizontale Ebene beschränkt. Ebenso wie bei der Mehrkanal-Stereofonie ist die optimale räumliche Ortung der Klangquellen aber auch bei Ambisonics auf einen zentralen „Sweet Spot“ beschränkt.

### 2.1.3 Wellenfeldsynthese

Mit der Wellenfeldsynthese (im Folgenden auch kurz WFS genannt) existiert ein weiteres Verfahren zur räumlichen Darstellung von Klangquellen, das im Gegensatz zu den bisher genannten Verfahren die realistische Wahrnehmung der Klangquellen nicht auf einen „Sweet Spot“ beschränkt.

#### Verfahren

Das Verfahren der Wellenfeldsynthese nutzt ein physikalisches Phänomen, das als Huygenssche Prinzip oder Huygens-Prinzip bekannt ist. Es besagt, dass jeder Punkt einer Wellenfront als Ausgangspunkt einer neuen Welle betrachtet werden kann, die sich phasengleich zur ursprünglichen Welle mit derselben Wellenlänge ausbreitet. Eine solche Welle wird Elementarwelle genannt. Durch Überlagerung aller Elementarwellen entsteht wiederum die ursprüngliche Wellenfront. Dieses Prinzip wurde bereits im 17. Jahrhundert durch den niederländischen Astronom, Mathematiker und Physiker Huygens entdeckt. Es dauerte aber noch bis 1988, bis Berkhout ein Konzept vorstellte, um auf Basis dieses Prinzips Töne wiederzugeben und dadurch eine realitätsnahe räumliche Darstellung dieser Töne zu ermöglichen [Ber88].

Bei der Wellenfeldsynthese nutzt man eine Vielzahl dicht beieinander liegender, einzeln ansteuerbarer Lautsprecher, die jeweils Elementar-Schallwellen erzeugen. Mit diesen Elementarwellen können Wellenfronten rekonstruiert werden, die sich so ausbreiten, als würden sie von einer hinter den Lautsprechern befindlichen tatsächlichen Klangquelle ausgestrahlt werden. Bei den so dargestellten imaginären Klangquellen spricht man von virtuellen Ton- oder Klangquellen (im Folgenden oft auch kurz „virtuelle Quellen“ genannt).

Abbildung 2.1 zeigt anhand der Ausbreitung einer einzelnen Schallwelle, wie dieses Prinzip funktioniert: eine virtuelle Klangquelle erzeugt eine virtuelle Schallwelle, die sich kreisförmig ausbreitet (a), bis sie von hinten auf eine Lautsprecher-Wand trifft (b). Jeder Lautsprecher der Lautsprecherwand erzeugt nun die entsprechende Elementarwelle (c). Durch Überlagerung der Elementarwellen setzt sich jenseits der Lautsprecherwand wiederum eine kreisförmige Wellenfront zusammen, die der ursprünglichen Schallwelle entspricht (e-f).

Die durch Überlagerung der Elementarwellen synthetisierten Schallwellen breiten sich also kreisförmig so aus, als wären sie tatsächlich im Mittelpunkt dieses Kreises entstanden. Dadurch

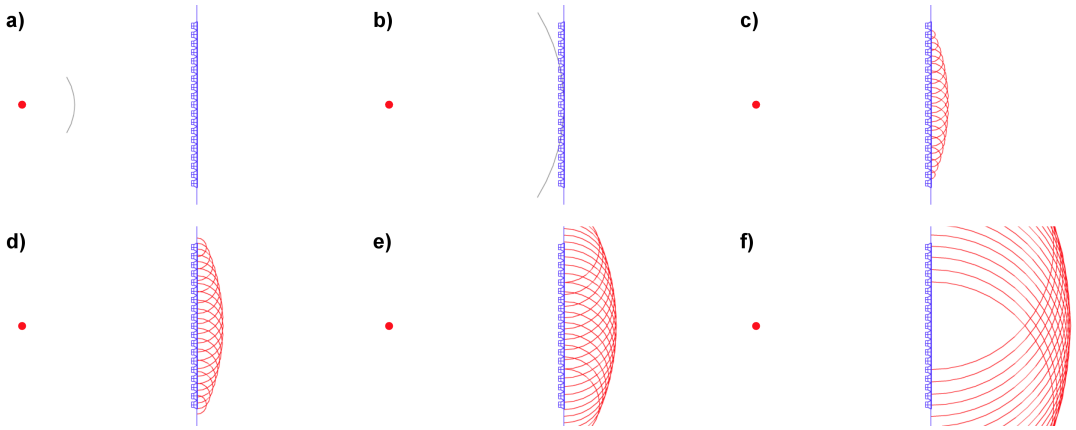


Abbildung 2.1: Ablauf der Synthese einer Schallwelle [Oel13]

lassen sie sich von jeder Hörerposition innerhalb des Abstrahlbereichs der Lautsprecherwand gleich gut räumlich orten: der Hörer bekommt an jeder Position den Eindruck, die Quelle der Schallwelle läge tatsächlich im Zentrum des von der Wellenfront geformten Kreises. Im Gegensatz zu den anderen erwähnten Verfahren ist die räumliche Wahrnehmung bei der Wellenfeldsynthese also nicht auf einen „Sweet Spot“ begrenzt.

Die obige Veranschaulichung reduziert das Verfahren auf zwei Dimensionen: reale Schallwellen breiten sich jedoch nicht kreis- sondern kugelförmig aus. Theoretisch ließe sich das Verfahren entsprechend auf die dritte Dimension erweitern. Es entspricht allerdings der heutigen Praxis bei den meisten Systemen zur Wellenfeldsynthese, die dritte Dimension außer Acht zu lassen und sich auf die horizontale Ebene zu beschränken. Dies liegt vor allem an dem hohen Rechen- und Hardware-Aufwand. Für jede virtuelle Quelle muss für eine Vielzahl von Lautsprechern berechnet werden, mit welcher Verzögerung und welcher Amplitude der jeweilige Lautsprecher die entsprechende Elementarwelle erzeugen soll. Je nach Größe des genutzten Raumes erfordert das Verfahren eine große Zahl separat ansteuerbarer Lautsprecher: bereits bei Systemen zur zweidimensionalen Wellenfeldsynthese kommen oft hunderte von separaten Wiedergabekanälen zum Einsatz.

### Unterscheidung von Quell-Typen

Neben der vor allem kostenbedingten Einschränkung auf zwei Dimensionen hat das Verfahren noch eine weitere Grenze: das oben dargestellte Prinzip lässt sich nur auf virtuelle Klangquellen anwenden, die hinter den Lautsprechern liegen. Klangquellen, die vor den Lautsprechern positioniert sind, lassen sich so nicht darstellen.

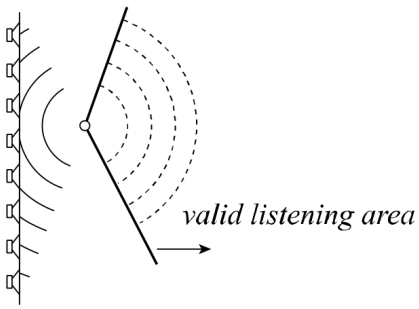


Abbildung 2.2: Fokussierte Quelle und der Bereich, in dem eine räumliche Ortung dieser Quelle möglich ist [Mon11, S. 16]

Diesem Problem wird begegnet, indem man zur Darstellung dieser Quellen die Lautsprecher so ansteuert, dass sich an der Position der Quelle durch Überschneidung der dort eintreffenden Schallwellen eine entsprechende kreisförmige Welle zusammensetzt. Dieses Verfahren ermöglicht eine realistische Darstellung innerhalb des Bereichs, der von den Lautsprechern aus betrachtet jenseits der virtuellen Klangquelle liegt. Für Hörer, die zwischen den Lautsprechern und der virtuellen Quelle stehen, ist eine realistische räumliche Wahrnehmung dieser Quellen nicht möglich. Abbildung 2.2 zeigt exemplarisch eine derartige, üblicherweise als „fokussierte Quelle“ bezeichnete Quelle und den Bereich, in dem eine räumliche Ortung der Quelle möglich ist.

Die meisten WFS-Systeme unterstützen neben diesen fokussierten Quellen und den regulären, hinter den Lautsprechern gelegenen Quellen („Punktquellen“) noch eine dritte Art virtueller Klangquellen, die sogenannten Linearquellen. Bei diesen breiten sich die Schallwellen nicht kreisförmig aus, sondern verlaufen als geradlinige Wellenfronten. Diese Quellen dienen dazu, Klangquellen darzustellen, die in sehr großer Entfernung hinter den Lautsprechern liegen.

Die drei genannten Typen virtueller Klangquellen, die in WFS-Systemen üblicherweise genutzt werden, sind in Abbildung 2.3 dargestellt.

## 2.2 Objektbasierte Audioformate

Für Mehrkanal-Stereofonie und Ambisonics existieren anerkannte Dateiformat-Standards, in denen vorproduzierte Audio-Inhalte für die Wiedergabe mit dem jeweiligen Verfahren gespeichert werden können. Für die Wellenfeldsynthese wurde bis heute kein entsprechender Standard etabliert. In diesem Zusammenhang wird seit nunmehr über zehn Jahren das Kon-

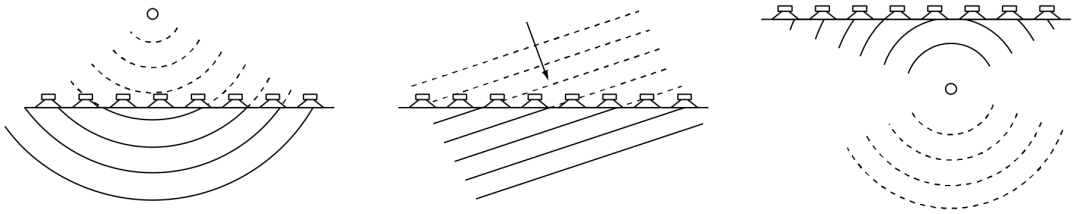

Abbildung 2.3: Unterschiedliche Typen virtueller Quellen eines WFS-Systems: Punktquelle, Linearquelle und fokussierte Quelle (von links nach rechts) [Mon11, S. 13]

zept des objektbasierten Audioformats diskutiert [MRB+03] [PK04], das an dieser Stelle kurz erläutert werden soll.

Bei der Produktion für Mehrkanal-Stereofonie ist es nach wie vor üblich, dass am Ende eines Produktionsvorgangs das Produkt in ein Datei-Format exportiert wird, bei dem die Audiodaten jedes Wiedergabekanals in einem separaten Daten-Strom gespeichert werden. Bei der Wiedergabe können dann die einzelnen Audio-Daten-Ströme direkt auf den entsprechenden Lautsprechern wiedergegeben werden. Man spricht dabei auch von kanal-basierten Audioformaten. Bei Verfahren mit nur wenigen separaten Laustprecherkanälen ist dies eine naheliegende, speichereffiziente Methode.

Bei der Wellenfeldsynthese ist diese Methode nicht mehr effizient anwendbar. Zum einen arbeitet man hier oft mit hunderten von separaten Laustprecherkanälen, was einen sehr hohen Speicheraufwand erfordern würde. Zum anderen wäre es nicht sinnvoll, ein Wellenfeldsynthese-spezifisches Dateiformat mit einer festen Anzahl von Laustprecherkanälen zu definieren, da die benötigte Anzahl an Laustprecherkanälen wesentlich von den Abmessungen des zu beschallenden Raumes abhängt.

Das Konzept des objektbasierten Audioformats sieht als Gegenentwurf vor, dass die Audiosignale während des Produktionsvorgangs als Audio-Objekte räumlich verteilt werden. Am Ende des Produktionsvorgangs werden diese dann in ein Format exportiert, bei dem die Audiodaten jedes Audio-Objektes mit ihren Raumverteilungs-Daten verknüpft separat gespeichert werden. Dieser Ansatz hat nicht nur den Vorteil, dass er speichereffizienter als der kanalbasierte Ansatz ist, sobald die Anzahl der Laustprecherkanäle die der Audio-Objekte übersteigt. Er trennt auch das Speicher-Format von dem genutzten Wiedergabeverfahren: auf Basis objekt-basierter Daten kann die Wiedergabe sowohl mittels Wellenfeldsynthese als auch mittels eines anderen Verfahrens erfolgen. Zudem ist auf Basis der objektbasierten Daten ein Export in jegliche kanal-basierte Formate weiterhin möglich.

| *Datenfeld* | *Speicherformat* |
|---|---|
| Positionsdaten folgen | 1 Bit |
| Azimutwinkel: 0° bis 360° | 8 Bit |
| Elevationswinkel: -90° bis 90° | 6 Bit |
| Abstand: 0m bis 14,5m (Maximalwert entspricht endlosem Abstand) | 5 Bit |
| Audioobjekt beinhaltet ein ambientes Signal | 1 Bit |
| Audioobjekt beinhaltet eine Mischung | 1 Bit |
| Audioobjekt gehört zu einem visuellen Objekt | 1 Bit |
| Audioobjekt ist ein Sprachsignal | 1 Bit |
| Bewegungen des Audioobjektes interpolieren | 1 Bit |

Tabelle 2.1: Vorschlag von Melchior für ein objektbasiertes Audioformat [MS11]

Trotz der offensichtlichen Vorteile eines solchen Formats konnte bisher noch kein allgemeiner Standard für ein objektbasiertes Audioformat entwickelt werden. Melchior machte 2010 einen Vorschlag, wie ein derartiges Format aussehen könnte – Tabelle 2.1 zeigt eine von ihm zur Diskussion gestellte Auflistung von zu speichernden Positions- und Metadaten [MS11]. Im April 2012 stellte die Firma Dolby als Teil ihrer „Dolby Atmos“-Technologie einen hybriden Ansatz vor. Hierbei können sowohl separate Audio-Objekte mit ihren entsprechenden Metadaten als auch kanal-basierte Audio-Spuren gemeinsam in eine Datei eingebettet werden. Mit entsprechender Dekoder-Hardware können auf Basis dieses Formates die einzelnen Ausgabe-Kanäle für verschiedene Lautsprecherkonstellationen berechnet werden [dol14, S. 6 ff.].

## 2.3 Audio-Produktionsumgebungen

Für die Erstellung und Bearbeitung von Audio-Inhalten gibt es eine große Vielfalt an Software. Im Folgenden soll die Gruppe der sogenannten „Digital Audio Workstation“-Programme vorgestellt werden. Sodann soll ein Überblick über die etablierten Schnittstellen gegeben werden, mit denen diese Programme um zusätzliche Funktionalitäten erweitert werden können.

### 2.3.1 Digital Audio Workstations

Unter der Bezeichnung „Digital Audio Workstations“ (im folgenden auch kurz DAW genannt) werden heute Programme zusammengefasst, die dafür konzipiert sind, sämtliche Vorgänge der Audio-Produktion innerhalb einer Software zu realisieren. Sie bieten die Möglichkeit, Audiosignale aufzuzeichnen und zu bearbeiten.

Dafür werden unter anderem verschiedenste Werkzeuge zur Signalverarbeitung bereitgehalten, die oft klassischen Effektgeräten aus dem Bereich der analogen Musikproduktion nachempfunden sind. Auch Änderungen von Parametern dieser „virtuellen Effekte“ oder der Lautstärke einzelner Signale können in ihrem zeitlichen Verlauf aufgezeichnet und bearbeitet werden – man spricht hierbei von „Automation“. Darüber hinaus stellen die meisten DAWs Werkzeuge zur Komposition sowie verschiedene virtuelle Klangerzeuger („virtuelle Instrumente“) zur Verfügung. Die einzelnen Audiosignale können schließlich zusammengemischt und als fertiges Produkt in verschiedene Ausgabeformate exportiert werden. Dabei unterstützen viele moderne DAW-Programme auch Mehrkanal-Stereofonie-Formate wie 5.1 und bieten die Möglichkeit, einzelne Audiosignale gemäß diesem Verfahren räumlich zu positionieren [BBB+12, S. 303 ff.][App13, S. 730 ff.][Avi14c, S. 1065 ff.]. Die Verarbeitung sämtlicher Audiosignale sowie der Nutzereingaben erfolgt in DAWs in Echtzeit.

Die Benutzerschnittstelle dieser Programme umfasst in der Regel ein Fenster, in dem die Inhalte des zu bearbeitenden Projekts auf einer Zeitachse dargestellt sind. Dort können die Inhalte in ihrem zeitlichen Verlauf bearbeitet werden. So ist es auch möglich, den Verlauf der Parameteränderungen zu manipulieren. Während die Zeitachse üblicherweise horizontal verläuft, ist das Fenster meist vertikal in sogenannte Spuren unterteilt, welche die verschiedenen Audiosignale darstellen. Ergänzt wird dieses Fenster üblicherweise unter anderem durch eine weitere Darstellung, die einem herkömmlichen analogen Mischpult nachempfunden ist und entsprechende Bedienfunktionen bieten soll. Abbildung 2.4 zeigt diese Fenster am Beispiel des Programms Cubase 7 LE.

Weit verbreitete kommerzielle DAW-Programme sind unter anderem das Programm Pro Tools[1] des Herstellers Avid, Logic Pro[2] (hergestellt von Apple) sowie Cubase[3] und Nuendo[4], die beide von der Firma Steinberg entwickelt werden. Eine beliebte Open-Source-Alternative ist Ardour[5]. Diese Aufzählung ist keineswegs vollständig. Die genannten kommerziellen Produkte heben sich aber dadurch hervor, dass ihre Hersteller Schnittstellen etabliert haben, mittels derer ihre Programme durch zusätzliche Software erweitert werden können.

### 2.3.2 Audio-Plugins

Die meisten DAW-Programme können durch sogenannte Audio-Plugins um zusätzliche Möglichkeiten zur Signalverarbeitung erweitert werden.

[1] http://www.avid.com/de/products/pro-tools-software -Abruf: 2014-06-20
[2] http://www.apple.com/de/logic-pro - Abruf: 2014-06-20
[3] http://www.steinberg.de/de/products/cubase/start.html - Abruf: 2014-06-20
[4] http://www.steinberg.net/de/products/nuendo.html - Abruf: 2014-06-20
[5] http://ardour.org - Abruf: 2014-06-20


Abbildung 2.4: Screenshot der DAW-Software Cubase 7 LE

### Allgemeines Prinzip

Audio-Plugins können zur Laufzeit in die DAW-Software geladen und bei Bedarf auch zur Laufzeit wieder entfernt werden. Aufgrund dieser Gastgeber-Rolle der DAW-Software spricht man auch von der DAW als „Host“. Üblicherweise ist ein geladenes Plugin einer konkreten Audio-Spur zugeordnet und verarbeitet dann die Audiosignale dieser Spur.

Um es anderen Anbietern zu ermöglichen, derartige Plugins für ihre jeweiligen Produkte zu entwickeln, haben einige Hersteller entsprechende Software-Schnittstellen definiert und bieten passende Software Development Kits (SDKs) an. Die Schnittstellen arbeiten dabei alle nach einem ähnlichen Prinzip: mittels einer zentralen Methode, die meist Namen wie *process* oder *render* trägt, wird dem Plugin ein Block von Audio-Samples übergeben. Das Plugin berechnet auf Basis dieser Samples dann einen entsprechenden Block von bearbeiteten Samples und schreibt diese in einen Speicherbereich, der ihm dafür ebenfalls vom Host übergeben wurde. Diese Methode wird vom Host wiederholt aufgerufen, um den jeweils aktuellen Block von Audiodaten zu bearbeiten. Diese Bearbeitung soll dabei in Echtzeit erfolgen. Bei der Entwicklung eines Audio-Plugins ist daher darauf zu achten, dass ein Aufruf der Methode schnellstmöglich abgearbeitet wird und auf keinen Fall blockieren kann.

Ergänzend zu der Übergabe von zu bearbeitenden Audio-Daten definieren die Plugin-Schnittstellen Möglichkeiten, dem Host mitzuteilen, wie viele und welche Parameter das

Plugin bei der Signalverarbeitung nutzt. Mittels entsprechender Methoden können dann einerseits Parameteränderungen, die in der Benutzeroberfläche des Plugins ausgelöst wurden, an den Host gemeldet werden. Andererseits kann der Host seinerseits dem Plugin mitteilen, wie die derzeitigen Werte dieser Parameter zu setzen sind. Auf diesem Weg können auch die zeitlichen Verläufe der Parameter eines solchen Plugins mittels der Automations-Möglichkeiten der Host-DAW aufgezeichnet und bearbeitet werden.

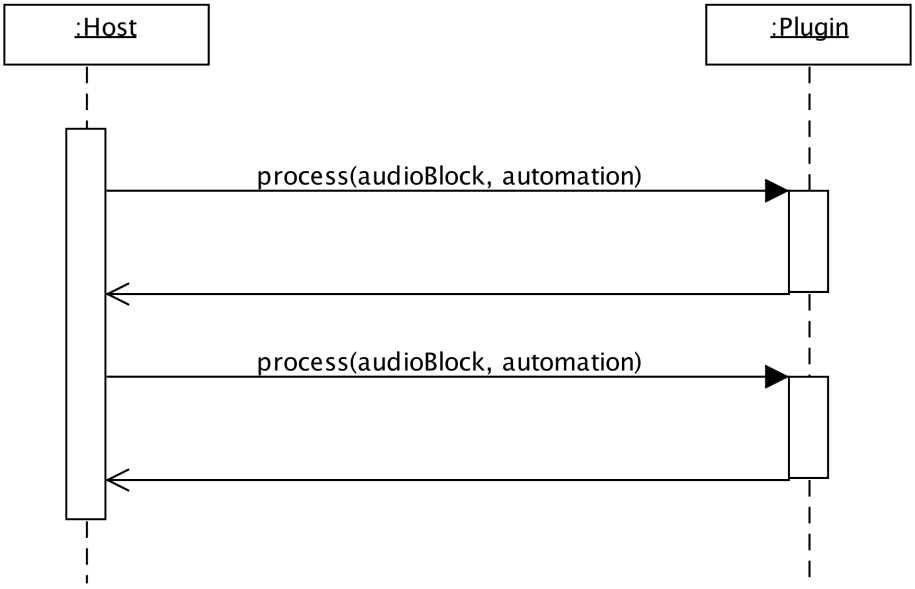


Abbildung 2.5: Aufrufe eines Audio-Plugins durch den Host

Abbildung 2.5 zeigt beispielhaft die wiederholten Aufrufe der *process*-Methode eines Audio-Plugins durch den Host. In diesem Beispiel werden die Automations-Daten der Parameter jeweils mit den entsprechenden Audio-Daten in einem Aufruf übergeben. Bei einigen Schnittstellen erfolgt die Übergabe der Parameter allerdings durch Aufrufe separater Methoden.

Zusätzlich zu den Funktionen zur Signalverarbeitung und der Parameter-Automation bieten die verschiedenen Plugin-Schnittstellen üblicherweise die Möglichkeit, Zustände des Plugins in einem vom Host bereitgestellten Speicherbereich zu speichern und bei Bedarf wieder herzustellen. Dies ermöglicht es zum einen dem Nutzer, häufig genutzte Einstellungen des Plugins als sogenannte Presets für verschiedene Projekte abzurufen, ohne die entsprechenden Einstellungen erneut vornehmen zu müssen. Zum anderen können auf diesem Wege die jeweils aktuellen Einstellungen eines Plugins gemeinsam mit dem gesamten in der DAW bearbeiteten Projekt gespeichert werden.

### Konkrete Schnittstellen

Von verschiedenen Herstellern wurde eine Reihe von Schnittstellen für solche Plugins etabliert, die allesamt für die Software-Entwicklung in C oder C++ ausgelegt sind:

- *Steinberg – VST:* VST (Virtual Studio Technology) ist eine Schnittstelle der Firma Steinberg[6]. Das dazugehörige SDK ist nach einem einfachen Registrierungsvorgang für jeden Entwickler kostenlos erhältlich. Es ermöglicht nicht nur die Entwicklung von VST-Plugins, vielmehr können auch eigene VST-Host-Programme entwickelt werden [ste13]. Dank dieser Möglichkeit sowie der Unterstützung mehrerer Betriebssysteme (Microsoft Windows und Apple OSX) ist VST die Plugin-Schnittstelle, für die die größte Auswahl an Host-Programmen zur Verfügung steht.

  VST liegt aktuell in der Version 3.6.0 vor. Mit dem Sprung auf Version 3.0.0 im Jahr 2008 wurde die VST-Schnittstelle so grundlegend geändert, dass die Versionen ab 3.0.0 nicht zu den vorherigen Versionen kompatibel sind. Daher wird allgemein zwischen den Standards VST3 und VST2 unterschieden. Da viele Hosts VST3 nach wie vor nicht unterstützen, werden auch heute noch neue Plugins für den VST2-Standard entwickelt.

- *Avid – RTAS und AAX:* RTAS (Real-Time AudioSuite) ist ein Standard, der von der Firma Avid[7] beziehungsweise ihrer Vorgänger-Firma Digidesign entwickelt wurde. Im Vergleich zu den Lizenzbestimmungen für das VST-SDK sind die entsprechenden Bestimmungen der Firma Avid deutlich restriktiver: nur ausgewählte Entwickler erhalten nach einer Bewerbung Zugang zu dem SDK, das grundsätzlich nicht für akademische Zwecke zur Verfügung gestellt wird [avi14a]. Avid erlaubt auch denjenigen Entwicklern, denen Zugang zu dem SDK gewährt wird, lediglich die Entwicklung neuer Plugins. Somit beschränkt sich die Auswahl der möglichen Hosts auf die Produkte der Firma Avid. Dies sind die DAW-Software Pro Tools sowie die Videobearbeitungs-Software Media Composer[8].

  Mit Pro Tools Version 10 wurde von Avid das Format AAX als Nachfolger von RTAS eingeführt. Seit Pro Tools Version 11 wird RTAS nicht mehr von Pro Tools unterstützt [Avi14b, S. 2].

- *Apple – Audio Units:* Die Firma Apple[9] hat in ihre Betriebssysteme OSX und iOS unter dem Namen Core Audio eine Infrastruktur für digitale Audio-Verarbeitung integriert

[6] http://www.steinberg.de - Abruf: 2014-06-20
[7] http://www.avid.com - Abruf: 2014-06-20
[8] http://www.avid.com/DE/products/family/Media-Composer - Abruf: 2014-06-20
[9] http://www.apple.com - Abruf: 2014-06-20

[App14]. Neben einer Abstraktions-Schicht für die Audio-Hardware beinhaltet Core Audio unter anderem auch eine Plugin-Schnittstelle namens Audio Units (oft kurz AU genannt). Unter dem Betriebssystem iOS, das für den Betrieb auf mobilen Geräten konzipiert ist, dient diese Schnittstelle lediglich der Nutzung der Audio Ein- und Ausgabe sowie einiger vorgegebener Funktionalitäten zur digitalen Signalverarbeitung. Unter OSX dagegen bietet Apple mit den Audio Units eine vollwertige Schnittstelle zur Entwicklung eigener Audio-Plugins [App14, S. 49]. Das entsprechende SDK ist nach einer einfachen Registrierung jedem Entwickler zugängig. Es ermöglicht sowohl die Entwicklung von Audio-Units-Plugins als auch entsprechender Host-Applikationen. Daher existieren neben den Apple-Produkten Logic Pro und GarageBand[10] eine Vielzahl weiterer Host-Programme, welche die Einbindung von Audio-Units-Plugins unterstützen. Hierzu gehört unter anderem die OSX-Version von Ardour.

- *Open Source – LADSPA und LV2:* Für das Linux-Betriebssystem wurde im Jahr 2000 der Standard LADSPA[11] als quelloffene Schnittstelle für Audio-Plugins entworfen [Fur00]. Die Definition der Schnittstelle besteht im Wesentlichen aus einer c-Header-Datei. Deren Verwendung ist nicht auf Linux beschränkt, sodass LADSPA auch auf anderen Betriebssystemen genutzt werden kann. Im Gegensatz zu den anderen hier genannten Plugin-Standards enthält LADSPA keine Schnittstelle zum Erstellen einer eigenen, Plugin-spezifischen Benutzeroberfläche. Mit LV2[12] existiert ein neuerer Standard, der als Nachfolger von LAPSDA konzipiert wurde und dieses sowie weitere Defizite des LADSPA-Standards beheben soll.

Viele der genannten Schnittstellen bieten Möglichkeiten, die über das erläuterte Grundkonzept der Audio-Plugins hinaus gehen. Üblich ist es zum Beispiel, neben Plugins zur Signalverarbeitung („virtuelle Effektgeräte“) auch Plugins entwickeln zu können, die als eigenständige Klangerzeuger dienen („virtuelle Instrumente“). An dieser Stelle sollen jedoch nur diejenigen Funktionalitäten geschildert werden, die für diese Arbeit relevant sind.

## 2.4 Verfügbare Wellenfeldsynthese-Systeme

In der jüngeren Vergangenheit wurden an einer Vielzahl von Forschungsinstituten Experimente zur Wellenfeldsynthese durchgeführt, wodurch eine schwer überschaubare Anzahl an entsprechenden forschungsorientierten Software-Systemen entstanden ist. Der folgende Abschnitt

[10] http://www.apple.com/de/mac/garageband - Abruf: 2014-06-20
[11] http://www.ladspa.org - Abruf: Abruf: 2014-06-20
[12] http://lv2plug.in - Abruf: 2014-06-20

kann daher keine vollständige Aufzählung aller WFS-Systeme bieten und beschränkt sich auf diejenigen Systeme, die aktuell als kommerzielle Produkte oder als OpenSource-Projekte verfügbar sind.

### 2.4.1 Kommerzielle Systeme

Zur Zeit werden drei WFS-Systeme für den kommerziellen Markt angeboten.

#### IOSONO

Die IOSONO GmbH aus Erfurt ist hervorgegangen aus dem Fraunhofer-Institut für Digitale Medientechnologie IDMT in Ilmenau. Sie bietet Systeme für die Wellenfeldsynthese an, die unter anderem in verschiedenen Kinos installiert wurden [Mel10].

Die Installationen bestehen dabei aus einem Verteilten System: ein Rechner (der „Cinema Server“) stellt die Audiodaten sowie die entsprechenden Positionsdaten bereit, ein Cluster von mehreren Rechnern ist für das Rendering zuständig, und ein einzelner Rechner steuert das System und bietet eine Nutzerschnittstelle an [Mel10, S. 55 f.].

#### Fraunhofer IDMT

Das Fraunhofer-Institut für Digitale Medientechnologie selbst vermarktet unter dem Namen „SpatialSound Wave“ ebenfalls ein WFS-System. Das System ist ausgelegt für Installationen mit bis zu 64 separaten Laustprecherkanälen. Das Rendering erfolgt dabei zentral auf einem Linux-basierten Rechner. Eine entsprechende Anlage befindet sich unter anderem im Planetarium Hamburg [fra14].

#### Sonic Emotion

Die Züricher Firma Sonic Emotion bietet ein Hardware-Gerät namens „Sonic Wave I“ an. Damit lassen sich bis zu 24 Audio-Eingänge als virtuelle Klangquellen auf bis zu 64 Laustprecherkanäle verteilen.

Das Angebot wird ergänzt durch die Programme „WaveDesigner“ zur Konfiguration der Lautsprecher-Aufstellung sowie „WavePerformer“ zur räumlichen Verteilung der virtuellen Klangquellen [Son13].

### 2.4.2 Open-Source-Systeme

Aus verschiedenen Forschungsprojekten sind WFS-Systeme entstanden, die als Open-Source-Projekte veröffentlicht wurden.

### WFS Designer

An der University Of Miami wurde von Montag das Programm „WFS Designer[13]“ entwickelt [Mon11]. Dieses Programm läuft zentral auf einem Rechner, wodurch die Rechenleistung des Systems entsprechend begrenzt ist.

Genaue Daten über die Leistungsfähigkeit der Software liegen nicht vor. Konzipiert wurde „WFS Designer“ für Experimente, bei denen fünf virtuelle Quellen auf 48 Lautsprecher verteilt werden [Mon11, S.44].

Da ein Schwerpunkt von Montags Forschungsarbeiten die Entwicklung eines dreidimensionalen WFS-Systems war, bietet die Nutzerschnittstelle des Programms die Möglichkeit, virtuelle Quellen dreidimensional zu positionieren und verschiedene (dreidimensionale) Lautsprecheraufstellungen zu konfigurieren. Eine Steuerung der Quellen-Positionen über den zeitlichen Verlauf ist nicht vorgesehen.

Das Programm ist plattformunabhängig in C++ programmiert.

### SoundScape Renderer

Die Telekom Innovation Laboratories, ein Forschungsinstitut der Deutschen Telekom, entwickelten in ihrem an der Technischen Universität Berlin angesiedelten Labor ein System namens „Soundscape Renderer[14]“ [GAS08], das inzwischen am Institut für Nachrichtentechnik an der Universität Rostock weiter entwickelt wird [GS13].

Das modular aufgebaute Software-System ist dafür ausgelegt, unterschiedliche Methoden der räumlichen Darstellung von Klangquellen in Echtzeit zu realisieren: die Nutzerschnittstelle ermöglicht die zweidimensionale Positionierung von Klangquellen im Raum unabhängig von dem zu nutzenden Darstellungsverfahren. Die verschiedenen Verfahren sind als Rendering-Komponenten mit einer gemeinsamen Software-Schnittstelle implementiert und können somit unabhängig von der Nutzerschnittstelle ausgetauscht werden.

Eine der entwickelten Rendering-Komponenten nutzt dabei das Verfahren der Wellenfeldsynthese.

Die SoundScape Renderer Software ist als Linux-basiertes verteiltes System konzipiert. Jedoch erfolgt auch hier das Audio-Rendering auf nur einem zentralen Rechner; eine Verteilung der Rechenlast für das Audio-Rendering ist nicht möglich.

[13] Quellcode: https://github.com/mmontag/WFS-Designer - Abruf: 2014-06-20
[14] Quellcode: https://github.com/SoundScapeRenderer/ssr - Abruf: 2014-06-20

#### WONDER

WONDER[15] (früher auch sWONDER genannt) ist ein System für die Wellenfeldsynthese, das an der Technischen Universität Berlin entwickelt wurde [Baa08, S. 50 ff.]. Es wurde als verteiltes System konzipiert, bei dem nicht nur unterschiedliche Aufgaben auf mehrere Rechner verteilbar sind, sondern auch die Rechenlast des Renderings auf mehrere Rechner aufgeteilt werden kann.

Damit ist WONDER unter den nicht-kommerziellen WFS-Systemen das leistungsfähigste. Die größte WONDER-basierte Installation befindet sich in einem Hörsaal der TU Berlin, wo 832 separate Lautsprecherkanäle über ein Cluster von 15 Rechnern angesteuert werden.

Die WONDER-Software ist in C++ für den Betrieb auf Linux-Rechnern entwickelt, kann aber auch unter OSX betrieben werden. Da auch das Wellenfeldsynthese-Labor an der HAW Hamburg auf WONDER basiert, wird das System im Verlauf dieser Arbeit noch genauer betrachtet werden.

## 2.5 Audio-Plugins für die Wellenfeldsynthese

Es existieren bereits einige Varianten von Audio-Plugins, die bestehende WFS-Systeme ergänzen oder (in Kombination mit der jeweiligen Host-Software) ein eigenständiges WFS-System darstellen sollen.

### 2.5.1 IOSONO S.A.W.

Die bereits erwähnte Firma IOSONO bietet ein Plugin namens „Spatial Audio Workstation“ (im weiteren kurz S.A.W.) an [Mel10].

Das Plugin, das aktuell in Version 2.2 erhältlich ist, ist ausschließlich für die Nutzung mit der DAW-Software Nuendo der Firma Steinberg konzipiert. Es nutzt dabei nicht Steinbergs VST-Schnittstelle [IOS12], sondern ist als sogenanntes „Core-“ bzw. „Program-Plugin“ deutlich enger in die Host-Software eingebunden. Dies ermöglicht, an bestimmten Punkten mit der Semantik herkömmlicher Audio-Plugins zu brechen: während normalerweise eine feste Zuordnung zwischen einem Plugin und der dazugehörigen Audio-Spur besteht, können mit S.A.W. zum Beispiel mehrere Audiodateien, die auf der selben Spur liegen, unterschiedlichen Quellen zugeordnet werden [MAGF09]. Es ist daher fraglich, ob S.A.W. noch als Audio-Plugin im engeren Sinne betrachtet werden kann, oder ob es vielmehr eine allgemeine Erweiterung der Host-Software darstellt.

[15] Quellcode: http://sourceforge.net/projects/swonder - Abruf: 2014-06-20

S.A.W. soll dem Nutzer die Möglichkeit bieten, die räumliche Verteilung des Audiomaterials unabhängig von einem konkreten Zielsystem vorzunehmen. Parameter wie die Positionen der einzelnen Quellen können dafür nicht nur als Automationsdaten im Host-Projekt gespeichert werden: IOSONO bietet zusätzlich den Export in ein proprietäres, objekt-basiertes Format an. In diesen sogenannten „audio scene files“ werden die Audiodaten mit ihren räumlichen Daten verknüpft gespeichert. Die in diesem Format vorliegenden Daten können dann einerseits auf den WFS-Systemen der Firma IOSONO abgespielt werden, andererseits als Grundlage für den Export in andere Raumklang-Formate dienen [Mel10, S. 54 ff.].

### 2.5.2 Sonic Emotion

Sonic Emotion entwickelten zur Steuerung des „Sonic Wave I“ ein Audio-Units-Plugin. Eine Vielfalt weiterer Plugins für andere Schnittstellen befindet sich in der Entwicklung. Ursprünglich angekündigt war deren Veröffentlichung für 2013 [CDI13], doch liegen bisher keine weitere Informationen zu diesen Plugins vor.

### 2.5.3 Bledas Plugin

Bleda u.a. stellten ein WFS-System vor, mit dem ein Konzept für die Anbindung von WFS-Systemen an DAW-Software demonstriert werden sollte [BLEP05a]. Die Besonderheit an diesem Ansatz ist das dazugehörige VST-Plugin: es erlaubt, die Positionsdaten der jeweiligen virtuellen Quelle zusammen mit den entsprechenden Audiodaten über eine Netzwerk-Verbindung an das Rendering-System zu streamen. Die Möglichkeiten zur Steuerung der Quellen beschränken sich dabei auf zweidimensionale Positionierung [BLEP05b].

### 2.5.4 „Wave Field Synthesis for all“

Lekschat und Göbel präsentierten einen Ansatz, bei dem das WFS-Rendering direkt in einem VST3-Plugin erfolgt [LG11]. In ihren Versuchen konnte damit eine einstellige Zahl virtueller Quellen auf 32 separate Laustprecherkanäle verteilt werden.

### 2.5.5 WONDER: LADSPA-Plugin

Für WONDER wurde ein LADSPA-Plugin entwickelt, das es erlaubt, die Positionierung einer einzelnen Quelle oder von Gruppen von zwei oder vier Quellen zu automatisieren [Baa08, S. 52 f.]. Das Plugin versendet ausschließlich Positionsdaten über das Netzwerk an die zuständigen WONDER-Komponenten. Weitere Informationen (Typ der Quelle, Winkel) werden nicht ver-

sandt. Das Empfangen von Positionsnachrichten, um diese mit den Automationsmöglichkeiten des jeweiligen Hosts aufzuzeichnen, ist ebenfalls nicht vorgesehen.

Das LADSPA-Plugin ist als Teil des offiziellen Quellcode-Pakets in der aktuellen WONDER-Version 3.1.0 enthalten.

### 2.5.6 WONDER: erstes VST-Plugin


Abbildung 2.6: Screenshot der Benutzeroberfläche des ersten VST-Plugins für WONDER [Roo14]

Sebastian Roos entwickelte ein erstes VST-Plugin für WONDER [Baa08, S. 50]. Eine Instanz des Plugins steuert dabei bis zu zehn Quellen, indem sie entsprechende Netzwerk-Nachrichten versendet. Abbildung 2.6 zeigt einen Screenshot der Benutzeroberfläche des Plugins. Auch dieses Plugin kann lediglich Nachrichten an WONDER verschicken, nicht aber empfangen [Roo14].

### 2.5.7 WONDER: SPURCEMENT

Ein komplexeres Plugin-Konzept zur Steuerung von WONDER veröffentlichte Mann [Man12]. Unter dem Namen „SPURCEMENT“ entwickelte er eine Kombination von zwei Plugins (für die Schnittstellen VST, AU und RTAS): ein Master-Plugin bietet die Haupt-Benutzerschnittstelle, über die die Steuerung aller virtueller Quellen vorgenommen werden kann. Ein Client-Plugin, von dem pro virtueller Quelle eine Instanz benötigt wird, fungiert als Schnittstelle zum Host, sodass pro Client-Plugin die Automationsdaten für eine Quelle verwaltet werden. Die Client-

Plugins kommunizieren dabei per Netzwerk-Nachrichten mit dem Master-Plugin, welches als einziges der Plugins mit dem restlichen WONDER-System kommuniziert.

Die Benutzeroberflächen der beiden Plugins sind in Abbildung 2.7 dargestellt.

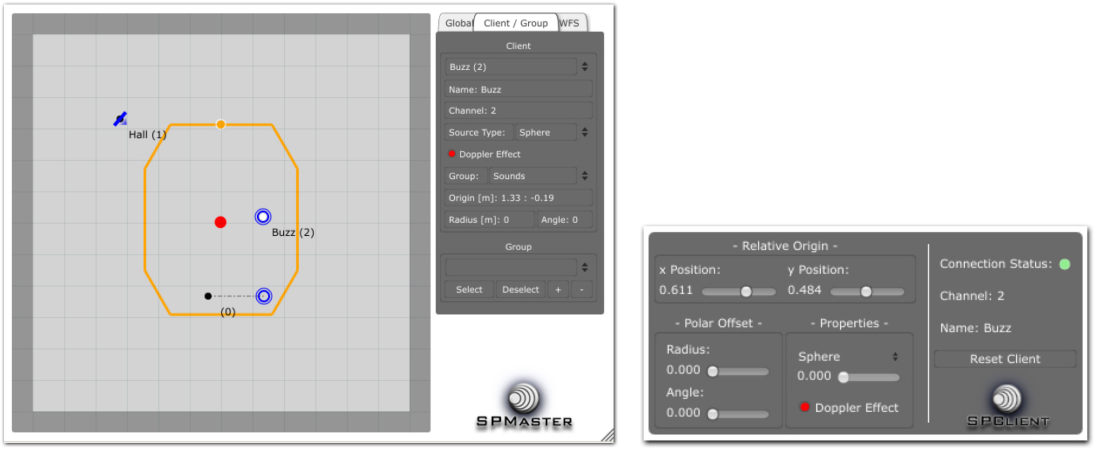


Abbildung 2.7: Screenshots der Benutzeroberflächen von SPURCEMENT-Master (links) und -Client (rechts) [Man12, S. 51 f.]

Auch bei diesem WONDER-Plugin-Konzept werden ausschließlich Steuer-Daten versendet – die Übermittlung der Audio-Daten erfolgt unabhängig davon. Anders als bei den früheren Plugins für WONDER können hier aber auch Steuerdaten von anderen WONDER-Komponenten empfangen und als Automationsdaten aufgezeichnet werden.

Im Gegensatz zum WONDER-Software-Paket selbst ist „SPURCEMENT“ nicht quelloffen.

## 2.6 Das Wellenfeldsynthese-Labor an der HAW

Seit 2011 existiert an der HAW Hamburg ein Labor für Wellenfeldsynthese [Foh13]. Hier ist in einem Laborraum eine WFS-Anlage der Firma Four Audio[16] installiert. In dem Raum sind auf einer Höhe von etwa zwei Metern 26 Lautsprecher-Module aufgehängt, sodass sie ein Rechteck bilden (Abmessungen ca 5 x 5,7 m). Jedes dieser Module verfügt über 26 Lautsprecher, die über acht separate Audio-Kanäle angesteuert werden. Somit verfügt das gesamte Lautsprecher-System über 208 separate Wiedergabekanäle.

Für den Betrieb dieser Anlage stehen vier Rechner bereit. Auf einem Cluster von drei Linux-Rechnern läuft die WONDER-Software zur Berechnung der Wellenfeldsynthese. Ein Apple

[16] http://www.fouraudio.com - Abruf: 2014-06-20

Mac speist das System mit den entsprechenden Audio-Eingangs-Daten. Auf diesem Rechner läuft zudem die grafische Benutzeroberfläche des WONDER-Systems.

Die Übertragung der Audiodaten zwischen den Rechnern und zu den Lautsprecher-Panels erfolgt mittels Dante, einem Netzwerkprotokoll zur Übertragung von Audiodaten, das von der Firma Audinate[17] entwickelt wurde. Damit die jeweiligen Netzwerkverbindungen zuverlässig und unbeeinflusst von anderem Netzwerkverkehr funktionieren, sind sowohl für die Dante-Audioübertragung als auch für die Kommunikation der Komponenten des WONDER-Systems jeweils separate, kabelgebundene Netzwerke installiert.

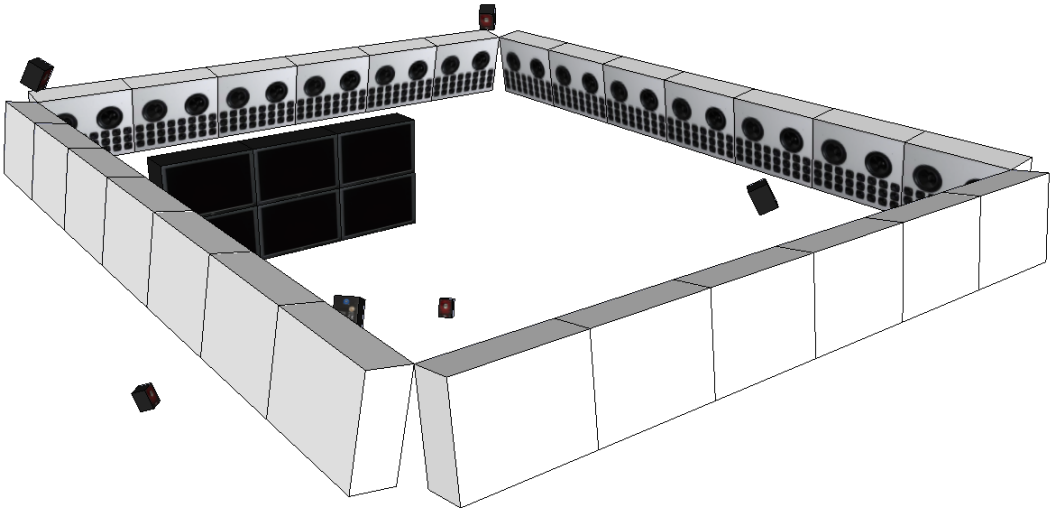

Abbildung 2.8: Skizze des Wellenfeldsynthese-Labors an der HAW [Nog12, S. 19]

Ergänzend steht in dem Labor eine sogenannte Powerwall zur Verfügung, eine aus sechs HD-Monitoren zusammengesetzte Bildschirmfläche, die – angesteuert durch einen separaten Windows-Rechner – wie ein großer zusammenhängender Bildschirm genutzt werden kann. Darüber hinaus ist ein Tracking-System der Firma Advanced Realtime Tracking[18] vorhanden, mit dem mittels Infrarot-Kameras die Positionen von dafür konzipierten Marker-Objekten verfolgt werden können. Dieses System kann dazu genutzt werden, Positionen von virtuellen Klangquellen der WFS-Anlage mittels Gesten zu steuern [FN13]. Zudem besteht dank einer Modifikation der WONDER-Software die Möglichkeit, anhand der Position eines Marker-Objektes die Darstellung von fokussierten Quellen dynamisch auf eine bestimmte Hörer-Position auszu-

[17] http://www.audinate.com - Abruf: 2014-06-20
[18] http://www.ar-tracking.com - Abruf: 2014-06-20

richten [Chr14]. Abbildung 2.8 zeigt den Aufbau des Labors mit Lautsprecherpanels, Powerwall und den Infrarot-Kameras des Tracking-Systems.

Zusätzlich zu der geschilderten Einrichtung werden in dem Labor derzeit je ein Video- und ein Audio-Produktions-Arbeitsplatz eingerichtet. Anhand dieser Arbeitsplätze, die sich in ihrer Ausstattung an herkömmlichen Video- und Audio-Produktionsumgebungen orientieren, sollen unter anderem neue Software-Konzepte entwickelt werden, um die WFS-Anlage besser in etablierte Audio-Produktions-Vorgänge einzubinden. Darüber hinaus wird die Laboreinrichtung durch ein Touchtable-System ergänzt, das als neue Benutzerschnittstelle in die vorhandene Laborumgebung eingebunden werden soll.

## 2.7 WONDER – genauere Betrachtung

Mit WONDER existiert ein leistungsfähiges, quelloffenes Software-System für die Wellenfeldsynthese. Die Besonderheit von WONDER im Gegensatz zu anderen Open-Source-WFS-Systemen besteht darin, dass WONDER aus mehreren Komponenten besteht, die auf mehrere Rechner verteilt werden können. Dadurch entsteht ein skalierbares System, das bei Einsatz entsprechend vieler Rechner auch sehr hohe Zahlen an separaten Laustprecherkanälen ansteuern kann.

### 2.7.1 Komponenten

Im Wesentlichen besteht das WONDER-System aus den folgenden Komponenten:

- *tWONDER (Rendering-Einheit):* Die Komponente tWONDER ist als Rendering-Einheit für die eigentliche Verteilung der Audiosignale auf die Lautsprecher zuständig. Hier werden eingehende Audiosignale in Abhängigkeit von den Positionsdaten einer ihnen zugeordneten virtuellen Quelle auf die Ausgangs-Kanäle der entsprechenden Lautsprecher verteilt. Dabei wird jeweils entsprechend den Prinzipien der Wellenfeldsynthese die Verzögerung des Signals sowie die Dämpfung der Amplitude passend berechnet. Zur Verteilung der Rechenlast ist jede tWONDER-Instanz jeweils nur für einen bestimmten Teil des Gesamt-Lautsprecher-Systems zuständig – entsprechend arbeiten in einem typischen WONDER-System meist mehrere tWONDER-Instanzen parallel.

- *Scoreplayer:* Die Scoreplayer-Komponente ist dafür konzipiert, die Positionsdaten der einzelnen virtuellen Klangquellen und ihre Bewegungen über den Verlauf der Zeit aufzuzeichnen, abzuspeichern und anschließend wieder abrufbar zu machen.

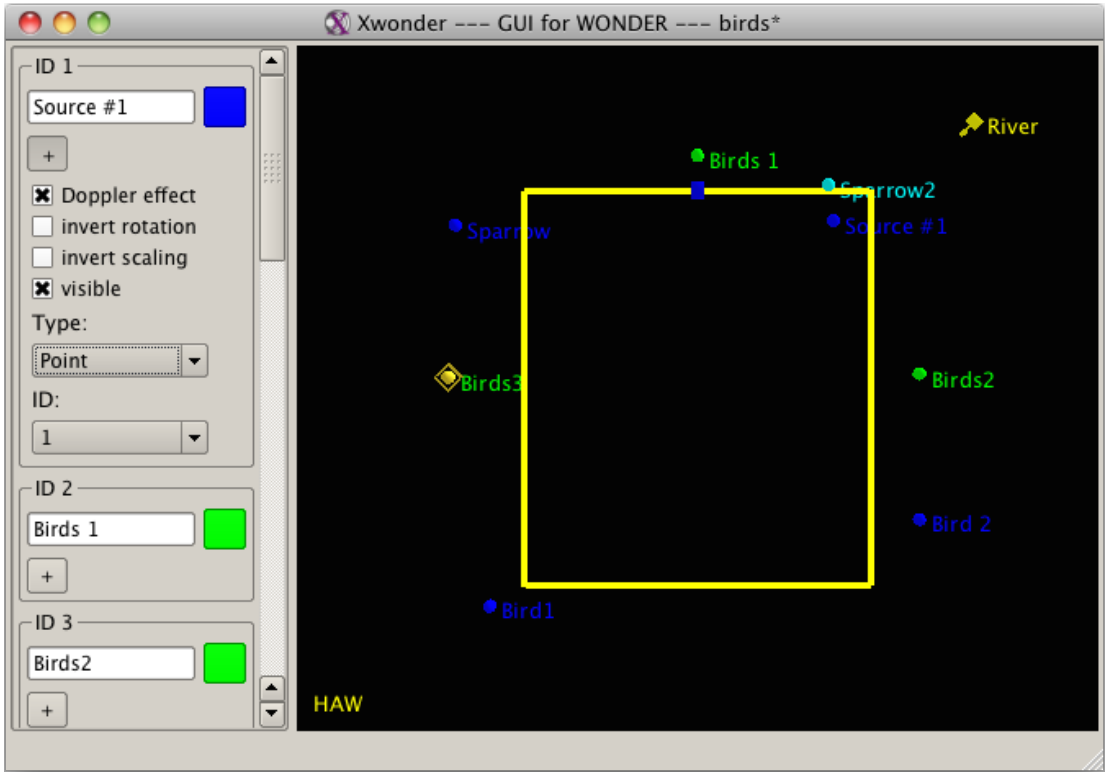


Abbildung 2.9: Screenshot der WONDER-Benutzeroberfläche xWONDER

- *xWONDER (Grafische Benutzeroberfläche):* xWONDER ist die grafische Benutzeroberfläche des Systems. Mit ihr lassen sich virtuelle Quellen erstellen und ihre Position sowie weitere Parameter bearbeiten, wie in Abbildung 2.9 zu erkennen ist. Darüber hinaus soll xWONDER aber auch ermöglichen, die Positions-Verlaufs-Daten des Scoreplayers grafisch zu editieren [Mon07, S. 33 ff.].

  Zudem kann man mit dem Programm mehrere Quellen zu Gruppen zusammenfügen. Sodann können sie als Gruppe neu positioniert oder auch um einen Bezugspunkt rotiert werden, wobei die relativen Positionen dieser Quellen zueinander jeweils erhalten bleiben. Im Gegensatz zu allen anderen Operationen, bei denen xWONDER lediglich als Benutzerschnittstelle dient und die Umsetzung der Benutzereingaben von anderen Komponenten übernommen wird, erfolgt hierbei die Berechnung der neuen Positionen der einzelnen Quellen innerhalb von xWONDER.

- *cWONDER (Kontroll-Einheit):* cWONDER ist die zentrale Kontroll-Einheit. Hier sind die jeweils aktuellen Positionen und sonstigen Parameter aller virtuellen Quellen zentral

gespeichert. Zudem dient cWONDER als zentraler Kommunikations-Knotenpunkt, über den sämtliche Kommunikation zwischen den übrigen Komponenten abgewickelt wird.

- *jfWONDER (Zeitbasis):* jfWONDER ist ein kleines Hilfsprogramm, das anhand eingehender Audiodaten regelmäßig (üblicherweise alle 1024 Samples) eine Nachricht an cWONDER schickt. Diese dienen cWONDER als Basis zur Berechnung der jeweils aktuellen Zeit. Ein Betrieb des Systems ist auch ohne diese Zeitbasis möglich, jedoch entfallen dadurch alle Funktionalitäten, bei denen Aktionen zu bestimmten, in der Zukunft liegenden Zeitpunkten erfolgen sollen.

### 2.7.2 Kommunikation zwischen den Komponenten

Wie bereits angedeutet dient cWONDER als zentraler Kommunikations-Knotenpunkt des WONDER-Systems. Sofern eine Komponente mit dem Rest des Systems kommunizieren muss, sendet sie entsprechende Nachrichten an cWONDER, von wo aus diese dann jeweils an die zuständige(n) Komponente(n) weitergeleitet werden.

cWONDER bietet hierfür die Möglichkeit, dass die beteiligten Komponenten einen sogenannten „Stream“ von Nachrichten beziehen können, der jeweils alle Nachrichten enthält, die für die Erledigung einer bestimmten Aufgabe (und damit für die jeweilige Komponente) relevant sind. Insgesamt gibt es vier verschiedene solcher „Streams“:

1. *„Visual Stream“:* Dieser Stream beinhaltet alle Informationen, die von einer Benutzerschnittstelle benötigt werden, um den Zustand des Systems darzustellen. xWONDER empfängt diesen Stream – er kann aber auch für andere, neu entwickelte Nutzerschnittstellen genutzt werden.

2. *„Score Stream“:* Der „Score Stream“ beliefert den Scoreplayer mit allen für ihn relevanten Nachrichten.

3. *„Render Stream“:* Mit diesem Stream erhalten die Rendering-Komponenten (tWONDER) die Informationen, die ihnen als Grundlage für ihre Berechnungen dienen.

4. *„Timer Stream“:* Über diesen Stream kann die jeweils aktuelle, auf Basis der jfWONDER-Komponente errechnete Systemzeit des WONDER-Systems von cWONDER bezogen werden. Mit jedem Tick der System-Zeit wird eine neue Nachricht mit dem jeweils aktuellen Tick-Zähler-Wert gesendet. Dieser Stream wird jedoch von keiner der bestehenden WONDER-Komponenten genutzt.

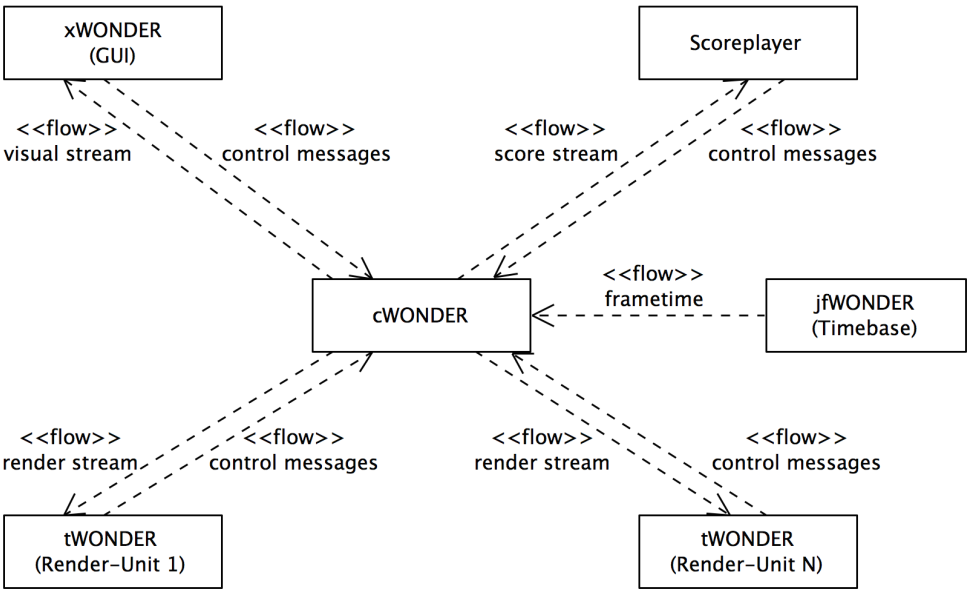


Abbildung 2.10: Informationsfluss zwischen den WONDER-Komponenten

Abbildung 2.10 zeigt den Informationsfluss zwischen den verschiedenen Komponenten – zur Vereinfachung sind dabei nur zwei tWONDER-Komponenten beispielhaft dargestellt.

Für diese Kommunikation nutzt WONDER das Nachrichtenformat Open Sound Control.

## 2.8 Open Sound Control

Open Sound Control (OSC) ist ein Netzwerk-Protokoll zur Kommunikation zwischen Computern, Synthesizern und anderen Multimedia-Geräten, das am Center for New Music and Audio Technology (CNMAT) der University of California in Berkeley entwickelt wurde [WFM03].

### 2.8.1 Format

Im Gegensatz zu vielen anderen Netzwerk-Protokollen beschränkt sich der OSC-Standard auf die Definition eines einheitlichen Formats für den Inhalt der übermittelten Nachrichten [FS09, S. 117]. OSC-Nachrichten bestehen im Wesentlichen aus drei Teilen [Wri02]:

1. *„Address Pattern“:* Die Nachricht beginnt mit einem String, der im OSC-Standard „Address Pattern“ genannt wird. Der String beginnt mit einem Schrägstrich-Symbol („/“), worauf eine Folge von einem oder mehreren Namen, gegebenenfalls getrennt durch weitere Schrägstriche, folgt. Hierdurch entsteht ein hierarchischer Namensraum (ähnlich

einer URI oder einem Datei-Pfad). Jedem „Address Pattern“ kann in der empfangenden Applikation eine Methode zugeordnet werden, die für die Verarbeitung der entsprechenden Nachricht zuständig ist. Das „Address Pattern“ wird oft auch als „OSC-Pfad“ oder „OSC-Befehl“ bezeichnet.

2. *„Type Tag String“:* Als nächstes folgt ein weiterer String, der Anzahl und Datentyp der übermittelten Argumente angibt. Er beginnt mit einem Komma, gefolgt von einem Buchstaben pro übermitteltem Argument. Der Buchstabe gibt den Typ des jeweiligen Arguments an.

3. *„Arguments“:* Schließlich folgt eine Sequenz von übermittelten Argumenten, die im Format den Angaben des „Type Tag Strings“ entsprechen müssen. Diese Argumente dienen üblicherweise als Parameter der Methode, deren Aufruf durch den Empfang der Nachricht beim Empfänger ausgelöst wird.

### 2.8.2 OSC und Multicast

Der OSC-Standard legt nicht fest, welches Netzwerk-Protokoll für den Transport der Nachrichten zu verwenden ist. Viele OSC-Anwendungen (darunter auch WONDER) nutzen dafür das UDP-Protokoll [Pos80]. Da UDP-Nachrichten per Multicast versendet werden können, gilt dies entsprechend auch für OSC-Nachrichten.

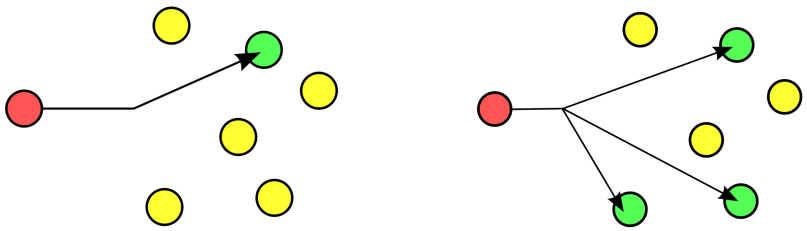

Abbildung 2.11: Unicast- (links) und Multicast-Kommunikation (rechts) [Public Domain]

Anders als beim herkömmlichen Ende-zu-Ende-Versand von Nachrichten (Unicast), wo eine Nachricht immer von genau einem Absender an genau einen Empfänger gesendet wird, kann der Absender bei Multicast eine Nachricht an eine Gruppe von mehreren Empfängern senden, wie in Abbildung 2.11 dargestellt ist. Sollen identische Nachrichten an eine Vielzahl von Empfängern gesendet werden, ist eine Übermittlung per Multicast deutlich effizienter: Der Absender muss nur eine einzige Nachricht versenden, die Verteilung der Nachricht an die Gruppe obliegt dann der Netzwerk-Infrastruktur. Für diese wiederum ist die Verteilung

einer Nachricht an eine Gruppe ebenfalls mit deutlich weniger Gesamt-Aufwand verbunden, als wenn der Absender jedem Mitglied der Gruppe die selbe Nachricht per Unicast schicken würde.

In Fällen, wo identische OSC-Nachrichten an eine Vielzahl von Adressaten gesendet werden sollen, ist der Versand mittels UDP-Multicast also eine Möglichkeit, dies mit geringem Aufwand umzusetzen.

# 3 Softwareentwurf und -implementierung

Die bisherigen Erfahrungen zeigten insbesondere bei der Wiedergabe von längeren, entsprechend vorbereiteten Inhalten wie zum Beispiel audiovisuellen Präsentationen einen deutlichen Mangel in der Bedienbarkeit des WONDER-Systems: Zwar ist es mittels der Benutzeroberfläche von xWONDER komfortabel möglich, die jeweils aktuellen Positionen virtueller Quellen zu steuern - möchte man jedoch Positionsänderungen in einem festen zeitlichen Verlauf festlegen und später wieder abrufen, stößt das bisherige System an seine Grenzen.

Mit dem ScorePlayer existiert zwar eine WONDER-Komponente, die für diesen Zweck vorgesehen ist. In der Praxis fand dieser aber keine Anwendung, da sich den bisherigen Nutzern des Labors das Bedienkonzept des ScorePlayers nicht erschloss. Stattdessen wurden für solche Zwecke bisher aufwändige Skripte geschrieben, die jede einzelne zu sendende OSC-Nachricht mit einem entsprechenden Versende-Zeitpunkt aufführten. Diese mussten dann jeweils synchron zum Abspielen der entsprechenden Audio-Daten manuell gestartet werden.

Zur Behebung dieses Mangels sollte im Rahmen dieser Arbeit eine entsprechende Lösung entwickelt werden, die die Steuerung der virtuellen Quellen des WONDER-Systems deutlich enger an die Steuerung der abgespielten Audio-Daten anbindet. Die Entwicklung dieser Software – von der Anforderungs-Analyse über den Entwurf und die Implementierung bis hin zu den abschließenden Tests – wird in diesem Kapitel geschildert.

## 3.1 Anforderungen

Es sollte also ein Werkzeug geschaffen werden, mittels dessen die Steuerung des WONDER-Systems in herkömmliche Audio-Produktions-Software integriert werden kann. Um eine einfache Einbindung in verschiedene DAW-Programme zu ermöglichen, sollten hierfür die verfügbaren Schnittstellen genutzt werden. Somit bestand das grundsätzliche Vorhaben darin, ein Audio-Plugin zu entwickeln, das als Mittler zwischen DAW-Software und cWONDER fungiert. Das Plugin soll die Möglichkeiten der Parameter-Automation nutzen und dadurch dem Anwender ermöglichen, die Positionen sowie weitere Parameter der virtuellen Quellen in ihrem zeitlichen Verlauf aufzuzeichnen, innerhalb der DAW zu bearbeiten und schließlich

wieder abzurufen. Als Bezeichnung des Plugins wurde der Name SPAOP (Spatial audio object positioner) gewählt.

Zur Konkretisierung des Vorhabens wurde eine Reihe von Anforderungen an die zu entwickelnde Software festgelegt.

### 3.1.1 Funktionale Anforderungen

Im einzelnen wurden die folgenden funktionalen Anforderungen an das Plugin ermittelt:

- *Steuerung der Parameter einer virtuellen Quelle:* Entsprechend der üblichen Zuordnung eines Audio-Plugins zu einer konkreten Audio-Spur der DAW soll jede Instanz des Plugins die Parameter einer virtuellen Quelle des WONDER-Systems kontrollieren. Dafür muss es die entsprechenden OSC-Nachrichten versenden können. Eine Steuerung von Quell-Gruppen ist zunächst nicht vorgesehen. Bei der Entwicklung des Quellcodes sollte aber berücksichtigt werden, dass eine solche Funktion eventuell in späteren Programmversionen ergänzt werden soll.

- *Automatisierung der Parameterverläufe:* Die Möglichkeiten zur Automatisierung von Parameterverläufen sollen genutzt werden, um die virtuellen Quellen über den zeitlichen Verlauf automatisiert steuern zu können.

- *Synchronisierung zum WONDER-Gesamtsystem:* Die im Plugin vorgehaltenen Parameter sollen konsistent mit dem tatsächlichen Zustand des Systems sein. Dafür muss das Plugin Nachrichten von cWONDER empfangen und verarbeiten können. Als Konsequenz aus dieser Anforderung ergibt sich, dass auch solche Parameteränderungen in die Automation mit einfliessen können, die durch andere Komponenten des WONDER-Systems ausgelöst wurden.

- *Benutzeroberfläche:* Die Position der vom Plugin gesteuerten Quelle sowie alle ihre Parameter sollen in der Benutzeroberfläche des Plugins dargestellt werden und von dort aus manipulierbar sein. Die Benutzeroberfläche soll zudem die jeweils aktuelle Lautsprecheraufstellung darstellen können.

- *Funktionsfähigkeit auch ohne Verbindung zum WONDER-System:* Es ist denkbar, dass Anwender die Automationsdaten auch dann bearbeiten wollen, wenn sie nicht mit dem WONDER-System verbunden sind. Dieser Anwendungsfall muss durch das Plugin bestmöglich unterstützt werden. Soweit sinnvoll, ist hierfür auch eine OSC-Kommunikation der einzelnen Plugin-Instanzen untereinander zu implementieren.

### 3.1.2 Weitere Anforderungen

Neben den konkreten funktionalen Anforderungen soll die entwickelte Software verschiedenen weiteren Anforderungen genügen.

#### Gebrauchstauglichkeit

Bezüglich der Gebrauchstauglichkeit soll sich das Plugin an bestehenden Bedienkonzepten für das WONDER-System sowie für Audio-Plugins orientieren.

Mit xWONDER besteht eine etablierte grafische Benutzeroberfläche für WONDER, bei deren Entwurf der Aspekt der Gebrauchstauglichkeit ein wesentlicher Faktor war [Mon07] und die sich in jahrelanger Praxisanwendung bewährt hat. Damit für Nutzer, die den Umgang mit xWONDER bereits gewohnt sind, die Bedienung des Plugins keine unnötige Einarbeitung erfordert, soll sich der Entwurf der SPAOP-Oberfläche im Wesentlichen an der Oberfläche von xWONDER-Oberfläche orientieren.

Anders als xWONDER, mittels dessen Benutzeroberfläche alle virtuellen Quellen des Systems gesteuert werden können, soll jede SPAOP-Instanz nur für die Steuerung einer virtuellen Quelle zuständig sein. Um dem Anwender aber eine bessere Orientierung bezüglich der Positionen der übrigen Quellen zu ermöglichen, sollen dennoch sämtliche aktiven Quellen in der Benutzeroberfläche jeder SPAOP-Instanz angezeigt werden können.

Zusätzlich zur Nutzung der Benutzeroberfläche des Plugins lassen sich Audio-Plugins wie SPAOP auch indirekt über die Host-DAW bedienen – insbesondere dadurch, dass man die Automationsdaten der Plugin-Parameter in der Host-Software bearbeitet. Es ist daher darauf zu achten, dass die entsprechenden Schnittstellen vollständig unterstützt werden.

#### Performanz

Grundsätzlich sollte bei der Entwicklung von Audio-Plugins immer so wenig Rechenleistung wie möglich genutzt werden, um das Gesamtsystem performant zu halten. Im Rahmen dieser Arbeit ist Performanz aber lediglich ein zweitrangiges Ziel. Es gilt zunächst vor allem, ein funktionsfähiges Plugin zu entwickeln. Mittels eines ersten Prototypen kann dann in Lasttests die Leistungsfähigkeit dieses Plugins getestet werden. Im Bedarfsfalle können sodann Optimierungen für eine bessere Performanz vorgenommen werden.

#### Wartbarkeit und Wiederverwendbarkeit

Das Plugin soll an der HAW auch nach Abschluss dieser Arbeit von anderen Studierenden weiter entwickelt werden. Es ist daher sicherzustellen, dass der entwickelte Quellcode leicht

wartbar ist. Neben einer ausführlichen Dokumentation erfordert dies auch, den Quellcode so verständlich wie möglich zu halten und keine unnötig komplizierten Code-Konstrukte zu nutzen.

Da an der HAW auch in Zukunft weitere Software für das WONDER-System entwickelt werden soll, ist zudem darauf zu achten, den Quellcode so modularisiert zu entwickeln, dass möglichst große Teile auch bei der Entwicklung anderer Anwendungen wiederverwendet werden können.

#### Kompatibilität

Wünschenswert wäre die Entwicklung eines Plugins, das zu allen denkbaren DAW-Systemen kompatibel ist. Dieses Ziel erscheint im Rahmen einer Bachelorarbeit jedoch als unrealistisch. Es gilt also, bezüglich der zu unterstützenden Plugin-Schnittstellen und Betriebssysteme Prioritäten zu setzen. Im WFS-Labor der HAW kommen zur Zeit die DAW-Programme Cubase 7 und Ardour auf einem Apple-Rechner unter OS X zum Einsatz. Cubase 7 unterstützt die Schnittstellen VST3 und VST2, Ardour unterstützt LADSPA und Audio Units. Im Sommer diesen Jahres (2014) wird das Labor um einen Windows-basierten Audio-Rechner ergänzt werden, auf dem ebenfalls aktuelle DAW-Programme der Firma Steinberg zum Einsatz kommen werden.

Damit das Plugin auf diesen beiden Rechnern genutzt werden kann, soll vor allem die VST3-Schnittstelle sowohl unter OS X als auch unter Windows unterstützt werden. Eine Unterstützung weiterer Schnittstellen soll ebenfalls anvisiert werden, hat aber geringere Priorität.

## 3.2 Verwendete Bibliotheken und Werkzeuge

Um den genannten Anforderungen – gerade auch bezüglich der Kompatibilität – gerecht zu werden, wurden verschiedene spezielle Programmbibliotheken genutzt. Im folgenden Abschnitt sollen diese zunächst erläutert werden, bevor dann ein kurzer Überblick über die sonstigen zur Entwicklung verwendeten Werkzeuge erfolgt.

### 3.2.1 JUCE

Die unterschiedlichen Audio-Plugin-Standards der verschiedenen Hersteller arbeiten im Wesentlichen nach dem gleichen Prinzip, unterscheiden sich aber stark in den konkreten Schnittstellen. So werden zum Beispiel für die Übergabe der Audio-Daten oder der Automations-Parameter bei VST3 und AU deutlich unterschiedliche Mechanismen genutzt. Ein VST3-Plugin kann somit nur mit größerem Aufwand an die Audio-Units-Schnittstelle angepasst werden.

Mit JUCE[1] gibt es jedoch eine Programm-Bibliothek, welche die Schnittstellen von RTAS, AAX, Audio Units, VST2 und VST3 abstrahiert und mittels entsprechender Wrapper auf eine eigene Audio-Plugin-Schnittstelle adaptiert. Dadurch wird ermöglicht, ein einziges Plugin zu entwickeln, von dem dann automatisch RTAS-, AAX-, AU-, VST2 und VST3-Versionen erzeugt werden können (sofern man über die entsprechenden SDKs verfügt). Da JUCE für betriebssystemunabhängige Softwareentwicklung ausgelegt ist, lassen sich diese Plugins sowohl für OSX als auch für Windows erstellen (sofern die jeweilige Schnittstelle das Betriebssystem grundsätzlich unterstützt).

JUCE ist jedoch nicht ausschließlich für die Entwicklung von Audio-Plugins konzipiert. Die Bibliothek soll vielmehr als umfassende Sammlung von C++-Klassen dienen und Hilfsmittel für die Entwicklung unterschiedlichster Programme bieten. So enthält JUCE zum Beispiel Klassen zur Speicherverwaltung, diverse GUI-Klassen oder auch (plattformunabhängige) Klassen für die Datei-Ein- und -Ausgabe.

Ergänzt wird die Bibliothek durch das Programm *Introjucer*. Mit diesem können Softwareprojekte unabhängig von der Ziel-Plattform und der genutzten Entwicklungsumgebung konfiguriert werden. Auf Basis eines *Introjucer*-Projekts können dann für verschiedene Entwicklungsumgebungen die entsprechenden Projekt-Dateien erzeugt werden. Zudem enthält das Programm einen „What you see is what you get“-GUI-Editor. Damit kann man GUI-Klassen im Rahmen einer grafischen Benutzeroberfläche bearbeiten und sodann den entsprechenden Quellcode generieren lassen.

Die JUCE-Bibliothek ist quelloffen[2] und ausführlich dokumentiert[3]. Sie unterliegt Version 2 der GNU Public Licence [Fre91], was bedeutet, dass sie für quelloffene Projekte kostenlos genutzt werden kann. Für nicht-quelloffene Projekte wird eine entsprechende kommerzielle, konstenpflichtige Lizenz angeboten [4].

### 3.2.2 Liblo

Liblo[5] ist eine C-Bibliothek zur Kommunikation mittels des OSC-Protokolls, die auch von sämtlichen Komponenten des WONDER-Systems genutzt wird.

Neben einfachen Funktionen zum Versenden von Nachrichten bietet sie die Möglichkeit, einen OSC-Server einzurichten. Dieser ruft bei jeder eingehenden Nachricht diejenige Funktion auf, die zuvor dem entsprechenden Nachrichtentyp anhand von OSC-Pfad sowie Anzahl und

[1] http://www.juce.com/about-juce - Abruf: 2014-06-20
[2] Quellcode: https://github.com/julianstorer/JUCE - Abruf: 2014-06-20
[3] http://www.juce.com/api - Abruf: 2014-06-20
[4] http://www.juce.com/documentation/commercial-licensing - Abruf: 2014-06-20
[5] http://liblo.sourceforge.net - Abruf: 2014-06-20

Typen der übermittelten Parametern zugeordnet wurde. Dieser Mechanismus kann wahlweise von einem selbst eingerichteten Thread oder mittels eines Liblo-eigenen Server-Threads ausgeführt werden.

Liblo ist quelloffen[6] und für die Nutzung unter Linux, OSX oder Windows ausgelegt. Für die Nutzung des Liblo-eigenen Server-Threads werden allerdings Thread-Funktionen nach POSIX-Standard [IEE13] benötigt, die unter Windows nicht vom Betriebssystem selbst unterstützt werden. Daraus ergeben sich unter Windows zwei Möglichkeiten, die Liblo-Bibliothek zu nutzen. Einerseits kann man eine zusätzliche Bibliothek verwenden, welche die POSIX-Thread-Funktionen verfügbar macht. Andererseits kann man auf den Liblo-eigenen Server-Thread verzichten und im Bedarfsfall einen vergleichbaren Thread selbst implementieren. Für die vorliegende Arbeit wurde die letztere Alternative gewählt.

### 3.2.3 C++11

Die genutzte Programmiersprache C++ hat mit dem Standard C++11 (definiert in ISO/IEC 14882:2011) einige Neuerungen erfahren, die inzwischen sowohl unter OSX als auch unter Windows von aktuellen Compilern unterstützt werden.

So wurden zum Beispiel erstmals etablierte Mechanismen zur nebenläufigen Programmierung wie Threads und Mutexe in die Standard-Bibliothek *std* aufgenommen [Str13, S. 121 ff.]. Zuvor wurden diese Funktionalitäten durch die verschiedenen Betriebssysteme auf höchst unterschiedliche Weise unterstützt, was es deutlich erschwerte, betriebssystemunabhängig zu programmieren.

Ebenfalls neu in die Standard-Bibliothek aufgenommen wurden die sogenannten „Smart-Pointer“ *std::unique_ptr* und *std::shared_ptr* [Str13, S. 118 ff.]. „Smart-Pointer“ dienen der dynamischen Speicherverwaltung: Pointer auf mit *new* allokierte Objekte werden einem „Smart-Pointer“ zugewiesen, der dann zu einem sinnvollen Zeitpunkt (abhängig vom jeweiligen Konzept des „Smart-Pointers“) den allokierten Speicher durch den entsprechenden *delete*-Aufruf wieder frei gibt.

Diese beiden neuen Funktionalitäten der Standard-Bibliothek werden in der entwickelten Software genutzt. Über Erweiterungen der Standard-Bibliothek hinaus ergänzt der neue Standard C++ auch um verschiedene neue Sprachelemente. Auf deren Nutzung wurde konsequent verzichtet, da diese Sprachelemente vielen Entwicklern noch unbekannt sind und den Programmcode somit weniger verständlich machen würden.

[6] Quellcode: http://sourceforge.net/p/liblo/git/ci/master/tree - Abruf: 2014-06-20

### 3.2.4 Sonstige Werkzeuge

Ergänzend zu den genannten Bibliotheken werden im Folgenden kurz die genutzten Standard-Werkzeuge zur Software-Entwicklung erläutert.

#### Entwicklungsumgebungen

Als Entwicklungsumgebungen kamen die jeweiligen Standard-Entwicklungsumgebungen der Betriebssystem-Hersteller zum Einsatz, wobei jeweils die derzeit aktuellste Version genutzt wurde. Unter OSX war dies Apples Xcode[7] (Version 5.1.1), unter Windows handelte es sich um Microsofts Visual Studio[8] Ultimate 2013.

#### Versionskontrolle mit Git

Zur Quellcode-Versionskontrolle wurde Git[9] genutzt. Git erfreut sich nicht zuletzt bei Open-Source-Projekten so großer Beliebtheit, dass man es inzwischen als De-facto-Standard ansehen kann. So liegen zum Beispiel sämtliche in dieser Arbeit genannten Open-Source-Projekte als Git-Repository vor. Eine ausführliche Schilderung der Git zugrundeliegenden Konzepte und Mechanismen sowie der Möglichkeiten, die Git zur Versionskontrolle bietet, würde den Rahmen dieser Arbeit sprengen. Es sei daher lediglich auf das Standard-Werk „Pro Git“ von Scott Chacon verwiesen [Cha09].

#### Dokumentation mit Doxygen

Die Dokumentation des Quellcodes erfolgte mit Doxygen[10]. Mit Doxygen kann man auf Basis von Quellcode und den darin enthaltenen Kommentaren automatisch eine entsprechende Dokumentation einschließlich sinnvoller Diagramme erzeugen lassen. Als Ausgabeformate für die Dokumentation stehen verschiedene Text-Formate sowie html zur Auswahl. Im Rahmen dieser Arbeit wurde lediglich eine html-Dokumentation erzeugt.

Es wurden alle entworfenen Klassen, Methoden und Konstanten ausführlich beschrieben. Insofern ist der Programmcode in der entstandenen Dokumentation deutlich detaillierter dokumentiert, als es im Text dieser Arbeit geschehen kann. Daher sei die Doxygen-Dokumentation als ein diese Arbeit ergänzendes Nachschlagewerk empfohlen.

[7] https://developer.apple.com/xcode - Abruf: 2014-06-20
[8] http://www.visualstudio.com/ - Abruf: 2014-06-20
[9] http://git-scm.com - Abruf: 2014-06-20
[10] http://www.stack.nl/~dimitri/doxygen - Abruf: 2014-06-20

## 3.3 Entwickelte Software-Komponenten

Die entwickelte Software besteht aus zwei Komponenten. Die Haupt-Komponente ist das SPAOP-Plugin. Im typischen Anwendungsfall des Plugins sind viele SPAOP-Instanzen gleichzeitig aktiv und empfangen identische Informationen mittels OSC-Nachrichten von WONDER. Zur Reduzierung der dafür erforderlichen Sendevorgänge ist vorgesehen, dass die Verteilung dieser Nachrichten an die Plugin-Instanzen mittels Multicast erfolgt. WONDER kann aber zur Zeit ausgehende OSC-Streams nur per Unicast versenden. Daher ist eine zusätzliche Applikation erforderlich, die die von WONDER per Unicast versendeten OSC-Nachrichten empfängt und an eine Multicast-Gruppe weiterleitet – der StreamMulticaster.

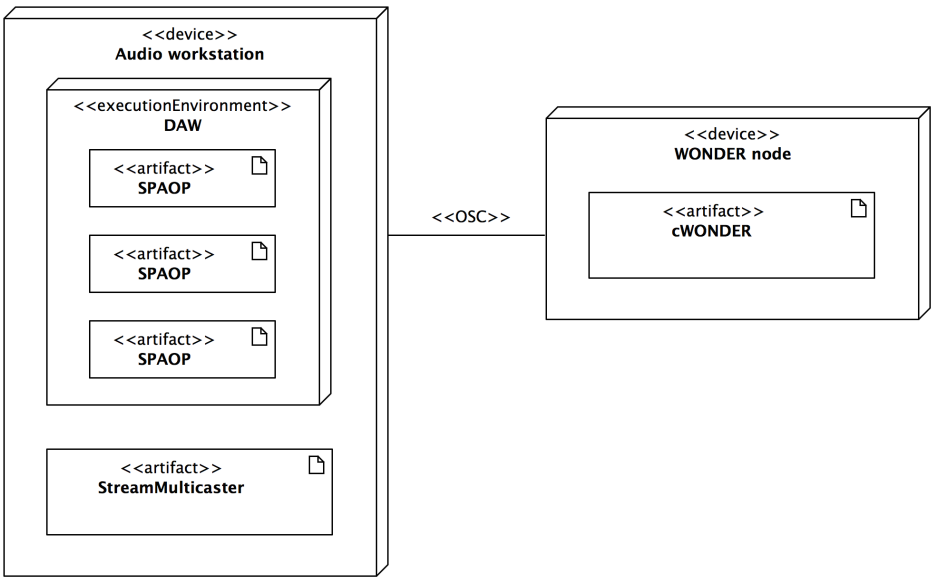


Abbildung 3.1: Typische Verteilung von Plugin und StreamMulticaster im System. Weitere WONDER-Nodes sind zur Vereinfachung nicht dargestellt.

Eine typische Verteilung der beiden Komponenten im System ist in Abbildung 3.1 zu sehen. Mehrere SPAOP-Instanzen laufen als Plugin der DAW-Host-Applikation auf dem selben Rechner wie der StreamMulticaster. Die Kommunikation mit dem WONDER-System erfolgt per OSC ausschließlich mit cWONDER. Als zentraler WONDER-Kommunikationsknotenpunkt übernimmt cWONDER dann gegebenenfalls die Kommunikation mit weiteren WONDER-Komponenten.

Die dargestellte Verteilung ist nicht alternativlos. Möglich wäre auch, dass der StreamMulticaster auf einem anderen Rechner läuft und die per Multicast weitergeleiteten Nachrichten das

Plugin über ein lokales Netzwerk erreichen. Dies wäre insbesondere dann sinnvoll, wenn noch weitere Komponenten auf anderen Rechnern den an die Multicast-Gruppe weitergeleiteten Datenstrom empfangen sollen. Ein denkbarer Fall für diese Konstellation wäre, dass mehrere synchronisierte DAW-Hosts auf mehrere Rechner verteilt sind und das WONDER-System mit Audio-Daten und vom Plugin versandten Steuer-Nachrichten versorgen sollen. So bietet zum Beispiel Cubase unter dem Namen „VST System Link“ die Funktion, zur Verteilung der Rechenlast mehrere, auf unterschiedlichen Rechnern laufende Cubase-Instanzen zu synchronisieren [BBB+12, S. 755 ff.].

## 3.4 OSC-Multicast-Kommunikation

Das Senden, Empfangen und Verarbeiten von OSC-Nachrichten ist eine der wesentlichen Funktionalitäten der entwickelten Software. Der folgende Abschnitt erläutert einerseits den Ablauf der Kommunikation mit WONDER und erklärt andererseits, wie die Kommunikation der Plugin-Instanzen untereinander verläuft.

### 3.4.1 Der „Visual Stream“

Von den vier verschiedenen OSC-Streams, die cWONDER verschickt, ist der „Visual Stream“ derjenige, der für die Rückmeldung von Zuständen des WONDER-Systems an Nutzerschnittstellen-Komponenten gedacht ist. Auch das SPAOP-Plugin verwendet diesen Stream. Daher wird im Folgenden geschildert, wie ein OSC-Kommunikations-Knoten sich mit diesem Stream verbinden kann, welche Nachrichten daraufhin übermittelt werden und welche dieser Nachrichten vom Plugin genutzt werden.

#### Verbindungsaufbau

Um sich mit cWONDER zu verbinden und den „Visual Stream“ zu beziehen, schickt man die OSC-Nachricht */WONDER/stream/visual/connect* an cWONDER. Daraufhin erhält man zunächst vier verschiedene Nachrichten:

1. */WONDER/global/maxNoSources:* Mit dieser Nachricht wird ein Integer-Parameter übermittelt, der angibt, wie viele virtuelle Quellen das sendende WONDER-System maximal verwalten kann. Da die DAW-Projekte, in denen das Plugin Anwendung finden soll, eventuell mit verschiedenen WONDER-Systemen genutzt werden sollen, erscheint es nicht sinnvoll, auf Basis dieser Nachricht das Verhalten des Plugins einzuschränken.

Die Nachricht wird daher ignoriert, das Prüfen eventueller Einschränkungen muss vom Anwender selbst vorgenommen werden.

2. */WONDER/global/renderpolygon:* Diese Nachricht enthält Informationen zu der aktuell vom WONDER-System genutzten Lautsprecherinstallation. Neben einem Namen werden dreidimensionale Koordinaten der Eckpunkte der Lautsprecher-Arrays übermittelt. Bei Empfang dieser Nachricht aktualisiert das Plugin die Darstellung der Lautsprecher-Positionen in seiner Benutzeroberfläche entsprechend.

3. */WONDER/project/xmlDump:* Hiermit erhält man einen String im XML-Format [Wor08], der detaillierte Informationen über alle aktuell aktiven virtuellen Quellen und Quell-Gruppen enthält. Die enthaltenen Informationen werden bei Empfang der Nachricht extrahiert und die im Plugin gespeicherten Informationen sodann entsprechend aktualisiert.

4. */WONDER/stream/connected:* Als letzte Information für den neu angemeldeten Stream-Client liefert diese Nachricht Informationen über die weiteren mit cWONDER verbundenen Komponenten. Pro verbundener Komponente wird eine dieser Nachrichten übermittelt, sie enthält IP, Port und den Namen der Komponente. Das Plugin ignoriert diese Nachrichten.

Ein erneutes Senden dieser Nachrichten durch cWONDER kann dadurch provoziert werden, dass man trotz bereits etablierter Verbindung die */WONDER/stream/visual/connect*-Nachricht erneut an cWONDER versendet.

#### Übermittelte Nachrichten

Nach erfolgtem Verbindungsaufbau erhält man fortan eine Vielzahl von Nachrichten, die Zustandsänderungen der virtuellen Quellen, der Quell-Gruppen oder des WONDER-Scoreplayers übermitteln.

Da sich bei Nutzung des SPAOP-Plugins mit einer DAW-Host-Software die Verwendung des Scoreplayers erübrigt, können die Scoreplayer-spezifischen Nachrichten ignoriert werden. Die Gruppierung von virtuellen Quellen sowie die Steuerung solcher Gruppen sind für das Plugin vorerst nicht vorgesehen, sodass diese Nachrichten ebenfalls ignoriert werden.

Relevant sind für das Plugin dagegen alle Nachrichten, die über die Zustände einzelner Quellen informieren. Tabelle 3.1 zeigt alle von cWONDER mit dem „Visual Stream“ versendeten Nachrichten, die tatsächlich vom Plugin verarbeitet werden. Eine Sonderrolle nimmt dabei die Nachricht */WONDER/stream/visual/ping* ein, die im Folgenden näher erläutert werden soll.

| *OSC-Pfad* | *Parameter* |
|---|---|
| /WONDER/source/activate | int sourceID |
| /WONDER/source/deactivate | int sourceID |
| /WONDER/source/type | int sourceID, int type |
| /WONDER/source/position | int sourceID, float x, float y |
| /WONDER/source/angle | int sourceID, float angle |
| /WONDER/source/name | int sourceID, string name |
| /WONDER/source/color | int sourceID, int r, int g, int b |
| /WONDER/source/dopplerEffect | int sourceID, int on |
| /WONDER/global/renderpolygon | string name, int numberOfVertices, (float x, float y, float z)* |
| /WONDER/project/xmlDump | int error, string xmlDump |
| /WONDER/stream/visual/ping | int pingCount |

Tabelle 3.1: Vom Plugin verarbeitete OSC-Nachrichten des „Visual Streams“

### Verbindungskontrolle

WONDER bedient sich eines einfachen Mechanismus zur Kontrolle ausgehender Stream-Verbindungen: in regelmäßigen Abständen (nach WONDER-Standardeinstellung einmal pro Sekunde) wird eine „Ping“-Nachricht verschickt, die einen von cWONDER gewählten Integer-Wert enthält. Jeder Stream-Empfänger muss auf diese Nachricht antworten, indem er diesen Wert mit einer entsprechenden „Pong“-Nachricht zurück an cWONDER sendet. Sollte eine „Ping“-Nachricht unbeantwortet bleiben, erhöht cWONDER einen entsprechenden Zähler. Überschreitet dieser Zähler einen gewissen Schwellwert, betrachtet cWONDER die Verbindung als abgebrochen und beendet das Versenden des Streams an den entsprechenden Empfänger.

Der StreamMulticaster muss daher entsprechende Antworten an cWONDER versenden, um den Stream fortwährend zu empfangen und an die Multicast-Gruppe weiterleiten zu können. Um den Plugins ihrerseits die Möglichkeit zu geben, anhand eingehender „Ping“-Nachrichten zu überprüfen, ob die Streamverbindung noch aktiv ist, leitet er die Ping-Nachrichten zudem an die Multicast-Gruppe weiter.

Durch entsprechende „Pong“-Antworten von den Plugins an den StreamMulticaster könnte dieser seinerseits überprüfen, wie viele Plugins derzeit die von ihm weitergeleiteten Nachrichten empfangen. Eine solche Funktionalität ist vorerst aber nicht vorgesehen.

#### Verbindungsabbau

In der Dokumentation der WONDER-Software wird eine */WONDER/stream/visual/disconnect*-Nachricht aufgeführt[11]. Die Vermutung liegt nahe, dass cWONDER bei Empfang dieser Nachricht aufhört, den entsprechenden Stream zu senden.

Tatsächlich ist eine entsprechende Funktion in cWONDER aber nicht implementiert. cWONDER kann somit momentan nur durch das Ausbleiben der „Pong“-Nachrichten dazu veranlasst werden, das Versenden eines ausgehenden Streams zu beenden.

### 3.4.2 Kommunikations-Modi des Plugins

Das Plugin kann grundsätzlich in zwei verschiedenen Betriebsmodi genutzt werden: einerseits im „Linked to WONDER“-, andererseits im „Standalone“-Modus.

#### „Linked to WONDER“

Der „Linked to WONDER“-Modus ist für den Fall vorgesehen, dass der Hostrechner des Anwenders mit einem WONDER-System verbunden ist. In diesem Fall werden sämtliche Nachrichten zur Steuerung virtueller Quellen an cWONDER geschickt. cWONDER leitet diese an alle relevanten Komponenten weiter. Unter anderem werden die Nachrichten auch an alle Empfänger des „Visual Streams“ weitergereicht. Sofern eine entsprechende Instanz des StreamMulticasters aktiv ist, leitet dieser den „Visual Stream“ an die Multicast-Gruppe weiter, deren Nachrichten alle Plugins empfangen. So erreicht eine ausgehende Nachricht also zunächst cWONDER und sodann alle anderen Plugin-Instanzen.

Neu instanziierte Plugins senden in diesem Modus eine */WONDER/stream/visual/connect*-Nachricht an den StreamMulticaster. Dieser sendet daraufhin selbst eine */WONDER/stream/visual/connect*-Nachricht an cWONDER. cWONDER antwortet darauf mit dem (erneuten) Senden der Nachrichten, die es als Antwort auf eine neue Verbindung zum „Visual Stream“ verschickt (s.o.), welche wiederum vom StreamMulticaster an die Multicast-Gruppe weiter geleitet werden. So erhält die neue Plugin-Instanz schließlich alle Informationen über den aktuellen Zustand des WONDER-Systems.

Abbildung 3.2 zeigt diesen Informationsfluss des „Linked to WONDER“-Modus als grobe Übersicht.

[11] siehe Datei *OSC_and_Commandline_Overview.ods* im Verzeichnis *documentation* des Quellcode-Paketes von WONDER-Version 3.1.0

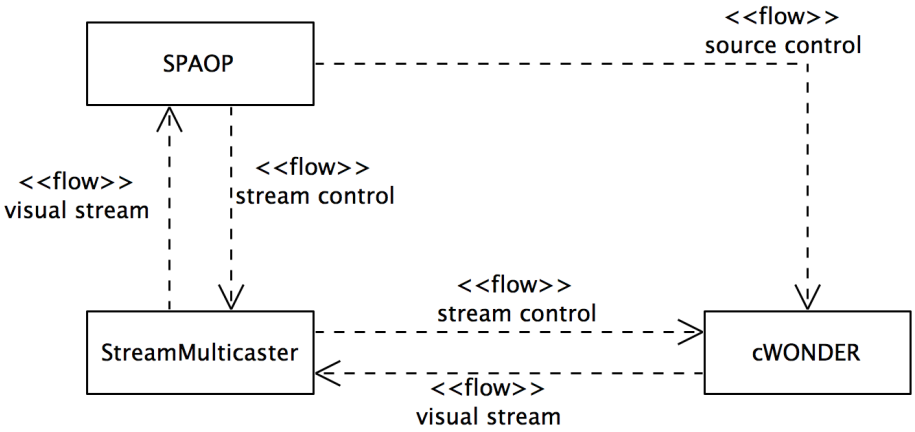


Abbildung 3.2: Informationsfluss im „Linked to WONDER“-Modus

### „Standalone“

Der „Standalone“-Modus dagegen ist für den Fall gedacht, dass der Hostrechner nicht mit einem WONDER-System verbunden ist. Damit trotzdem jede SPAOP-Instanz die Positionen aller virtuellen Quellen anzeigen kann, werden in diesem Fall alle Nachrichten direkt an die Multicast-Gruppe versandt und erreichen somit alle Plugin-Instanzen ohne den Umweg über cWONDER.

Im „Standalone“-Modus senden neu instanziierte Plugins auch ihre */WONDER/stream/visual/connect*-Nachricht an die Multicast-Gruppe. Die anderen Plugins antworten daraufhin, indem sie jeweils den Zustand der von ihnen kontrollierten Quelle (Typ, Position, Winkel, Farbe, DopplerEffekt-Einstellung, Name) mittels der entsprechenden Nachrichten schicken. So ist sichergestellt, dass ein neu instanziiertes Plugin auch im „Standalone“-Modus umgehend über den aktuellen Zustand aller Quellen informiert ist.

Eine Gesamtübersicht aller vom Plugin versendeten OSC-Nachrichten sowie der jeweiligen Adressaten in den beiden Modi ist in Tabelle 3.2 aufgeführt. Zwei Sonderfälle stellen dabei die Nachrichten */WONDER/stream/visual/pong* und */WONDER/plugin/standalone* dar. */WONDER/stream/visual/pong* wird unabhängig vom Betriebsmodus immer an den Absender der entsprechenden „Ping“-Nachricht gesendet. Die neu eingeführte Nachricht */WONDER/plugin/standalone* wird im Folgenden näher erläutert.

| *OSC-Pfad* | *Parameter* | *Empfänger im Linked-Modus* | *Empfänger im Standalone-Modus* |
|---|---|---|---|
| /WONDER/source/activate | int sourceID | MC-Gruppe | cWONDER |
| /WONDER/source/deactivate | int sourceID | MC-Gruppe | cWONDER |
| /WONDER/source/type | int sourceID, int type | MC-Gruppe | cWONDER |
| /WONDER/source/position | int sourceID, float x, float y | MC-Gruppe | cWONDER |
| /WONDER/source/angle | int sourceID, float angle | MC-Gruppe | cWONDER |
| /WONDER/source/name | int sourceID, string name | MC-Gruppe | cWONDER |
| /WONDER/source/color | int sourceID, int r, int g, int b | MC-Gruppe | cWONDER |
| /WONDER/source/dopplerEffect | int sourceID, int on | MC-Gruppe | cWONDER |
| /WONDER/stream/visual /connect | string name | MC-Gruppe | StreamMulticaster |
| /WONDER/stream/visual /disconnect | - | MC-Gruppe | StreamMulticaster |
| /WONDER/stream/visual/pong | int pingCount | Ping-Absender | Ping-Absender |
| /WONDER/plugin/standalone | int standAlone | MC-Gruppe | MC-Gruppe |

Tabelle 3.2: Vom Plugin versendete OSC-Nachrichten und ihre Adressaten in den Modi „Linked to WONDER“ und „Standalone“

### Umschalten des Kommunikationsmodus

Mit der Nachricht */WONDER/plugin/standalone* lässt sich der Kommunikationsmodus global umschalten. Im Gegensatz zu allen anderen genutzten OSC-Nachrichten, die bereits vor dieser Arbeit für das WONDER-System definiert wurden, wurde diese Nachricht für diesen Zweck neu eingeführt. Wird an einem Plugin der Kommunikationsmodus über die Benutzeroberfläche umgeschaltet, wird diese Nachricht an die Multicast-Gruppe gesendet. Sie sorgt dafür, dass alle anderen Plugin-Instanzen automatisch ihren Modus entsprechend umschalten.

Um sicherzustellen, dass alle Kommunikations-Teilnehmer nach einem derartigen Wechsel über den aktuellen Zustand der Quellen informiert sind, sendet jedes Plugin bei jedem Wechsel alle relevanten Informationen über den Zustand der von dem jeweiligen Plugin kontrollierten Quelle.

## 3.5 Architektur des Plugins

Das folgende Kapitel schildert den Aufbau des Plugins in seinen wesentlichen Details. Für ausführliche Beschreibungen aller Klassen und ihrer Methoden sei ergänzend auf die Doxygen-Dokumentation verwiesen.

### 3.5.1 Komponenten-Entwurf

Die Architektur des Plugins ist in vier wesentliche Komponenten aufgeteilt:

1. *OSC-Messaging:* Die OSC-Messaging-Komponente enthält die Funktionalitäten zum Versenden und Empfangen von OSC-Nachrichten. Hierzu gehören auch geeignete Thread-Klassen sowie Callback-Mechanismen, um als Reaktion auf eingehende Nachrichten entsprechende Methodenaufrufe auslösen zu können.

2. *SourceController:* Der SourceController ist die zentrale Komponente: er verwaltet die Daten der virtuellen Quellen und beinhaltet die Logik zur Reaktion auf jegliche Steuersignale. Eingaben des Nutzers, vom Host übergebene Automationsdaten sowie eingehende OSC-Nachrichten werden hier verarbeitet.

3. *PluginProcessor:* Diese Komponente dient als Schnittstelle zur Host-Software. Hier werden alle Plugin-typischen Methoden implementiert, was in den meisten Fällen aber lediglich bedeutet, diese Aufrufe an den SourceController weiterzuleiten.

4. *PluginEditor:* Der PluginEditor ist die grafische Benutzeroberfläche des Plugins. Er stellt den Zustand aller Quellen dar und bietet dem Nutzer die Möglichkeit, die von der jeweiligen Plugin-Instanz kontrollierte Quelle zu steuern, den Kommunikationsmodus umzuschalten sowie die Zuordnung der Plugin-Instanz zu einer virtuellen Quelle zu konfigurieren.

Abbildung 3.3 zeigt die beschriebenen Komponenten und ihre Schnittstellen.

### 3.5.2 Entkopplung der Fremdbibliotheken

Ein Ziel des Softwareentwurfs war, möglichst große Teile der entwickelten Software unabhängig von den genutzten Fremdbibliotheken wiederverwenden zu können. Dazu wurden entsprechende Interfaces definiert und die Teile des Programmcodes, die von Fremdbibliotheken abhängen, in entsprechende Namespaces gegliedert.

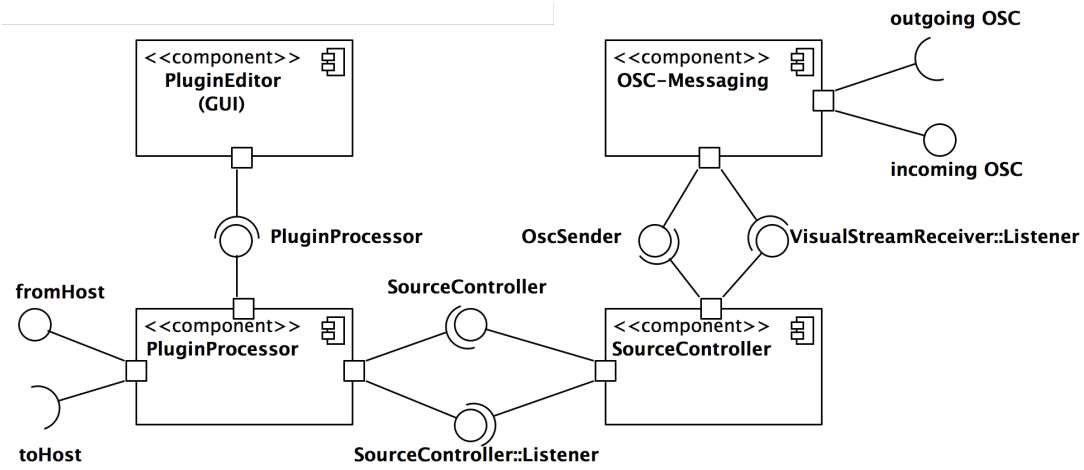


Abbildung 3.3: Architektur des Plugins

### Interfaces, Factories und Listener

Wo immer die Funktionalitäten einer fremden Bibliothek benötigt wurden, wurden Interfaces definiert, welche die genutzten Methoden beschreiben. Diese Interfaces ermöglichen es, im Bedarfsfall für die konkrete Implementierung der geforderten Funktionalitäten auch andere Bibliotheken zu verwenden. So könnten zum Beispiel die Klassen der OSC-Messaging-Komponente auch auf einer anderen OSC-Bibliothek aufgebaut und im Programmcode ausgetauscht werden, ohne dass am sonstigen Code etwas geändert werden müsste.

Eine Herausforderung bei diesem Ansatz ist, dass man keine neuen Objekte mittels des *new*-Operators erzeugen kann, wenn man von der konkreten Implementierung unabhängig sein möchte. Als Lösung wurden gemäß dem Abstract-Factory-Pattern [GHJV94, S. 87 ff.] entsprechende Factory-Methoden definiert, mit deren Hilfe neue Objekte des jeweiligen Interface-Typen erzeugt werden können. In der Regel sind diese in dem jeweiligen Interface als Methode einer inneren Klasse (genauer gesagt: eines inneren Interfaces) *Factory* definiert. Sonderfälle sind die Interfaces *OscSender* und *OscSenderThread*: Objekte dieser Klassen können über Factory-Methoden erzeugt werden, die im Interface *VisualStreamReceiver* enthalten sind. Die Factory-Objekte werden schließlich über die Konstruktoren an all diejenigen Klassen übergeben, in denen neue Objekte des jeweiligen Typs erzeugt werden müssen.

Eine weitere Art von innerer Klasse, die in einigen dieser Interfaces enthalten ist, ist die Listener-Klasse: sie definiert Callback-Methoden, die von der äußeren Klasse aufgerufen werden sollen. So kann man zum Beispiel bei einem *VisualStreamReceiver*-Objekt ein *VisualStream-*

*Receiver::Listener*-Objekt registrieren, um bei eingehenden OSC-Nachrichten entsprechende Aufrufe zu empfangen.

| *Interface* | *Verwendungszweck* |
|---|---|
| wonder::OscSender | Versand der OSC-Nachrichten |
| wonder::OscSenderThread | Versand der OSC-Nachrichten mittels eigenem Thread |
| wonder::VisualStreamReceiver | Empfang von OSC-Nachrichten des „Visual Streams“ |
| wonder::ConnectionTimer | Zyklischer Timer (zur Ping-Überwachung) |
| wonder::XmlParser | XML-Parsing eingehender WONDER-Projekt-Daten |

Tabelle 3.3: Interfaces zur Entkopplung von Fremdbibliotheken und ihr jeweiliger Verwendungszweck

Die einheitliche Struktur und Benennung der beiden Arten von inneren Klassen dient dazu, anderen Entwicklern diese Konzepte leichter erkennbar zu machen. Wer zum Beispiel Sinn und Zweck von *ConnectionTimer::Factory* und *ConnectionTimer::Listener* verstanden hat, wird leicht erkennen, wozu *VisualStreamReceiver::Factory* und *VisualStreamReceiver::Listener* dienen.

Tabelle 3.3 gibt einen Überblick über die fünf Interfaces, die zur Entkopplung der genutzten Fremdbibliotheken definiert wurden.

### Namespaces

Der Programmcode wurde in mehrere Namespaces gegliedert. Neben einer Gliederung gemäß dem Verwendungszweck der unterschiedlichen Klassen erfolgte hierbei auch eine Unterteilung nach Abhängigkeit von Fremdbibliotheken.

| *Namespace* | *Kurzbeschreibung* | *Eingebundene Bibliothek* |
|---|---|---|
| lowrappers | C++-Wrapper für die Liblo-Bibliothek | Liblo |
| wonderlo | WONDER-spezifische OSC-Kommunikations-Klassen (auf Basis der Klassen aus lowrappers) | Liblo |
| thread | Allgemeine Thread-Klassen | - |
| wonder | Allgemeine, von Fremdbibliotheken unabhängige Logik-Klassen | - |
| wonderjuce | Verschiedene JUCE-basierte Implementationen von in Namespace wonder definierten Interfaces | JUCE |

Tabelle 3.4: Namespaces und ihre Abhängigkeiten von Fremdbibliotheken

Tabelle 3.4 gibt einen Überblick über diese Namespaces. Der Namespace *wonder* ist dabei als der Haupt-Namespace anzusehen, der die zentrale Programmlogik der *SourceController*-Komponente enthält.

### 3.5.3 OSC-Messaging-Klassen

Für das Versenden und Empfangen der WONDER-spezifischen OSC-Nachrichten wurden auf Grundlage der entsprechenden, im Namespace *wonder* definierten Interfaces verschiedene Liblo-basierte Klassen implementiert. Dabei war ein Ziel, unabhängig von der konkreten Plugin-Anwendung eine möglichst flexible, wiederverwendbare Sammlung an C++-Klassen für die WONDER-spezifische OSC-Kommunikaton zu schaffen, die auch anderen Projekten als Grundlage dienen kann. Diese Klassen sollen im Folgenden näher erläutert werden.

#### lowrappers

Um konsequent objektorientiert in C++ arbeiten zu können, wurden im Namespace *lowrappers* die Liblo-Funktionen zur OSC-Kommunikation in entsprechende Objekte gekapselt. Dabei wurde auf die Nutzung des Liblo-eigenen, POSIX-basierten OSC-Server-Threads *lo_server_thread* verzichtet und stattdessen ein Server-Thread auf Basis des mit C++11 eingeführten *std::thread* implementiert. Dieser verhält sich im Wesentlichen wie eine C++-Version des *lo_server_thread*, kann aber deutlich einfacher auf Windows portiert werden.

#### OscSender

Das Interface *wonder::OscSender* definiert eine Schnittstelle zum Versenden der OSC-Nachrichten. Neben einigen Netzwerkkommunikations-typischen Methoden (zum Beispiel zum Setzen der Ziel-Adresse oder des Time-To-Live-Wertes) findet sich hier eine Vielzahl von Methoden, die jeweils dem Versenden genau einer der WONDER-spezifischen Nachrichten dienen.

Die davon abgeleitete Klasse *wonderlo::WonderOscSender* implementiert all diese Methoden auf Basis der Klasse *lowrappers::Address*.

In *wonderlo::WonderOscSender* erfolgt die Umsetzung von Sende-Methodenaufrufen in die entsprechenden OSC-typischen Aufrufe mit ihrem jeweiligen Pfad und den weiteren Parametern. Dabei werden diese Aufrufe nicht direkt an eine der Sende-Methoden von *lowrappers::Address* weiter gereicht. Stattdessen wird zunächst ein entsprechendes Nachrichten-Objekt erzeugt und an die Methode *wonderlo::WonderOscSender::wonderSend* übergeben.

Erst von dieser zentralen Sende-Methode aus wird die jeweilige Nachricht dann mittels *lowrappers::Address::send* endgültig versendet. Dadurch wird es ermöglicht, in abgeleiteten

Klassen durch Überschreiben einer einzigen Methode den Ablauf des Nachrichten-Versendens neu zu definieren, ohne die Umsetzung von Methodenaufrufen in OSC-Pfade erneut vornehmen zu müssen.

Sollte also eine neu eingeführte OSC-Nachricht den verschiedenen Versende-Klassen hinzugefügt werden, so muss diese Änderung nur an zwei Stellen vorgenommen werden: zum einen muss die Methode im Interface *wonder::OscSender* deklariert werden, zum anderen muss sie in *wonderlo::WonderOscSender* derart implementiert werden, dass sie ein entsprechendes Nachrichten-Objekt an *wonderlo::WonderOscSender::wonderSend* übergibt.

Eine abgeleitete Klasse, die diesen Vererbungsmechanismus nutzt, ist die Klasse *wonderlo::WonderOscServerSender*: sie versendet alle Nachrichten von dem Port eines ihr im Konstruktor übergebenen OSC-Server-Threads (während *wonderlo::WonderOscSender* die Nachrichten von einem beliebigen freien Port sendet). Weitere entsprechend abgeleitete Klassen sind die Sende-Threads, die im folgenden Abschnitt genauer erläutert werden.

### OscSenderThread

Das Versenden von Nachrichten kann im Plugin von verschiedenen Threads ausgelöst werden. Dies können auch Threads sein, die für das Echtzeitverhalten der Host-DAW-Software kritisch sind. So werden zum Beispiel die Automationsdaten bei VST3-Hosts von einem Thread übergeben, der auch für die Echtzeitverarbeitung der Audiosignale zuständig ist – üblicherweise dienen die Automationsdaten schließlich als Parameter für die Audioverarbeitung. Das Versenden von Netzwerknachrichten dagegen ist ein Zugriff auf Systemressourcen, bei dem nicht auszuschließen ist, dass der zuständige Thread womöglich auf diese Ressource warten muss. Daher ist es geboten, dass der Nachrichtenversand nicht von einem echtzeitkritischen Thread übernommen wird. Stattdessen muss diese Aufgabe an einen eigenen Thread delegiert werden.

Zu diesem Zweck wurde das Interface *wonder::OscSender* in einer abgeleiteten Variante *wonder::OscSenderThread* um die Thread-typischen Funktionen erweitert. Entsprechende Liblo-basierte Klassen wurden dann wiederum im Namespace *wonderlo* implementiert.

Diese überschreiben die von *wonderlo::WonderOscSender* geerbten *wonderSend*-Methoden so, dass die übergebene Nachricht in eine Queue eingereiht wird. Aus dieser Queue wird sie dann von einem eigenen Thread übernommen, der sie schließlich versendet. Auch die Art und Weise, wie der Thread die Nachricht versendet, kann durch Überschreiben einer Methode *wonderlo::WonderOscSenderThread::sendMethod* in abgeleiteten Klassen geändert werden. So gibt es wiederum eine Variante *wonderlo::WonderOscServerSenderThread* des Sender-Threads, die ihre Nachrichten von dem Port eines Server-Threads verschickt, statt einen beliebigen freien Port zu nutzen.

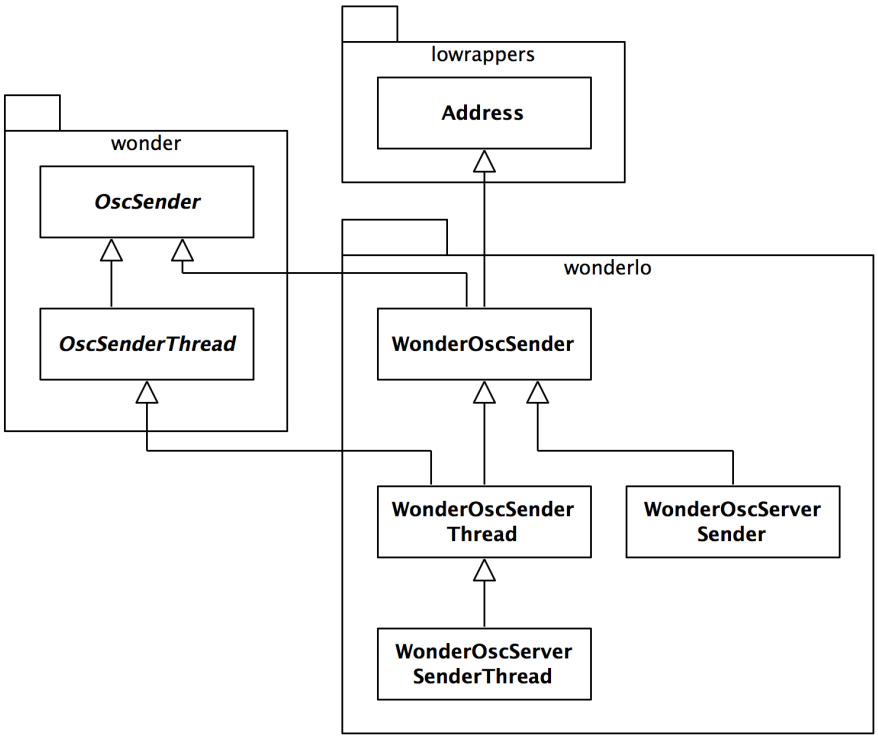


Abbildung 3.4: Überblick über die von *wonder::OscSender* abgeleiteten Klassen

Somit wurden insgesamt vier verschiedene Liblo-basierte Klassen entwickelt, die das *wonder::OscSender*-Interface implementieren. Eine Übersicht dieser Klassen bietet Abbildung 3.4.

### VisualStreamReceiver

Die Schnittstellen für einen OSC-Server-Thread, der die eingehenden OSC-Nachrichten empfängt und in geeignete Funktionsaufrufe umsetzt, wurden mit den Interfaces *wonder::VisualStreamReceiver*, *wonder::VisualStreamReceiver::Listener* und *wonder::VisualStreamReceiver::PingHandler* definiert.

Eine konkrete Implementierung des *wonder::VisualStreamReceiver*-Interfaces findet sich in *wonderlo::VSReceiver*: auf Basis des allgemeinen, Liblo-basierten Server-Threads *lowrappers::ServerThread* werden OSC-Nachrichten empfangen. Je eine Instanz von *wonder::VisualStreamReceiver::Listener*- und *wonder::VisualStreamReceiver::PingHandler*-Objekten kann dem *wonderlo::VSReceiver* übergeben werden, damit dieser dann bei eingehenden Nachrichten die entsprechenden Callback-Methoden aufrufen kann.

Das Interface *wonder::VisualStreamReceiver::PingHandler* nimmt dabei ausschließlich Aufrufe bei eingehenden */WONDER/stream/visual/ping*-Nachrichten entgegen, während das *wonder::VisualStreamReceiver::Listener*-Interface die Callback-Methoden zu allen anderen relevanten Nachrichten bietet.

#### PingControl

Zur Überwachung der eingehenden „Ping“ -Nachrichten wurde die Klasse *wonder::PingControl* entworfen. Ein zyklischer Timer muss hier mit jedem eingehenden Ping zurückgesetzt werden. Läuft der Timer ab, wird ein entsprechender Zähler erhöht. Überschreitet dieser Zähler einen Schwellwert (gesetzt mit dem Define *MAX_PINGS_LOST*), gilt die Verbindung als abgebrochen und ein entsprechender Listener *wonder::PingControl::Listener* wird benachrichtigt. Das Timer-Intervall muss mit dem Define *PING_TIMEOUT_INTERVAL* sinnvoll gesetzt werden.

Für den Timer selbst wurde das Interface *wonder::ConnectionTimer* entworfen. Eine konkrete, JUCE-basierte Implementierung des Interfaces findet sich in der Klasse *wonderjuce::JuceConnectionTimer*.

### 3.5.4 Anbindung an die Host-Software

Die Anbindung des Plugins an die Host-Software erfolgt in der Komponente PluginProcessor. Im Folgenden wird zunächst der allgemeine Aufbau dieser Komponente geschildert. Es folgen einige Details zur Übergabe von Automationsparametern zwischen Host und Plugin, bevor schließlich einige notwendige Modifikationen an der JUCE-Bibliothek und ihren Mechanismen zur Anbindung an VST3- und AU-Hosts erläutert werden.

#### PluginProcessor als Vermittler

Die Komponente PluginProcessor besteht im Wesentlichen aus der Klasse *SpaopAudioProcessor*, die von der JUCE-Klasse *juce::AudioProcessor* abgeleitet ist. Von *juce::AudioProcessor* erbt sie alle (teils abstrakten) Methoden, die benötigt werden, um mit den JUCE-eigenen Mechanismen die Plugins in den von JUCE unterstützten Formaten zu erstellen. Dies sind im Wesentlichen diverse Methoden zur Übergabe von Parametern oder von zu verarbeitenden Audio-Daten sowie Akzessoren für die aktuellen Parameter und für die grafische Benutzeroberfläche. Hinzu kommen Methoden zum Speichern des aktuellen Zustands des Plugins in einen vom Host übergebenen Speicherblock sowie zum Wiederherstellen eines Plugin-Zustands aus einem derartigen Speicherblock. Bei einem regulären Audio-Plugin wäre die *juce::AudioProcessor*-Klasse also typischerweise diejenige Klasse, in der die Audiosignalverarbeitung stattfindet.

Da das SPAOP-Plugin aber die Audiodaten unverändert lässt und die zentrale Programmlogik in der JUCE-unabhängigen Komponente SourceController enthalten ist, ist die Rolle der PluginProcessor-Komponente im vorliegenden Fall im Wesentlichen auf das Vermitteln zwischen Host, GUI und SourceController reduziert. Der PluginProcessor reicht die Automations-Parameter-spezifischen Aufrufe an den SourceController weiter und macht den SourceController für die GUI-Komponente verfügbar.

Lediglich das Abspeichern des aktuellen Zustands des Plugins in einen vom Host übergebenen Speicherblock sowie das Rekonstruieren eines solchen gespeicherten Zustands erfolgt in dieser Komponente selbst.

#### Speichern und Rekonstruieren von Plugin-Zuständen

Die Möglichkeit, den Zustand des Plugins zu speichern und zu rekonstruieren erfolgt unter Zuhilfenahme der JUCE-eigenen XML-Funktionalitäten. Die zu speichernden Parameter werden in das XML-Format umgesetzt und in den vom Host übergebenen Speicherblock geschrieben. Zum Wiederherstellen eines solchen Zustandes können sie dann wieder ausgelesen und die entsprechenden Parameter auf den ausgelesenen Wert gesetzt werden. Zu diesem Zweck finden sich einige Hilfsfunktionen zentral gebündelt in der Klasse *wonderjuce::XmlFactory*.

Gespeichert werden dabei sämtliche Parameter, die vom Nutzer beeinflusst werden können, also sowohl die lokal gespeicherten Daten über den Zustand des WONDER-Systems als auch der aktuelle Kommunikationsmodus sowie der Zustand der grafischen Benutzeroberfläche.

#### Automatisierbare Parameter

Zur Übergabe von Parameterwerten vom Host an das Plugin und vom Plugin an diesen zurück arbeitet die *juce::AudioProcessor*-Klasse mit einem Index-System. Jedem Parameter ist eine fortlaufende natürliche Zahl zugeordnet. Bei $N$ Parametern sind diese also über die Indizes 0 bis $N - 1$ adressierbar. Die Zuordnung dieser Zahlen zu den konkreten Parametern des vorliegenden Plugins ist in dem Enum *wonder::Source::AutomatedParameters* definiert.

Automatisiert werden können alle Parameter einer einzelnen virtuellen Quelle: der Typ, die Position (x- und y-Koordinaten), der Abstrahlwinkel sowie die Doppler-Effekt-Einstellung. Da das Plugin das Konzept verfolgt, jeder Audio-Spur genau eine virtuelle Quelle zuzuordnen, ist die Zuordnung des Plugins zu einer Quellen-ID nicht automatisierbar. Ebenfalls nicht automatisierbar sind vorerst sämtliche Parameter einer Quelle, die der Steuerung von Quell-Gruppen dienen (also: die Gruppen-ID, die Rotations-Richtung sowie die Skalierungs-Richtung). Dies könnte gegebenenfalls ergänzt werden, sobald ein sinnvolles Konzept zur Gruppierung von virtuellen Quellen besteht.

### Parameter-Skalierung

Die Übergabe der Parameterwerte zwischen Host und Plugin erfolgt bei der *juce::AudioProcessor*-Klasse mittels auf das Intervall $[0.0; 1.0]$ normalisierter float-Werte. Das bedeutet, dass die Werte der Parameter entsprechend umgewandelt werden müssen.

Bei den Parametern für Winkel und Doppler-Effekt ist dies denkbar einfach: beim Winkel wird der Wertebereich $[0.0; 360.0]$ auf das entsprechende Intervall skaliert, beim Doppler-Effekt steht $1.0$ für „an“ und $0.0$ für „aus“.

Bei den Positions-Koordinaten ist dagegen nicht eindeutig, wie sie auf den Bereich $[0.0; 1.0]$ zu skalieren sind. Seitens WONDER könnten sie im gesamten float-Wertebereich, also im Intervall $[-\infty; \infty]$ liegen. Dieser Bereich lässt sich nicht sinnvoll auf das Intervall $[0.0; 1.0]$ abbilden. Der Wertebereich der Koordinaten muss also angemessen eingeschränkt werden. Neben mathematischen Überlegungen spielt dabei auch eine Rolle, wie fein aufgelöst die Automations-Daten bearbeitet werden sollen: die Benutzeroberflächen der verschiedenen DAW-Programme bieten üblicherweise die Möglichkeit, die Automationsdaten anhand eines Kurvenverlaufs zu manipulieren – je größer der Gesamt-Wertebereich eines Parameters ist, desto schwerer wird es, mit diesen Hilfsmitteln kleine Wertesprünge zu editieren.

An dieser Stelle muss also ein Kompromiss zwischen möglichst großem Wertebereich einerseits und fein aufgelösten Editier-Möglichkeiten andererseits eingegangen werden. Da es keine hinreichenden Erfahrungswerte gibt, in welchem Bereich Anwender die Quellen üblicherweise positionieren wollen, ist die Skalierung der Positions-Koordinaten nicht endgültig festgelegt. Mit den Defines *COORD_MIN* und *COORD_MAX* kann der Wertebereich der Koordinaten zum Zeitpunkt des Kompilierens auf das Intervall $[$*COORD_MIN* ; *COORD_MAX*$]$ eingestellt werden – die Skalierung erfolgt dann entsprechend. Die Default-Werte für diese Defines begrenzen die Koordinaten-Werte auf das Intervall $[-100.0; 100.0]$

Denkbar wäre auch, den Wertebereich der Koordinaten derart variabel zu halten, dass er im laufenden Betrieb des Plugins geändert werden kann. Diese Idee wurde jedoch verworfen, da dies ermöglichen würde, die Bedeutung aufgezeichneter Automationsdaten nachträglich zu verändern: je nach gesetztem Wertebereich würde die Skalierung der Koordinaten aus dem Automationsdaten-Intervall $[0.0; 1.0]$ auf absolute Koordinaten ein anderes Ergebnis liefern. Der Nutzer soll aber darauf vertrauen können, dass einmal erstellte Koordinaten-Verläufe die entsprechende Quelle beim Abspielen immer an die ursprünglich beabsichtigten Koordinaten positionieren. Daher erscheint die Festlegung des Wertebereichs zur Compile-Zeit am sinnvollsten.

#### Erweiterungen der JUCE-VST3- und AU-Wrapper

Im Verlauf der Entwicklung von SPAOP zeigte sich, dass zwei für die Bearbeitung von Automations-Daten innerhalb der Host-Software wesentliche Aufrufe der VST3- und AU-Schnittstellen von der JUCE-Bibliothek nicht angemessen unterstützt werden.

Beide Schnittstellen definieren jeweils eine Methode, bei der ein konkreter Wert eines Plugin-Parameters in eine entsprechende String-Repräsentation umgewandelt werden soll. Ohne korrekte Implementation dieser Methode kann ein Plugin zwar grundsätzlich funktionieren, jedoch werden in der Darstellung der Automations-Daten innerhalb der Host-DAW nicht die korrekten Parameterwerte angezeigt.

Entsprechend definieren beide Schnittstellen auch eine Methode, um einen String, der einen konkreten Plugin-Parameterwert darstellen soll, in den entsprechenden Fliesskomma-Wert umzuwandeln. Sollen die Möglichkeiten, Parameter des Plugins auch mittels der Benutzeroberfläche der DAW zu manipulieren, vollständig unterstützt werden, muss auch diese Methode angemessen implementiert werden.

Um diese Mängel der JUCE-Bibliothek zu beheben, wurden entsprechende Änderungen am JUCE-Quellcode vorgenommen. Einerseits musste die Klasse *juce::AudioProcessor* um Methoden ergänzt werden, die die fehlenden Funktionalitäten bieten. Andererseits wurden die VST3- und AU-Wrapper so angepasst, dass sie innerhalb der entsprechenden Methoden der Schnittstelle die neuen *juce::AudioProcessor*-Methoden aufrufen.

Für die Versionskontrolle dieser Änderungen wurde ein eigener Fork des JUCE-Repositories erstellt[12]. Dieser wiederum wurde als Submodul [Cha09, S. 152 ff.] in das SPAOP-Repository integriert. Dadurch können Aktualisierungen der offiziellen JUCE-Bibliothek weiterhin (durch Aktualisierung des Forks) in den geänderten JUCE-Quellcode übernommen werden. Durch die Integration als Submodul kann zudem in der Versionskontrolle des SPAOP-Repositories darauf verwiesen werden, welche Version des JUCE-Forks jeweils zu einer SPAOP-Version gehört.

### 3.5.5 Der SourceController

Die Komponente SourceController beinhaltet die wesentliche Programmlogik und die Datenverwaltung des Plugins. Sie besteht aus der zentralen Klasse *wonder::SourceController* und diversen weiteren Klassen, wie in Abbildung 3.5 in einem vereinfachten Klassendiagramm dargestellt ist. Über entsprechende Schnittstellen ist sie an die Komponenten OSC-Messaging und PluginProcessor angebunden. Der Aufbau der Komponente und die enthaltene Logik werden im Folgenden erläutert.

[12] https://github.com/MartinHH/JUCE

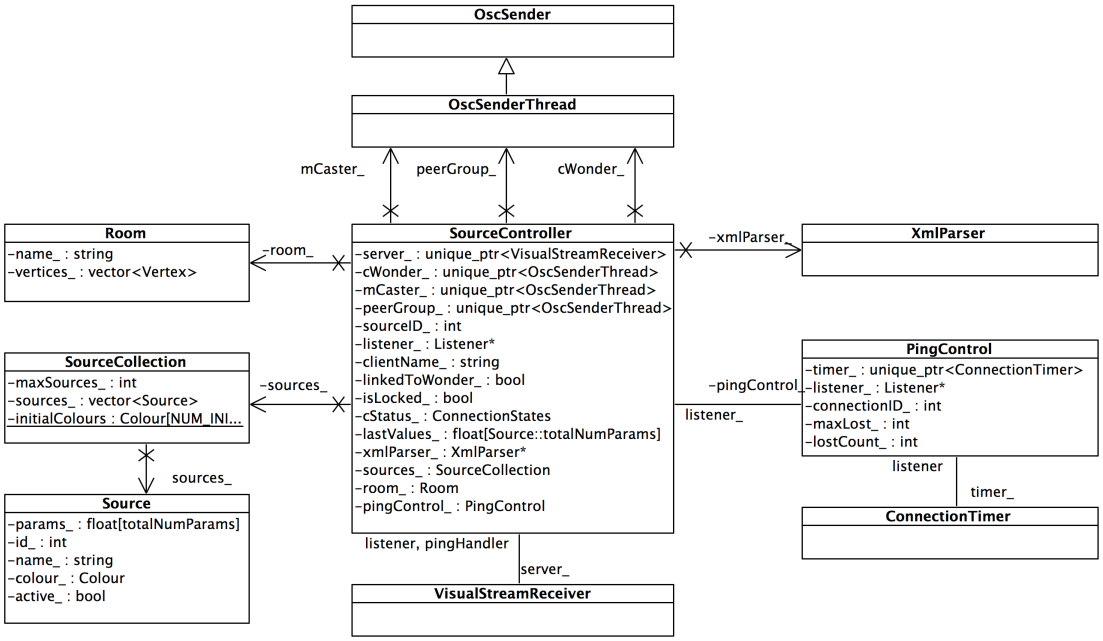


Abbildung 3.5: Vereinfachtes Klassendiagramm der Komponente SourceController
(Alle Klassen liegen im Namespace *wonder*. Klassen-Methoden und einige von *wonder::SourceController* implementierte Interfaces sind aus Gründen der Übersichtlichkeit nicht dargestellt.)

### Anbindung an das OSC-Messaging

Die Klasse *wonder::SourceController* ist direkt an das OSC-Messaging angebunden. Sie verfügt über drei *wonder::OscSender*-Threads: einen, um Nachrichten an cWONDER zu schicken, einen für Nachrichten an den StreamMulticaster und schließlich einen für die direkte Kommunikation mit der Multicast-Gruppe. Abhängig vom Kommunikationsmodus („Linked to WONDER“oder „Standalone“) wird dann beim Versenden von Nachrichten der jeweils passende Sende-Thread gewählt.

Zudem hält die *wonder::SourceController*-Klasse einen *wonder::VisualStreamReceiver*-Thread, der eingehende Nachrichten der Multicast-Gruppe empfängt und dann entsprechende Methoden der *wonder::SourceController*-Klasse aufruft, die die beiden Callback-Interfaces *wonder::VisualStreamReceiver::Listener* und *wonder::VisualStreamReceiver::PingHandler* implementiert.

### Anbindung an die PluginProcessor-Komponente

Die PluginProcessor-Komponente hält die *wonder::SourceController*-Instanz des Plugins und hat somit direkten Zugriff auf ihre Methoden.

Umgekehrt implementiert die *SpaopPluginProcessor*-Klasse das Interface *wonder::SourceController::Listener* und macht es dem *wonder::SourceController*-Objekt verfügbar. Über dieses Interface können via OSC eingehende Parameteränderungen an die PluginProcessor-Komponente gemeldet werden, die diese dann wiederum der Host-Software meldet.

### Aktivierung des Plugins

Da jede Plugin-Instanz für genau eine virtuelle Quelle verantwortlich ist, muss ihr die entsprechende ID dieser Quelle zugeordnet werden.

Wünschenswert wäre, dem Plugin anhand der Spur, in der es geladen ist, automatisch eine ID zuzuweisen. Leider bieten die genutzten Plugin-Schnittstellen keine Möglichkeit, vom Host entsprechende Informationen zu erhalten. Daher kann dies nur manuell im Plugin erfolgen.

Nach dem Laden einer neuen Plugin-Instanz muss also zunächst der Nutzer einstellen, für welche Quelle das Plugin zuständig ist. Um zu vermeiden, dass vorher schon ungewollte Änderungen an anderen Quellen stattfinden, muss diese Zuordnung entsprechend bestätigt werden. Hierfür ist ein Flag *isLocked_* vorgesehen: die ID darf nur geändert werden, wenn dieses Flag nicht gesetzt ist. Aber erst wenn dieses Flag gesetzt ist, werden lokale Änderungen an den Parametern der Quelle per OSC versendet und per OSC eingehende Parameteränderungen an die Host-Software gemeldet.

Beim Umschalten dieses Flags wird der Rest des Systems durch das Senden der entsprechenden Nachricht über die (De-)Aktivierung der Quelle benachrichtigt.

### Datenverwaltung

Die Parameter je einer virtuellen Quelle werden in Objekten der Klasse *wonder::Source* gespeichert. Die Speicherung derjenigen Parameter, die durch die Host-Software automatisierbar sein sollen, erfolgt dabei normalisiert auf den Wertebereich $[0; 1]$. Für alle Parameter gibt es passende Getter- und Setter-Methoden. Für die automatisierbaren Parameter gibt es zudem Akzessoren, mittels derer man auf diese Parameter über ihren in *wonder::Source::AutomatedParameters* definierten Parameter-Index zugreifen kann.

Jede Plugin-Instanz verfügt über die Daten aller Quellen. Dafür werden in der Klasse *wonder::SourceCollection* $N$ *wonder::Source*-Objekte vorgehalten, wobei $N$ über den Define *MAX_WONDER_SOURCES* konstant gesetzt ist. Die einzelnen *Source*-Objekte erhalten dabei

die IDs 0 bis $N-1$ und können somit über ihre ID mittels entsprechender Akzessor-Methoden der *wonder::SourceCollection*-Klasse angesprochen werden.

Die Klasse *wonder::SourceController* verfügt wiederum über solch ein *wonder::SourceCollection*-Objekt. Ihre Schnittstelle zur PluginProcessor-Komponente (also zur DAW-Host-Software und zur an die PluginProcessor-Komponente angebundenen GUI-Komponente) erlaubt schreibenden Zugriff nur für die eine Quelle, der das Plugin zugeordnet ist. Zugriff auf die *SourceCollection* ist auf diesem Weg nur lesend erlaubt. Somit können die anderen Quellen zwar in der GUI dargestellt werden – manipuliert werden kann aber nur eine Quelle pro Plugin-Instanz.

Neben den Daten der Quellen hält die Klasse *wonder::SourceController* auch noch ein *wonder::Room*-Objekt, in dem die Koordinaten des aktuellen Lautsprecher-Arrays gespeichert sind.

#### Verhalten bei eingehenden OSC-Nachrichten

Wenn eine OSC-Nachricht eingeht, die eine Zustandsänderung einer Quelle signalisiert, wird der entsprechende Parameter in der *SourceCollection* aktualisiert. Handelt es sich bei der aktualisierten Quelle um diejenige, die von der vorliegenden Plugin-Instanz kontrolliert wird, und handelt es sich um einen automatisierbaren Parameter, wird die Änderung zudem über die PluginProcessor-Komponente an den Host gemeldet. Einen Sonderfall stellt dabei die Nachricht */WONDER/source/deactivate* dar: wird diese für die Quelle empfangen, die vom jeweiligen Plugin selbst kontrolliert wird, wird sofort eine neue */WONDER/source/activate*-Nachricht geschickt. Schließlich gilt die Quelle so lange als aktiv, wie ein für sie zuständiges Plugin existiert.

Auf die übrigen eingehenden OSC-Nachrichten wird je nach Art der Nachricht angemessen reagiert: als Reaktion auf */WONDER/global/renderpolygon* wird das *wonder::Room*-Objekt aktualisiert, als Reaktion auf */WONDER/stream/visual/ping* wird die entsprechende */WONDER/stream/visual/pong*-Nachricht versendet, als Reaktion auf */WONDER/plugin/standalone* wird der Sendemodus entsprechend umgeschaltet.

#### Verhalten bei lokal ausgelösten Parameteränderungen

Wird dem SourceController eine Parameteränderung durch lokale Methodenaufrufe (also von der Benutzerschnittstelle oder von der Hostsoftware) mitgeteilt, so wird diese Änderung ebenfalls in der *SourceCollection* gespeichert.

Zudem wird die Änderung aber auch über OSC nach außen (also je nach Kommunikations-Modus an cWONDER oder an die Multicast-Gruppe) gemeldet. Um das System nicht unnötig mit Nachrichten zu überfluten, werden bei einigen Parametern Änderungen nur dann nach außen gemeldet, wenn die Differenz zum zuletzt gesendeten Wert eine gewisse Schwelle

überschreitet. Positionsänderungen werden nur gemeldet, wenn eine Änderung von mindestens 1 cm vorliegt; Änderungen des Abstrahlwinkels werden erst ab einer Änderung von mindestens 0,1° gemeldet. Diese Schwellwerte können über entsprechende Defines *COORD_PRECISION* und *ANGLE_PRECISION* geändert werden.

Denkbar wäre an dieser Stelle auch, das Aufkommen an ausgehenden Nachrichten nach zeitlichen Kriterien zu begrenzen, um das WONDER-System zusätzlich zu entlasten. Man könnte zum Beispiel ein Zeitintervall definieren, das nach dem Senden einer Positions-Nachricht mindestens vergangen sein muss, bevor die nächste Positions-Nachricht gesendet wird. Um diesen Ansatz sinnvoll umzusetzen, bedürfte es aber einer präzisen Analyse der Leistungsfähigkeit von WONDER: nur wenn genaue Informationen vorlägen, inwiefern eine solche Reduzierung der Nachrichten das Gesamtsystem entlasten würde und welche Faktoren dabei eine Rolle spielen, wäre eine entsprechende Begrenzung sinnvoll zu implementieren. Eine ausreichende Analyse dieser Faktoren war im zeitlichen Rahmen dieser Arbeit nicht möglich.

#### Reaktion auf ausbleibende Pings

Das Plugin reagiert nicht aktiv auf Probleme beim Empfang des „Visual Streams“. Die *wonder::SourceController*-Klasse bietet lediglich Informationen über den Verbindungsstatus in Form eines Enums *ConnectionStates* und dessen String-Repräsentation an, damit diese in der Benutzeroberfäche angezeigt werden können. Dabei gibt es drei verschiedene Zustände:

1. *inactive:* Ist das Plugin im Standalone-Modus, so ist die Verbindung „inactive“. Sie bleibt dies, bis sie nach Wechsel in den „Linked to WONDER“-Modus die erste „Ping“-Nachricht erhält.

2. *active:* Sobald im „Linked to WONDER“-Modus die erste „Ping“-Nachricht erhalten wurde, wechselt der Verbindungsstatus zu „active“.

3. *error:* Sind mehr als *MAX_PINGS_LOST* erwartete „Ping“-Nachrichten nicht eingetroffen, wechselt der Verbindungsstatus zu „error“.

### 3.5.6 Grafische Benutzeroberfläche

Mit Hilfe der JUCE-Bibliothek wurde eine grafische Benutzeroberfläche für das SPAOP-Plugin entwickelt. Im Folgenden wird das grafische Konzept der Oberfläche erläutert, bevor einige Details der Umsetzung geschildert werden.

### Grafisches Konzept

Mit xWONDER existiert eine etablierte grafische Benutzeroberfläche für WONDER. Um für Nutzer, die den Umgang mit xWONDER bereits gewohnt sind, die Bedienung von SPAOP möglichst einfach zu machen, orientiert sich der Entwurf der SPAOP-Oberfläche wesentlich an der xWONDER-Oberfläche.

Die Darstellung der virtuellen Quellen und der Lautsprecher-Arrays im zweidimensionalen Raum wurde gezielt xWONDER nachempfunden. Diese Darstellung findet sich im linken Teil der Benutzeroberfläche, wie in Abbildung 3.6 zu sehen ist. Da jedes Plugin nur eine einzige Quelle steuert, sind die übrigen Quellen (anders als in xWONDER) leicht transparent dargestellt, so dass sie vor dem schwarzen Hintergrund deutlich weniger hervorstechen als die Quelle, die dem Plugin zugeordnet ist. Als Ergänzung zu dem zweidimensionalen Panel zur Anzeige und Steuerung der Quellen-Position finden sich darunter Text-Felder, mittels derer konkrete Koordinatenwerte für die Quelle angezeigt und eingegeben werden können.

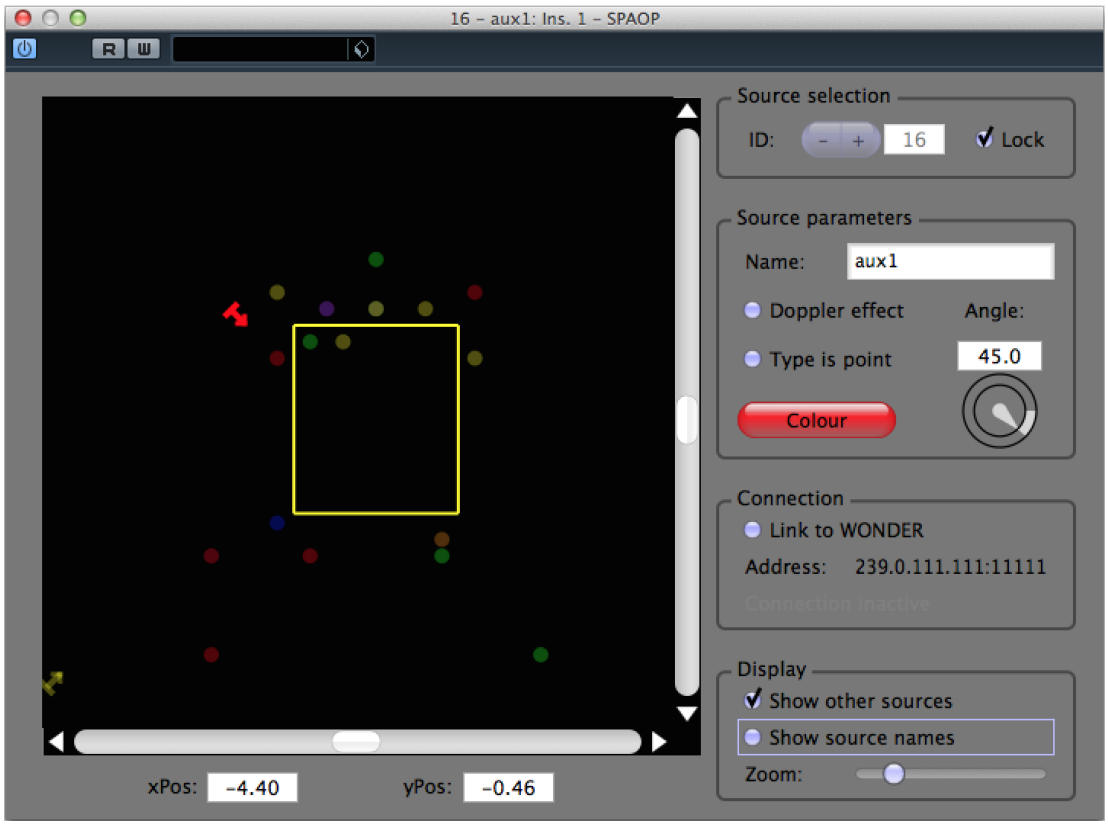


Abbildung 3.6: Grafische Benutzeroberfläche des Plugins

Der rechte Bereich der Benutzeroberfläche ermöglicht die Darstellung und Bedienung aller weiteren Parameter – er wurde hierfür mit entsprechenden Rahmen-Elementen in vier Bereiche unterteilt:

1. *Source selection:* Hier erfolgt die Zuordnung des Plugins zu einer konkreten virtuellen Quelle. Die entsprechende ID kann ausgewählt werden und die Auswahl mit einem entsprechenden Haken bestätigt werden. Ist der Haken gesetzt, ist das Plugin aktiviert – die ID kann dann nicht geändert werden, was durch die farblich abgeschwächte Darstellung der entsprechenden Bedienelemente verdeutlicht wird.

2. *Source parameters:* Bis auf die Positionierung erfolgt hier die Darstellung und Bedienung aller Parameter der vom Plugin kontrollierten Quelle. Der Name der Quelle, die Doppler-Effekt-Einstellung, ihr Typ, ihre Anzeigefarbe sowie ihr Abstrahlwinkel können hier manipuliert werden. Ist die Quelle eine Punkt-Quelle, sind die Bedienelemente für den Abstrahlwinkel deaktiviert.

3. *Connection:* In diesem Bereich kann der Sendemodus zwischen „Standalone“ und „Linked to WONDER“ umgeschaltet werden. Je nach Modus wird die Netzwerk-Adresse, zu der die Steuer-Nachrichten gesendet werden, angezeigt (also entweder die Adresse der Multicast-Gruppe oder die von cWONDER). Im „Linked to WONDER“-Modus wird darunter zudem der aus den eingehenden „Ping“-Nachrichten ermittelte Verbindungsstatus angezeigt.

4. *Display:* Im unteren rechten Bereich können schließlich Einstellungen für die Darstellung der Quell-Positionen im linken Bereich vorgenommen werden. Man kann die Darstellung der anderen, nicht vom Plugin kontrollierten Quellen sowie die Anzeige der Namen der Quellen ein- und ausschalten. Zudem kann man den dargestellten Bereich über einen Zoom-Faktor manipulieren.

#### Umsetzung mit Hilfe von JUCE

Die Benutzeroberfläche besteht größtenteils aus Standard-Komponenten der JUCE-Bibliothek. Daher konnte die Haupt-GUI-Klasse *SpaopPluginEditor* in wesentlichen Teilen mit dem „What you see is what you get“-Editor des *Introjucers* erstellt werden. Mit diesem können die einzelnen Komponenten in der *Introjucer*-Benutzeroberfläche arrangiert werden. Der *Introjucer* generiert dann automatisch den entsprechenden Quellcode, der sodann um die entsprechende Programm-Logik ergänzt werden muss.

Lediglich für die zweidimensionale Darstellung der Quell-Positionen mussten zwei eigene GUI-Komponenten entwickelt werden: die Klasse *wonderjuce::SourcePanel*, in der die eigent-

liche Darstellung der Quellen erfolgt, sowie die Klasse *wonderjuce::SourceZoomPort*, die ein *wonderjuce::SourcePanel* enthält und um die Möglichkeit zum Zoomen und Scrollen ergänzt.

Um möglichst viele Elemente des jeweiligen Quellcodes wiederverwendbar zu machen, wurde dabei jeweils eine Vererbungshierarchie entwickelt, die allgemeine Methoden von spezifischeren Funktionalitäten trennt. Die Klasse *wonderjuce::ZoomPort* enthält als Basis-Klasse von *wonderjuce::SourceZoomPort* allgemeine Funktionalitäten zum Vergrößern und Verkleinern von beliebigen GUI-Komponenten. Entsprechend bietet die Klasse *wonderjuce::SourceDisplay* als Basis-Klasse von *wonderjuce::SourcePanel* grundlegende Funktionalitäten zur Darstellung des Lautsprecher-Arrays und einzelner virtueller Quellen. Diese Basis-Klassen könnten dann zum Beispiel auch als Basis einer Benutzeroberfläche dienen, mittels der nicht nur eine, sondern alle Quellen des Systems gesteuert werden können.

#### Anbindung an die PluginProcessor-Komponente

Die *SpaopPluginEditor*-Klasse hat direkten Zugriff auf die PluginProcessor-Komponente. Durch entsprechende Akzessor-Methoden der *SpaopPluginProcessor*-Klasse hat sie zudem Zugriff auf die SourceController-Komponente und somit auch lesenden Zugriff auf das dort enthaltene *wonder::SourceCollection*-Objekt, das die Daten aller virtueller Quellen enthält.

Ausgelöst durch einen Timer aktualisiert sich die Benutzeroberfläche in regelmäßigen Abständen selbst anhand des Zustandes des SourceControllers sowie des *wonder::SourceCollection*-Objekts. Das Aktualisierungs-Intervall ist mittels des Define *GUI_REFRESH_TIME* auf $100ms$ gesetzt.

Sämtliche Bedienelemente der GUI melden Nutzereingaben zunächst an die *SpaopPluginEditor*-Klasse. Aktionen, die automatisierbare Parameter betreffen, werden von dort an die *SpaopPluginProcessor*-Klasse gemeldet, von wo aus sie entsprechend an den Host und die SourceController-Komponente weitergeleitet werden. Nutzeraktionen, die sonstige Parameter betreffen, werden direkt in Aufrufe an die SourceController-Komponente umgesetzt.

## 3.6 Der StreamMulticaster

Damit das SPAOP-Plugin Nachrichten von cWONDER über eine Multicast-Gruppe empfangen kann, wurde der StreamMulticaster entwickelt. Dabei handelt es sich um ein einfaches Programm, das die Nachrichten eines WONDER-OSC-Streams an eine beliebige Multicast-Gruppe weiterleitet. Da die Nachrichten für das Beziehen der Streams einer einheitlichen Semantik und Syntax folgen (alle Nachrichten haben das Format */WONDER/stream/<Stream-Name>/<Befehl>*), ist das Programm nicht auf das Weiterleiten des „Visual Streams“ beschränkt,

sondern kann jeden der vier WONDER-OSC-Streams weiterleiten. Hierfür muss lediglich der jeweilige Stream-Name als Parameter übergeben werden.

Das Programm besteht im Wesentlichen aus der Klasse *wonderlo::StreamMulticaster*. Sie enthält zwei OSC-Server-Threads, die auf zwei Ports auf eingehende Nachrichten warten. Der eine (*streamIn_*) empfängt den Stream und leitet alle empfangenen Nachrichten an die Multicast-Gruppe weiter. Damit die Verbindung erhalten bleibt, beantwortet er eingehende „Ping"-Nachrichten mit der entsprechenden „Pong"-Nachricht. Der andere Server-Thread (*connectIn_*) wartet auf */WONDER/stream/<Stream-Name>/connect*-Nachrichten. Sobald er eine empfängt, schickt er selbst eine */WONDER/stream/<Stream-Name>/connect*-Nachricht an cWONDER, um die entsprechenden Antworten auszulösen (die dann wiederum vom *streamIn_*-Thread an die Multicast-Gruppe weitergeleitet werden).

## 3.7 Test der Software

Ein wichtiger Teil der Entwicklung neuer Software sind die Tests des entwickelten Produkts. Neben einer Prüfung darauf, ob SPAOP und der StreamMulticaster ihre Aufgaben erwartungsgemäß erfüllen, wurden auch ein Hörtest sowie verschiedene Messungen durchgeführt, um die Übertragungsdauer der OSC-Nachrichten sowie das Verhalten des Plugins unter hoher Last zu evaluieren.

### 3.7.1 Funktionale Tests

Im Rahmen der funktionalen Tests wurde zunächst überprüft, ob das entwickelte Programm grundsätzlich so funktioniert, wie es geplant wurde. Da verscheidende Varianten für verschiedene Plugin-Schnittstellen entwickelt wurden, kamen für diese Tests eine Reihe unterschiedlicher Test-Konfigurationen in Frage.

#### Getestete Konfigurationen

Aufgrund des zeitlichen Rahmens dieser Arbeit konnten nicht alle denkbaren Konstellationen von Host-Software, Betriebssystem und Plugin-Schnittstelle ausführlich getestet werden. Während der Entwicklung diente eine Apple Macbook Pro (mit 2,8 GHz Intel Core i7 Prozessor, 16 GB RAM und Betriebssystem OSX 10.9.2) mit einer Vollversion von Cubase 5 als Referenz-Testumgebung, auf der regelmäßig die VST3-Version des Plugins getestet wurde. Diese Konfiguration ist dementsprechend diejenige, mit der SPAOP am intensivsten getestet wurde.

| *Test-Konfiguration* | | | *Geteste Funktionalitäten* | | | |
|---|---|---|---|---|---|---|
| *Betriebssystem* | *Host* | *Schnittstelle* | *GUI* | *OSC* | *Autom.* | *Presets* |
| OSX 10.9.2 | Cubase 5 | VST3 | i | i | i | i |
| OSX 10.9.2 | Cubase 5 | VST2 | o | o | o | o |
| OSX 10.9.2 | Cubase 7(32 Bit) | VST3 | o | i | i | o |
| OSX 10.9.2 | Cubase 7 (64 Bit) | VST3 | o | i | i | o |
| OSX 10.9.2 | Garageband 10 | AU | o | o | o | o |
| OSX 10.9.2 | Ardour 2.8.16 | AU | o | o | o | o |
| Windows 7 | Cubase 5 | VST3 | o | o | o | o |

Tabelle 3.5: Getestete Host- und Plugin-Konfigurationen sowie Test-Intensitäten („i“ steht für „intensiv“, „o“ für „oberflächlich“)

Darüber hinaus wurden verschiedene andere Kombinationen von Betriebssystemen, DAW-Host-Programmen und Plugin-Schnittstellen getestet, jedoch mit unterschiedlicher Intensität. Tabelle 3.5 gibt einen Überblick über die verschiedenen Test-Konfigurationen und die Intensitäten mit denen die unterschiedlichen Funktions-Bereiche des Plugins jeweils getestet wurden. Die Funktionsbereiche sind dabei unterteilt in die Bedienung über die Benutzeroberfläche (GUI), die OSC-Kommunikation (OSC), die Nutzung der Automationsmöglichkeiten des Hosts (Autom.) und das Speichern und Wiederherstellen von Plugin-Zuständen (Presets). Die Test-Intensität ist unterteilt in die Kategorien „intensiv“ und „oberflächlich“. „Oberflächlich“ bedeutet dabei, dass alle wesentlichen Funktionalitäten mindestens einmal auf erwartungsgemäßes Verhalten überprüft wurden. „Intensiv“ bedeutet dagegen, dass über einen längeren Zeitraum hinweg der praktische Einsatz des Plugins mit verschiedenen Bedien-Szenarien simuliert wurde.

Mit allen Test-Konfigurationen konnte festgestellt werden, dass die jeweils getestete Version des Plugins grundsätzlich funktionsfähig ist. Es wurden aber auch einige Mängel erkannt.

### Erkannte Mängel

Obwohl alle Versionen des Plugins grundsätzlich lauffähig sind, bestehen insbesondere bei den Versionen, die (im Gegensatz zu der VST3-Version) nicht schon während des Entwicklungsprozesses regelmäßig getestet wurden, noch einige Mängel:

- *Texteingabefeld:* Bei der VST2-Version in Cubase 5 (unter OSX) und der AU-Version in Ardour konnte Folgendes beobachtet werden: Klickt man mit der Maus in das Texteingabefeld für den Namen der Quelle, kann man dort keinen Text eingeben. Stattdessen werden durch die Tastatureingaben die entsprechenden Tastatur-Befehle der Host-Software

ausgeführt. Bezüglich der AU-Version konnte festgestellt werden, dass dieses Problem bei der Nutzung in GarageBand nicht auftritt.

- *Anzeige normalisierter Parameterwerte in Ardour:* Ardour zeigt an verschiedenen Stellen seiner Benutzeroberfläche die auf $[0.0; 1.0]$ normalisierten Parameterwerte an. So werden zum Beispiel beim Editieren der Automationskurve des Abstrahlwinkels nicht Werte zwischen $0.0$ und $360.0$, sondern zwischen $0.0$ und $1.0$ angezeigt. Es ist zu vermuten, dass Ardour die Methode der AU-Schnittstelle zur Umwandlung eines Parameterwertes in seine String-Repräsentation nicht nutzt und stattdessen direkt den jeweiligen Zahlenwert des Parameters darstellt.

- *Fehlerhafte Parameteranzeige bei der VST2-Version:* Bei der VST2-Version werden in Cubase 5 (unter OSX) die Parameterwerte beim Editieren von Automationsdaten nicht korrekt angezeigt. Die Vermutung liegt nahe, dass wie die AU- und VST3-Schnittstellen auch die VST2-Schnittstelle eine Methode definiert, die der Umwandlung von Parameterwerten in ihre String-Repräsentation dient und von JUCE nicht unterstützt wird. Sollte dies der Fall sein, könnte man den JUCE-VST-Wrapper entsprechend den in Kapitel 3.5.4 für AU und VST3 beschriebenen Maßnahmen anpassen. Da die Firma Steinberg inzwischen aber für ältere Versionen der VST-Schnittstelle keine Dokumentation mehr bereitstellt, kann dies nicht mit vertretbarem Aufwand durchgeführt werden.

- *Zustands-Wiederherstellung in Cubase 5 (OSX):* In Cubase 5 unter OSX 10.9.2 kommt es bei der Wiederherstellung von gespeicherten Plugin-Zuständen immer wieder zu Speicherzugriffs-Fehlern, die zu einem kompletten Absturz von Cubase führen. Dieser Fehler ist der Art und Weise geschuldet, wie der VST3-Wrapper der JUCE-Bibliothek die entsprechenden Aufrufe der VST3-Schnittstelle umsetzt. Es handelt sich also um einen Mangel der JUCE-Bibliothek. Da im Laufe dieser Arbeit von den JUCE-Entwicklern mehrfach Änderungen an dem betreffenden Quellcode vorgenommen wurden, wurde davon abgesehen, diesen Mangel selbst zu beheben.

- *Ausbleibende Antworten von cWONDER bei erneutem Verbindungsaufbau:* Wie in Kapitel 3.4.1 beschrieben, liegt dem Entwurf des Plugins die Annahme zugrunde, dass cWONDER auf jede */WONDER/stream/visual/connect*-Nachricht mit dem Versenden einiger Nachrichten reagiert, die den aktuellen Zustand des Systems wiedergeben. Es zeigte sich, dass diese Annahme zur Zeit (WONDER-Version 3.1.0) nicht korrekt ist. Ist der jeweilige Stream-Empfänger bereits bei cWONDER registriert, werden diese Nachrichten nicht konsequent an ihn versendet. Vielmehr werden die Nachrichten an denjenigen

Stream-Empfänger versendet, der sich als letzter bei cWONDER für den Empfang des Streams angemeldet hat.

Im Regelfall wird zunächst das WONDER-System inklusive einer Instanz von xWONDER gestartet, bevor eine DAW mit SPAOP sowie der StreamMulticaster in Betrieb genommen werden. In diesem Fall hat der beschriebene Fehler keine bemerkbaren Auswirkungen. Startet man jedoch xWONDER nach SPAOP und dem StreamMulticaster, macht sich dies deutlich bemerkbar. Zum einen erhalten neu angemeldete SPAOP-Instanzen nicht alle für sie relevanten Nachrichten. Zum anderen erhält xWONDER nun diese Nachrichten, was seitens xWONDER zu unerwartetem Verhalten führt.

Ein Blick in den Quellcode von cWONDER (Methoden *Cwonder::visualStreamConnect* und *OSCStream::connect*) lässt vermuten, dass das beschriebene Verhalten so nicht beabsichtigt ist und hier ein Fehlverhalten seitens cWONDER vorliegt, das zu beheben ist.

#### Automations-Interpolation

Ein Problem, das schon während der ersten Tests früher Versionen des Plugins zu erkennen war, ist, dass Host-Programme beim Aufzeichnen von Automationsdaten vermeintlich irrelevante Parameteränderungen interpolieren. Dies macht sich insbesondere bei den Koordinaten der virtuellen Quellen bemerkbar: durch den insgesamt recht großen Wertebereich erscheinen Koordinaten-Änderungen von wenigen Metern, wenn diese auf die interne Darstellung im Intervall $[0.0; 1.0]$ skaliert wurden, als relativ gering. Im Ergebnis zeichnet keines der getesteten Host-Programme in seiner Standard-Konfiguration Bewegungen einer virtuellen Quelle mit ausreichender Präzision auf. Kurvenförmige Bewegungen werden häufig zu einer gradlinigen Bewegung reduziert; teilweise werden komplexe Bewegungsabläufe auf eine lineare Bewegung vom Start- zum Endpunkt der Bewegung reduziert.

Bei Cubase bietet die Vollversion des Programms die Möglichkeit, dieses Interpolationsverhalten über einen sogenannten „Reduktionsfaktor“ zu regeln [BBB$^+$12, S. 321]. Setzt man diesen Wert auf $0\%$, so werden die Automationsdaten nicht interpoliert und somit auch die Positionsänderungen korrekt aufgezeichnet.

Für VST3-Hosts, die eine derartige Einstellung nicht anbieten, konnte ein weiterer Ansatz zur Lösung dieses Problems gefunden werden. Über eine entsprechende Methode der VST3-Schnittstelle kann man dem Host mitteilen, ob der Parameter über einen kontinuierlichen Wertebereich verfügt (also jeder Wert im Intervall $[0.0; 1.0]$ möglich ist), oder ob der Wertebereich in eine feste Anzahl diskreter Schritte unterteilt ist. Unterteilt man den Wertebe-

reich eines Parameters in diskrete Schritte, werden zumindest in Cubase unabhängig von der „Reduktionsfaktor“-Einstellung sämtliche Änderungen dieses Parameters ohne Interpolation aufgezeichnet. Der Nachteil dieser Lösung ist allerdings, dass dadurch einige Möglichkeiten zum Editieren der Automationsspur entfallen. Daher wurde ein entsprechendes Compiler-Flag *SPAOP_DISCRETE* eingeführt. Damit kann beim Kompilieren festgelegt werden, ob die Koordinaten sowie der Abstrahlwinkel der Quellen über einen kontinuierlichen Wertebereich verfügen oder ob ihr Wertebereich in diskrete Schritte unterteilt wird.

Somit konnten zumindest für die verschiedenen Versionen von Cubase sinnvolle Lösungen für dieses Problem gefunden werden.

### 3.7.2 Verhältnis von OSC- und Audio-Latenz

Bei der Übertragung der Audiosignale von der DAW-Software über die WONDER-Rendering-Komponenten bis hin zu den Lautsprechern werden die Audiosignale mehrfach durch Pufferung verzögert. Dieser Umstand sorgt dafür, dass die Wiedergabe von Audiosignalen auf der WFS-Anlage im Labor der HAW mit einer deutlichen Latenz von derzeit etwa 110 ms erfolgt [Foh13].

Unter dieser Voraussetzung erscheint es interessant, inwieweit eine Positions-Änderung, die in der Host-DAW zeitgleich mit einem Audio-Ereignis erfolgt, tatsächlich auch zeitlich synchron mit der Wiedergabe des Audio-Ereignisses umgesetzt wird. Ein solches Verhalten wäre erwünscht – jedoch ist zu erwarten, dass Positions-Änderungen deutlich vor der Wiedergabe der ihnen zugeordneten Audio-Ereignisse umgesetzt werden, da die Übertragung der OSC-Nachrichten nicht entsprechend verzögert wird. Um einen Eindruck von dem zeitlichen Verhältnis zwischen der Umsetzung von Audiosignalen und OSC-Steuersignalen zu erhalten, wurde ein einfacher Hörtest durchgeführt.

#### Methodik

Für den Test wurde wiederum das oben beschriebene MacBook genutzt. Die Verbindungen zu den OSC- und Audio-Netzwerken erfolgten jeweils mit einem handelsüblichen Gigabit-Ethernet-Adapter der Firma Apple. Getestet wurde mit der VST3-Version von SPAOP – der exakte Stand des Quellcodes zum Zeitpunkt der Tests ist im Quellcode-Repository unter dem Tag *latency-test* festgehalten. Der Audio-Puffer der Host-Software war bei diesem Test auf eine Größe von 128 Samples eingestellt, die Sample-Frequenz lag bei 44,1 kHz.

In Cubase 5 wurde ein Projekt erstellt, bei dem einmal pro Sekunde für 100 ms ein Sinus-Testton abgespielt wird. Für diesen Testton wurde der Koordinatenverlauf mittels eines SPAOP-Plugins automatisiert. Die Automationsspur setzt für je zwei Testtöne die Koordinaten auf

$(3.0; -9.0)$, was im WFS-Labor der HAW bei Blick auf die Powerwall deutlich jenseits der linken Lautsprecher liegt. Für jeden dritten Testton wechselt die Position 600 ms vor dem Testton auf $(3.0; 9.0)$, also deutlich jenseits der rechten Lautsprecher. Der Wechsel zurück zu $(3.0; -9.0)$ erfolgt mit jedem Durchgang zu einem anderen Zeitpunkt: im ersten Durchgang ist dies 150 ms nach dem Ende des Testtons, in jedem weiteren Durchgang liegt dieser Zeitpunkt dann jeweils 10 ms früher. Ein Screenshot dieses Cubase-Projekts mit einem Ausschnitt der Automationsspur ist in Abbildung 3.7 zu sehen.

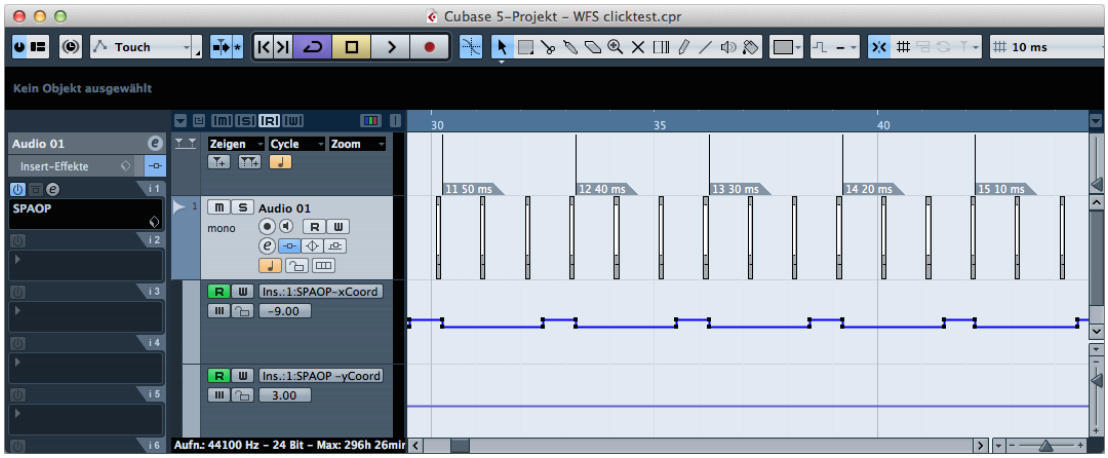


Abbildung 3.7: Screenshot der für den Latenzverhältnis-Test genutzten Automationsspur

Da zu erwarten war, dass die Positions-Änderungen deutlich schneller umgesetzt werden als die Audiosignale, sollte beim Abspielen dieses Projekts deutlich hörbar sein, ab welcher Verzögerung eine eigentlich nach dem Ende des Testtons eingeplante Positionsänderung dennoch vor dem Ende des Testtons ausgeführt wird.

### Beobachtungen und Analyse

Im Hörtest konnte in zehn Durchläufen gleichermaßen beobachtet werden, dass der rechts positionierte Testton bei Automations-Positionsänderungen bis inklusive 110 ms nach Ende des Testtons eindeutig von rechts, also von seiner designierten Position zu hören war. In den Fällen, wo die Positionsänderung für 100 oder weniger ms nach dem Testton eingeplant war, konnte man hören, wie die Position der virtuellen Quelle während der Wiedergabe des Testtons von rechts nach links wechselte. Es galt dabei zudem: je früher der Positionswechsel eingeplant war, desto größer war der Anteil des Testtons, der von links wiedergegeben wurde. Selbst wenn die Positionsänderung in der Automationsspur 20 ms vor dem Ende des Testtons eingeplant war, konnte man zu Beginn des Testtons noch ein leichtes Knacken von der rechten Seite

wahrnehmen. Bei Positionsänderungen, die 30 ms oder mehr vor dem Ende Testtons angesetzt waren, war der Testton schließlich ausschließlich von links zu hören.

Aus diesen Beobachtungen lässt sich schließen, dass die Umsetzung der Positionsänderungen in Relation zu den Audiosignalen ungefähr 80 bis 100 ms zu früh beziehungsweise deren Wiedergabe entsprechend zu spät erfolgt. Im Vergleich mit dem zuvor gemessenen Audio-Latenz-Wert von 110 ms ergibt sich der Eindruck, dass die Umsetzung der Positions-Änderungen mit einer relativ geringen Latenz erfolgt. Hierbei gilt es allerdings zu bedenken, dass der Wert von 110 ms mit einem anderen Audio-Zuspiel-Rechner gemessen wurde und insofern nur bedingt aussagekräftig ist.

Bemerkenswert erscheint darüber hinaus, dass es ein Zeitfenster von mindestens 120 ms gibt, innerhalb dessen eine Positionsänderung so ausgelöst werden kann, dass von beiden Positionen aus jeweils ein Teil des Testtons wiedergegeben wird. Da der Testton exakt 100 ms lang ist, wäre zu erwarten gewesen, dass dieses Zeitfenster ebenfalls nur 100 ms groß ist. Aufgrund der mehrfachen Durchführung des Tests kann ausgeschlossen werden, dass diese Beobachtung zufälligen Schwankungen der OSC-Nachrichten-Laufzeit geschuldet ist. Es ist zu vermuten, dass dieses Verhalten in der Art und Weise begründet ist, wie in tWONDER die Puffer für die Verzögerung der einzelnen Lautsprechersignale gefüllt und verwaltet werden. Unabhängig von der Ursache zeigt dieses Phänomen, dass die Positionierung der virtuellen Quellen selbst bei Berücksichtigung der unterschiedlichen Latenzen nur mit einer zeitlichen Präzision von ungefähr 20 ms erfolgen könnte, da das Rendering des WONDER-Systems nicht genauer arbeitet.

Um den festgestellten Versatz zwischen Umsetzung der Steuersignale und Wiedergabe der Audiosignale dennoch zumindest teilweise auszugleichen, könnte man dafür sorgen, dass die Verarbeitung der OSC-Nachrichten entsprechend verzögert wird. Dies könnte einerseits durch eine Verzögerung des Nachrichtenversands im SPAOP-Plugin erfolgen. Andererseits unterstützt WONDER theoretisch für Positions- und Winkel-Änderungen auch eine OSC-Nachricht, bei der als zusätzlicher Parameter ein Zeitintervall übermittelt wird, um welches die Ausführung der Parameteränderung durch cWONDER verzögert wird. Zur Nutzung dieser Nachrichten im WFS-Labor der HAW müsste dort zunächst die WONDER-Zeitbasis-Komponente jfWONDER in Betrieb genommen, die dort zum Zeitpunkt dieser Arbeit nicht in Betrieb war. Es erscheint jedoch insgesamt sinnvoller, die Bemühungen bezüglich der Optimierung von Latenzen eher darauf auszurichten, die hohe Audio-Latenz zu verringern, anstatt andere Latenzen zu erhöhen.

Erste praktische Erfahrungen anhand verschiedener Beispiel-Projekte zeigten zudem, dass der festgestellte Latenz-Unterschied beim Bearbeiten von WFS-Audio-Projekten keine wesentliche Einschränkung darstellt, sofern man sich der Problematik bewusst ist.

### 3.7.3 OSC-Übertragungsdauer

Anhand des beschriebenen Hörtest konnte eine grobe Einschätzung erfolgen, welche Verzögerungen bei der Übermittlung der OSC-Nachrichten auftreten können. Um genauer zu ergründen, wie groß diese Verzögerungen sein können, wurde die Übertragungsdauer der OSC-Nachrichten bei verschiedenen System-Konfigurationen gemessen.

#### Methodik

Als Test-Gegenstand wurde exemplarisch die Verbreitung der Positions-Nachrichten gemessen.

Der SPAOP-Quellcode wurde durch eine *wonder::PosMsgLogger*-Klasse ergänzt, die den Inhalt einer Positions-Nachricht mit einem Zeitstempel speichert und bei Aufruf ihres Destruktors die Messdaten in einer Log-Datei speichert. Mit dieser Klasse wurde einerseits der Zeitpunkt gemessen, an dem die Positionsdaten an den Sende-Thread übergeben werden. Andererseits wurde der Zeitpunkt gemessen, an dem der OSC-Server-Thread bei Eingang der Nachricht die entsprechende Callback-Methode des SourceControllers aufruft. Die für diesen Test genutzte Version des SPAOP-Quellcodes ist im Git-Repository unter dem Tag *rtt-test* zu finden.

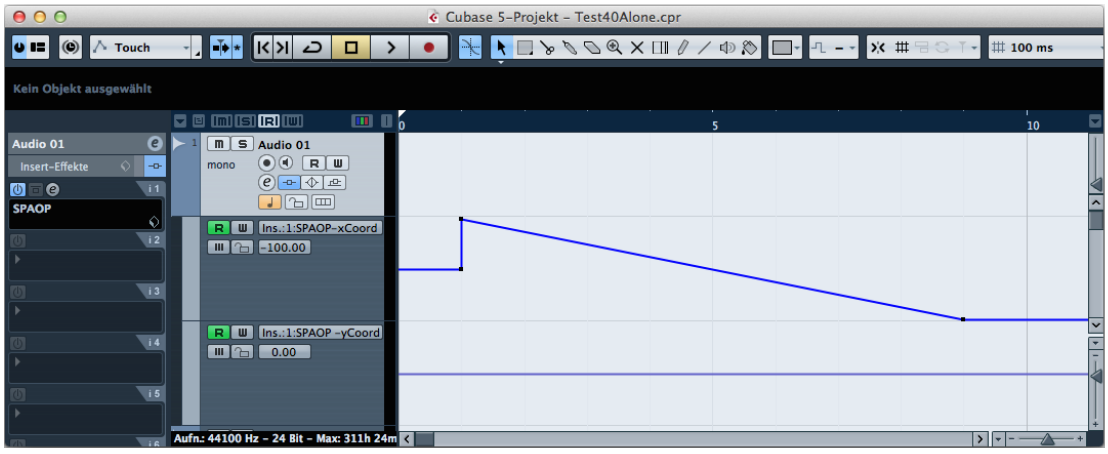


Abbildung 3.8: Screenshot der für die Übertragungsdauer-Tests genutzten Automationsspur

In einem Cubase-5-Projekt wurde eine Automationsspur angelegt, welche die x-Koordinaten einer Quelle innerhalb von acht Sekunden linear von ihrem Maximal-Wert zu ihrem Minimum verlaufen lässt. Abbildung 3.8 zeigt einen entsprechenden Screenshot des Cubase-Projekts.

Von diesem Projekt wurden drei Versionen angelegt. In jeder Version sendet nur ein Plug-in die automatisierten Positionsdaten. Insgesamt instanziiert wurde jedoch einmal nur ein Plugin, einmal 15 und einmal 40 Instanzen. Mit diesen Projekten wurden jeweils die Zeiten

im „Standalone“- und im „Linked to WONDER“-Modus gemessen, sodass insgesamt mit sechs verschiedenen Test-Konstellationen gemessen wurde.

Für die Messung des „Linked“-Modus wurde das WONDER-System an der HAW genutzt: die Nachrichten liefen also von dem Test-MacBook über einen Switch an den cWONDER-Rechner und von dort über den Switch zurück an das Test-Macbook, wo der StreamMulticaster sie schließlich an die Plugins weiterleitete. Die Audio-Puffergröße von Cubase war bei diesem Test auf 128 Samples eingestellt, die Sample-Frequenz lag bei 44,1 kHz.

### Auswertung

Durch die in jeder Nachricht unterschiedlichen Koordinaten lässt sich jeder gemessene Empfangs-Zeitpunkt eindeutig einem Sende-Zeitpunkt zuordnen. Da beide Messpunkte ihre Zeitstempel von der selben Systemuhr des selben Rechners beziehen, kann durch Differenzbildung die jeweilige Round-Trip-Time berechnet werden.

| *Sendemodus* | *1 Instanz* *Min. ; Max. (in $\mu s$)* | *15 Instanzen* *Min. ; Max. (in $\mu s$)* | *40 Instanzen* *Min. ; Max. (in $\mu s$)* |
|---|---|---|---|
| Standalone | 63 ; 198 | 71 ; 1233 | 97 ; 1943 |
| Linked | 453 ; 1137 | 471 ; 2098 | 497 ; 2153 |

Tabelle 3.6: Minima und Maxima der gemessenen Round-Trip-Times

Die Minimal- und Maximal-Werte der Round-Trip-Times der sechs Messungen sind in Tabelle 3.6 dargestellt. Abbildung 3.9 zeigt die entsprechenden Durchschnitts-Werte, Abbildung 3.10 zeigt ergänzend die relativen Häufigkeiten der Laufzeiten (aufgeteilt auf die Bereiche [0 ms ; 1 ms], (1 ms ; 2 ms] und (2 ms ; 3 ms]).

Es zeigt sich, dass die Laufzeiten der Nachrichten deutlich zunehmen, je mehr SPAOP-Instanzen gleichzeitig aktiv sind. Ebenfalls erkennt man, dass der Umweg der Nachrichten über cWONDER im „Linked to WONDER“-Modus die Gesamt-Laufzeit deutlich erhöht.

Jedoch liegen die Laufzeiten selbst bei 40 Plugin-Instanzen noch in einer Größenordnung, wie sie auch bei der sonstigen OSC-Kommunikation innerhalb des WONDER-System üblich ist. Da vergleichbare Verzögerungen und Laufzeit-Schwankungen zum Beispiel auch bei der Kommunikation zwischen cWONDER und den Rendering-Komponenten entstehen können, liegen die gemessenen Laufzeiten und Laufzeit-Schwankungen innerhalb eines Bereiches, der für die vorliegende Anwendung als akzeptabel betrachtet werden kann. Optimierungsmaßnahmen zur Verbesserung der Nachrichtenlaufzeit wären entsprechend nur dann sinnvoll, wenn auch das restliche WONDER-System diesbezüglich optimiert würde.

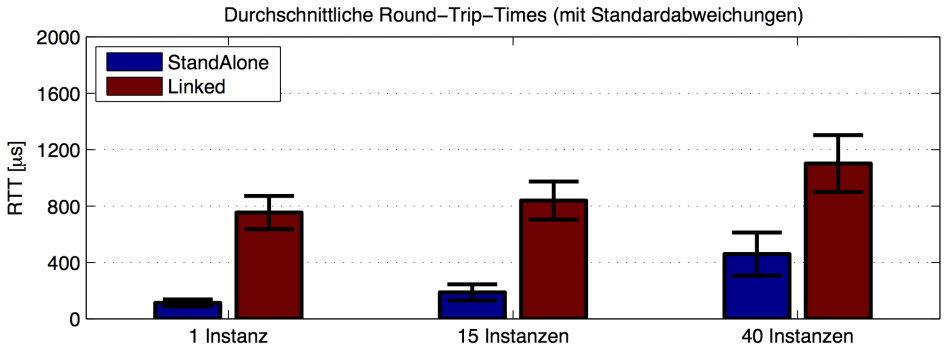


Abbildung 3.9: Durchschnittliche Round Trip Times der Positions-Nachrichten in den Modi „Standalone“ und „Linked to WONDER“ bei einer, 15 und 40 aktiven SPAOP-Instanzen

Betrachtet man darüber hinaus die Erkenntnis aus dem oben beschriebenen Hörtest, dass die Positionierung von virtuellen Quellen mittels WONDER ohnehin nur mit einer Präzision in der Größenordnung von etwa 20 ms möglich ist, erscheinen die gemessenen Laufzeiten für diesen Anwendungszweck vollkommen ausreichend.

### 3.7.4 Lasttests bei verschiedenen Puffergrößen

Bei DAW-Programmen wie Cubase lässt sich üblicherweise die Größe des Audiopuffers einstellen, der für die Übergabe der Audio-Daten an das Audio-Interface genutzt wird. In der Regel bestimmt diese Einstellung auch, wie groß die Audio-Daten-Blöcke sind, die jeweils den geladenen Plugins zur Verarbeitung übergeben werden. Hierdurch wiederum wird festgelegt, wie häufig ein Plugin aufgerufen wird und wie häufig der Host dabei Parameteränderungen an das Plugin meldet. Insofern bestimmt die Puffergröße im Falle des SPAOP-Plugins auch die maximale Frequenz, mit der Automations-Daten in ausgehende Nachrichten umgesetzt werden können.

Zudem legt die Größe des Audiopuffers fest, wie viel Spielraum die DAW-Software beim Scheduling der Plugin-Aufrufe hat. Je größer der Audiopuffer, desto größer ist der Zeitraum, in dem es der DAW theoretisch frei steht, wann genau die einzelnen Aufrufe an die einzelnen Plugins erfolgen.

Um zu untersuchen, welche Auswirkungen unterschiedliche Puffer-Größen auf die Leistungsfähigkeit des Plugins sowie auf das zeitliche Verhalten beim Nachrichtenversand haben, wurde daher das Plugin unter unterschiedlicher Last mit unterschiedlichen Audiopuffer-Einstellungen getestet.

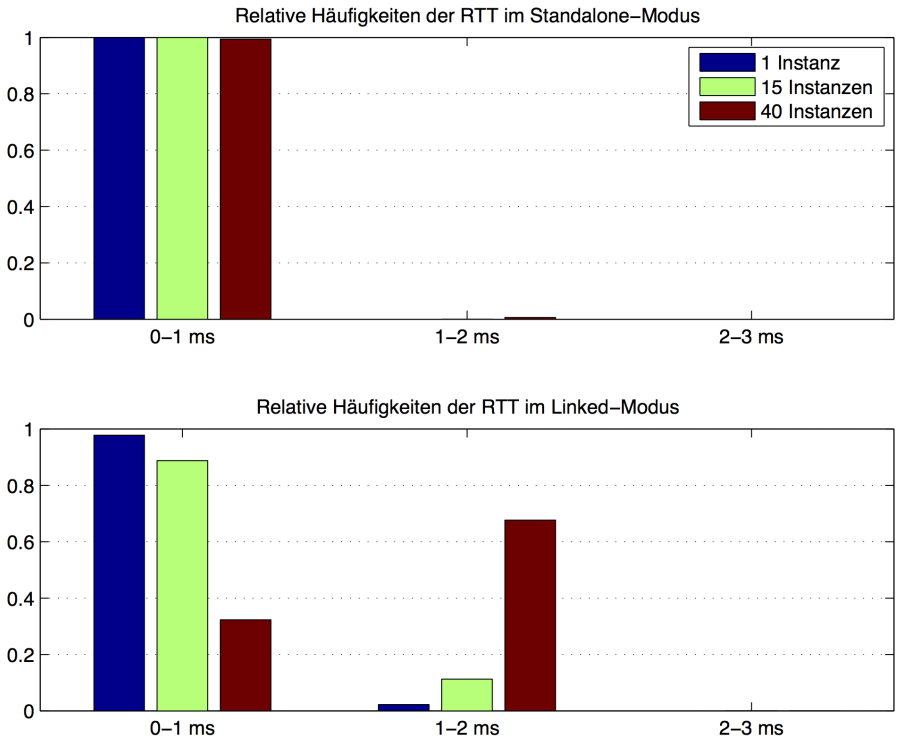


Abbildung 3.10: Relative Häufigkeiten der Round Trip Times der Positions-Nachrichten in den Modi „Standalone“ und „Linked to WONDER“ bei einer, 15 und 40 aktiven SPAOP-Instanzen

### Methodik

Auf dem Test-MacBook wurden in Cubase 7 drei Projekte angelegt, bei denen eine, 15 oder 40 SPAOP-Instanzen zeitgleich automatisiert den gleichen linearen x-Koordinaten-Verlauf vollziehen, der auch schon für die Tests der OSC-Übertragungsdauer verwendet wurde. Mit der *wonder::PosMsgLogger*-Klasse wurden die Zeitpunkte der Übergabe von Positions-Parametern an den Sende-Thread protokolliert. Die entsprechende Quellcode-Version findet sich im Git-Repository unter dem Tag *load-test*.

Die drei Test-Projekte wurden mit Puffergrößen-Einstellungen von 128, 256 und 512 Samples bei einer Sample-Frequenz von 44,1 kHz abgespielt. Dies erfolgte jeweils einmal im „Standalone“-Modus und einmal im „Linked to WONDER“-Modus. Bei letzterem war jedoch weder cWONDER noch der StreamMulticaster aktiv, sodass die Nachrichten nicht an die Plugins weiterge-

leitet wurden. Dadurch sollte getestet werden, welche Auswirkungen das Empfangen (oder Nicht-Empfangen) von Nachrichten auf das Gesamtverhalten hat.

### Auswertung

Bereits bei der Durchführung der Tests fiel auf, dass das Cubase-Projekt mit 40 Plugin-Instanzen bei Puffergrößen von 128 und 256 Samples nicht mehr ordnungsgemäß abspielbar war, sofern Nachrichten sowohl gesendet als auch empfangen wurden. Das Programm geriet deutlich ins Stocken, und der Positions-Zeiger sowie die Zeit-Anzeige machten abrupte Sprünge. Wurden keine Nachrichten empfangen (weil deren Weiterleitung unterbrochen war), waren derlei Probleme nicht bemerkbar.

Die beschriebenen Beobachtungen spiegeln sich auch in den Messdaten wider. Zur Analyse wurde aus jeweils zwei Sende-Zeitstempeln eines Plugins das dazwischenliegende Intervall berechnet. Tabelle 3.7 zeigt die aus diesen Intervallen jeweils berechneten Durchschnittswerte sowie die jeweils gemessenen Minimal- und Maximalwerte der Intervalle.

| *Nachrichten-Empfang* | *Anzahl Plugins* | *Puffer-Größe* | *Gemessener Durchschn.* | *Standard-Abw.* | *Gemessenes Minimum* | *Gemessenes Maximum* |
|---|---|---|---|---|---|---|
| Ja | 1 | 128 | 2903 | 93,0 | 2654 | 5776 |
| Nein | 1 | 128 | 2903 | 72,6 | 2718 | 5849 |
| Ja | 15 | 128 | 2903 | 82,6 | 2558 | 5952 |
| Nein | 15 | 128 | 2903 | 91,1 | 2599 | 5942 |
| Ja | 40 | 128 | 3296 | 8969,0 | 18 | 256535 |
| Nein | 40 | 128 | 2902 | 77,2 | 2601 | 3303 |
| Ja | 1 | 256 | 5809 | 159,0 | 5443 | 11628 |
| Nein | 1 | 256 | 5809 | 158,5 | 5583 | 11557 |
| Ja | 15 | 256 | 5805 | 73,1 | 5511 | 6097 |
| Nein | 15 | 256 | 5809 | 168,0 | 5527 | 11545 |
| Ja | 40 | 256 | 6104 | 4447,9 | 36 | 96723 |
| Nein | 40 | 256 | 5809 | 167,9 | 5124 | 11747 |
| Ja | 1 | 512 | 11610 | 100,3 | 11260 | 11952 |
| Nein | 1 | 512 | 11625 | 431,9 | 11158 | 23279 |
| Ja | 15 | 512 | 11610 | 85,5 | 11279 | 11929 |
| Nein | 15 | 512 | 11625 | 423,8 | 11307 | 23191 |
| Ja | 40 | 512 | 11625 | 430,8 | 11104 | 23246 |
| Nein | 40 | 512 | 11610 | 100,8 | 11037 | 12229 |

Tabelle 3.7: Gemessene Intervalle zwischen den einzelnen Parameterübergaben bei verschiedenen Puffergrößen (alle Angaben in $\mu s$)

Abgesehen von den beiden erwähnten Fällen (Nachrichtenempfang aktiv, 40 Instanzen, Puffergröße 128 oder 256 Samples) ergibt sich ein relativ einheitliches Bild: Die durchschnittlichen Intervalle entsprechen jeweils ungefähr den Puffergrößen (bei 44,1 kHz entsprechen 128 Samples einem Intervall von 2902,5 $\mu s$, 256 Samples entsprechen 5805 $\mu s$, 1024 Samples entsprechen 11610$\mu s$). Auch die Abweichungen vom Durchschnittsintervall liegen in einem Rahmen, in dem sie weitaus weniger ins Gewicht fallen als die zeitlichen Abweichungen durch Nachrichten-Laufzeiten, die im WONDER-System grundsätzlich immer in Kauf genommen werden müssen.

Auffällig ist allerdings, dass bei mehreren Test-Durchgängen ein maximales Intervall gemessen wurde, das ungefähr dem doppelten des erwarteten Wertes entspricht. Es entsteht der Eindruck, dass in diesen Fällen eine erwartete Parameterübergabe ausblieb. Bei genauerer Betrachtung der Testdaten zeigt sich, dass in jedem der betroffenen Test-Durchgänge dieser Fall pro Plugin-Instanz genau einmal eintrat. Dabei waren es pro Test-Durchgang für alle Instanzen die gleichen zwei Parameterübergaben, zwischen denen derart viel Zeit verging. Dieses Phänomen trat unabhängig von Puffergröße, Anzahl der Plugins und Nachrichtenempfangs-Konstellation auf. Daher ist zu vermuten, dass dieses Verhalten in dem internen Scheduling von Cubase begründet ist. Denkbar wäre zum Beispiel, dass an diesen Stellen Cubase abweichend von seinem üblichen Verhalten einen Plugin-Aufruf ausführte, bei dem ein Audio-Daten-Block übergeben wurde, der die doppelte Größe des eingestellten Audio-Puffers hatte. Eine Überprüfung dieser Vermutung oder eine sonstige weitergehende Untersuchung dieses Phänomens waren im zeitlichen Rahmen dieser Arbeit leider nicht möglich.

Bei den Testdaten für Tests mit mehreren Plugin-Instanzen konnte (mit Ausnahme der beiden Testfälle, bei denen das System insgesamt überlastet war) festgestellt werden, dass pro Testlauf allen Plugins die gleichen Koordinaten-Werte in der gleichen Reihenfolge übergeben wurden. Da die Automationsspuren aller Plugins identisch angelegt wurden, lassen sich dadurch die Zeitstempel der $N$ verschiedenen Plugins Sätzen von jeweils $N$ Parameterübergaben zuordnen, die aus Benutzersicht gleichzeitig stattgefunden haben sollten. Um zu analysieren, inwieweit diese Übergaben tatsächlich weitestgehend zeitgleich erfolgten, wurde bei den so gruppierten Zeitstempeln jeweils der größte und der kleinste gewählt und davon die Differenz gebildet. Die dadurch berechnete Zeitspanne gibt an, wie viel Zeit zwischen der ersten und der letzten Übergabe von vermeintlich zeitgleich an alle Plugins übergebenen Parametern tatsächlich vergangen ist.

Tabelle 3.8 zeigt die Durchschnittswerte, Minima und Maxima der so berechneten Werte. Auch hier zeigt sich: im Verhältnis zu den im WONDER-Gesamtsystem zu erwartenden Nachrichten-Laufzeitschwankungen sind die zeitlichen Abweichungen relativ gering.

| *Nachrichten-Empfang* | *Anzahl Plugins* | *Puffer-Größe* | *Gemessener Durchschn.* | *Standard-Abw.* | *Gemessenes Minimum* | *Gemessenes Maximum* |
|---|---|---|---|---|---|---|
| Ja | 15 | 128 | 33,9 | 35 | 1 | 222 |
| Nein | 15 | 128 | 62,0 | 39 | 2 | 233 |
| Ja | 40 | 128 | k.A. | k.A. | k.A. | k.A. |
| Nein | 40 | 128 | 55,7 | 45 | 4 | 261 |
| Ja | 15 | 256 | 160,4 | 73 | 9 | 425 |
| Nein | 15 | 256 | 111,6 | 59 | 2 | 366 |
| Ja | 40 | 256 | k.A. | k.A. | k.A. | k.A. |
| Nein | 40 | 256 | 128,6 | 107 | 6 | 921 |
| Ja | 15 | 512 | 193,6 | 85 | 4 | 357 |
| Nein | 15 | 512 | 188,2 | 88 | 1 | 338 |
| Ja | 40 | 512 | 65,2 | 91 | 4 | 614 |
| Nein | 40 | 512 | 163,9 | 104 | 10 | 693 |

Tabelle 3.8: Gemessene Zeitspannen von Übergaben gleicher Parameter an mehrere Plugins (alle Angaben in $\mu s$ – „k.A.“ bedeutet: „keine sinnvolle Angabe möglich“)

Insgesamt ergibt diese Testreihe einen gemischten Eindruck: Auf der einen Seite ist positiv zu bemerken, dass die Übergabe der Parameter vom Host an die Plugins im Regelfall relativ präzise zu den Zeitpunkten stattfindet, zu denen der Nutzer sie erwartet. Insbesondere erscheint diese Präzision gegenüber der festgestellten zeitlichen Präzision des WONDER-Renderings vollkommen ausreichend.

Auf der anderen Seite zeigen die Ergebnisse klare Grenzen der Leistungsfähigkeit des Plugins auf. Hier fällt insbesondere ins Gewicht, dass jede Plugin-Instanz jede gesendete Nachricht verarbeitet und der Aufwand mit vielen gleichzeitig sendenden Plugins quadratisch steigt (senden $N$ Plugins gleichzeitig eine Nachricht, müssen von diesen Plugins insgesamt $N^2$ Nachrichten empfangen und verarbeitet werden). Wäre das Ziel bei der Entwicklung dieses Plugins ausschließlich maximale Performanz, so wäre eine alternative Lösung in Betracht zu ziehen, bei der jede Plugin-Instanz nur die Nachrichten erhält, die die von ihr kontrollierte virtuelle Klangquelle betreffen. Dies hätte jedoch den Preis, dass einerseits die einzelnen Plugin-Instanzen nicht mehr die Positionen aller Quellen anzeigen könnten, andererseits eine entsprechend komplexere Programmlogik zur Nachrichten-Verteilung entwickelt werden müsste.

In Anbetracht der Tatsache, dass bei einer Puffergröße von 512 Samples auch mit 40 SPAOP-Instanzen ein problemloser Betrieb möglich ist, erscheint die vorliegende Lösung als ein gangbarer Kompromiss zwischen Performanz und Nutzungskomfort.

# 4 Zusammenfassung und Ausblick

Abschließend werden in diesem Kapitel die Ergebnisse dieser Arbeit zusammengefasst. Zudem werden einige Ansätze für mögliche, auf diesen Ergebnissen aufbauende Arbeiten erläutert.

## 4.1 Fazit

Im Rahmen dieser Arbeit wurde ein Werkzeug entwickelt, dass es in Verbindung mit bestehenden DAW-Systemen komfortabel ermöglicht, Inhalte für WONDER-basierte Wellenfeldsynthese-Anlagen zu produzieren. Dank der Integration in etablierte Software sollte es erfahrenen Anwendern der jeweiligen Host-Programme ohne großen Lernaufwand möglich sein, das Plugin zu nutzen.

Mit Hilfe der JUCE-Bibliothek und den in C++11 eingeführten Ergänzungen der C++-Standard-Bibliothek konnte erreicht werden, dass die entwickelte Software sowohl zu unterschiedlichen Betriebssystemen als auch zu verschiedenen Plugin-Schnittstellen kompatibel ist. Es zeigte sich allerdings, dass JUCE die einzelnen Schnittstellen nicht vollständig unterstützt und die Bibliothek an einigen Stellen entsprechend angepasst werden musste. Ebenso musste festgestellt werden, dass die JUCE-Bibliothek bei bestimmten Konfigurationen von Betriebssystem, Plugin-Schnittstelle und Host-Software nicht fehlerfrei funktioniert. Insofern hat sich diese Bibliothek einerseits als äußerst hilfreich erwiesen, andererseits aber auch für zusätzliche Herausforderungen gesorgt.

Durch die Verwendung von OSC-Multicast konnte bei einem insgesamt hohen Aufkommen an OSC-Nachrichten die Anzahl der erforderlichen Sendevorgänge deutlich reduziert werden. Es wurden allerdings auch klare Grenzen der Leistungsfähigkeit der Software erkannt, die durch die relative hohe Zahl an Nachrichten-Empfangsvorgängen bedingt sind. Diese Grenzen zeigen sich jedoch erst bei einer hohen Belastung des Gesamtsystems, von der anzunehmen ist, dass sie in der praktischen Anwendung nicht auftreten wird. Sollte sich in Zukunft zeigen, dass diese Annahme falsch ist, liegen konkrete Vorschläge vor, wie man das Nachrichtenaufkommen reduzieren und dadurch die Leistungsfähigkeit erhöhen könnte.

Über das SPAOP-Plugin hinaus ist dank des modularen Aufbaus und der ausführlichen Dokumentation der entwickelten Software zudem eine kleine Quellcode-Bibliothek entstanden,

die als Grundlage für viele weitere, C++-basierte Programme zur Steuerung des WONDER-Systems dienen könnte.

Die entwickelte Software wurde unter Version 3 der GNU General Public License [Fre07] veröffentlicht. Der Quellcode findet sich unter:

https://github.com/MartinHH/SPAOP

Da beabsichtigt ist, das Programm auch über diese Arbeit hinaus weiter zu entwickeln, ist der exakte Stand der Entwicklung bei Abschluss dieser Arbeit dort unter dem Tag *v0.8* festgehalten. Die mit Doxygen erstellte Dokumentation findet sich zudem unter:

https://MartinHH.github.io/SPAOP

## 4.2 Ausblick

Im Verlauf dieser Arbeit ergab sich eine Reihe von Ideen, wie das entwickelte Plugin um weitere Funktionalitäten ergänzt werden könnte und für welche Zwecke der entwickelte Quellcode sowie die gewonnenen Erkenntnisse über das Plugin hinaus verwendet werden könnten.

### 4.2.1 Optimierung des bestehenden Plugins

Neben der Behebung der kleineren, im Rahmen der Tests gefundenen Mängel und einer weitergehenden Überprüfung der Kompatibilität zu den weniger intensiv getesteten Systemkonfigurationen existieren verschiedene Ideen, wie das Plugin weiter verbessert werden könnte:

#### Konfiguration der Netzwerkadressen zur Laufzeit

Zur Zeit werden sämtliche Netzwerkadressen während des Kompilierens festgelegt. Wünschenswert wäre, wenn die Adressen von cWONDER oder auch die Multicast-Gruppe zur Laufzeit eingestellt werden könnten. Die Klassen zum Versenden von OSC-Nachrichten unterstützen bereits das Ändern der Zieladresse im laufenden Betrieb. Es müssten also lediglich die Benutzeroberfläche um entsprechende Konfigurationsmöglichkeiten ergänzt werden und die dort vorgenommenen Einstellungen entsprechend an die OSC-Sende-Threads weiter gereicht werden. Den größten Aufwand bei der Umsetzung dieser Änderungen würde vermutlich die erforderliche Überprüfung der Eingaben auf ihre Gültigkeit verursachen.

### Eigenständige Regulierung der maximalen Nachrichten-Frequenz

Die Lasttests mit verschiedenen Puffergrößen zeigten, dass die Leistungsfähigkeit des Plugins wesentlich davon abhängt, wie häufig Parameteränderungen mittels der entsprechenden OSC-Nachrichten versendet werden. Aktuell lässt sich dies nur beeinflussen, indem man in der genutzten Host-Software die Größe des genutzten Audio-Puffers ändert. Denkbar wäre, das Plugin um einen Timer zu ergänzen, der dafür sorgt, dass nach dem Versand einer Nachricht ein bestimmtes Zeitintervall verstreichen muss, bevor erneut eine Nachricht des gleichen Typs versendet werden kann. Dadurch ließe sich die maximale Frequenz, mit der Nachrichten versendet werden, unabhängig von der Größe des Audio-Puffers manipulieren. Um hierfür ein optimales Intervall festlegen zu können, sollte zusätzlich untersucht werden, wie sich unterschiedliche Nachrichten-Frequenzen auf die Leistungsfähigkeit von WONDER und auf die Qualität der Wiedergabe auswirken.

### Steuerung aller Quellen mit jeder Plugin-Instanz

Die im Rahmen der Tests gemessenen Nachrichten-Laufzeiten sind so gering, dass es nicht zwingend erforderlich erscheint, eine virtuelle Quelle ausschließlich direkt mittels der Benutzeroberfläche der für sie zuständigen Plugin-Instanz zu steuern. Stattdessen könnte man die Benutzeroberfläche so erweitern, dass jede Quelle von jeder Plugin-Instanz aus positioniert werden kann. Die feste Zuständigkeit eines Plugins für eine bestimmte Quelle würde sich dann auf den Austausch von Automations-Daten mit dem Host beschränken. Um dies umzusetzen, müsste vor allem die GUI-Klasse *wonderjuce::SourcePanel* so ergänzt werden, dass nicht nur eine Quelle per Mausklick positionierbar ist. Zudem müsste die Klasse *wonder::SourceController* so abgeändert werden, dass für alle Quellen die entsprechenden Parameter gesetzt und die dazugehörigen Nachrichten versendet werden können. Diese Funktionalitäten sind im Wesentlichen schon vorhanden, sind jedoch momentan absichtlich so gekapselt, dass die Aufrufe nur für die eine Quelle erfolgen können, für die das Plugin zuständig ist.

### Verzicht auf Darstellung aller Quellen

Ein dazu gegenläufiges Konzept würde die Erkenntnis berücksichtigen, dass der vielfache Empfang von OSC-Nachrichten bei vielen Plugin-Instanzen maßgeblich die Leistungsfähigkeit des Plugins begrenzt. Würde man darauf verzichten, dass jede Plugin-Instanz alle aktiven Quellen darstellen kann, würde jede Plugin-Instanz nur noch die Nachrichten empfangen müssen, welche die jeweils vom Plugin kontrollierte Quelle betreffen. Hierfür müsste die StreamMulticaster-Komponente durch eine Art Kommunikationsknoten-Komponente ersetzt

werden, die den gesamten „Visual Stream“ von cWONDER empfängt und jede empfangene Nachricht dann entsprechend der jeweils betroffenen virtuellen Quelle per Unicast ausschließlich an die zuständige Plugin-Instanz weiterleitet. Die wesentliche Aufgabe der Plugin-Instanzen wäre dann (ähnlich wie bei den Client-Plugins von SPURCEMENT [Man12, S. 38 ff.]) eher auf den Austausch von Automations-Daten mit dem Host reduziert. Um bei der Positionierung der Quellen einen Überblick über alle aktiven Quellen zu haben, müsste man zusätzlich eine andere Benutzeroberfläche nutzen. Dies könnte einerseits xWONDER sein, andererseits wäre denkbar, die neu geschaffene Kommunikationsknoten-Komponente um eine entsprechende Benutzeroberfläche zu erweitern. Beide Varianten würden den Bedienkomfort der einzelnen Plugin-Instanzen wesentlich beinträchtigen und das System insgesamt komplexer machen – das Gesamt-Nachrichtenaufkommen würde aber deutlich reduziert werden.

### Unterstützung von LV2

Um eine noch größere Palette an Betriebssystemen und Host-Programmen zu unterstützen, könnte man eine LV2-Version von SPAOP entwickeln. Dazu würden sich zwei unterschiedliche Herangehensweisen anbieten:

Zum einen wurde bereits ein LV2-Wrapper für die JUCE-Bibliothek veröffentlicht [Coe11]. Das entsprechende Quellcode-Repository ist allerdings zur Zeit nicht mehr verfügbar. Man könnte aber Kontakt mit dem Entwickler aufnehmen und untersuchen, ob man auf diesem Wege eine entsprechende Lösung findet, oder aber einen eigenen LV2-Wrapper für JUCE entwickelt.

Ein alternativer Ansatz wäre, auf die Nutzung von JUCE zu verzichten. Durch die Entkopplung der einzelnen SPAOP-Komponenten ließen sich die zentrale Programmlogik sowie die Klassen zur OSC-Kommunikation auch unabhängig von JUCE verwenden. Man müsste jedoch neue, JUCE-unabhängige Klassen für die Benutzeroberfläche, die XML-Verarbeitung und den Timer zur Verbindungskontrolle entwickeln. Sodann könnte SPAOP auch unabhängig von JUCE an die LV2-Schnittstelle angebunden werden.

### Gruppierung virtueller Quellen

Eine weitere denkbare Erweiterung wäre es, eine Möglichkeit zu schaffen, mehrere Quellen zu gruppieren und diese als Gruppe zu steuern. Bereits jetzt bietet xWONDER solche Funktionalitäten. Allerdings werden die entsprechenden Berechnungen direkt in xWONDER ausgeführt und sodann die Quellen der Gruppe mittels entsprechender Nachrichten an cWONDER einzeln neu positioniert. Eine Nutzung dieser Funktion durch andere Komponenten ist derzeit nicht möglich.

Man könnte nun das SPAOP-Plugin seinerseits um vergleichbare Möglichkeiten erweitern. Naheliegend wäre, dass eine SPAOP-Instanz in eine mehrkanalige Spur der Host-DAW geladen wird und sodann die Möglichkeit bietet, die Audio-Kanäle dieser Spur als Gruppe von virtuellen Quellen zu steuern. Dadurch würde jedoch eine weitere WONDER-Komponente geschaffen, die zwar die Möglichkeit bietet, Gruppen von virtuellen Quellen zu steuern, diese Möglichkeit aber zunächst nicht dem Gesamtsystem verfügbar macht.

Eine Alternative wäre, das WONDER-System um eine zentrale, an cWONDER angebundene Komponente zur Steuerung von Quell-Gruppen zu ergänzen. Eine solche zentrale Lösung könnte dann mittels entsprechender OSC-Nachrichten von jeder beliebigen WONDER-Nutzerschnittstelle aus angesteuert werden. Auch das SPAOP-Plugin könnte dann um entsprechende Funktionalitäten zur Steuerung dieser Komponente ergänzt werden.

### 4.2.2 Unterstützung weiterer Wiedergabe-Verfahren

Aus Sicht des Anwenders ist es wünschenswert, die Positionierung der einzelnen Quellen auch dann nur einmal vornehmen zu müssen, wenn das Ergebnis seiner Produktion mit verschiedenen Verfahren zur räumlichen Audio-Wiedergabe genutzt werden soll. Im SPAOP-Plugin liegen einerseits die Positionsdaten vor, andererseits werden die entsprechenden Audio-Daten vom Host in Echtzeit an das Plugin übergeben.

Aktuell werden die Audio-Daten unbearbeitet an den Host zurückgegeben. Es wäre aber auch möglich, diese Daten zu nutzen und auf Basis der Positionsdaten die einzelnen Ausgabe-Kanäle von kanalbasierten Audio-Formaten wie 5.1 oder Ambisonics zu berechnen. Auf diesem Wege könnten die selben Positionsdaten für die Ausgabe in mehreren unterschiedlichen Formaten genutzt werden.

Darüber hinaus könnte ein Konzept entwickelt werden, um die vorliegenden Positions- und Audio-Daten gemeinsam in ein objektbasiertes Audioformat zu exportieren. Auf Basis der so exportierten Daten könnte das Rendering für verschiedene Wiedergabesysteme dann unabhängig von dem Plugin erfolgen. Im Bereich dieser Formate ist noch kein Standard etabliert – die Frage, wie genau ein solcher Standard aussehen könnte, wird momentan noch diskutiert. Durch Entwicklung entsprechender Referenz-Implementationen auf Basis des SPAOP-Plugins könnte ein Beitrag zu dieser Diskussion geleistet werden.

### 4.2.3 Entwicklung einer neuen WONDER-Benutzeroberfläche

Der entwickelte Quellcode könnte auch als Basis für eine eigenständige, neue grafische Benutzeroberfläche für WONDER dienen. Mit xWONDER existiert zwar bereits eine etablierte

Benutzeroberfläche für das System, jedoch wurde das WONDER-System an der HAW inzwischen um Funktionen erweitert, die von xWONDER nicht unterstützt werden (hier sei insbesondere das Festlegen und Verändern einer Hörer-Position als Bezugspunkt für das Rendering fokussierter Quellen erwähnt [Chr14]). Zudem wurde an der HAW unlängst ein Multitouch-Table angeschafft, der unter anderem zur Steuerung des WONDER-Systems genutzt werden soll, was spezielle Anforderungen an die entsprechende Benutzeroberfläche stellt.

Zumindest die Programmlogik sowie die OSC-Kommunikation könnten im Wesentlichen mit Hilfe des bereits bestehenden SPAOP-Quellcodes implementiert werden. Man müsste wiederum lediglich die Kapselung der Klasse *wonder::SourceController* so aufbrechen, dass nicht nur eine Quelle zur Zeit gesteuert werden kann. Darüber hinaus müssten noch einige Funktionen zum Verwalten der WONDER-internen Projekt-Dateien ergänzt werden – dies würde sich voraussichtlich aber durch die Unterstützung einiger zusätzlicher OSC-Nachrichten (*/WONDER/project/*$*****$) mit wenig Aufwand umsetzen lassen.

Auch für die eigentliche Benutzeroberfläche könnten sicherlich die bereits für SPAOP entwickelten, JUCE-basierten Klassen angepasst werden. Hier müsste vor allem die Reaktion auf Maus-Klicks so angepasst werden, dass alle dargestellten Quellen ausgewählt und positioniert werden können. Für die Anwendung mit einer Multitouch-Oberfläche wäre allerdings zu prüfen, ob diese von der JUCE-Bibliothek ausreichend unterstützt wird. Sollte dies nicht der Fall sein, müssten entsprechende Klassen komplett neu entwickelt werden.

### 4.2.4 Umstellung des gesamten WONDER-Systems auf OSC-Multicast

Die Erfahrungen mit dem entwickelten Plugin haben gezeigt, dass die Verteilung von OSC-Nachrichten mittels Multicast für den Versand identischer Nachrichten an mehrere Empfänger eine interessante Alternative zum Unicast-Versand darstellt und von der Liblo-Bibliothek komfortabel unterstützt wird. Im WONDER-System versendet cWONDER regelmäßig identische Nachrichten an mehrere Empfänger: insbesondere der „Render Stream“ wird üblicherweise an eine Reihe von tWONDER-Instanzen gesendet. Im Labor der HAW sind es allein 14 tWONDER-Instanzen, die mittels dieses Streams identische Nachrichten empfangen, wofür cWONDER 14 einzelne Sende-Vorgänge ausführt. Es liegt auf der Hand, dass cWONDER insgesamt deutlich entlastet würde, wenn es jeden seiner vier Streams lediglich einmal an eine entsprechende Multicast-Gruppe versenden müsste. Durch diese Umstellung würde cWONDER zwar über keine Informationen darüber verfügen, welche Komponenten aktuell verbunden sind – anscheinend werden diese Informationen aber ohnehin lediglich dazu genutzt, um sie in der Benutzeroberfläche anzeigen zu können.

Um das System entsprechend umzustellen, wäre in mehreren Schritten vorzugehen: Zunächst müsste cWONDER um die Funktion erweitert werden, jeden Stream zusätzlich zu der derzeit genutzten Unicast-Stream-Abonnenten-Liste auch noch an eine Multicast-Gruppe zu versenden. Sodann müssten die übrigen WONDER-Komponenten so angepasst werden, dass sie ihre Nachrichten fortan über die entsprechende Multicast-Gruppe beziehen. In diesem Schritt könnte eventuell erforderlich sein, kleinere Anpassungen beim Anmeldevorgang vorzunehmen. Wenn dann alle Komponenten erfolgreich umgestellt sind und das umgestellte System ausreichend getestet wurde, könnte man die cWONDER-Funktionen zum Stream-Versand per Unicast endgültig deaktivieren.

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Literaturverzeichnis


[App13] Apple Inc. (Hrsg.): *Logic Pro X : Benutzerhandbuch : Für OS X.* Cupertino, CA : Apple Inc., 2013. – Online verfügbar unter http://manuals.info.apple.com/de_DE/logic_pro_x_benutzerhandbuch.pdf - Abruf: 2014-06-16

[App14] Apple Inc. (Hrsg.): *Core Audio Overview.* Cupertino, CA : Apple Inc., 2014. – Online verfügbar unter https://developer.apple.com/library/mac/documentation/ musicaudio/Conceptual/CoreAudioOverview/CoreAudioOverview.pdf - Abruf: 2014-06-16

[avi14a] Avid Technology, Inc. (Hrsg.): *Avid Audio Developer Registration.* Daly City, CA, 2014. – http://www.avid.com/us/partners/Developer-Program/AudioPlugin - Abruf: 2014-06-16

[Avi14b] Avid Technology, Inc. (Hrsg.): *Audio Plug-Ins Guide : Version 11.1.* Daly City, CA : Avid Technology, Inc., 2014. – Online verfügbar unter: http://akmedia.digidesign.com/support/docs/Audio_Plug_ Ins_Guide_80351.pdf - Abruf: 2014-06-16

[Avi14c] Avid Technology, Inc. (Hrsg.): *ProTools® ReferenceGuide : Version 11.1.* Daly City, CA : Avid Technology, Inc., 2014. – Online verfügbar unter: http://akmedia.digidesign.com/support/docs/Pro_Tools_11_1_Reference_Guide_80560.pdf - Abruf: 2014-06-16

[Baa08] Baalman, Marije A.: *On Wave Field Synthesis and Electro-acoustic Music - With a Particular Focus on the Reproduction of Arbitrarily Shaped Sound Sources.* Saarbrücken : VDM, 2008. – ISBN 978–3–639–07731–5. – Online verfügbar unter: http://opus4.kobv.de/opus4-tuberlin/files/1801/baalman_marije.pdf. - Abruf: 2014-06-16

[BBB+12] Bachmann, Cristina ; Bischoff, Heiko ; Bröer, Marion ; Kaboth, Christina ; Mingers, Insa ; Pfeifer, Sabine ; Schütte, Benjamin ; Steinberg Media Technologies (Hrsg.): *Benutzerhandbuch : Cubase 7 Advanced*

*Music Production System, Cubase Artist 7 Music Production System.* Hamburg: Steinberg Media Technologies, 2012. – Online verfügbar unter: http://download.steinberg.net/downloads_software/Cubase_7_and_Cubase_Artist_7/7.0.2/Manuals/docs_deutsch.zip (Datei Operation_Manual_de.pdf). - Abruf: 2014-06-16

[Ber88] Berkhout, Augustinus J.: A holographic approach to acoustic control. In: *Journal of the Audio Engineering Society* 36 (1988), Nr. 12, S. 977–995

[BLEP05a] Bleda, Sergio ; Lopez, Jose J. ; Escolano, Jose ; Pueo, Basilio: Design and Implementation of a Compatible Wave Field Synthesis Authoring Tool. In: Audio Engineering Society (Hrsg.): *118th convention spring papers : Audio Engineering Society : held May 28 - 31, 2005, Barcelona, Spain.* New York, NY : Audio Engineering Society, 2005

[BLEP05b] Bleda, Sergio ; Lopez, Jose J. ; Escolano, Jose ; Pueo, Basilio: A Flexible Authoring Tool for Wave Field Synthesis. In: International Computer Music Conference (Hrsg.): *Conference proceedings : International Computer Music Conference : Escola Superior de Música de Catalunya, Barcelona, Spain, September 4 -10, 2005.* Barcelona : International Computer Music Conference, 2005, S. 267–270

[CDI13] Corteel, Etienne ; Damien, Arnault ; Ihssen, Cornelius: Spatial sound reinforcement using Wave Field Synthesis. A case study at the Institut du Monde Arabe. In: Hoeg, Wolfgang (Hrsg.): *Expertise in audio media : TMT 27 : 27. Tonmeistertagung, Köln 2012.* Bergisch-Gladbach : Verband Deutscher Tonmeister, 2013. – Online verfügbar unter: http://euphonia.fr/WFS/TMT2012_CorteelEtAl_IMA_121124.pdf. - Abruf: 2014-06-16

[Cha09] Chacon, Scott: *Pro Git : Everything you need to know about the Git distributed source control tool.* Berkeley, CA : Apress, 2009. – Online verfügbar unter: http://git-scm.com/book. - Abruf: 2014-06-16

[Chr14] Christoffel, Carola: *Modifikation der Software einer Wellenfeldsyntheseanlage zur Wiedergabe fokussierter Quellen in Abhängigkeit der Zuhörerposition.* Hamburg, Hochschule für Angewandte Wissenschaften, Bachelorarbeit, 2014

[Coe11] Coelho, Filipe: *JUCE Forums : Audio Plugins : JUCE LV2 Plugin Wrapper.* 2011. – http://www.juce.com/forum/topic/juce-lv2-plugin-wrapper. - Abruf: 2014-06-16

[dol14] Dolby Laboratories, Inc. (Hrsg.): *Dolby® Atmos® Next-Generation Audio for Cinema : WHITE PAPER*. San Francisco, CA, 2014. – Online verfügbar unter http://www.dolby.com/uploadedFiles/Assets/US/Doc/Professional/Dolby-Atmos-Next-Generation-Audio-for-Cinema.pdf - Abruf: 2014-06-16

[FN13] Fohl, Wolfgang ; Nogalski, Malte: A Gesture Control Interface for a Wave Field Synthesis System. In: *Proceedings of the 2013 Conference on New Interfaces for Musical Expression (NIME13), Daejeon + Soul, Korea Republic*. Daejon : Korea Advanced Institute of Science and Technology, 2013, S. 341–346. – Online verfügbar unter http://aimlab.kaist.ac.kr/nime2013/program/papers/day2/paper7/106/106_Paper.pdf - Abruf: 2014-06-16

[Foh13] Fohl, Wolfgang: The Wave Field Synthesis Lab at the HAW Hamburg. In: Bader, Rolf (Hrsg.): *Sound - Perception - Performance*. Heidelberg : Springer International Publishing, 2013 (Current Research in Systematic Musicology Vol. 1). – ISBN 978–3–319–00106–7, S. 243–255. – Online verfügbar unter: http://link.springer.com/chapter/10.1007/978-3-319-00107-4_10 - Abruf: 2014-06-16

[fra14] Fraunhofer-Institut für Digitale Medientechnologie IDMT (Hrsg.): *SpatialSound Wave – Die neue Generation räumlicher Klangerlebnisse*. Ilmenau : Produktbroschüre, 2014. – Online verfügbar unter: http://www.idmt.fraunhofer.de/content/dam/idmt/de/Dokumente/Publikationen/Produkt-informationen/SpatialSound%20Wave/SpatialSoundWave.pdf. - Abruf: 2014-06-16

[Fre91] Free Software Foundation, Inc. (Hrsg.): *GNU General Public License : Version 2, June 1991*. Boston, MA : Free Software Foundation, Inc., 1991. – Online verfügbar unter http://www.gnu.org/licenses/gpl-2.0.txt - Abruf: 2014-06-16

[Fre07] Free Software Foundation, Inc. (Hrsg.): *GNU General Public License : Version 3, 29 June 2007*. Boston, MA : Free Software Foundation, Inc., 2007. – Online verfügbar unter http://www.gnu.org/licenses/gpl-3.0.txt - Abruf: 2014-06-16

[FS09] Freed, Adrian ; Schmeder, Andy: Features and future of Open Sound Control version 1.1 for NIME. In: *Proceedings of the 2009 Conference on New Interfaces for Musical Expression (NIME09), Pittsburgh, USA*. Pittsburgh, PA : Carnegie Mellon School of Music, Pittsburgh, 2009, S. 116–120. – Online verfügbar unter: http://www.nime.org/proceedings/2009/nime2009_116.pdf. - Abruf: 2014-06-16

[Fur00] Furse, Richard W.: *A Plugin API*. E-Mail an die Linux Audio Developer Mailing Liste 2000-02-28, 2000. – Online verfügbar unter http://www.ladspa.org/original_api.txt - Abruf: 2014-06-16

[GAS08] Geier, Matthias ; Ahrens, Jens ; Spors, Sascha: The SoundScape Renderer: A Unified Spatial Audio Reproduction Framework for Arbitrary Rendering Methods. In: Audio Engineering Society (Hrsg.) ; Audio Engineering Society (Veranst.): *124th Audio Engineering Society convention 2008 : May 17 - 20, 2008, Amsterdam, the Netherlands*. Red Hook, NY : Curran, May 2008. – Online verfügbar unter: http://www.labs-exit.net/public/Deutsch/Publikationen/Documents/Download_Geier_2008_SSR.pdf. - Abruf: 2014-06-16

[GHJV94] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design Patterns : Elements of Reusable Object-Oriented Software*. Reading, MA : Addison-Wesley, 1994

[GS13] Geier, Matthias ; Spors, Sascha: Spatial Audio with the SoundScape Renderer. In: Hoeg, Wolfgang (Hrsg.): *Expertise in audio media : TMT 27 : 27. Tonmeistertagung, Köln 2012*. Bergisch-Gladbach : Verband Deutscher Tonmeister, 2013, S. 646–655. – Online verfügbar unter: http://www.int.uni-rostock.de/fileadmin/user_upload/publications/spors/2012/Geier_TMT2012_SSR.pdf. - Abruf: 2014-06-16

[IEE13] IEEE (Hrsg.): *IEEE Std 1003.1, 2013 Edition : Standard for Information Technology - Portable Operating System Interface (POSIX(R)) Base Specifications, Issue 7*. New York, NY : IEEE, 2013. – Online verfügbar unter http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6506091 - Abruf: 2014-06-16

[IOS12] IOSONO GmbH (Hrsg.): *Spatial Audio Workstation 2 Operation Manual*. Erfurt: IOSONO GmbH, 2012. – Online verfügbar unter: http://www.iosono-sound.com/uploads/downloads/SpatialAudioWorkstation_Manual_2.2.pdf. - Abruf: 2014-06-16

[itu12] International Telecommunications Union (Hrsg.): *Recommendation BS.775-3 (08/12) : Multichannel stereophonic sound system with and without accompanying picture*. Genf, 2012. – Online verfügbar unter http://www.itu.int/dms_pubrec/itu-r/rec/bs/R-REC-BS.775-3-201208-I!!PDF-E.pdf - Abruf: 2014-06-16

[LG11] Leckschat, Dieter ; Göbel, Sebastian: Wave Field Synthesis for all: A VST3-Plug-In for WFS. In: Teile, Günther (Hrsg.): *Expertise in audiomedia : TMT 26 : 26. Tonmeistertagung, Leipzig 2010.* Bergisch-Gladbach : Verband Deutscher Tonmeister, 2011, S. 667–669

[MAGF09] Meltzer, Stefan ; Altman, Lutz ; Gräfe, Andreas ; Fischer, Jens-Oliver: An object oriented mixing approach for the design of spatial audio scenes. In: Verband Deutscher Tonmeister (Hrsg.): *Bericht der 25. Tonmeistertagung 13.-16.11.2008, Leipzig.* Bergisch-Gladbach : Verband Deutscher Tonmeister, 2009, S. 723–729

[Man12] Mann, Maik: *Entwicklung eines Plug-Ins zur Spatialisierung von Audio-Objekten in der Wellenfeldsynthese.* Berlin, Technische Universität Berlin, Masterarbeit, 2012. – Online verfügbar unter: http://www2.ak.tu-berlin.de/~akgroup/ak_pub/abschlussarbeiten/2012/MannMaik_MasA.pdf. - Abruf: 2014-06-16

[Mel10] Melchior, Frank: Wave field synthesis and object-based mixing for motion picture sound. In: *SMPTE Motion Imaging Journal* 119 (2010), Nr. 3, S. 53–57

[MM95] Malham, David G. ; Myatt, Anthony: 3-D Sound Spatialization using Ambisonic Techniques. In: *Computer Music Journal* 19 (1995), Nr. 4, S. 58–70. – Online verfügbar unter http://www.jstor.org/stable/3680991 - Abruf: 2014-06-16

[Mon07] Mond, Hans-Joachin: *Implementierung einer graphischen Benutzeroberfläche zur Steuerung eines Wellenfeldsynthesesystems.* Berlin, Technische Universität Berlin, Masterarbeit, 2007. – Online verfügbar unter: http://www.ak.tu-berlin.de/fileadmin/a0135/Magisterarbeiten/HJ_Mond_MagA.pdf. - Abruf: 2014-06-16

[Mon11] Montag, Mathew N.: *Wave field synthesis in three dimensions by mutiple line arrays*, University Of Miami, Master Thesis, May 2011. – Online verfügbar unter: http://www.mattmontag.com/projects/wfs/Montag Thesis 2011 - Wave Field Synthesis in Three Dimensions by Multiple Line Arrays.pdf. - Abruf: 2014-06-16

[MRB+03] Melchior, Frank ; Röder, Thomas ; Brix, Sandra ; Wabnik, Stefan ; Riegel, Christian: Authoring Systems for Wave Field Synthesis Content Production. In: Audio Engineering Society (Hrsg.): *Convention paper presented at the 115th convention - Audio Engineering Society : 2003 October 10 - 13, New York, NY, USA.* New York, NY : Audio Engineering Society, Oct 2003

[MS11] Melchior, Frank ; Sporer, Thomas: 3D Audio - Einfach ein paar Kanäle dazu? Ein universelles objektbasiertes Audioformat. In: Teile, Günther (Hrsg.): *Expertise in audiomedia : TMT 26 : 26. Tonmeistertagung, Leipzig 2010.* Bergisch-Gladbach : Verband Deutscher Tonmeister, 2011, S. 236–242

[Nog12] Nogalski, Malte: *Gestengesteuerte Positionierung von Klangquellen einer Wellenfeldsynthese-Anlage mit Hilfe eines kamerabasierten 3D-Tracking-Systems.* Hamburg, Hochschule für Angewandte Wissenschaften, Bachelorarbeit, 2012. – Online verfügbar unter: http://edoc.sub.uni-hamburg.de/haw/volltexte/2013/1948/pdf/Bachelorarbeit_Malte_Nogalski_Small.pdf. - Abruf: 2014-06-16

[Oel13] Oellers, Helmut: *Holophony - the path to true spatial audio reproduction : Wave Field Synthesis.* 2013. – http://www.syntheticwave.de/Wavefieldsynthesis.htm. - Abruf: 2014-06-16

[PK04] Pellegrini, Renato ; Kuhn, Clemens: Wave Field Synthesis: Mixing and Mastering Tools for Digital Audio Workstations. In: Audio Engineering Society (Hrsg.): *Full set of convention papers presented at the 116th AES convention : 2004, May 8 - 11, Berlin, Germany.* New York, NY : Audio Engineering Society, May 2004

[Pos80] Postel, Jonathan B.: *RFC 768 : User Datagram Protocol.* Fremont, CA : The Internet Engineering Task Force (IETF), 1980. – Online verfügbar unter http://tools.ietf.org/pdf/rfc768.pdf - Abruf: 2014-06-16

[Roo14] Roos, Sebastian: *Re: Fragen zum WONDER-VST-Plugin.* E-Mail an Martin Hansen 2014-04-22, 2014

[Son13] Sonic Emotion Pro Systems (Hrsg.): *Sonic Wave I : The pro listening experience.* Oberglatt (Zürich) : Produktbroschüre, 2013. – Online verfügbar unter: http://www2.sonicemotion.com/wp-content/uploads/2014/02/BROCHURE_SonicWaveI_5_November_2013_FINAL_Lowres.pdf. - Abruf: 2014-06-16

[ste13] Steinberg Media Technologies GmbH (Hrsg.): *Steinberg VST Plug-In SDK Licensing Agreement (Version "3.6.0– 22.11.2013").* 2013. – Veröffentlicht als pdf-Datei als Teil des VST Audio Plug-Ins SDK (Version 3.6.0)

[Str13] Stroustrup, Bjarne: *A tour of C++.* Upper Saddle River, NJ : Addison Wesley, 2013. – Online verfügbar unter: http://isocpp.org/tour. - Abruf: 2014-06-16

[WFM03] Wright, Matthew ; Freed, Adrian ; Momeni, Ali: Opensound control: State of the art 2003. In: Thibault, François (Hrsg.): *Proceedings of the 2003 Conference on New Interfaces for Musical Expression (NIME-03), Montreal, Canada.* Montréal : McGill University, Faculty of Music, 2003, S. 153–159. – Online verfügbar unter: http://www.music.mcgill.ca/musictech/nime/onlineproceedings/Papers/NIME03_Wright.pdf. - Abruf: 2014-06-16

[Wor08] World Wide Web Consortium (Hrsg.): *Extensible Markup Language (XML) 1.0 (Fifth Edition) : W3C Recommendation 26 November 2008.* Cambridge, MA : World Wide Web Consortium, 2008. – Online verfügbar unter http://www.w3.org/TR/2008/REC-xml-20081126 - Abruf: 2014-06-16

[Wri02] Wright, Matthew: *The Open Sound Control 1.0 Specification.* 2002. – http://opensoundcontrol.org/spec-1_0. - Abruf: 2014-06-16

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

Hamburg, 27. Juni 2014 Martin Hansen