Seminarfacharbeit Klasse 11/12

Audiosort

Erstellung eines frei konfigurierbaren Musikplayers unter Berücksichtigung verschiedener Audio-Formate

Daniel Birnstiel, 12as,
Max Ulbrich, 12as

Fachbetreuer: Herr Neumann
Seminarfachbetreuer: Frau Hahn

Erfurt, den 21.12.2012

Inhaltsverzeichnis

1 Einleitung

Rockmusik, klassische Klavierstücke oder Hörbücher – jeder Mensch bevorzugt eine andere Art an Unterhaltung mit Hilfe hörbarer Medien. Die Möglichkeiten, sich die unterschiedlichen Werke anzuhören, sind dabei fast genauso vielfältig wie die einzelnen Genres selbst. Abspielgeräte entwickelten sich von tragbaren Kassetten- und CD-Spielern zu MP3-Playern und iPods, aber auch für Computer gibt es eine unüberschaubare Menge an Musikplayern. Der Vorteil dieser heutigen Abspielgeräte ist die Einfachheit, mit der sich einzelne Lieder in eine Playlist speichern lassen und Musik sortiert werden kann. Man kann außerdem nicht mehr nur eine CD, etwa mit seinen Lieblingsliedern, sondern schier unendlich große Sammlungen mit Hilfe dieser Programme verwalten.

In unserer Seminarfacharbeit wollten wir uns genau mit diesem Thema auseinandersetzen: Wir setzten uns als Ziel, einen eigenen Musikplayer zu entwerfen und verschiedene Audio-Formate zu implementieren, wobei wir besonderen Wert auf eine effiziente Datenbank und eine simple, intuitive Benutzeroberfläche legten.

Ein weiterer Aspekt, auf den wir uns beziehen wollten, ist die Entwicklung und Nutzung verschiedener Speichertypen dieser Medien, die sich ebenfalls immer weiter entwickelten. Da jeder einzelne Typ besondere Eigenschaften aufweist ist es notwendig, sich auch mit diesen auseinander zu setzen, um in unserem Musikplayer das Dekomprimieren und Abspielen mehrerer Datentypen zu ermöglichen.

Unser Musikplayer sollte sich dabei durch seine Einfachheit und den Verzicht auf aus unserer Sicht unnötige Funktionen von anderen abheben. Mit Hilfe der Einarbeitung in die Themen der digitalen Musikspeicherung, MP3, Datenkompression, Datenbanken, CDs und Tagging von Musikdateien, die für das Erstellen unseres Programms zwingend notwendig sind, wollten wir unser Wissen erweitern und ein besseres Verständnis für diese Abläufe gewinnen.

An dieser Stelle möchten wir unserem Fachbetreuer Herrn Neumann für Hilfe und Ratschläge sowie zahlreiche Konsultationen danken. Weiterhin bedanken wir uns auch bei unserer Seminarfachbetreuerin Frau Hahn für die Unterstützung bei Fragen zum Prozess und zur schriftlichen Ausarbeitung der Arbeit.

2 Theorie der digitalen Musikspeicherung

2.1 Grundlagen der digitalen Musikspeicherung

Um die gesetzten Ziele umzusetzen, mussten wir uns zuerst mit den Grundlagen der digitalen Musikspeicherung auseinandersetzen. Bei der Aufnahme eines Musikstücks werden zunächst die Schallwellen durch einen Schallwandler, wie etwa ein Mikrofon, in elektrische Spannung und somit in ein analoges Signal umgewandelt. Um dieses nun auf einer Festplatte oder einem anderen digitalen Speichermedium speichern zu können, muss es anschließend in ein digitales Format konvertiert werden.

Für diese Funktion werden analog-digital-Umsetzer eingesetzt. Diese wandeln ein analoges Eingangssignal in ein digitales Signal um. Ein Verfahren, das besonders im Bereich der Audiotechnik häufig zum Einsatz kommt, ist die Pulsamplitudenmodulation (kurz PAM). Dabei wird das Eingangssignal in bestimmten, fest vorgegebenen Schritten abgetastet und der aktuelle Wert der Spannungskurve aufgenommen[1]. Die Qualität der entstehenden Daten ist dabei von der Abtastrate abhängig. Da die Kurve nicht in unendlich kleinen Schritten abgetastet werden kann, ist statt einer Spannungskurve eine Art Treppenfunktion zu beobachten, welche eine Näherung der Eingangsdaten darstellt. Diese Werte können dann in digitaler Form gespeichert und wieder abgespielt werden.

Das Abspielen läuft umgekehrt zur Aufnahme ab. Aus den verschiedenen Spannungsstufen wird versucht, eine analoge Spannungskurve zu interpolieren. Interpolation beschreibt ein Verfahren, bei dem versucht wird, aus einzelnen Messwerten eine Kurve zu rekonstruieren. In den meisten Fällen wird dabei der Mittelwert der einzelnen Stufen verwendet und zur Ausgabe an die Soundkarte geschickt. Diese spielt das Signal dann über einen Lautsprecher oder ein anderes Gerät ab.

Das auf diese Weise generierte digitale Signal ist jedoch aufgrund der beschränkten Bandbreite verlustbehaftet, da es bei der Interpolation der Treppenfunktion in eine analoge Kurve zu Rundungen der Werte und zum Wegfallen kleinerer Schwankungen kommt. Mit einer höheren Auflösung der Abtastung ist es möglich, das Eingangssignal genauer einzulesen und die Qualität zu erhöhen. Dabei steigert sich allerdings der Speicherbedarf der Daten, da nun weitaus mehr Werte eingelesen und gespeichert werden müssen. Es muss somit ein Kompromiss zwischen Speicherbedarf und Qualität der Daten gefunden werden. Im heutigen Zeitalter ist jedoch die Datenmenge aufgrund stetig steigender Speicherkapazitäten ein untergeordnetes Problem, dafür steht die Qualität der Aufnahme im Fokus.

Die Pulsamplitudenmodulation ist eine Vorstufe zur Pulscodemodulation (PCM), welche als Speicherformat für Audiodateien in verschiedenen Codecs[2] verwendet wird und die Grundlage aller

[1] siehe Abbilung 2 auf Seite 22

[2] engl. Kodierungsverfahren für Audio-Signale

Komprimierungsalgorithmen darstellt. Beim PCM-Verfahren wird das eingelesene PAM-Signal in ein Treppenförmiges umgewandelt. Jeder Spannungsstufe wird nun ein entsprechender Binärcode zugewiesen, welcher für die Übertragung des digitalen Signals verwendet wird. Daher lässt sich das PCM-Signal als eine Folge von Stufenhöhen auffassen. Wichtig ist dabei, dass sowohl bei der Kodierung als auch bei der Dekodierung der Daten die gleiche Tabelle für das Umwandeln der Stufe in die Binärdaten verwendet wird. Nur dann können Ein- und Ausgangssignal problemlos in einander umgewandelt werden. Die stufenweise Einteilung der Kurve wird auch als Quantisierung bezeichnet.

Neben der Verwendung der gleichen Kodierungstabellen beim Komprimieren und Dekomprimieren ist auch die sinnvolle Einteilung der Spannungsstufen entscheidend. Beim Einlesen des Signals kann es durch Rundung des Wertes zu Abweichungen von maximal einer halben Spannungsstufe kommen. Sind diese Bereiche zu klein gewählt, kann es aufgrund enormer Schwankungen im digitalen Signal zu einem stark bemerkbaren Qualitätsverlust kommen. Die Qualität des Ausgangssignals steht dabei in Abhängigkeit zum Speicherbedarf der Daten. Auch hier muss für die Speicherung von Musikdateien ein Mittelwert zwischen Musikqualität und Dateigröße gefunden werden.

Die endgültige Speicherung der Datei erfolgt letztendlich in einem Audioformat, wie etwa dem Wave-Format, welches unter anderem auch die unkomprimierte Speicherung des Signals erlaubt. Im Gegensatz dazu wird das erzeugte digitale Signal bei komprimierten und dabei verlustbehafteten Formaten, wie etwa MP3, unter Berücksichtigung verschiedener Algorithmen so gespeichert, dass die Qualität zwar geringfügig abnimmt, jedoch der benötigte Speicherplatz stark verkleinert werden kann. Dies ist insbesondere für tragbare Abspielgeräte von Vorteil.

2.2 Einstieg in die Dekompression mit dem Wave-Codec

Das erste Dateiformat, mit dem wir uns in unserer Arbeit beschäftigten, war das Wave-Dateiformat. Dieses war nach unserer Ansicht am simpelsten zu implementieren und bot gleichzeitig eine ähnliche Grundstruktur, wie das CDA- und das MP3-Format, weshalb wir gewonnene Erkenntnisse anschließend wieder anwenden konnten.

Das Wave-Format ist ein Containerformat zur Speicherung von Audiodateien, welches auf dem RIFF-Format[3] aufbaut. Das bedeutet, dass innerhalb des Containers weitere Formate als Datenblöcke in der Datei zu finden sind. Im Speziellen sind dies der sogenannte Format-Chunk und der Sound-Data-Chunk.

Ein RIFF-Container beginnt mit einem Header[4]. Dieser beinhaltet die Bytefolge RIFF, die Dateigröße abzüglich des Headers und den Dateityp der Daten. Als Datentyp wird im Fall von Wave-Dateien die Zeichenkette „WAVE“ zu finden sein. Auf den Header folgt anschließend eine beliebige

[3] Resource Interchange File Format

[4] Kopfelement zur Beschreibung der folenden Daten

Anzahl von Chunks. Jeder dieser Blöcke hat wiederum eine Bytefolge zur Identifikation und einem Datenblock.

Abhängig vom Blocktyp müssen verschiedene Aktionen durchgeführt werden. Der fmt-Chunk beinhaltet Informationen über die Kodierung der Audio-Daten. Dazu gehören Komprimierungsverfahren, Bitrate des Audio-Signals und die Anzahl der Kanäle. Im data-Block liegen die Audio-Daten als Binärfolge vor. Bei unkomprimierten Daten kommt das bereits beschriebene PCM-Verfahren zum Einsatz. Ansonsten muss das Audio-Signal entsprechend des im fmt-Chunks angegebenen Verfahrens dekodiert werden.

Die Daten können sowohl als Mono- als auch Stereo-Signal vorliegen. Mono beschreibt dabei das Vorhandensein einer einzigen Audiospur, die vom Gerät abgespielt wird. Ein Stereo-Signal hingegen beinhaltet zwei Audiospuren, die jeweils für das linke und rechte Ohr vorgesehen sind. Hierbei verdoppelt sich der Datenaufwand, jedoch wird die Klangqualität merklich besser.

Neben den beschriebenen Chunks können in einer Wave-Datei auch noch weitere Informationen vorliegen. Über den cue-Block können innerhalb der Audiodatei verschiedene Haltemarken gesetzt werden, zwischen denen die Möglichkeit besteht, zu springen. Diese kommen beispielsweise beim Einlesen einer CD in eine Audiodatei vor, um die einzelnen Titel von einander abzutrennen und einzeln ansprechen zu können. Eine ähnliche Funktion hat der plst-Chunk, welcher eine Abspielliste (engl. playlist) beinhaltet. Diese erlaubt die Wiedergabe der Datei in einer von ihrer gespeicherten Struktur abweichenden Reihenfolge.

Der list-Block erlaubt die Speicherung von Zusatzdaten über den Titel. Diese Meta-Informationen[5] können von Musikplayern verwendet werden, um die Musikdatenbank zu organisieren und weitere Details zu finden.

Da das Wave-Format nur einen Container darstellt, kommen verschiedene Komprimierungsverfahren zum Einsatz. Diese unterscheiden sich in ihrer Komplexität und dem benötigten Datenaufwand enorm. Beispielsweise verbraucht ein unkomprimiertes Signal von drei Minuten Länge rund zehnmal so viel Speicher wie eine mit dem MP3-Algorithmus komprimierte Audiospur.

Weitere Verfahren, die zum Einsatz kommen, sind μLaw, ADPCM[6] oder das Truespeech-Verfahren.

2.3 Musikspeicherung auf der CompactDisc

Neben dem Speichern von Musikdateien auf einer Festplatte ist auch die CD ein weit verbreitetes Medium der heutigen Zeit. Die Compact Disc wurde in der 80er-Jahren des 20. Jahrhunderts als Nachfolger der bis dahin vorherrschenden Schallplatte oder Kassette erfunden. An der Entwicklung waren die Konzerne Sony und Philips maßgeblich beteiligt.

[5] Zusatzinformationen
[6] Adaptive Differential Pulse Code Modulation

Eine CD ist eine zwölf Zentimeter große Kunststoffscheibe, die aus einem Polycarbonat mit einer Metallbeschichtung besteht. Die Daten werden spiralförmig auf einer von innen nach außen verlaufenden Spur gespeichert, wobei diese in die Metallschicht der Oberfläche eingebrannt wird. Aufgrund unterschiedlicher Längen und Tiefen der Mikrostruktur entsteht bei der Bestrahlung mit einem Laser der Wellenlänge 780nm[7] ein Reflektionsmuster, welches als digitaler Code aufgefasst werden kann. Beim Auslesen dreht sich die CD unter einem Laser, der die Datenspur abfährt. Abhängig von der Position wird dabei die Drehgeschwindigkeit nach außen hin geringer, um einen konstanten Lesefluss zu gewährleisten. Die Abtastung erfolgt dabei kontaktlos, wodurch Verschleißerscheinungen, wie sie noch bei der Schallplatte zu beobachten waren, nicht auftreten. Die einzelnen Spuren haben eine Breite von ungefähr 1,6μm[8], woraus sich eine maximale Abspiellänge von 74 Minuten ergibt.

Die Daten auf der CD werden nicht direkt als Binärmuster gespeichert. Um die Integrität der Daten sicherzustellen, wird die Eight-To-Fourteen-Modulation eingesetzt. Diese wandelt ein 8-Bit[9] langes Signal in ein 14-Bit-Signal um. Die Konvertierung erfolgt dabei anhand fest vorgegebener Tabellen. Der Standard sieht es vor, dass zwischen zwei Folgen von Einsen mindestens zwei Nullen stehen. Dies ist darauf zurück zu führen, dass der Lesekopf eine gewisse Spur- bzw. Signalbreite zur eindeutigen Erfassung benötigt. Auf diese Weise können Lesefehler, die auf die Ungenauigkeit des Lasers zurückgehen, reduziert bzw. vermieden werden.

Um Musikdaten von einer CD lesen zu können, muss zunächst eine Art Inhaltsverzeichnis eingelesen werden. Es beinhaltet die Länge der CD-Titel sowie die Startpositionen des ersten und letzten Titels. Weiterhin ist eine Liste mit Informationen zu jedem einzelnen Titel enthalten, wobei zu jedem Titel verschiedene Zusatzdaten gespeichert sind. Diese beinhalten auch die Positionen der Daten im Speicherblock, die für das Abspielen essentiell sind. Ohne ein gültiges Inhaltsverzeichnis kann eine CD nicht wiedergegeben werden, da das Abspielgerät nicht zwischen den einzelnen Audiospuren unterscheiden kann.

Die Audiodaten liegen als unkomprimierter Datenstrom im PCM-Format mit einer Frequenz von 44,1 kHz[10] vor. In den meisten Fällen teilt sich dieser in zwei Kanäle auf, die bei einem Stereo-Signal für den linken und rechten Lautsprecher stehen.

[7] $1nm = 10^{-9}m$
[8] $1\mu m = 10^{-6}m$
[9] 8 Bit =1 Byte
[10] $1kHz = 1.000Hz$

2.4 MP3 als effizientes und populäres Speicherverfahren

2.4.1 Grundlagen der MP3-Kompression

Das Dateiformat MP3[11] wurde 1982 entwickelt und ist heutzutage die wahrscheinlich bekannteste und weitest verbreitete Art zum Speichern von Musik und Ton. Es beruht auf einer verlustbehafteten Kompression digital gespeicherter Audiodateien. Ein Vorteil dieser Methode ist, dass es auf der sogenannten Psychoakustik beruht. Es werden die Audio-Signale entfernt, die für den Menschen bewusst nicht mehr hörbar sind. Dadurch belastet sie den Speicher im Vergleich zu anderen Methoden weniger, obwohl sich aus der Sicht des Zuhörers an der Qualität des Tons nichts ändert.

2.4.2 Aufbau einer MP3-Datei

Eine MP3-Datei kann als eine Folge von Frames[12] aufgefasst werden, wobei jedes aus einem Header[13] und einem Datenblock besteht. Der Kopf eines jeden Chunks beginnt mit einem Synchronisierungsblock. Dieser 12 Bit lange Block beinhaltet den Wert 4095 (FFF_{16}[14]) und dient der genauen Trennung der einzelnen Blöcke. Weiterhin ermöglicht er die Überprüfung, ob ein Frame erfolgreich übermittelt wurde. Es folgen drei Bits, die die Version des MPEG-Formates beschreiben. Für MP3 steht dort die Bitfolge 101. Weiterhin werden Informationen zur Kodierung der Audiodaten gespeichert. Dazu zählen Bitrate, Frequenz oder die Anzahl der Kanäle.

Der Datenblock folgt direkt auf den Header. Er teilt sich wiederum in einen Kopf- und Datenbereich auf. Neben komprimierter Audiodaten sind weitere Informationen gespeichert, die zur Dekomprimierung notwendig sind. Jedes MP3-Frame enthält Daten für ungefähr 26ms Audio-Signal.

Der Block aus MP3-Frames kann zusätzlich in verschiedene Meta-Informationen eingebettet sein, die weitere Details zu dem gespeicherten Audio-Signal beinhalten. Ein Beispiel hierfür ist das ID3-Tag, das Daten über Titel, Künstler oder Album speichert[15].

2.4.3 Kompression im Detail

Die Kompression eines Audio-Signals lässt sich beim MP3-Algorithmus in fünf verschiedene Schritte einteilen, die nacheinander abgearbeitet werden. Für die Dekomprimierung werden diese in umgekehrter Reihenfolge abgearbeitet. Ein Schema der Dekomprimierung ist in Abbildung 5 auf Seite 23 zu sehen.

[11] MPEG Audio Layer III
[12] engl. Datenblock
[13] siehe Abbildung 4 auf Seite 23
[14] 4095 im Hexadezimalsystem
[15] siehe Kapitel 3.1 auf Seite 10

Zunächst wird eine Subband-Transformation des Signals durchgeführt, wobei es in mehrere Frequenzbänder aufgeteilt wird. Hierbei wird sich zu Nutze gemacht, dass das menschliche Ohr zwar eine große Variation an Tönen wahrnehmen kann, jedoch bei der Überlagerung dieser es zu Informationsverlust kommt. So werden leise Töne in Kombination mit lauteren meist überhört und können aus dem Signal entfernt werden. Dadurch kommt es zu einer Verringerung des Datenaufwandes.

Die einzelnen Frequenzbänder werden nun einer MDCT-Transformation[16] unterzogen. Dabei findet eine spektrale Auflösung der einzelnen Subbänder in weitere Frequenzbänder statt. Jedes dieser Bänder beinhaltet nun Frequenzen eines kleinen Teils des Klangspektrums, wobei sich immer zwei benachbarte überlappen. Sie ergeben gemeinsam wieder das vollständige Klangbild.

Nach der Aufteilung des Frequenzbandes erfolgt seine Matrizierung. Sofern ein Stereo-Signal vorliegt, kann nun entschieden werden, ob dieses als Mono-, Stereo-, Joint-Stereo- oder Dual-Channel kodiert wird. Da im Stereo- und im Dual-Stereo-Verfahren für das linke und rechte Ohr jeweils eine Audiospur kodiert werden müssen, steht nur die Hälfte der Bandbreite zur Verfügung. Dadurch kommt es zu einer Verringerung der Aufnahmequalität. Beim Dual-Channel versucht man diesem Problem entgegen zu wirken, indem zwei vollwertige Mono-Spuren gespeichert werden. Allerdings verdoppelt sich somit auch der Speicheraufwand und manche Geräte zeigen Probleme beim Abspielen beider Spuren. Auf diese Weise gehen auch hier Informationen verloren. Eine weitere Möglichkeit ist das Joint-Stereo-Verfahren, bei dem aus den beiden Eingangskanälen ein Mittelkanal (L+R) und ein Kanal für die Lautstärkedifferenz (L-R) gebildet werden. Man unterscheidet zwischen Intensitäts- und Mid/Side-Stereo. Der Unterschied ist, dass bei Ersterem der Laufzeitunterschied beider Signale vernachlässigt wird. Der Vorteil des Joint-Stereo-Verfahrens ist im Gegensatz zu den anderen, dass Redundanzen in den beiden Kanälen aufgehoben werden. Dadurch kann entweder der Speicherbedarf reduziert oder die Bitrate zu Gunsten der Qualität erhöht werden. Auch an dieser Stelle kann die Einsparung von Frequenzen, die für das menschliche Ohr nicht mehr hörbar sind, eine höhere Kompressionsrate erreicht werden.

Der verlustreichste Schritt der Komprimierung ist die Quantisierung, welche allerdings den Kompressionsfakter am meisten beeinflusst. Dabei werden benachbarte Frequenzbänder zu Gruppen von so genannten Bins zusammengefasst. Außerdem wird jeder Gruppe ein Skalenfaktor zugewiesen, welcher für die Quantisierung eine enorme Rolle spielt und die Genauigkeit bestimmt. Er bestimmt, ab welchen Frequenzbereichen ein Ton als nicht hörbar eingestuft wird und entfernt werden kann. Beim MP3-Verfahren kommt ein so genannter „power-law"-Quantisierer zum Einsatz. Dabei wird versucht, kleine Werte möglichst genau zu quantisieren, bei größeren werden allerdings Rundungsfehler in Kauf genommen. Fehler bei der Quantisierung sind durch ein Rauschen oder ein verwaschenes Signal bemerkbar. Allerdings können diese bei der Dekomprimierung mit Hilfe eines geeigneten Faktors, welcher ebenfalls in der resultierenden Datei gespeichert wird, reduziert

[16] Modifizierte diskrete Kosinustransformation, http://www.mp3encoding.de/index_10.html

werden.

Abschließend werden die quantisierten Audio-Signale einer Huffman-Kodierung[17] unterzogen. Bei der Huffman-Kodierung handelt es sich um ein Komprimierungsverfahren, welches sich die Häufigkeit einzelner Buchstaben zu Nutze macht und ihnen entsprechend lange Codewörter zuweist. Der Algorithmus arbeitet verlustfrei, bietet aber dennoch eine hohe Kompressionsrate. Beim MP3-Verfahren stehen 32 verschiedene Code-Tabellen zur Verfügung, aus denen das Programm den für das Signal effizientesten Eintrag auswählt. Die kodierten Daten werden dann jeweils in einzelnen MP3-Frames gespeichert. Hinzu kommen Informationen wie der Quantisierungsfaktor und die Huffman-Code-Tabelle, die ebenfalls im Datenteil abgelegt werden.

[17] Weitere Informationen: http://www.iti.fh-flensburg.de/lang/algorithmen/code/huffman/huffman.htm

3 Tagging von Musikdateien

3.1 ID3-Tags zur Speicherung von Meta-Informationen

Um verschiedene Informationen über Musiktitel anzuzeigen, müssen diese zuerst in der Datei gespeichert werden. Für diese Aufgabe wird das ID3-Tag oder auch „Identify an MP3“ Format angewendet. Dabei werden je nach verwendeter Version an unterschiedlichen Positionen der Audiodatei Informationen über Interpret, Titelname, Album, Genre und dateispezifische Angaben gespeichert.

Die erste Version ID3v1 wurde 1996 von Eric Kemp entwickelt. Dabei wurden die entsprechenden Informationen wurden an das Ende der Datei in einem 128 Bytes großen Block angefügt. Dies brachte allerdings einige Nachteile mit sich, da zum Beispiel die Länge der Datenfelder für Titel und Album auf 30 Byte begrenzt war. Außerdem war es aufgrund des am Ende der Datei liegenden Tags nichtmöglich, die Dateien über das Internet zu streamen[18], was besonders aus heutiger Sicht ein entscheidender Nachteil ist. Ein Stream bezeichnet dabei die Datenübertragung über das Internet, wobei das Dateiende nicht bestimmt ist. Diese Technik kommt beispielsweise bei Live-Übertragungen oder Internet-Radios zum Einsatz.

Aufgrund verschiedener Nachteile wurde 1998 von Matin Nilsson das ID3v2 Format entwickelt. Dieses wurde über die Jahre bis zur seit November 2000 aktuellen Version ID3v2.4.0 weiterentwickelt. Es erlaubt zahlreiche Zusatzinformationen und eine variable Position der ID3-Informationen. Das ID3v2-Tag wird in der Datei durch den Header „ID3“ und die verwendete Version eingeleitet. Außerdem ist es nun möglich, Cover-Bilder direkt in den Meta-Informationen abzulegen.

Die wichtigste Eigenschaft der verschiedenen Versionen ist die Abwärtskompatibilität innerhalb der Standards. Dadurch mussten wir im Rahmen unserer Arbeit nur die aktuelle Version, also ID3v2.4.0, implementieren und konnten alle vorausgehenden Versionen ebenfalls verwenden.

Um nun die Metainformationen einer Audiodatei auslesen zu können, muss zuerst die Datei eingelesen und der ID3-Header gesucht werden. Dieser besteht aus 10 Bytes, wobei die ersten drei zur Identifikation dienen. Die beiden folgenden geben die Versionsnummer an. Anschließend wird über verschiedene Flags die genaue Struktur des Tags beschrieben. Ein Flag kann einen Wert von 0 oder 1 annehmen und gibt an, ob eine bestimmte Einstellung gesetzt ist oder nicht. Beispielsweise kann ein erweiterter Header eingefügt werden, der zusätzliche Informationen enthält. Abschließend ist die Größe des Headers angegeben, welche in nur 32 Bit gespeichert wird. Dadurch können maximal 256 Megabytes an Daten adressiert werden.

Nach dem Finden und Einlesen des Headers werden die verschiedenen Informationen über die Datei eingelesen. Beim Einlesen ist zu beachten, dass zwischen ID3-Header und Daten weitere Informationen oder ein Puffer-Block vorhanden sein können. Genauere Informationen darüber werden dem Header entnommen.

[18] engl. Stream, Datenstrom

Sollte etwa das Flag für den Extended-Header gesetzt sein, wird dieser als nächstes ausgelesen. Innerhalb dieses zusätzlichen Teils werden unter anderem Auskünfte über Beschränkungen der Speichergröße des Albumbildes oder des Tags gespeichert. Wie im eigentlichen Header müssen hierbei nicht alle Möglichkeiten genutzt werden, sodass einige Informationen nicht unbedingt gespeichert sind.

Die eigentlichen Informationen werden in verschiedenen Frames abgespeichert. Jedes Frame beginnt mit einer eindeutigen, 4 Byte großen Zeichenfolge, die die Daten darin beschreibt. Über verschiedene Flags können auch hier weitere Informationen über beispielsweise eine Kodierung oder Komprimierung der Werte angegeben werden.

Das ID3-Tag wird meistens mit einem so sogenannten „Footer“ abgeschlossen, um die Metainformationen der Datei schneller finden zu können. Dieser wird durch „3DI“ eingeleitet und enthält nochmals die Informationen des Headers, sodass je nach Einstiegspunkt der Suche ein schnelleres Ergebnis zu ermitteln ist. Ein Footer ist wichtig, sofern das ID3-Tag am Ende der Datei zu finden ist, da somit nur die ersten und letzten 10 Byte einer Datei auf das Vorhandensein entsprechender Werte untersucht werden müssen.

In unserem Mediaplayer implementierten wir eine entsprechende Klasse, welche zuerst den ID3-Header in einer MP3-Datei findet und anschließend die Informationen ausliest. Die so gewonnen Informationen werden dann in die Musikdatenbank eingetragen.

3.2 Erkennen von Musik-CDs mit Hilfe der CDDB-ID

Ein weiteres Kernelement unserer Arbeit war nicht nur das Einlesen von CDs, sondern auch die automatische Zuordnung der Tag-Informationen zu den Titeln. Diese beinhalten Informationen über den Titel, dessen Urheber oder das Album, in dem er veröffentlicht wurde. Aus diesem Grund implementierten wir eine Autotagging-Funktion für unseren Player. Diese sollte einer eingelegten CD mithilfe der Compact-Disc-Database (CDDB) die entsprechenden Informationen zuordnen.

Die Idee der Datenbank stammt aus dem Jahr 1995 und wurde unter dem Namen „Gracenote“ veröffentlicht. Da jedoch Sony 2008 diese aufkaufte, sind die Daten nicht mehr kostenlos verfügbar. Aus diesem Grund spalteten sich einige weitere Gruppen ab, die an einer kostenlosen Version arbeiten. Ein Beispiel hierfür ist die Seite freedb.org, von welcher wir unsere Informationen beziehen. Sie stellt die Daten unter der GNU-Lizenz zur Verfügung und ist somit frei nutzbar.

Die Identifikation der CDs erfolgt über die so genannte CDDB-Id. Diese ist ein Hashwert, der aus verschiedenen Informationen wie Länge der einzelnen Titel oder deren Position berechnet wird. Dieser Wert stellt eine eindeutige Zeichenfolge dar, die jede CD identifiziert. Die Chance, dass zwei verschiedene Datenträger die gleiche Id haben ist sehr gering. Daher kann dieses Verfahren als zuverlässig bezeichnet werden. Über ein Online-Framework ist es möglich, über diesen Wert die

gesuchten Informationen abzufragen. Die Rückgabe der Werte erfolgt über ein Textformat, welches von unserem Programm anschließend ausgewertet und weiterverarbeitet wird.

Für die Berechnung der CDDB-ID muss zunächst eine Art Inhaltsverzeichnis („Table of Contents“, TOC) von der CD geladen werden. Der TOC enthält dabei alle relevanten Informationen für die Berechnung, wie die Anzahl und die Reihenfolge der Titel auf der CD sowie die jeweilige Länge. Entscheidend dabei sind die sogenannten „Track Frame Offsets“, welche die Anfangszeiten eines Titels darstellen und – graphisch veranschaulicht – den Moment darstellen würden, in welchem keine Frequenz ausgegeben wird. Wird beispielsweise für den ersten Titel ein Offset von 150 bestimmt so beginnt dieser bei einer Frame-Rate von 75 Hz nach 2 Sekunden zu spielen.

Aus diesen Werten wird anschließend die Disc-ID ermittelt. Diese besteht aus einem Hexadezimal-Wert mit 8 Stellen und ist somit 4 Byte lang. Das erste Byte beinhaltet eine Prüfsumme, die aus der Summe der einzelnen Anfangszeiten besteht. Um diesen Wert im Bereich von 0 bis 255 zu halten, wird mit Hilfe der Modulo-Operation der Rest der Division durch 255 berechnet. Die nächsten zwei Bytes repräsentieren die Gesamtlaufzeit der CD in Sekunden, gemessen vom Start des ersten Titels, bis zum Ende des letzten. Abschließend stellen die letzten Bytes die Gesamtzahl an Titeln dar, welche auf der CD enthalten sind.

Sind diese Informationen aus den einzelnen Dateien ermittelt und ist der Hash-Wert nach dem entsprechenden Algorithmus ermittelt, wird dieser nun in der Compact-Disc-Database gesucht. Wird die entsprechende ID gefunden, können nun die entsprechenden Werte in unserer eigenen Datenbank gespeichert und angezeigt werden.

4 Erstellen unseres eigenen Musikplayers

4.1 Vorüberlegungen und Anforderungen an unser Programm

Ein wichtiges Kriterium für die Wahl der Programmiersprache war für uns, dass diese das Konzept der objektorientierten Programmierung verfolgt und einen großen Umfang an Standardbibliotheken und Funktionen bietet. Die Wahl fiel dabei auf die Sprache C#, die auf dem .NET-Framework von Microsoft aufsetzt. Sie ist vollständig objektorientiert aufgebaut und zwingt den Programmierer, dieses Konzept zu verfolgen. Außerdem bietet das .NET-Framework einen enormen Umfang an Bibliotheken, die für die Erstellung unseres Musikplayers wichtig sind. Dazu gehören verschiedene Möglichkeiten, auf Datenbanken zuzugreifen oder unterschiedliche Methoden der Benutzeroberflächengestaltung. Weiterhin bietet die Benutzeroberfläche Visual Studio eine umfangreiche Projektverwaltung, mit deren Hilfe sich größere Projekte einfach planen und realisieren lassen. Dabei ist auch die Arbeit in Gruppen über eine integrierte Versionsverwaltung mit Hilfe von git[19] oder svn[20] möglich. Verschiedene Designer für Benutzeroberflächen und Datenbanken erleichtern die Planung und Umsetzung des Programmes, da ein großer Teil durch einfache Drag-and-Drop-Möglichkeiten erleichtert wird.

Die Benutzeroberfläche gestalteten wir mit Hilfe der Windows Presentation Foundation, kurz WPF. Diese erlaubt moderne Oberflächen, die sich über verschiedene Möglichkeiten gestalten lassen. Die Bibliothek beinhaltet einen Designer, über den sich die einzelnen Steuerelemente entweder per Maus frei positionieren oder mit Hilfe verschiedener Container automatisch ausrichten lassen. Alternativ dazu lässt sich die Oberfläche durch die Beschreibungssprache XAML erstellen, welche auf einem XML-Dialekt basiert. Auf diese Weise lässt sich sogar die Benutzerschnittstelle modular aufbauen und erlaubt die einfache Integration verschiedener Designs.

Im Grunde besteht ein Musikplayer aus drei verschiedenen Komponenten. Der wichtigste Teil ist das Dekodieren und Abspielen von Musikdateien. Diese müssen in einer Musikbibliothek organisiert werden, die vom Programm gespeichert und verwaltet wird. Die Interaktion mit dem Anwender erfolgt über eine grafische Benutzeroberfläche.

Diese Komponenten werden in unserem Programm durch verschiedene Teile realisiert, die das Projekt gliedern. Weiterhin nutzen wir die in C# integrierte Methode des Namespacings[21], wodurch verschiedene Programmteile einen anderen Namensraum besitzen. Auf diese Weise wird eine klare Abgrenzung der einzelnen Bereiche erreicht. Das Hauptprogramm, welches in der Benutzeroberfläche integriert ist, bindet die anderen Bibliotheken über Verweise ein und kann somit auf deren Funktionen und Klassen zugreifen. Beim Erstellen des Projektes werden diese dann als Anwen-

[19] http://git-scm.com/
[20] http://subversion.apache.org/
[21] Einteilung der Programmteile in Namensräume

dungserweiterung[22] eingebunden und dynamisch zur Laufzeit geladen.

Zum Ausführen unseres Programmes wird die aktuelle Version des .NET-Frameworks[23], die Windows Presentation Foundation[24], sowie das aktuelle DirectX-Framework. Bibliotheken, die zusätzlich eingebunden werden, sind im Verzeichnis des Programms vorhanden.

4.2 Konzept und Aufbau der Programmdatenbank

Einer der wichtigsten Bestandteile eines Musikplayers ist die Datenbank, in der Informationen zu einzelnen Musiktiteln, Künstlern oder Alben gespeichert werden. Um diese so effizient wie möglich zu halten, ist eine genaue Planung und Strukturierung von großer Bedeutung.

Die Wahl eines Datenbanksystems fiel in unserer Arbeit auf das DataSet-Modell, welches im .NET-Framework von Microsoft bereits integriert ist. Wie in den meisten anderen Systemen lassen sich die Daten in verschiedenen Tabellen ablegen, welche über Verweise mit einander verknüpft sind. Ein Vorteil ist ein in die Entwicklungsumgebung Visual Studio 2010 integrierter Designer, welcher das Erstellen und Implementieren von Datenstrukturen enorm vereinfacht.

Eine Tabelle in der Datenbank besteht dabei aus mehreren Spalten. Für jede dieser lassen sich ein eindeutiger Name, Datentyp und Standardwert festlegen. Entsprechend können in einer Spalte nur Werte eines Typs gespeichert werden. Über so genannte Indizes lassen sich Spalten festlegen, über die ein Datensatz eindeutig identifiziert werden kann. In den meisten Fällen wird hierfür eine Ganzzahl (Integer) verwendet, welche beim Einfügen neuer Daten automatisch generiert wird. Über den (Primär-) Index lassen sich beispielsweise Querverweise auf andere Tabellen realisieren, um den Speicheraufwand zu reduzieren.

In der Musikdatenbank müssen verschiedene Daten gespeichert werden. Im Folgenden sind die Tabellen unserer Bibliothek dargestellt. Eine detailliertere Übersicht kann der Abbildung 6 auf Seite 24 entnommen werden.

Die Tabelle „Titel" beinhaltet Informationen über jedes einzelne Musikstück der Bibliothek. Dazu gehören Name, Interpret oder der Dateiname. Diese Daten werden vom Programm soweit möglich automatisch beim Einlesen einer Musiksammlung generiert und entsprechend zugeordnet. Um den Speicherbedarf der Datenbank zu verringern, werden in der Titel-Tabelle nur Verweise auf Interpret und Autor gespeichert. Dieser erfolgt über eine ID, welche in der Interpreten- bzw. Album-Tabelle entsprechend zugeordnet wird. Somit muss anstelle einer Zeichenkette für den Interpreten nur eine Zahl gespeichert werden, wodurch der Aufwand bei größeren Datenmengen enorm reduziert wird. Weiterhin muss anstelle jedes einzelnen Verweises bei der Korrektur eines Wertes nur ein Datensatz

[22] engl. dynamic link library, .dll-Datei
[23] Framework-Version 4 (http://www.microsoft.com/de-de/download/details.aspx?id=17851)
[24] http://msdn.microsoft.com/de-de/library/vstudio/ms754130.aspx

geändert werden. Ebenso wird mit dem Genre verfahren, welches ebenfalls jedem Titel zugeordnet wird.

In der Datenbank können beliebig viele Playlists gespeichert werden. Die Tabelle enthält für jede Abspielliste eine eindeutige Identifikationsnummer, über die die einzelnen Titel zugeordnet werden, sowie einen Namen. Die einzelnen Einträge in einer Playlist werden in dieser Tabelle gespeichert. Sie enthält einen Verweis auf den Datensatz in Playlist- und Titel-Tabelle.

Neben den Informationen über die Musikbibliothek werden auch die Einstellungen des Programmes in einer Datenbank gespeichert. Das Datenbankmodell AudiosortConfig enthält eine Tabelle für eine Zuordnung von Schlüssel-Wert-Paaren, die vom Programm abgefragt werden können. Die Einstellungen werden dann automatisch vom Programm geladen und beim Beenden gespeichert. Die Werte können in der Tabelle nur als Zeichenkette gespeichert werden. Andere Datentypen, wie beispielsweise Listen oder Zahlen, müssen zunächst in einen String umgewandelt und beim Laden entsprechend konvertiert werden.

Das Datenbankschema ist dabei im Programm integriert. Somit müssen nur die tatsächlich enthaltenen Werte in einer Datei gespeichert werden. Um die Verarbeitung der Daten durch unser Programm, aber auch durch externe Applikationen zu erleichtern, speichern wir die Datenbank in einer XML-Datei. Diese erhöht zwar zunächst den Speicherbedarf auf der Festplatte, jedoch sind die Dateien einfach mit einem Texteditor zu öffnen. Weiterhin lassen sich XML-Dateien von vielen Programmen weiterverarbeiten, sodass auch andere Programme auf unsere Bibliothek zugreifen können. Ein weiterer Vorteil ist, dass über so genannte XSL-Stylesheets die XML-Datei formatiert und somit in einem beliebigen Webbrowser geöffnet werden kann. Dieser zeigt die Daten dann in Tabellenform an. Ein Beispiel hierfür ist in Abbildung 3 auf Seite 22 zu sehen.

Beim Programmstart wird versucht, die in der Konfigurationsdatei angegebene Datenbank zu laden. Ist diese nicht vorhanden, wird eine leere Datei angelegt und gespeichert. Ist die Datenbank geladen, kann das Programm mit der Verarbeitung beginnen und die entsprechenden Daten dem Anwender anzeigen.

Die Abfrage der Daten erfolgt dabei über das in die Sprache C# integrierte Verfahren LINQ[25] (Language Integrated Query). Dieses erlaubt SQL-ähnliche Anweisungen, welche direkt im Quelltext angegeben und vom Compiler vorverarbeitet werden. Im Gegensatz zu regulären SQL-Ausdrücken haben LINQ-Abfragen den Vorteil, dass sie nicht aus einer Zeichenkette bestehen und somit die Interpretation nicht erst zur Laufzeit erfolgt, sondern bereits beim Erstellen eine Optimierung vorgenommen wird. Dadurch ist eine Performance-Erhöhung bemerkbar. Weiterhin lassen sich LINQ-Ausdrücke sowohl auf Datenbanken als auch auf Listen oder ähnliche Datenstrukturen anwenden, wodurch wir diese Möglichkeit durch unser gesamtes Programm hinweg verwenden konnten.

Zur Laufzeit werden über eine solche Abfrage dann die gesamte Musikbibliothek erstellt und die

[25] http://msdn.microsoft.com/de-de/library/vstudio/bb397926.aspx

einzelnen Querverweise aufgelöst. Dies geschieht über so genannte Joins, bei denen Daten aus mehreren Tabellen geladen werden. Kriterium für die Auswahl der Daten ist dabei die Übereinstimmung bestimmter Werte, in unserem Fall die Index-Nummern von Interpret, Album und Genre des Eintrags.

Um die Datenbank mit META-Informationen zu füllen, müssen verschiedene Tag-Informationen aus den Musikdateien gelesen werden, wobei das bereits beschriebene ID3-Tag[26] zum Einsatz kommt. Die Klasse DatabaseFiller aus unserem Datenbankmodul ist für diese Aufgabe zuständig. Bei einem gegebenen Verzeichnis werden alle Musikdateien aufgelistet und auf das Vorhandensein von Zusatzinformationen untersucht. Falls vorhanden werden diese ausgelesen und in die Datenbank übernommen. Dabei wird vor dem Einfügen des Eintrags auf Duplikate untersucht und gegebenfalls der Datensatz nur aktualisiert. Die Prüfung geschieht dabei auf Basis von Dateiname und Titel des Musikstücks.

Da Interpret, Album und Genre des Titels über Verweise gespeichert werden, muss zunächst nach einem entsprechenden Eintrag in der Datenbank gesucht werden. Ist dieser vorhanden, wird dessen Nummer in den neuen Eintrag gespeichert. Falls ein neuer Interpret gefunden wurde, wird ein neuer Eintrag angelegt und ebenfalls eingetragen.

Falls in einer Spalte kein Wert angegeben wurde, wird in dieser entweder eine leere Zeichenkette oder der Wert -1 gespeichert. Das Programm erkennt diese und erlaubt dem Benutzer, diese nachträglich manuell zu editieren.

4.3 Benutzeroberflächengestaltung mit der Windows Presentation Foundation

Für die Entwicklung der grafischen Benutzeroberfläche für unseren Musikplayer entschieden wir uns für das Windows Presentation Foundation-Framework. Durch die darin vorgegebene Struktur ist eine strikte Trennung von Präsentations- und Anwendungslogik möglich. Weiterhin bietet WPF einen großen Umfang an dynamischen Steuerelementen, mit denen wir unsere Oberfläche frei gestalten konnten.

Für das Design der Oberfläche gibt es zwei Möglichkeiten. Zum einen kann ein in die Entwicklungsumgebung Visual Studio integrierter Designer verwendet werden, der das Hinzufügen neuer Elemente per Mausklick ermöglicht. Auf der anderen Seite kann eine XML-Beschreibungssprache verwendet werden. Microsoft stellt hierfür die Sprache XAML[27] zur Verfügung. Dabei lassen sich die einzelnen Steuerelemente als XML-Tags definieren und über verschiedene Attribute genauer beschreiben. Über so genannte Event-Handler kann eine Verbindung zur Programmlogik herge-

[26] siehe Kapitel 3.1 auf Seite 10
[27] Extensible Application Markup Language

stellt werden. Diese werden beispielsweise aufgerufen, wenn eine Schaltfläche gedrückt oder ein Schieberegler verschoben wurde.

Ein Vorteil, den WPF mit sich bringt, ist, dass sich die einzelnen Steuerelemente frei positionieren lassen und über externe Style-Dateien erst ihre Form und Farbe erhalten. Somit kann der Grundaufbau einer Anwendung fest stehen (Abbildung 7 auf Seite 24), aber beliebig in Form und Farbe verändert werden (Abbildung 8 auf Seite 25). Dieses Prinzip machen wir uns bei der Implementation eines Style-Systems zu Nutze. Ein Design besteht in unserem Fall aus einer Bibliothek, welche den Klassentyp StyleInformation definiert. Dieser verweist auf verschiedene Styles, die in der Bibliothek zu finden sind. Jeder Style beinhaltet eine Liste mit XAML-Dateien, über die das Design des Programmes bestimmt wird. Sie werden automatisch aus der Assembly[28] geladen und über den Resourcenmanager von WPF in die Oberfläche eingebunden.

Grafiken werden dabei dynamisch mit Hilfe verschiedener Zeichentools erstellt oder aus bereits erstellten Bildern geladen. Für die Schaltflächen in unserem Programm verwendeten wir so genannte SVG-Grafiken[29]. Diese bieten den Vorteil, dass sie mit wenigen Befehlen aus einer XML-Datei erstellbar sind und sich in beliebiger Größe skalieren lassen. Auf diese Weise konnten wir einmal erstellte Bilder in verschiedenen Größen einsetzen und mussten nicht für jede Anwendung oder Änderung eine neue Grafik erstellen.

Einzelne Menü-Icons entstammen dem Silk-Iconset von Mark James[30] und sind nach der Creative Commons Lizens[31] frei verwendbar.

Beim Design der Oberfläche stand für uns im Vordergrund, eine schlichte und einfache Umgebung zu schaffen[32]. Wie in den meisten Anwendungen auch sollte der Musikplayer im oberen Bereich eine Menüleiste besitzen, über die verschiedene Programmfunktionen angesprochen werden können. Der untere Fensterrand wird von einer Schaltflächenleiste zur Steuerung der Wiedergabe ausgefüllt. Zwischen diesen Leisten ist die Medienbibliothek angebracht. Da diese einen großen Umfang an Daten beinhaltet, bekommt sie auch den größten Platz im Programmfenster zugewiesen. Horizontal ist weiterhin eine Übersicht mit verschiedenen Wiedergabelisten und Abspielgeräten wie CDs angeordnet. Auf diese Weise bleibt die Oberfläche übersichtlich und der Anwender hat alle benötigten Funktionen sofort im Blick.

Bei der Farbgebung unseres Programmes entschieden wir uns für eine Mischung aus Orange und Grau. Auf diese Weise konnten wir gute Kontraste erzeugen, die alle Details auch bei schlechten Auflösungen oder Grafikeinstellungen erkennen lassen. Die verwendeten Farbverläufe werden dabei zur Laufzeit dynamisch generiert und sind somit auf der Oberfläche frei skalierbar. Nahezu

[28] Programmbibliothek
[29] Scalable Vector Graphics, skalierbare Vektorgrafiken
[30] http://www.famfamfam.com/lab/icons/silk/
[31] http://creativecommons.org/licenses/by/2.5/
[32] siehe Skizze in Abbildung 9 auf Seite 25

alle Details der Oberfläche lassen sich über verschiedene Style-Dateien verändern. Auf diese Weise kann der Anwender, sofern er über die benötigten Kenntnisse verfügt, den Musikspieler seinen Wünschen entsprechend anpassen.

4.4 Umsetzung der Audio-Codecs im Programm

Für die Umsetzung der einzelnen Audio-Formate war eine Abstraktion der einzelnen Klassen notwendig. Um eine Audio-Datei im Speicher zu repräsentieren, entwarfen wir die Klasse Audio-Stream. Diese Klasse stellt einen Stream dar und ist somit ein Speicher, auf den verschiedene Lese-Zugriffe ausgeführt werden können. Sie verfügt über Funktionen für das Auslesen von Daten, das Erstellen aus einer Datei oder Speicherbereich und das Kopieren in einen anderen Speicher. Weiterhin ist ein Verweis auf ein WaveFormat-Objekt vorhanden, über das am Ende die Sound-Ausgabe gesteuert wird. Darin sind Eigenschaften wie Frequenz oder Bitrate des Audio-Signals gespeichert.

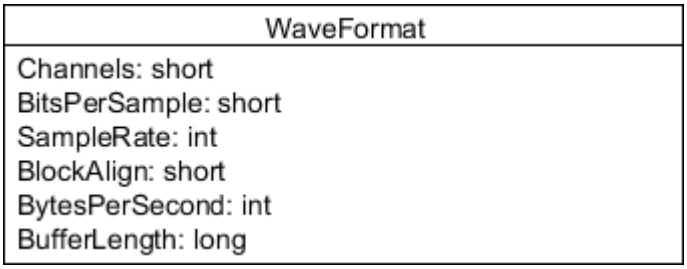

Abbildung 1: Struktur der WaveFormat-Klasse

Von dem abstrakten Datentyp Audio-Stream lassen sich nun weitere Klassen ableiten. Der Vorteil dieses Verfahrens ist, dass es für das Abspielen unwichtig ist, welches Musikformat vorliegt, da sich alle Objekte durch dieselbe Klasse darstellen lassen. Einzig beim Aufruf der verschiedenen Methoden ändert sich das Verhalten der Funktion. Auf diese Weise lassen sich einfach neue Algorithmen in das Programm implementieren, ohne etwas an der eigentlichen Struktur ändern zu müssen.

Eine abgeleitete Klasse initialisiert zunächst die WaveFormat-Struktur. Für den Fall, dass ein unkomprimierter PCM-Stream vorliegt, wird das Signal als ein Stereo-Signal mit einer Frequenz von 44,1 kHz interpretiert. Eine Weiterverarbeitung der Daten ist nicht nötig. Bei Dateien, die im Wave-Format vorliegen, müssen zunächst die gewünschten Informationen aus der Dateistruktur extrahiert werden[33]

Bei Verfahren, bei denen zunächst eine Umwandlung der Daten nötig ist, geschieht dies ebenfalls im Konstruktor der Klasse. So ist beispielsweise die Klasse MP3-Stream für die Dekomprimierung

[33] siehe Kapitel 2.2 auf Seite 4

einer MP3-Datei zuständig. Da wir es im gegebenen Zeitraum nicht schafften, die Dekodierung des Formates selbst vorzunehmen, griffen wir auf die Bibliothek MP3Sharp von Robert Burke[34] zurück. Diese stellt eine Portierung des Projekts JavaLayer[35] in die Sprache C# dar und beinhaltet einen vollständigen MP3-Dekompressor. Dadurch war es uns möglich, nach umfangreichem Studium die benötigten Klassen und Funktionen in unser Programm einzubinden. Die Bibliothek wird im Programm wie alle weiteren Bibliotheken in Form einer Anwendungserweiterung eingebunden.

Das Einlesen einer CD kann nicht auf direktem Wege geschehen. Um auf die Rohdaten, die auch das Audio-Signal beinhalten, zuzugreifen, verwendeten wir die Windows-API[36], die uns die benötigten Funktionen zur Verfügung stellte. Hierbei ergab sich das Problem, dass diese nur als native C-Bibliothek existiert und somit nicht direkt von einer Sprache wie C# angesprochen werden kann. Aus diesem Grund schrieben wir einen Satz an Container-Funktionen (engl. wrapper), die diesen Zugriff regelten. Auf diese Weise war es uns möglich, die benötigten Daten auszulesen.

Wird ein Audio-Stream auf einer CD angefragt, so wird zunächst wie in Kapitel 2.3 auf Seite 5 beschrieben das Inhaltsverzeichnis (TOC) eingelesen. Aus diesem liest das Programm verschiedene Informationen über die einzelnen Titel aus. Sofern der angefragte Titel gültig ist, wird die Leseposition im Speicher an die entsprechende Stelle der CD gesetzt und die Daten können anschließend weiter verarbeitet werden. Da diese unkomprimiert vorliegen, ist keine weitere Verarbeitung notwendig.

Weiterhin besitzt die Klasse CDDevice Hilfsfunktionen für den Umgang mit CDs. Beispielsweise kann sie alle möglichen optischen Laufwerke auflisten oder sie ein- bzw. ausfahren. Eine wichtige Funktion, die wir in unserem Programm implementierten, war die automatische Erkennung mit Hilfe der CDDB[37]. Dazu wird zunächst wie bereits beschrieben die CDDB-Id berechnet. Anschließend wird eine Verbindung zum Server der Seite freedb.org[38] aufgebaut und die Anfrage gesendet. Diese beinhaltet die berechnete Id, sowie die einzelnen Positionen der Musikstücke, um eine genauere Erkennung zu ermöglichen. Abhängig vom zurückgelieferten Statuscode reagiert das Programm nun auf verschiedene Wege:

[34] http://www.robburke.net/mle/mp3sharp/
[35] http://www.javazoom.net/javalayer/javalayer.html
[36] Application Programming Interface, Programmschnittstelle
[37] siehe Kapitel 3.2 auf Seite 11
[38] http://www.freedb.org/en/about_freedborg.2.html

Statuscode	Beschreibung	Vorgehensweise
200	Exakter Treffer gefunden	Der Treffer wird ausgewertet und zurückgegeben.
210 211	Mehrere Treffer gefunden	Jeder Treffer wird einzeln ausgewertet und als Liste mit mehreren Möglichkeiten zurückgeliefern
202	Kein Treffer gefunden	Eine leere Liste wird zurückgegeben.
403	Fehler der Datenbank	Die Kommunikation mit der Datenbank wird beendet und eine leere Liste zurückgegeben.
409	Problem bei der Kommunikation	

Auf diese Weise gefundene Informationen enthalten Titelnamen, Künstler, Album und Genre der CD. Sie können dann in die Musikbibliothek übernommen werden.

Für die letztendliche Ausgabe der Musik am Computer verwenden wir das von Microsoft bereit gestellte DirectX-Framework. Dieses stellt eine umfangreiche Bibliothek für die Erstellung von Grafik und Sound dar, wobei wir nur den Teil DirectSound benutzen. Da auch dieses Framework nur für C-Programme entwickelt wurde, griffen wir auf das Container-Framework SharpDX[39] zurück, welches kostenlos zur Verfügung steht. Angesprochen wird es im Programm von der Klasse SharpDXOutput, die sich von der Klasse WaveOutput ableitet. Die Basisklasse schreibt vor den Funktionsumfang vor, den eine Soundausgabe besitzen muss. Dazu gehören Abspielen, Pausieren, Setzen und Abfragen von Abspielposition sowie Lautstärke.

Um Musik über das DirectSound-Framework abzuspielen, muss zunächst eine Verbindung mit der passenden Schnittstelle (engl. device) hergestellt werden. Diese wird an das Programmfenster gebunden. Anschließend gilt es, einen Primär- und Sekundärbuffer zu initialisieren. Diese stellen einen Speicher zur Verfügung, in dem die unkomprimierten Musikdaten an die Soundkarte gesendet werden können. Über die bereits beschriebene WaveFormat-Struktur wird weiterhin mitgeteilt, was beim Abspielen der Daten zu beachten ist. Sind beide Buffer initialisiert, können die Daten aus dem Audio-Stream kopiert werden und sind bereit zum Abspielen.

[39] http://sharpdx.org/

5 Zusammenfassung

Zu Beginn unserer Seminarfacharbeit zum Thema „Audiosort - Erstellung eines frei konfigurierbaren Musikplayers unter Berücksichtigung verschiedener Audio-Formate" setzten wir uns das Ziel, einen vollwertigen Musikplayer zu entwickeln. Dabei wollten wir auf verschiedene Speicherungs- und Komprimierungsverfahren eingehen und diese selbstständig in unserem Programm umsetzen.

Zunächst galt es, sich in die Grundlagen einzuarbeiten. Nach einem ausführlichen Literaturstudium gelang es uns, erste Audiodateien zu dekodieren und abzuspielen. Um Programmlogik und Oberfläche zu trennen, entwickelten wir parallel an Musikplayer, Oberfläche und Datenbank. Auf diese Weise gelang es uns, gleichzeitig an einem Projekt zu arbeiten. Jedoch begannen an dieser Stelle die ersten Probleme. Die Bibliotheken, die wir zu Beginn für die Musikausgabe verwendeten, schienen inkompatibel mit unserer Benutzeroberfläche zu sein. Da wir die entstandenen Fehlermeldungen zunächst nicht korrekt deuten konnten, warf uns dies in unserem Zeitplan zurück.

Ein Ziel, welches wir uns zu Beginn gesetzt hatten, war das Konvertieren von Musikdateien. Allerdings ergaben sich hierbei zunächst rechtliche Probleme, da die Kodierung von MP3-Dateien speziellen Lizensen unterliegt. Weiterhin wäre es nicht möglich gewesen, diesen Programmteil im vorgegebenen Zeitrahmen fertig zu stellen, weshalb wir ihn aus unserer Arbeit entfernten. Gleiches gilt für die Online-Integration unseres Musikplayers in Plattformen wie Facebook oder Twitter. Auch hier mussten Abstriche gemacht werden.

Der MP3-Algorithmus, den wir zu Beginn unserer Seminarfacharbeit eigenständig implementieren wollten, erwies sich als zu umfangreich, um im gegebenen Zeitrahmen umgesetzt werden zu können. Aus diesem Grund mussten wir in Absprache mit unserem Fachbetreuer auf eine vorhandene Bibliothek zurückgreifen, die in das Programm eingebunden werden musste.

Die Zusammenarbeit in der Gruppe lief reibungslos. Durch eine geschickte Aufteilung des Programms und der theoretischen Aspekte sowie einem regelmäßigen Informationsaustausch gestaltete sich das Erstellen der Arbeit ohne Probleme für uns. Bei Problemen konsultierten wir unsere Fach- bzw. Seminarfachbetreuer und diskutierten eine eventuelle Lösung innerhalb der Gruppe.

Es gelang uns schnell, die benötigten Grundlagen zu erlernen. Nachdem wir uns in einfache Komprimierungsverfahren wie Wave oder die CD-Speicherung eingearbeitet hatten, gelang es uns, auch komplizierte Zusammenhänge wie den MP3-Codec zu erschließen. Gemeinsam erlernten wir komplexe Programmiertechniken der Sprache C# und damit Verbunden dem WPF-Framework für Benutzeroberflächen.

In der Zeit bis zum Kolloquium ist es unser Ziel, das Programm weiter auszubauen und um zusätzliche Funktionen zu erweitern.

6 Anhang

Weiterführende Abbildungen

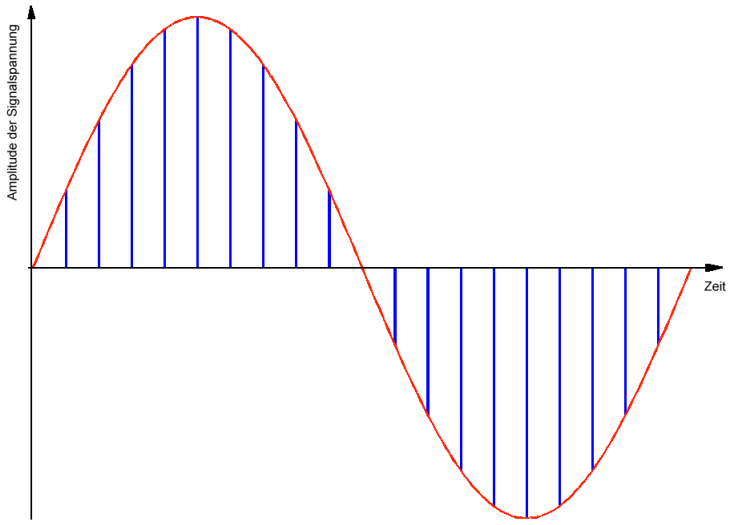

Abbildung 2: PAM moduliertes Signal

#	Name	Interpret	Album	Dauer	Track	Pfad	Genre	Jahr	Bewertung
0	1968		1968 (Live aus dem TIPI-Zelt am Kanzleramt Berlin)	0	1	D:\Musik\Rainald Grebe\1968 (Live aus dem TIPI-Zelt am Kanzleramt Berlin)\01 1968.mp3	Pop/Rock	2008	0
1	Es gibt kein richtiges Leben im falschen		1968 (Live aus dem TIPI-Zelt am Kanzleramt Berlin)	0	5	D:\Musik\Rainald Grebe\1968 (Live aus dem TIPI-Zelt am Kanzleramt Berlin)\05 Es gibt kein richtiges Leben im f.mp3	Pop/Rock	2008	0
2	Die Ratte		1968 (Live aus dem TIPI-Zelt am Kanzleramt Berlin)	0	6	D:\Musik\Rainald Grebe\1968 (Live aus dem TIPI-Zelt am Kanzleramt Berlin)\06 Die Ratte.mp3	Pop/Rock	2008	0
3	Der Pr?sident	Rainald Grebe	1968 (Live aus dem TIPI-Zelt am Kanzleramt Berlin)	0	7	D:\Musik\Rainald Grebe\1968 (Live aus dem TIPI-Zelt am Kanzleramt Berlin)\07 Der Präsident.mp3	(37)	2008	0
4	Als ich jung war		1968 (Live aus dem TIPI-Zelt am Kanzleramt Berlin)	0	8	D:\Musik\Rainald Grebe\1968 (Live aus dem TIPI-Zelt am Kanzleramt Berlin)\08 Als ich jung war.mp3	Pop/Rock	2008	0
5	Kann ich kann ich		1968 (Live aus dem TIPI-Zelt am Kanzleramt Berlin)	0	10	D:\Musik\Rainald Grebe\1968 (Live aus dem TIPI-Zelt am Kanzleramt Berlin)\10 Kann ich kann ich.mp3	Pop/Rock	2008	0
6	Begr??ung	Rainald Grebe	Das Abschiedskonzert	245773	1	D:\Musik\Rainald Grebe\Das Abschiedskonzert\1-01 Begrüßung.mp3	Comedy	2004	0
7	Das westdeutsche Reihenhaus	Rainald Grebe	Das Abschiedskonzert	156173	12	D:\Musik\Rainald Grebe\Das Abschiedskonzert\1-12 Das westdeutsche Reihenhaus.mp3	Comedy	0	0
8	Die Herkunft der Zimtstaschen	Rainald Grebe	Das Abschiedskonzert	55893	18	D:\Musik\Rainald Grebe\Das Abschiedskonzert\1-18 Die Herkunft der Zimtstaschen.mp3	Comedy	0	0
9	Arbeitslos in Gr?nland	Rainald Grebe	Das Abschiedskonzert	165200	2	D:\Musik\Rainald Grebe\Das Abschiedskonzert\2-02 Arbeitslos in Grönland.mp3	Comedy	2004	0
10	Fruchtfliegen	Rainald Grebe	Das Abschiedskonzert	14573	3	D:\Musik\Rainald Grebe\Das Abschiedskonzert\2-03 Fruchtfliegen.mp3	Comedy	0	0
11	Faust	Rainald Grebe	Das Abschiedskonzert	227573	4	D:\Musik\Rainald Grebe\Das Abschiedskonzert\2-04 Faust.mp3	Comedy	0	0
12	Ich bin ein Wochenendseminar	Rainald Grebe	Das Abschiedskonzert	211280	6	D:\Musik\Rainald Grebe\Das Abschiedskonzert\2-06 Ich bin ein Wochenendseminar.mp3	Comedy	2004	0
13	Hermann und Dorothea	Rainald Grebe	Das Abschiedskonzert	158400	8	D:\Musik\Rainald Grebe\Das Abschiedskonzert\2-08 Hermann und Dorothea.mp3	Comedy	0	0
14	Atlantik	Rainald Grebe	Das Abschiedskonzert	287666	11	D:\Musik\Rainald Grebe\Das Abschiedskonzert\2-11 Atlantik.mp3	Comedy	0	0
15	Blaues Blut	Rainald Grebe	Das Abschiedskonzert	266973	14	D:\Musik\Rainald Grebe\Das Abschiedskonzert\2-14 Blaues Blut.mp3	Comedy	0	0
16	Ich bin ein Indianer	Rainald Grebe	Das Abschiedskonzert	160695	15	D:\Musik\Rainald Grebe\Das Abschiedskonzert\2-15 Ich bin ein Indianer.mp3	Comedy	0	0
17	Apachenjunge Lukas	Rainald Grebe	Das Abschiedskonzert	63333	16	D:\Musik\Rainald Grebe\Das Abschiedskonzert\2-16 Apachenjunge Lukas.mp3	Comedy	2004	0
18	Drei Hymnen	Rainald Grebe	Das Abschiedskonzert	213146	17	D:\Musik\Rainald Grebe\Das Abschiedskonzert\2-17 Drei Hymnen.mp3	Comedy	2004	0
19	Alles was man tut ist gut	Rainald Grebe	Das Abschiedskonzert	234826	18	D:\Musik\Rainald Grebe\Das Abschiedskonzert\2-18 Alles was man tut ist gut.mp3	Comedy	0	0
20	Es ist gut	Rainald Grebe	Das Abschiedskonzert	230226	22	D:\Musik\Rainald Grebe\Das Abschiedskonzert\2-22 Es ist gut.mp3	Comedy	0	0
21	Freitag	Rainald Grebe	Das Robinson Crusoe Konzert	0	5	D:\Musik\Rainald Grebe\Das Robinson Crusoe Konzert\05 Freitag.mp3	Cult	2007	0
22	Bernd	Rainald Grebe	Das Robinson Crusoe Konzert	0	8	D:\Musik\Rainald Grebe\Das Robinson Crusoe Konzert\08 Bernd.mp3	Cult	2007	0
23	Hotel Mama	Rainald Grebe	Das Robinson Crusoe Konzert	0	12	D:\Musik\Rainald Grebe\Das Robinson Crusoe Konzert\12 Hotel Mama.mp3	Cult	2007	0
24	F?r wen mach ich das	Rainald Grebe	Das Robinson Crusoe Konzert	0	16	D:\Musik\Rainald Grebe\Das Robinson Crusoe Konzert\16 Für wen mach ich das.mp3	Cult	2007	0
25	ICE	Rainald Grebe	Rainald Grebe	230906	1	D:\Musik\Rainald Grebe\Rainald Grebe\01 ICE.mp3	Comedy	0	0
26	Beckenbauer	Rainald Grebe	Rainald Grebe	180200	2	D:\Musik\Rainald Grebe\Rainald Grebe\02 Beckenbauer.mp3	Comedy	0	0
27	Guido Knopp	Rainald Grebe	Rainald Grebe	224573	3	D:\Musik\Rainald Grebe\Rainald Grebe\03 Guido Knopp.mp3	Comedy	0	0
28	Eberhard und Emma	Rainald Grebe	Rainald Grebe	21666	8	D:\Musik\Rainald Grebe\Rainald Grebe\08 Eberhard und Emma.mp3	Comedy	0	0
29	Er und Sie	Rainald Grebe	Rainald Grebe	320920	9	D:\Musik\Rainald Grebe\Rainald Grebe\09 Er und Sie.mp3	Comedy	0	0
30	Brandenburg	Rainald Grebe	Rainald Grebe	231375	11	D:\Musik\Rainald Grebe\Rainald Grebe\11 Brandenburg.mp3	Comedy	0	0
31	Die Fete	Rainald Grebe	Rainald Grebe	654426	13	D:\Musik\Rainald Grebe\Rainald Grebe\13 Die Fete.mp3	Comedy	0	0
32	Castingallee	Rainald Grebe	Volksmusik	0	5	D:\Musik\Rainald Grebe\Volksmusik\05 Castingallee.mp3	Comedy	2007	0
33	Drei?ig?hrige P?rchen	Rainald Grebe	Volksmusik	0	7	D:\Musik\Rainald Grebe\Volksmusik\07 Dreißigjährige Pärchen.mp3	Comedy	2007	0

Abbildung 3: Musikbibliothek mit angewandtem XSL-Stylesheet

Detail of an MP3 Header

Bits	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Binary	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	0
Hex	F				F				F					B			A			
Meaning	MP3 Sync Word												Version	Layer		Error Protection	Bit Rate			
Value	Sync Word												1 = MPEG	01 = Layer 3		1 = No	1010 = 160			

21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	1	0	0	0	0	0	0
		0				4				0	
Frequency		Pad. Bit	Priv. Bit	Mode		Mode Extension (Used With Joint Stereo)		Copy	Original	Emphasis	
00 = 44100 Hz		0 = Frame is not padded	Unknown	01 = Joint Stereo		0 = Intensity Stereo Off	0 = MS Stereo Off	0 = Not Copy-righted	0 = Copy Of Original Media	00 = None	

Abbildung 4: Struktur eines MP3-Frames

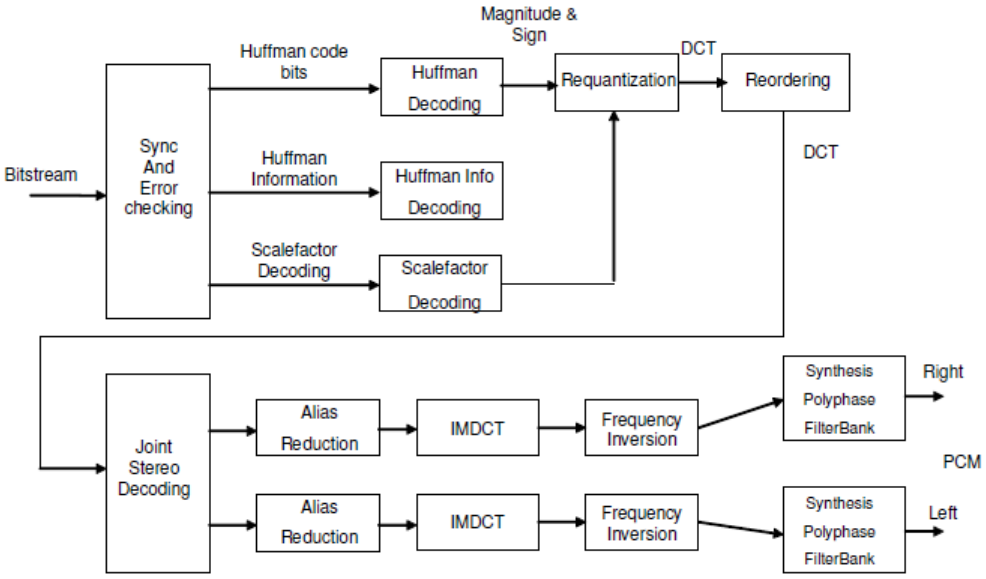

Abbildung 5: Schema der MP3-Kompression

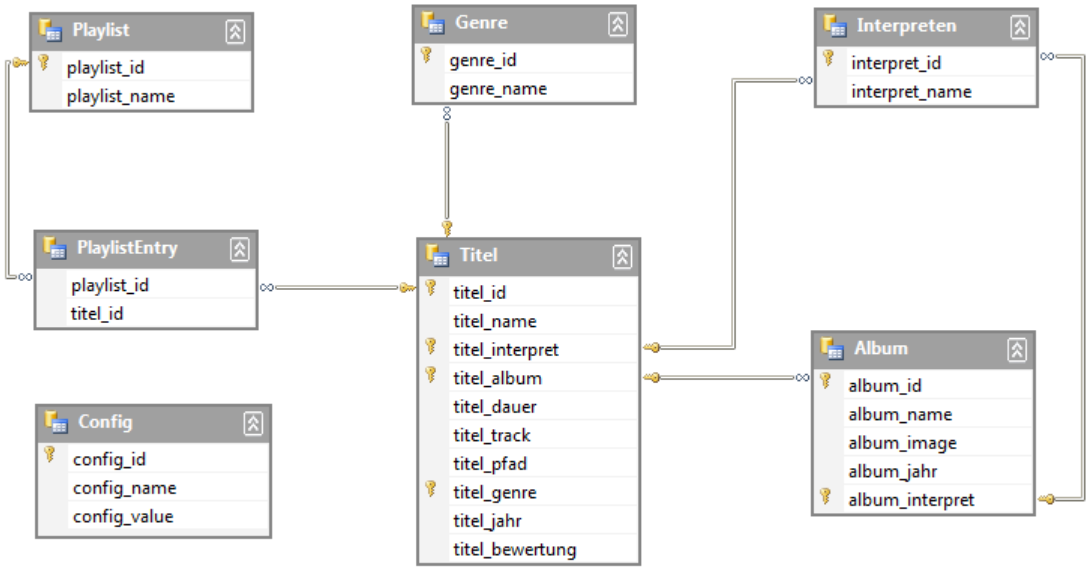

Abbildung 6: Datenbankschema unserer Musikbibliothek

Audiosort

Datei Ansicht Tools Hilfe

Titel	Interpret	Album	Länge	Track	Genre	Jahr	Bewertung	Pfad
1968		1968 (Live aus	0	1	Pop/Rock	2008	0	D:\Musik\Rainald Gre
Es gibt kein ric		1968 (Live aus	0	3	Pop/Rock	2008	0	D:\Musik\Rainald Gre
Die Ratte		1968 (Live aus	0	6	Pop/Rock	2008	0	D:\Musik\Rainald Gre
Der Pr?sident	Rainald Grebe	1968 (Live aus	0	7	(57)	2008	0	D:\Musik\Rainald Gre
Als ich jung wa		1968 (Live aus	0	8	Pop/Rock	2008	0	D:\Musik\Rainald Gre
Kenn ich kann		1968 (Live aus	0	10	Pop/Rock	2008	0	D:\Musik\Rainald Gre
Begr??ung	Rainald Grebe	Das Abschiedsl	245773	1	Comedy	2004	0	D:\Musik\Rainald Gre
Das westdeuts	Rainald Grebe	Das Abschiedsl	156173	12	Comedy	0	0	D:\Musik\Rainald Gre
Die Herkunft d	Rainald Grebe	Das Abschiedsl	55893	18	Comedy	0	0	D:\Musik\Rainald Gre
Arbeitslos in G	Rainald Grebe	Das Abschiedsl	165200	2	Comedy	2004	0	D:\Musik\Rainald Gre
Fruchtfliegen	Rainald Grebe	Das Abschiedsl	14573	3	Comedy	0	0	D:\Musik\Rainald Gre
Faust	Rainald Grebe	Das Abschiedsl	227573	4	Comedy	0	0	D:\Musik\Rainald Gre
Ich bin ein Wo	Rainald Grebe	Das Abschiedsl	211280	6	Comedy	2004	0	D:\Musik\Rainald Gre
Hermann und I	Rainald Grebe	Das Abschiedsl	158400	8	Comedy	0	0	D:\Musik\Rainald Gre
Atlantik	Rainald Grebe	Das Abschiedsl	287666	11	Comedy	0	0	D:\Musik\Rainald Gre
Blaues Blut	Rainald Grebe	Das Abschiedsl	266973	14	Comedy	0	0	D:\Musik\Rainald Gre
Ich bin ein Indi	Rainald Grebe	Das Abschiedsl	160693	15	Comedy	0	0	D:\Musik\Rainald Gre
Apachenjunge	Rainald Grebe	Das Abschiedsl	63333	16	Comedy	2004	0	D:\Musik\Rainald Gre
Drei Hymnen	Rainald Grebe	Das Abschiedsl	213146	17	Comedy	2004	0	D:\Musik\Rainald Gre
Alles was man	Rainald Grebe	Das Abschiedsl	234826	18	Comedy	0	0	D:\Musik\Rainald Gre
Es ist gut	Rainald Grebe	Das Abschiedsl	230226	22	Comedy	0	0	D:\Musik\Rainald Gre
Freitag	Rainald Grebe	Das Robinson (	0	5	Cult	2007	0	D:\Musik\Rainald Gre
Bernd	Rainald Grebe	Das Robinson (	0	8	Cult	2007	0	D:\Musik\Rainald Gre
Hotel Mama	Rainald Grebe	Das Robinson (	0	12	Cult	2007	0	D:\Musik\Rainald Gre
F?r wen mach i	Rainald Grebe	Das Robinson (	0	16	Cult	2007	0	D:\Musik\Rainald Gre
ICE	Rainald Grebe	Rainald Grebe	230906	1	Comedy	0	0	D:\Musik\Rainald Gre

Zurück Play Stop Weiter

Abbildung 7: Benutzeroberfläche ohne Style-Informationen

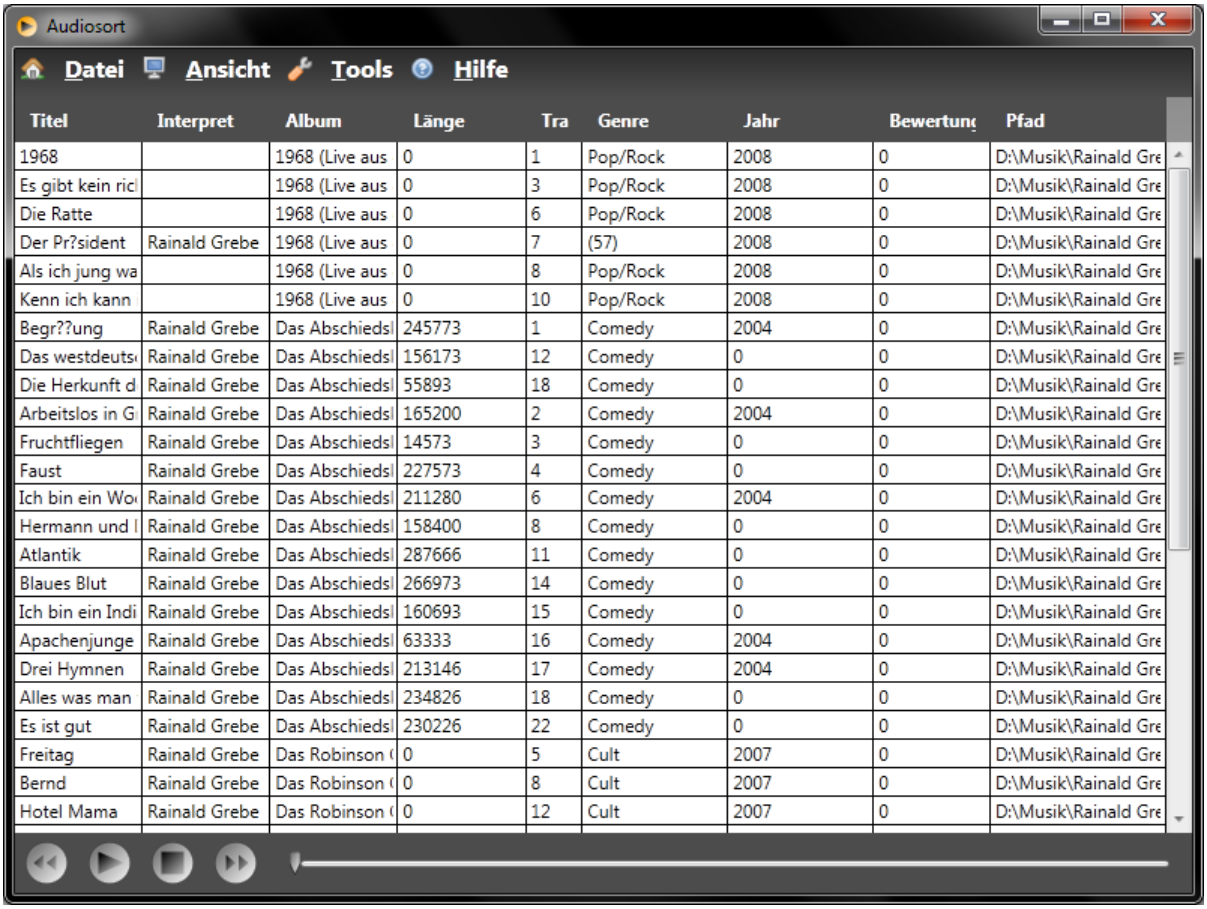

Abbildung 8: Benutzeroberfläche mit geladenem Design

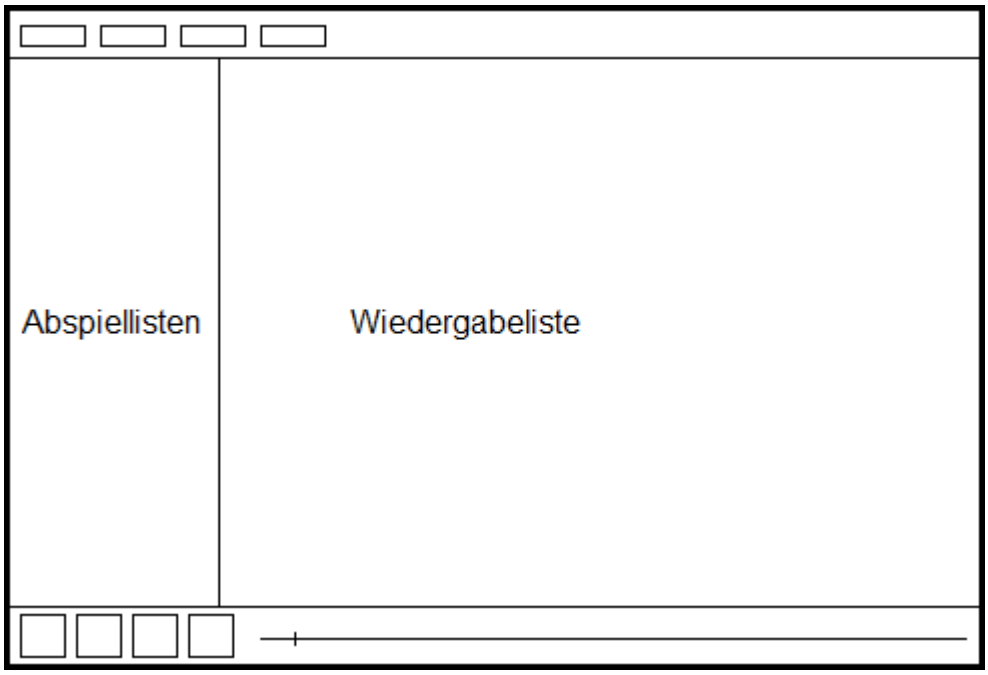

Abbildung 9: Konzeptskizze unserer Benutzeroberfläche

7 Literatur- und Abbildungsverzeichnis

Literatur

[1] *AD-Wandler*. http://www.mikrocontroller.net/articles/AD-Wandler, Abruf: August 2012

[2] *Audio CD (CD-DA) Data Structure*. http://www.herongyang.com/CD-DVD/Audio-CD-Data-Structure.html, Abruf: Dezember 2012

[3] *avlab.de: MP3*. http://www.avlab.de/audio-codecs-container/mp3.html, Abruf: November 2012

[4] *CDDB.pm - a high-level interface to cddb protocol servers (freedb and CDDB)*. http://search.cpan.org/~rcaputo/CDDB-1.220/lib/CDDB.pm, Abruf: September 2012

[5] *Eight-to-Fourteen-Modulation*. http://de.wikipedia.org/wiki/Eight-to-Fourteen-Modulation, Abruf: Dezember 2012

[6] *Einführung in WPF*. http://msdn.microsoft.com/de-de/library/aa970268.aspx#Y1710, Abruf: Dezember 2012

[7] *FH Düsseldorf: WAV-Format*. http://swlab.et.fh-duesseldorf.de/pc_pool/lernmodule/multimediadateien/Kapitel33.htm, Abruf: November 2011

[8] *freedb: Howto*. http://ftp.freedb.org/pub/freedb/misc/freedb_howto1.07.zip, Abruf: September 2012

[9] *freedb-Website*. http://www.freedb.org/en/, Abruf: September 2012

[10] *How MP3 Files Work*. http://computer.howstuffworks.com/mp31.htm

[11] *ID3 tag version 2.4.0 - Main Structure*. http://id3.org/id3v2.4.0-structure, Abruf: Oktober 2012

[12] *ID3 tag version 2.4.0 - Native Frames*. http://id3.org/id3v2.4.0-frames, Abruf: Oktober 2012

[13] *ITWissen: EFM-Code*. http://www.itwissen.info/definition/lexikon/eight-to-fourteen-modulation-EFM-EFM-Code.html, Abruf: Dezember 2012

[14] *Mikrofon*. http://de.wikipedia.org/wiki/Mikrofon, Abruf: August 2012

[15] *MP3 and AAC: MDCT Processing*. http://cnx.org/content/m32113/latest/, Abruf: November 2012

[16] *MP3 Filestructure*. http://en.wikipedia.org/wiki/MP3#File_structure, Abruf: November 2012

[17] *Perceptual Coding: How Mp3 Compression Works*. http://www.soundonsound.com/sos/may00/articles/mp3.htm, Abruf: November 2012

[18] *Puls-Code-Modulation*. http://www.elektronik-kompendium.de/sites/kom/0312281.htm, Abruf: August 2012

[19] *Puls-Code-Modulation*. http://de.wikipedia.org/wiki/Puls-Code-Modulation, Abruf: August 2012

[20] *Pulsamplitudenmodulation*. http://de.wikipedia.org/wiki/Pulsamplitudenmodulation, Abruf: August 2012

[21] *Quantisierung*. http://de.wikipedia.org/wiki/Quantisierung, Abruf: August 2012

[22] *RIFF WAVE*. http://de.wikipedia.org/wiki/RIFF_WAVE, Abruf: November 2011

[23] *Rob Ryan: RIFF WAVE (.WAV) file format*. http://www.neurophys.wisc.edu/auditory/riff-format.txt

[24] *The Theory Behind Mp3*. http://www.mp3-tech.org/programmer/docs/mp3_theory.pdf, Abruf: November 2012

[25] *Thorsten Scheuermann: MP3-Audiokompression*. http://goethe.ira.uka.de/seminare/redundanz/vortrag14/#quantisierung, Abruf: November 2012

[26] *University of California: Specification and Design of a MP3 Audio Decoder*. http://www.cecs.uci.edu/~cecs/technical_report/TR05-04.pdf, Abruf: November 2012

[27] *WAVE PCM soundfile format*.
`https://ccrma.stanford.edu/courses/422/projects/WaveFormat/`, Abruf: November 2011

[28] *Windows Presentation Foundation (WPF)*.
`http://msdn.microsoft.com/de-de/vstudio/aa663326.aspx`, Abruf: Dezember 2012

Abbildungen

[Abbildung 1] Im Privatbesitz von: Daniel Birnstiel/Max Ulbrich

[Abbildung 2] http://de.wikipedia.org/w/index.php?title=Datei:PAM-Prinzip.png

[Abbildung 3] Im Privatbesitz von: Daniel Birnstiel/Max Ulbrich

[Abbildung 4] http://en.wikipedia.org/wiki/File:Mp3filestructure.svg

[Abbildung 5] http://www.cecs.uci.edu/∼cecs/technical_report/TR05-04.pdf

[Abbildung 6] Im Privatbesitz von: Daniel Birnstiel/Max Ulbrich

[Abbildung 7] Im Privatbesitz von: Daniel Birnstiel/Max Ulbrich

[Abbildung 8] Im Privatbesitz von: Daniel Birnstiel/Max Ulbrich

[Abbildung 9] Im Privatbesitz von: Daniel Birnstiel/Max Ulbrich

Eidesstattliche Erklärung

Hiermit versichern wir an Eides statt, dass wir diese Seminarfacharbeit selbstständig und nur unter Zuhilfenahme der angegebenen Quellen erstellt haben.

Erfurt, den 21.12.2012 Unterschriften: