Technische Universität Ilmenau
Fakultät für Elektrotechnik und Informationstechnik
Fakultät für Informatik und Automatisierung

Medienprojekt

Entwurf einer plattformunabhängigen Konfigurationssoftware für die Bedienoberfläche eines MIDI-Controllers

vorgelegt von: Volker Dümke
Jacob Korn

Verantwortlicher Professor:
Prof. Dr.-Ing. Karlheinz Brandenburg
Dr.-Ing. Heinz-Dietrich Wuttke

Betreuender wiss. Mitarbeiter:
Dipl.-Ing. Ulrich Reiter

Beginn der Arbeit: 1.4.2006

Ende der Arbeit: 30.9.2006

Ilmenau, den 2. Oktober 2006

Inhaltsverzeichnis

1 Einführung

1.1 Einleitung / Motivation

Seit vielen Jahrzehnten sind Musiker und Toningenieure darum bemüht, ergänzend zu den bekannten Klängen der akustischen elektroakustischen und mechanischen Instrumente, neuartige unnatürliche oder skurrile Klänge zu verwenden. Das technische Hilfsmittel hierfür war und ist der Synthesizer in seinen verschiedensten Ausprägungen der Klangsynthese. Während anfangs natürliche Instrumente mehr oder minder geglückt imitiert wurden, versuchten Klangforscher im Laufe der Zeit die Grenzen des akustischen Spektrums auszuloten.

In älteren sowie aktuellen Synthesizern werden verschiedene Ideen der Bedienung verfolgt. Auf eine übersichtliche Darstellung und ergonomische Veränderung der Klangparameter wurde bei der Entwicklung der Geräte wenig Wert gelegt. Speziell in den 90er Jahren wurden aus Kostengründen Synthesizer in 19 Zoll Rack-Bauweise gefertigt (siehe Abbildung 1.1) und mit minimalsten Bedienelementen ausgestattet. Dadurch lassen sich solche Geräte sehr schwer und mühselig bedienen. Da aber die meisten über eine MIDI Schnittstelle verfügen, die es ermöglicht sämtliche Parameter über MIDI Nachrichten zu steuern, liegt die Idee nahe, die Bedienbarkeit auf diesem Wege zu verbessern. Hardware-MIDI-Controller ermöglichen im Idealfall eine schnelle intuitive Bedienung durch übersichtlich und ergonomisch angeordnete Taster und Drehregler. Sind diese gut beschriftet, ist eine optimale Arbeitsweise, auch im Echtzeitbetrieb, am Gerät gewährleistet. In enger Zusammenarbeit mit dem Medienprojekt „universelle Bedienoberfläche für MIDI-Synthesizer“ wurde ein MIDI-Controller entwickelt, der es ermöglicht verschiedene Synthesizer komfortabel zu programmieren. Da dieser Controller über zwei große 240x64 Pixel-Displays verfügt, ist neben einer guten Reglerbeschreibung, welche mit einer „Beschriftung“ gleichgesetzt werden kann, auch eine Navigation auf dem Gerät möglich. Dieser MIDI-Controller soll im späteren Studiobetrieb nicht nur einfach zu bedienen sein, sondern auch universell für die verschiedenen Synthesizer konfigurierbar sein. Zu diesem Zweck beschäftigt sich das vorliegende Medienprojekt mit der Entwicklung einer Software, welche eine effiziente und einfache „Programmierung“ des MIDI-Controllers ermöglicht. Somit wird erst die volle Funktionalität der Hardware und die Bedienung von nahezu jedem MIDI-fähigen Synthesizer sichergestellt.

Abbildung 1.1: Beispiel für 90er Jahre-Bedienoberfläche

1.2 Grundlagen MIDI

1.2.1 Synthesizer-Ansteuerung

Die Ansteuerung von Synthesizern ist nicht ganz neu. Bereits in den 70er Jahren verfügten Synthesizer über Ein- und Ausgänge, um mit Drumcomputern und Sequenzern zu kommunizieren. Dies erfolgte über Sync oder auch CV (Control Voltage) und funktionierte leider nur monophon. Dieser Standard wurde dann Anfang der 80er Jahre durch MIDI (Musical Instruments Digital Interface) abgelöst, welches beispielsweise polyphon arbeitet und ebenso noch zusätzliche Funktionen der Echtzeitsteuerung und Datenübertragung zwischen Synthesizern und Sequenzer/Computer ermöglicht, z.B. für Speicherung von Sequenzen, Songs, und Klangprogrammen. Das beste Beispiel für einen gut bedienbaren und „musikalischen“ Synthesizer ist mit Abstand der Moog „Minimoog“. Dieser ermöglichte den Zugang zu den wichtigesten Klangparameter über eine übersichtliche Oberfläche und große, gut bedienbare Dreh- und Kippschalter. Dieses Instrument inspirierte viele spätere Synthesizer-Hersteller. Typische Klanggestaltungsparameter eines subtraktiven Synthesizers, welche in den 70er Jahren auf der Frontplatte editierbar waren, sind z.B. Amplituden-/Pitch-/Filterhüllkurven, Filtergrenzfrequenz/ -resonanz, Oszillatorwellenform/-frequenz, Niederfrequenzoszillatoren mit Modulationszielen und einstellbarer Tiefe, Frequenz und Wellenform. Das sind nur einige von vielen Parametern, welche auch in Echtzeit steuerbar sein sollten, da sie einen enormen Einfluss auf die Klangerzeugung haben und somit auch die Ausdrucksmöglichkeit des Spielenden erweitern. Nur wenige Hersteller konnten aber aus Kostengründen mit dieser Bedienoberfläche mithalten. Diese Klangparameter sind heute über handelsübliche MIDI-Controller-Boxen steuerbar, wobei der Controller über eine Software programmiert werden kann, um dem Synthesizer speziell angepasst zu werden. Der Controller kann dann dem Synthesizer entsprechend bezeichnet werden und somit als Ersatz für die fehlenden Regler am Synthesizer, auch stand-alone Verwendung finden. Man kommt dem Idealkonzept des Minimoogs somit näher und erweitert es sogar noch um die Möglichkeit der Klangprogrammspeicherung und des Total Recalls.

1.2.2 MIDI-Protokoll

MIDI (Musical Interface Digital Interface) ist eine digitale Schnittstelle, welche seriell arbeitet und der Datenstrom auf Empfängerseite in jeweils 8bits, also einem Byte zusammengesetzt wird. Abzüglich des Statusbytes (das Bit 7 aus dem Byte entscheidet über Status) stehen noch 7 Bit, also 128 Stufen für die Wertebytes zur Verfügung. Um eine bessere Parameterauflösung zu erzielen, werden zwei aufeinanderfolgende Werte ausgewertet (MSB / LSB oder auch Most- und Least Significant Byte), was einen Wertebereich von 0 bis 16383 (14 Bit) ermöglicht.

Es gibt folgende drei grundlegende Arten von MIDI-Nachrichten. „Channel Voice Messages“ (Note Off, Note On, Polyphon Pressure, Control Change, Channel Pressure, Program Change und Pitch Bend), „System Common Messages“ (System Exclusive, MIDI Time Code, Song Position Pointer, Song Select und Tune Request) und „System Realtime Messages“ (Timing Clock, Start, Stop, Continue, Active Sensing und System Reset). Letztere, sowie MIDI Sample Dump sind für unser Projekt weniger relevant und wurden nicht implementiert.

System Exclusive Daten (Sysex) stellen einen Sonderfall da, da diese jeder Hersteller selber festlegen kann. Die Länge ist Variabel, das Status-Byte ist 0xF0 (hexadezimal), gefolgt von der Hersteller-ID, Modell-ID, Device-ID, den variablen Daten und dem End of Sysex 0xF7. In unserer Lösung ist ein Senden von Sysex vorgesehen.

2 Zusammenwirken von MIDI-Controller und Bediensoftware

2.1 Aufbau / Setup

Unseres Projekt-Aufbau sieht wie in Abbildung 2.2 ersichtlich, folgendermaßen aus. Zum einen benötigen wir die MIDI-Box, welche in dem anderen Medienprojekt zeitgleich geplant und gefertigt wurde. Auf der anderen Seite ist einen Computer vorausgesetzt, welcher ein beliebiges Betriebssystem haben kann. Es muss lediglich das Java Runtime Environment (download bei www.java.sun.com) der Version 1.5 oder aufwärts installiert sein, da dies die Java Virtual Machine und das API bereitstellt, und somit die Lauffähigkeit ermöglicht. Dieser Rechner ist mit einem MIDI-Interface ausgestattet, welches in unserem Fall über USB an den PC angeschlossen ist. Es handelt sich um eine bidirektionale Verbindung mit dem MIDI-Controller, um einen Datenaustausch zu ermöglichen.

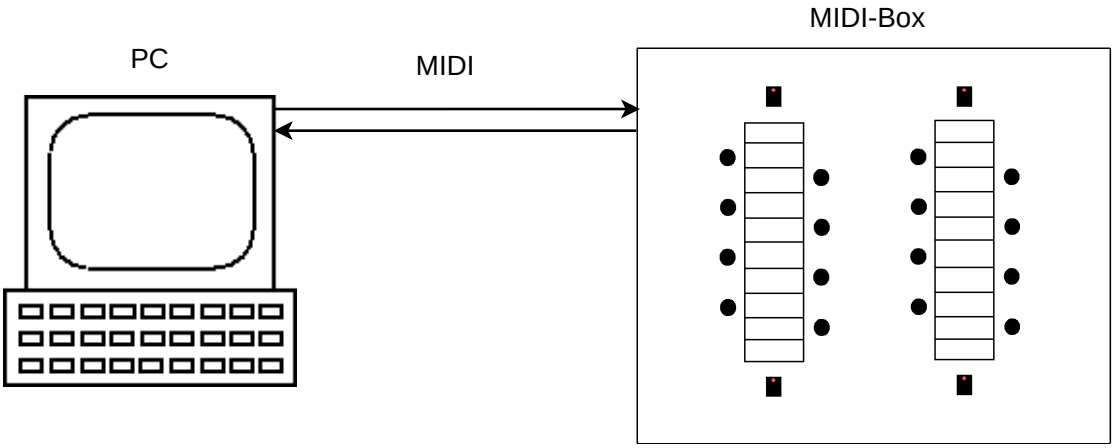

Abbildung 2.2: Hardware-Setup

Die Verbindung zum PC kann nach erfolgreicher Konfiguration unterbrochen werden. Dadurch kann der als „MIDIBox GLCD“ bezeichnete Controller ohne den Computer, also stand-alone, arbeiten. Hierfür müssen mindestens ein MIDI-Keyboard oder ein Sequenzer / Sequenzer Programm und ein Synthesizer angeschlossen werden. Dies geschieht folgendermaßen: Der Synthesizer wird wie auf Abbildung 2.3 ebenfalls bidirektional an den MIDI IN und -OUT mit dem MIDI-Controller verbunden, da bei bestimmten Funktionen Rückfragen nötig sind. Das Keyboard / Sequenzer wird benötigt um die Noten des Klangerzeugers auszulösen. Wenn das Keyboard an den zweiten MIDI-IN des Controllers angeschlossen wird, wird die MIDIBox GLCD in ein bestehendes Studio-Setup integriert. Daraufhin leitet ein integrierter MIDI-Merger die Nachrichten an den Ausgang weiter. Man definiert nun noch den „hörenden“ Klangerzeuger in seinem Setup, indem man einen von 16 MIDI-Kanälen auswählt, auf welchen das Soundmodul eingestellt ist. So kann man die bis zu 16 Synthesizer in einem Studio editieren und gleichzeitig den geänderten Sound

auf Probe spielen. Falls man im Besitz einer Sysex-fähigen MIDI-Kreuzschiene ist, kann für jedes Gerät eine spezifische Nachricht generiert werden, welche vor dem Editieren des Synthesizers an die Kreuzschiene gesendet wird und somit die Steuerung des Geräts ermöglicht.

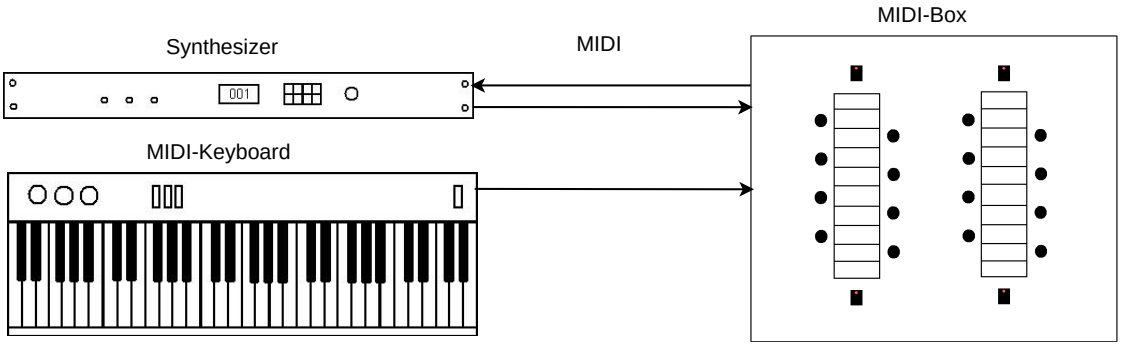

Abbildung 2.3: stand-alone-Betrieb

2.2 Funktionsumfang des MIDI-Controllers

Der hardwareseitige Aufbau der Bedienoberfläche sieht folgendermaßen aus. Der Controller besteht aus sechzehn Drehreglern mit Druckfunktion, vier beleuchteten Tastern und zwei jeweils mittig angeordneten 240x64 Pixel Displays. Diese sind in Abbildung 2.4 schematisch gezeichnet. Softwareseitig können darauf Menüs, Para-

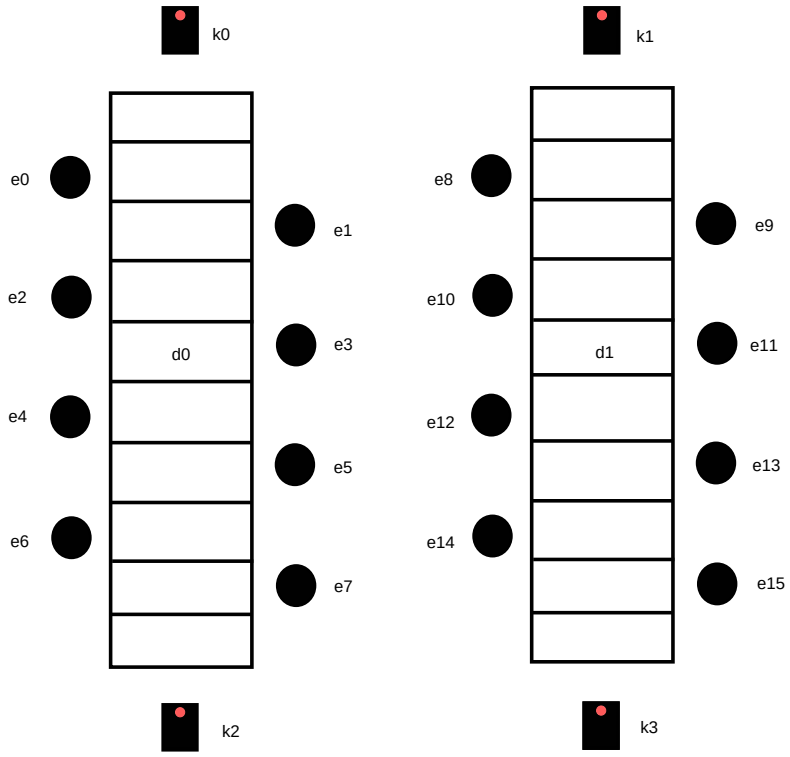

Abbildung 2.4: Arrangement MIDI Controller

meternamen und Parameterwerte, also eine variable Bedienoberfläche, dargestellt werden. Diese Nutzeroberfläche ist, wie in Abbildung 2.5 ersichtlich, in drei Bedienebenen aufgebaut. Die Device-Ebene, die Modul-Ebene mit vier Bänken und die Parameter-Ebene. Eine Navigation durch diese Ebenen ist mittels Druck auf die sechzehn Encoder möglich, welche sich neben der Beschreibung des Menüs befinden, in welches man gelangen möchte.

In der Device-Ebene kann man eines der sechzehn Geräte auswählen, um es zu editieren. Darüber hinaus können in der Java-Software synthesizerspezifische Sysex-Befehle eingegeben werden, welche den Editbuffer eines Klangerzeugers auslesen können, und somit überhaupt eine Einstellung des Geräts erlauben. Dies geschieht automatisch, wenn man ein Device anwählt. Es wird danach in die Ebene der Module eingetaucht.

In der Modulebene wird eine bestimmte Gruppierung von Parametern ausgewählt, beispielsweise das Modul Filter, oder Oszillator. Dies geschieht über vier Bänke wo man jeweils sechzehn Funktionsgruppen auswählen kann. Eine Sysex-Eingabe ist hier software-seitig ebenfalls vorgesehen, da es Geräte mit mehreren Editbuffers gibt.

Hat man sich nun für ein Modul entschieden, so gelangt man in dessen Parameteransicht. In dieser Ebene können sechzehn Parameter eingestellt werden. Diese werden über die Drehregler, bzw. die Tasterfunktion der MIDI-Box an den Synthesizer gesendet und die Klangeinstellungen werden in diesem Gerät modifiziert.

Die umgekehrte Navigation durch das Menü erfolgt nur über die Bank-Taster. Um von einer beliebigen Ebene zurück zur Device Ebene zu gelangen, drückt man die oberen Bank-Taster gleichzeitig.

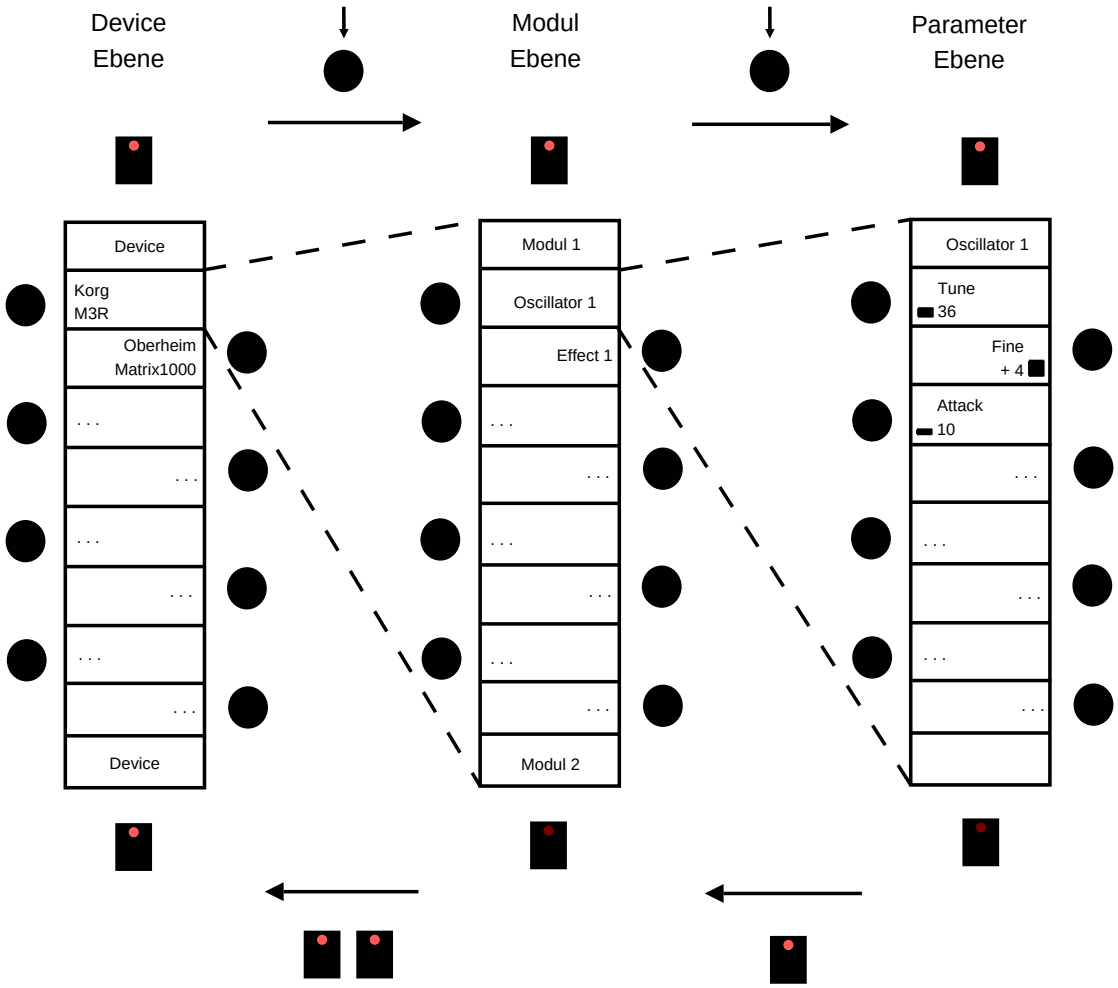

Abbildung 2.5: Menünavigation anhand eines Displays

2.3 Funktionsumfang der Bediensoftware

2.3.1 Anforderungen

- Die Bediensoftware soll die Verwaltung und Editierung von Konfigurationen für verschiedene Synthesizer ermöglichen. Für jeden Synthesizer stehen vier Bänke mit jeweils sechzehn Modulen und sechzehn Parametern pro Modul zur Verfügung.

- Die Vielzahl der Einstellmöglichkeiten soll über eine übersichtliche grafische Bedienoberfläche verwaltet werden.

- Die Persistenz, also das Abspeichern auf nichtflüchtigen Datenträgern, muss gegeben sein. Der Benutzer soll seine editierten Devices exportieren und importieren können. Das Speicherformat ist das Standard-Format XML. So ist es auch möglich erstellte Konfigurationen mit anderen Nutzern auszutauschen. Da es sich bei dem Projekt um ein Opensource-Projekt handelt, welches auf www.ucapps.de veröffentlicht wird, kann die Gemeinschaft auch an dem Format weiterentwickeln und Verbesseungen vornehmen.

- Es soll möglich sein, der Midibox eine Sysex-Machricht zu übermitteln, die diese an einen Synthesizer überträgt, um Klangeinstellungen anzufordern. Mit solch einer „Request Message“ kann der Editbuffer des Synthesizers ausgelesen werden.

- Die Übertragung der Devices an den Hardwarecontroller über Sysex-Nachrichten. Es soll ebenfalls die Möglichkeit bestehen einzelne Devices in einen bestimmten Slot auf dem Controller zu platzieren.

2.3.2 Bedienung

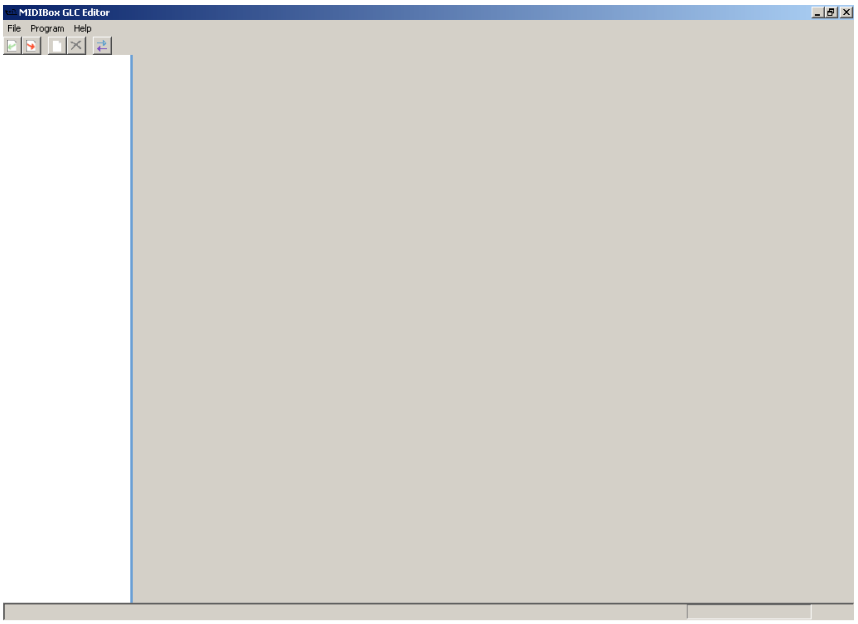

Abbildung 2.6: MidiboxEditor

Unsere Software ist, wie in Abbildung 2.6 zu erkennen ist, in drei logische Teile unterteilt. Ganz oben befindet sich die Menüleiste, welche die grundsätzlichen Programmfunktionen, wie exportieren und importieren der Daten, Auswahl der Sprache und des MIDI-Interface und das Übertragen der Daten zugänglich macht. Darunter an der linken Seite ist die Baumansicht zu finden, welche die Navigation im Programm übernimmt. An der rechten Seite befindet sich die Editieransicht. Sie bietet die Möglichkeit zur Eingabe von Namen und Hexadezimalzahlen sowie die Auswahl von Werten aus Drop-Down Menüs für die Konfiguration.

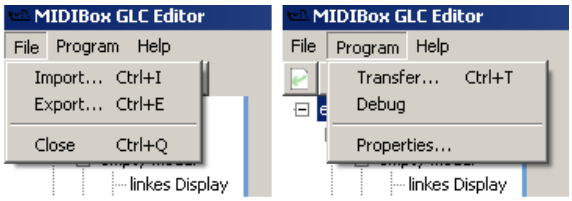

Abbildung 2.7: EditorMenuebar

- Menüleiste Die Menüleiste, wie in Screenshot 2.7 zu erkennen ist, bildet den ersten interaktiven Bereich unseres Programms. Das Drop-down-Menü „File“ hat die Funktionen „Import“ von Devices (Strg+I), „Export“ von Devices (Strg+E) und „Close“(Strg+Q). „Export“ ist zuständig für Speicherung von gerade erstellten Devices als *.mhp-Datei. „Import“ läd vorher erstellte Dateien des Typs *.mhp, welche alle definierten Informationen zu den Devices wiederherstellt. „Close“ beendet das Programm. Das zweite Menü „File“ hat

die Funktionen „Transfer“, „Properties“ und „Debug“. „Transfer“ bewirkt sofortiges Senden der Daten als Sysex-Nachricht. In „Properties“ gibt es zwei Auswahl Möglichkeiten, zum einen „Misc...“, wo man zwischen der Sprache Deutsch und Englisch auswählen kann und zum anderen „MIDI-Options“. Hier ist eine Auswahl des MIDI-Interfaces möglich, um die Kommunikation mit der Hardware zu gewährleisten. Über „Help“ kann die Hilfe-Datei aufgerufen werden.

Abbildung 2.8: Toolbar

Unter der Menüleiste befindet sich die Toolbar (Screenshot 2.8). Sie dient der schnellen Steuerung von häufig gebrauchten Funktionen, wie „Import“, „Export“, „Neues Device“, „Device Löschen“ und „Transfer“. „Neues Device“ erzeugt in der Baumansicht (linkes großes Feld, siehe nächsten Punkt) ein neues Device, welches durch Klicken auf das „+“ bearbeitet werden kann. „Device Löschen“ löscht das selektierte Device. „Transfer“ steht für die Übertragung eines auszuwählenden Devices auf den Controller. Aktiviert man Transfer kann man über den Transfer Dialog den Slot bestimmen in den das Device übertragen wird. An dieser Stelle kann dieses dann auf der Hardware aufgerufen werden.

- Baum-Ansicht

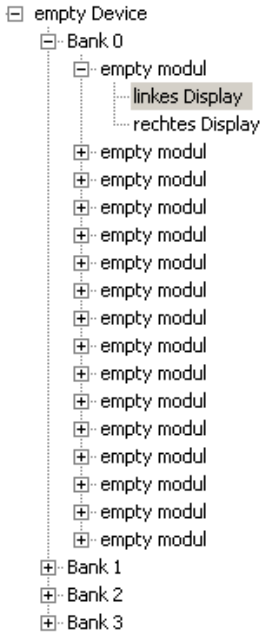

Abbildung 2.9: Navigation über Baum im Editor

Da eine Aufteilung in logische Module für die Benutzerführung sinnvoll ist, haben wir einen Klangerzeuger als Device deklariert, welchem Bänke mit Mo-

dulen untergeordnet sind, und welche ebenfalls wieder eine Untergruppe besitzen, die Parameter. Nach verschiedenen Oberflächendesigns auf Papier sind wir zum Entschluss gekommen, dass für diese Menge der 16384 Werte, also 16 Presets * 4 Bänke * 16 Module * 16 Parameter am besten eine hierarchische Struktur geeignet ist. Der Baum (Screenshot 2.9) ist in vier Ebenen unterteilbar. Die erste Ebene ist das Device, die zweite Ebene die Bänke, die dritte Ebene sind die Module, und die vierte Ebene sind die Parameter, welche in linkes und rechtes Display unterteilt sind. Diese Baumstruktur hat den Vorteil, dass man bereits durch das Betriebssystem oder andere Programme daran gewöhnt ist mit solch einer Struktur zu arbeiten. Desweiteren kann der User sich seine Komplexität selber einstellen, indem er Verzweigungen bewusst auf- oder zuklappt. Dies ist besonders wichtig, falls man beispielsweise schnell Einstellungen in anderen Synthesizern oder in anderen Modulen vergleichen möchte. Eine effiziente Navigation ist ebenfalls über diese Struktur möglich, da man mit vier Klicks direkt in der Parameter Ebene seines gewünschten Devices ist und dort sofort mit der Editierung beginnen kann. Die Unterteilung in linkes und rechtes Display stellt hier eine gewisse Notlösung dar, da ein Display einerseits schon eine abgeschlossene Einheit bildet. Andererseits aber ist die Bedienung hardwareseitig auf zwei Displays ausgelegt. Wir mussten uns dieses Kunstgriffes bedienen, da wir feststellten, dass die Kommentare der jeweiligen Eingabefelder mehr Platz in Anspruch nahmen als geplant. Die übersichtliche Anordnung der einzelnen Parameter mit eindeutigen Labels, ging in diesem Fall einer Gesamtübersicht über beide Display vor. Dieser Lösungsansatz ist noch immer konzeptkonform, da man als Modul ein Display hat und auf zwei Displays unterschiedliche Inhalte darstellen kann. Falls dies nicht der Fall sein sollte, so kann man einfach zwischen den beiden Ansichten in der Baumansicht wechseln.

Innerhalb des Baums ist es ebenfalls möglich, Devices zu erstellen und zu löschen. Dies geschieht mit einem Falschklick in der Baum-Ebene und der Auswahl im Dialog „Create New Device“. Der Nutzer kann nach wie vor alternativ über die Buttons Devices erstellen und löschen. Wichtig ist in dem Fall, dem User mehrere, für ihn gut nutzbare Wege bereitzustellen, zwischen welchen er wählen kann. In diesem Baum wird wie folgt navigiert, wenn ein Device erstellt wurde. Klickt man auf das „+“ oder doppelt auf den Namen des Devices, so betritt man die nächst tiefere Bank-Ebene. Diese 4 Speicherbänke stehen nun zur Verfügung, wenn sie mehr als 16 Module mit jeweils 16 Parametern nutzen möchten. In die Modul-Ebene gelangt man durch Klicken auf + oder Doppelklick auf die Bänke 1 bis 4. Durch das Klicken auf + oder Doppelklick auf den Modulnamen öffnet sich nun die letzte Ebene, es werden 16 Parameter sichtbar, welche wiederum durch Klick / Doppelklick in linkes und rechtes Dis-

play aufgefächert werden können. Synchron zur aktiven Ebene ändert sich die Editieransicht dynamisch mit und wechselt zwischen 4 verschiedenen Seiten mit unterschiedlichen Eingabemöglichkeiten. Hier können Namen eingegeben, und parameterspezifische Einstellungen getroffen werden.

- Editieransicht

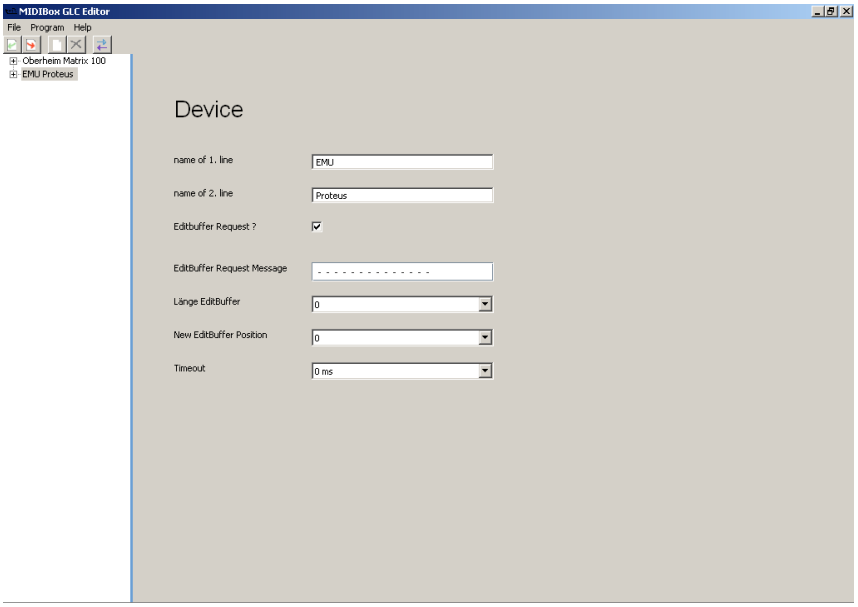

Abbildung 2.10: Devicelevel

Wie in Screenshot 2.10 ersichtlich, sind bei dem gewählten Device 2 Eingabefenster für jeweils 10 Zeichen vorhanden. Doppelklick selektiert den Eintrag und man kann einen Namen eingeben. Dieser Platz kann idealerweise mit dem Namen des Synthesizer Herstellers in der ersten Zeile und dem Modell in der zweiten Zeile ausgefüllt werden, um die Navigation auf der Hardware zu erleichtern. Weiterhin befindet sich eine Checkbox zur Anfrage des Editbuffers auf dieser Seite. Setzt man diese aktiv, wird ein Texteingabefeld und drei Drop-Down-Menüs sichtbar. Im ersten Feld kann man die „Request-Nachricht“ eingeben, um den Editbuffer des Synthesizers anzufordern. Dazu sind noch zwei weitere Eingaben erforderlich. Zum einen die Länge des Editbuffers, welche zur Verifizierung der gesendeten Daten benötigt wird. Zum anderen „Begin of Edit-Buffer“, also die Position des Editbuffers innerhalb der Sysex-Nachricht des angefragten Synthesizers. Dies ist notwendig, da der Synthesizer vor dem Editbuffer einen Header schreibt. Der Parameter Timeout gibt an, wie viele Sekunden der Controller auf einen angefragten Editbuffer wartet. Es kann die Zeitdauer in Sekunden-Intervallen von 0 bis 15 s eingestellt werden.

In der Bank-Ebene ist keine Namens-Eingabe ausführbar, da die Bänke auf dem Controller nicht benannt werden konnten. Eine Identifikation der Bänke ist aber durch die auf der Frontplatte befindlichen, beleuchteten Taster möglich,

welche dort für die Auswahl sorgen. Dies hat den Grund, dass die Bänke als Erweiterung der Module gesehen werden und man wahrscheinlich in der normalen Anwendung in nur einer Bank arbeiten wird.

In der Modul-Ebene sind die gleichen Einstellungen wie bei der Device-Ebene möglich.

In der Parameter-Ebene (Screenshot 2.11) kann man alle synthesizerspezifischen Einstellungen treffen, welche im nächsten Punkt ausführlich behandelt werden.

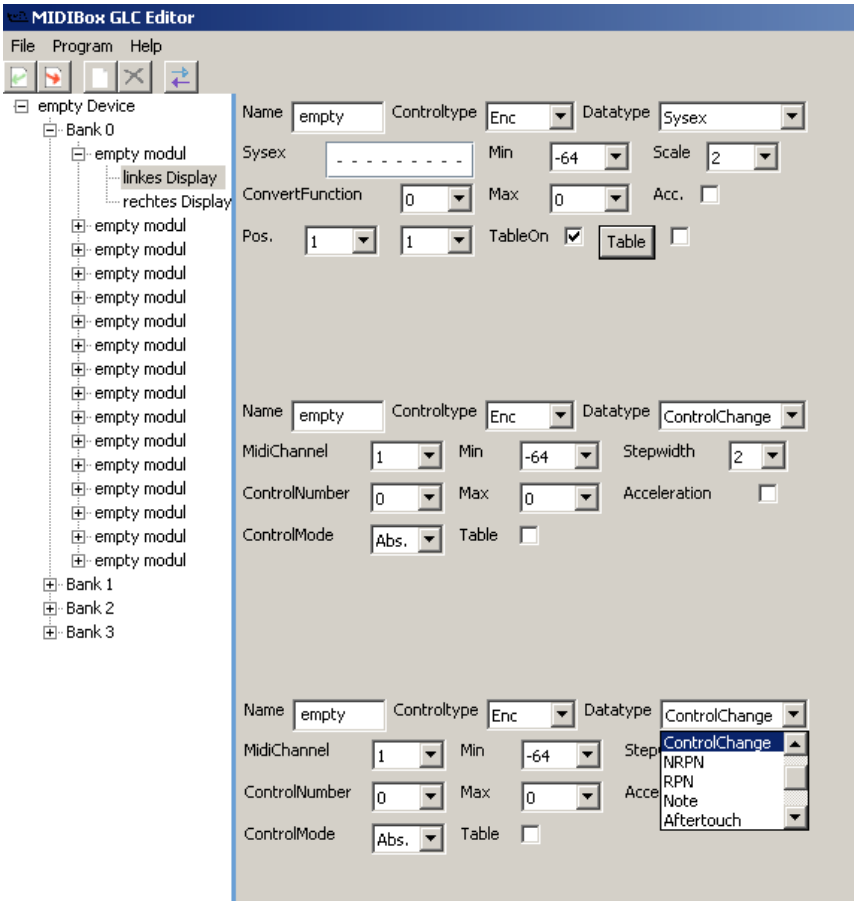

Abbildung 2.11: Parameterlevel

2.3.3 Zu verwaltetende Parameter

Neben den schon beschriebenen Eingabefeldern der Device- und Modul-Ebene gibt es noch weitere Eingabemöglichkeiten, welche sich, in der Parameter-Ebene befinden. In den beiden Parametertabellen sieht man welche Eingabemöglichkeiten pro Mididatentyp möglich sind.

- „Name“ ist ein Eingabefeld für den Namen des Parameters.
- „Controltype“ ist ein Drop-down-menü zur Wahl zwischen Drehregler und Drucktaster.
- Bei „Modi“ hat man die Wahl zwischen dem Nachrichtenmodus Off, Program Change, ControlChange, NRPN, RPN, Note, Aftertouch, Pitch Bend, Sysex, und Editbuffer. „Off“ weist dem Regler keine Funktion zu und ist implementiert worden um auf der Box „Dummy-Schalter“ einstellen zu können, die keine Funktion haben. „Editbuffer“ist ein spezieller Modus der für den Fall konzipiert ist, in dem ein Synthesizer die Parameteränderung nur über das Versenden des Editbuffers zulässt. Hier werden der Midibox Daten übergeben, die beschreiben an welcher Stelle im Editbuffer Werte geändert werden sollen. Controlchange ist die Standardeinstellung, da dies meistens der am häufigsten verwendete Modus ist.
- „Midi Channel“ist bei allen Datentypen einstellbar und wählt einen Midikanal zwischen 1 und 16.
- „MSB“ und „LSB“ ist nur bei dem Mididatentyp Programchange einstellbar. Es kann ein Wertebereich von 0 bis 127 gewählt werden. Dieser definiert einen 14 Bit großen Wert mit dem dann maximal 16384 Programme angewählt werden.
- „Acceleration“ beschreibt die Beschleunigung der Drehbewegung. Es werden je nach Drehgeschwindigkeit mehrere Werte übersprungen. Dies ist besonders sinnvoll für 14 Bit-Werte, um schnell in die gewünschte Größenordnung zu gelangen und sich dann schrittweise zu nähern.
- „Controller Number“ ist nur bei dem Mididatentyp Controlchange einstellbar. Es können Werte von 0 bis 127 gewählt werden. Dieser Wert spricht einen konkreten Effekt oder Klangparameter in einem Synthesizer an.
- Bei „Control Mode“ mit den Möglichkeiten Abs, Abs(14), Rel1/2/3, Rel1/2/3(14), Inc/Dec kann ausgewählt werden in welcher Art die Controllerdaten erzeugt werden. Es können absolute und relative Controllerdaten in jeweils 7 Bit und 14 Bit eingestellt werden. 14 Bit kann verwendet werden, falls eine höhere

ENC / SWITCH	MIDI DATA TYPE	MIDI CHANNEL	PARA. NAME	VAL. 1
ENCODER	**Off**			
SWITCH				
ENCODER	**PROGRAM CHANGE**	1 – 16	Off, Bank Sel MSB	Off, Bank Sel LSB
ENCODER	**CONTROLCHANGE**	1 – 16	CC 0-127	Min.Val.: -64 - 127 -8192 - 16383
SWITCH		1 – 16	CC 0-127	On Val.: -64 - 127 -8192 - 16383
ENCODER	**NRPN/RPN**	1 – 16	NRPN-Nr.	Min.Val.: -64 - 127 -8192 - 16383
SWITCH		1 – 16	NRPN-Nr.	On Val.: -64 - 127 -8192 - 16383
SWITCH		1 – 16	Note Nr.	On Val.: Velocity fixed
ENCODER	**AFTERTOUCH**	1 – 16	Key Nr. 0 - 127, All (All=Chan.Aftertouch)	Min.Val.: -64 - 127
SWITCH		1 – 16	Key Nr. 0 - 127, All (All=Chan.Aftertouch)	On Val.: -64 - 127
ENCODER	**PITCHBEND**	1 – 16	Range	Range: 0 - 127 -8192 - 16383
ENCODER	**SYSEX**	1 – 16	Position 1 + 2	Position 1 , 2: 1 - 15
SWITCH		1 – 16	Position 1 + 2	Position 1 , 2: 1 - 15
ENCODER	**EDITBUFFER**	1 – 16	Position	Position: 1 - 15
SWITCH		1 – 16	Position	Position: 1 - 15

Abbildung 2.12: Parametertabelle Teil

ENC / SWITCH	VAL. 2	CTL. MODE	NAME	SCALING	ACCEL.	CONV.	TABLE	HEADER
ENCODER								
SWITCH								
ENCODER	Off, 0 - 127	Abs	X					
ENCODER	Max.Val.: 0 - 127 0 - 16383	Abs, Abs (14) Rel1, Rel2, Rel3 Rel1, Rel2, Rel3 (14) INC / DEC	X	0, 1, 2, 4, 8, 16, 32, 64	X		X	
SWITCH	Off Val.: 0 - 127 0 - 16383	Abs, Abs (14) Rel1, Rel2, Rel3 Rel1, Rel2, Rel3 (14) Toggle On / Off INC / DEC	X					
ENCODER	Max.Val.: 0 - 127 0 - 16383	Abs, Abs (14) Rel1, Rel2, Rel3 Rel1, Rel2, Rel3 (14) INC / DEC	X	0, 1, 2, 4, 8, 16, 32, 64	X			
SWITCH	Off Val.: 0 - 127 0 - 16383	Abs, Abs (14) Rel1, Rel2, Rel3 Rel1, Rel2, Rel3 (14) Toggle On / Off INC / DEC	X					
SWITCH		Abs	X					
ENCODER	Max.Val.: 0 - 127	Abs	X					
SWITCH	Off Val.: 0 - 127	Abs	X					
ENCODER		Abs, Abs (14)	X	0, 1, 2, 4, 8, 16, 32, 64	X			
ENCODER	Min: / Max: -64 - 127 / 0 - 127	Abs, Abs (14)	X	0, 1, 2, 4, 8, 16, 32, 64	X	0 - 255	X	15 Byte (Hex)
SWITCH	Min: / Max: -64 - 127 / 0 - 127	Toggle On / Off	X			0 - 255	X	15 Byte (Hex)
ENCODER	Min: / Max: -64 - 127 / 0 - 127	Abs	X	0, 1, 2, 4, 8, 16, 32, 64	X	0 - 255	X	
SWITCH	Min: / Max: -64 - 127 / 0 - 127	Abs	X			0 - 255	X	

Abbildung 2.13: Parametertabelle Tei2

Auflösung benötigt wird. „Abs“ gibt absolute Datenwerte aus. Dabei können Sprünge bei Werte Änderung auftreten. „Rel“ ändert den Wert unabhängig von der Position des Reglers. . „Inc/Dec“ dient der schrittweisen Erhöhung bzw. Verminderung des Wertes. Bei Pitchbend steht nur ControllerMode 7 Bit oder 14 Bit zu Verfügung.

- „Min“ und „Max“ erlaubt die Festlegung des Wertebereich eines Regler. Der Wertebereich kann maximal 127 betragen und auch negative Werte für die untere Grenze enthalten. „Scale“legt dann fest wie der Wertebereich an den Synthesizer übertragen wird.

- Bei „Scale“ ist folgende Auswahl möglich 0, 1, 2, 4, 8, 16, 32, 64. Damit kann die Skalierung berechnet werden. Zur Berechnung wurden zwei MIDI-Skalen eingeführt (siehe dazu Abbildung 2.14). Zum einen gibt es die MIDI-Skala, welche über den MIDI-Ausgang in den Grenzen von 0 bis 127 bzw. 0 bis 16383 gesendet wird. Zum anderen wurde von uns die Userskala festgelegt, welche die Display-Ausgabe repräsentiert und bei „Min“ und „Max“ eingestellt wird. Da man über die MIDI Schnittstelle keine negativen Werte senden kann, muss zwischen diesen Skalen mit gewissen Vorgaben umgerechnet werden. Bei den verschiedenen „Scale“ Einstellungen wird die Umrechnung zwischen MIDI-Skala und der User Skala auf verschiedene Arten vorgenommen. Folgendes gilt für alle Modi: Wenn das User-Skalen Minimum Null oder positiv ist, so ist die User Null gleich der MIDI-Skalen Null. Ist das User Minimum negativ, entspricht die User-Skalen Null dem Wert 64 bei 7 Bit und 8192 bei 14 Bit. Dieser Wert erhöht oder verringert sich um die positiven und negativen Werte.

  Nun zu den verschiedenen Modi.

  Bei „Scale“ 0 wird der gesamte MIDI Bereich auf die User Skala skaliert. Das bedeutet, dass beim Drehen eines Encoders kontinuierlich alle Werte der MIDI Skala gesendet werden, aber die Anzeige nur an bestimmten Stellen auf den nächsten User-Wert umspringt.

  „Scale“1: Es gibt genau so viele MIDI-Werte wie User Werte. Beim Drehen eines Encoders entspricht die Werte Anderung der User Skala der der MIDI Skala.

  Stepwidth 2-64: Die User Skala verändert sich mit jedem Drehschritt um + -1, die MIDI-Skala verändert sich mit jedem Drehschritt um den in Stepwidth eingestelleten Wert. Dies ist sinvoll, um Wertebereiche in der MIDI Skala zu überspringen, und weniger User- Werte anzusteuern.

  „Scale“ 0 : Es wird eine „Auffächerung“ der eventuell kleineren User Skala auf die MIDI-Skala vorgenommen, man kann auch von einer Projektion sprechen. In allen anderen Modi beschreibt der Wert in „Scale“ die Schrittweite in der MIDI Skala pro User-Wertesprung.

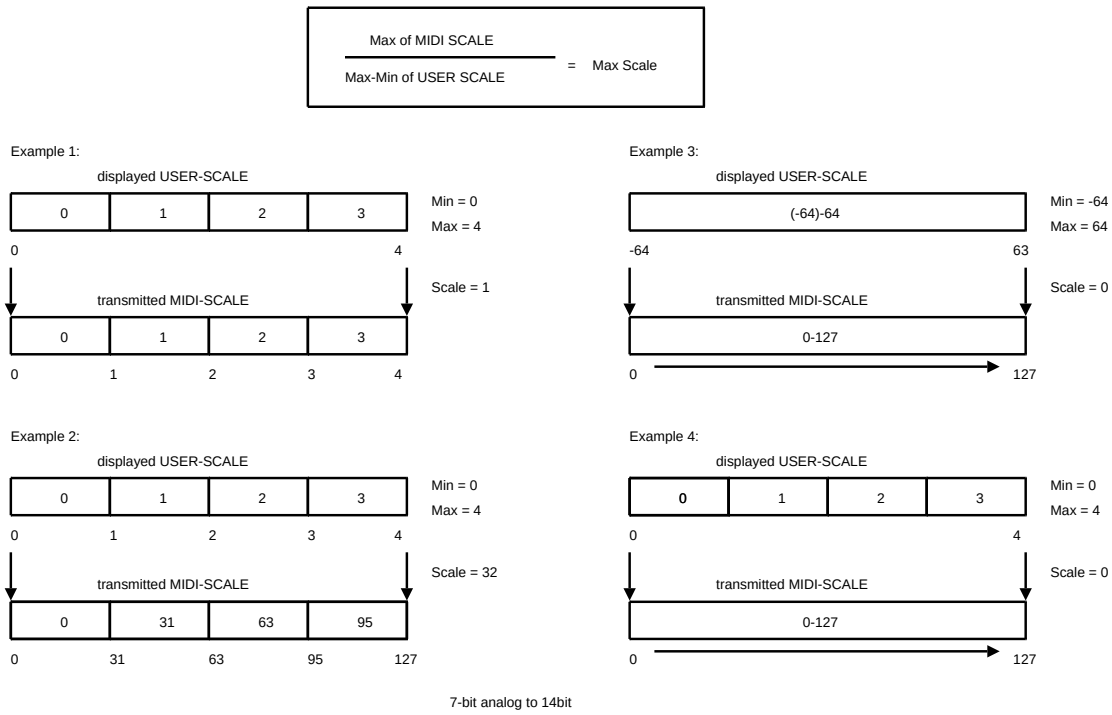

Abbildung 2.14: Zusammenhang zwischen MIDI- und User-Skala

- Die Checkbox „Table“ ermöglicht es, über den dann angezeigten Button ein Popup-Fenster aufzurufen. In diesem kann jedem Schritt des Reglers ein Name zugewiesen werden. Ein Beispiel hierfür ist eine Unterteilung der 128 Werte in 4 Stufen. Dies lohnt sich z. B. in der Oszillator-Sektion eines Synthesizers (vergleiche Screenshot 2.15). Hier können dann die Wellenformen mit ihrem Namen (z.B. Rechteck, Dreieck, Sägezahn, Sinus) angezeigt werden.

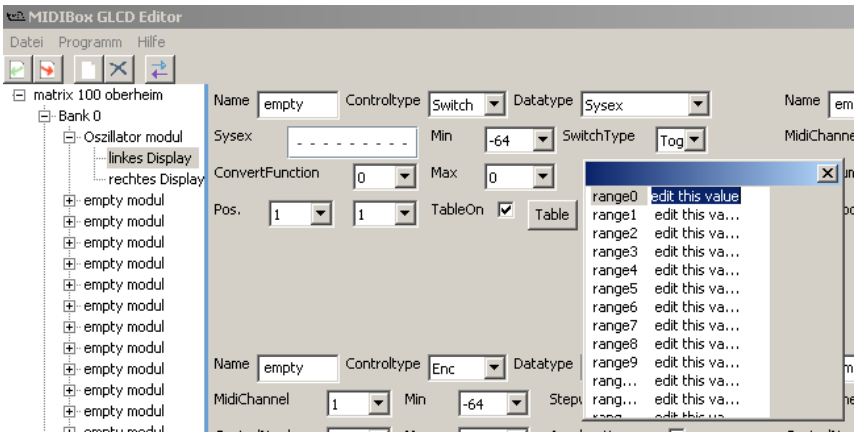

Abbildung 2.15: Checkbox Table mit Popup

- „NRPN“ oder „RPN“ Nummmern besitzen keine genormte Zuordnung. Der Hersteller kann diese, ähnlich dem Sysex, definieren. Hier stehen durch Verwendung von MSB und LSB 14 Bit zur quasi sprungfreien Parameteränderung zur Verfügung.

- „NoteNumber“ ist einstellbar von 0 bis 127 und kann nur bei Mididatatype Note gewählt werden. Der Wert beschreibt welche Note der Midi-Tastatur gewählt wird.

- „Velocity“ ist ebenfalls nur bei Note wählbar. Hier kann die Anschlagsstärke von 0 bis 127 eingestellt werden. Dieser Modus wird wahrscheinlich eher zum „Triggern“ von Sequenzen/Sounds oder Stummschalten von Spuren Verwendung finden, als zum Spielen von MIDI-Noten. Eine Ansteuerung von LEDs ist ebenfalls denkbar.

- „Switchtype“steht zu Verfügung, falls die Switchfunktion aktiviert wurde. Hier wird definiert, wie der Knopf fungieren soll. Abhängig vom „ControlMode“ sind folgende Einstellungen möglich. Standardmäßig kann zwischen „ToggleOn“ oder „Toggle Off“ gewählt werden. „ToggleOn“gleicht einer Schalterfunktion. Bei jedem Druck wird abwechselnd der On und Off Wert gesendet. „ToggleOff“ entspricht einer Tasterfunktion. Der On Wert wird solange gesendet wie der Knopf gedrückt bleibt. Beim Loslassen wird der Off Wert gesendet. Falls ein relativer Controlmode gewählt wird kann zwischen „Inc“ und „Dec“ entschieden werden. Hierbei fungiert der Knopf als Taster und erhöht oder vermindert den Wert um den eingestellten Wertebereich.

- „Key Nr.“ erlaubt eine Werte Einstellung von 0 bis 127 und die Einstellung „All“. Diese Einstellung ist nur für „Aftertouch“ vorgesehen. Diese Änderung kann für eine Taste (0-127) angewendet werden, man spricht dabei von Key Pressure. Oder man wendet es auf alle Tasten (All), bzw. den ganzen Kanal an, was dann Channel Pressure heißt.

- „Range“ reicht von 0 bis 127 und definiert die Grenzen bei Pitchbend. Wird der Wert durch zwei dividiert ergibt er die positive und negative Grenze.

- „Sysex“: In diesem Eingabefeld [15 byte] wird die Nachricht eingegeben, die es zu versenden gilt. Diese wird immer im Format von Hexadezimalzahlen eingegeben. Es wird sowohl das Sysex Statusbit und der Header als auch die variablen Daten eingegeben. An der Stelle der Variablen muss man 00 setzen. Desweiteren muss die Position der Variablen (in Byte) mit Position var. Data [1-14] angegeben werden.

- „ConvertFunction“ stellt die Auswahl der Konvertierungs Funktion bereit. Es sind 255 Funktionen auswählbar. In der Midibox sind bereits 3 Konvertierungs Funktion implementiert. 0 stellt keine zu Verfügung, 1 ist SingleByteBigEndian, 2 ist DoubleByteBigEndian und 3 ist DoubleByteLittleEndian. Ist Editbuffer als Mididatentyp gewählt dann wird für einen Wert von 1 die Konvertierungsfunktion des Oberheim Matrix 1000 gewählt und für den Emu Proteus die 2.

- “Pos.“ bezeichnet ein Auswahlfeld für die Position der variablen Daten. Im „Editbuffer“ Modus kann an dieser Stelle über ein zweites Auswahlfeld die Position der Checksumme bestimmt werden, ist 0 gewählt wird keine Checksumme berechnet.

2.3.4 Anwendungsfälle (Usecases)

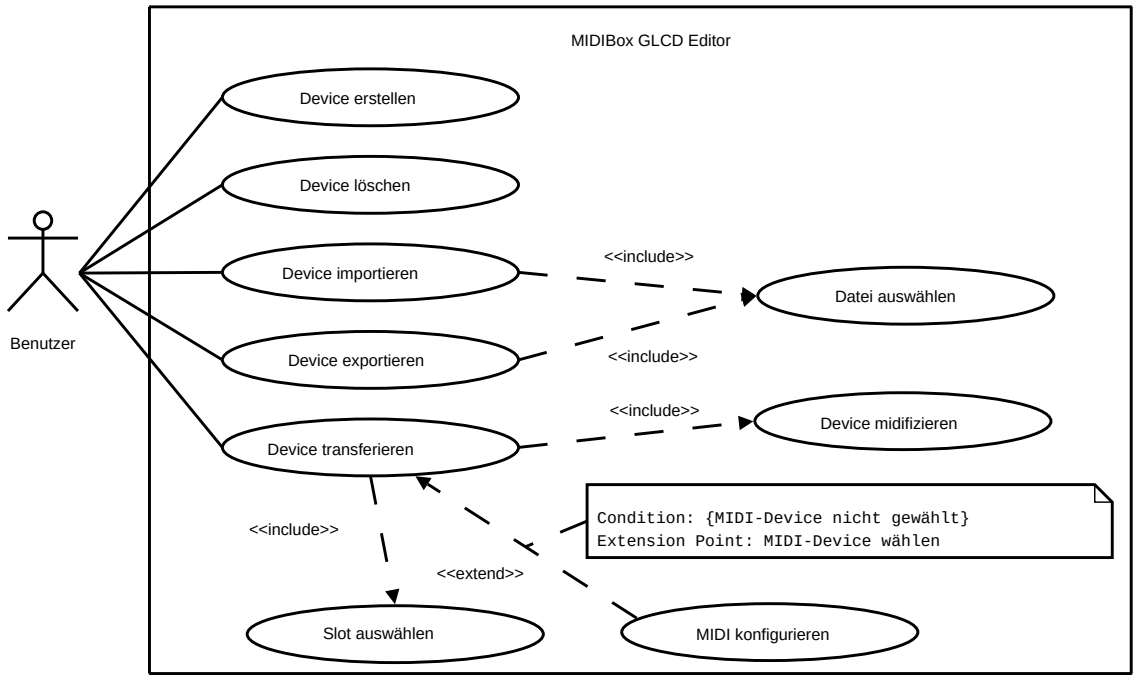

Abbildung 2.16: Editor Usecases

Wie in der Abbildung 2.16 zu sehen ist, befindet sich der Benutzer mit dem System „MIDIBox GLCD Editor“ in Interaktion. Bestimmte Handlungen werden hier als Usecase zusammengefasst. Im vorliegenden Beispiel wird vereinfacht erklärt, welche Möglichkeiten der Benutzer hat, mit den Devices zu arbeiten. Es gibt Usecases, wie z.B. „Device erstellen“ und „Device löschen“, welche nur mit dem Aktuer assoziiert sind. Ferner sind Usecases vorhanden, welche sowohl mit dem Akteur, als auch mit weiteren Usecases über eine gestrichelte Linie verbunden sind. Diese sind wiederum mit den Stereotypen „include“ und „extend“ bezeichnet. „Device importieren“ und „-exportieren“ beinhalten beispielsweise immer einen neuen Usecase, das „Datei-Auswählen“ über einen Dialog. Diese Usecases werden dem zu folge mit einem „include“ Stereotyp verknüpft. Genau so verhält es sich auch beim „Device transferieren“, wo vorher unbedingt eine „Midifizierung“, sprich Überführung der Usereingaben in die Übertragunsstruktur Sysex erfolgen muß. Ebenso muß ein Speicher-Slot für dieses Device auf dem MIDI-Controller gewählt werden („Slot auswählen“), um dieses zu speichern. Eine Besonderheit bildet beim Usecase „Transferieren“, die Erweiterung des extend-Stereotyps um den Usecase „MIDI konfigurieren“. Hier gibt es eine Bedingung „MIDI Device nicht gewählt“ am Erweiterungspunkt „MIDI Device wählen“, welche wahr sein muß, um zum neuen Usecase „MIDI konfigurieren“ zu gelangen. Dies klingt zunächst unlogisch, weil eine Auswahl des MIDI-Interface für eine Übertragung grundlegend ist. Da dieses MIDI-Interface aber eventuell schon zu einem früheren Zeitpunkt konfiguriert werden konnte, und man beispielsweise nun das zweite Device auf den Controller übertragen möchte, ist diese Darstellungsweise

notwendig. Ein erneutes Eingeben des Interface ist unnötig und somit muss der Usecase „MIDI konfigurieren“ nicht bereitgestellt werden, und stellt somit auch keine „include“ Assoziation dar. In den folgenden Beispielen sollen die erwähnten Usecases tiefgründiger beschrieben werden, um einen Einblick in die volle Funktionsweise des Programms zu geben.

- Beispiel A: Control Change

  Sie wollen beispielsweise im Sequential Cirquits „SixTrak“ den Parameter Filter Cutoff editieren. Dazu erzeugen sie zuerst ein neues Device mit Klick auf den „Neues Device“-Button. Dieser erscheint nun in der Baumansicht und kann mit Klick auf das + in der Editieransicht benannt werden. Sie geben nun mit Doppelklick in der ersten Zeile SEQUENTIAL und in der zweiten Zeile SIX-TRAK ein. Danach können sie sich eine Bank aussuchen, in welcher sie arbeiten wollen. Es empfiehlt sich Bank 0 zu wählen, da diese standardmäßig beim ersten Aufruf des Devices im Controller initialisiert ist . Klicken sie dazu auf das + neben Bank 0 und sie gelangen in die Modul-Ansicht dieser Bank. Wählen sie nun eines der sechzehn Modulen aus, um es mit Filter zu benennen. Durch Klicken auf das + neben Filter Modul kommen sie nun an die 16 Parameter des Modul Filter. Wählen sie nun das linke der beiden Displays und benennen sie einen der 8 Parameter mit Cutoff. Beachten Sie, dass ihnen hier wieder nur 10 Zeichen zur Verfügung stehen. Die Auswahl für Encoder oder Button kann auf Enc gesetzt bleiben, die Auswahl des Controller Mode kann ebenfalls auf ControlChange bleiben. Wählen sie nun ihren MIDI-Kanal des Geräts aus: Kanal 01 ist voreingestellt und sollte bei den meisten Synthesizern auch voreingestellt sein. Die Controller-Nummer für den Filter-Cutoff müssen sie sich nun aus dem Handbuch ihres Klangerzeugers in dieses Feld übertragen. In diesem Fall ist es Nr. 21, welche über den Scrollbar eingegeben wird. Der Control Mode kann auf Abs.(absolut) gelassen werden, da dieser Synthesizer keine relativen Daten verarbeiten kann. Wie im Handbuch ersichtlich, beträgt die Auflösung der Werte 7 Bit. Eine hohe Schrittweite ist bei einem Parameter, wie der Frequenz unsinnig, daher können wir die 1 auswählen. Die Grenzen werden nun zwischen Minimum 0 und Maximum 127 gesetzt, um über den vollen Wertebereich verfügen zu können. Die Checkbox Table unterhalb des Minimums wird benötigt, wenn sie sich im Display statt Werten lieber einen String anzeigen lassen wollen. Beispielsweise kann man im Six-Trak den Filter-Keyboard-Track in drei Stufen ändern: off/half und on, welche sie dann auf dem Display ausgegeben bekommen können. Klicken sie dazu die Checkbox an, der Button Table wird sichtbar, und sie können in die Tabelle nun diese drei Wörter eingeben. Für den Fall der Cutoff-Frequenz ist dies allerdings nicht gewünscht. Schließlich können sie nun auswählen, ob sich ihr Encoder konstant verhält, oder exponentiell beschleunigt, die Acceleration ist besonders sinnvoll

für 14 Bit-Werte.

- Beispiel B: System-Exclusive

  Erstellen wir nun eine Sysex-Nachricht, um den Filter-Cutoff im Synthesizer Oberheim „Matrix 1000“ zu ändern. Synchron zu Beispiel A wird ein neues Device mit dem Modul Filter und dem Parameter Cutoff-Frequenz erzeugt. Wählen sie wieder Enc aus, da sie damit Variablen verändern wollen und stellen sie Sysex im Control Mode-Dropdown ein. Sehen sie nun in das Matrix-Handbuch und schauen sie, wie die Nachricht „Remote Parameter Edit“ gebildet wird. Wir benötigen jetzt folgende Bytes im Hexadezimal-Format: F0-10-06-06-21-00-F7, wobei 21 für den Parameter VCF-Frequency, und die 00 (bitte immer 00 verwenden) für die Variablen Werte in 7 Bit stehen. Wählen sie nun die Konvertierungs Funktion Nummer 1, welche speziell für den Matrix 1000 implementiert wurde. Schauen sie bei weiteren Fragen bitte in die Dokumentation des Hardware Kontrollers. Die Position der variablen Daten wird nun eingegeben, um eine Änderung der Variablen an dieser Stelle zu ermöglichen. In unserem Fall ist es Byte 6 (00). Wir befinden uns bei diesem Parameter in 7 Bit, welche sonst niedriger als 7 Bit sind und einer Skalierung eventuell bedürfen. In diesem Fall ist aber der gesamte Wertebereich gewünscht und die Skalierung wird auf 0gesetzt, das Minimum ebenfalls auf 0 und das Maximum wird auf 127 gesetzt. Die Table-Checkbox muß nicht gesetzt werden, Acceleration ist auch nicht gewünscht.

- Beispiel C: Speichern

  Da sie sich jetzt ihren Synthesizer konfiguriert haben, können sie ihre Arbeit sichern, indem sie auf den Button Export klicken oder über Datei/Exportieren zum Dialog gelangen. Hier können sie sich einen Namen für die Datei und den Ort auf der Festplatte auswählen. Es wird nun eine *.mhp-Datei erzeugt, welche sie wieder importieren können. Vorsicht, es ist keine Überschreiben-Warnung vorhanden, sie können sich so eventuell ein erstelltes Device löschen!

- Beispiel D: Importieren

  Dies entspricht in seiner Behandlung dem Beispiel C, mit der Ausnahme, dass eine *.mhp-Datei geöffnet werden muß. Dies kann ebenfalls sinnvoll sein, wenn sie ein Gerüst für viele Synthesizer verwenden wollen, bei welchen sie nur die Parameter auf das jeweilige Gerät anpassen.

- Beispiel E: Gerät übertragen

  Übertragen wir nun ein erstelltes Device, welches sich in unserer Baumansicht befindet. Falschklicken sie nun dieses Device und wählen sie Gerät Übertragen

aus. Es öffnet sich ein Dialog, welcher die Position in einem Slot des Midicontrollers auswählen lässt. Klicken sie auf das gewünschte Feld auf der grafischen MIDI-Box und sie können das Device an dieser Stelle auf der Hardware aufrufen.

2.4 Protokoll / Speicheraufteilung

2.4.1 Protokoll

Zur Übertragung der Daten wird ein Protokoll verwendet, welches den Festspeicher des MIDI-Controllers beschreibt. Dazu müssen die zu übertragenden Daten in maximal 64 Byte große Blöcke aufgeteilt werden. Jeder dieser Blöcke wird mit einem Header (siehe Tabelle 2.1) versehen, der die Adresse wie in Abbildung 2.17 gezeigt, spezifiziert. Desweiteren enthält der Header Felder, um die Versionsnummer von Soft- und Hardware zu übermitteln. Damit die Hardwareplattform erkennt, dass es sich um eine an sie gerichtete Speicheranfrage handelt, werden Hersteller-Codes und eine Funktionsnummer im Header übertragen. Anschließend wird der Datenblock übertragen, dessen Ende durch das End of Exclusive (EOX oder 0xF7) gekennzeichnet ist.

Nach der Übertragung der Speicheranfrage sendet der Controller eine Antwort. Diese entspricht dem in der Tabelle 2.2 gezeigten Protokoll. Dieses ermöglicht der Software die Übertragung zu verifizieren und Benutzer über Erfolg und Misserfolg zu informieren.

Feld	Inhalt	Aufgabe
0	0xF0	SysEx Start
1	0x55	Hersellercode
2	0x4d	Hersellercode (erweitert 1)
3	0x42	Hersellercode (erweitert 2)
4	0x00-0x7F	Empfängercore
5	0x00-0x7F	Funktion
6	0x00-0x7F	Protokollversion MAJOR
7	0x00-0x7F	Protokollversion MINOR
8	0x00-0x7F	Speicheradresse1
9	0x00-0x7F	Speicheradresse2
10	0x00-0x7F	Speicheradresse3
11	n*0x00-0x7F	Daten
12	0xF7	SysEx Ende

Tabelle 2.1: Nachricht von der Java Software zur MIDIbox

Feld	Inhalt	Aufgabe
0	0xF0	SysEx Start
1	0x54	Hersellercode
2	0x00-0x7F	Antwortcode (siehe Tabelle 2.3)
3	0x00-0x7F	Checksum
4	0xF7	SysEx Ende

Tabelle 2.2: Antwort der MIDIbox auf eintreffende Nachrichten

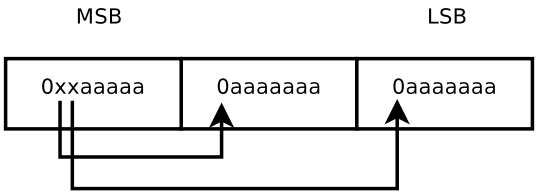

Abbildung 2.17: Aufteilung der Adressbits für die Programmierung der Software.

Code	Bedeutung
0x00	OK
0x01	Daten inkorrekt
0x02	kein Speicherplatz vohanden bzw. ausserhalb des adressierbaren Speicherbereiches
0x03	Protokollversionen stimmen nicht überein (MINOR)
0x04	Protokollversionen stimmen nicht überein (MAJOR)

Tabelle 2.3: Antwort-Codes

2.4.2 Device- / Modulstruktur

Die Device-Struktur enthält einen 20 Zeichen langen String, der in 2 Zeilen auf dem Display dargestellt wird. Anschließend werden 2 Byte zur Adressierung (siehe Tabelle 2.4) einer Datenstruktur des dynamischen Speichers verwendet. Selbiges gilt für die Modulstruktur, da sie aus der selben Klasse abgeleitet sind.

Datentyp	Feldname	Aufgabe
char[20]	name	zweizeiliger Name der auf dem Display angezeigt wird
unsigned int	dump_addr	Adresse auf eine Datenstruktur im dynamischen Speicher, um einen Editbuffer anzufordern

Tabelle 2.4: Device- und Modul-Struktur

2.4.3 Parameterstruktur

Die Struktur der Parameter, welche per Sysex übertragen werden und welche für die Erstellung von MIDI Nachrichten verantwortlich sind, wird mit der Tabelle 2.5 beschrieben. Wie schon im Kapitel 1.2.2 erwähnt, ist das höchstwertige Bit einer Variablen 0, da über MIDI nur 7 Bit Dateninformationen übermittelt werden. Entspricht das letzte Bit dagegen 1, wird das Byte als Statusbyte ausgewertet. Die MIDIBox GLCD unterstützt folgende MIDI Nachrichten: „Note On/Off“, „Aftertouch Polyphon/Channel Pressure“, „Pitchbend“, „Control Change“, „RPN“ und

„NRPN“. Außerdem wird das Versenden von MIDI SysEx Nachrichten und Editbuffers von der Box unterstützt.

Variablen	Länge (Hex)
name	10 Byte
messageChannel	1 Byte
modeSelect	1 Byte
number	2 Byte
valueMax	2 Byte
valueMin	2 Byte
value	2 Byte
scale	1 Byte
edit	2 Byte
sswitch	2 Byte

Tabelle 2.5: Aufbau der Datenstruktur eines Parameters

- Die Bytes 0 bis 9 werden für den Namen reserviert. Es kann also ein Name mit 10 Zeichen eingegeben werden.

- Das Message Channel-Byte steht an der Stelle 10. Die drei X im ersten Nibble des Bytes definieren den MIDI Data Type. Es sind hier sieben verschiedene Typen implementiert (siehe dazu 2.6). Kein Typ (Off), Note / Aftertouch / Pitchbend, Control Change, Program Change, NRPN, RPN und SYSEX. Diese Bits definieren gewissermaßen die Art, wie die späteren Bytes ausgewertet werden.

  Im zweiten Nibble des Bytes wird der MIDI-Channel festgelegt (siehe Tabelle 2.6).

0 X X X 0 0 0 0	*messageChannel*
Bitkombination	MIDI Nachrichtentyp
0 0 0	Off
0 0 1	Note On/Off, Aftertouch, Pitchbend
0 1 0	*** frei ***
0 1 1	Control Change
1 0 0	Program Change
1 0 1	NRPN
1 1 0	RPN
1 1 1	SysEx und Editbuffer

Tabelle 2.6: *messageChannel* - MIDI Nachrichtentyp

- Das Byte an Stelle 11 (siehe Tabelle 2.7) ist abhängig vom vorhergehenden Byte. Wenn im Message Channel Byte nun Control Change oder NRPN / RPN gewählt wurden (siehe Tabelle 2.6), definieren die drei mit X gekennzeichneten Bits den Controller Mode. So steht beispielsweise 00000000 für Absolut und 01000000 für Increment / Decrement. Wurde im Message-Channel-Byte dagegen 00010000 ausgewält, so steht nun 0000 für NOTE, 0001 für AFTERTOUCH und 0010 für PITCHBEND. Ähnlich verhält es sich auch für PROGRAM CHANGE, wenn im Vorgängerbyte 01000000 gewählt wurde: Stehen nun 00000000 für kein MSB oder LSB, 0001 nur für MSB, 0010 nur für LSB und 0011 für MSB und LSB.

  Darüber hinaus werden im zweiten Nibble weitere Informationen übertragen. So steht 00001000 für den Wechsel zwischen Drehpoti und Druckknopf, wobei 0 für Dreh- und 1 für Druckknopf steht (siehe Tabelle 2.8).

  Die Beschleunigung kann im Bit 00000100 mit 1 aktiviert werden (siehe Tabelle 2.9), im Bit 00000010 wird zwischen 7 Bit- und 14 Bit-Betrieb gewechselt. Bei Encodertasten im „Absolute“ Controller Mode steht Bit 2 für „Toggle Off/On“. Bei „Toggle Off“ wird beim Drücken der Wert aus $valueMax$ gesendet und beim Loslassen direkt der Wert aus „Value Min“. Bei „Toggle On“ wird beim Loslassen keine Nachricht gesendet, sondern erst beim erneuten Drücken. So kann man die Encodertaste als Schalter oder Taster verwenden. In den relativen Controller Modi legt dieses Bit fest, ob beim Drücken eine Incrementer oder eine Decrementer MIDI Nachricht versendet werden soll.

  Mit Bit 1 (siehe Tabelle 2.10) von „Mode Select“ stellt man die Auflösung der gewünschten MIDI Nachricht ein.

  Das Bit 0 (siehe Tabelle 2.11), also 00000001 zeigt an, ob der Wert in „Value Min“ ein negatives Vorzeichen hat. Diese Angabe ist wichtig beim Anzeigen von negativen Wertebereichen.

- Die Bytes „number“ (Stelle 12+13) sind die ersten zwei der folgenden Da-

0 X X X 0 0 0 0	*modeSelect*
Bitkombination	Note On/Off, Aftertouch, Pitchbend
0 0 0	Note On/Off
0 0 1	Aftertouch Channel Pressure
0 1 0	Aftertouch Polyphon Pressure
0 1 1	Pitchbend
	Controller Mode bei NRPN, RPN und Control Change
0 0 0	Absolute
0 0 1	Relativ1
0 1 0	Relativ2
0 1 1	Relativ3
1 0 0	Inc/Dec
	Bankselect bei Program Change
0 0 0	senden ohne Bankselect
0 0 1	senden mit Bankselect (MSB)
0 1 0	senden mit Bankselect (LSB)
0 1 1	senden mit Bankselect (MSB und LSB)

Tabelle 2.7: *modeSelect* - Bitkombinationen - Bit4 bis Bit6

0 0 0 0 X 0 0 0	*modeSelect*
Bitkombination	Encoder oder Encodertaste
0	Encoder aktiv
1	Encodertaste aktiv

Tabelle 2.8: *modeSelect* - Bitkombinationen - Bit3

tenbytes. Hier können zwei Hexadezimalzahlen, also 16384mögliche Dezimalzahlen dargestellt werden, welche für 14 Bit-Parameter wie CONTROLLER NUMBER und NRPN NUMBER benötigt werden. Bei 7 Bit wird Byte 13 gleich Null gesetzt und es stehen 128 Werte zur Verfügung. Dies ist der Fall bei KEY NUMBER, welche nur 7 Bit-Auflösung besitzt, kann aber auch für CONTROLLER NUMBER im 7 Bit-Modus gelten.

- „Value Min“ (Stelle 14+15) sind die nächsten Datenbytes in der Reihe, und stehen für den minimalen Wert eines Wertebereichs. Es gibt hier ebenfalls die 14 Bit- und 7 Bit-Betriebsart. Beim 7 Bit-Betrieb wird wieder das zweite Byte gleich Null gesetzt.

- „Value Max“ (Stelle 16+17) gibt die Obergrenze des Wertebereichs an, die Funktionsweise ist zu Value Min identisch.

0 0 0 0 0 X 0 0	*modeSelect*
Bitkombination	Acceleration bei Encoder
0	Acceleration nicht aktiv
1	Acceleration aktiv
	Toggle Off/On bei Encodertaste und Controller Mode „Absolute“
0	Toggle Off
1	Toggle On
	Inc/Dec bei Encodertaste und Controller Mode „Relativ1“, „Relativ2“, „Relativ3“, „Inc/Dec“
0	Incrementer Mode
1	Decrementer Mode

Tabelle 2.9: *modeSelect* - Bitkombinationen - Bit 2

0 0 0 0 0 0 X 0	*modeSelect*
Bitkombination	7Bit oder 14Bit Auflösung
0	7Bit
1	14Bit

Tabelle 2.10: *modeSelect* - Bitkombinationen - Bit 1

- „Value“ (Stelle 18+19) dagegen sollte ursprünglich einen Fixen Wert von der Bediensoftware zum Controller und von dort zum Synthesizer senden. Dieses Vorhaben wurde aber durch die Weiterentwicklung am Editbuffer nicht umgesetzt. Value wird daher mit Nullen gefüllt.

- Die Variable „Scale“ (Stelle 20) beinhaltet den Skalierungsfaktor. Es handelt sich um eine 7 Bit-Variable, welche durch die Benutzeroberfläche nur gewisse gerasterte Werte annehmen und an den Controller weitergeben kann. Scale operiert in den Schritten von 0, 1, 2, 4, 8, 16, 32, und 64.

- Bei „Edit“ (Stelle 21+22) handelt es sich um die beiden Edit-Buffer-Bytes. Die ersten 12 Bit codieren eine Adresse im Editbuffer, aus welcher der Parameterwert gelesen und in die er wieder eingetragen werden kann. Die anschließenden Bits „Send Editbuffer“ und „with switch“ werden danach definiert. Das niederwertigste Bit zeigt an, ob die „sswitch“-Struktur im dynamischen Speicher verwendet werden soll.

- Die Variable „sswitch“ (Stelle 23+24) stellt Bezeichner für zu übertragende Parameter bereit. Diese könne über die Checkbox „Table“ eingegeben werden. Die beiden Bytes dienen weiterhin der Adressierung des dynamischen Speichers.

0 0 0 0 0 0 0 X	*modeSelect*
Bitkombination	Ist der Wert in *valueMin* negativ?
0	*valueMin* positiv
1	*valueMin* negativ

Tabelle 2.11: *modeSelect* - Bitkombinationen - Bit 0

2.4.4 dynamischer Speicher

In diesem Speicherbereich werden mögliche Erweiterungen zu den Datenstrukturen des statischen Speicherblocks abgelegt. Es können drei Strukturen im dynamischen Block abgelegt werden. Diese werden im Folgenden beschrieben.

- **Sysex-Struktur**

  Die Sysex-Struktur erweitert die Parameterstruktur um die Möglichkeit, Sysex-Nachrichten an Synthesizer zu schicken. Dazu werden die in Tabelle 2.12 dargestellten Daten und ein Sysex-String benötigt, der im dynamischen Speicher anschließend an die Sysex-Struktur abgelegt ist. Die Länge dieses Strings wird im Feld „length“ angegeben. Weiterhin muß eine Konvertierungsfunktion und die Adresse auf der sie arbeitet, spezifiziert werden.

Datentyp	Feldname	Aufgabe
unsigned char	convert_func	numerischer Bezeichner der Konvertierungsfunktion
unsigned char	convert_func_addr	Adresse an der die Konvertierungsfunktion den Wert des Parameters schreiben soll
unsigned char	length	Länge der im Speicher folgenden Daten

Tabelle 2.12: SysEx-Struktur

- **sswitch-Struktur**

  Die „sswitch-Struktur“ besteht aus mehreren 6 Byte langen Zeichenketten, welche den verschiedenen Werten eines Parameters zugeordnet werden. Die Anzahl der Strings ist gleich der Anzahl der Werte, die ein Parameter annehmen kann.

- **Dump-Struktur**

  Damit bei der Auswahl eines Devices oder Moduls, ein Editbuffer vom Synthesizer angefordert werden kann, enthält die Dump-Struktur (siehe Tabelle 2.13) Informationen über Anfrage und Verwaltung des Editbuffers (siehe Tabelle 2.13). Für das Anfordern des Editbuffers wird der Dumpstruktur ein Sysex-String angehangen, dessen Länge im Feld „length“ spezifiziert ist. Für

diese Anfrage wird auch das Feld „wrong_header_length“ benötigt, so dass der MIDI-Controller über den Beginn des Editbuffers in der Antwort des Synthesizers informiert ist. Zum Zurücksenden des Editbuffers wird ein neuer Header benötigt, der nach dem Anfrage-String gespeichert ist. Dessen Länge ist durch das Feld „wrong_header_length“ bestimmt. Zur Übermittlung muß auch Checksumme des Editbuffers gebildet werden. Mit dem Feld „checksum_func“ und dem Feld „checksum_func_addr“ wird analog zur Konvertierungsfunktion der Device und Modulstrukturen eine Checksum-Funktion und eine Arbeitsadresse im Editbuffer spezifiziert. Um Parameterwerte aus dem Editbuffer zu laden und rück zu speichern, wird eine Konvertierungsfunkion benötigt, die in dem Feld „EB_convert“ codiert ist. Die Adresse für diese Funktion ist in der Parameterstruktur abgelegt.

Datentyp	Feldname	Aufgabe
unsigned char	typ	Art der Nachricht die verschickt werden soll (Editbuffer Request, SysEx Nachricht) : Timeout (Zeit bis nächster Dump der Kette abgearbeitet wird : Flag welches Anzeigt ob dieem Dump ein weiterer folgt
unsigned char	eb_convert	numerischer Bezeichner der Konvertierungsfunktion
unsigned char	wrong_header_length	Die Länge des vom Synthesizer zurückgegebenen Headers
unsigned char	edit_buffer_length	Die Länge des Editbuffers
unsigned char	length	Länge der im Speicher folgeden Daten
unsigned char	header_length	Länge des Editbuffer Headers der nach dem SysEx String im Speicher steht

Tabelle 2.13: Dump-Struktur

3 Aufbau der Bediensoftware

Die Software wurde in der Programmiersprache Java erstellt und zu diesem Zweck wurde das J2SE Developmentkit 5.0 verwendet. Diese Entscheidung brachte einige Vorteile mit sich. Zum einen ist die Software plattformunabhängig, sie kann also auf jedem beliebigen Betriebssystem laufen, zum anderen vereinfacht sich die Programmierung durch die Verwendung der Java Bibliotheken. So wurde zum Beispiel für die Gestaltung der grafischen Oberfläche das „Standard Widgets Toolkit“ von Eclipse benutzt [4]. Zur Erstellung der Software verwendeten wir die freie Entwicklungsumgebung „Netbeans“ von Sun Microsystems [3].

3.1 Hauptprogramm

3.2 Überblick

Grundsätzlich ist die Software in 3 Pakete unterteilt. Das Frontend, das Backend und das dritte, welches im weiterem Verlauf Mainend genannt wird. Die Aufgaben des Programms unterteilen sich somit wie folgt. Das Frontend Paket enthält sämtliche Klassen, die für den Aufbau der Benutzeroberfläche und dessen korrekte Funktionalität sorgen. Das Backend übernimmt teilweise Export und Import Funktionen und ist aber hauptsächlich für die Speicherung, Verarbeitung und Midifizierung [1] der eingegeben Daten verantwortlich. Das Mainend Paket ist sozusagen der Hauptteil, es beinhaltet die Klassen für den Transfer der MIDI-Daten an den Hardware Controller , den XML Import und Export und die Konfiguration der Software, also die Erkennung der MIDI-Schnittstelle und die Sprachauswahl. In der Abbildung 3.18 ist diese Aufteilung und die wichtigsten Klassen dargestellt. Die Rechtecke symbolisieren die Klassen und sind in den entsprechenden Paketen enthalten. Im Frontend sind die Klassen für die Dialoge und die Composites [2] zur besseren Übersicht nochmals in Paketen zusammengefasst.

[1] Die Midifizierung bedeuted die Serialisierung der Daten in einen Bytestream, der über die Midischnittstelle versendet und vom Hardware Controller als Konfiguration erkannt wird

[2] Ein Composite ist ein leeres Fenster ohne Titelleiste oder sonstige Dekorationen

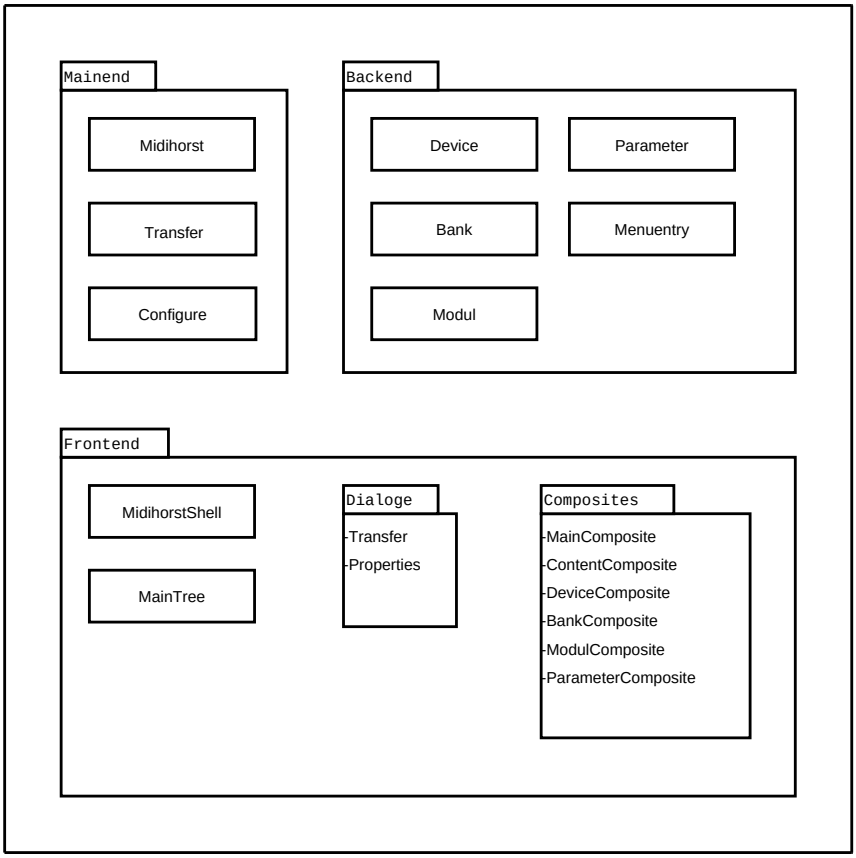

Abbildung 3.18: Die 3 Pakete der Software

3.3 Kommunikation mit Front- und Backend

Die Arbeitsweise des Programms hängt entscheidend von der Kommunikation dieser 3 Pakete ab. Die meisten Anwendungsfälle sind im Mainend implementiert. Diese Funktionen basieren auf den Daten des Backends und werden vom Frontend aufgerufen. Auf Abbildung 3.19 zeigt diese Verknüpfung. Das Mainend stellt die Basisfunktionen des Programms für den Benutzer zur Verfügung. Ausgangspunkt hierfür ist die Klasse *Midihorst*. Die Java Software startet mit der „main“-Methode (*public static void main()*). In dieser wird die Klasse *MidihorstShell* [1] aus dem Frontend Paket instanziert und das Programm startet die Oberfläche. Möchte der Benutzer ein „Device“ exportieren oder importieren werden im Frontend die Methoden *exportDevices()* bzw. *importDevices()* aufgerufen. Diese rufen wiederum *exportDevices()/importDevices()* [2] in *Midihorst* auf. Hier findet dann der eigentliche XML Datenaustausch statt. Analog dazu ist es auch beim Transfer der MIDI-Daten. Der Benutzer startet die „transfer“-Methode über die Oberfläche, diese wiederum ruft *transfer()* [3] in *Mididhorst* auf. Für die Spracheinstellung sind im Frontend Aufrufe von *getString()*[4] aus der Klasse *Midihorst* verankert. Dort wird über eine Konfigurationsdatei der zugehörige Text ausgewählt. Die Informationen die der Benutzer über die Oberfläche dem Programm mitteilt, also zum Beispiel Namen oder Parameter Werte, werden dem Backend über die sogenannten „setter“-Methoden [5] übermittelt. Hier wird für jede Eingabe im Frontend der entsprechende Wert im Ba-

ckend gesetzt und einer Variablen zugeordnet. Wird zum Beispiel bei „Midichannel“ eine 1 gewählt, dann sorgt *setChannel()* dafür dass das Backend den Wert übergeben bekommt. Für eine aktuelle Darstellung der Werte im Frontend, muss dieses die eingegebenen Daten zurückbekommen. Ändert der Benutzer die Ansicht und gelangt dadurch in eine Ebene, in der er zuvor Daten eingegeben hatte, werden die Werte aus dem Backend über „getter“-Methoden [6] zurückgegeben und im Frontend gesetzt. Die wichtigste Kommunikation findet zwischen dem Mainend und dem Backend statt.[7] Da das Backend als reiner Datenspeicher implementiert ist, wird dieses im Mainend vollständig verwaltet. Sowohl anlegen, löschen als auch zugreifen auf verschiedene „Devices“ wird innerhalb der Klasse *Midihorst* realisiert. Die „Devices“ werden dort in einer Datenstruktur (*Vector <Device> devices*) gespeichert.

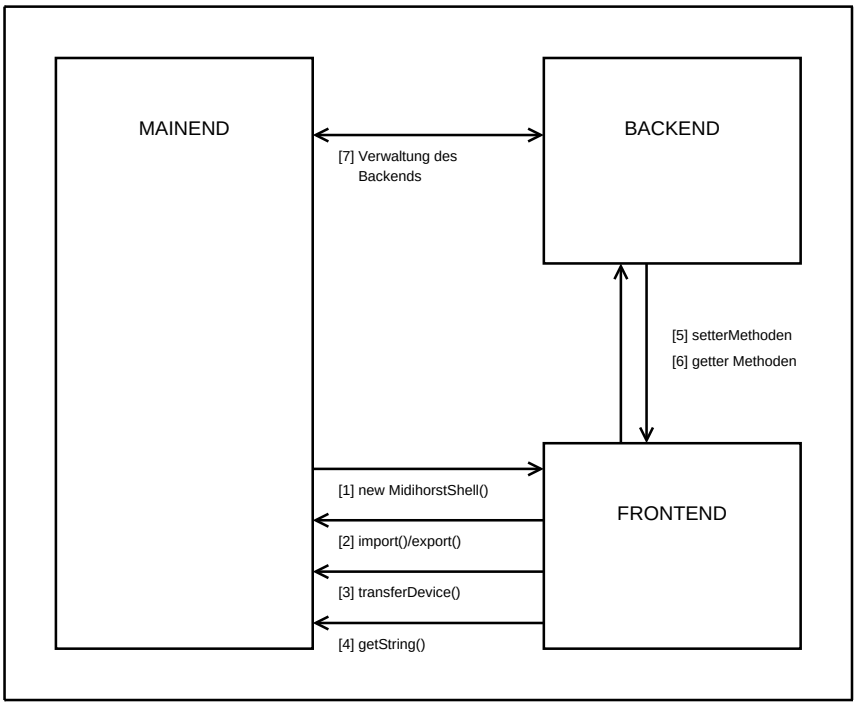

Abbildung 3.19: Kommunikation der 3 Pakete

3.4 Grundsätzliche Funktionen

3.4.1 XML-Import/Export

Für das Erstellen, Verarbeiten und Ausgeben der XML Daten wurde die auf Java basierende JDOM Bibliothek verwendet [1]. Die Methoden für den Import und Export sind in der Klasse *Midihorst* zu finden. Durch den Aufruf des Konstruktors der Klasse *org.jdom.document* wird ein XML-Dokument erzeugt. Diesem wird ein JDOM-Element als „root“-Element (*<midihorst type=“export“></midihorst>*) hinzugefügt [5]. *exportDevices(List<Device> devices, String filename)*,parametrisiert mit Dateinamen und Referenzen auf „Devices“, veranlasst dass sich die zu exportierenden Devices in das zuvor konstruierte „root“-Element eintragen. Nach dem

Speichern des XML Documents (org.jdom.document) in einer Datei ist der Export abgeschlossen. Beim Aufruf von *importDevices(String filename)* bekommt diese Methode den Namen der gewählten Datei aus dem Frontend übergeben und parst die in die XML-Daten enthaltenen „Devices“. Die Serialisierung eines oder mehrerer Devices ist im Anschnitt 2.3 detailliert beschrieben.

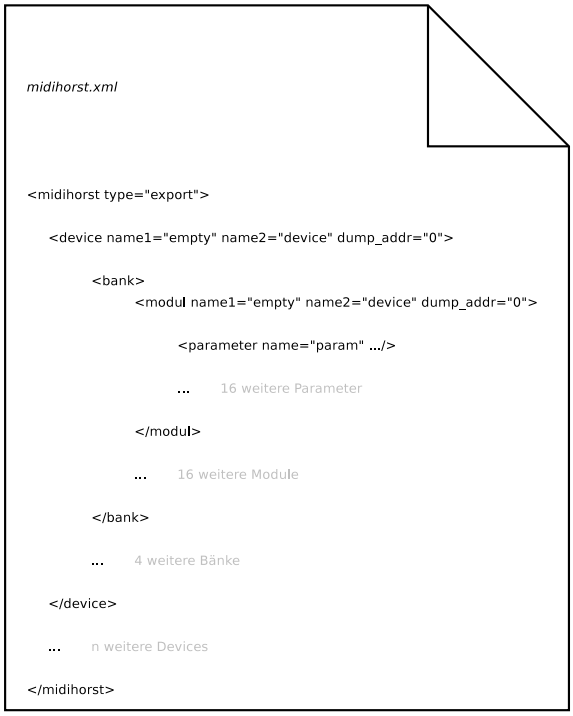

Abbildung 3.20: Beispielhaftes XML Dokument

3.4.2 Transfer

Der MIDI-Daten Transfer wird von der Klasse *Transfer* übernommen. Die Transferfunktionalität des Mainends überträgt die midifizierten Daten des Backends zum Hardware Controller. Dafür wird die Java-MIDI-Schnittstelle (Javax.sound.midi) benutzt. Diese ermöglicht es dem Programm SYSEX-Nachrichten zu versenden und zu empfangen. Da diese zeitintensiven Operationen das Frontend blockieren würden, ist das Senden und der Empfang in Threads ausgelagert. Zur Synchronisation des „Haupt-Threads“ mit dem „Transfer-Thread“ werden die zu transferierenden Daten zu „Transfer-Jobs“ zusammengefasst. Diese „Jobs“ werden über eine „Queue“ an den „Transfer-Thread“ übergeben. Dieser „Thread“ läuft seinerseits in einer Endlosschleife, welche „Transfer-Jobs“ aus der „Queue“ abfragt. Die Abruf Operation ist dabei blockierend, so dass der Thread keine unnötigen Systemressourcen verschwendet, Polling wird somit unterbunden. Der „Empfangs-Thread“ seinerseits, wird durch das Abfragen des MIDI-Eingangs blockiert, bis eine MIDI-Nachricht eintrifft. Diese Nachricht wird verarbeitet und an einen „Event-Listener“ verschickt. Danach kehrt der „Thread“ wieder zum blockierenden Empfangs Aufruf zurück.

Registrierter „Event-Listener“[3] ist das „Transfer-Objekt“, das bei gewünschter bidirektionaler Kommunikation mit dem Hardware Controller über dessen Antwort informiert wird. Über diese Information ist der „Transfer-Thread“ mit dem „Transfer-Objekt“ synchronisiert. Die Klasse *Transfer* ist als Singleton implementiert, so dass es nur eine Instanz im Programm geben kann. Dadurch wird ein sequentielles Abarbeiten der „Transfer-Jobs“ erzwungen. Bei der Instanzierung der *Transfer* Klasse wird versucht, dass in der „Konfigurations-Klasse“ gespeicherte MIDI-Gerät zu initialisieren. Misslingt dies, wird der Fehler abgefangen, woraufhin der Aufrufer den „Properties-Dialog“ startet. Am Programm Ende gibt es 2 Möglichkeiten den „Transfer-Thread“ zu beenden. Zum einen lässt sich der Thread „hart“ während des Übertragens beenden. Die 2. und auch empfohlene Möglichkeit ist das „weiche“ Beenden, bei dem der Thread nur dann unterbrochen wird falls dieser gerade auf einen neuen „Transfer-Job“ wartet. Die Kommunikation zwischen den Threads ist in der Abbildung3.21 dargestellt.

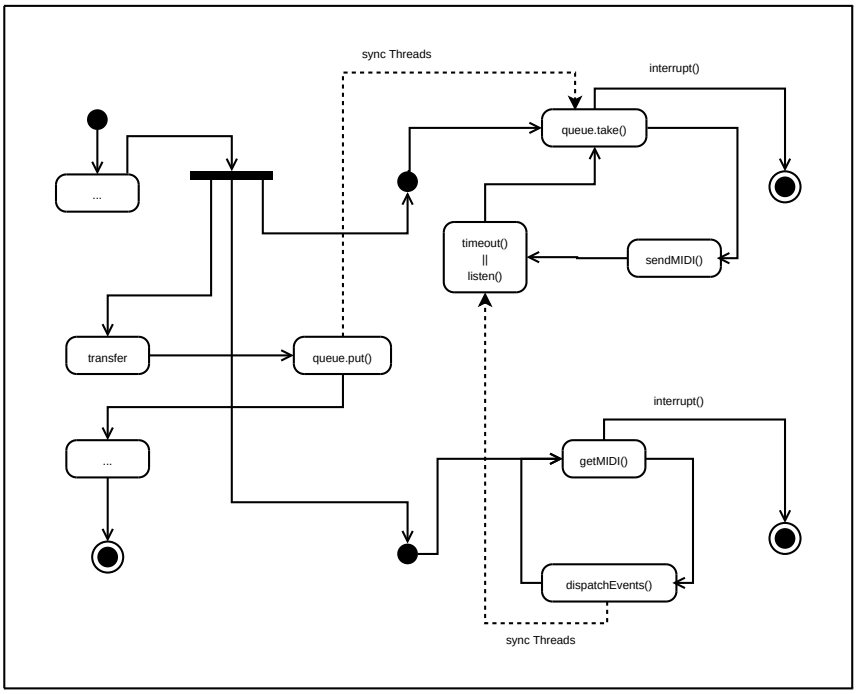

Abbildung 3.21: UML-State-Diagram für den Transfer

3.4.3 Konfiguration

Das „Konfigurations-Objekt“ als Singleton implementiert, bündelt alle globalen Programm Eigenschaften und enthält Schreib und Lese Algorithmen um diese Informationen persistent zu machen. Angepasst werden diese Eigenschaften durch den im Frontend enthaltenen „Properties“-Dialog. Dort kann man ein MIDI-Gerät sowohl für den Eingang aus den Ausgang wählen. Desweiteren ist es möglich sich zwischen

[3]Event-Listener sind Objekte, die Methoden zur Behandlung von Ereignissen bereitstellen. In der Regel werden diese über ein Interface vom Objekt abstrahiert

einer bidirektionalen oder einer unidirektionalen Übertragung mit konfigurierbaren Timeout zu entscheiden. Außerdem kann bei der Sprachauswahl zwischen Deutsch und Englisch gewählt werden.

3.5 Backend

3.5.1 Gesamtstruktur

Das Backend besteht aus einem Verbund von Objekten und Klassen die bei Bedarf instanziert werden, deshalb hat es einen passiven Charakter. Aufgabe des Backends ist es die vom Benutzer eingegeben Daten zu verwalten und dem Mainend sowie dem Frontend zu Verfügung zu stellen. Das Frontend übergibt die Werte aus jeder Ebene (Device, Bank, Modul, Parameter) jeweils der zugehörigen Klasse im Backend. Wird in der Parameter Ebene ein Wert eingegeben dann wird dieser in der Klasse *Parameter* im Backend gesetzt und gespeichert. Dies verhält sich in den anderen Ebenen genauso. Damit das Mainend den vollständigen Datensatz von einem „Device“ enthält, sind die Klassen im Backend miteinander verknüpft. Das Mainend speichert eine Referenz auf das Objekt „Device“, dieses wiederum enthält direkte oder indirekte Referenzen auf Objekte der untergeordneten Backend Klassen, die zur Speicherung eines vollständigen „Device“ Datensatzes erforderlich sind. Die Abbildung 3.22 zeigt die Verbindung zwischen den Klassen, um dies zu ermöglichen. Es gibt für jede Ebene eines „Devices“ eine zugehörige Klasse. Da *Device* und *Modul* eine ähnliche Struktur haben sind diese von der Klasse *Menuentry* abgeleitet. Ein „Device“ enthält 4 „Bänke“, die wiederum 16 „Module“ enthalten. Jedes „Modul“ enthält 16 „Parameter“. Diese Struktur wurde analog zum Hardware Konzept so auf das Backend übertragen. *Device* aggregiert *Bank* mit der Kardinalität 4, *Bank* aggregiert wiederum *Modul* mit einer Kardinalität von 16, welches *Parameter* aggregiert. Wird nun einer der Basisfunktionen, wie „Device“ erstellen oder importieren aufgerufen, dann instanziert die Klasse *Midihorst* ein „Device“ Objekt und speichert dessen Referenz. Durch den Aufruf des Konstruktors werden auf Grund der oben erklärten Verbindung zwischen den Klassen des Backend zusätzlich auch Referenzen auf die ungeordneten Ebenen gespeichert, erzeugt und in den betreffenden Objekten abgelegt. So steht dem Mainend durch die Instanzierung des einen Objekts der komplette Datensatz zu Verfügung. Diese Verkettung von Referenzen ist grundlegend für die Midifizierung und den XML-Import/Export.

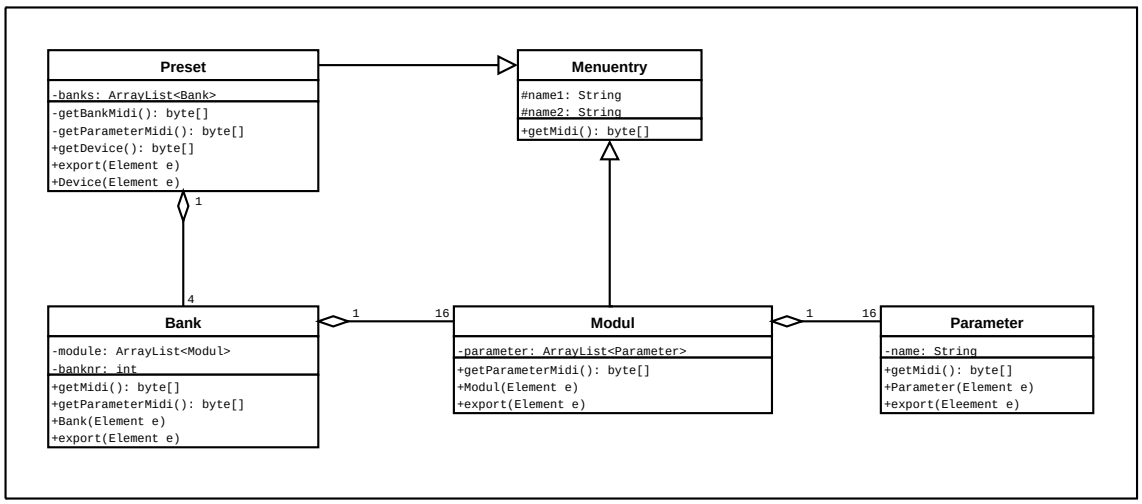

Abbildung 3.22: UML-Klassen-Diagramm für das Backend

3.5.2 Midifizierung

Die Serialisierung der Daten ist mit der Hilfe von “getter“-Methoden umgesetzt. Die Abbildung 3.23 zeigt ein Sequenz Diagramm, welches die zeitliche Abfolge der Methoden Aufrufe darstellt. Von *getDevice()* aus wird die Sequenz gestartet. Die *getDevice()* Methode des „Device“-Objektes implementiert die vollständige Serialisierung eines „Devices“. Aus diesem Grund enthält sie *getMidi()*, *getBankMidi()* und *getParameterMidi()*. Die Methode *getMidi()* bewirkt die eigentliche Serialisierung, sie beinhaltet die Operationen für die Umsetzung der Namen und Werte in eine Sequenz von Bytes. *getBankMidi()* und *getParameterMidi()* dagegen rufen wiederum direkt und indirekt *getMidi()* auf, um aus der jeweiligen Ebene die Werte zu midifizieren. Da die „Bank“ Ebene nur für die Kapselung von 16 „Modulen“ verantwortlich ist wird die Midifizierung der in der „Bank“ enthaltenen „Module“ zu einer Sequenz konkateniert. Jede dieser Methoden hat einen Rückgabe Wert in Form eines Byte Arrays. Diese Verschachtelung bewirkt, dass am Ende mit dem Aufruf von *getDevice()* das „Device“-Objekt in 3 Byte Arrays serialisiert wurde und somit übertragen werden kann.

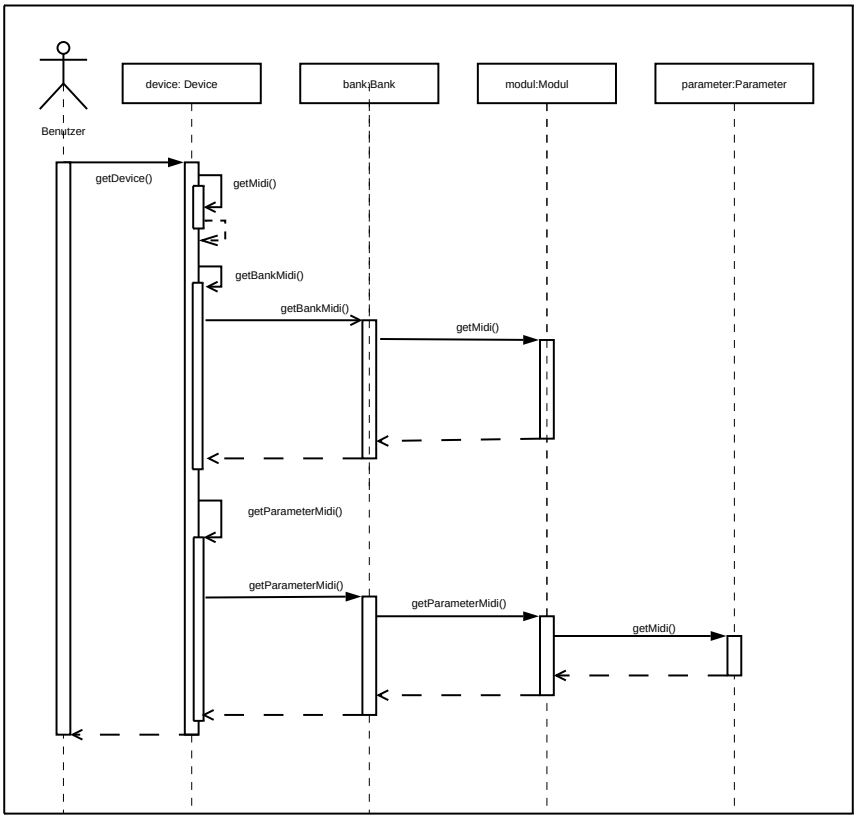

Abbildung 3.23: UML-Sequenz-Diagramm für die Midifizierung

3.5.3 XML-Import/Export im Detail

Bei der Ausgabe werden alle zu serialisierenden „Device“ Objekte als Kinder Knoten[4][link!] im „root“Element abgelegt. Dafür wird die *export*-Funktion jedes „Devices“ mit dem *midihorst*-Element aufgerufen. Jedes „Device“ muss seine Vater Klasse (*MenuEntry*) mit dem neu erstellten *device*-Knoten aufrufen, so dass dessen Eigenschaften persistent gemacht werden können. Des Weiteren sind auch die 4 *export*-Funktionen der „Bänke“ jedes „Devices“ aufzurufen, so dass auch diese sich in den *device*-Knoten einfügen. Dieser Algorithmus gilt auch für die restlichen Objekte (Bank, Modul und Parameter) wie in Abbildung 3.24 zu sehen ist.

[4]folgend werden Knoten und Element synonym verwandt

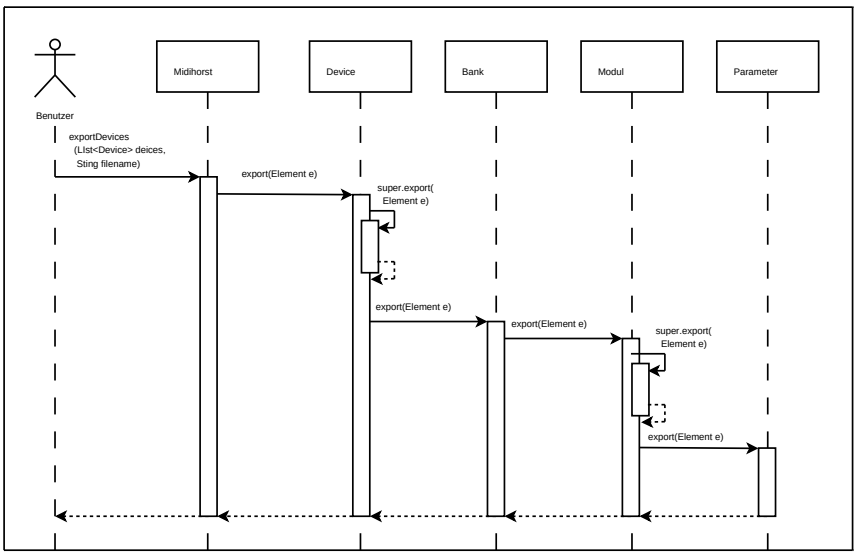

Abbildung 3.24: UML-Sequenz-Diagramm für den Export

Aufgabe der Importfunktion ist es, aus einer durch den Benutzer gewählten XML-Datei neue „Device“ Objekte zu erstellen. Die XML-Datei muss als „root“-Element den Knoten *midihorst* besitzen und dessen Attribut *type* den Wert *export* haben. Sind diese Bedingungen erfüllt werden im nächsten Schritt alle *device*-Knoten, die in *midihorst* enthalten sind extrahiert. Für jedes „Device“-Element wird der *Device* Konstruktor, mit diesem Element parametrisiert und aufgerufen. Aus diesem XML-Knoten lässt sich nun das gesamte „Device“ mit all seinen untergeordneten Objekten konstruieren. Zu Beginn ruft *Device* den Konstruktor seiner Vater Klasse *MenuEntry*,mit dem ihm übergebenen XML-Element auf. Aus den Attributen *name1*, *name2* und *dump_addr* lassen sich die Eigenschaften einer *MenuEntry* Instanz konstruieren. Nach der Initialisierung der Vater Klasse müssen jetzt noch die 4 „Bänke“, die ein „Device“ aggregiert erstellt werden. Dafür werden alle Kinder-Knoten vom Typ *bank* des übergebenen *device*-Elementes in einer Schleife durchlaufen und wiederum die Konstruktoren der „Bänke“, parametrisiert mit dem aktuellen Kinder-Knoten aufgerufen. Für „Bänke“, „Module“ und „Parameter“ erfolgt diese Konstruktion wie in Abbildung 3.25 zu sehen ist analog zu der Instanzierung der *Device* Klasse.

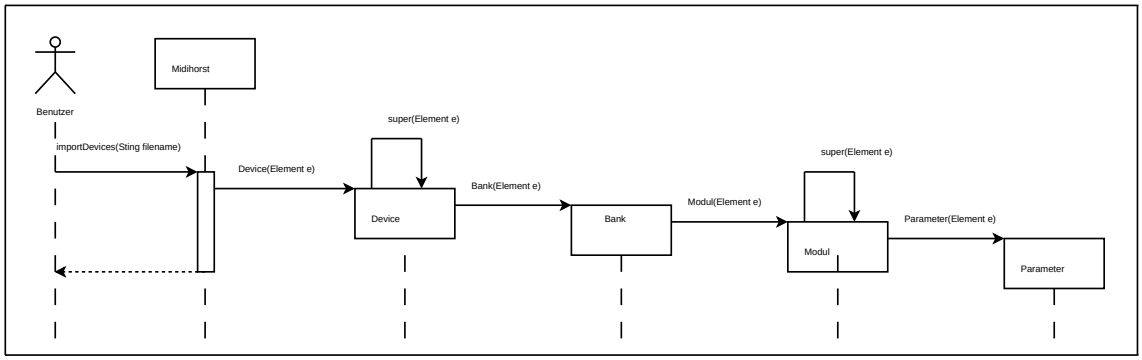

Abbildung 3.25: UML-Sequenz-Diagramm für den Import

In den beiden Abbildungen wurde auf die Darstellung der Kardinalität der aggregierten Objekte zur besseren Übersicht verzichten. Sie entspricht der in Kapitel 2.1 gezeigten Verknüpfung der Klassen des Backends.

3.6 Frontend

3.6.1 Gesamtstruktur

Das Frontend lässt sich grundsätzlich in 3 Komponenten aufteilen. Die Klassen *MidihorstShell*, *Maintree* und das *MainComposite*. Ersteres erstellt das Hauptfenster des Programms mit Menüleiste und Symbolleiste. Maintree erstellt den Baum, um zwischen „Devices“, „Modulen“ und „Parameter“ zu navigieren. Das *Maincomposite* stellt einen Platzhalter für *ContentComposites* dar. *Contentcomposite* steht den Klassen, für die Konfiguration von „Device“, „Modul“ und „Parameter“, als Interface zur Verfügung. Alle diese Klassen sind von *ContentComposite* abgeleitet. Mit Hilfe dieses Interfaces kann der Maintree die aktuell selektierte Ebene mit einem entsprechenden „Composite“ anonymisiert in die *Maincomposite* einbetten. Dies bedeutet, dass das der selektierten Ebene zugehörige „Composite“ in *Maincomposite* angezeigt wird. Abbildung 3.26 zeigt diese Aufteilung nochmal aus der Sicht des Benutzers. Eine Ausnahme bildet hier die Parameter Ebene(Die zugehörige Klasse ist in der Skizze nicht dargestellt). Das *ParameterComposite* enthält nur die Ansicht zur Editierung eines Parameters, daher werden durch 2 *Parameterscomposites* werden jeweils 8 *ParamterComposites* dargestellt, so dass über 2 Einträge im Baum die 16 Parameter eines Moduls verfügbar sind.

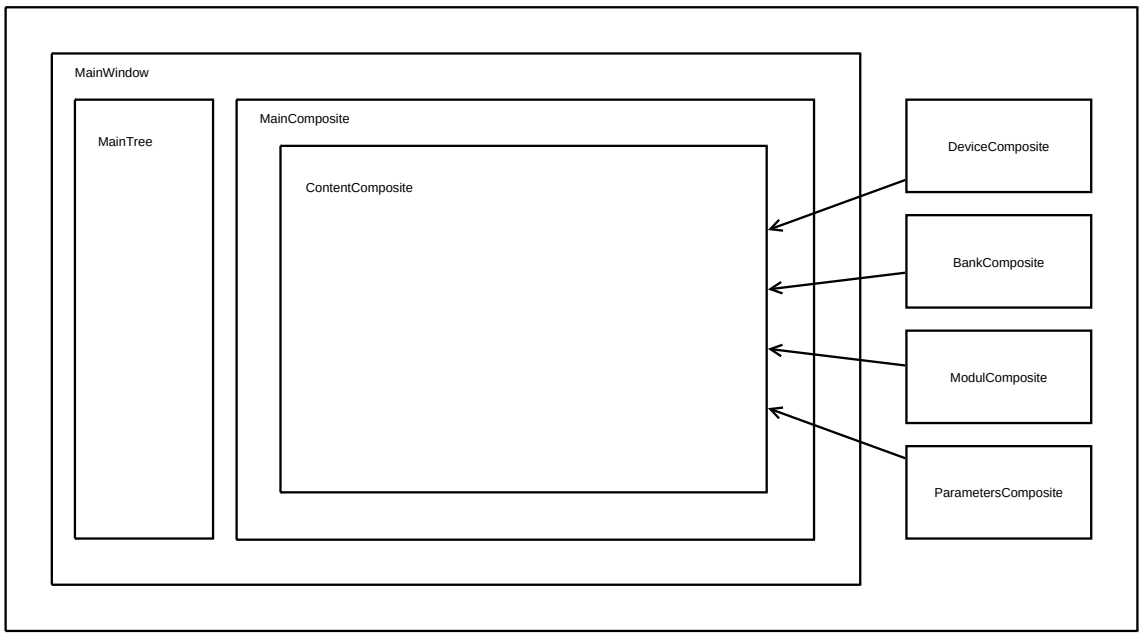

Abbildung 3.26: Übersicht zu den Composites

3.6.2 Erstellung der Oberfläche

Die Generierung der Benutzeroberfläche verläuft in den Klassen des Frontends nach dem gleichen Prinzip. Jede Klasse konstruiert zunächst ein „Composite“, welches entweder direkt auf dem Hauptfenster oder auf untergeordneten „Composites“ angebracht ist. Dieses „leere“ Fenster wird dann mit Hilfe von Methoden Aufrufen mit „Widgets“ gefüllt. Ein „Widget“ ist eine Komponente der Oberfläche, die eine Interaktion mit dem Benutzer erlaubt. In der „Parameter“ Ebene sind dies zum Beispiel Eingabefelder für Namen, Auswahlmenüs für die Werte Eingabe oder Knöpfe. Diese werden, wenn nötig mit Inhalt gefüllt und am jeweiligem „Composite“ ausgerichtet. Jede Komponente enthält zudem „listener“ die Veränderungen durch den Benutzer registrieren und „setter“-Methoden aus dem Backend aufrufen. Damit keine widersprüchlichen oder nicht vorgesehenen Eingaben an das Backend übertragen werden, wird die Eingabe bereits im Frontend beschränkt. Dies ist über verschiedene *switch-case*-Anweisungen und *if*-Schleifen realisiert. So wird zum Beispiel bei der Auswahl des Wertebereichs die Größe des Maximums automatisch an die des Minimums angepasst. So kann man keinen Minimum Wert eingeben, der größer als der Maximum Wert ist.

3.6.3 Dialoge

- Der „Hilfe“-Dialog ist über die Menüleiste zu erreichen und bietet alle Informationen über die Bedienung der Software.
- Im „Properties“-Dialog wird die Konfiguration der des Programms vorgenommen.
- Über den „Transfer“-Dialog kann über eine grafische Darstellung des Hardware Controllers der Slot bestimmt werden, in den das zu übertragen „Device“ gespeichert werden soll
- Der „About“-Dialog ist ebenfalls über die Menüleiste zu erreichen und enthält Informationen über das Projekt.

3.6.4 Baum

Der Baum zeigt die in der Klasse *Midihorst* gespeicherten „Devices“, deren „Bänke“, „Module“ und „Parameter“ an. Zu jedem dieser Einträge werden auch Referenzen auf das jeweilige Backend-Objekt gespeichert. Wählt nun der Benutzer einen dieser Einträge wird ein *ContentComposite* instanziert, welches dem Eintrag entspricht(zum Beispiel *DeviceComposite*). Dem Konstruktor des *ContentComposites* wird dazu die Referenz auf das Backend-Objekt übergeben, so dass dieser das „Composite“ mit den entsprechenden Daten füllen kann. Ändert der Nutzer die Werte im „Composite“ können diese mit Hilfe der Referenz im Backend geändert werden.

4 Fazit

4.1 Probleme / Ausblick

Im Laufe der Projektarbeit traten verschiedene Probleme und Schwierigkeiten auf, welche Workarounds und teilweise auch Kompromisse erforderten.

- Projektplanung

  Bei der Projektplanung begannen die Schwierigkeiten bereits, als es darum ging das Einsatzfeld der „Bedienoberfläche eines MIDI-Controllers“ abzugrenzen. So wurde die Produktgruppe der MIDI-Controller genauer untersucht und die Konzepte der Hersteller kritisch hinterfragt. Desweiteren war es nötig die Gruppe der zu bedienenden Geräte, also die Synthesizer und Klangerzeuger zu begutachten. Dies hatte sowohl für zeitgenössische Produkte, als auch für Geräte aus der Anfangszeit des MIDI-Standards zu geschehen, um wirklich eine universelle Bedienoberfläche gestalten zu können. Problematisch war hierbei, von alten Geräten verschiedener, teilweise nicht mehr existenter Hersteller, die Handbücher mit den nötigen Informationen und MIDI-Implementationen zu erhalten und auszuwerten. Es wurden ca. 150 Handbücher verschiedener Geräte aus dem Zeitrahmen von ca. 20 Jahren untersucht. Daraus wurde bestimmt, welche Arten der MIDI-Nachrichten und Parameter editierbar sein müssen, und man kam zu dem entschluß, da jeder Hersteller eigene Vorstellung der Ansteuerung hatte, nach Möglichkeit alle Parameter des MIDI-Standards (außer MMC und Sampledump-Befehlen) zu implementieren. Daraufhin wurden verschiedene Bedienkonzepte auf dem Papier skizziert und mit Absprache der Hardwaregruppe verworfen und im Abgleich zu ihrem Bedienkonzept erweitert. Ein Problem welches hier auftrat, war die schwierige Machbarkeitsanalyse und die Validierung der Konzepte für dieses Hard- und Softwareumfassende Projekt.

- Programm-Strukturierung

  Kompliziert war ebenso, alle möglichen Usecases in der Theorie durchzuspielen, die Erfahrung der Projektgruppe mit MIDI-Klangerzeugern war dabei sehr hilfreich. Eine große Aufgabe war ebenso ein eigenes Protokoll für die Kommunikation der Soft- und Hardware zu entwickeln, unter Berücksichtigung Bedienkonzepte und der umfangreichen Parameterimplementation. Themen wie Speicheraufteilung auf der Hardware-Plattform waren nicht nur für das andere Projekt relevant, sondern auch für die Software-Gruppe, da das Programm Daten im richtigen Format auf den physikalischen Speicher schreiben soll. Desweiteren stellte die Plangung der Algorithmen für Sysex aufgrund der Hersteller-Freiheiten ein Problem dar. Ein Auslesen / Überschreiben von Edit-Puffern war ebenfalls schwierig in der Planung, welches aber durch die

Einführung der „Convert Functions“ gelöst wurde. Ein Nachteil ist hier allerdings, dass man, falls man einen speziellen Synthesizer besitzt, sich seine eigene Funktion programmieren muß, da die Anzahl der Verschiedenen Fälle einfach zu groß ist um sie fest zu implementieren.

- Implementierung

  Ein großes Problem war die abschätzung des Umfangs der einzelnen Programmteile, wie Front-, Main- und Backend. Die starke Gewichtung zu Seiten des Frontends war nicht absehbar und wurde falsch priorisiert. Während der MIDI-Im-/ Export, sowie die Speicherung im XML-Format duchr die vorhandenen APIs gut umsetzbar war, verschlang die Implementierung des Frontends den Großteil der Zeit. Trotz des Funktionierenden MIDI-Im/Export, stand man vor einer schwierigen Kommunikation mit dem Hardwarecontroller, teils wegen des eigenen Protokolls und teils wegen Hardware-seitiger Probleme. Diese waren verschiedenen Natur, Hauptproblem war der falsch dimensionierte Programmspeicher, welcher durch Lieferzeiten (auch bei anderen Bauteilen) zu Verzögerungen führte. So war kurz vor Projektende keine vollständig funktionstüchtige Hardware (inclusive Midi-Merger) zum testen vorhanden.

- Ausblick

  Für zukünftige Projekte könnte man sich eine große Bibliothek an „Convert Functions“ für die gängigsten Synthesizer vorstellen zu implementieren, damit der Musiker keine Mühe hat sich seinen Synthesizer zu konfigurieren. Andererseits ist die Community von www.ucapps.de, auf welcher dieses Projekt inklusive der Sourcecodes zum download bereit stehen wird, sehr aktiv und hilfsbereit. Ein Tausch von Devices und eventuell auch „Convert Functions“ kann angenommen werden. Zudem wären Verbesserungen der Oberfläche und der Funktionalität, wie z.B. Copy and Paste oder Drag and Drop von Parametern für eine schnellere Bedienung wünschenswert.

Literatur

[1]] www.jdom.org, aufgerufen bis 31.09.2006

[2] www.zem-collage.de/midi, aufgerufen bis 31.09.2006

[3] www.netbeans.org, aufgerufen bis 31.09.2006

[4] www.eclipse.org/swt, aufgerufen bis 31.09.2006

[5] www.w3.org/TR/2006/REC-xml11-20060816/, aufgerufen bis 31.09.2006

[6] Richter, M.; Stöcklmeier, C.; Prang H.: „Universelle Bedienoberfläche für MIDI-Synthesizer: Hardware und Microcontroller-Code“, Medienprojekt

Anhang

CD Inhalt:

- Internetseiten
- Projektarbeit
- Sourcen

Erklärung:
Die vorliegende Arbeit ist eine Kollektivarbeit. Von Herrn Jacob Korn wurden Kapitel 1 ,Kapitel 2 bis Abschnitt 2.4 und Kapitel 4 verfasst.
Von Herrn Volker Dümke wurden Abschnitt 2.4 und Kapitel 3 verfasst.
Die enthaltenen Bilder wurde alle gemeinsam erstellt bis auf Abbildung 1.1, 2.2, 2.3, 2.4, 2.5, 2.17 und alle Tabellen aus Abschnitt 2.4. Diese wurde von dem Medienprojekt Richter, M.; Stöcklmeier, C.; Prang H.: „Universelle Bedienoberfläche für MIDI-Synthesizer: Hardware und Microcontroller-Code erstellt“
Wir erklären, dass wir diese Arbeit selbststaendig durchgeführt und abefasst haben. Quellen, Literatur und Hilfsmittel, die von uns benutzt wurden, sind als solche gekennzeichnet.

Unterschriften

Ilmenau, den 2. Oktober 2006