



TECHNISCHE
UNIVERSITÄT
MÜNCHEN

Systementwicklungsprojekt

Erstellung eines umfangreichen Tools zur
Audiobearbeitung

Fabian Prasser & Aleksandar Hristov



Institut für Informatik
Lehrstuhl für Informatik mit Schwerpunkt
Wissenschaftliches Rechnen

Technische
Universität
München



Systementwicklungsprojekt

Erstellung eines umfangreichen Tools zur Audiobearbeitung

Fabian Prasser & Aleksandar Hristov

Aufgabensteller: Univ.-Prof. Dr. T. Huckle
Betreuer: Dipl.-Inf. D. Pflüger
Abgabetermin: 01.10.2006

Inhalt:

1. Aufgabenstellung.....	4
2. Lösungsansatz und Implementierung.....	4
3. Das Rahmenprogramm.....	6
4. Audiosamples.....	7
4.1. Kodierung und Dekodierung von Audiodaten.....	7
4.2. Darstellung von Audiodaten.....	8
5. Tools zur Bearbeitung von Audiodaten.....	9
5.1. Der Audioeditor.....	10
5.1.1. Effekte.....	11
5.1.1.1. Tonhöhenanpassung (Pitch).....	11
5.1.1.2. Hüllkurve.....	12
5.1.1.3. Mittelwert-Filter.....	13
5.1.1.4. Echo.....	13
5.1.1.5. Chorus.....	14
5.1.2. Mögliche Verbesserungen.....	15
5.2. Das Frequenzanalyse-Tool.....	15
5.2.1. Mögliche Verbesserungen.....	16
5.3. Der Spektrogramm-Betrachter	16
5.3.1. Mögliche Verbesserungen.....	18
5.4. Der Oszillator.....	18
5.4.1. Mögliche Verbesserungen.....	18
5.5. Der Sequencer.....	19
5.5.1. Mögliche Verbesserungen.....	20
6. MIDI.....	21
6.1. Midi-Editor.....	22
6.1.1. Mögliche Verbesserungen.....	24
7. Fazit.....	25
8. Quellenangaben.....	26
9. Anhang: Kurzreferenz „JAudio Tool Suite“.....	27

1. Aufgabenstellung

Ziel des Projektes war ein Programm welches von jugendlichen Mädchen im Praktikum „Musik und Computer“ im Rahmen der Herbstuniversität an der TU München verwendet werden konnte. Thema des Praktikums war eine Einführung in physikalische, mathematische und computerrelevante Aspekte der Musik. Zu diesem Zweck mussten die Schülerinnen der gymnasialen Oberstufe verschiedene Aufgaben lösen, wie z.B. das Wiederherstellen und Erkennen verfremdeter Musikstücke, das Entfernen von Störgeräuschen oder das nachkomponieren von bekannten Melodien.

Die Aufgabenstellung der „Entwicklung eines umfangreichen Tools zur Audiotbearbeitung“ wurde bewusst nicht näher spezifiziert, da zum Zeitpunkt der Aufgabenstellung noch nicht absehbar war, was sich alles im gegebenen Zeitrahmen von 10 Monaten verwirklichen lässt. Einzige Vorgabe war, dass sowohl digitalisierte analoge Audiosignale als auch Midi-Dateien (Musical Instrument Digital Interface) erstellt und bearbeitet werden können.

2. Lösungsansatz und Implementierung

Bei der Implementierung stellte sich zuerst die Frage, welche Sprache verwendet werden sollte. Nach einigen Recherchen erschien uns, vor allem aufgrund der angestrebten Plattformunabhängigkeit, Java für unser Vorhaben als geeignet. Das Programm konnte somit ohne weiteren Aufwand zur Herbstuniversität auf den Arbeitsplatzrechnern am Lehrstuhl (Fedora Core 6) verwendet werden, als auch den Schülerinnen im Falle eines weiteren Interesses an diesem Thema für Arbeiten am heimischen Rechner zur Verfügung gestellt werden, auf denen wohl ausschließlich Microsofts Windows-Betriebssystem zum Einsatz kommen dürfte.

Die „Java Sound API“ von Sun stellt darüber hinaus umfangreiche Hilfsmittel zum Kodieren und Dekodieren von digitalisierten analogen Audio-Daten als auch äquivalente Werkzeuge für das Midi-Protokoll zur Verfügung.

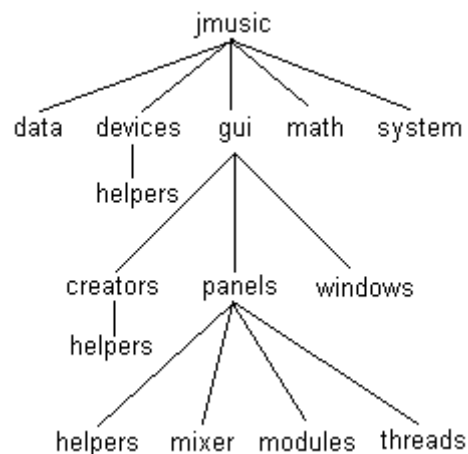
Aus Gründen der Übersichtlichkeit entschieden wir uns für eine Aufteilung der verschiedenen Bearbeitungs- und Analysemöglichkeiten auf kleine Tools, welche aus einem Rahmenprogramm heraus aufgerufen werden können.

Grundlage des Rahmenprogramms ist eine Medienbibliothek, welche im Unterordner „samples“ der Installation des Projektes lokalisiert ist.

Die thematische Zweiteilung des Themas führte uns dazu, auch bei der Bearbeitung des Projektes getrennt vorzugehen. Aleksandar Hristov beschäftigte sich dabei mit dem Midi-Protokoll und Fabian Prasser mit der Bearbeitung von digitalisierten analogen Audiosignalen und der Erstellung des Rahmenprogramms.

Für die grafische Benutzeroberfläche kommen, ebenfalls aufgrund der Plattformunabhängigkeit, Swing-Komponenten zum Einsatz. Für die Realisierung eines Drehknopfes, wie er bei Audioanwendungen häufig zum Einsatz kommt, wurde die frei verwendbare Klasse DKnob.java des Autors Joakim Eriksson verwendet.

Bei der Implementierung ergab sich folgende Struktur:



jmusic.data – Enthält Klassen zur abstrakten Repräsentation von Midi-Tracks und Operationen auf diesen.

jmusic.devices - Enthält Klassen zum Dekodieren / Aufnehmen / Abspielen von Audio- und Midi-Daten.

jmusic.devices.helpers - Enthält von Thread abgeleitete Klassen, welche von den Klassen aus jmusic.devices verwendet werden.

jmusic.gui.creators - Enthält die verschiedenen Tools. Alle Klassen sind von JFrame abgeleitet.

jmusic.gui.creators.helpers - Enthält von den Tools benötigte Dialoge. Alle Klassen sind von JDialog abgeleitet.

jmusic.gui.panels - Enthält Panels zur Bearbeitung und Visualisierung von Audio- und Midi-Daten. Alle Klassen sind von JPanel abgeleitet.

jmusic.gui.panels.helpers - Enthält Hilfsklassen für die Panels.

jmusic.gui.panels.mixer - Enthält Klassen zur Repräsentation und zum Rendern von Sequencer-Projekten

jmusic.gui.panels.modules - Enthält eine Klasse zur Repräsentation des Fortschrittes bei der Durchführung einer rechenintensiven Aufgabe

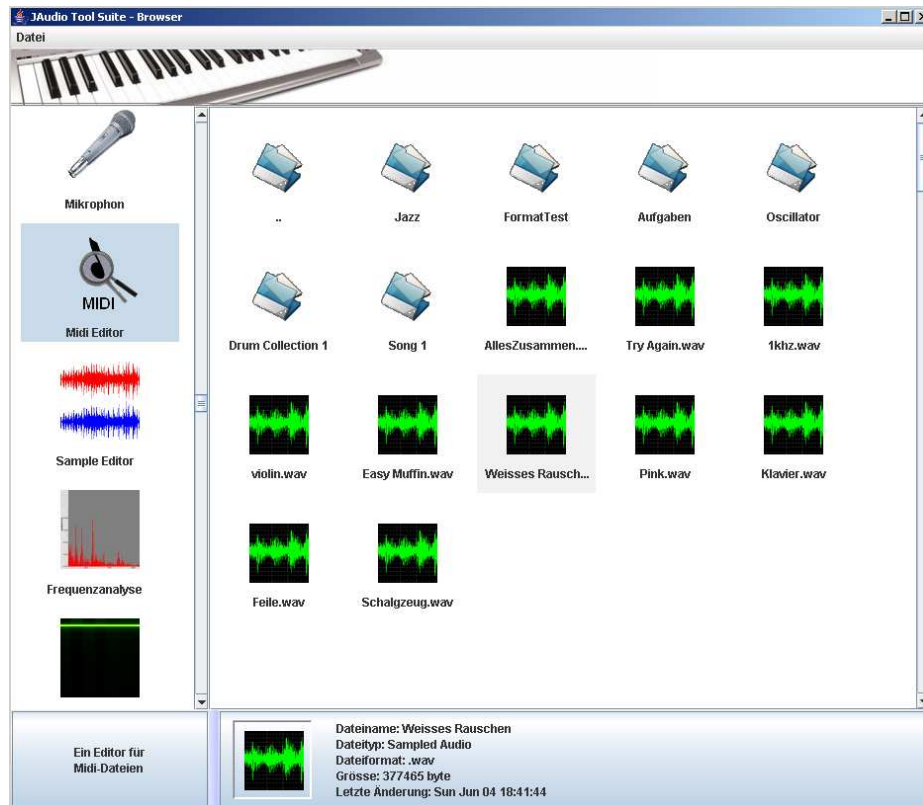
jmusic.gui.panels.threads - Enthält Threads, welche von den Panels benötigt werden. Alle Klassen sind von Thread abgeleitet.

jmusic.gui.windows - Enthält das Rahmenprogramm und den Splash-Screen. Alle Klassen sind von JFrame abgeleitet

jmusic.system - Enthält die Klasse System. Sie stellt die zentrale statische Einheit dar, über welche auf die von den verschiedenen Packages zur Verfügung gestellten Services zugegriffen werden kann. Enthält die main-Methode.

3. Das Rahmenprogramm

Wie bereits erwähnt, stellt das Rahmenprogramm alle nötigen Aktionen zum Zugriff sowohl auf die zu bearbeitenden Dateien als auch auf die verschiedenen Tools bereit. Im Quellcode wird das Rahmenprogramm durch die Klasse `MainWindow` repräsentiert welche von `JFrame` abgeleitet ist. Es präsentiert sich in folgendem Gewand:



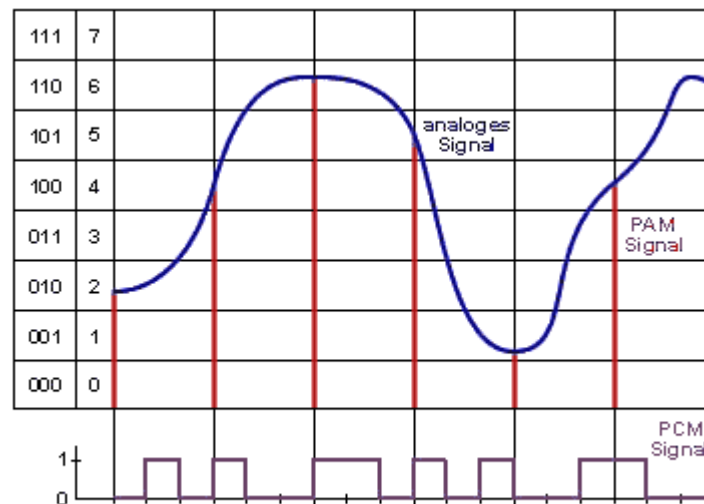
Im linken Teil des Fensters ist dabei eine Liste der vorhandenen Tools lokalisiert, der rechte Teil stellt einen Datei-Browser dar. Einfaches Anklicken eines Elementes aus der Liste zeigt im linken unteren Bereich eine Beschreibung des ausgewählten Tools an, der rechte untere Bereich zeigt Informationen zur aktuell markierten Datei. Mittels Rechtsklick stehen im Dateibrowser über Kontextmenüs weitere Optionen zur Verfügung, wie z.B. das Erstellen oder Löschen von Ordnern, das Importieren und Exportieren von Dateien in die Medienbibliothek oder das Öffnen einer Datei in einem mit diesem Dateityp verknüpften Tool.

Die Anzahl der zur Verfügung stehenden Tools wurde dabei variabel gehalten. Neue Tools müssen zu diesem Zweck bloß in der Klasse `service` des Packages `jmusic.gui.creators` eingetragen werden. Das Rahmenprogramm greift nun über die Funktionen `string[] getListOfCreators()`, `String getDescriptionOf(String creator)`, `ImageIcon getSymbolOf(String creator)` und `startCreator(String creator)` der Klasse `service` auf die verfügbaren Tools zu. Eine globale Instanz der Klasse ist über die statische Klasse `system` verfügbar.

4. Audiosamples

Ein Audiosample ist ein digitalisiertes analoges Audiosignal, welches durch Abtastung entsteht. Das Grundprinzip der Abtastung ist, dass eine physikalische Größe über einen gewissen Zeitraum akkumuliert wird und dann der akkumulierte Wert in eine digitale Darstellung des Messwertes umgewandelt wird. Dies geschieht periodisch mit der Abtastfrequenz. Im Falle des analogen Audiosignals werden über einen A/D Wandler Ausschnitte (Samples) entnommen und gespeichert. Man erhält also eine diskrete Darstellung des zeitlichen Verlaufs einer Schwingung, indem man zu bestimmten Zeitpunkten die Auslenkung einer Schwingung festhält.

Die Digitalisierung eines analogen Signals erfolgt in zwei Schritten. Nach der Abtastung erfolgt die Quantisierung des zeitdiskreten aber noch wertkontinuierlichen Signals. Dadurch entsteht ein zeit- und wertdiskretes Signal. Folgende Grafik verdeutlicht den Sachverhalt:



Das analoge Signal (im Bild blau dargestellt) wird im ersten Schritt durch das Abtasten mit gegebener Abtastrate in ein sogenanntes PAM Signal (Pulsamplitudenmodulation) übergeführt. Im zweiten Schritt werden die immer noch diskreten Amplitudenwerte in eine begrenzte Zahl von Quantisierungsstufen unter Berücksichtigung der Bittiefe eingeteilt (hier: 3 Bit). Der nun vorliegende Bitstrom wird als PCM-Signal bezeichnet (Pulse-Code-Modulation).

Die Qualität des entstehenden Audiosamples wird dabei im Wesentlichen durch die Abtastrate und die Bittiefe bestimmt. Die Bittiefe gibt dabei den Wertebereich bei der Quantisierung der Werte des Signals und die Abtastrate die Anzahl der Samples pro Sekunde an.

Heutige Audio-CDs verwenden eine Bittiefe von 16 Bit, bei einer Abtastrate von 44100 Hz und liefern dabei 2 unabhängige Kanäle (Stereo).

Eine Sekunde digitalisierte Audiodaten benötigen in diesem Format $44100 * 16 * 2 \text{ Bit} = 1411200 \text{ Bit} = 176400 \text{ Byte} = 172 \text{ kB}$ an Speicherplatz.

Bei einer Länge von ca. 3 Minuten für ein durchschnittliches Musikstück ergibt sich somit ein Speicherplatzbedarf von über 30 MB, was auch die Beliebtheit verschiedener Datenkompressionsmethoden erklärt.

4.1. Kodierung und Dekodierung von Audiodaten

Zum aktuellen Zeitpunkt unterstützt das Programm ausschließlich Daten im Wave-Format und diese wiederum nur in der Variante mit PCM-Rohdaten im Little-Endian-Format (Wave-Standard) gelesen werden können Daten mit Bittiefen von 8 und 16 Bit, Stereo oder Mono, mit beliebigen Abtastraten.

Das Sound API liefert unter Verwendung der passenden Funktionen ein Byte-Array welches nun unter Berücksichtigung von Sampling-Rate, Bittiefe und Kanalanzahl zu dekodieren ist. Dabei gilt es folgendes zu berücksichtigen:

1. Die Samples mehrerer Kanäle zu einem Zeitpunkt t bilden zusammen einen sogenannten Frame. Im Byte-Array folgt immer ein Frame auf den nächsten. Dies bedeutet, dass zum Beispiel bei einem Stereo-Signal immer abwechselnd Samples des linken und des rechten Kanals dekodiert werden müssen. Bei einem Mono-Signal entspricht ein Sample einem Frame:



2. Abhängig von der Bittiefe werden die Amplitudenwerte unterschiedlich kodiert. Bei einer Bittiefe von 8Bit wird der Datentyp unsigned char (Wertebereich 0 bis 255) , bei einer Bittiefe von 16 Bit der Datentyp signed short (Wertebereich -32768 bis 32767) verwendet.

Aufgrund des internen Audioformates von 16 Bit, 44100 Hz, Stereo gehen bei Bittiefen von mehr als 16 Bit und Abtastraten über 44100 beim Öffnen Teile der Daten verloren. Zur Anpassung der Abtastrate wird das Sample nach dem Einlesen um den Faktor $p = (44100 / \text{Abtastrate})$ gepitcht, also in seiner Länge angepasst. Für $p > 1.0$ kommt dabei lineare Interpolation zum Einsatz.

Das Speichern von erstellten oder bearbeiteten Daten ist bisher nur im internen Audioformat möglich. Zu diesem Zweck müssen die vorher getrennt im Speicher als short-Arrays abgelegten Kanäle nach dem Frame-Muster wieder zusammengesetzt und in Bytes konvertiert werden.

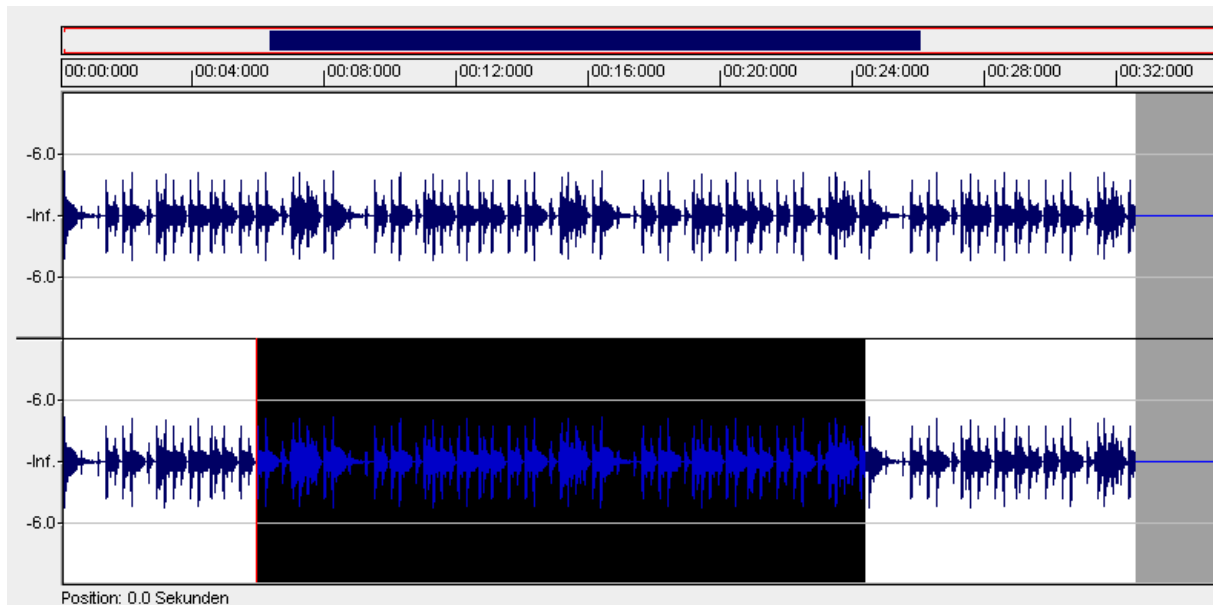
Die zum Auslesen, Dekodieren, Kodieren und Speichern benötigten Funktionen werden von der Klasse `service` des Packages `jmusic.devices` bereitgestellt. Eine globale Instanz ist über die statische Klasse `system` verfügbar.

Die Klasse umfasst im wesentlichen folgende Funktionen:

```
- byte[] create16BitStereoByteArrayFromFile(String fileName)
- void write16BitByteArrayToFile(String fileName)
- Vector getAudioFormat(String fileName)
- byte[] convertShortArrayToByteArray(short[] array)
- byte[] packByteArraysToAudioData(byte[] leftChannel, byte[] rightChannel)
- short[] convertRaw16BitMonoAudioDataToShort(byte[] data)
- byte[][] convertRaw16BitStereoAudioDataToByteArrays(byte[] data)
- short[][] convertRaw16BitStereoAudioDataToShortArrays(byte[] data)
```

4.2. Darstellung von Audiodaten

Für die klassische Darstellung der Audiodaten im Zeit-Amplituden-Raum stehen verschiedene Vergrößerungsfaktoren (2^0 bis 2^{14}) zur Verfügung, um eine exakte Auswahl des gewünschten Bereiches zu ermöglichen. Abhängig vom Vergrößerungsfaktor werden dabei 2^n Samples auf eine Zeiteinheit (ein Pixel in x-Richtung) abgebildet. Es ergibt sich die klassische Ansicht:



Dies wird in der Klasse `wavePanel` des Packages `jmusic.gui.panels` verwirklicht und kommt in mehreren Tools zum Einsatz. Neben den Audiodaten selbst, ist im oberen Bereich sowohl eine Zeitskala, als auch ein Bereich vorhanden, welcher den gerade sichtbaren (roter Balken) und den aktuell markierten Bereich anzeigt (blauer Balken), um die Übersichtlichkeit zu gewährleisten.

Um eine korrekte Abbildung der Samples zu erreichen ist es nötig alle Samples zu untersuchen, jeweils den maximalen und minimalen Wert von 2^n Samples zu bestimmen und beide auf eine Zeiteinheit abzubilden.

Bei geringen Vergrößerungsfaktoren von bis zu $2^9=512$ Samples pro Zeiteinheit ist es möglich die Darstellung in Echtzeit zu berechnen, da z.B. bei einer Breite von 600 Pixeln nur $600 \cdot 512 = 307200$ Samples betrachtet werden müssen. Dies ist gerade noch ohne einen merklichen Einfluss auf die Performance möglich.

Spätestens ab einem Vergrößerungsfaktor von 2^{10} übersteigt die Anzahl der zu betrachtenden Samples die Grenze dessen, was in Echtzeit möglich ist. Beispielhaft wären bei einem Faktor von $2^{15}=32768$ und einer Breite von 600 Pixeln $32768 \cdot 600 = 19660800$ Samples zu betrachten, was den Sachverhalt veranschaulicht.

Um diese Problematik zu lösen werden alle Ansichten mit einem Vergrößerungsfaktor von mehr als 2^9 im Falle einer Änderung der Daten vorberechnet und im Arbeitsspeicher bereitgehalten. Der dafür nötige zusätzliche Speicherplatz ist sehr gering.

Für beliebige Audiodaten entsteht für die vergrößerten Ansichten ein zusätzlicher Speicherbedarf von: $(1/(2^{10}) + 1/(2^{11}) + 1/(2^{12}) + 1/(2^{13}) + 1/(2^{14}) + 1/(2^{15}))$ oder mit der geometrischen Reihe:

$$\sum_{i=0}^{15} \frac{1}{2^i} - \sum_{i=0}^9 \frac{1}{2^i} = \frac{\frac{1}{2^{16}} - \frac{1}{2^{10}}}{-\frac{1}{2}} = \frac{1}{2^9} - \frac{1}{2^{15}} = 0.001923$$

Der zusätzliche Speicherbedarf liegt also bei unter 0.2 Prozent.

Dabei ist es durchaus sinnvoll nicht alle Ansichten im Voraus zu berechnen, sondern Ansichten mit einem Vergrößerungsfaktor von weniger als 2^{10} in Echtzeit zu errechnen. Durch eine Speicherung aller Ansichten ergäbe sich ein Speicherbedarf von

$$\sum_{i=0}^{15} \frac{1}{2^i} = 2 - \frac{1}{2^{15}} = 1.999$$

Es würde also nahezu die doppelte Menge an Speicher benötigt.

5. Tools zur Bearbeitung von Audiodaten

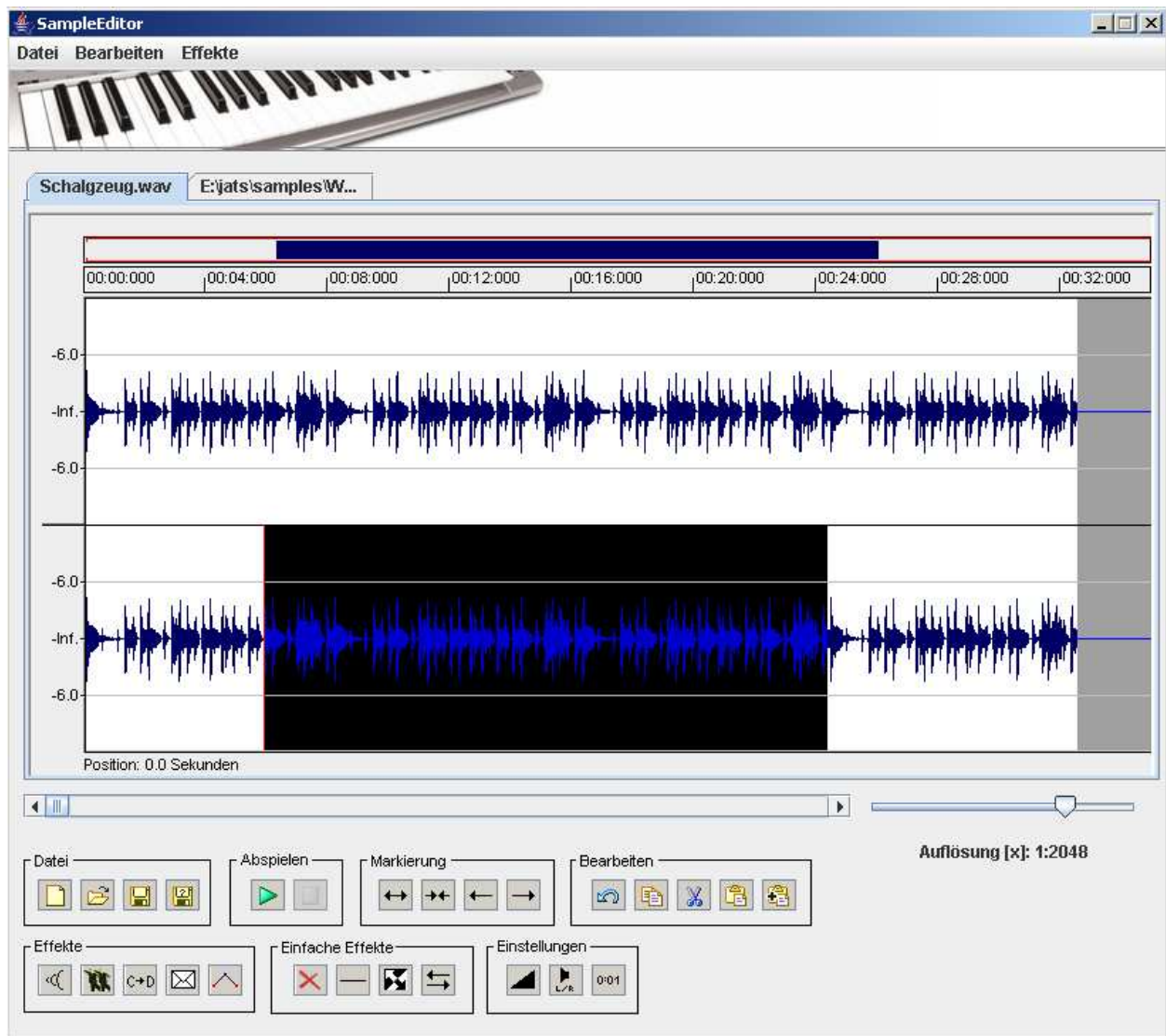
Zur Bearbeitung Erzeugung und Analyse von Audiodaten stehen 4 Tools zur Verfügung. Ein Frequenzanalyse-Tool, welches die Daten in den Frequenz-Amplituden-Raum abbildet, ein Sonogramm-Betrachter, welcher die Daten in den Zeit-Frequenz-Amplituden-Raum abbildet, ein Audio-Editor mit umfangreichen Optionen einen Oszillator zur Erzeugung von einfachen Schwingungen, ein Aufnahme-Tool und ein Sequencer zur Erstellung von Musik. Diese werden im Folgenden detailliert vorgestellt.

5.1. Der Audioeditor

Der Audioeditor ermöglicht das Bearbeiten von bis zu 5 Audiodateien gleichzeitig. Dabei können Daten zwischen den einzelnen Dateien hin und her kopiert werden. Beide Kanäle können getrennt bearbeitet werden. Es stehen Funktionen wie Ausschneiden, Kopieren, Einfügen, Mischen, Invertieren, Stille Einfügen und Umdrehen zur Verfügung. Des Weiteren können Lautstärke, Balance und Länge beliebig angepasst werden.

Die Grundlage des Audio-Editors stellen bis zu fünf Instanzen der Klasse `wavePanel` dar, welche von der sie beinhaltenden Klasse `sampleEditor` gesteuert werden. Die Klasse `wavePanel` stellt dabei die Funktionalität zur Manipulation der Daten bereit und bietet auch die Möglichkeit, den letzten Vorgang rückgängig zu machen. Mittels der Funktion `byte[] getSelectedAudio()` ist es möglich, den gerade markierten Bereich der Daten eines `wavePanel`'s abzurufen. Dies ermöglicht das Kopieren der Daten zwischen den geöffneten Dateien.

Der Audio-Editor präsentiert sich in folgendem Gewand:



Die Reiter am oberen Rand des Fensters ermöglichen die Navigation zwischen den geöffneten Dateien. Direkt darunter befindet sich das bereits bekannte WavePanel. Unterhalb des Panels befindet sich eine Scrollbar zur Navigation im Audiosample und ein Slider zur Auswahl des Vergrößerungsfaktors. Die Buttons unterhalb der Navigationsleiste dienen der Auswahl der gewünschten Aktion. Diese können ebenfalls über das Menü des Fensters oder über ein Kontextmenü im Datenbereich ausgewählt werden.

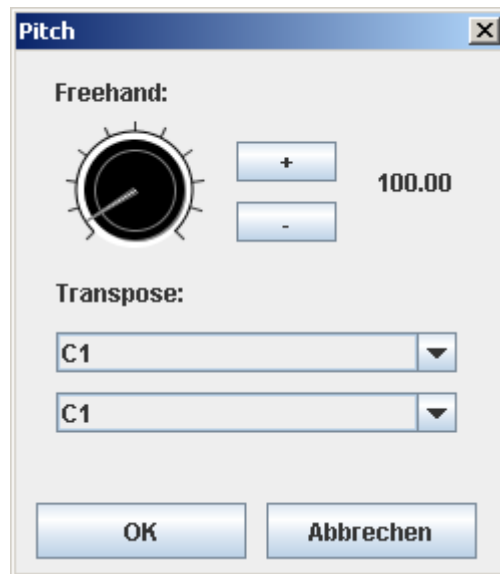
5.1.1. Effekte

Neben den klassischen Schnittfunktionen bietet der Audio-Editor weitere umfangreiche Effektmöglichkeiten, welche im Folgenden kurz vorgestellt werden. Zur Bedienung der Effekte wurden Dialoge erstellt. Diese befinden sich im Package `jmusic.gui.creators.helpers` und sind alle von `JDialog` abgeleitet.

5.1.1.1 Tonhöhenanpassung (Pitch)

Der Pitch dient der Längenanpassung des Samples, also einer Stauchung oder Dehnung der Daten. Dies beeinflusst auch die Tonhöhe des Audiosamples, da sich die Wellenlänge der Schwingung verändert.

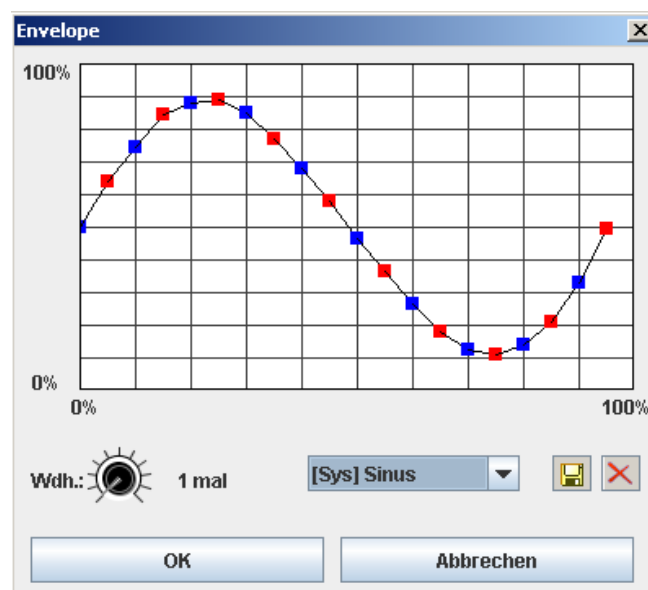
Der Nutzer kann entweder eine Prozentzahl wählen, um anzugeben wie das Sample gepitcht werden soll, oder die Daten unter Angabe des aktuellen Tones gezielt transponieren.



Eine Stauchung der Daten wird durch das Entfernen entsprechender Samples aus den Daten erreicht, im Falle einer Streckung der Daten kommt, wie bei der Anpassung der Abtastfrequenz, lineare Interpolation zum Einsatz.

5.1.1.2. Hüllkurve

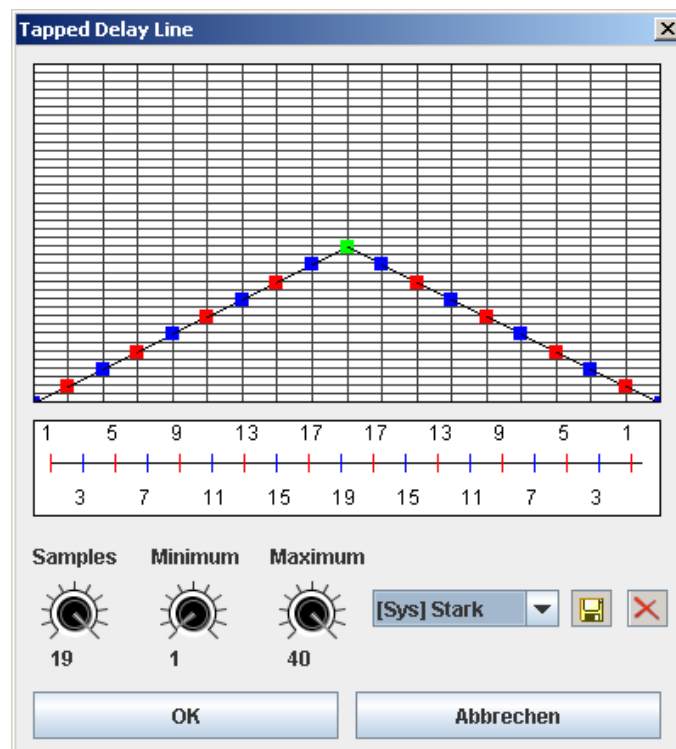
Des Weiteren besteht die Möglichkeit eine Hüllkurve über die Daten zu legen. Dabei kann aus einer großen Anzahl unterschiedlicher Kurven gewählt werden. Auch das Zeichnen und Speichern von Kurven, sowie eine beliebige Wiederholung der ausgewählten Kurve ist möglich. Dabei kommt folgender Dialog zum Einsatz:



Technisch besteht eine Hüllkurve aus einem Array der Länge 20, in welchem Werte von 0.0 bis 1.0 die Hüllkurve beschreiben. Je nach der Anzahl der Wiederholungen wird dieses Array dupliziert. Das nun entstehende Array wird, wieder unter Verwendung linearer Interpolation, auf die Länge der Audiodaten ausgedehnt und anschließend jeder Wert der Daten mit dem entsprechenden Wert aus dem ausgedehnten Hüllkurven-Array multipliziert.

5.1.1.3. Mittelwert-Filter

Mit Hilfe des Mittelwertfilters ist es möglich hohe Frequenzen zu filtern und so zum Beispiel Rauschen zu entfernen. Zu diesem Zweck wird über einer ungeraden Anzahl von Samples unter der Berücksichtigung von Gewichten ein Mittelwert gebildet und der Wert des mittleren Samples durch diesen Mittelwert ersetzt. Dieser Vorgang wird nun unter Inkrementierung des Anfangssamples für das gesamte Audiosample wiederholt. Dem Benutzer steht folgender Dialog zur Verfügung:



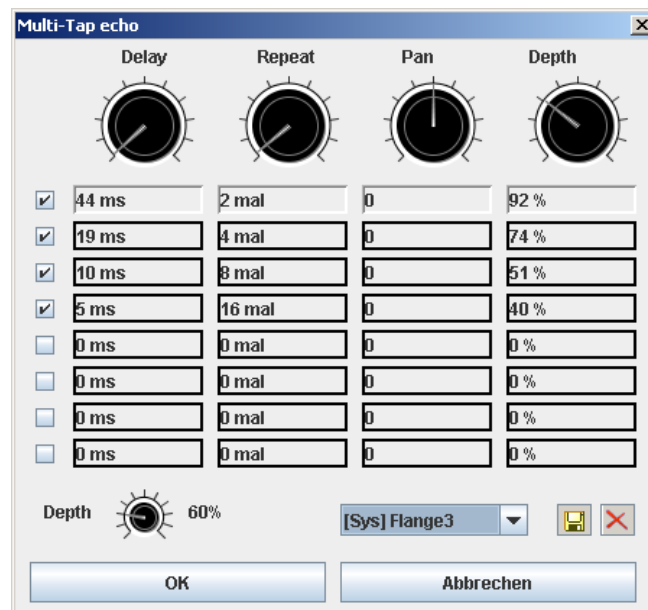
Hier besteht die Möglichkeit die Anzahl der zu bemittelnden Samples, das minimale und das maximale Gewicht zu wählen. Nun können die einzelnen Samples mittels einer grafischen Oberfläche Gewichtet werden. Auch hier besteht die Möglichkeit Konfigurationen zu laden und zu speichern.

5.1.1.4. Echo

Es besteht die Möglichkeit die Daten mit bis zu 8 Echos gleichzeitig zu versehen. Für jedes einzelne Echo muss dazu die Verzögerung (Delay) die Anzahl der Wiederholungen (Repeat) die Balance (Pan) und die Stärke (Depth) angegeben werden. Des weiteren kann die Gesamtstärke des Echos gewählt werden.

Technisch wird bei der Anwendung eines Echo-Effektes auf den Daten eine Kopie der

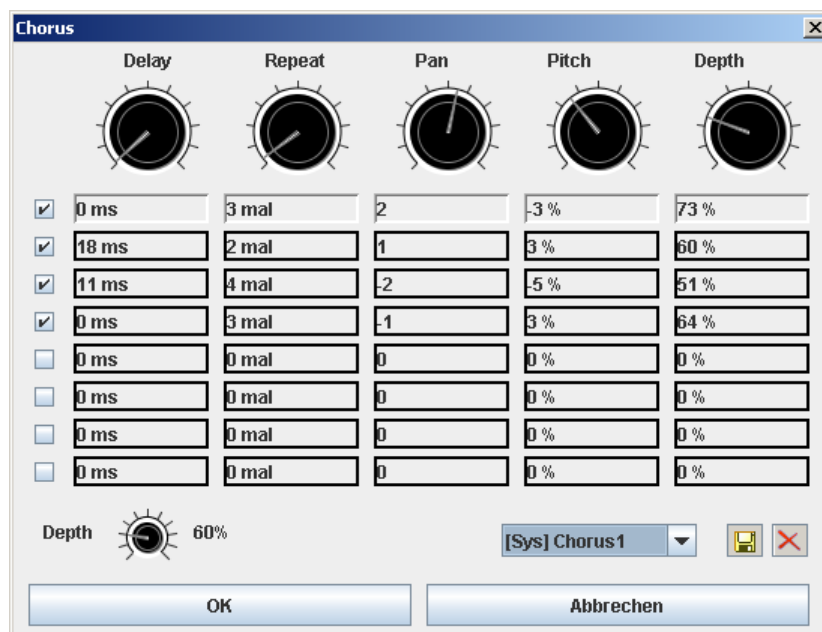
markierten Daten erstellt und diese unter Berücksichtigung der Angaben des Benutzers mehrfach auf die ursprünglichen Daten aufaddiert. Folgender Dialog dient dem Echo-Effekt:



Durch die Wahl verschiedener Verzögerungen und Wiederholungsraten lassen sich unterschiedlichste Echo-Effekte erzeugen. Auch hier können erstellte Konfigurationen geladen und gespeichert werden.

5.1.1.5. Chorus

Der Chorus-Effekt dient der Simulation von Mehrstimmigkeit und unterscheidet sich nur geringfügig vom Echo-Effekt. Hier kann zusätzlich zu den vom Echo bekannten Einstellungen für jedes einzelne Echo noch ein Pitch angegeben werden. Folgender Dialog findet dabei Verwendung:



Speichern und Laden gehört auch hier zum Funktionsumfang.

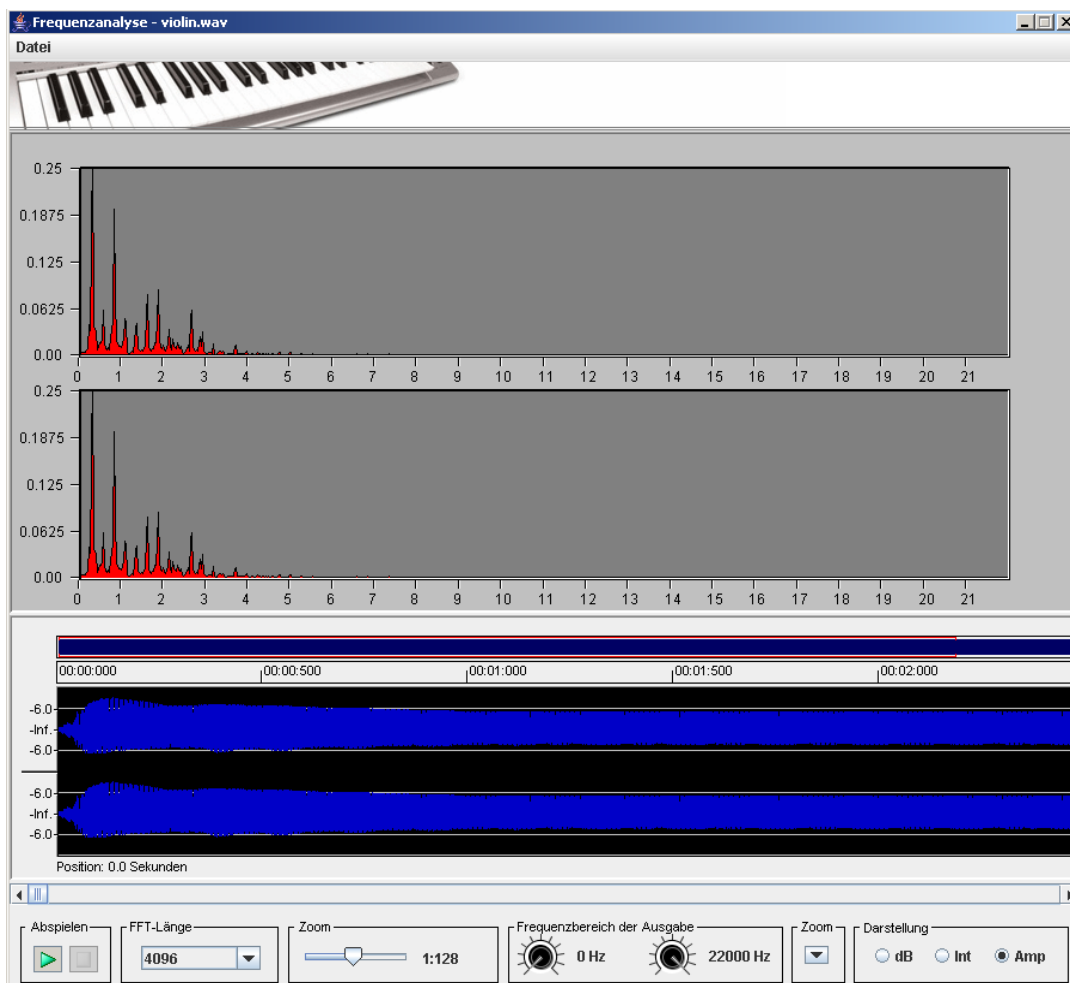
5.1.2. Mögliche Verbesserungen

Bei einer erneuten Realisierung würden wir einige Aspekte des Audio-Editors anders umsetzen. Zum einen fehlt grundsätzlich (also nicht nur im Audio-Editor) eine abstrakte Repräsentation der Audiodaten, wie z.B. eine Klasse `AudioData`, welche häufig benötigte Funktionalitäten und ein einheitliches Datenformat bereitstellt. Stattdessen werden bisher Arrays verschiedener Datentypen verwendet, wobei jede Klasse, abhängig davon, wofür sie die Daten benötigt, ein byte-Array oder ein short-Array enthält. Es stellte sich am Ende heraus, dass dieses Fehlen an Struktur häufige Konvertierungen zwischen den Formaten nötig macht und der Übersichtlichkeit und Verständlichkeit des Codes nicht zuträglich ist.

Auch die Tatsache, dass die Funktionen zur Anwendung der Operationen auf den Daten von der Klasse `wavePanel` bereitgestellt werden entpuppte sich als unpassend. Hier wäre es besser, diese Funktionalität entweder in der abstrakten Repräsentation der Daten zu kapseln oder eine Klasse `AudioDataManipulator` oder dergleichen bereitzustellen, welche diese Aufgaben bei Bedarf übernimmt.

5.2. Das Frequenzanalyse-Tool

Mit Hilfe des Frequenzanalyse-Tools ist es möglich den zeitlichen Verlauf der Stärke der einzelnen Frequenzen im Signal zu betrachten.



Dazu wird unter Berücksichtigung der angegebenen Länge eine schnelle Fouriertransformation über einem Datenpaket berechnet. Ausschlaggebend für die Position des Datenpaketes ist die aktuelle Position innerhalb des Audiosamples. Diese kann zum einen statisch von Hand ausgewählt werden, zum anderen wird sie beim Abspielen der Daten immer mit der aktuellen Position synchronisiert.

Die Daten werden im Frequenz-Amplituden-Raum als klassisches Frequenzspektrum angezeigt. Dabei kann das betrachtete Frequenzintervall frei gewählt werden. Des Weiteren stehen 3 mögliche Skalierungen der Daten zur Verfügung: eine logarithmische, eine absolute und eine Skala bei der alle Werte relativ zum größten Wert angezeigt werden.

Grundlage der Darstellung der Daten bildet die Klasse `FFTPanel`. Eine Instanz dieser Klasse ist wiederum, neben einer Instanz der Klasse `wavePanel`, in die Klasse `FrequencyAnalyzer` eingebettet, welche das Frequenzanalyse-Tool darstellt.

Die Berechnung der FFT und die Skalierung des Ergebnisses finden, aufgrund der Rechenintensivität, in einem Thread statt (Klasse `FFTCalculationThread`), welcher ständig mit aktuellen Datenpaketen gefüttert wird und das Ergebnis mit Hilfe einer Callback-Funktion an das `FFTPanel` weitergibt.

5.2.1. Mögliche Verbesserungen

Im Frequenzanalyse-Tool ist die Interpretation der Ergebnisse der FFT im Bezug auf die Einheiten der Werte für die einzelnen Frequenzen noch nicht korrekt umgesetzt. Als zukünftiges Feature wäre es denkbar eine Möglichkeit bereitzustellen die Frequenzanalyse auch über einem beliebig ausgewählten Teilstück der Daten durchführen zu können.

5.3. Der Spektrogramm-Betrachter

Mit Hilfe des Spektrogramm-Betrachters ist es möglich die Audiodaten im dreidimensionalen Zeit-Frequenz-Amplituden-Raum zu betrachten. Die X-Achse stellt die Zeit, die Y-Achse stellt die Frequenz dar. Die dritte Dimension, also die Amplitude einer Frequenz zum Zeitpunkt t , wird durch einen Farbverlauf dargestellt. Dieser erstreckt sich über das gesamte Spektrum der Farbe grün, also von schwarz über die verschiedenen Grüntöne, bis weiß. Auch hier kann wieder die FFT-Länge und das zu betrachtende Frequenzintervall angegeben werden.

Die Darstellung des Spektrogrammes wird von der Klasse `sonogramPanel` des Packages `jmusic.gui.panels` übernommen. Dieses startet zur Berechnung der Grafik einen Thread, den `sonogramCalculationThread`, welcher wiederum über eine Callback-Funktion den aktuellen Status der Berechnungen an das `sonogramPanel` zurückgibt.

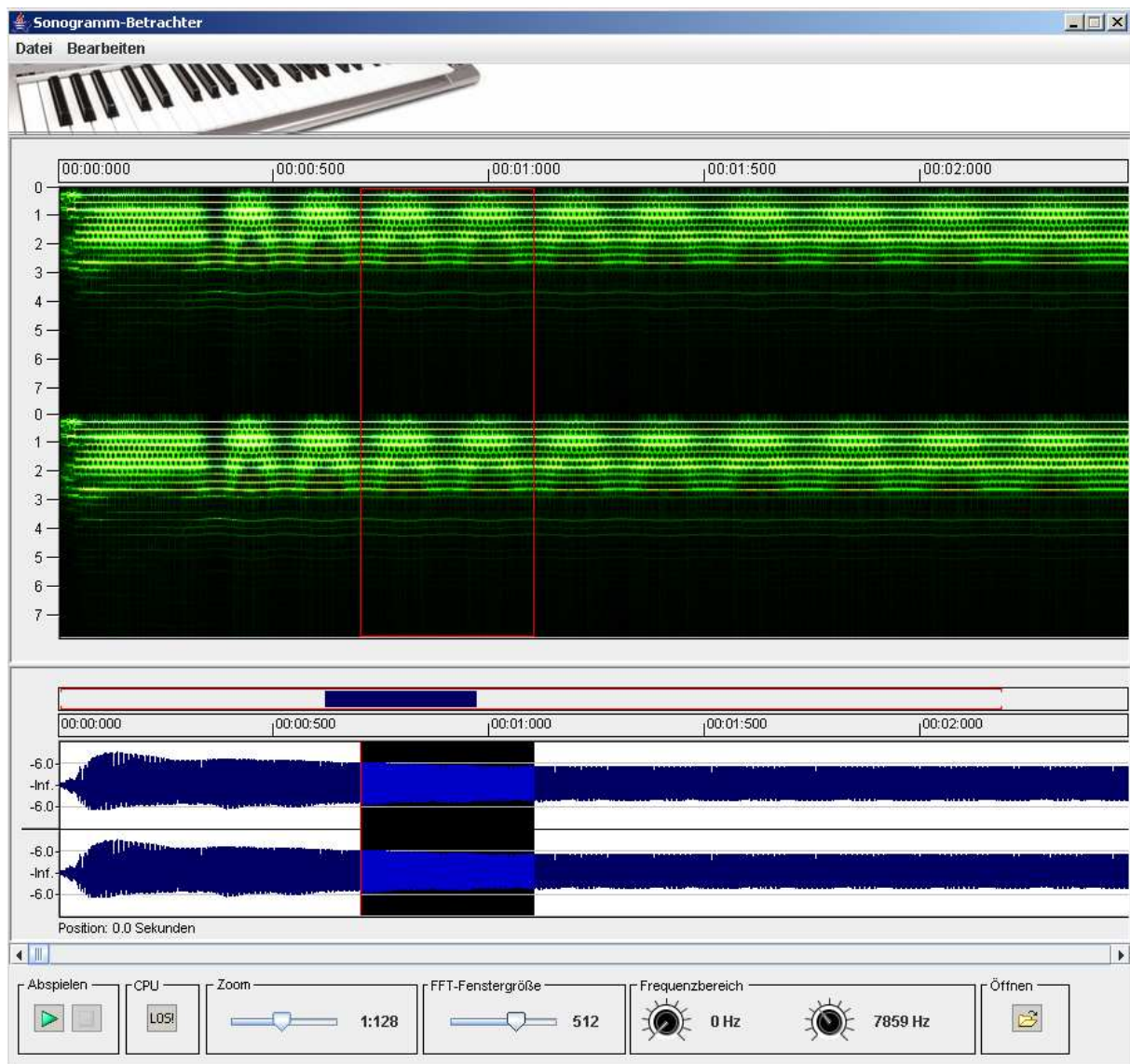
Der Berechnungsfortschritt wird von einer Instanz der Klasse `statusModule` repräsentiert.

Grundlage der Berechnung der Grafik ist neben der FFT-Länge die Länge in Pixeln der Audiodaten im ebenfalls in den Spektrogramm-Betrachter eingebetteten `wavePanel` mit aktuellem Vergrößerungsfaktor. Hieraus ergibt sich auch die Anzahl der durchzuführenden Fouriertransformationen, da eine Transformation genau die Daten für eine Zeiteinheit auf der X-Achse liefert.

Ergebnis der Berechnungen ist eine Instanz der Klasse `Image` der Größe $[Länge\ der\ Daten\ in\ Pixel] * [FFT-Länge]$ (Es wird also eigentlich eine FFT der Länge $FFT-Länge * 2$ berechnet). Aus diesem wird im `sonogramPanel1` entsprechend den aktuellen Einstellungen für das Frequenzintervall, den Vergrößerungsfaktor und die aktuelle Position der entsprechende Ausschnitt errechnet.

Da dieser Vorgang mit Antialiasing sehr rechenaufwändig ist wird, während der Nutzer Einstellungen verändert (`MouseDragged`-Event), auf diese Option verzichtet und erst beim Abschluss der Änderungen (`MouseUp`-Event) ein Ausschnitt mit Antialiasing berechnet. (Diese Option lässt sich in den Einstellungen deaktivieren)

Der Spektrogramm-Betrachter besitzt folgende grafische Benutzeroberfläche:



Wie in der Grafik ersichtlich ist es möglich einen Bereich der Daten nicht nur im `wavePanel1`, sondern auch im `sonogramPanel1` auszuwählen. Diese werden durch die Klasse `sonogramViewer` synchronisiert.

5.3.1. Mögliche Verbesserungen

Eine mögliche Funktion bei einer Weiterentwicklung des Tools wäre eine Wahlmöglichkeit zwischen verschiedenen Farbschemata und ein Farbschema bei dem das gesamte Spektrum der Farben abgedeckt ist.

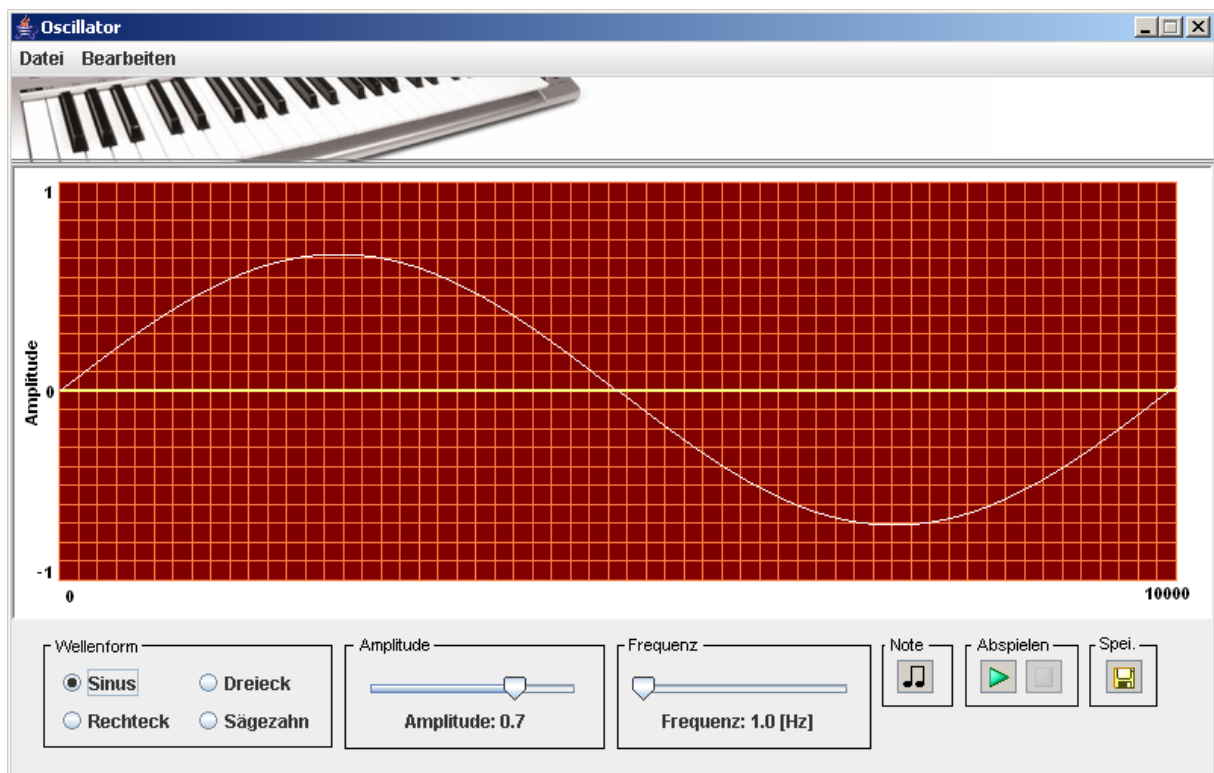
Weiterhin ist die Darstellung aufgrund der Relativität der Farbwahl (der kleinste Wert wird durch Schwarz, der größte Wert durch Weiss repräsentiert) nur bedingt aussagekräftig, weshalb es denkbar wäre einen Darstellungsmodus anzubieten welcher einen festen Zusammenhang zwischen Wert und Farbe herstellt.

Die Möglichkeit der Datenmanipulation in dieser Ansicht wäre ein weiteres interessantes Feature.

5.4. Der Oszillator

Der Oszillator ermöglicht es dem Benutzer, unter Angabe einer Form, einer Frequenz und einer Amplitude, Audiosamples einer beliebigen Länge zu erzeugen.

Auch besteht die Möglichkeit eine Frequenz auszuwählen, welche einer Note entspricht.



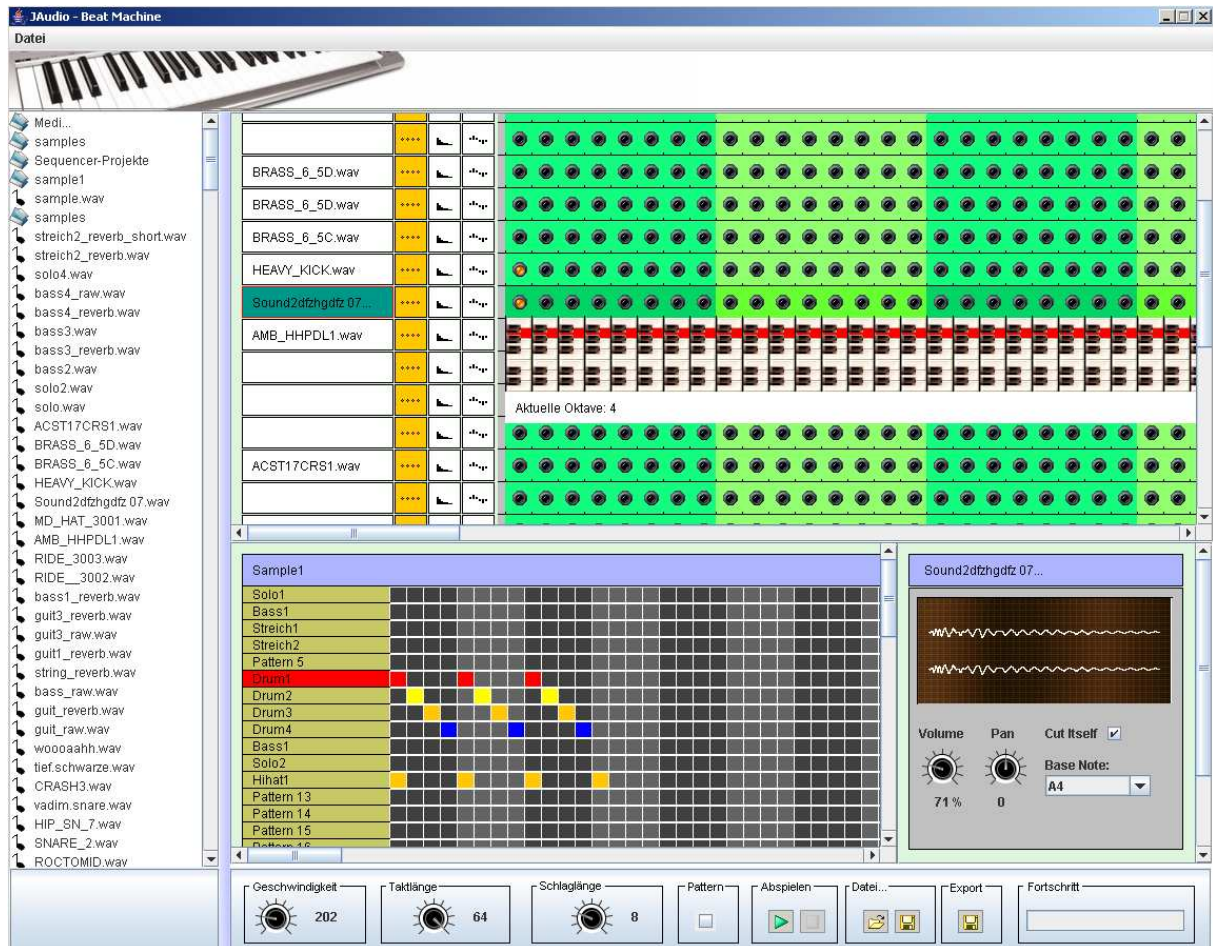
Zur Erzeugung der Audiosamples wurde aufgrund der damit verbundenen Zeitersparnis auf die frei verwendbare Klasse `Oscillator.java` des Autors Matthias Pfisterer zurückgegriffen.

5.4.1. Verbesserungsmöglichkeiten

Die Darstellung der Wellenform ist nicht korrekt und müsste überarbeitet werden.

5.5. Der Sequencer

Der Sequencer bietet dem Benutzer die Möglichkeit sich kreativ auszuleben und Musik zu produzieren. Folgende Benutzeroberfläche wurde dazu geschaffen:



Ein Musikstück besteht hier aus einer Reihe von Patterns. Jedes dieser Patterns besteht wiederum aus einer angegebenen Anzahl von Ticks. Im unteren, mittleren Fenster sind die Patterns angeordnet. Die Zeitachse verläuft von links nach rechts, die Patterns sind von oben nach unten angeordnet. Es können beliebig viele Patterns gleichzeitig gespielt werden.

Ein Pattern wiederum besteht aus beliebig vielen Spuren, auf welche Audiosamples gesetzt werden können. Das Sample der aktuell markierten Spur ist im Fenster rechts unten abgebildet. Dort können auch Lautstärke, Balance und weitere Einstellungen verändert werden.

Innerhalb eines Patterns sind nun jeder Spur (in Y-Richtung angeordnet) eine angegebene Anzahl von Ticks zugeordnet. Die Zeitachse verläuft in X-Richtung. Nun kann nicht nur angegeben werden an welchem Tick welches Audiosample gespielt werden soll, sondern auch mit welcher Lautstärke, mit welcher Balance und sogar mit welcher Tonhöhe. Dazu dient die Einstellung „Base Note“ im Audiosample-Fenster. Ausgehend von diesem Basiston wird das Sample dann auf die im Klavier ausgewählte Tonhöhe gepitcht.

Alle Patterns besitzen die selben Spuren. Man kann einstellen wie viele Millisekunden ein Pattern dauert und in wie viele Ticks ein Pattern eingeteilt wird.

Die Option „Cut-Itself“ einer Audiospur gibt an, ob sich 2 Samples, die auf der selben Spur abgespielt werden, abschneiden, falls das eine noch nicht zu Ende ist oder sie sich überlappen. Zum besseren Verständnis der Funktionsweise des Sequencers sei hier aufgrund des Umfangs auf die Kurzreferenz im Anhang hingewiesen.

Erstellte Songs können entweder als Wave-Datei im Standardformat ausgegeben werden, oder in eine XML-Datei exportiert werden.

Der Sequencer wird durch folgende Klassen realisiert:

Die Klasse `MixerProject` repräsentiert ein Projekt und führt die Spuren mit den Patterns und der Anordnung der Patterns zusammen. Die Patterns wiederum werden von der Klasse `Pattern` repräsentiert. Die Klasse `MixerProjectLoader` eine Unterklasse von `DefaultHandler` für den SAX-XML-Parser ist in der Lage exportierte XML-Dateien zu parsen und daraus Instanzen der Klasse `MixerProject` zu erzeugen. Die Klasse `MixerProjectRenderer` erzeugt eine entsprechende Wave-Datei. Alle Klassen befinden sich im Package `jmusic.gui.creators.mixer`.

Die vom `JFrame` abgeleitete Klasse `Mixer` des Packages `jmusic.gui.creators` bildet den Container für eine von `JPanel` abgeleitete Klasse `MixerPanel`, welches wiederum jeweils eine Instanz der Klasse `ShowPatternPanel`, `ShowPatternArrangementPanel` und `ShowMixerLinePanel` enthält, die sich um die Darstellung und Editierbarkeit der einzelnen Aspekte kümmern.

5.5.1. Mögliche Verbesserungen

Im nachhinein erwies es sich als nicht ideal die Auswahl der in das Projekt integrierbaren Samples auf die Medienbibliothek der Applikation zu beschränken, da der Baum beim Öffnen erzeugt wird und sich Veränderungen in der Bibliothek nicht anpasst. Dies führt dazu, dass man sich vor dem Erstellen eines Projektes sicher sein muss, dass alle benötigten Dateien in der Medienbibliothek verfügbar sind.

Des weiteren dauert das Erzeugen der XML-Dateien wohl aufgrund einer ineffizienten Implementierung recht lange. Hier wäre es bei einer Neuimplementierung oder Weiterentwicklung nötig einen anderen Weg, zum Beispiel unter Verwendung der Klasse `StringBuffer`, zu wählen.

6. MIDI - Musical Instrument Digital Interface

MIDI ist ein Kommunikationsprotokoll zur Übermittlung, Aufzeichnung und Wiedergabe von musikalischen Steuerinformationen zwischen digitalen Instrumenten oder Rechnern. Bei MIDI-Daten handelt es sich nicht um Audiosamples, sondern um einfache Befehle (MIDI-Events genannt), welche an ein Midi-Endgerät, wie zum Beispiel ein Synthesizer, gesendet werden. Das Endgerät interpretiert die Midi-Befehle und kann daraufhin zum Beispiel synthetisierte, digitale Töne generieren. Dabei ist die Palette der möglichen Midi-Befehle nahezu unbegrenzt, es können also nicht nur Töne abgespielt oder Lautstärken verändert werden, sondern, zum Beispiel bei einem Liveauftritt, Einstellungen von Audioverstärkern manipuliert werden..

Heutzutage wird der Midistandard von nahezu allen Soundkarten unterstützt. Hier ein kurzer Überblick über einige beispielhafte Midi-Messages:

Bytes (hexadezimal)	Folgebytes	Status
0x8n kk w	Note, Release velocity	Note Off
0x9n kk w	Note, Anschlagsdynamik	Note On
0xA n kk w	Note, Dynamic	Polyphonic Aftertouch
0xB n cc w	Controller, Wert	Control Change
0xC n pp	Programmnummer	Program Change
0xD n w	Wert	Monophonic Aftertouch
0xE n w [w]	Wert1, Wert2 (optional)	Pitch Bending
0xF n xx...	Geräteabhängig	System Message

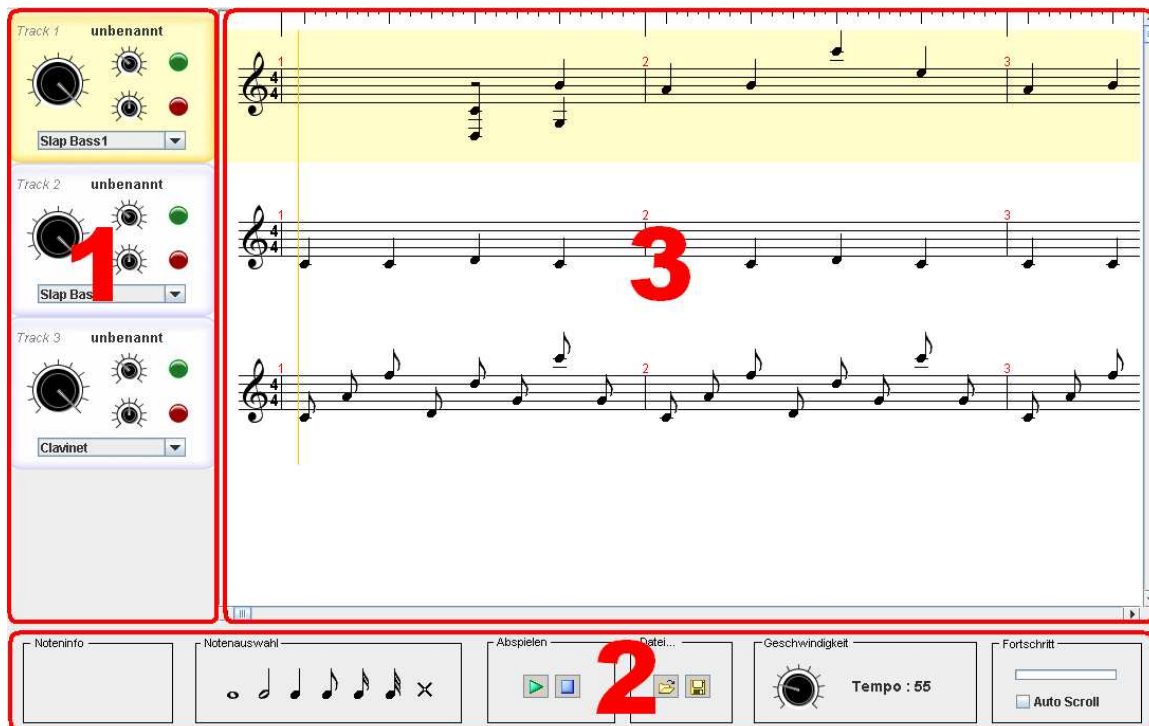
Eine Mididatei stellt schlicht und ergreifend eine Sequenz von Midi-Events dar, die mit Hilfe eines Sequencers ausgelesen und dann von einem Synthesizer abgespielt werden können. Dabei versendet der Sequencer die einzelnen Befehle in Echtzeit an einen in der Soundkarte integrierten Synthesizer, einen Softwaresynthesizer oder auch an ein externes Midigerät.

6.1 Der Midi-Editor

Die Tatsache, dass Midi-Dateien bloß eine Aneinanderreihung von Midi-Events darstellen, führt dazu, dass Einschränkungen vorgenommen werden müssen, um den Arbeitsaufwand im Rahmen zu halten und einen Editor zu erhalten, welcher einfach zu bedienen ist.

Der Midi-Editor ist in der Lage Midi-Dateien der Version 1 zu kodieren und dekodieren. Es ist möglich vorhandene Dateien zu bearbeiten oder neue zu erstellen.

Der Midi Editor besteht aus drei Hauptbereichen, welche in die von `JFrame` abgeleitete Klasse `MidiPaint` des Packages `jmusic.gui.creators` eingebettet sind.



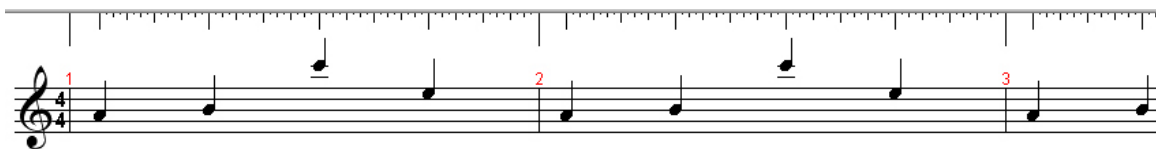
1. Der erste Bereich enthält Instanzen der von `JPanel` angeleiteten Klasse `MidiTrackPanel`, welche Einstellungen für einen gesamten Track einer Sequenz bereitstellen.

Folgende Einstellungen können vorgenommen werden:

- Lautstärke (Volume)
 - Halleffekt (Reverb)
 - Stereoregler (Pan)
 - Mute & Solo
 - Instrument
 - Name
2. Der zweite Bereich besteht aus einer Instanz der Klasse `MidiControlPanel`, mit dessen Hilfe man die Sequenz steuern und editieren kann. Das Panel besteht aus 6 Unterelementen:
 - Informationen zur aktuell markierten Note
 - Notenauswahl, zum Einfügen neuer Noten
 - Abspielen

- Datei speichern / öffnen.
- Tempo / Abspielgeschwindigkeit
- Fortschritt – Enthält eine Progressbar und eine Autoscrollschaltfläche (Mitscrollen beim Abspielen)

3. Der Bereich 3 stellt den Hauptbereich des Editors dar. In ihm können per Mausklick bzw. Drag & Drop Noten gelöscht, eingefügt, kopiert und verschoben werden. Er besteht aus einer Instanz der Klasse `MidiSheetPanel`, welche alle Tracks einer Midisequenz abbildet. Beim Öffnen einer Midi-Datei werden die Tracks in Takte aufgeteilt und je nach Takt Pausen berechnet. Für eine übersichtliche Darstellung der Takte wurde ein Lineal am oberen Rande integriert.



Mit der rechten Maustaste ist es über ein Kontextmenü möglich neue Tracks einzufügen, Tracks zu entfernen oder Noten zu transponieren. Es besteht auch die Möglichkeit einen Track in einen Schlagzeugtrack zu transformieren.

Nachdem man die Tracks vorbereitet hat, kann man mit dem Editieren beginnen. Dabei stehen folgende Operationen zur Verfügung:

- Noten aus dem MidiControlPanel per Drag&Drop auf die Notenlinien setzen. Im gleichen Panel steht auch ein Radiergummi zur Verfügung, um Noten löschen zu können.
- Die Noten auf dem Notenblatt per Drag & Drop verschieben.
- Jede einzelne Note kann individuell bearbeitet werden, wie zum Beispiel:
 - Lautstärke ändern
 - Note um einen Halbtonschritt erhöhen
 - Note punktieren
 - Note löschen
 - Note abspielen
- Eine Gruppe von Noten selektieren. Für die Selektion sind folgende Aktionen verfügbar:
 - Selektion löschen
 - Selektion transponieren
 - Selektion kopieren / ausschneiden
 - Kopierte Selektion auf einem beliebigen Track einfügen
 - Die Lautstärke linear ändern

Eine Hilfsfunktion, die dem Benutzer beim Editieren helfen soll, ist die Auswahl der Noteneingaberresolution. Mit deren Hilfe kann man entscheiden, auf welchen horizontalen Positionen auf den Notenlinien die gerade bewegte Note abgesetzt werden darf.

Es ist zur jeder Zeit möglich die Sequenz abzuspielen oder zu speichern. Der Takt der Sequenz kann allerdings nur am Anfang, beim Erstellen einer neuen Midi-Datei, festgelegt werden.

Eine Midi Sequenz besteht alleine nur aus Midi Events, die ihrerseits aus Midi Messages bestehen. Da diese Events normalerweise in Echtzeit von einem Midigerät (meistens einem Sequencer) an ein anderes Midigerät (meistens einen Synthesizer) verschickt werden, wäre es ziemlich schwer, mit dieser Form der Daten zu arbeiten.

Der erste Schritt ist, diese Midi-Sequenz zu dekodieren und in eine abstraktere Repräsentation zu überführen. Diese Aufgabe wird von der Klasse `JMidiDecoder` des Packages `jmusic.data` erledigt. Die Midi-Sequenz wird geparkt, die wichtigsten Information extrahiert und eine Instanz der Klasse `JDecodedMidiData` erzeugt. Dieses Objekt enthält Informationen wie z.B. die Taktinformation, die Anzahl der Takte, Anzahl der Tracks, Tempo und eine der Anzahl der Tracks entsprechende Anzahl von Arrays mit Instanzen der Klasse `JMidiNote`, welche eine Note repräsentiert. Diese Klasse enthält alle Informationen, wie die Bezeichnung, die Länge, die Tonhöhe, den Anfangszeitpunkt, den Endzeitpunkt und den dazugehörigen Track und Takt. Für die grafische Darstellung der Noten enthält das Objekt zusätzlich ein entsprechendes `ImageIcon`.

Anhand der Noten in einem Takt sind wir auch in der Lage die entsprechenden Pausen zu berechnen. Zu diesem Zweck werden die vorhandenen Leerräume in den Takten analysiert und durch entsprechende Pausen ersetzt. Ergebnis ist wieder eine der Anzahl der Tracks entsprechende Anzahl von Arrays von Objekten vom Typ `JMidiPause`.

Die Klasse `MidiSheetPanel` zeichnet diese dekodierten Daten dann auf ein interaktives Notenblatt auf dem Bildschirm. Dieses Notenblatt kann man dann mit der Maus steuern und die Daten editieren. Die einzelnen Editieroperationen werden von einer Instanz der Klasse `MidiSequenceManipulator` direkt auf der Midi-Sequence ausgeführt.

6.1.1 Mögliche Verbesserungen

Ein Performanzgewinn ließe sich wahrscheinlich durch eine Neugestaltung der zugrundeliegenden Strukturen erreichen. Die abstrakte Repräsentation der Midi-Daten (also die Klassen `JMidiNote`, `JMidiPause`, `JDecodedMidiData`) bilden zum jetzigen Zeitpunkt nur die Grundlage der Darstellung der Daten.

Manipulationen werden ausschließlich auf der zugrundeliegenden Midi-Sequenz durchgeführt, welche deshalb nach jeder Änderung neu geparkt und dekodiert werden muss. Hier wäre es wohl bei einer Weiterentwicklung geschickter die vorhandenen Klassen auszubauen und ein Kodieren bzw. Dekodieren der Daten nur im Falle einer Öffnung, Speicherung oder vor dem Abspielen durchzuführen.

7. Fazit

Bedingt durch die Tatsache, dass wir zu Beginn unserer Arbeit auf den vorliegenden Themengebieten sehr unerfahren gewesen sind, ist das Programm Stück für Stück gewachsen. Dabei entstand ein mächtiges Werkzeug mit vielen Möglichkeiten zur Erstellung, Bearbeitung und Analyse von Audio- und Mididateien, wobei auch auf eine ansprechende Benutzeroberfläche geachtet wurde.

Da uns das Thema sehr interessiert und zunehmend immer mehr begeistert hat konnten wir durch einen oftmals großen Arbeitsaufwand viele Features verwirklichen von denen wir eigentlich dachten sie wären zu anspruchsvoll oder aufwändig. Gleichzeitig mussten wir feststellen, dass manche Tücken im Detail liegen, was oftmals zeitaufwändige Korrekturen nötig machte.

Unter Berücksichtigung der Erfahrungen und Kenntnisse, welche wir bei der Durchführung dieses Projektes erlangt haben, würden wir bei einer neuerlichen Umsetzung trotzdem manches anders machen, um in einigen Abschnitten des Programms mehr Performanz und Struktur zu erlangen. Vieles wurde, bedingt durch das langsame Wachsen der Funktionalität, speziell den hier vorliegenden Problemstellungen angepasst und die Möglichkeiten einer objektorientierten Sprache nicht ausreichend in Anspruch genommen.

Die Wahl von Swing-Komponenten für die graphische Benutzeroberfläche zeichnet sich im Nachhinein als nicht ideal ab. Zum einen gehen die dort vorhandenen Möglichkeiten, leider auf Kosten der Performanz, weit über das eigentlich Benötigte hinaus. Bei einer späteren Testimplementierung auf Basis von awt-Komponenten konnte bei gleicher Darstellung ein deutlicher Performanzgewinn festgestellt werden.

Zum anderen wurde das Single-Thread-Prinzip von Swing teilweise nicht ausreichend berücksichtigt. Hier würden wir, bei einer neuerlichen Implementierung stärker darauf achten rechenintensiven Vorgänge in andere Threads auszulagern.

Trotzdem ist es mit dem resultierenden Programm möglich auch anspruchsvolle Aufgaben zu lösen und seiner Kreativität freien Lauf zu lassen. Eine vergleichbare Kombination aus Audibearbeitung, Analysewerkzeugen, einem Sequencer und einem Midi-Editor ist uns in einem bezahlbaren Preissegment bisher nicht bekannt.

8. Quellenangaben

<http://www.developer.com/tech/article.php/617571> - Stand 23.11.06

<http://java.sun.com/products/java-media/sound/> - Stand 23.11.06

<http://www.jsresources.org/> - Stand 23.11.06

<http://www.eumus.edu.uy/docentes/jure/docs/>

Niemitalo_DSPForTheBraindead.html - Stand 23.11.06

<http://de.wikipedia.org/wiki/Midi> - Stand 23.11.06

http://de.wikipedia.org/wiki/WAV_%28Format%29 - Stand 23.11.06

<http://www.developer.com/java/other/article.php/3457251> - Stand 23.11.06

<http://www.informatik.uni-hamburg.de/~schaetti/musiksynth/technik/>

MIDI.html - Stand 23.11.06

<http://www.midi.org/about-midi/specinfo.shtml> - Stand 23.11.06

9. Anhang