„Krachmacher“

Technischer Entwurf

Informatik-Praktikum im Grundstudium

Sommersemester 2004

Gruppe 2

Tutor: Aaron Ruß

Inhaltsverzeichnis

Einleitung:

Es soll eine Software programmiert werden, mit der Musikdateien verwaltet und abgespielt werden können. In einer erweiterten Version, deren Realisierung nicht gesichert ist, sollen auch Videodateien verwaltet und gespielt werden können.
Die in dem Analysedokument beschriebenen Anforderungen werden in zwei Stufen realisiert:
1. Spezifikation: Zuerst wird eine Aufteilung der Anforderungen nach Komponenten vorgenommen, und für diese Komponenten werden Schnittstellen spezifiziert.
2. Konstruktion: Danach erfolgt die Konstruktion der Komponenten selbst. Die Programmierung wird auf Teams aufgeteilt, die jeweils für eine Komponente zuständig sind. Es werden entsprechend dem Versionsplan schrittweise die Anforderungen erfüllt.
Das vorliegende Dokument beschreibt die Planung der Spezifikation und Konstruktion, den technischen Aufbau der Software, sowie ihren inneren Zusammenhalt anhand der definierten Schnittstellen nach innen, zu den Komponenten, und nach außen, zur Software-Umgebung.
Der Schwerpunkt des Entwurfs liegt in der Planung und Realisierung der Basisversion, die bereits ausreichende Benutzer-Funktionalität haben soll. Für die Basisversion sind Testläufe vorgesehen, deren Definition hier zunächst nur allgemein geschieht.

Zentrale Konzepte des Entwurfs:

1. Modularisierung: Das zu realisierende Programm besteht aus mehreren Komponenten, die ihrerseits wieder aus Moduln bestehen können. Es gibt eine zentrale, gemeinsam genutzte Komponente, welche zur Laufzeit nur genau einmal vorkommen darf: der "Controller". Alle anderen Komponenten kommunizieren miteinander nur über die Botschaftenverwaltung des Controllers. Diese Aufteilung erlaubt es zur Laufzeit Komponenten auszutauschen, mehrere Komponenten gleichen Typs zu nutzen (z.B. mehrere Playlists oder mehrere GUIs), und zusätzliche Komponenten zu schreiben. Die Beschreibung von zusätzlichen Komponenten ist nicht Bestandteil dieses Dokuments, aber man könnte z.B. eine Steuerung des Programms über die Kommandozeile verwirklichen.
2. Für die Konstruktion verwendete fremde Bibliotheken sind portabel oder sind auf allen relevanten Zielsystem als äquivalente Implementierung vorhanden.
3. Die graphische Oberfläche gewährleistet die Kommunikation mit dem Benutzer; jede weitere Funktionalität ist möglichst in die anderen Komponenten verlagert.

Versionsplanung:

Es sind drei Versionen geplant.

1. Basisversion:
   - Unterstützung der Audio-Formate OGG, MP3, WAV
   - Abspielen der unterstützten Formate
   - Verwaltung von Informationen zu Musikstücken (etwa Kategorien: Dateiname, Komponist, Interpret, Genre, Jahr der Aufnahme, und viele weitere)
   - Katalogisieren vorhandener lokaler Datenbestände (Einlesen aus dem Dateisystem)
   - Automatisches Lesen von Informationen aus den Dateien (sofern das Format geeignete Daten enthält) bzw. aus ihren Namen beim Einlesen
   - Persistente Speicherung aller Daten in einem geeigneten Format
   - Sortierte Anzeige der gespeicherten Daten nach verschiedenen Kriterien
   - Effiziente Suchfunktion unter allen gespeicherten Informationen nach verschiedenen Kriterien
   - Zusammenstellen, Verwalten und Abspielen eigener Abspiellisten
   - Abspeichern und Laden eigener Abspiellisten
   - ansprechende und übersichtliche graphische Benutzeroberfläche mit Mausbedienung

2. Increment 1: Mit folgenden Erweiterungen:
   - Equalizer (Anpassen des Frequenzspektrums)
   - Unterstützung zusätzlicher Audio-Formate

3. Increment 2: Mit folgenden Erweiterungen:
   - Mixer (Mischen von zwei oder mehr Dateien zu einer neuen Multimedia-Datei)
   - Karaokefilter (Filterung von Stimmen)
   - Abspielen von Videoformaten (DivX, MPEG, eventuell weitere)
     (Wobei bis Ende des Semesters nur Increment 1 fertig gestellt werden wird)

Während der Konstruktion wird es mehrere Versionen geben, bevor die Anforderungen der Basisversion erreicht werden können.
Die einzelnen Versionsplanungen der einzelnen Gruppen werden nun aufgeführt:

Versionsplanung der Datenverwaltung:

V1:
- Implementierung aller Schnittstellen zu den Komponenten mit Stubs (Dummy-Funktion).
- Definition der Datentypen und Klassen, die von den äußeren Schnittstellen verwendet werden.
- Dummy-Funktionen zur JDBC-Schnittstelle.

V2:
- Suchanfragen nach einem String und in einem Feld der Datenbank (Spalte einer Tabelle) werden durchgeführt. Die Ergebnisse werden der Playlist/GUI in einer Botschaft zurückgegeben. Die Suchanfragen werden intern konvertiert um sie mit der JDBC-Schnittstelle zu verwenden, und dann darüber an die Datenbank weitergegeben.
- Anhand eines Strings für die URL einer Multimedia-Datei wird überprüft, ob diese URL bereits in der Datenbank vorhanden ist.
- Multimedia-Datei-Einträge in der Datenbank können eingefügt, gelöscht werden. Beim Einfügen wird zuerst überprüft ob die URL bereits vorhanden ist (Suche, siehe oben), und wenn nicht, dann wird bei der Datenverarbeitungs-Komponente angefragt (Botschaft) ob unter der URL eine gültige Datei vorhanden ist, und wenn ja, dann werden die in dieser Datei enthaltenen Informationen zu Titel und Jahr zurückerwartet. Diese Informationen werden mit der URL und einer generierten ID in die Tabelle der Datenbank eingetragen. Die Lied-ID wird statisch erhöht (IDs gelöschter Einträge bleiben ungenutzt, bis die Zahlen wieder bei 0 ankommen).
- Die Datenbank-Tabelle besitzt Spalten für Lied-ID, URL, Titel, Jahr. Der Inhalt dieser Einträge wird entsprechend Anfragen (Botschaften) der Playlist/GUI übermittelt.

V3: (Basisversion)
- Die Tabellen der Datenbank sind vollständig entsprechend dem Datenbank-Schema.
- Operationen auf der Datenbank über JDBC verknüpfen mehrere Tabellen.
- Verknüpfte Suchanfragen sind möglich.
- Sortieren nach mehreren Kategorien (Feldern) ist möglich.
- Änderungen der Einträge können vorgenommen werden. Dazu wird einerseits die Datenbank aktualisiert und andererseits die Datenverarbeitung informiert, daß die betreffende Datei geändert werden soll. Die auslösenden Botschaften werden beantwortet (Erfolg, Fehler, Multimedia-Datei verändert). Für die Änderung der Informationen in einer Multimedia-Datei wird eine Botschaft an die Datenverarbeitung gesendet, und eine Rückmeldung (Erfolg, Fehler) erwartet.
- Aktualisierungen und Änderungen in der Datenbank, werden anhand der betroffenen IDs an die Playlist übermittelt.

- Anlage einer neuen Datenbank (hier ist das erstmalige Füllen der Tabellen mit Einträgen gemeint. dazu Botschaft and Datenverarbeitung: rekursiv in einem gegebenen Pfad nach Multimedia-Dateien suchen und URLs und auch die Informationen aus den Dateien an Datenverwaltung übermitteln).

Versionsplanung des Players/ der Datenverarbeitung:

V1:
- Implementierung der rudimentären Steuerelemente für die Multimedia-Datei. Es wird möglich sein die Datei zu starten und sie zu stoppen.
- Zudem kann man den Pfad einer Multimedia-Datei setzen, eine einzigartige Identifikation zu ihr anlegen und überprüfen, ob die gewünschte Datei überhaupt vorhanden ist.
- Außerdem wird es möglich sein ein MMF zu erstellen, welches man öffnen, schließen, lesen und schreiben kann. Diese Klasse ist mit einem Datenstream zu vergleichen und ist das Element, welches an die Soundkarte o.ä. geht

V2:
- In dieser Version werden zusätzlich zu den schon vorhandenen Methoden noch das Pausieren und wieder Aufnehmen des Liedes.
- Es wird in dieser Version auch möglich sein die Lautstärke einzustellen oder auch einfach nur den Ton auszustellen.
- Auch bekommt man nun die Möglichkeit das nächste und vorherige Lied abzuspielen. Dies wird für die Playlist benötigt.

V3: (Basisversion)
- Zur nun vorliegenden Basisversion sollen auch die Schnittstellenmethoden zur Datenverwaltung implementiert werden, mit deren Hilfe z.B. die Playlist die Tag-Informationen auslesen und bearbeiten kann.
- Es wird auch z.B. der Datenverwaltung die Möglichkeit gegeben Textdateien zu lesen und zu schreiben. Auch soll eine Methode zur Verfügung gestellt werden die es der Datenverwaltung ermöglicht ein Verzeichnis nach Multimedia-Dateien zu durchsuchen.

V4: (Increment 1)
- Nun soll der Equalizer implementiert werden.
- Außerdem soll zur Verfügung gestellt werden das Lied vor- und zurückzuspulen und die Position ab der man das Lied hören möchte anzugeben.

V5: (Increment 2)
- Implementierung der Cutting-Funktion, die es ermöglichen soll eine große Multimedia-Datei in mehrere kleine (durch Pausenerkennung oder manuell) zu schneiden

Versionsplanung des Controllers:

V1:
- einfaches Grundkonzept implementieren: Austausch bzw. Verteilung von Nachrichten

V2:
- Verbessertes Ressourcenmanagement, Implementierung eines differenzierteren Nachrichtensystems (Verteilung von Nachrichten an nur bestimmte Komponenten einer Gruppe)

V3: (Basisversion)
- Dynamisches Laden von Komponenten

Versionsplanung der Playlist:

V1:
- Stubs aller Schnittstellen-Funktionen

V2:
- Anlegen einer neuen leeren Playlist.
- Weiterleitung von Anfragen zur Änderung von Einträgen, und von Suchanfragen (Such-String, Suche in einem Feld), an die Datenverwaltung.
- Lieferung von formatierten Text-Feldern an die GUI zur Darstellung der Playlist.

V3: (Basisversion)
- Mitteilung an die GUI, daß sich Informationen der Playlist geändert haben.
- Botschaft an Datenverwaltung (oder Datenverarbeitung?) zum Laden, Speichern von Playlists.
- Hinzufügen von Multimedia-Einträgen zu der Playlist.
- Sortieren (von Teilen) der Playlist.
- Weiterleiten von allen Suchanfragen, die die Datenverwaltung unterstützt.

Versionsplanung der GUI:

V1:
- vorläufiges Layout ohne Funktionen (Prototyp)

V2:
- Bedienung des Players: Start, Stopp, Pause, SkipNext, SkipPrev
- Bedienung der Playlist: Anzeigen, abzuspielenden Titel auswählen
- Bedienung der Datenbank: einfache Suchanfragen
- Aussehen: funktional

V3: (Basisversion)
- Vollständige Unterstützung der Komponentenfunktionen
- Player: Anzeigen von Titelinformationen, Zeitinformationen, Datei öffnen
- Playlist: Bearbeiten, umsortieren (Sortierfunktionen), …
- Datenbank: optional Bearbeitung von allen anderen Datenbankobjekten (Interpret, Alben,…)

Installation und Integration:
Es wird ein Installationsskript geschrieben. Nach Möglichkeit wird das Installationskript plattformunabhängig verwirklicht. Nur wenn sich dies bei der technischen Ausarbeitung als unmöglich erweist, werden plattform-spezifische Installationsskripte verwendet.
Das Installationsskript überprüft die notwendigen Voraussetzungen für eine lauffähige Installation und informiert den Benutzer über Abweichungen. Vor allem für Benutzer mit wenig Erfahrung mit Computern ist diese Unterstützung bei der Installation wichtig.

0. Installations-Pfade werden dem jeweiligen Betriebssystem angepasst, funktionieren also auf allen Systemen gleich.
1. Die installierte Java-Umgebung wird abgefragt. Wenn kein Java vorhanden ist, oder wenn die Java-Version zu alt ist, oder wenn geforderte Bibliotheken fehlen, wird der Benutzer über die fehlenden Voraussetzungen informiert.
2. Es wird geprüft ob eine für die Software verwendbare Datenbank installiert ist, und ob die Eigenschaften dieser Datenbank eine fehlerfreie Nutzung durch die Software erlaubt.
   Sind diese Voraussetzungen nicht erfüllt, wird dem Benutzer weit reichende Information anhand gegeben, damit er in die Lage versetzt wird, eine Datenbank selbst zu installieren. Nach der Installation ist die reibungslose Interaktion der Datenbank mit der Software gewährleistet.

3. In der erweiterten Version des Programms kann es notwendig werden, auf eine vorhandene Installation des Programms zu prüfen, und dann entsprechend der Situation Änderungen an dieser Installation vorzuschlagen. Auch eine bereits vorhandene Datenbank wird entsprechend den Anforderungen der neuen Software geprüft und es wird angeboten, eine alte Version der Datenbank an die neuen Erfordernisse anzupassen.

Einordnung des SW-Systems:

Das von uns anvisierte Programm soll ein plattformunabhängiger und kostenloser MultimediaPlayer werden. Zusätzlich soll er erweiterbar sein. Die Besonderheit des Programms liegt in seiner Fähigkeit die abspielbaren Multimedia-Dateien auch zu verwalten. Suchen, Sortieren und Ändern von Dateiinformationen in einer Datenbank werden unterstützt.

Schnittstellenbeschreibung:

Um eine hohe Erweiterbarkeit unseres Programms zu gewährleisten, wurde ein Controller-Konzept ausgearbeitet, was es ermöglicht Software-Komponenten anderer Programmierer leicht einzubinden.
Zudem soll es einige Möglichkeit geben externe Hardware an den Computer und an das Programm anzuschließen, geplant sind sowohl die Möglichkeit Input von externen CD-Playern zu bekommen als auch ein externes Bedienteil für den Player anzuschließen (z.B. Mischpult).
Zum Schluss ist auch angedacht zwei Audio Ausgänge des Computers getrennt ansteuern zu können.

Benutzerschnittstellen:

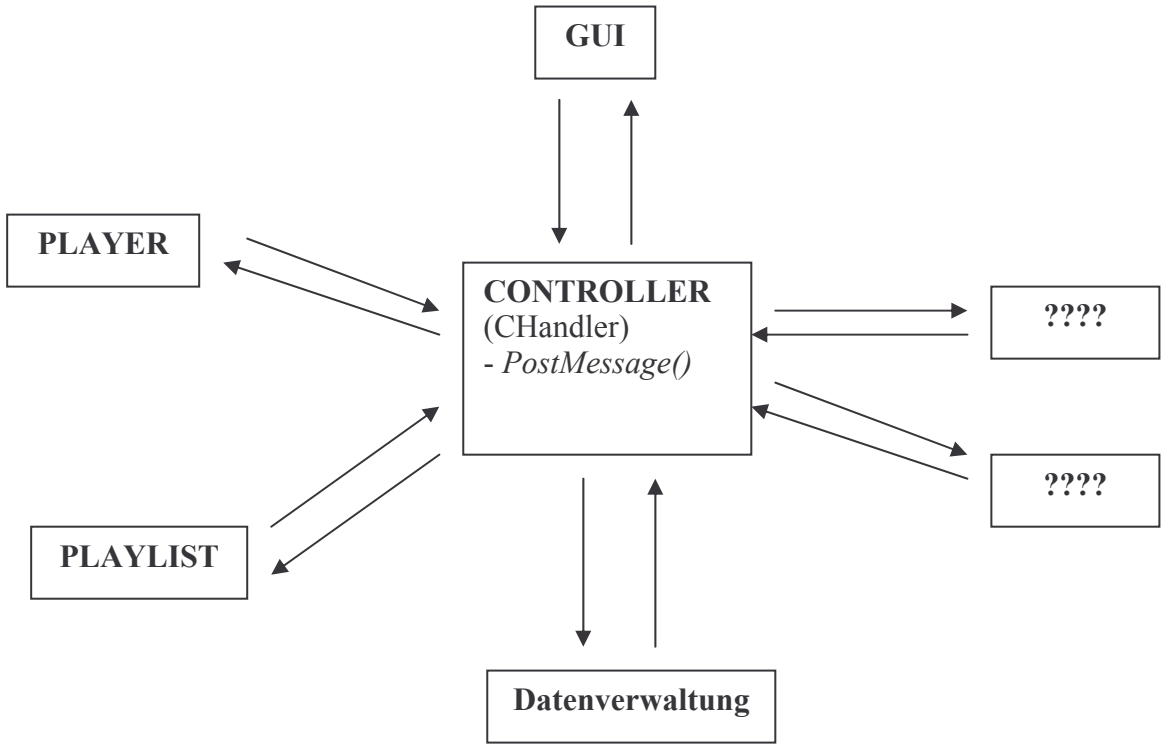
Da unser Programm ein Player zum Abspielen von Multimedia-Dateien ist, gibt es genau eine Schnittstelle zum Benutzer, nämlich die GUI. Der Benutzer interagiert nur mit der GUI. Die GUI wird verschiedene Arten von Knöpfen und Zeigern zur Verfügung stellen um dem Benutzer eine intuitive Bedienung des Players zu ermöglichen. Die Suchanfragen werden mit Hilfe von Suchmasken durchgeführt. Es wird versucht dem Benutzer eine schon von anderen vertrauten Umgebungen leicht zu bedienende Oberfläche anzubieten.

Der Benutzer wird die Möglichkeit haben alle Standardfälle eines Multimedia-Players auszuführen. Zu den Standardinteraktionen gehören sowohl die Wiedergabe einer Multimedia-Datei, deren Pausieren, deren Stoppen und das Vor- und Zurückspulen innerhalb der Datei. Somit wäre die gröbste Multimediadateisteuerung erwähnt. Da wir auch eine Playlist implementieren und die Dateien auch Eigenschaften haben, wird dem Benutzer auch die Möglichkeit gegeben die Eigenschaften der Dateien angezeigt zu bekommen und diese auch entsprechend zu verändern. Die Playlist stellt nun allerdings die Möglichkeiten Multimedia-Dateien zu wechseln (man kann also zum vorherigen Lied oder zum Nachfolgenden wechseln)(skip) zur Verfügung, sowie auch die Möglichkeit eine Multimedia-Datei der Playlist direkt auszuwählen und abzuspielen. Natürlich wird auch die Möglichkeit vorhanden sein Einträge in die Playlist einzufügen und zu entfernen. Die Anordnung der Playlist kann manuell oder automatisch nach bestimmten Kriterien erfolgen. Über die Playlist kann auch der Wiedergabemodus eingestellt werden, z.B. Wiederholung(repeat), Zufall(random). Die Playlist kann vom Benutzer neu erstellt, gespeichert und geladen werden. Als letztes soll dem Benutzer die Möglichkeit gegeben werden per Drag&Drop Dateien in die Playlist zu übernehmen.
Der Benutzer kann auch einen Filter, z.B. Karaoke, für die Multimedia-Datei auswählen, dass Frequenzspektrum anpassen und eine Multimedia-Datei schneiden. Als letztes soll der Benutzer auch die Datenbank erweitern können und die vorhandene exportieren können.

Basissystem-Schnittstellen:
Nur die Datenverwaltungs-Komponente benutzt als Schnittstelle zur HSQL-Datenbank das JDBC.
Nur die Player/Datenverarbeitungs-Komponente nutzt JavaSound-Bibliotheken.
Das Gesamtprojekt hat als Schnittstelle zum Betriebssystem die JavaVirtualMachine.
Insgesamt sind dieses plattformunabhängige Schnittstellen.

Konstruktion des SW-Systems:

Überblick über die Zerlegung in Komponenten. Die Fragezeichen stellen weitere mögliche Komponenten dar.



Anforderungen an die Komponenten:
Jede Komponente muss das Protokoll vom Controller implementieren und die Anfragen bearbeiten um das entsprechende Ergebnis zurückzuliefern.
(siehe Schnittstellendefinition)

Ablaufstruktur:
Um auf die einfachen Abspielfunktionen des Players zugreifen zu können, stehen leicht erreichbare, große Schaltflächen zur Verfügung. Will der Benutzer spezielle Informationen eines Datenbankobjekts ((aktuelles) Musikstück, Interpret,…) angezeigt haben, dann stehen je nach Information Ordnungs- Anzeigeoptionen in verschachtelten Menüs in den jeweiligen Fenstern zur Verfügung.
Das Playerfenster zeigt im Erweiterungsmodus die aktuelle Abspielposition sowie weitere erwünschte Informationen an. Diese werden über ein einblendbares Optionsfenster eingestellt. Durch kleine Buttons können die anderen Fenster hinzugeschaltet werden.
Die Playlist kann vom Benutzer in ihrer Form als Tabelle durch Drag and Drop umstrukturiert werden. Spalten und Zeilen können vertauscht indem man sie am Rand der Tabelle an ihrer Bezeichnung mit dem Mauscursor aufnimmt und an der gewünschten Stelle fallen lässt.

Weiterhin können sie aus- bzw. eingeblendet werden und der angezeigte Inhalt kann durch Menüs eingestellt werden. Der aktuell abzuspielende Titel wird mit einem Doppelklick auf einen Eintrag neu festgelegt.
Über ein GUI-Fenster, das der Datenbank zugeordnet ist, können Eingabemasken angefordert werden, die es dem Benutzer erlauben Datenbankobjekte zu modifizieren. Dieses Fenster wird wie die Playlist Drag and Drop- fähige Tabellen enthalten, die sich ebenso vielseitig durch Schaltflächen für die oft gebrauchten und verschachtelte Menüs für selten gebrauchten Möglichkeiten zusammenstellen lassen. Der Benutzer kann hier Beziehungen zwischen den Objekten durch Auswahlmenüs aufbauen (Titel-> Interpret, Interpret-> Gruppe). Der Import von Objekten und den Musikdateien wird über Verzeichnisstrukturfenster im Explorerstil abgewickelt. Die Fenster liegen lose auf dem Desktop und können durch Drag und Drop verschoben werden, wobei sie bei Annäherung der Kanten an einander „andocken“ und fortan im Verbund verschoben werden können. Durch Ziehen der Fensterränder an allen vier Seiten und vier Ecken mit dem Cursor können sie beliebig skaliert werden.

Beschreibung der Komponenten:

Schnittstellen und Beschreibung der Playlist:

Die Komponente der Playlist ist das, was man sich unter ihrem Namen auch vorstellt. Eine Playlist ist ein Fenster, in dem eine Reihe von Dateien angezeigt werden und verschieden Wiedergabemodi, z.B. zufällige Wiedergabe oder Wiederholung eingestellt werden können. Außerdem stellt die Playlist eine Menge von Funktionen zur Verfügung um die Liste der ausgewählten Dateien zu sortieren, filtern, usw.
Das Klassenmodell der Playlist ist als graphische Aufbereitung in der angehängten Datei „Playlist-Klassenmodell.gif“ enthalten.
Die Schnittstellen der Playlist sind in bekannter Java-Notation gehalten und mit den leicht verständlichen Javadoc Kommentaren versehen. Die Überschrift der einzelnen Interfaces zeigt schnell an, wer mit wem das Interface besitzt.

Als Legende sei gesagt:
Benennung: interface XxxToYyy
heißt, dass das Modul Xxx Methoden für das Modul Yyy bereitstellt.
d.h. XxxToYyy wird von Modul Xxx implementiert.

Player - Playlist

```
/**
 * Das Interface PlayIterator wird von der Playlist-Komponente bereitgestellt.
 * Es dient dazu, mehreren Playern den Zugriff auf eine Playlist zu
 * ermöglichen. Die von den ListIterator-Funktionen next() und previous()
 * zurückgegebenen Objekte sind alle vom Typ PlaylistEntry.
 *
 * Einen neuen Iterator über einer Playlist bekommt der Player über die
 * Funktion getPlayIterator() des PlaylistToPlayer-Interfaces.
 *
 * @see PlaylistEntry
 * @see PlaylistToPlayer#getPlayIterator
 */
```

```
interface PlayIterator implements java.util.ListIterator {
   void setRepeat(boolean repeat);
   void setRandom(boolean random);
   boolean isRepeat();
   boolean isRandom();
}

interface PlaylistToPlayer {
   /**
    * Gibt ein PlayIterator-Objekt zurück, das eine logische
    * Abspielreihenfolge der Stücke in der Playlist definiert.
    *
    * @param   startIndex      Der Index des Stückes in der sichtbaren
    *          Playlist, von dem die Wiedergabe starten soll.
    * @return  das PlayIterator-Objekt
    */
   PlayIterator getPlayIterator(int startIndex);
}

interface PlayerToPlaylist {
   /**
    * Wird aufgerufen, wenn die dem Iterator zugrunde liegende Playlist
    * nicht mehr existiert, bzw. der Iterator keine definierte Reihenfolge
    * mehr angeben kann (z.B. Ende der Playlist erreicht).
    *
    * Der Player sollte daraufhin in den Stop-Zustand fallen, sobald das
    * aktuelle Stück beendet worden ist.
    */
   void iteratorInvalidated();
}
```

GUI - Playlist

```
interface GUIToPlaylist {
   /**
    * Wird aufgerufen, wenn sich an der Playlist etwas geändert hat,
    * z.B. wenn die Einträge sortiert oder wenn Stücke entfernt wurden.
    */
   void updateDisplay();
}

interface PlaylistToGUI {
   /**
    * Gibt die Liste der Multimedia-Eigenschaften-Namen zurück, die von dieser
    * Playlist angezeigt werden können. Die Einträge im Array entsprechen
    * den Datenfeldern des Datenbankschemas in der Reihenfolge, in der
    * sie bei der Datenbankabfrage angefordert wurden.
    *
    * Bei einer Abfrage der Form
    * <code>SELECT artist, title, length FROM some_table</code> würde
    * diese Playlist die Eigenschaften
    * <code>{"artist", "title", "length"}</code> kennen und in dieser
    * Funktion zurückgeben.
```

```
 *
 * Eine Anwendungsmöglichkeit dieser Einträge bietet sich z.B. als
 * Tabellenkopf-Einträge einer in Tabellenform dargestellten Playlist.
 *
 * @return  Array von Eigenschaftennamen
 */
String[] getProperties();

/**
 * Gibt die der Playlist zu Grunde liegende List mit PlaylistEntrys
 * zurück. Auf diese Liste bzw. deren Iteratoren darf <b>nur lesend</b>
 * zugegriffen werden, ansonsten ist das Programmverhalten undefiniert.
 *
 * @return java.util.List mit PlaylistEntry-Einträgen.
 */
List getVisibleList();

/**
 * Veranlasst die Playlist, das Musikstück an der angegebenen Position
 * URL/Dateiname ans Ende der Playlist hinzuzufügen.
 * Die GUIs werden von dieser Änderung automatisch über
 * updateDisplay() benachrichtigt.
 *
 * @param   path    Pfad bzw. URL zur Multimedia-Datei.
 * @see     GUIToPlaylist#updateDisplay
 */
void addFile(String path);

/**
 * Veranlasst die Playlist, das Stück an Position index an die
 * Position destination zu verschieben. Die GUIs werden von dieser
 * Änderung automatisch über updateDisplay() benachrichtigt.
 *
 * @param   index           Ausgangsindex (Zählung ab 0)
 * @param   destination     Zielindex (Zählung ab 0)
 * @see     GUIToPlaylist#updateDisplay
 */
void moveEntry(int index, int destination);

/**
 * Veranlasst die Playlist, das Stück an Position index aus der
 * Liste zu löschen. Die GUIs werden von dieser
 * Änderung automatisch über updateDisplay() benachrichtigt.
 *
 * @param   index   Index des zu löschenden Eintrags (Zählung ab 0)
 * @see     GUIToPlaylist#updateDisplay
 */
void deleteEntry(int index);

/**
 * Sortiert die Einträge der Liste nach Dateinamen.
 *
 * @param   selection       Eine Untermenge der zur Anzeige verwendeten
 *          Liste von PlaylistEntrys. Falls NULL, wird die gesamte
```

```
     *          Liste sortiert.
     * @param   ascending       True, wenn aufsteigen sortiert werden soll.
     */
    void sortByFilename(java.util.Collection selection, boolean ascending);

    /**
     * Sortiert die Einträge der Liste nach Pfaden und Dateinamen.
     *
     * @param   selection       Eine Untermenge der zur Anzeige verwendeten
     *          Liste von PlaylistEntrys. Falls NULL, wird die gesamte
     *          Liste sortiert.
     * @param   ascending       True, wenn aufsteigen sortiert werden soll.
     */
    void sortByPath(java.util.Collection selection, boolean ascending);

    /**
     * Sortiert die Einträge zufällig.
     *
     * @param   selection       Eine Untermenge der zur Anzeige verwendeten
     *          Liste von PlaylistEntrys. Falls NULL, wird die gesamte
     *          Liste sortiert.
     */
    void sortShuffle(java.util.Collection selection);

    /**
     * Sortiert die Einträge der Liste nach dem übergebenen Kriterium.
     * Hierbei muss es sich um eine der Multimedia-Eigenschaften handeln,
     * die von der Playlist angezeigt werden können.
     *
     * @param   property        Das Kriterium, nach dem sortiert werden soll.
     * @param   selection       Eine Untermenge der zur Anzeige verwendeten
     *          Liste von PlaylistEntrys. Falls NULL, wird die gesamte
     *          Liste sortiert.
     * @param   ascending       True, wenn aufsteigen sortiert werden soll.
     * @see     #getProperties
     */
    void sortBy(String property, java.util.Collection selection,
            boolean ascending);
}
```

Datenverwaltung - Playlist

```
/**
 * Das Interface PlaylistEntry wird vom Playlistmodul definiert und stellt
 * die einzelnen Musikstück-Einträge der Playlist zusammen mit einer
 * für die Playlist relevanten Auswahl von Multimedia-Eigenschaften aus
 * der Datenbank dar. Die Eigenschaften werden in einer java.util.Map
 * festgehalten, bei der die Feldnamen der Datenbank die Schlüssel
 * zur Map darstellen
 * (siehe {@link PlaylistToGUI#getProperties getProperties}),
 *
 * Die Implementierung des Interfaces geschieht im Modul Datenverwaltung.
 */
```

```
interface PlaylistEntry {
   PieceID getID();
   String getPath();
   Map getPropertyMap();
}

interface DatabaseToPlaylist {
   /**
    * Ermittelt anhand der übergebenen Position eines Musikstücks die
    * zugehörige Datenbank-ID (PieceID) und gibt sie zurück.
    *
    * Falls die Datei existiert, aber in der Datenbank noch nicht
    * vorhanden ist, sollten ggf. per Benutzerinteraktion die
    * erweiterten Multimedia-Eigenschaften ermittelt werden.
    *
    * @param   path    Pfad bzw. URL zur Multimedia-Datei.
    * @return  NULL, falls die Datei nicht vorhanden ist und der angegebene
    *          Position in der Datenbank noch nicht verzeichnet ist.
    */
   PieceID getPieceID(String path);

   /**
    * Sucht in der Datenbank nach den angegebenen Eigenschaften des
    * Stückes mit der angegebenen ID und gibt ein neues
    * PlaylistEntry-Objekt zurück.
    *
    * @param   id      ID des abzufragenden Stückes
    * @param   properties  Liste von Multimedia-Eigenschaftennamen,
    *          die im PlaylistEntry-Objekt übergeben werden sollen
    * @return  Ein neues PlaylistEntry-Objekt mit den gewüschten
    *          Eigenschaften des Stückes.
    * @see PlaylistEntry
    */
   PlaylistEntry lookupEntry(PieceID id, String[] properties);

   /**
    * Sucht in der Datenbank nach den angegebenen Eigenschaften der
    * Stücke mit den angegebenen IDs und gibt eine java.util.List mit
    * neuen PlaylistEntry-Objekten zurück.
    *
    * @param   ids     java.util.List mit IDs der abzufragenden Stücke
    * @param   properties  Liste von Multimedia-Eigenschaftennamen,
    *          die in den PlaylistEntry-Objekten übergeben werden sollen
    * @return  Eine java.util.List mit neuen PlaylistEntry-Objekten, die
    *          die gewünschten Eigenschaften der Stücke enthalten. Die
    *          Reihenfolge der PlaylistEntrys muss der der IDs im
    *          ids-Parameter entsprechen.
    * @see PlaylistEntry
    */
   List lookupEntries(List ids, String[] properties);
}
```

```
interface PlaylistToDatabase {
    /**
     * Veranlasst die Playlist, das Stück mit der angegebenen ID am Ende
     * der Liste hinzuzufügen. Die Playlist fragt die benötigten
     * Multimedia-Eigenschaften des Stückes selbständig über
     * {@link DatabaseToPlaylist#lookupEntry lookupEntry()} bei der
     * Datenbank ab.
     *
     * @param   id Datenbank-ID des hinzuzufügenden Stückes
     */
    void addEntry(PieceID id);

    /**
     * Veranlasst die Playlist, die Stücke mit den angegebenen IDs am Ende
     * der Liste hinzuzufügen. Die Playlist fragt die benötigten
     * Multimedia-Eigenschaften der Stücke selbständig über
     * {@link DatabaseToPlaylist#lookupEntries lookupEntries()} bei der
     * Datenbank ab.
     *
     * @param   ids java.util.List mit Datenbank-IDs der
     *          hinzuzufügenden Stücke
     */
    void addEntries(List ids);

    /**
     * Gibt eine Liste mit Datenbank-IDs der Stücke in der Playlist
     * zurück. Die Reihenfolge der IDs entspricht der der Stücke in
     * der angezeigten Liste.
     *
     * @return  java.util.List mit PieceIDs der Stücke.
     */
    List getEntryList();

    /**
     * Hiermit wird der Playlist mitgeteilt, dass sich Eigenschaften des
     * angegebenen Stücks geändert haben.
     *
     * @param   id  ID des geänderten Stückes.
     */
    void updatedEntry(PieceID id);

    /**
     * Hiermit wird der Playlist mitgeteilt, dass sich Eigenschaften der
     * angegebenen Stücke geändert haben.
     *
     * @param   ids java.util.List mit IDs der geänderten Stücke.
     */
    void updatedEntries(List ids);
}

/**
 * Dieses Interface muss der Controller für die Datenverwaltung
 * bereitstellen, damit die Datenverwaltung bei einer neuen Abfrage
 * ein neues Playlist-Objekt erhalten kann. Da der Controller die
```

```
 * Thread-Pools verwaltet, mus er als Playlist-Factory dienen.
 */
interface ControllerToDatabase {
    ...

    /**
     * Erzeugt eine Playlist, die die angegebenen Eigenschaften anzeigen kann.
     */
    Playlist createPlaylist(String[] properties);

    ...
}
```

Schnittstellen und Beschreibung des Players/ der Datenverarbeitung:

Zuerst gibt es die die Schnittstelle zur Playlist, die allerdings schon bei der Playlist beschrieben steht. Danach werden die anderen Schnittstellen des Players zu den anderen Komponenten aufgezeigt. Um die Funktionsweise der einzelnen Methoden und Komponenten besser verständlich zu machen, wurde auch dieses Schnittstellensystem graphisch aufbereitet. Zuerst folgt die in üblicher Javanotation gehaltene Beschreibung der Schnittstellen, danach verdeutlicht das Schaubild noch einmal leicht verständlich die Zusammenhänge. Zudem muss gesagt werden, dass nur der Player bzw. die Datenverarbeitung auf Daten zugreifen können. Fordert eine Komponente eine Datei an oder möchte sie verändern, so geschieht dies immer uber die Komponente Player/Datenverarbeitung.
Das Schaubild ist in der Datei im Anhang „Klassenmodell-Player_Datenverarbeitung.doc“ enthalten. Erläuterungen zum Schaubild:
die Hauptklasse unseres Moduls ist die MMFC, die sämtliche Interfaces beinhaltet und die Objekte Equalizer, Cutting und MMF benutzt, um die gewünschte Funktionalität zur Verfügung zu stellen. Die hier vorliegende Komponente soll den Zugriff auf die physikalisch vorhandenen Multimedia-Dateien realisieren und die eigentliche Wiedergabe ermöglichen.

Player - GUI

```
interface GUIToPlayer {
    /**
     * Gibt Positionsänderungen innerhalb einer MMD an die GUI weiter,
     * damit diese z.B. den Slider bewegen kann. Dies geschieht beim Start
     * und beim Spulen von MMDs; die GUIs müssen während der restlichen Zeit
     * durch einen Timer o.ä. die Zeit simulieren.
     *
     * @param   newPosition   neue Position innerhalb der MMD in Millisekunden
     */
    void setPosition(long newPosition);

    /**
     * Wird benutzt, wenn eine neue MMD abgespielt werden soll. Die ID dieser
     * Datei (Format ist noch festzulegen, siehe MMDID) wird an die GUI
     * gesendet, die sich dann alle benötigten Informationen von der Daten-
     * verwaltung besorgen kann.
     *
     * @see     MMDID
     * @param   newID   ID der neuen MMD
```

```
     */
    void setMMD(MMDID newID);
}

interface PlayerToGUI {
    /**
     * Startet die Wiedergabe einer MMD.
     *
     * @return  konnte Wiedergabe gestartet werden?
     */
    boolean start();

    /**
     * Stoppt die Wiedergabe einer MMD.
     *
     * @return  konnte Wiedergabe gestoppt werden?
     */
    boolean stop();

    /**
     * Unterbricht die Wiedergabe einer MMD.
     *
     * @return  konnte Wiedergabe unterbrochen werden?
     */
    boolean pause();

    /**
     * Stoppt die Wiedergabe der MMD und startet die Wiedergabe der
     * (logisch) nächsten MMD (durch Kommunikation mit der Playlist).
     *
     * @return  konnte nächste MMD gestartet werden?
     */
    boolean next();

    /**
     * Stoppt die Wiedergabe der MMD und startet die Wiedergabe der
     * (logisch) vorherigen MMD (durch Kommunikation mit der Playlist).
     *
     * @return  konnte vorherige MMD gestartet werden?
     */
    boolean previous();

    /**
     * Spult die MMD vorwärts und gibt sie von der neuen Position an wieder.
     *
     * @return  konnte MMD vorwärts gespult werden?
     */
    boolean forward();

    /**
     * Spult die MMD rückwärts und gibt sie von der neuen Position an wieder.
     *
     * @return  konnte MMD rückwärts gespult werden?
```

```
     */
    boolean rewind();

    /**
     * Setzt die neue Lautstärke der Wiedergabe (Volume noch nicht definiert).
     *
     * @see     Volume
     * @param   newVolume  die neue Lautstärke der Wiedergabe
     * @return  neue Lautstärke
     */
    Volume setVolume(Volume newVolume);
}
```

Player - Datenverwaltung

```
/**
 * interface DatabaseToPlayer
 * nicht benötigt
 */

interface PlayerToDatabase {
    /**
     * Liest eine Textdatei aus und gibt sie zurück.
     *
     * @param   path    der Pfad der auszulesenden Datei
     * @return  der Inhalt der ausgelesenen Datei
     */
    string readTxtFile(string path);

    /**
     * Schreibt eine Textdatei.
     *
     * @param   path    der Pfad der zu schreibenden Datei
     * @param   content der Inhalt der zu schreibenden Datei
     * @return  konnte die Datei geschrieben werden?
     */
    boolean writeTxtFile(string path, string content);

    /**
     * Liest Zusatzinformationen aus einer MMD aus und gibt sie zurück
     * (TagInfo noch nicht definiert).
     *
     * @see     TagInfo
     * @param   path    der Pfad der auszulesenden MMD
     * @return  die enthaltenen Zusatzinformationen der ausgelesenen MMD
     */
    TagInfo readTagInfo(string path);

    /**
     * Schreibt (alle möglichen) Zusatzinformationen in eine MMD.
     * (TagInfo noch nicht definiert).
     *
     * @see     TagInfo
```

```
    * @param   path    der Pfad der zu verändernden MMD
    * @param   infos   die Zusatzinformationen für die MMD
    * @return  konnte die MMD geändert werden?
    */
   boolean writeTagInfo(string path, TagInfo infos);

   /**
    * Erstellt eine eindeutige ID zu einer MMD
    * (MMDID noch nicht definiert).
    *
    * @see     MMDID
    * @param   path    der Pfad der MMD
    * @return  die eindeutige ID der MMD
    */
   MMDID createUniqueMMDID(string path);

   /**
    * Überprüft, ob eine MMD physikalisch vorhanden ist.
    *
    * @param   path    der Pfad der zu überprüfenden MMD
    * @return  ist die MMD vorhanden?
    */
   boolean isFile(string path);

   /**
    * Scannt ein Verzeichnis nach MMDs, die dann in die Datenverwaltung
    * aufgenommen werden können (TagInfo und TagInfoList noch nicht
    * definiert).
    *
    * @see     TagInfo
    * @see     TagInfoList
    * @param   directory   das zu scannende Verzeichnis
    * @param   recursively soll das Verzeichnis rekursiv durchsucht werden?
    * @return  die Liste der Zusatzinformationen aller im Verzeichnis
    *          gefundenen MMDs
    */
   TagInfoList scanDirectory(string directory, boolean recursively);
}
```

Erläuterungen:
die Hauptklasse unseres Moduls ist die MMFC, die sämtliche Interfaces beinhaltet und die Objekte Equalizer, Cutting und MMF benutzt, um die gewünschte Funktionalität zur Verfügung zu stellen.

Die hier vorliegende Komponente soll den Zugriff auf die physikalisch vorhandenen Multimedia-Dateien realisieren und die eigentliche Wiedergabe ermöglichen.

Schnittstellen und Beschreibung der Datenverwaltung:

Die Datenverwaltung verwaltet Informationen zu Multimedia-Dateien. Dazu spielt sie eine Mittlerrolle zwischen Datenverarbeitung, welche auf Dateiebene arbeitet, und den benutzernahen Komponenten Playlist und GUI. Die Datenverwaltung benutzt für die Verwaltung der Informationen die HSQL-Datenbank über JDBC. Die Datenbank/JDBC unterliegen für Nutzer der

Datenverwaltungs-Komponente dem Geheimsnisprinzip. Anfragen an die Datenverwaltung geschehen mit Botschaften, von denen viele nach Bearbeitung bestätigt werden. Anfragen über Daten aus Datenbank, z.B. durch Suche oder explizite Angabe mit eindeutiger ID, werden mit Botschaften beantwortet, welche Datenobjekte der Datenverwaltung enthalten (als Verweis). Wichtigstes Objekt dieser Art ist die "Piece", welche Kopien von Strings aus Datenbank-Einträge enthält. Diese Kopien sind für die Darstellung in der GUI und die Sortierung in der Playlist bestimmt. Die Verantwortung für die Korrektheit der Daten hat der Empfänger von Piece. Allerdings werden die GUI/Playlist über jede vorgenommene Änderungen an der Datenbank informiert (mit welchen Inhalten, ist noch in Arbeit). Die Datenverwaltung stellt Schnittstellen zur Verfügung welche eine Erweiterung zulassen, indem Felder der Datenbank nur durch Kategoriennamen zugänglich sind. Die Namen müssen nicht den Feldnamen der Datenbank entsprechen, können dies aber aus Gründen der Leserfreundlichkeit tun. Hinzufügen von weiteren Feldern in der Datenbank führt zu zusätzlichen Kategoriennamen. Welche Kategoriennamen vorhanden sind, wird nach einer Anfrage an die Datenverwaltung mitgeteilt.
(Weiteres siehe auch die Versionsplanung mit der Version V3 zu der Datenverwaltung und siehe auch die kommentierten Schnittstellen.)

Datenverwaltung - GUI

```
/**
 * Liefert ein Array aus Bezeichnern für jedes Datenfeld, welches in der
 * Datenbank vorhanden ist und zur Suche oder Anfrage verwendet werden darf
 * FieldType ist eine Klasse der Datenverwaltung
 */
FieldType[] getFields();

/**
 * Liefert ein Array mit den bei Suchanfragen unterstützten Vergleichs-
 * Operatoren. Ein Operator wird dabei in der üblichen Schreibweise
 * codiert (zum Beispiel "<" für kleiner)
 */
String[] getCmpOps();

/**
 * Liefert ein Array mit den bei Suchanfragen unterstützten booleschen
 * Operatoren. Beispiel für einen boolschen Operator ist "AND"
 */
String[] getBoolOps();

/*
 * Liefert die Anzahl der in der Datenbank verzeichneten Mediendateien
 * Kann 0 sein.
 */
int getNoOfPieces();

/**
 * Liefert für den Eintrag einer Multimedia-Datei in der Datenbank, welche
 * die PieceID besitzt, die angeforderten Informationen. Piece enthält
 * dann für die angegeben Kategorien die Kopien der Einträge aus der
 * Datenbank für diesen Multimedia-Datei-Eintrag.
 * Kategorien sind Bezeichner für diejenigen Felder/Spalten in der
 * Datenbank, welche von aussen abrufbar sind.
 * Wenn die PieceID nicht existiert, dann ist es Aufgabe der Aufrufer
```

```
 * zu entscheiden ob ein neuer Eintrag in der Datenbank angelegt werden
 * soll (andere Methoden).
 *
 * @return Piece    NULL, wenn Piece mit der PieceID id nicht vorhanden
 *                  sonst Piece mit der entsprechenden PieceID
 */
Piece getPieceByID(PieceID id, String[] categories);

/**
 * Die Reihenfolge der ids in dem Parameter List gibt die Reihenfolge der
 * Piece s in der List der Rückgabe vor, d.h. die Ordnung ist die gleiche.
 * @see Piece getPieceByID(PieceID id, String[] categories)
 * @param List          eine Liste von PieceID s
 * @return List         eine Liste von Piece s
 */
List getPiecesByID(List ids, String[] props);

/**
 * Dies ist die Methode für Suchanfragen.
 * Liefert ein Array von Medienstücken mit den in Anfrage Request[]
 * enthaltenen Suchkriterien. Jedes Piece in dem Array enthält Kopien
 * der Einträge der Datenbank zu den gewünschten Kategorien.
 * Kategorien sind Bezeichner für diejenigen Felder/Spalten in der
 * Datenbank, welche von aussen abrufbar sind.
 * Request ist eine Klasse der Datenverwaltung
 */
Piece[] getPieces(Request[] anfrage, String[] categories);

/**
 * Ändert den Eintrag in der Datenbank, der die gleiche PieceID wie
 * newEntry hat: übernimmt in alle in newEntry enthaltenen Felder.
 * Typischerweise ist newEntry ein Piece welches vorher durch eine
 * Suchanfrage oder ein Anfrage auf eine ID erhalten wurde.
 * Achtung: Die Methode versucht auch die geänderten Informationen
 * an die Datenverarbeitung über die URL der Datei weiterzureichen
 * um eventuell in der Datei vorhandenen entsprechenden Informationen
 * (zum Beispiel Tags) zu ändern.
 * Achtung: Als Botschaft hat der Aufruf der Methode eine wichtige Aufgabe,
 * nämlich dem Aufrufer zu melden, dass Datenbank Einträge verändert
 * wurden. Ob diese Rückmeldung eine eigene Botschaft sein wird, oder
 * in die Botschaft eingetragen wird, hängt von den Botschaften-Typen ab.
 * Weiterhin wird zurückgemeldet ob eine Multimedia-Datei mit dem
 * zu diesem Zeitpunkt angegeben Pfad (wenn newEntry einen Pfad enthält,
 * wird dieser verwendet) verändert werden konnte (d.h. verändert wurde).
 * Die Datenverwaltung wird dann auch die Checksum/Hash (see PieceID)
 * zu diesem Zeitpunkt aktualisieren.
 */
modifyEntry(Piece newEntry);

/**
 * einen Eintrag zu einer Multimedia-Datei aus der Datenbank unwiederruflich
 * entfernen. Die ID wird dadurch ungültig.
 * Der Aufrufer wird per Botschaft über die Änderung informiert
 */
```

```
removeEntry(PieceID id);
/**
 * einen Eintrag zu einer Multimedia-Datei in die Datenbank einfügen.
 * es kann zu Mehrfach-Eintragungen von Dateien vorkommen. Deshalb
 * ist noch die Möglichkeit in Arbeit, eine Liste von PieceID s zu liefern
 * die die gleiche physikalische Checksum haben. Eine weitere Möglichkeit
 * für Aufrufer ist, nach dem Aufruf alle Einträge mit gleichem Pfad,
 * oder gleichen Titel+Interpreten zu suchen, und es dem Benutzer zu
 * überlassen, welche davon nun überschrieben werden soll (oder keine).
 * Achtung: hier fehlt noch wie ein Pfad auszusehen hat
 * Achtung: hier fehlt noch eine Rückmeldung per Botschaft, dass ein neuer
 * Einträge zu der Datenbank hinzugekommen ist.
 */
addEntry(String path);

 /**
  * ein Verzeichnis wird rekursiv nach Multimedia-Dateien durchsucht
  * und alle passenden Dateien werden in die Datenbank aufgenommen.
  * hierzu wird die Komponente Datenverarbeitung/Player in Anspruch
  * genommen.
  * @see addEntry(String path);
  * @param String  der Parameter heisst anders, weil hier ein Pfad
  *                erwaretet wird und nicht ein Pfad mit Dateiname
  */
addEntriesByDirectory(String pathlabel);

 /**
  * Liefert eine Liste von IDs, die eindeutig in der Datenbank Einträge
  * identifizieren, welche eben nicht zu einer Multimedia-Datei gehören.
  * Die Idee dahinter ist, dass eine Multimedia-Datei aus der Datenbank
  * entfernt worden sein kann, aber dann noch Einträge in anderen Tabellen
  * zurückgeblieben sind, die nun keiner Multimedia-Datei mehr zuzuordnen
  * sind. Dem Benutzer soll die Möglichkeit gegeben werden diese Einträge
  * zu sehen, zu verändern, und zu löschen
  * Achtung: Dieses Konzept ist NEU. Es ist weder ausgearbeitet
  * noch ist sicher, dass es überhaupt eingeführt wird, da erst
  * noch die Funktionsweise der Datenbank hierzu geklärt wird
  * @return List      von UnlinkedIDs
  */
List findUnusedEntries();
```

Datenverwaltung - Playlist

```
/**
 * Liefert die PieceID eines Eintrags in der Datenbank für eine
 * Multimedia-Datei. Die Multimedia-Datei wird anhand einer Pfad-Angabe
 * spezifiert.
 * Falls der Pfad in der Datenbank nicht vorkommt, wird NULL zurückgegeben.
 * Es ist Aufgabe des Aufrufers die Aufnahme einer noch nicht vorhandenen
 * Multimedia-Datei zu iniitieren (andere Methoden), wenn gewünscht.
 *
 * @param   path    Pfad bzw. URL zur Multimedia-Datei.
 * @return  NULL, falls die URL in der Datenbank nicht vorkommt, ansonsten
```

```
     *eine Liste aus PieceID s, die diesen Pfad haben.
     */
    List getPieceID(String path);

    /*
     * wurden aus Schnittstelle herausgenommen, weil äquivalent zu getPieceByID:
     *
     * PlaylistEntry lookupEntry(PieceID id, String[] properties)
     * List lookupEntries(List ids, String[] properties)
     */


/*
 * Klassen
 */

    /**
     * Datenbankeintrag zu einer Multimedia-Datei
     * getCategoriesMap() gibt eine Map zurück, wo die Kategorien Keys sind.
     * Gibt es einen Key nicht so ist dies ein Fehler, ist der Inhalt eines
     * Eintrags aus der Datenbank leer, so steht unter dem Key nichts.
     * Ein Eintrag zu einer Multimedia-Datei in der Datenbank hat immer einen
     * Pfad und eine ID. Die IDs sind eindeutig. Gültige IDs können 1 und höher
     * sein (0 ist keine gültige ID). Es wird nicht gewährleistet das der Pfad gültig
     * oder eindeutig ist.
     */
    interface PlaylistEntry {
        PieceID getID();
        String getPath();
        Map getCategoriesMap();
    }

    public Piece implements PlaylistEntry{
        PieceID getID();
        String getPath();
        Map getCategoriesMap();
                String getCategory(String category);
    }

    /**
     * Datentyp um mittzuteilen, welche Datenbank-Felder für Suche und Anfragen
     * zur Verfügung stehen. Enthält einen Bezeichner für die Kategorie, d.h.
     * das Feld in der Datenbank (Kategorie statt Feld, um Mehrfachbedeutung
     * von Feld zu vermeiden). Der Bezeichner ist von der Datenverwaltung
     * gewählt und entspricht nicht 1:1 den Spaltennamen der Tabellen.
     * Die Kategorie (ihr Name) kann zum Beispiel von Aufrufern für Suchanfragen
     * an die Datenverwaltung verwendet werden.
     * @param String   Bezeichner einer Kategorie, kann für andere
     *                 Anfragen an die Datenverwaltung als Parameter verwendet
     * @param booleangibt an, ob es die Kategorie in der Datenbank nur
     *                 Zahlen als Einträge hat. Wenn ja, dann kann der
     *                 Aufrufer zum Beispiel eine andere Darstellung als
     *                 für Text wählen.
     */
```

```
public FieldType{
      String category;
      Boolean isNumeric;
   }

   /**
    * Idenfikationsnummer für Einträge von Multimedia-Dateien in der
    * Datenbank. Einmal festgelegt, ändert sich diese Nummer nicht mehr,
    * aber sie kann durch Löschen eines Eintrags wieder verschwinden.
    * Wird von der Datenverwaltung vergeben, und wird von Aufrufern verwendet
    * um Einträge aus der Datenbank zu identifizieren.
    * Achtung: in Arbeit ist eine zusätzliche Nummer, wie Checksum/Hash für
    * eine physikalische Multimedia-Datei um gleiche Dateien mit anderen
    * Pfaden zu erkennen bzw. wieder-zuerkennen.
    */
   public PieceID {
      long id;
   }

   /**
    * Enthält eine Suchanfrage für ein durchsuchbares Feld der Datenbank
    * (category), den zu benutzenden Operator (cmpOP) und einen String
    * mit dem Suchtext, welcher bei der Suche als Teilstring gesucht wird.
    * Wird ein boolOp angegeben (String ist nicht NULL), müssen weitere
    * Requests folgen, ansonsten ist die Suchanfrage fehlerhaft.
    * Verknüpfte Suchanfragen sind Arrays aus Request s, wobei die Reihenfolge
    * im Array signifikant ist, und die Priorität in absteigender Ordnung
    * festlegt.
    * @see String[] getFields()
    */
   public Request {
      String category;
      String cmpOp;
      String content;
      String boolOp;
   }
```

Schnittstellen und Beschreibung des Contollers:

In dem hier nun folgenden Interface ist nochmals genau beschrieben, wie der Controller mit den Komponenten zusammenspielen soll und wie die einzelnen Klassen entwickelt werden.

```
   /**
    * Generelles Publish/Subscribe Konzept.
    * Die abstrakte Klasse CHandler wird von jeder Komponente implementiert
    * Der Handler dient dazu die Nachrichten, welche an die Komponente gerichtet sind
    * zu verarbeiten und entsprechend darauf zu reagieren (mittels Methodenaufruf etc.).
    *
    * Die Methode MessageReceived(CMessage msg) muss dazu von den einzelnen
    * Komponenten entsprechend implementiert werden.
    *
    * Über die geerbte Methode SendMessage(CMessage msg) des CHandler wird eine
    * Nachricht an den Controller gesendet.
```

```
*
* Die zentrale Klasse des Konzepts ist CMessage.
* Jede Komponente sendet Nachrichten ausschliesslich an den Controller und dieser sendet
* sie dann weiter an die entsprechend zuständigen Komponenten. Über diese Nachrichten
* tauschen somit alle Komponenten alle nötigen Daten aus, bzw. lösen Aktionen aus. Je nach
* Nachrichtentyp/-inhalt wird eine vordefinierte (eindeutige) Messageklasse benutzt die von
* CMessage abgeleitet wird (die Definitionen der einzelnen Nachrichtenklassen bzw. deren
* Inhalt etc. müssen jedoch von den einzelnen Gruppen erfolgen, bzw. in Rücksprache mit der
* Controller-Gruppe entwickelt werden). Anhand des Nachrichtentyps leitet der Controller dann
* die Nachricht an die zugehörigen Komponenten weiter (also Datanbankanfragen an die
* Datenbank, Stop/Start-Aktionen an den Player, usw).
* Eingehende Nachrichten in der MessageReceived()-Methode müssen dann entsprechend
* bearbeitet werden, und falls eine Antwort nötig ist, diese erstellen und senden.
*/
```

Es ist zudem vorherigen Interface ebenfalls eine proof-of-concept Implementierung angehängt (Controller-proof-of-concept.zip) worden. Das proof-of-concept ist allerdings auch als solches zu verstehen, der CHandler wird noch "einfacher" in der Benutzung, so dass nur noch MessageReceived() von den einzelnen Komponenten implementiert werden muss. Die interne Struktur des CLooper wird noch verändert, sowie der Controller im Großen und Ganzen auch. Allgemein kann man sich aber schon anschauen (in TestModule.java) wie das Prinzip aussehen sollte.
Wenn V1 fertig gestellt ist gibt es dazu auch Beispielkomponenten und eine Erläuterung wie der Controller und CHandler zu benutzen bzw. zu integrieren sind.

Schnittstellen und Beschreibung der GUI:

Die GUI stellt die zentrale Komponente für die Interaktion mit dem Benutzer zur Verfügung. Der Benutzer des Programms hat nur die Möglichkeit über die GUI Daten und Einträge zu verändern. Die GUI wurde weiter oben unter Benutzerschnittstellen schon beschrieben.
Zu den schon vorher aufgeführten Interfaces, muss nur noch gesagt werden, dass eine allgemeine Update-Funktion noch zur Verfügung gestellt werden muss, um die GUI immer auf dem neusten Stand zu halten.

Datenbasisentwurf:

Der Entwurf der Datenbasis ist in dem im Anhang befindlichen Dokument „Datenmodell.png" graphisch dargestellt.

Datenbank-Schema:

Das von uns entwickelte Datenbank Schema ist in der angefügten Datei „RDB-Entwurf.png" graphisch aufbereitet worden. In diesem Diagramm sieht man auch sehr gut die Zugriffsmechanismen und -operationen, die durch die Datenbank zur Verfügung gestellt werden bzw. von ihr unterstützt werden.

Systemtest-Entwurf:

Testziele sind das Auffinden von Fehlern in der Programmfunktion und im Programmcode. Dadurch soll die fehlerfreie Erstellung des Programmcodes gesichert werden sowie die Anforderungen entsprechend der Spezifikation gesichert werden.
Die Tests werden protokolliert. Bei Fehlern wird zusätzlich ein Fehlerprotokoll ausgefüllt, welches für den Verantwortlichen des Programmteils bestimmt ist.
Die Komponenten werden in 4 Schichten eingeteilt: 1. Controller, 2. GUI, 3. Datenverarbeitung/Player, 4. Datenverwaltung/Playlist. Fehlerprotokolle betreffen nur eine Schicht.
Jede Klasse, die in die gemeinsame Klassenstruktur auf dem CVS-Server eingebracht wird, wird vorher getestet.
Der inkrementelle Test läuft in Stufen ab, von denen spätere nur ausgeführt werden, wenn in allen vorherigen keine Fehler auftraten. Schritte des inkrementellen Tests sind 1. Ctrl, 2. GUI, 3. Datenverarbeitung/Player, 4. Datenverwaltung und Playlist.
Der Komponententest baut auf festgelegten Datensätzen auf, die für das klassenweise Testen aller Methoden verwendet werden. Für diesen Test wird JUnit verwendet.
Der Integrationstest testet die Zusammenarbeit der verschiedenen Schichten. Testfälle sind hier Anforderungen der Komponenten untereinander.
Der Systemtest testet die Use-Cases, und prüft hier insgesamt das Zusammenspiel der Komponenten mit dem umgebenden System.
Für die Details der Dokumentation von Tests und der Struktur von Fehlerprotokollen wird auf das Handout des Votrags "Testplanung für das Projekt" verwiesen.
Zu testende Use-Cases sind:

1.) Player:
- Multimedia-Datei abspielen,
- Vor- und Zurückspulen
- Pause/Wiedergabe
- Stop
- Skip

2.) Playlist:
- Player starten
- Neue Playlist erstellen
- Repeat- und Zufallsmodus deaktivieren
- Einzelne MM-Dateien über Dateiauswahl in Liste einfügen
- Einzelne MM-Dateien über Drag & Drop in Liste einfügen
- Playlist speichern
- gespeicherte Playlist öffnen
- erstes Stück abspielen
- Skip vor und zurück
- Aktives Stück aus Playlist löschen
- Playlist nach Titeln neu sortieren
- Playlist von hand sortieren
- letztes Stück abspielen
- Repeat-Modus aktivieren
- Zufallsmodus aktivieren

3.) Datenverwaltung:
- Neuen Medienindex aus einem Verzeichnis mit MM-Dateien erstellen lassen
- Abfrage nach bestimmten Interpreten starten
- Eintrag modifizieren
- Eintrag entfernen
- Anzeigen der Eigenschaft einer mp3-/ogg-/wav-Datei

- Bearbeiten/Löschen verschiedener Eigenschaften
- Speichern der veränderten Eigenschaften in der Datei

4.) Datenverarbeitung:
- MM-Datei öffnen
- Equalizer einstellen
- Karaokefilter testen
- Mixer testen
- Audioschnitt-Konfiguration
- Audioschnitt testen (=> Damit Test der Pausenerkennung)

Anlagen: (die folgenden Dateien sind im Ordner „Anhang“ zu finden)
Playlist-Klassenmodell.gif
Klassendiagramm-Player-Datenverarbeitung.pdf
Controller-proof-of-concept.zip
Datenmodell.png
RDP-Entwurf.png
GUI-Prototyp1.jpg
GUI-Prototyp2.jpg