

# MIDI VIDI VICI

Software zur Visualisierung von Musik

Lennart Kruse

Medienkonzeption Spezial, WS 06/07

## Inhaltsverzeichnis

1	Grobkonzept	
	• Ideefindung	S. 3
	• Das MIDI-Format	S. 5
	• Technik	S. 7
2	Feinkonzept	
	• Programmablauf	S. 11
	• Modulübersicht	S. 13
	• Einlesen und verarbeiten von MIDI-Dateien	S. 14
	• Kommunikation zwischen Java und Flash	S. 15
3	Ausblick und Fazit	S. 17
4	Anhang	S. 19

## 1 Grobkonzept – Ideenfindung

In der Vorlesung „Medienkonzeption Spezial“ war ein Konzept zu entwickeln, das in der zugehörigen Vorlesung „Medienproduktion Spezial“ umgesetzt werden sollte. Das vorgeschlagene Thema dieses Konzeptes lautete „Visuelle Musik“, bzw. die Visualisierung von Musik. Es waren auch eigene Themen möglich, aber dieses entsprach meinen Interessen: Nicht speziell die Visualisierung, aber der allgemein digitale Umgang mit Musik vereinigt zwei Gebiete, die mich auch privat beschäftigen. Deshalb reizte mich sogar die „feste“ Vorgabe eines Themas – ich war gespannt auf mein eigenes Ergebnis.

Visuelle Musik funktioniert erstmal in beide Richtungen: Ausgehend von etwas visuellem kann man – in welcher Form auch immer – Töne erzeugen, oder man setzt Audioinformationen auf visuellem Weg um. Das kann das „Medium“ Mensch erledigen, wie im Beispiel einer Lichtorgel. Ich als Fast-Informatiker sah mich allerdings eher durch einen algorithmischen Weg der Umsetzung herausgefordert. Dies ist nicht trivial: Unabhängig von der Richtung des Informationsflusses (auditiv nach visuell oder vice versa) ist das Erkennen und Interpretieren von Mustern nötig; ein aktuelles Forschungsgebiet der Wissenschaft, das auch auf absehbare Zeit nicht abgeschlossen sein wird.

## 1 Grobkonzept – Ideenfindung

Deshalb traf ich schon in dieser frühen Phase die Entscheidung, mich auf MIDI-Daten zu beschränken (vgl. nächstes Kapitel). Im Gegensatz zur visuellen Welt gibt es mit diesem Format ein auditives Medium, das hinreichend einfach maschinenlesbar, und gleichzeitig durchaus verbreitet ist. Natürlich nicht so weit verbreitet wie etwa MP3; außerdem mittlerweile eher veraltet. Aber es wurde z.B. in Mobiltelefonen eingesetzt, um die so genannten „polyphonen Klingeltöne“ zu realisieren.

Das algorithmische Umwandeln von MIDI-Dateien in visuelle Form bildete also die Basis meiner Idee. Mir wurde jedoch schnell klar, dass vergleichbare Funktionen, und zwar für „alle“ Audioformate, in jeder aktuellen Audioplayersoftware bereits vorhanden sind. Es fehlte ein Aspekt, der das Ergebnis von diesen unter-

scheidbar macht – abgesehen von meiner Beschränkung auf MIDI. Den Ausschlag gab die Vorstellung eines anderen Projektes: Auch dort fehlte der letzte Schliff, und es wurden verschiedene Vorschläge gemacht, das Projekt interaktiver zu gestalten. Das wollte ich auch für mein Projekt erreichen: Durch den „Spieltrieb“ einen Anreiz zu schaffen, sich mit der Software zu beschäftigen, der den Nachteil der Beschränkung auf MIDI vielleicht mindern könnte. Dazu sollte die Visualisierung zwar immer noch algorithmisch erfolgen, die Parameter der Visualisierung sollten jedoch frei definierbar, bzw. editierbar sein.

Man kann sich das fertige Produkt folglich wie einen Musikvideogenerator vorstellen, der automatisiert arbeitet, aber trotzdem „Videos“ im eigenen Stil des Users erzeugt.

## 1 Grobkonzept – Das MIDI-Format

Im Folgenden soll eine Kurzübersicht über das MIDI-Format gegeben werden, soweit es für das Projekt relevant ist. Eine vollständige Spezifikation kann im Internet gefunden werden (1)(2). Eine MIDI-Datei besteht nicht aus Audioinformationen, wie z.B. die Wellenform einer WAV-Datei. Sie enthält viel mehr Steuerzeichen, die hier an die Soundhardware eines Computers gesendet werden, wo sie in Töne umgewandelt werden. Etwas vereinfacht (eine vollständige Liste hängt auch von der verwendeten Hardware ab) gibt es die folgenden Signale:

Note on:  
Spielt einen Ton, bestehend aus Tonhöhe und Anschlagsstärke

Note off:  
Beendet einen Ton, ebenfalls mit einer gewissen „Stärke“ (Fade out)

Polyphonic Aftertouch:  
Ändern des Drucks auf eine bereits gedrückte Taste

Monophonic Aftertouch:  
Ändern des Drucks auf alle Tasten gemeinsam

Control Change:  
Änderung an einem Controller (Dämpfung, Gesamtlautstärke, Chorus, ...)

Program Change:  
Wechseln des Instrumentes

Pitch Bending:  
Ändern der Tonhöhe

System Message:  
Gerätespezifische Steuermeldungen

## 1 Grobkonzept – Das MIDI-Format

Diese Befehle arbeiten immer auf einem bestimmten „Kanal“, der genau einem Instrument (zur Zeit) entspricht. Im Regelfall gibt es 16 Kanäle, durch bestimmte Techniken kann diese Zahl stark erhöht werden, dies soll hier aber nicht beachtet werden. Dadurch sind die Möglichkeiten der Klangerzeugung stark eingeschränkt, was es ermöglichte, ein entsprechendes Signal im Rahmen dieses Projektes verarbeiten zu können. Die beschriebenen Signale ergeben einen begrenzten Satz an „Events“, die zusammengesetzt das Audiosignal generieren lassen. Diese Events galt es nun, auszulesen.

## 1 Technisches Grobkonzept

Für die Visualisierung wollte ich gerne Adobe Flash (3) einsetzen, da diese Umgebung einfachere Möglichkeiten der visuellen Gestaltung bietet, als sie etwa mit Java (4) möglich sind. Flash bietet von Haus aus keine Möglichkeit zur Wiedergabe von MIDI, diese Funktionalität muss mittels eines Plugins nachgerüstet werden (5).

Dieses Plugin ermöglicht allerdings nicht den Zugriff auf die zur Analyse des „Audiosignals“ nötigen Informationen der Steuerzeichen, es spielt diese nur ab. Das Dateiformat selber ist „verschlüsselt“, die Informationen liegen innerhalb der Datei nicht in Reintext vor. Für das Auslesen der Datei muss also auf eine mächtigere Umgebung als Adobe Flash zurückgegriffen werden. Was die Unterstützung von „exotischen“ Dateiformaten im Allgemeinen und

deren weitere Verarbeitung im Speziellen angeht, bietet sich Java mehr an, schon durch das komfortable Package-System, das das Einbinden von externer Funktionalität stark vereinfacht.

Tatsächlich wird MIDI schon mit den Standard-Bibliotheken (genauer: `javax.sound.midi.*`) rudimentär unterstützt, durch das Plugin `jMusic` (6) werden außerdem stark darüber hinausgehende Funktionen angeboten. Als Beispiel die folgende Grafik, die eine Darstellung einer MIDI-Datei in Notenform zeigt – der diese Ansicht effektiv erzeugende Programmcode ist eine Zeile lang:

## 1 Technisches Grobkonzept



Die MIDI-Steuersignale lassen sich ähnlich leicht auslesen (vgl. Kapitel „Einlesen und Verarbeiten der MIDI-Dateien“), womit sich nur die Frage nach ihrer Verwertung stellt. Wie

bereits erwähnt, ist grafische Ausgabe, egal in welcher Form, nicht trivial unter Java. Ich blieb deshalb bei meiner Entscheidung, die Visualisierung in Flash zu implementieren, die Daten müssen also zwischen zwei Programmmodulen ausgetauscht werden. Der entsprechende Code sollte aber trotzdem einfacher sein als die Visualisierung über Java (vgl. Kapitel „Kommunikation zwischen Java und Flash“).

Aus den erwähnten Möglichkeiten des MIDI-Formates ergeben sich bestimmte Eigenschaften, die das auszugebende Musikstück definieren, und die unterschieden werden können in „globale“, allgemeine Eigenschaften eines Stückes, und die „lokalen“ Ereignisse, die abhängig von der aktuellen Abspielposition sind. Auf diese Eigenschaften könnte eine dynamische Visualisie-

## 1 Technisches Grobkonzept

nung reagieren, diese Daten wären also von Java auszuwerten und von Flash zu verarbeiten.

Globale Eigenschaften:

- Kürzeste, längste Note; Durchschnitt
- Tiefster, höchster Ton; Durchschnitt
- Verwendete Instrumente
- Anzahl an Instrumenten
- Pitches: Anzahl, Dauer, ...
- Geschwindigkeit
- Dur/Moll
- Anzahl der verwendeten Kanäle

Lokale Ereignisse:

- (bestimmte) Note an/aus
- Lautstärke eines (bestimmten) Tones erhöhen/verringern
- Lautstärke aller Töne erhöhen/verringern
- Tonhöhe ändern
- Instrument wechseln
- (Geschwindigkeit ändern)

Die von mir „globale Eigenschaften“ genannten Informationen kann man sich auch als eine Art „Histogramm“ des MIDI-Stückes vorstellen. Sie stellen eine Art grundsätzliches Setting des Stückes dar, und könnten als solches auch für ein „Setting“ des resultierenden Videos verwendet werden. Das Prinzip wird am klarsten, wenn man an Dur/Moll denkt: Ein Video eines Dur-Liedes sieht (potentiell) erstmal anders aus als das Video eines Moll-Liedes, unabhängig von der speziellen Melodie usw.

Sowohl MIDI-Histogramm, wie auch die Playback-Events können vom User mit Flasheffekten verknüpft werden, vergleiche nächstes Kapitel. Jeder Event kann dabei mit einem Flash-Effekt verbunden werden, außerdem können die möglichen

## 1 Technisches Grobkonzept

grundlegenden Parameter des Settings mit ihren visuellen Pendanten verknüpft werden. Zusammen ergeben sie ein Profil, das auch gespeichert werden kann, und das zu Beginn der Visualisierung an den Flashteil des Programms übergeben wird. Nun sind nur noch die Events zu übergeben, sobald sie auftreten, was den Datenverkehr minimiert. Flash spielt dann „on the fly“ die nötigen Effekte ein. Zur weiteren Individualisierung sind die vorhandenen Effekte erweiterbar, sie können als einzelne Flashdateien gespeichert werden, die dynamisch eingelesen werden.

Eine Erweiterung der Events, also ein Reagieren auf Events, die von der bisherigen Struktur nicht abgefangen werden, ist nicht vorgesehen. Dies macht auch keinen Sinn, da das MIDI-Format nur bestimmte

Events zulässt, die alle behandelt werden. Einzige Ausnahme stellen spezielle Controllerbefehle dar, die nur von bestimmten Endgeräten unterstützt werden. Aber auch diese können potentiell mit der hier konzeptionierten Software abgefangen werden: Auch spezielle Kommandos können doch nur in der engen Struktur der MIDI-Befehle formatiert sein – diese Software könnte sie alle abfragen (es sind nur begrenzt viele), und dem User für eine Verknüpfung mit Effekten anbieten. Ob die Kommandos im speziellen MIDI-File benutzt werden, ist an der Stelle genauso irrelevant wie ihre ursprüngliche Bedeutung für die Hardware, für die sie konzipiert wurden. Um die Übersichtlichkeit für den Anwender zu erhalten, sollen sie im Folgenden aber nicht beachtet werden.

## 2 Programmablauf

Es müssen folgende Anwendungsfälle unterschieden werden:

a. Programmstart:

Es wird der Javateil des Gesamtprogramms geladen. Dieser durchsucht ein Verzeichnis, in dem sich die verfügbaren Flasheffekte befinden.

b. Es sollen die Visualisierungsparameter (das „Profil“) geändert werden:

Der User bekommt einerseits die insgesamt möglichen Events und Settingparameter angeboten, und andererseits die verfügbaren Effekte. Er kann diese nun verbinden, wobei unterschiedliche Detailgrade möglich sind: Das Ereignis Note an/aus kann aufgesplittet werden auf jeden Kanal, auf jedes Instrument und/oder bestimmte Noten. Die getroffenen Einstellungen werden im aktuellen Profil gespeichert.

c. Ein Profil soll gespeichert werden.

Der User wählt die entsprechende Funktion an, kann einen Namen vergeben, und das Programm speichert die aktuellen Einstellungen unter einer Datei des angegebenen Namens

d. Ein Profil soll geladen werden.

Der User wählt die entsprechende Funktion an, kann eine Profildatei auswählen, und das Programm überschreibt die aktuellen Einstellungen mit denen des gewählten Profils

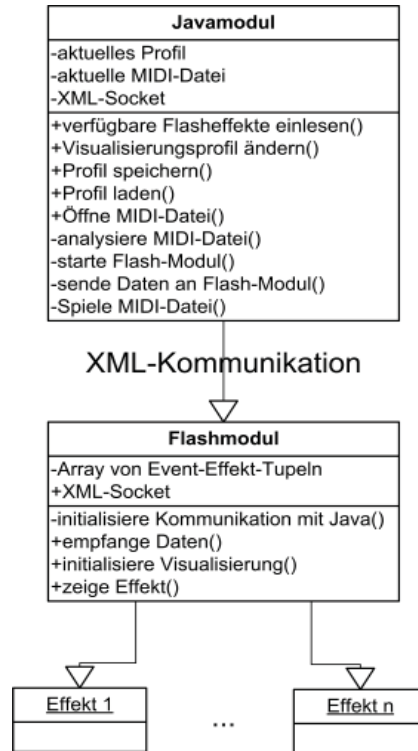
e. Es sollen neue Visualisierungen eingebunden werden. Neue Effekte kann der User mit Flash erstellen, und dort auch abspeichern. Jede Form von Flashdatei ist möglich, da diese dynamisch eingelesen werden.

## 2 Programmablauf

f. Der User will eine MIDI-Datei visualisieren lassen: Er öffnet die entsprechende MIDI-Datei über ein Interface, das ihm von dem in Java implementierten Teil des Programms angeboten wird. Das Programm analysiert dann die Datei, und startet den für die Visualisierung zuständigen und in Flash implementierten Teil des Programms. Es wird das Visualisierungsprofil von Java an Flash übertragen. Flash initialisiert die entsprechenden Module, die dem Profil nach benötigt werden. Dann wird das Playback auf Seiten von Java gestartet. Während der Wiedergabe auftretende „Events“ werden an Flash übergeben, das sie entsprechend darstellt.

## 2 Modulübersicht

Aus dem oben beschriebenen Programmablauf lässt sich der rechts dargestellte Modulaufbau der Software ableiten (schematische, nicht UML-konforme Darstellung):



## 2 Einlesen und Verarbeiten der MIDI-Dateien

Das Einlesen ist abhängig von der gewünschten Weiterverarbeitung. Soll das MIDI-File „nur“ abgespielt werden, so ist dies mit den Java-API-eigenen Mitteln möglich: Es wird eine Instanz eines virtuellen MIDI-Sequencers erzeugt, der abhängig von der jeweiligen Hardware (der Soundkarte) ist:

```
Sequencer sequencer =  
    MidiSystem.getSequencer();
```

Diesem Sequencer kann dann eine Sequenz zugewiesen werden, die aus einem MIDI-File gelesen wird. Der Sequencer spielt diese Sequenz dann ab:

```
sequencer.open();  
File file = new File(„eineDatei.mid“);  
Sequence seq = MidiSystem.getSequence(file);  
sequencer.setSequence(seq);  
sequencer.start();
```

Das Zugreifen auf die MIDI-Steuerdaten ist über ein „Score“-Objekt der jMusic-Bibliothek möglich. Dieses Objekt kann dann auf vielerlei Arten ausgegeben werden – etwa, wie im eingangs gezeigten Beispiel, als Notenblatt, oder als Stringliste von MIDI-Befehlen:

```
Score score = new Score(„tempScore“);  
Read.midi(score, „eineDatei.mid“);  
View.notate(score);  
View.print(score);
```

Nun würde man die extrahierten Steuersignale einzeln an die Soundkarte schicken, um während der so aufgesplitteten Wiedergabe jederzeit auftretende Events an das Flash-Modul schicken zu können (vgl. nächstes Kapitel).

## 2 Kommunikation zwischen Java und Flash

Die Kommunikation zwischen Flash und Java läuft über einen so genannten XML-Socket. Das hat den Vorteil, dass eine Socketverbindung – die normalerweise für die Kommunikation über ein Netzwerk gedacht ist – ständig besteht. Flash bietet neben dieser Form der Kommunikation auch z.B. http-Kommunikation an, doch ist http ein verbindungsloses Protokoll; eine Verbindung müsste bei jedem Event neu aufgebaut werden, was für eine Realtimeanwendung wie im vorliegenden Fall ungeeignet ist.

Das Java Programm erzeugt einen Socket-Server unter einen bestimmten Port (hier: Port 2001), startet dann die Flashdatei, und wartet erst dann auf eine Antwort der Flashdatei. Während auf ein Signal am angegebenen Port gewartet wird, kann Java nichts anderes unternehmen

(nicht ohne einen zweiten Thread), daher ist diese Reihenfolge der Einfachheit halber angebracht. Der folgende Code funktioniert nur auf Windowsystemen! Da ein Flashfilm nicht direkt über einen Konsolenbefehl zu starten ist, wurde an dieser Stelle eine bat-Datei als Zwischenschritt gewählt, die in dieser Form nur auf Windowsystemen lauffähig ist. In der Umsetzungsphase wäre diese Lösung zu erweitern.

```
ServerSocket mySocketServer =  
    new ServerSocket(2001);
```

```
// nur Windows-Systeme:  
Runtime.getRuntime().exec("startflash.bat");
```

```
Socket mySocket = mySocketServer.accept();  
mySocketServer.close();
```

Der Socketserver kann geschlossen werden, wenn die Socketverbindung besteht.

## 2 Kommunikation zwischen Java und Flash

Über diese Verbindung kann nun byteweise geschrieben werden – ein String muss dazu erst in seine Bytes umgewandelt werden:

```
InputStream in = mySocket.getInputStream();
OutputStream out =
    mySocket.getOutputStream();
BufferedOutputStream buffout =
    new BufferedOutputStream(out);
String s = „zu schreibender Text“;
buffout.write(s.getBytes()); // 8 bit ASCII
buffout.write(0);
buffout.flush(); // Cache leeren
```

Auf Flashseite wird ebenfalls ein Socket erzeugt, der sich zu dem entsprechenden Server (hier: Der „Localhost“ unter der IP-Nummer 127.0.0.1) und dem Zielport 2001 verbindet. Wird etwas empfangen, so wird eine entsprechende Eventhandler-Funktion aufgerufen:

```
var socket:XMLSocket = new XMLSocket();
socket.onData = function(data) {
    // hier wird auf Input reagiert
};
socket.connect("127.0.0.1", 2001);
```

Das Weiterbestehen der Verbindung kann auf Javaseite über das „Abhören“ des Sockets realisiert werden:

```
do { } while (in.read(new byte[1], 0, 1) > -1);
in.close();
out.close();
mySocket.close();
```

Die while-Schleife wird nur abgebrochen, so lange am Port eine eingehende Verbindung anliegt. Im fertigen Programm würde diese Schleife durch eine entsprechende MIDI-Wiedergabe ersetzt werden, die bei Verlust der Verbindung ggf. abgebrochen wird.

### 3 Ausblick und Fazit

Es bleiben einige offene Punkte für die Umsetzungsphase: Zuerst sei die grafische Oberfläche genannt, ein übersichtliches Verknüpfen von Event und Effekt stellt sicher eine Herausforderung dar.

Des Weiteren wäre eine Editorfunktion für neue Effekte wünschenswert. Dies gestaltet sich allerdings schwierig, da das Flashformat ein geschlossenes ist, und deshalb nicht ohne weiteres von einem eigenen Programm ausgegeben werden kann. Evt. wäre ein schon vordesignter „Container-Movieclip“ nützlich, der mit Flash geöffnet und nur angepasst werden müsste. Dies würde aber fast sicher die Anzahl an Möglichkeiten eines neuen Effektes einschränken.

Trotzdem bleibt der Kritikpunkt, dass das Erzeugen neuer Effekte für die meisten User nicht trivial ist – davon abgesehen, dass Flash nicht kostenfrei ist, und deshalb den meisten Usern nicht zur Verfügung steht.

Auf der anderen Seite gibt es genug erfolgreiche Beispiele für Software, die vergleichsweise komplexe Funktionen zur Erweiterung und Individualisierung bietet, die auch tatsächlich genutzt wurden. Das geht bei prominenten Beispielen wie dem Computerspiel „Counterstrike“ los, das eine Erweiterung des Spiels „Half-Life“ ist, und endet noch nicht bei der auch in diesem Projekt verwendeten Technik der Java-Packages.

### 3 Ausblick und Fazit

Alles in allem stellt das vorliegende Konzept eine eher mächtige als dem unbedarften User eingängige Lösung dar, was letztlich aber auch der Ausgangsidee entspricht. Wie in den ersten Kapiteln beschrieben, existiert bereits Visualisierungssoftware, etwa als Plugin für Wiedergabesoftware. Oft kann der User zwischen verschiedenen Stilen wählen. Ziel dieses speziellen Projektes war es, diese existierenden Möglichkeiten um eine zusätzliche Dimension zu erweitern, und in diesem Sinne geht der genannte Kritikpunkt konform mit der Projektidee.

## 4 Anhang - Quellennachweis

- (1) <http://de.wikipedia.org/wiki/Midi>
- (2) <http://home.snafu.de/sicpaul/midi/midi0a.htm>
- (3) <http://www.adobe.com/de/products/flash/flashpro/>
- (4) <http://java.sun.com/>
- (5) <http://www.dlexisisaac.net/products/flashMidi/>
- (6) <http://jmusic.ci.qut.edu.au/>