Schriftliche Dokumentation zum Thema: „Ansteuerung einer selbst entwickelten Multimedia-Software, mittels MIDI-Controller.“

Verfasser: Peter Müller

Max-Eyth-Schule Kassel

Besondere Lernleistung

Prüfer: Herr A. John

und Herr C. Dippel

Abgabedatum: 24.03.2010

Inhaltsverzeichnis

1. Einleitung

Ein Begriff, der gerade in der heutigen Zeit, mit den Innovationen der Technik immer mehr an Bedeutung erlangt ist Multimedia. Wie wichtig dieser Faktor ist sieht man schon daran, dass große Konzerne neue Zielgruppen unter Privatpersonen erschließen, anstatt sich weiter ausschließlich auf ihre Business-Partner zu konzentrieren. Beispiele dafür sind unter anderem die nachträglich veröffentlichte Media-Center Edition des Windows XP Betriebssystems, externe Multimedia-Festplatten, die Video-Plattform YouTube oder Apples bekannte iPod-Reihe. Aus diesen Gründen und eigenem Interesse habe ich bereits in der Jahrgangsstufe 12 im Rahmen eines Projektes der Datenverarbeitungstechnik einen Mediaplayer für Windows-basierte Systeme entwickelt.

Als Langjähriger Festival-Besucher ist mir ebenfalls die Entwicklung ausnahmslos aller DJs, weg von den Schallplatten, hin zur MP3, bekannt. So gehören zum Equipment eines DJs nur noch sein Laptop und ein USB-Controller, falls diesen nicht schon der Veranstalter stellt. Einen solchen USB-Controller besitze ich auch, seit ich vor einigen Jahren als Hobby mit dem so genannten DJing angefangen habe. Der Controller lässt sich laut Hersteller Numark nicht nur für die mitgelieferte DJ-Software, sondern für jede beliebige Software, welche das MIDI-Protokoll unterstützt verwenden.

Daher habe ich mich entschlossen als besondere Lernleistung den angesprochenen Mediaplayer so zu modifizieren, dass er als DJ-Software verwendet und von dem MIDI-Controller Numark TotalControl (folgend als NTC abgekürzt) gesteuert werden kann. Der Schwerpunkt dieser schriftlichen Ausarbeitung liegt, wie bei dem gesamten Arbeitsprozess, auf der Realisierung der Kommunikation zwischen der Hardware und Software. Zu beachten ist der Abgabetermin, da auftretende Probleme und deren Lösung, bzw. Alternative Lösungswege nur bis zu diesem Zeitpunkt dokumentiert werden können.

2. Kurzfassung des Arbeitsprozesses

Nach ausgiebigen Recherchen über das MIDI-Protokoll und dessen Funktionsweise wird eine eigene C#-Klasse zu entwerfen sein, die möglichst komfortabel die Kommunikation zwischen der Software und dem NTC ermöglicht. Die Klasse ist auf diesen Verwendungszweck zu optimieren, da das MIDI-Protokoll weit funktionsreicher ist als letztendlich benötigt. Zu realisieren ist zudem eine beidseitige Kommunikation, da auch die Software auf dem NTC befindliche LEDs ansteuern können sollte. Es wird also sowohl der Empfang von MIDI-Daten, wie auch deren Versenden nötig sein. Zur Steuerbarkeit der Software ist logischerweise auch die Analyse der empfangenen Daten Voraussetzung.

Aufbauend auf dem Mediaplayer wird eine Deck-Klasse sinnvoll sein. Vom Mediaplayer wurde lediglich die Wiedergabe einer Musikdatei verlangt. Eine DJ-Anwendung kommt ohne ein zweites Deck nicht aus, wodurch der fließende Übergang vom aktuellen zum nächsten Song erst möglich wird. Des Weiteren plane ich eine Playlist-Klasse pro Deck fest ein, welche mir persönlich bei anderen DJ-Programmen fehlt. Akustische Effekte wie z.B. Echo existieren bereits im Mediaplayer, genauso wie ein Equalizer. Diese beiden Funktionen werden vermutlich eins zu eins übernehmbar sein. Eine letzte Schwierigkeit dürfte das Realisieren des, aus der DJ-Szene bekannten, scratchens sein.

3. Die Hardware - Numark TotalControl

3.1. Verwendung und Zielgruppe

Der NTC richtet sich sowohl an Hobby- als auch Profi-DJs. Beispielsweise bei Musikfestivals werden heutzutage ausschließlich solche MIDI-Controller anstelle der einstigen Turntables (Plattenteller) genutzt. Grund dafür ist der meist eng zusammengestellte Zeitplan und der Arbeitsaufwand in den Umbauphasen. Während zur Zeit der Turntables noch extra alle Vinyls (Schallplatten) mitgeliefert werden mussten, genügt mittlerweile das digitalisierte Pendant - die Musikdatei auf einem Laptop. Mit entsprechender DJ-Software wie zum Beispiel Cue, ebenfalls von Numark, können diese Musikdateien wie echte Vinyls gemixt und performt werden. Ein Nachteil ist, dass diese Schritte vom DJ ungewohnt mit der Maus und Tastatur des Laptops durchgeführt werden müssten. Bei diesem Problem kommt der NTC zum Einsatz. Der NTC wird via USB-Schnittstelle an den Laptop angeschlossen und ist somit in der Lage die Software anstelle der Maus und Tastatur zu bedienen. Dies ermöglicht eine intuitive Nutzung der Software, da das Layout des Controllers wie der frühere Equipment-Umfang eines DJs aufgebaut ist. So werden zwei Turntables und ein Mixer, welcher beide Soundsignale der Turntables zusammenfügte und an die Soundanlage weiterleitete, in einem Gerät vereint – dem Numark TotalControl. Der große Vorteil des NTC ist der stark verringerte Einsatz von Equipment. Die Vinyls befinden sich zudem in digitaler Form auf dem Laptop. So ist zum einen der Transport, Aufbau und Einsatz für Profi-DJs um einiges vereinfacht, als auch der finanzielle Rahmen für Hobby-DJs bei der Anschaffung des Equipments erschwinglicher.

3.2. Komponenten und Layout

Auf der Bedienoberfläche des MIDI-Controllers befinden sich insgesamt 31 Buttons (Knöpfe), 20 Drehregler, 5 Fader (Überblendregler) und 2 sogenannte Jog Wheels (frei übersetzt Schubsrad). Das Gerät ist vom Layout an einen Zweikanal-Mixer mit je einem Turntable angelehnt, bietet also zwei identisch auf-

gebaute Decks und zusätzliche, globale Komponenten (Buttons und Drehregler) welche sich auf beide Decks auswirken.

Im Folgenden gehe ich kurz auf die Funktionalität des NTC ein, welche sich noch auf die Cue DJ-Software von Numark bezieht. Die von mir entwickelte Software soll ich jedoch an der Cue orientieren und möglichst denselben Funktionsumfang erreichen. Des Weiteren wird jeder erläuterten Komponente eine Nummer zugeordnet, damit veranschaulicht ist, wo sich diese auf dem NTC befindet. Eine Skizze mit dem NTC befindet sich dafür im Anhang.

3.2.1. Deckabhängige Buttons und Regler

Auf jedem Deck gibt es drei Equalizer-Drehregler für das Anpassen der Höhen (80/82), Mitten (85/81) und des Basses (83/84). Diese Drehregler fungieren gleichzeitig als Buttons (16/17/18/19/20/21), wodurch beim Drehen zum Beispiel der Bass verstärkt werden kann, beim Drücken den Buttons aber die sogenannte Kill-Funktion gesetzt wird. Die Kill-Funktion des Bass-Reglers reduziert alle Bässe abrupt auf ein Minimum, beziehungsweise setzt diese bei wiederholtem Drücken auf den Ausgangswert zurück.

Direkt darüber befindet sich der Gain-Regler (13/14) welcher ebenfalls für jedes Deck individuell den Verstärkungsfaktor der Lautstärke festlegt.

Sind mehrere Soundkarten beim eingesetzten Laptop verfügbar, kann eine für die Wiedergabe auf der Soundanlage und die andere für zusätzliche Kopfhörer des DJs verwendet werden. Da zur Soundausgabe in diesem Fall zwei verschiedene Signale gleichzeitig wiedergegeben werden können, ist der DJ in der Lage den nächsten Song vorzuhören, während das Publikum nur den aktuellen Song hört. Um festzulegen welches der beiden Decks für die Ausgabe auf dem Kopfhörer zuständig ist, dient der PFL-Button (48/55). PFL steht hierbei für Pre-Fader Listener also dem Vorhören des nächsten Songs.

Beim Überblenden der Songs, also vom einen auf das andere Deck, sollten beide Songs dieselbe Geschwindigkeit haben und synchron verlaufen. Beim betätigen des Sync-Button (64/71) werden beide Geschwindigkeiten anhand der BPM (Beats-Per-Minute, Schläge-Pro-Minute) angeglichen und zeitlich versetzt, so dass die Beats beider Songs zeitgleich verlaufen. Das Überblenden der Songs ist nun hörbar dynamischer; der Wechsel zwischen den Songs unauffälliger.

Nach dem Überblenden kann die Geschwindigkeit mit dem Pitch-Fader (11/12) auf die normale BPM-Zahl des Songs zurückgesetzt werden. Durch das anpassen des Pitch-Faders ändert sich neben der Geschwindigkeit auch die Tonlage, wodurch der Gesang beim Verlangsamen tiefer wird und beim Verschnellern der so genannte Chipmunk-Effekt der Stimmen entsteht. Um dies zu verhindern, kann man den Keylock-Button (56/63) aktiviert. Er bewirkt, dass die Tonhöhe auf dem Selben Niveau festgesetzt wird, auch während die Geschwindigkeit angepasst wird.

Neben den einzelbedienbaren Elementen der Oberfläche gibt es drei Sektionen für Effekte, Sampler und Loops. Die Loop-Sektion bildet eine Gruppe aus vier Buttons, wobei die anderen beiden Sektionen je zwei Buttons und zwei Drehregler besitzen.

Mit der Effekt-Sektion lassen sich Sound-Effekte auf den aktuellen Song des Decks anwenden. Mit dem Select-Drehregler (0/4) blättert man durch die Auswahl an Effekten, wie zum Beispiel Hall-, Echo- oder Flanger-Effekte. Der Control-Regler (1/5) steuert die Intensität des aktuellen Effektes. Der Parameter-Button (50/54) ermöglicht es, für den Fall das mehrere Intensitäten eines Effektes eingesellt werden können, zwischen eben diesen Parametern zu wechseln. Der On/Off-Button (49/53) aktiviert, beziehungsweise deaktiviert letztendlichen den eingestellten Effekt.

Ähnlich aufgebaut ist die Sampler-Sektion. Anstelle von Effekten wird mit dem Select-Regler (2/6) durch Samples navigiert. Samples sind kurze Audio-Sequenzen die neben dem aktuell laufenden Song zusätzlich eingespielt werden können ohne diesen stoppen zu müssen. Auch auf das zweite Deck muss so nicht zurückgegriffen werden. Die Samples können von Stimmen und Gesang bis zu einzelnen Instrumenten reichen, wobei keine kompletten Songs als Sample genutzt werden, da sich diese sonst mit dem eigentlich spielenden Song „beißen“. Ein Beispiel für ein Sample ist eine Stimme, die mit der Aufforderung „Make some noise!“ versucht die Stimmung im Publikum „anzuheizen“. Der Wet/Dry-Regler (3/7) kann zur Lautstärkeabstimmung des Samples verwendet werden. Möchte man aus dem aktuell laufenden Song einen Teil herausnehmen und als eigenständiges Samples verwenden, kann der Rec-Button (58/62) für die Aufnahme genutzt werden. Der Play-Button (57/61) spielt das ausgewählte Sample letztendlich solange ab, bis der Button erneut gedrückt wurde. Für Samples sind demnach auch Loops geeignet.

Die Loop-Sektion kommt ohne Drehregler aus. Loop beschreibt das ständige Wiederholen eines Teilbereiches vom aktuellen Song. Beim scheinbar normal laufenden Song markiert der gedrückte Loop-In-Button (73/77) die Anfangsposition des Loops. Davon merkt das Publikum zunächst nichts, bis der DJ auch den Loop-Out-Button (74/78) betätigt und somit die Endposition des Loops festlegt. Der Song spielt nicht wie gewohnt weiter sondern springt immer wieder von der End- zur Anfangsposition. Erst ein weiteres bedienen des Loop-Out-Buttons beendet den Loop und der Song läuft normal weiter. Zudem ist es möglich den Loop-Bereich mit den Buttons Loop-« (65/69) und Loop-» (66/70) zu verkleinern, beziehungsweise zu vergrößern. Der Loop-«-Button halbiert den Loop-Bereich und der Loop-»-Button verdoppelt ihn ausgehend von der Anfangsmarkierung.

Das Herzstück jedes Decks bildet das so genannte Jog Wheel (25/24), welches frei übersetzt das Stoß- oder Schups-Rad bezeichnet. Mit dem Jog Wheel kann der DJ mit der bekannten Scratch-Bewegung der Hand die Position des aktuellen Songs ruckartig vor- und zurückstoßen. Dabei entsteht der so genannte Scratch-Effekt, obwohl bei den meisten Scratch-Varianten auch der Crossfader zum Einsatz, welcher

noch erläutert wird. Ist das Deck gerade nicht für das Publikum zu hören, wird das Jog Wheel auch alternativ zur Maus benutzt um innerhalb des Songs zu einer bestimmten Stelle vorzuspulen. Eine Besonderheit der Jog Wheels im Vergleich zu den Drehreglern ist zudem, dass diese nicht die aktuelle Ausrichtung an den Laptop übermitteln, sondern die Geschwindigkeit mit der sie sich nach rechts oder links drehen.

Direkt unterhalb vom Jog Wheel befindet sich eine Gruppe von drei Buttons. Der Cue-Button (51/60) setzt dem Anfangsbeat für den Song fest. Das ist von großer Bedeutung, da vor und nach vielen Songs noch Stille ist oder der Song erst aus der Stille eingeblendet wird. Das setzten des Cue stellt sicher, dass beim Überblenden vom einen zum nächsten Song wirklich vom richtigen Zeitpunkt aus gestartet wird.

Das einmalige drücken des Pause-Buttons (59/68) stoppt den Song und die aktuelle Position wird auf die ursprüngliche Cue-Position zurückgesetzt. Ein wiederholtes Drücken setzt die Position auf die absolute Anfangsposition des Songs zurück.

Der Play/Stutter-Button (67/76) startet die Wiedergabe. Beim wiederholten Drücken während der Wiedergabe, springt die Position an die Stelle zurück, wo zuletzt der Play-Button betätigt wurde. Durch ständiges Drücken des Buttons entsteht somit ein Stutter-, beziehungsweise Stotter-Effekt.

Auch ein Load-Button (75/52) ist auf jedem Deck vorhanden. Ist im Datei-Browser der Software ein Song markiert und der Load-Button wird betätigt, so wird dieser Song in das Deck geladen. Befindet sich in diesem Deck bereits ein Song, welcher zudem gerade wiedergegeben wird, erhält der DJ eine Warnung die ihn darüber informiert. Ein weiteres Betätigen des Load-Buttons bewirkt das Stoppen des aktuellen Songs und das Laden des markierten Songs.

Neben den bereits angesprochenen Pitch-Fadern, besitzt jedes Deck einen Volume-Fader (8/9), der den Lautstärke-Pegel für das jeweilige Deck bestimmt.

3.2.2. Globale Buttons und Regler

Zusätzlich zu den Steuerkomponenten welche doppelt, also für jedes Deck einmal vorkommen, gibt es weitere Bedienelemente welche sich Global, also auf beide Decks gleichzeitig auswirken.

Zu den wichtigsten zählt der bereits angesprochene Crossfader (10). Er befindet sich, wie alle globalen Steuerelemente in der Mitte zwischen beiden Decks. Der Crossfader wird zum Überblenden vom einen Deck auf das andere genutzt. Je nachdem in welcher Position sich der Fader befindet, ist für den Hörer das linke oder rechte Deck zu hören. Dies geschieht Stufenlos, so dass beide Decks mit halber Lautstärke zu hören sind, wenn sich der Crossfader in der Mitte befindet. Je mehr er in eine Richtung bewegt wird, desto mehr wird dieses Deck ein- und das andere ausgeblendet.

Eine besondere Bedeutung erlangt der Crossfader bei den verschiedenen Scratch-Methoden die von DJs angewendet werden. Anders als Viele denken wird nicht der Song der gerade gespielt wird gescratcht, sondern ein zusätzlicher Song, welcher sich im anderen Deck befindet. Durch das hin- und herschieben des Crossfaders ist der Scratch-Effekt mal mehr und mal weniger zu hören. Durch verschiedene Kombinationen und Techniken unterscheidet man hauptsächlich zwischen 6 Varianten des Scratchens in denen der Crossfader mit einbezogen ist.

Der Master-Drehregler (23) steuert die Gesamtlautstärke, also die globale Lautstärke beider Decks.

Der PH-Vol-Regler (15) steuert die Kopfhörerlautstärke, sofern mehrere Soundkarten verwendet werden. Ansonsten sind Master und Kopfhörerlautstärke identisch.

Der PH-Mix-Regler (22) steuert die Balance zwischen der Master und der Kopfhörerlautstärke, sofern mehrere Soundkarten verwendet werden. Ansonsten sind Master und Kopfhörerlautstärke identisch.

Der Scratch-Button (72) ermöglicht den Wechsel zwischen dem Scratch- und Pitch-Modus der Jog Wheels. Im Scratch-Modus lässt sich der aktuelle Song wie gewohnt scratchen. Im Pitch-Modus bewirkt das schubsen des Jog Wheels ein kurzfristiges hoch-, bzw. herunterpitchen des Songs, welches sich beim Stehenbleiben wieder normalisiert.

Im Zentrum der Bedienoberfläche sitzt der Track-Drehregler (79), welcher wie auch die Equalizer-Drehregler gleichzeitig als Button (26) fungiert. Dieser Drehregler wirkt sich als Einziger nicht auf die Soundwiedergabe aus, sondern lässt den DJ durch die Ordner-Strukturen auf dem Laptop blättern. Durch Drücken des Buttons öffnet, beziehungsweise schließt sich ein Ordner im Datei-Browser der Software. Sind in einem Ordner Musikdateien enthalten, kann der bereits angesprochene Load-Button von einem Deck genutzt werden, den markierten Song in dieses Deck zu laden.

4. Das MIDI-Protokoll

Die Abkürzung MIDI steht für „Musical Instrument Digital Interface“ und stellt eine digitale Schnittstelle für elektronische Musikinstrumente dar. Das MIDI-Protokoll dient der Datenübertragung zwischen elektronischen Instrumenten wie zum Beispiel Keyboards aber auch bei PCs und Laptops kommt es zum Einsatz.

4.1. Anschlüsse

Als MIDI im Jahr 1983 als Standard eingeführt wurde, gab es drei verschiedene Anschlüsse zur Verwendung solcher Geräte, umgangssprachlich MIDI-Trio genannt. MIDI-Trio wird aus den Anschlüssen MIDI-

In, MIDI-Out und MIDI-Thru gebildet. MIDI-In ist für den Empfang und MIDI-Out für das Senden des Signals verantwortlich. MIDI-Thru wird zum „Durchschleifen“ der Daten verwendet. In der Regel sind die Anschlüsse als fünfpolige DIN-Buchsen realisiert.

Die bei jedem MIDI-Befehl mitgelieferte Kanalnummer, welche 4 Bit (0b1111 = 15) groß ist, ermöglicht das Ansteuern von 16 (0 – 15) verschiedenen Kanälen mit einem Anschluss. Je nach Gerät und Verwendungszweck kann es zu Engpässen führen, wenn mehr Kanäle benötigt werden. Durch die immer gängigere Verwendung von USB-Geräten wurden schon bald MIDI-fähige Musikgeräte um USB- aber auch FireWire-Schnittstellen erweitert. Diese konnten nun mit Hilfe neuer USB-MIDI-Treiber mehrere virtuelle Verbindungen zur selben Zeit anlegen, womit das Problem begrenzter Kanäle wegfiel. Jede virtuelle Verbindung emuliert ein reales MIDI-Kabel; je virtueller Verbindung stehen also wiederum 16 Kanäle zur Verfügung.

4.2. USB-MIDI Event Pakete

Im Folgenden gehe ich gezielt auf das USB-MIDI-Protokoll ein, da dieses beim NTC via USB-Anschluss zum Tragen kommt. Bei jedem Vorgang (Event) auf dem NTC, welcher über das USB-MIDI-Protokoll dem PC mitgeteilt werden soll, wird ein USB-MIDI Event Paket zum Computer gesendet. Jedes Event Paket hat eine feste Länge von 32 Bit, also 4 Byte.

Das erste Byte bildet den Paket Header, welcher die Kabelnummer (4 Bits) und die Code-Index-Nummer (4 Bits) enthält. Anhand der Kabelnummer können 16 unterschiedliche virtuelle Kabelverbindungen verwaltet werden, was in einer Gesamtzahl von 256 möglichen MIDI-Kanälen resultiert. Die Code-Index-Nummer (CIN) klassifiziert das MIDI-Event, beschreibt also welcher Art die aktuelle MIDI-Nachricht ist.

Information 1: Code Index Nummer Klassifikation

CIN	MIDI_x Größe	Beschreibung
…	…	…
0x9	3 Byte	NoteOn-Nachricht
0xA	3 Byte	PolyKeyPress-Nachricht
0XB	3 Byte	ControlChange-Nachricht
0xC	2 Byte	ProgrammChange-Nachricht
0xD	2 Byte	ChannelPressure-Nachricht
0xE	3 Byte	PitchBendChange-Nachricht
0xF	1 Byte	Single Byte

Aus dem Tabellen-Auszug lässt sich ablesen, dass nicht jede Nachricht die vollen letzten 3 Byte des Event Paketes benötigen. Für diesen Fall werden ungenutzte Bytes mit Nullen aufgefüllt, um die geforderte Länge von 32 Bit pro Paket zu erhalten.

4.2.1. Beispiele für USB-MIDI Event Pakete

9n kk vv ist ein Beispiel einer MIDI-Version 1.0 Übertragung. Die 9 weist auf eine Note-On Nachricht hin, n steht für den Kanal, kk bezeichnet die Key-Nummer und vv die Velocity. Auf den NTC bezogen teilt dies mit, dass der Button mit der Nummer kk auf den Wert vv gesetzt wurde. Via USB-MIDI auf dem virtuellen Kabel 1 übertragen ergibt sich als Kabelnummer 0x1 und als CIN 0x9; wodurch das Event Paket folgendermaßen aussieht: 19 9n kk vv

Eine vergleichbare ControlChange-Nachricht auf dem virtuellen Kabel 10 wäre: AB Bn pp vv Die Kabelnummer 10 steht hexadezimal für 0xA, CIN ist wegen der Art der ControlChange-Nachricht 0xB, n steht wiederum für einen der 16 Kanäle innerhalb des virtuellen Kabels, pp ist spezifiziert als Programmnummer, ist im Grunde aber das Selbe wie der kk Wert der Note-On Nachricht und vv ist der Velocity-Wert.

Zusammengefasst ist jedes Event Paket folgendermaßen aufgebaut, wobei sich die Bedeutung der Bits und Bytes hier auf die Nutzung mit dem NTC beziehen:

Information 2: Zusammensetzung eines USB-MIDI Event Paketes

Byte 0		Byte 1		Byte 2	Byte 3
Kabel Nummer	Code Index Nummer	Art der Nachr./ Status	Kanal Nummer	Komponenten-Nummer welche das Event ausgelöst hat	Der neue Wert auf den die Komponente gesetzt wurde

4.2.2. MIDI-Nachrichten und deren Werte

ControlChange-Nachrichten werden im Falle des NTC immer für Vorgänge genutzt wo ein ganzer Bereich an Werten gesendet werden kann. Mit anderen Worten bei allem außer Buttons, welche nur 0 (nicht gedrückt) oder 127 (gedrückt) als Note-On Nachricht senden. Bei Drehreglern, Fadern und auch den beiden Jog Wheels werden ControlChange-Nachrichten gesendet, da auch alle Werte zwischen 0 und 127 möglich sind.

Zu Beachten ist auch, welche Komponente auf dem NTC das Event ausgelöst hat. Bei Komponenten mit einem min- und maximalen Anschlag bezieht sich der Verlockt-Wert der ControlChange-Nachricht auf die aktuelle Position. Ist ein Drehregler beim maximalen Anschlag angelangt sendet er den Wert 127, also 100%. Beim minimalen Anschlag beträgt der Wert logischerweise 0 und bei der Hälfte 64, für 50%.

Komponenten wie die beiden Jog Wheels sind können frei rotieren, besitzen also keinen Start- und Endpunkt. Das Übertragen der aktuellen Position ist daher nicht möglich. Anstelle dessen wird die aktuelle Drehgeschwindigkeit übertragen. Eine Drehung nach links sendet Werte im Intervall [0…63], eine Rechtsdrehung hingegen Werte im Intervall [64…127]. Ist die Drehgeschwindigkeit gering werden Werte

im Bereich um 0, respektive 127 gesendet. Je größer die Geschwindigkeit wird, umso weiter reichen die Werte an 63, respektive 64 heran.

5. Die PulseMIDI-Klassenbibliothek

Nachdem nun bekannt ist in welcher Art und Weise das MIDI-Protokoll Daten überträgt, kann die Verbindung zwischen der NTC Hardware und dem von mir entwickelten Mediaplayer realisiert werden. Den Mediaplayer habe ich unter dem Namen „Pulse“ entwickelt. Aufgrund der nun hinzukommenden DJ-Variante erhält der Mediaplayer den Zusatz MP und die DJ-Version MJ (für MediaJockey). Da für die Dokumentation lediglich die DJ-Edition von Relevanz ist, spreche ich weiterhin ab hier von der Pulse Software. Die Klasse zur MIDI-Kommunikation wird im Folgenden als PulseMIDI beschrieben.

Pulse ist in der Programmiersprache C-Sharp geschrieben um die Vorzüge des .Net Frameworks, einer Sammlung von Klassenbibliotheken, ausnutzen zu können. Laut Microsoft sind in der aktuellen Version 3.5 um die 11.000 Klassen einsatzbereit vorhanden, um Entwicklern die Arbeit bei der Realisierung komplexer Anwendungen zu erleichtern. Für die Unterstützung des MIDI-Protokolls ist derzeit noch keine Klasse vorhanden, welche für die geplante Modifizierung von Pulse verwendet werden kann.

Nach weiteren Recherchen bin ich im Microsoft Developer Network (MSDN) auf die Win32-API gestoßen, welche MIDI in der Windows Multimedia Klassenbibliothek unterstütz. Die API besteht aus dem C-Header "MMSystem.h" und der angesprochenen Klassenbibliothek winmm.dll. Da die Klasse im MSDN ausführlich beschrieben ist habe ich mich dazu entschlossen die Win32-API-Funktionalität in einer eigenen PulseMIDI-Klassenbibliothek zu verpacken und speziell für die Verbindung vom NTC zum Computer zu optimieren. Ist die Klasse implementiert und kompiliert erhält man die Datei PulseMIDI.dll, welche das letztendliche Verbindungsstück zwischen Soft- und Hardware bildet.

5.1. PulseMIDI.Win32Wrapper

Damit PulseMIDI zunächst die MIDI-Funktionalität erhält, müssen alle MIDI-relevanten Funktionen mittels P/Invoke importiert werden. Wie genau der Plattformaufrufdienst P/Invoke funktioniert ist ebenfalls im MSDN ausführlich beschreiben. Im Grunde werden die benötigten Funktionen importiert und in einer Hüllenklasse (Wrapper-Klasse) verpackt, um den Umgang mit diesen Funktionen zu ermöglichen und zu erleichtern. Die MIDI Referenz in der MSDN-Dokumentation verschafft einen Überblick über die zur Verfügung stehenden Funktionen und deren Nutzung.

Importiert habe ich die Funktionen zum öffnen und schließen eines MIDI-Gerätes, sowie zum Senden via Ausgabegerät und Empfangen via Eingabegerät. Des Weiteren Funktionen zum ermitteln der Anzahl der

am Computer angeschlossenen MIDI-Geräte, der Abfrage der Geräteeigenschaften und der Umwandlung eines Fehlercodes in die entsprechende Fehlermeldung. Benötige Konstanten wie unter anderem die Fehlercodes wurden dem C-Header "MMSystem.h" entnommen und der Wrapper-Klasse beigefügt. Neben dem Handle für das Gerät und einer CAPS Struktur welche die Funktionalität des Gerätes beschreibt, die zuvor ermittelt wurde, existiert noch ein so genanntes Callback Delegate.

Delegaten sind im Prinzip Zeiger auf Funktionen. Mit dem Callback Delegate wird die Funktion deklariert, welche bei einem bestimmten Event „zurückgerufen“ werden soll. Es dient also der Kommunikation zwischen dem Betriebssystem und der Anwendung.

5.1.1. Beispiel für Plattformaufrufdienst P/Invoke

Code 1: Import einer externen Funktion aus einer DLL-Datei

```
[DllImport("winmm.dll", SetLastError = true)]
public static extern UInt32 midiOutGetNumDevs();
```

Dieses Beispiel importiert die Funktion `midiOutGetNumDevs()`, zum ermitteln der Anzahl von verfügbaren MIDI-Ausgabegeräten. Importiert wird aus der winmm.dll, als public-statische-externe Methode. Die Funktion liefert eine vorzeichenlose 32-Bit-Ganzzahl zurück. Der auf true gesetzte Bool-Wert bewirkt das Ablegen eines Codes im Falle eines Fehlers. Mittels `GetLastWin32Error()` kann dieser Fehler abgefragt werden.

5.1.2. Nicht-Typsichere Funktionen

Bei der Implementierung der Wrapper-Klasse traten keine Fehler auf. Der Import der Funktionen zur weiteren Nutzung war demnach erfolgreich umgesetzt. Jedoch wies der Compiler mittels Warnungen auf sechs Funktionen hin, die nicht typsicher seien, was im späteren Verlauf bei der Verwendung zu Problemen hätte führen können. Zu den Funktionen gehören je für Aus- und Eingabegerät die Funktion zum Öffnen der Verbindung, zum Ermitteln der Geräteeigenschaften und das Ermitteln des Fehlertextes.

Code 2: Bereitstellen eines typsicheren Methodenaufrufes

```
[DllImport("winmm.dll", SetLastError = true)]
private static extern UInt32 midiInGetErrorText(…);

public static UInt32 midiInGetErrorText(…)
{
   return midiInGetErrorText(…);
}
```

Eine kleine Anpassung des DLL-Imports genügt um das Fehler-Risiko und somit die Warnung zu beheben. Die importierte Funktion wird lediglich als private deklariert. So ist sichergestellt, dass nur die Wrapper-Klasse selbst diese Funktionen aufrufen können. Um die Funktion außerhalb nutzbar zu ma-

chen, wird eine gleichnamige public-Variante bereitgestellt, welche einfach das Ergebnis der Nicht-Typsicheren private-Methode zurückgibt. Die public-Methode ist dadurch typsicher.

5.1.3. NullReferenceException und CallbackOnCollectedDelegate

Fehlermeldung 1: NullReferenceException

HIN-WEIS

```
System.NullReferenceException wurde nicht behandelt.
Der Objektverweis wurde nicht auf eine Objektinstanz festgelegt.
```

Diese Fehlermeldung ist im späteren Verlauf aufgetreten, als die PulseMIDI-Klasse fertig ausformuliert und bereit zum Testen war. Da die Lösung Änderungen an der Wrapper-Klasse erfordert, erläutere ich dieses Problem bereits hier. Mithilfe der gesamten PulseMIDI-Klasse konnten bereits die LEDs auf dem NTC angesteuert werden und in der Testanwendung auf dem Computer-Monitor konnte richtig dargestellt werden welcher Regler und Buttons wie betätigt wurde. Einzig der Versuch beide Jog Wheels zeitgleich mit möglichst viel Geschwindigkeit rotieren zu lassen führte zum Absturz der Anwendung mit obiger Fehlermeldung.

Bedeuten tut sie, dass zum Beispiel eine Variable welche gerade verwendet werden sollte, zwar angelegt aber nicht auf einen bestimmten Wert gesetzt, also null sei. Der Compiler ermöglicht es, direkt nach einer solchen Ausnahme normalerweise das Einsehen der Werte, um gegebenenfalls die Variable zu finden, deren Wert null gerade null ist. Diese war in diesem Fall nicht zu finden. Weshalb wurde mir dann später klar. Mittels eines Try-Catch-Blockes im SourceCode kann eine solche Ausnahme normalerweise abgefangen werden, sodass die Applikation nicht abstürzt. Auch dies funktionierte nicht, was mich auf den Gedanken brachte, dass der Fehler in der DLL zustande kommt, was fatal die gerade fertiggestellte PulseMIDI-Klasse gewesen wäre. Dieser Fehler hätte nicht behoben werden können, da lediglich der Umgang mit der DLL im MSDN beschrieben wird, dort aber nicht der SourceCode veröffentlicht ist. Damit hätte ich entweder mit der Gefahr von Programmabstürzen leben müssen, oder die PulseMIDI-Klasse von Grund auf neu über einen alternativen Weg entwerfen müssen.

Fehlermeldung 2: CallbackOnCollectedDelegate

HIN-WEIS

```
CallbackOnCollectedDelegate wurde erkannt.
Für den von der Garbage Collection gesammelten Delegaten vom Typ
"PulseMIDI!PulseMIDI.Win32Wrapper+MidiInProc::Invoke" wurde ein Rückruf durchgeführt.
Dies kann Anwendungsabstürze, Datenbeschädigung und -verlust zur Folge haben. Beim
Übergeben von Delegaten an nicht verwalteten Code müssen die Delegaten von der
verwalteten Anwendung beibehalten werden, bis sichergestellt ist, dass sie nie aufgerufen
werden.
```

Ein einziges Mal erschien unmittelbar vor der ersten Fehlermeldung, diese neue Meldung. Warum dies nicht jedes Mal geschieht konnte ich leider noch nicht in Erfahrung bringen. Diese Meldung gab vom Text her mehr Anhaltspunkte auf die Ursache des Fehlers, wodurch dieser doch noch recht schnell

behoben werden konnte. Zum einen wird auf einen Delegaten und zum anderen auf den Garbage Collector hingewiesen. Der Garbage Collector (GC), welcher unter anderem in C-Sharp und Java existiert, erleichtert dem Entwickler den Umgang mit Objekten welche mit dem Schlüsselwort new während der Laufzeit angelegt wurden. Diese Objekte müssen zum Beispiel unter C++ auch wieder vom Entwickler mit delete aus dem Speicher gelöscht werden, um diesen Speicherplatz wieder für andere Programme verfügbar zu machen. Eben das tut der GC sobald kein Verweis mehr auf ein bestimmtes Objekt existiert. Der Entwickler muss das Objekt nicht mehr per Hand löschen.

Jetzt wurde auch klar auf was sich die NullReferenceException bezogen hat. Nicht wie angenommen auf eine Variable, sondern auf einen Delegaten, wovon allein der Delegate auf die Methode betroffen sein konnte, welche zum Analysieren der eingehenden MIDI-Daten gedacht war.

Code 3: Angepasste Funktion zum Öffnen einer MIDI-Verbindung

```
public static UInt32 midiInOpen(…)
{
   GC.KeepAlive(dwCallback);
   GC.KeepAlive(dwCallbackInstance);

   return midiInOpen(…, dwCallback, dwCallbackInstance, …);
}
```

Die Lösung des Problems bestand also darin, dem GC mittels `KeepAlive()` zu sagen, dass der Callback Delegate und dessen Instanz nicht automatisch gelöscht werden sollen. Geschehen tut dies immer beim Anlegen, während eine neue Verbindung zu einem MIDI-Eingabegerät geöffnet wird.

5.2. PulseMIDI.Win32Utilities

Nachdem die Funktionalität der Win32-API nun importiert und einsatzbereit ist habe ich entschlossen in einer separaten Klasse Win32Utilities nützliche Funktionen bereit zu stellen. Diese betreffen das Unterscheiden, Senden und Dekodieren (Analysieren) von MIDI-Nachrichten sowie weitere Konvertierungen.

5.2.1. Senden einer MIDI-Nachricht

Code 4: Funktion zum Senden einer NoteOn-Nachricht

```
public static UInt32 sendNoteOnMessage(Win32Wrapper.HMIDIOUT hmo, int channel, int note, int velocity)
{
   if (channel < 0 || channel > 15)
      throw new ArgumentOutOfRangeException("Kanal ist außerhalb der Reichweite.");

   if (note < 0 || note > 127)
      throw new ArgumentOutOfRangeException("Note ist außerhalb der Reichweite.");

   if (velocity < 0 || velocity > 127)
      throw new ArgumentOutOfRangeException("Geschwindigkeit ist außerhalb der Reichweite.");

   return Win32Wrapper.midiOutShortMsg(hmo,(UInt32)(0x90 | channel | (note << 8) | (velocity << 16)));
}
```

Beim Senden einer MIDI-Nachricht, wie zum Beispiel einer NoteOn-Nachricht, werden wie bereits

beschrieben der Kanal, die Note und der Wert übertragen. hmo ist das Handle auf das MIDI-Gerät, welches beim Öffnen der Verbindung angelegt wurde. Die Werte für channel, note und velocity sind je nach Verwendungszweck belegt. Vor dem Senden wird überprüft ob es sich um eine gültige NoteOn-Nachricht handelt, das heißt der channel zwischen 0 und 15 und note sowie velocity zwischen 0 und 127 liegt. Ansonsten wird eine Ausnahme ausgelöst, da diese Bereiche überschritten wurden. Sind die Werte gültig wird die Nachricht mittels der zuvor importierten Win32-API Funktion gesendet. Ein erneuter Blick in das MSDN verrät, wie die Werte Bitweise zusammenzufügen sind.

Beispiel 1: Das bitweise Zusammenfügen eines Beispiel-Datensatzes

```
DATENSATZ:
status:     0x90   =   0d144   =   0b1001 0000
channel:    0x01   =   0d001   =   0b0000 0001
note:       0x2F   =   0d047   =   0b0010 1111
velocity:   0x7E   =   0d126   =   0b0111 1110

0x90 | 0x01                                 =>                    1001 0001
0x90 | 0x01 | note << 8                     =>            0010 1111 1001 0001
0x90 | 0x01 | note << 8 | velocity << 16     =>   0111 1110 0010 1111 1001 0001
```

Das letzte, respektive erste Byte bleibt ungenutzt. Daher ist es wichtig die Daten via UInt32 zu senden, um die geforderte Größe von 32-bit zu erhalten. Das gezeigte Beispiel überträgt eine NoteOn-Nachricht, wofür der Status auf 0x90 zu setzen ist. Gesendet wird an den Kanal 1, die Note 47 mit dem Wert von 126. Die Daten sind je in hexadezimaler, dezimaler und binärer Schreibweise gegeben, sowie Schrittweise deren Zusammenführung erläutert. Zu beachten ist, dass die Nachricht in umgekehrter, also Low-Byte-Reihenfolge zusammengesetzt werden muss. Die folgenden Werte werden in den höherwertigen Bytes erwartet. Realisiert wird dies mit einer Linksverschiebung bei note-Werten um 8 Bit und bei velocity-Werten um 16 Bit. Der <<-Operator kennzeichnet die Linksverschiebung und der |-Operator die OR-Verknüpfung.

5.2.2. Unterscheiden von MIDI-Nachrichten

Code 5: Funktion zum Prüfen auf eine gültige NoteOn-Nachricht

```
public static bool IsNoteOnMessage(…)
{
   return ((int)dwParam1 & 0xF0) == 0x90;
}
```

Beispiel 1: Das bitweise Herausfiltern des Status-Byte einer MIDI-Nachricht

```
dwParam1:  0x91 =   0d145   =   0b1001 0001
2. Wert:   0xF0 =   0d240   =   0b1111 0000

   1001 0001
&  1111 0000
=  1001 0000
```

Die gezeigte Funktion `IsNoteOnMessage()` erhält die Daten einer MIDI-Nachricht. Sie gibt einen Bool-Wert zurück; true wenn es sich um eine NoteOn-Nachricht handelt, andernfalls false. Da das erste Byte

sowohl die Art der Nachricht als auch die Kanalnummer enthält muss wieder Bitweise maskiert werden. Indem man mit dem hexadezimalen Wert für 240 AND-Verknüpft, filtert man die erwünschten Informationen heraus. Entspricht das Ergebnis 0x90 handelt es sich um eine NoteOn-Nachricht, bei 0xB0 wäre es eine ControlChange-Nachricht. Mit der Funktion können später komfortabel eingehende Nachrichten unterschieden werden.

5.2.3. Dekodieren einer MIDI-Nachricht

Code 6: Funktion zum Dekodieren einer NoteOn-Nachricht

```
public static void DecodeNoteMessage(out channel, out note, out velocity, out timestamp)
{
   if (!IsNoteOnMessage(dwParam1, dwParam2))
      throw new ArgumentException("Keine Note Nachricht.");

   channel = (Byte)((int)dwParam1 & 0x0f);
   note = (Byte)(((int)dwParam1 & 0xff00) >> 8);
   velocity = (Byte)(((int)dwParam1 & 0xff0000) >> 16);
   timestamp = (UInt32)dwParam2;
}
```

Ist die Art einer eingehenden MIDI-Nachricht erkannt, können mithilfe der entsprechenden `Decode…Message()` Funktionen deren Werte herausgelesen werden. Wie zuvor beschrieben kommt hier ebenfalls die Bitmaskierung zum Einsatz. Um die Kanalnummer zu ermitteln geht man exakt wie in der Unterscheidung vor, Verknüpft jedoch mit 0x0F um die anderen 4 Bit des ersten Byte herauszufiltern. Für die note- und velocity-Werte werden ebenso nur die betroffenen Bits herausgefiltert wobei die Rechtsverschiebung zu beachten ist, da die Werte zuvor wie angesprochen linksverschoben wurden. Der zweite Parameter dwParam2 dient als Zeitstempel. Falls nötig weiß die Software somit wann die Nachricht eingegangen ist. Die Zeit-Einheit sind Millisekunden, beginnend bei Null seit der geöffneten Verbindung zu dem MIDI-Eingabegerät.

Bei dem Zeitstempel handelt es sich deshalb um den Datentyp UInt32, weil der Parameter dwParam2 vom Datentyp DWORD war. Es gilt: 1 QWORD = 2 DWORD = 4 WORD = 8 Byte = 64 Bits Dementsprechend sind 1 DWORD = 32 Bit.

Weiterhin ist zu beachten, dass eine Funktion bekanntlich nur einen Rückgabewert haben kann. Die Dekodier-Methode muss jedoch vier Werte zurückgeben. Um dies zu realisieren gibt es mehrere Wege. Man könnte eine Struktur definieren, welche alle vier Werte beinhaltet und würde am Ende der Funktion ein Objekt dieser Struktur zurückgeben. Des Weiteren gibt es die Möglichkeit mit Zeigern zu arbeiten. Soll heißen man übergibt vier Zeiger auf bekannte Stellen im Speicher, wo die vier Ergebnisse abgelegt werden sollen. So weiß die Software nach der Beendigung der Funktion, wo es die Resultate finden kann. Ich habe mich hier für einen Weg entschieden, der in dieser Form noch nicht Bestandteil des DV-Unterrichtes war. Im Prinzip ist es eine Vereinfachung der Zeiger-Variante. Mittels des Schlüsselwortes `out` vor einem Parameter wird in der aufgerufenen Funktion auf dasselbe Objekt

gearbeitet welches von der aufrufenden Funktion übergeben wurde. Wichtig ist dabei noch, dass `out` sowohl im Funktionskopf wie auch im Funktionsaufruf enthalten sein muss.

5.2.4. Konvertierungen

Code 7: Lesbare Strings für die MIDICAPS_* Konstanten

```
public static Dictionary<UInt32, string> extraFeatureNames = new Dictionary<…,…>()
{
   {Win32Wrapper.MIDICAPS_VOLUME,    "Lautstärkeregelung"},
   {Win32Wrapper.MIDICAPS_LRVOLUME, "Getrennte L / R Lautstärkeregler"},
   {Win32Wrapper.MIDICAPS_CACHE,     "Patch-Caching"},
   {Win32Wrapper.MIDICAPS_STREAM,    "Direct stream support"}
};
```

Da ich zu Beginn nicht genau einschätzen konnte inwiefern ich bestimmte Konstanten verwenden werde, habe ich wie im gezeigten Beispiel Dictionarys mit entsprechend lesbaren Strings für manche Konstanten erstellt. Dies war für mich unter anderem bei den Geräteeigenschaften sinnvoll. Dictionarys kann man sich vom Aufbau durchaus wie Wörterbücher vorstellen. Man definiert eine Reihe von sogenannten Keys, welche je einen Wert haben. Der Key, also Schlüssel, ist hierbei die Konstante und der zugeordnete Wert ist ein String mit der Beschreibung.

5.3. PulseMIDI.Nachrichten

In der Nachrichten-Klasse der PulseMIDI-Klassenbibliothek sind alle unterschiedlichen Nachrichtenarten deklariert. Die abstrakten Klassen Nachricht, GeräteNachricht und KanalNachricht werden je von der vorherigen Klasse abgeleitet.

Nachricht bildet also die Basisklasse für alle Nachrichten. Das Attribut BeatTime speichert den bereits angesprochenen Zeitstempel, den jede Nachricht enthält.

Die GerätNachricht-Klasse gilt für alle Nachrichten, welche sich auf ein spezielles Gerät beziehen. Sie enthält somit ein Objekt vom Typ Gerät, welcher noch erläutert wird. Nötig ist die Unterteilung aufgrund des „Microsoft GS Wavetable Synth“, ein Software Synthesizer im Windows-Betriebssystem welcher ebenfalls als MIDI-Ausgabegerät dienen kann. Wird dieser mittels der PulseMIDI-Klassenbibliothek als Ausgabegerät ausgewählt, erzeugt der Synthesizer wirklich einen bestimmten Ton einer Note, anstatt das beim NTC die LEDs angesteuert werden. Obwohl also vielleicht der NTC das einzige angeschlossene MIDI-Gerät ist, wird eine Auswahlmöglichkeit nötig, da ebenfalls der Software Synthesizer in den meisten Windows-Betriebssystemen vorhanden ist.

Die nächste Klasse ist die KanalNachricht, welche in einem int-Attribut die Kanalnummer speichert. Alle Nachrichten einer KanalNachricht beziehen sich also auf einen speziellen Kanal.

Die KanalNachricht wird als letztes zweimal abgeleitet, diesmal jedoch nicht abstrakt. Man erhält die letztendliche NoteOnNachricht und die ControlChangeNachricht. Andere Nachrichtentypen kommen beim NTC wie bereits erwähnt nicht zum Einsatz. Beide Nachrichten besitzen int-Attribute zum übermitteln der Komponenten-Nummer auf dem NTC und dem neuen Wert der Komponente.

5.4. PulseMIDI.Exceptions

In der Exceptions-Klasse sind gesammelt alle neu definierten Ausnahmen vorhanden. Dazu zählt die GerätException, welche ausgelöst wird, wenn ein Vorgang auf einem MIDI-Gerät nicht erfolgreich war. Neben dem Standard-Konstruktor stehen drei weitere Überladungen bereit, welche auch einen Fehlertext übermitteln können.

5.5. PulseMIDI.Gerät

Ähnlich wie die Nachricht-Klasse als Basisklasse für alle Nachrichten gilt, habe ich eine Gerät-Klasse implementiert. Diese enthält den Namen des Gerätes und wird abgeleitet zur Ausgabegerät-Klasse, beziehungsweise zur Eingabegerät-Klasse. Weitere Attribute als der Name waren zum Abgabge-Zeitpunkt der Dokumentation nicht nötig.

5.5.1. PulseMIDI.Ausgabegerät

Code 8: Private Konstruktor der Ausgabegerät-Klasse

```
private Ausgabegerät(UIntPtr deviceId, Win32Wrapper.MIDIOUTCAPS caps) : base(…)
{
   this.deviceId = deviceId;
   this.caps = caps;
   this.isOpen = false;
}
```

Wie man im gezeigten Code-Ausschnitt sieht, habe ich den Konstruktor der Klasse Ausgabegerät als private deklariert. Das hat den Grund, dass ja kein neues Gerät angelegt wird, sondern man arbeitet auf ein Gerät aus einer Liste angeschlossener Geräte. Die deviceId zeigt also auf die Position der Liste von Ausgabegeräten, in welcher sich der Eintrag mit dem Gerät befindet, dass ich verwenden möchte. In caps werden die Geräteeigenschaften gespeichert und der Bool-Wert isOpen beschreibt ob bereits eine Verbindung geöffnet wurde. Dies ist direkt beim Anlegen natürlich noch nicht der Fall.

Code 9: Funktion zum Zusammenstellen der Liste aller angeschlossenen Ausgabegeräte

```
private static Ausgabegerät[] MakeDeviceList()
{
   uint outDevs = Win32Wrapper.midiOutGetNumDevs();
   Ausgabegerät[] result = new Ausgabegerät[outDevs];
   for (uint deviceId = 0; deviceId < outDevs; deviceId++)
   {
      Win32Wrapper.MIDIOUTCAPS caps = new Win32Wrapper.MIDIOUTCAPS();
      Win32Wrapper.midiOutGetDevCaps((UIntPtr)deviceId, out caps);
      result[deviceId] = new Ausgabegerät((UIntPtr)deviceId, caps);
   }
   return result;
}
```

Die Methode MakeDeviceList gibt ein Array von Ausgabegeräten zurück. Um dieses Array zusammenzustellen wird die Anzahl an angeschlossenen Ausgabegeräten ermittelt. Dies ergibt die Größe des Arrays. In einer for-Schleife werden die Geräte durchlaufen und nacheinander in das Array hinzugefügt.

Weitere private-Methoden dieser Klasse sind Abfragen ob die Verbindung zum Gerät bereits geöffnet, beziehungsweise nicht geöffnet ist und eine Methode welche eine GerätException auslöst, falls ein Rückgabecode nicht der MIDI-Konstanten für „kein Fehler“ entspricht.

Code 10: Funktion zum Öffnen der Verbindung eines Ausgabegerätes

```
public void Open()
{
   lock (this)
   {
      CheckNotOpen();
      CheckReturnCode(Win32Wrapper.midiOutOpen(out handle, deviceId, null, (UIntPtr)0));
      isOpen = true;
   }
}
```

Auch das Ausgabegerät erhält die Methoden zum Öffnen und Schließen einer Verbindung, sowie zum Senden und Resetten der Nachrichten. Interessant ist noch der `lock`-Befehl, welcher beim Thema Threads nötig wird. Der Befehl stellt sicher, dass nicht zwei Threads zur selben Zeit auf einen problematischen Abschnitt des Codes zugreifen. Wird einer dieser Codebereiche bereits verwendet, wartet der andere Thread bis die Blockierung aufgehoben wurde, der erste Thread also fertig ist.

Code 11: Konvertierung von Control-Nummer zu Control-Name und umgekehrt

```
public static string ControlNumberToControlName(int controlNumber)
{
   return ControlNames[controlNumber].ToString();
}

public static int ControlNameToControlNumber(string controlName)
{
   for (int i = 0; i != 87; i++)
      if (ControlNames[i] == controlName)
         return i;

   return -1;
}
```

Im Bezug auf den NTC habe ich dann noch ein Array mit Control-Namen definiert. Jede Komponente, also jeder Button oder Regler auf dem NTC wird zur Unterscheidung mit einer entsprechenden Control-Nummer angesprochen, welche für ein und das selbe Control auch noch bei der Ein- und Ausgabe von einander abweichen. Daher habe ich jeder Control-Nummer einen Control-Namen gegeben, um die Handhabung zu vereinfachen und nicht jedesmal in der Liste nachzuschauen welcher Button nun das Event ausgelöst hat. Die Konvertierung geschieht mit den beiden gezeigten Funktionen. Bei der ersten

wird die Bezeichnung für eine bestimmte Control-Nummer zurückgeliefert, bei der anderen erhält man die Nummer für eine bestimmte Bezeichnung. Existiert die Bezeichnung nicht, wird -1 zurückgeliefert.

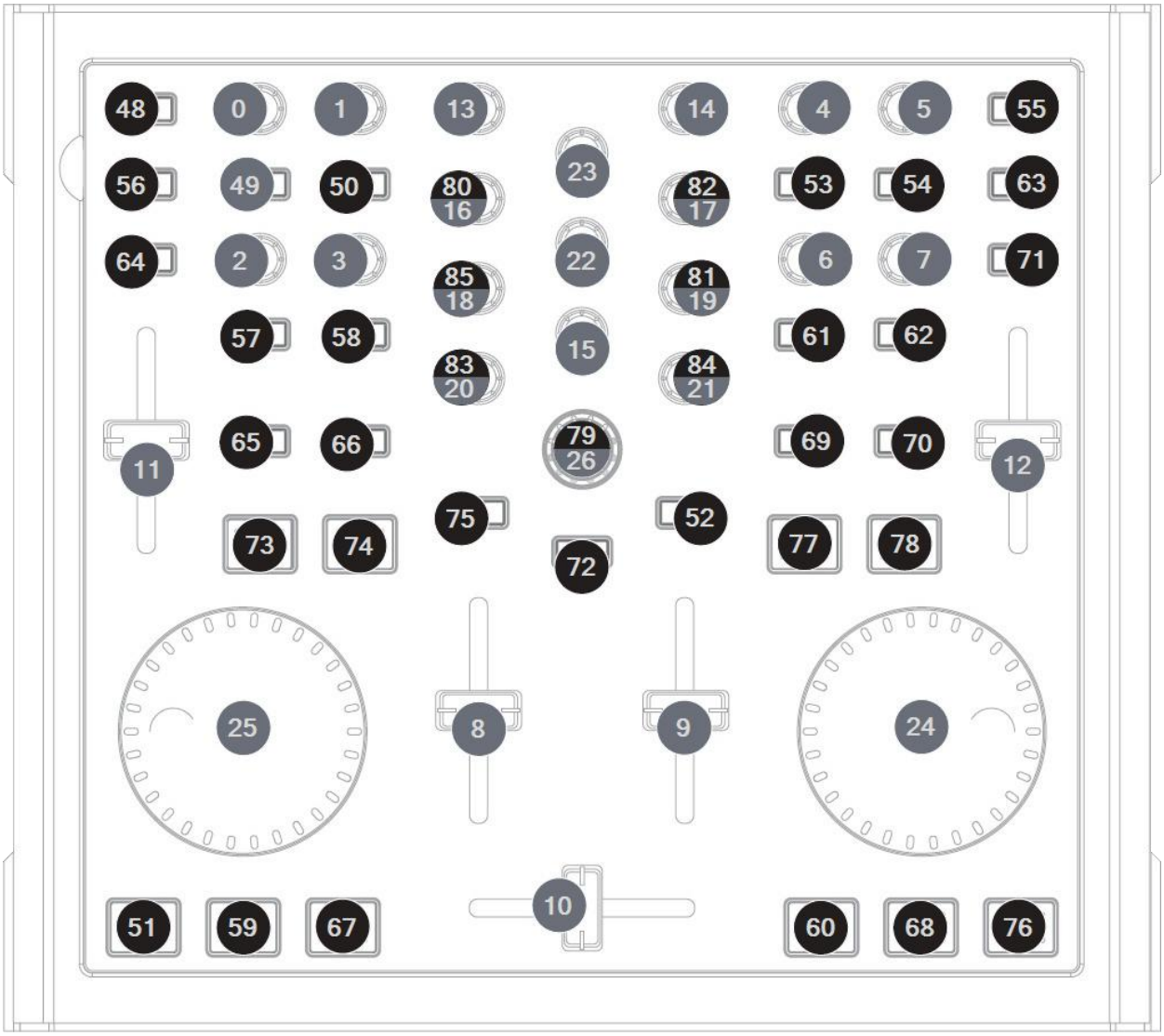
5.5.2. PulseMIDI.Eingabegerät

Die Eingabegerät-Klasse entspricht größtenteils der Ausgabegerät-Klasse. Sie muss jedoch die beiden EventHandler für die Nachrichtentypen NoteOn- und ControlChange-Nachricht bereitstellen. Zudem muss nach dem Öffnen der Verbindung explizit der Empfang von Nachrichten gestartet werden. Dies erlaubt das zwischenzeitliche stoppen des Nachrichten-Empfangs ohne die Verbindung schließen zu müssen. Ebenso wird ein TimeDelegate beim Öffnen der Verbindung gesetzt um den Zeitstempel von diesem Moment an berechnen zu können.

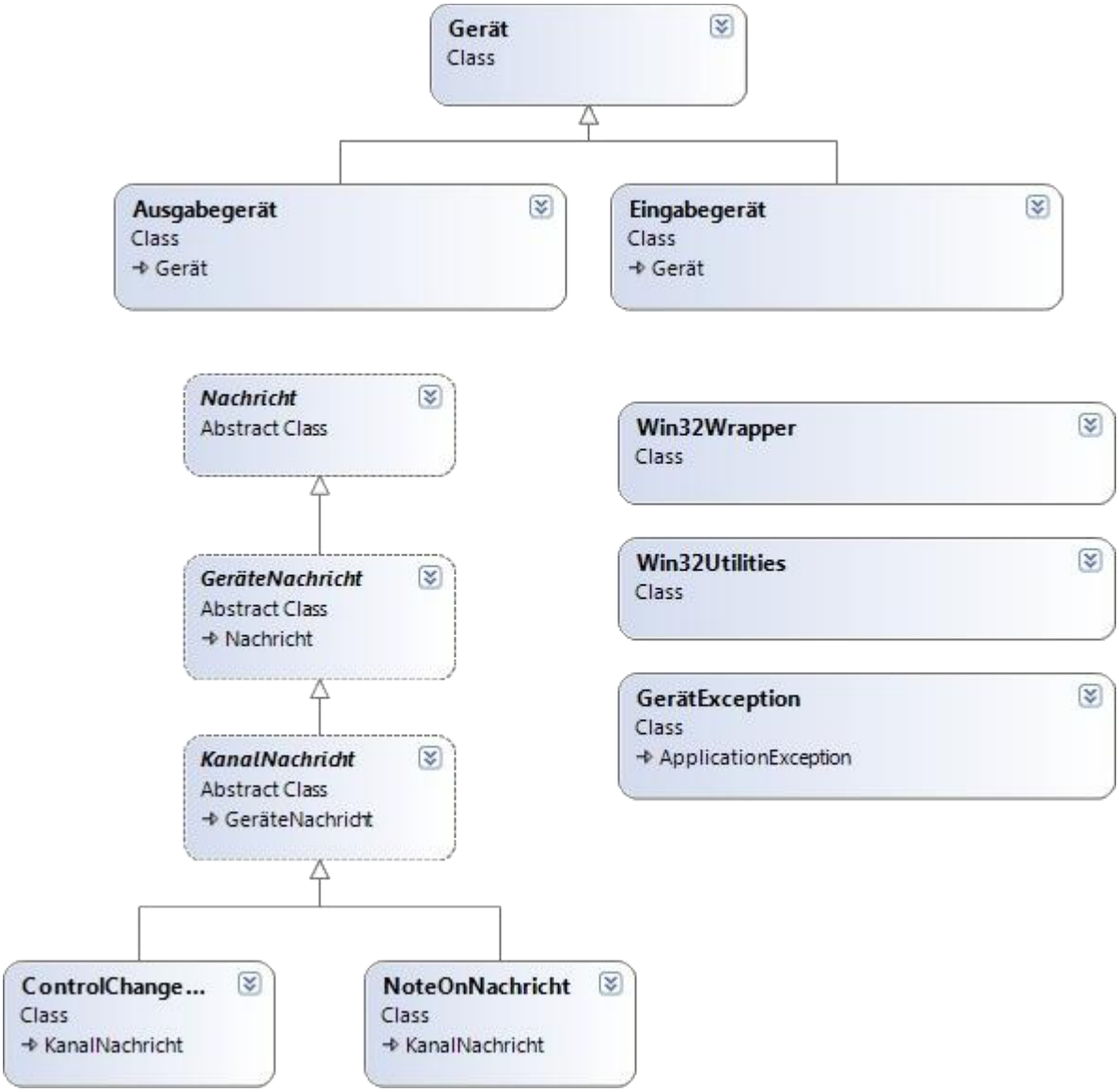
Damit ist die PulseMIDI-Klassenbibliothek fertiggestellt und kann in kompilierter DLL-Datei-Form in jedem Software-Projekt verwendet werden, welche eine Ansteuerung mittels NTC erhalten soll.

6. Anhang

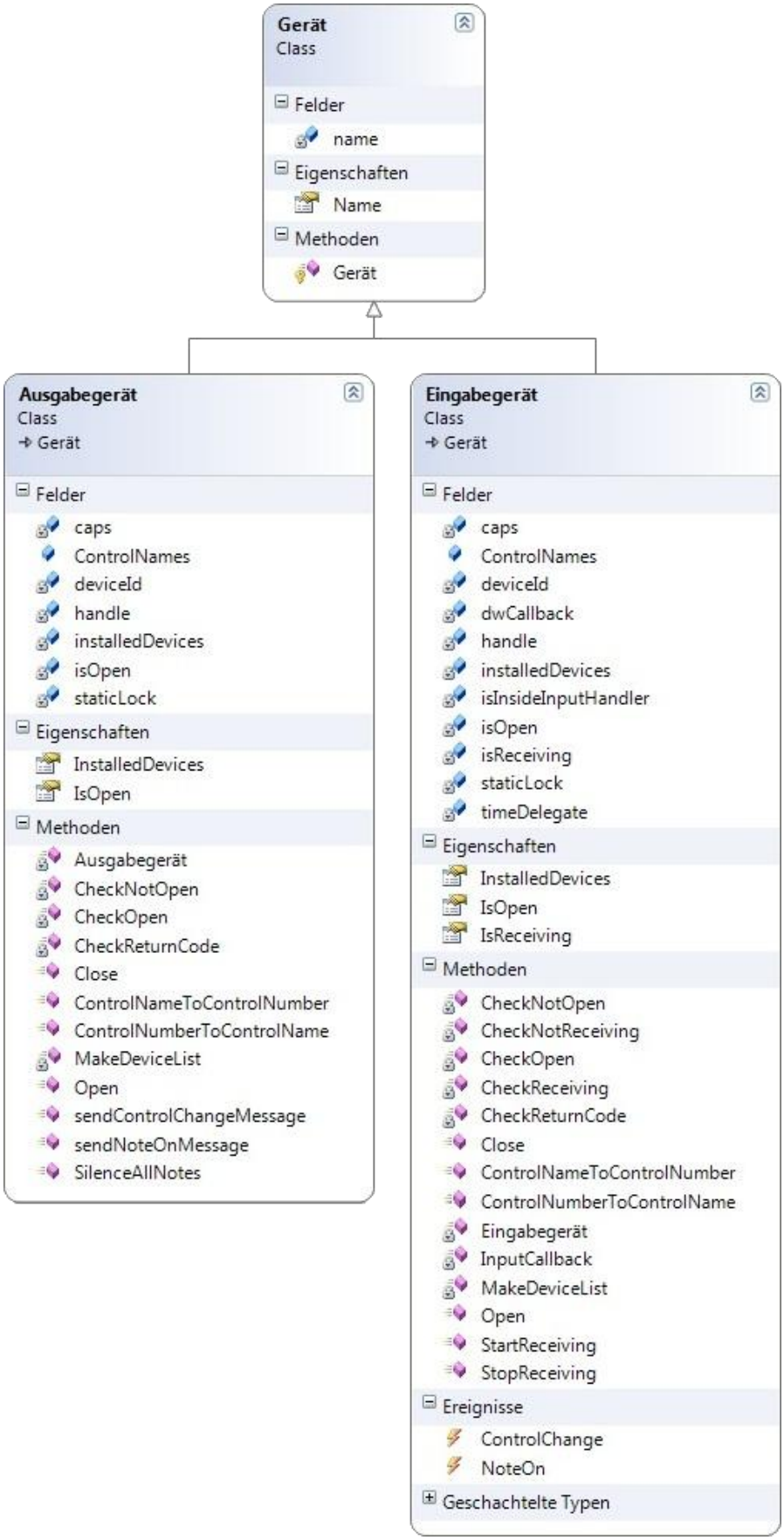
Anhang A: Numark TotalControl MIDI-Map



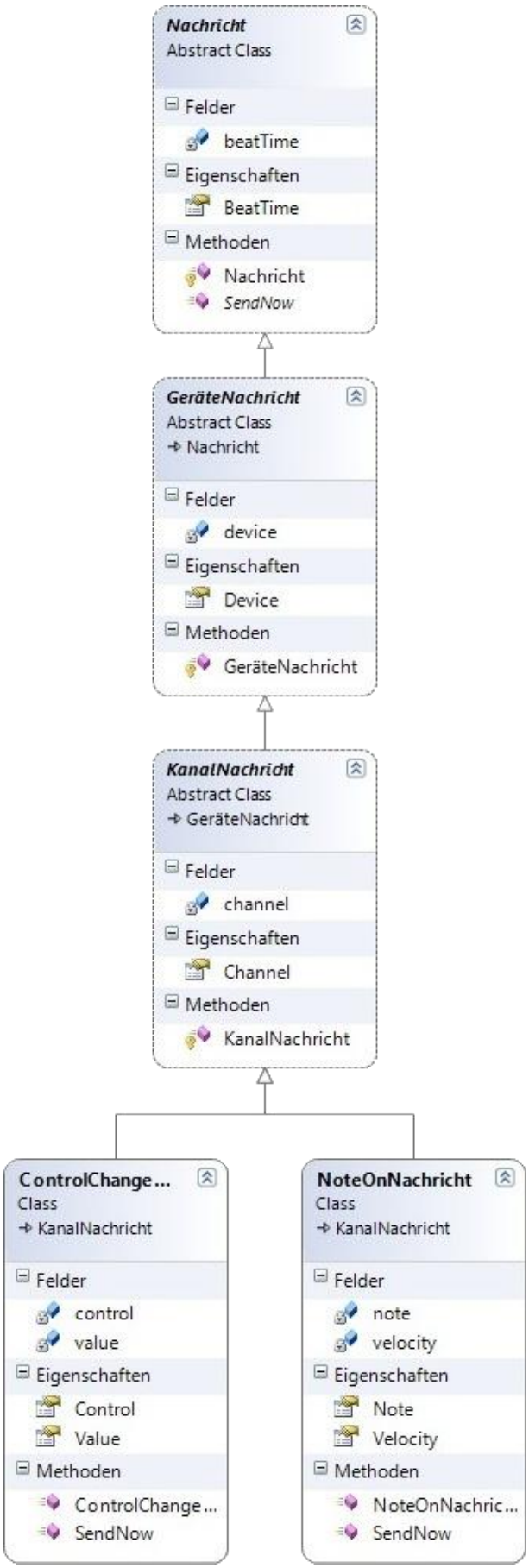
Anhang B: Klassendiagramm der gesamten PulseMIDI-Klassenbibliothek



Anhang C: Klassendiagramm der Gerät-Klassen mit Attributen



Anhang D: Klassendiagramm der Nachricht-Klassen mit Attributen



Anhang E: Inhalt der beigelegten CD-Rom

Im Quellverzeichnis der beigelegten CD-Rom befindet sich diese Dokumentation in digitaler Form, als PDF-Datei. Weiter sind drei Ordner vorhanden.

Im Quellen-Ordner befinden sich alle genutzten Quellen, die der Recherche zur Funktionsweise des MIDI-Protokolls, dem Umgang mit dem Numark TotalControl und der Umsetzung der PulseMIDI-Klassenbibliothek verwendet wurden. Besonders die für den letzten Punkt erforderlichen MSDN-Artikel sind nochmals im MSDN-Ordner abgelegt. An welcher Stelle im SourceCode, welcher MSDN-Artikel zum Einsatz kam, ist den Kommentaren des SourceCode zu entnehmen.

Der SourceCode befindet sich im Ordner Visual Studio 2008-Projekt. Die Projektmappe umfasst 4 Projekte. Dazu zählen die PulseMIDI-Klassenbibliothek, LightsOnOff - das Test-Projekt zur MIDI-Ausgabe und InputRecognition – das Test-Projekt zur MIDI-Eingabe. LightsOnOff sendet NoteOn-Nachrichten um die LEDs ein- und auszuschalten. InputRecognition hingegen empfängt NoteOn- und ControlChange-Nachrichten und gibt die Daten in der Konsolen-Version aus, beziehungsweise zeigt dies grafisch auf der NTC-Skizze in der GUI-Version.

Von beiden Test-Projekten existieren Videos im gleichnamigen Ordner im CD-Quellverzeichnis.

7. Quellenverzeichnis

- Alle MIDI-Spezifischen Informationen wurden der offiziellen „Universal Serial Bus Device Class Definition for MIDI Devices“ entnommen. Sie wurde am 29. November 2009 um 13:04 von der Internet-Quelle „http://www.usb.org/developers/devclass_docs/midi10.pdf“ bezogen. Eine Kopie befindet sich auf der CD-Rom im entsprechenden Quellen-Verzeichnis. In die Dokumentation übernommen wurde der Tabellen-Auszug aus Information 1 (siehe Seite 10) und das Diagramm in angepasster Form aus Information 2 (siehe Seite 11).

- Der Anhang A wurde dem mitgelieferten NTC-Handbuch entnommen und zusätzlich in digitaler Form am 29. November 2009 um 13:04 von der Internet-Quelle „http://www.numark.com/stuff/contentmgr/files/1/9e0e1954f2b6a7df5d19e96a4994163a/file/totalcontrol_midimap.pdf“ bezogen. Eine Kopie befindet sich auf der CD-Rom im entsprechenden Quellen-Verzeichnis.

- Weiterhin wurde die Win32-Dokumentation der Windows Multimedia Klassenbibliothek, welche zum verfassen des SourceCodes benötigt wurde, den entsprechenden Artikeln des Microsoft Developer Network entnommen. Da es sich dabei um die Internet-Quelle „http://msdn.microsoft.com/“ handelt, liegen Kopien aller verwendeten Artikel auf der CD-Rom im entsprechenden Quellen-Verzeichnis. An welcher Stelle ein spezieller MSDN-Artikel zum Einsatz kam ist dem SourceCode der PulseMIDI-Klassenbibliothek zu entnehmen.

- Weiterhin konnten außerschulische Vorkenntnisse im Bezug auf die Implementierung einer eigenen Klassenbibliothek umgesetzt werden. In dem früheren, aus Eigeninitiative gestarteten Projekt wurde eine Klassenbibliothek zum Senden und Empfang von E-Mails via SMTP und POP3 realisiert, da das .Net Framework lediglich das Senden von E-Mails umfasst.

8. Versicherung

Ich versichere, dass ich die vorgelegte Facharbeit ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Ich bestätige ausdrücklich, Zitate und Quellenangaben mit größter Sorgfalt und Redlichkeit in der vorgeschriebenen Art und Weise kenntlich gemacht zu haben. Die genutzten Internettexte habe ich alle auf beiliegender CD-Rom ordnungsgemäß gespeichert.