Universität Regensburg

Philosophische Fakultät III
Sprach- , Literatur- und Kulturwissenschaften
Institut für Information und Medien, Sprache und Kultur (I:IMSK)
Lehrstuhl für Medieninformatik

Advanced Software Engineering
MEI-M 25.2 (M.Sc.)
WS13/14 - SS14
Leitung: Prof. Dr. Christian Wolff

Dokumentation:

Collaborative Tabletop Music Search

Michael Adlfinger, Lukas Lamm, David Lechler, Tobias Semmelmann
1499800, 1511167, 1510484, 1509311
1-2 Semester M.Sc. Medieninformatik

Abgegeben am 19.05.2014

Inhalt

Abbildungsverzeichnis

1 Einleitung

Das Projekt Collaborative Tabletop Music Search entstand im Rahmen des Masterkurses Advanced Software Engineering am Lehrstuhl Medieninformatik der Universität Regensburg. Das Projekt setzt auf dem Multitouchtisch Samsung SUR40 des Lehrstuhls auf.

Dieses Dokument dient der Dokumentation des Projekts. Zuerst werden einige zusätzliche Angaben gemacht über die verwendeten Bibliotheken und weitere Ressourcen. Danach werden Problemstellung und Ziel des Projekts zusammengefasst. Darauf folgen eine kurze Setup-Anleitung, eine Projektbeschreibung sowie Angaben zur Architektur und Implementierung des Projekts. Eine Beschreibung der Durchführung und der Ergebnisse eines informellen Usability Tests sowie ein Ausblick runden diese Dokumentation ab.

2 Zusätzliche Angaben

Nachfolgend werden alle Ressourcen aufgelistet, die für die Implementierung des Projektes verwendet worden sind:

- Echonest API Key
- Spotify API Key
- Spotify Premium Account
- Samsung SUR40
- Tangibles
- Libspotify
- OhLibSpotify
- NAudio
- Json.NET
- Wpf Application Framework
- PieInTheSky
- log4net

- Blake.NUI.WPF, Blake.NUI WPF Surface, Blake.NUI WPF SurfaceToolkit
- .Net 4.0 Framework
- Surface 2.0 SDK
- Git Repository, Branch master (https://github.com/UniRegensburg/ASE-CollaborativeTabletopMusicSearch)

Alle Bibliotheken und Ressourcen liegen dem Projekt bei, mit Außnahme des Spotify Premium Accounts, der benötigt wird um Songs von Spotify zu streamen. Weiterhin werden Tangibles mit den ID's 00 bis 05 benötigt.

2.1 Zusammenfassung

Das Projekt CTMS ermöglicht mehreren Benutzern gleichzeitig das Suchen von Musik in der Datenbank von Spotify, das Speichern der Suchergebnisse in Playlisten und das Streamen von Musik. Zu den Suchergebnissen können weitere detaillierte Informationen zu den Künstlern aus der *Echonest*-Datenbank abgerufen werden. Eine ausführlichere Beschreibung liefern die Punkte 3 und 4.

2.2 Setup-Anweisungen

Sobald der SUR40 TableTop gestartet wurde und sich der Nutzer auf dem Desktop befindet, kann das Projekt über "ctms.exe" gestartet werden. Die Datei befindet sich nach Erstellung der Projektmappe unter %\ASE\app\Ctms.Presentation\bin\Debug und kann per Maus oder Touch gestartet werden. Falls die Datei *spotify_appkey.key* nicht gefunden werden kann muss das Projekt *MusicStream* vor Ausführung der Anwendung neu erstellt werden.

3 Problemstellung

Das Projekt *CTMS* stellt eine Multiuser-Anwendung auf dem Samsung SUR40 Multitouchtisch dar. So können zwei oder mehr Benutzer gleichzeitig nach Musik suchen und diese abspielen lassen oder in Playlisten festhalten. Als Zielgruppe kommen u.a. diejenigen Personen in Frage, die entweder beruflich mit Musik zu tun haben und Inspirationsquellen, beispielsweise Komponisten, Arrangeure, Textdichter und DJs, oder Personen, die in ihrer Freizeit gerne Musik hören und zusammen mit anderen Gleichgesinnten neue Musikrichtungen entdecken und teilen möchten. Eine

weitere Zielgruppe sind in einem gemeinsamen Raum arbeitende Personen, welche sich auf einen Musikgeschmack für begleitende Hintergrundmusik einigen wollen.
Ziel des Projektes soll es sein, durch gemeinsame Nutzung des Tabletops und der Tangibles den kollaborativen Aspekt in den Vordergrund zu stellen und jedem Teilnehmer die Möglichkeit zu bieten, sich vom Musikgeschmack seines Gegenüber inspirieren zu lassen, neue Lieder zu entdecken und auf direktem Wege über Musik zu diskutieren.

4 Projektbeschreibung

Mit Collaborative Tabletop Music Search, kurz *CTMS*, können–mehrere Personen gleichzeitig nach Musik suchen, die ihrem Geschmack entspricht und die Ergebnisse in gemeinsamen Playlists festhalten. Sobald die Anwendung gestartet wurde, wird der Benutzer zunächst aufgefordert, sich mit einem vorhandenen Premium-Account bei *Spotify* einzuloggen, um die Anwendung nutzen zu können. Nach erfolgreicher Anmeldung kann im angezeigten Menü entweder eine schon vorhandene Playlist geöffnet werden oder der Benutzer kann eine neue Playlist anlegen. Durch die Anmeldung bei *Spotify* können zuvor dort angelegte Playlisten ebenfalls wiedergegeben werden und auch die neu erstellten werden automatisch gespeichert.

Durch Klick auf eines der Lupen-Icons gelangt man zum Hauptbildschirm der Anwendung. Hier kann durch Platzieren eines oder mehrerer Tangibles die Musiksuche gestartet werden. Stellt man ein Tangible auf dem Tisch, so erscheint ein Menü um das Objekt herum, in dem Künstler, Titel, Genre oder Parameter definiert werden können. Der durchschrittene Pfad wird mittels *Breadcrumb*-Navigation angezeigt und kann so Schritt für Schritt nachvollzogen und auch rückgängig gemacht werden. Weiterhin wird um jedes Tangible ein farbiger Kreis angezeigt, um sie von anderen unterscheiden zu können.

Hat der Benutzer beispielsweise nach deinem Künstler gesucht und diesen gefunden, ist das Tangible mit dem aktuellen Ergebnis belegt. Durch Klick auf den Suchbutton, der sich in der Mitte des Tisches befindet, wird die Bibliothek von Echonest durchsucht und die Ergebnisse werden am Tisch verteilt angezeigt. Jedes Ergebnis wird durch eine Kassette visualisiert, auf der Interpret und Titel dargestellt

werden. Zusätzlich wird eine dünne Linie auf der Kassette dargestellt, welche die Zuordnung zum betreffenden Tangible erleichtern soll.

Durch Doppeltippen auf ein Ergebnis wird dieses im Prelisten-Modus abgespielt, um das Lied schneller zu erkennen. Um ein Lied von Beginn an anzuhören, muss dieses aus der Playlist heraus geöffnet und abgespielt werden. Jede Kassette kann per Drag and Drop auf eines der beiden Playlisticons am Rand der Anwendung zur aktuell geöffneten Playlist hinzugefügt werden. Durch größerziehen des Ergebnisses wird daraus eine Detailansicht, in der Informationen, wie beispielsweise Bilder, andere Titel oder allgemeine Neuigkeiten über den Künstler enthalten sind.

Desweiteren lassen sich auch mehrere Tangibles miteinander kombinieren, indem sie ausreichend nahe zueinander platziert werden. Sind die Tangibles kombiniert, wird dies durch eine Animation veranschaulicht, bei der von jedem Tangible aus eine kleine Kugel zum Mittelpunkt der Tangiblegruppe wandert.

Werden zwei Tangibles mit unterschiedlichen Genres belegt und kombiniert, so erhält man bei anschließender Suche nur Ergebnisse, die auch beiden Genres zugeordnet werden können. Durch Definition eines Parameters mit einem dritten Tangible können zusätzliche Einschränkungen vorgenommen werden, wie zum Beispiel die minimale Geschwindigkeit eines Liedes (BPM).

Um die aktuell angezeigten Ergebnisse zu löschen, müssen zuerst alle Tangibles vom Tisch entfernt werden und anschließend der Button in der Mitte des Tisches gedrückt werden.

5 Architektur und Implementierung

5.1 Windows Presentation Foundation

Als grundlegende Technologie wird das Framework *Windows Presentation Foundation* (WPF), welches Bestandteil des *.NET*-Frameworks ist, unter Verwendung der Programmiersprache C# und der Markup-Sprache *XAML* verwendet. Da die Anwendung für den Multitouch-Tisch Samsung SUR40 mit Microsoft *PixelSense*-Technologie entwickelt wurde, dient zusätzlich das *Surface 2.0 SDK*, welches die Verwendung der Version 4.0 des *.NET*-Frameworks voraus setzt.

5.2 WPF Application Framework

Ein *Model-View-ViewModel* (MVVM) - Pattern, welches sich für WPF-Anwendungen eignet, wird mithilfe des *WPF Application Framework*[1] (WAF) angewendet. Als Vorlage diente hierfür das Beispielprojekt *BookLibrary*[2] Dadurch wird die Repräsentation des Status und Verhaltens als unabhängig von den GUI-Elementen gestaltet. An einigen die Datenstruktur betreffend komplexeren Stellen wurden dazu ebenfalls durch WAF unterstützte *DataModels* verwendet, welche das Model mit zusätzlichen GUI-Daten erweiteren. Loose Kopplung wird in der Architektur u.a. dadurch realisiert, dass die *Controller* zwischen den nicht untereinander kommunizierenden ViewModels vermitteln.

Durch die Beachtung einer geschichteten Architektur, welche die Anwendung in mehrere Ebenen, welche in diesem Fall einzelnen Projekten einer Projektmappe entsprechen, unterteilt, wird eine korrekte Strukturierung umgesetzt und werden Ringverweise verhindert. Niedrigere Ebenen sind allgemeiner gehalten und betreffen low-level-Funktionen als auch generelle Services, höhere Ebenen sind dagegen applikationsspezifischer.

Dadurch, dass höhere Ebenen die niedrigeren aufrufen dürfen aber nicht umgekehrt, werden Koppelungen und Abhängigkeiten reduziert und somit die Wiederverwendbarkeit, Testbarkeit und Übersichtlichkeit der Software verbessert.[3]
Vor Beginn der Implementierung wurden Sketches sowie Paper Prototypes erstellt, mit Probanden getestet und vielfach iteriert. Anschließend konnte eine grundlegende Softwarearchitektur unter Verwendung des WAF entworfen und codiert werden. Dank der detaillierten Anforderungsanalyse konnte die grobe Projektstruktur über die gesamte Entwicklungszeit hin größtenteils unverändert bleiben, mit Fokus auf die Aufteilung der Views und somit der *Controller*, *ViewModels*, *Models* usw.

In der folgenden Abbildung 1 ist eine Übersicht über alle zu der Anwendung zugehörigen Projekte zu sehen.

---

[1] https://waf.codeplex.com/

[2] https://waf.codeplex.com/releases/view/98850

[3]https://waf.codeplex.com/wikipage?title=Architecture%20-%20Get%20The%20Big%20Picture&referringTitle=Documentation

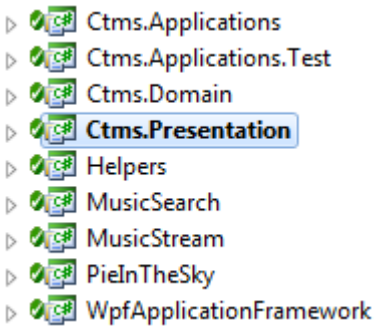

Abbildung 1: Übersicht aller Projekte der Anwendung

Im Folgenden sind die Zuständigkeiten der WAF-spezifischen Projekte aufgelistet:

- Ctms.Applications
  - Allgemein: Workflow,
  - Controller: Workflow der Anwendung, Kommunikation zwischen *ViewModels*
  - Repository: Hinzufügen, Entfernen, Selektieren und Filteren von Datensätzen (eigene Ergänzung)
  - DataFactories: Erstellen von Objektinstanzen (eigene Ergänzung)
  - DataModels: Erweiterung der ViewModels um GUI-spezifische Daten
  - Services: Zugriff auf Views, Initialisierung der Tags
  - ViewModels: Repräsentieren Status und Verhalten der *Presentation*
  - Views: Interfaces zur Kommunikation zwischen *ViewModels* und *Views*
  - Workers: zur Minimierung der *Controller*, führen wichtige Funktionen durch:
    - FftWorker: Equalizer-Berechnungen
    - InfoWorker: Darstellung von allgemeinen und Tag-spezifischen Infos
    - MusicStreamAccountWorker: Spotify Login etc.
    - PlaylistWorker: Playlistenverwaltung
    - ResultWorker: Auswertung der durch SearchWorker übergebenen Ergebnisse, Filterung der abspielbaren Songs, gleichmäßiger Ergebnisanteil aller Tangibles, Wiederholung der Suchanfragen bei zu wenig Ergebnissen

- SearchOptionWorker: Logik der Menüführung der Tangibles (Ebenenunterteilung), Suchanfrage nach Vorschlägen für Künstler und Songs, Inputvalidierung,
- Searchworker: Auswertung der zu den Tags zugewiesenen Suchbegriffe, Durchführung von kombinierten und unkombinierten Suchanfragen, Laden der Details zu Künstlern
- StreamingWorker: Vorhören von Songs, Wiedergabe der Playlists
- TagCombinationWorker: Überprüfung der Kombinierbarkeit der Tags, Berechnung der Distanzen und Mittelpunkte,

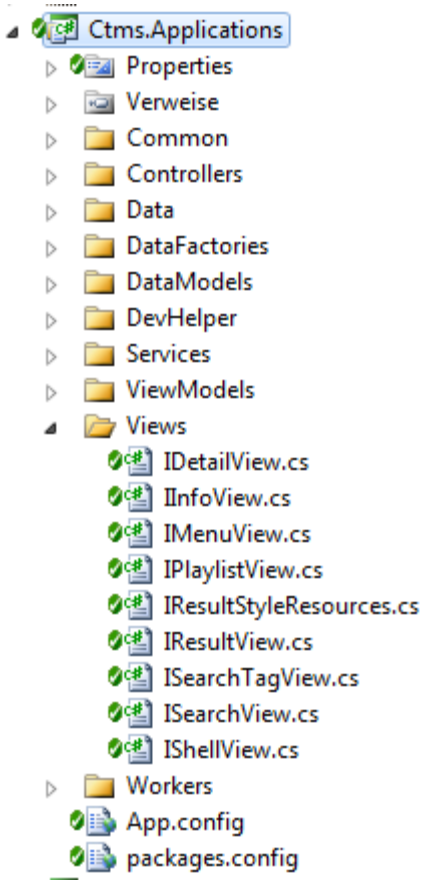

Abbildung 2: Detailansicht der *Applications*-Ebene

- Ctms.Domain
  - Allgemein: Objektstruktur
  - Models

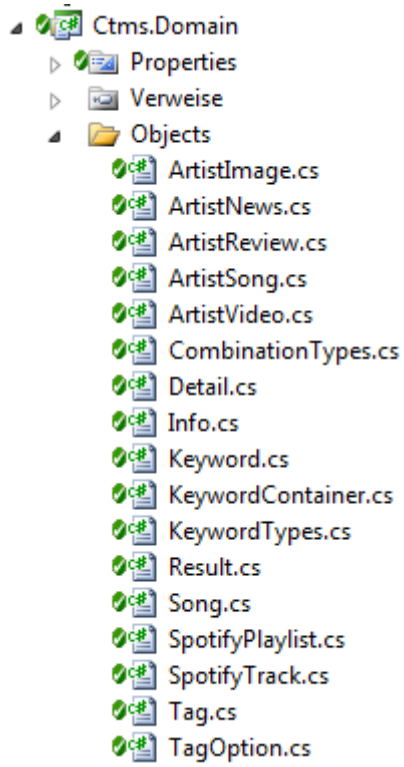

Abbildung 3: Detailansicht der *Domain*-Ebene

- Ctms.Presentation:
  - Allgemein: GUI, Look and Feel,
  - Views: beinhalten *Controls*, Datenbindung
  - Controls: GUI-Elemente wie Textboxes
  - Resources: Bilder, Schriftarten
  - Styles: Aussehen der *Controls*
  - ValueConverters: Konvertierung von Werten wie *boolean* zu *visibility*

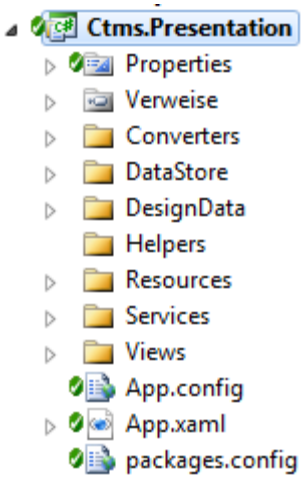

Abbildung 4: Detailansicht der *Presentation*-Ebene

In Abbildung 5 ist dargestellt, wie Kommunikation zwischen den Elementen der Ebenen *Applications*, *Domain* und *Presentation* verläuft

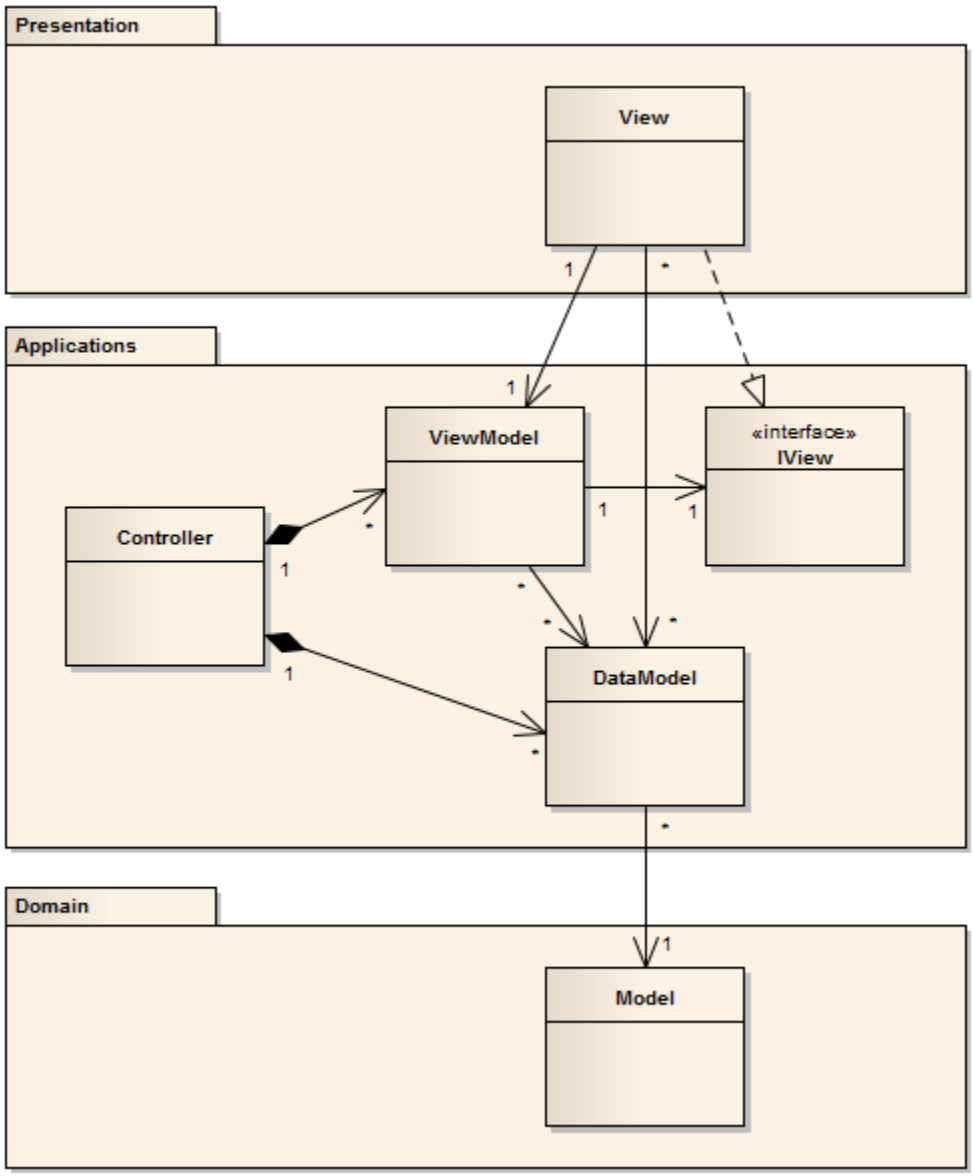

Abbildung 5: WAF - Kommunikation zwischen den Elementen[1]

Da die Worker einen Großteil der Applikationslogik steuern traten hier einige Hürden auf. So erforderte im *SearchOptionWorker* die Umsetzung der Tagmenüs, welche durch die vier verschiedenen Typen von Suchbegriffen Artist, Title, Parameter und Genre führen, eine Herausforderung dar. Dies ist zum einen durch den je Typ differenzierten Workflow als auch durch die je Tag unabhängig zu behandelnden Datensätze bedingt. Eine Befüllung des Kreismenüs im Code-behind des *Views*, eine Unterteilung der Logik in Ebenen, eine passende Objektstruktur der Tags, Tagoptionen und Stichwörter sowie

eine stetige Übergabe der Tag-Ids waren zur Umsetzung nötig. Durch weitere Filterungsoptionen im *ViewModel* konnte das Scrollen durch die Optionen in einer Endlosschleife realisiert werden.

Der *FftWorker* analysiert die durch das Streaming gelieferten Audiodaten unter Verwendung der Fourier-Analyse hinsichtlich des Anteils der einzelnen Frequenzspektren und steuert die Visualisierung dieser als Equalizer. Die korrekte Umsetzung der Analyse stellte hier ein großes Problem dar, konnte aber verwirklicht werden.

Grundlegende Funktionen zur Darstellung von Informationen für den Nutzer werden durch den *InfoWorker* realisiert. Dieser bietet Optionen zur Anzeige von einerseits anwendungsweiten und andererseits nutzerspezifischen Infofenstern. Für letztere Darstellungweise mussten die Positionen der Tags sowie deren derzeitige Orientierung berücksichtigt werden.

Der *MusicStreamAccountWorker* regelt das *Spotify* Login sowie Logout und regelt die Abspielbarkeit der Songs, während sich der *PlaylistWorker* und der *StreamingWorker* um das letztendliche Abspielen der Songs kümmern.

Für die korrekte Überprüfung, Erstellung und Darstellung der Tagkombinationen ist der *TagCombinationWorker* zuständig. Eine Schwierigkeit ergab sich hier durch die durch *EchoNest* limitierte Kombinierbarkeit der Suchbegriffe, denn diese ist eingeschränkt auf entweder fünf Genres mit x Parametern oder einen Künstler mit x Parametern. Durch die weitere notwendige Berücksichtigung von bereits bestehenden Kombinationen, dem Belegunsstatus und der Distanz zwischen den Tags stieg die Komplexität des Algorithmus. Da die Kombiniertbarkeit bei jeder Bewegung sowie Belegung eines Tangibles überprüft werden muss, wurde auf performante Programmierung Wert gelegt.

Das Starten einer Suchanfrage unter Verwendung des *MusicSearch*-Projekts verwaltet der *SearchWorker*. Dieser sammelt alle Tags, denen bereits Suchbegriffe zugeordnet wurden, überprüft, ob diese mit anderen Tags kombiniert wurden und veranlasst den SearchManager durch Übergabe der Begriffe zum Sammeln von Ergebnissen. Eine Herausforderung war hier die Auslagerung der Operationen in einen eigenen Thread und die damit verbundene komplexe Fehlerbehandlung sowie

Infoverwaltung. Weiterhin war zur Erstellung der Suchobjekte frei von Redundanz die Verwendung von Reflection nötig.

Sobald der *SearchWorker* Ergebnisse erhält leitet er diese an den *ResultWorker* weiter, welcher die Abspielbarkeit der Songs überprüft, bei zu geringer Anzahl an validen und unterschiedlichen Ergebnissen die Suchanfrage mehrmals wiederholt und für ein ausgeglichenes Maß an Resultaten pro Tag sorgt. Weiterhin werden durch den *ResultWorker* die Ergebnisse der Suche nach Künstlerdetails dargestellt.
Neben durch WAF vorgegebenen Projekten wurden einige Funktionen in separate Projekte ausgelagert, sodass diese auch durch Anwendungen, welche z.B. nicht für die Verwendung unter WPF entwickelt wurden, wiederverwendet werden können.

5.3 MusicSearch

Neben durch WAF vorgegebenen Projekten wurden einige Funktionen in separate Projekte ausgelagert, sodass diese auch durch Anwendungen, welche z.B. nicht für die Verwendung unter WPF entwickelt wurden, wiederverwendet werden können.
Eines dieser Projekte ist MusicSearch, welches für die Suche nach musikbezogenen Daten verantwortlich ist. Als Quelle wurde hierfür die API[4] von *Echonest* eingebunden. Besonders die Vielfalt der Informationen, der große Umfang an Informationen und die Kooperation zwischen *Echonest* und *Spotify* sind ausschlaggebend für die Auswahl von *Echonest*.
Eine Http-Anfrage an *Echonest*, bspw. nach einem bestimmten Titel, sieht folgendermaßen aus:

> "http://developer.echonest.com/api/v4/song/profile?api_key=L5WMC
> POK4F2LA9H5X&format=json&bucket=id:spotify-
> WW&limit=true&bucket=tracks&bucket=audio_summary&id=SOSFP
> GO13134390A22"

Die immer gleich bleibenden Standard-Url “http://developer.echonest.com/api/v4/” wird ergänzt durch den entsprechenden API-Teil, in diesem Fall “song/profile?”, welcher je nach Art der Anfrage variiert. Darauf folgt der API-Key, der unverzichtbar

[4] http://developer.echonest.com/docs/v4

für jegliche Anfragen an *Echonest* ist. Diesen stellt Echonest nach einer Registrierung kostenlos zur Verfügung.

Die restlichen Bestandteile des Http-Requests werden mit dem kaufmännischen 'und' aneinandergereiht und unterliegen keiner festen Reihenfolge.

Url-Bestandteil	Erläuterng
&format=json	Gewünschtes Format der Antwort (JSON / XML)
&bucket=id:spotify-WW	Anforderung von Spotify-Id's
&limit=true&	Limitierung der Ergebnisse. Nur Treffer mit Spotify-Id's
&bucket=tracks	Anforderungen aller Versionen des Ergebnisses (RadioEdit, Remix,...)
&bucket=audio_summary	Anforderung von Details über Ergebnis (Tempo, Länge, Lautstärke,...)
&id=SOSFPGO13134390A22	ID des gesuchten Titels

Tabelle 1: Bestandteile und Erläuterung einer Beispiel-Anfrage an Echonest

Bei fehlerfreier Syntax liefert *Echonest* eine *JSON*-formatierte Antwort. Die Antwort auf obige Bespiel-Anfrage sieht folgendermaßen aus:

```
{
  - response: {
      - status: {
            version: "4.2",
            code: 0,
            message: "Success"
        },
      - songs: [
          - {
                title: "Feuer frei!",
                artist_name: "Nachtmahr",
              - artist_foreign_ids: [
                  - {
                        catalog: "spotify-WW",
                        foreign_id: "spotify-WW:artist:1wGcs6i2IJbgXxE3bDIm9k"
                    }
                ],
              - tracks: [
                  - {
                        album_type: "single",
                        album_date: "2007-09-14",
                        foreign_release_id: "spotify-WW:release:1wN035nEzcyuf8PHVBIDPh",
                        catalog: "spotify-WW",
                        foreign_id: "spotify-WW:track:3wH2Xq4WB6C5bBH7d1HoyC",
                        album_name: "Kunst ist Krieg",
                        id: "TRTCMTW1338651D800"
                    },
                  - {
                        foreign_release_id: "spotify-WW:release:1fOvstPCrRsAjT2IMxvZ9K",
                        catalog: "spotify-WW",
                        foreign_id: "spotify-WW:track:6cg9JKpaQPJPKOLbo03A0Z",
                        id: "TREMCNU13A9BE88DA1"
                    }
                ],
                artist_id: "AR2NLBP1187B9A3A05",
                id: "SOSFPGO13134390A22",
              - audio_summary: {
                    key: 0,
                    analysis_url: http://echonest-analysis.s3.amazonaws.com/TR/v-PvewK_n7FI
                    Signature=pCi%2BfLHwrGbRhep6tdwM6tzHPAA%3D,
                    energy: 0.975855,
                    liveness: 0.407332,
                    tempo: 130.035,
                    speechiness: 0.043144,
                    acousticness: 0.000254,
                    mode: 1,
                    time_signature: 4,
                    duration: 278.02621,
                    loudness: -4.507,
                    audio_md5: null,
                    valence: 0.527006,
                    danceability: 0.601633
                }
            }
        ]
    }
}
```


Abbildung 6: Antwort von EchoNest

Zur besseren Lesbarkeit der Antwort, wurde diese in einem *JSON*-Viewer dargestellt. Die *JSON*-typische Verschachtelung der Bestandteile in geschweifte und eckige Klammern, ermöglicht es diese über die verwendete *JSON.Net*[5] Bibliothek in Objekte zu parsen.

[5] http://james.newtonking.com/json

Hierzu wurde die Klasse *ResponseContainer* erstellt, die das Schema der *JSON*-formatierten Antworten imitiert. Letztlich genügt nach hinreichender Modifizierung des Antwort-Strings ein Aufruf der *JSON*-Bibliothek um ein *ResponseContainer*-Objekt zu erstellen und mit Inhalt zu befüllen.
Die Antwort-Strings mussten meistens leicht manipuliert werden, da *Echonest* einerseits nicht darstellbare Sonderzeichen in Antwort-Strings mitschickt, als auch die eigene *JSON*-Formatierung teilweise unsauber verschickt.

5.4 MusicStream

Eine weitere Komponente der Anwendung bildet das Unterprojekt *MusicStream.* Innerhalb dieses Projekts erfolgt die Kommunikation und Abwicklung des Musik-Streaming von Spotify. Da die offizielle *libspotify API* in der Programmiersprache C geschrieben ist musste der C#-Wrapper *OhLibSpotify* von OpenHome[6] verwendet werden. Im weiteren Verlauf der Implementierungsphase stellte sich allerdings heraus, dass diese Bibliothek nicht vollständig fehlerfrei funktioniert. Dadurch können bei Operationen mit Spotify Fehler beim Lesen und Schreiben im Speicher auftreten. Diese *AccessViolationException*s werden vom Programm aufgefangen, produzieren aber häufig Folgefehler.

Grundlage der Komponente bildet der *MusicStreamSessionManager.* Dieser stellt Methoden für außenstehende Projekte bereit um die Session zu verwalten. Dazu gehört neben dem Login und Logout von Usern auch das Erstellen und Verwalten von Playlisten. Auch das Streaming von Spotify wird innerhalb des *MusicStreamSessionManagers* abgehandelt. Um die von der Spotify-Bibliothek benötigten Callback-Methoden zu implementieren wurde für die Spotify-Session eine *MusicStreamSessionListener*-Klasse sowie eine *MusicStreamPlaylistContainerListener*- und eine *MusicStreamPlaylistListener*-Klasse für Callbacks der Playlistcontainer und Playlisten erzeugt.

[6] https://github.com/openhome/ohLibSpotify

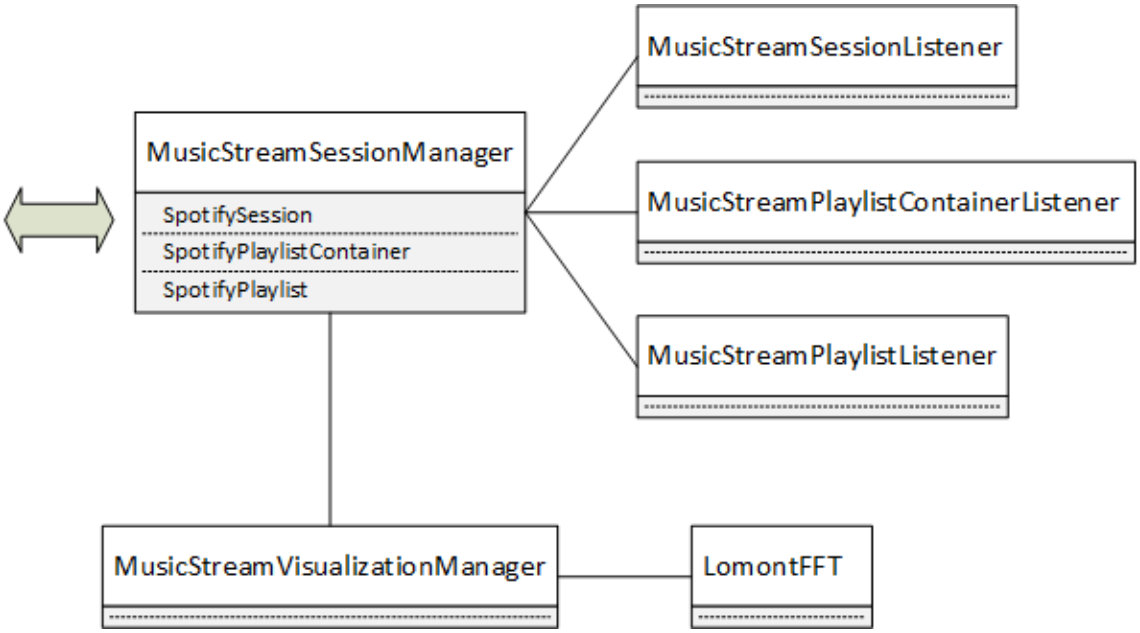

Abbildung 7: Überblick über das Projekt *MusicStream*

Die Klasse *MusicStreamVisualizationManager* erhält vom *MusicStreamSessionManager* den aktuellen byte-Array an Musikdaten. Zunächst werden die *PCM*-Daten[7] aufbereitet, bevor diese an die *LomontFFT*[8] Klasse weitergegeben werden. Die *LomontFFT* Klasse führt letztlich die *Fast Fourier Transformation* durch, wodurch das eingegangene Signal in seine Frequenzen unterteilt wird. Anhand der Abtastrate und Samplerate sind Rückschlüsse der gewonnene Daten auf die jeweiligen Frequenzen möglich.

Die Zuweisung der Frequenzbereiche zu den Rechtecken der Visualisierung dieser Daten erfolgt nach dem der durchschnittliche Ausschlag des Frequenzbereichs normalisiert, berechnet und gewichtet wurde. Die Berechnung und Gewichtung zielt hierbei weniger auf die korrekte Interpretation der Daten, als auf eine harmonische und glaubwürdige Visualisierung des jeweils abgespielten Liedes ab.

5.5 NAudio

Für das Abspielen der von *Spotify* erhaltenen PCM-Daten wurde die Bibliothek *NAudio*[9] verwendet. Diese unterstützt das Anlegen eines *BufferedWaveProvider*-Objekts, wobei es sich um eine *first-in-first-out* (FIFO) - Warteschlange handelt. Im Callback, der auf den Datentransfer von *Spotify* reagiert werden die Byte-Daten aus dem *Pointer* in

[7] Puls Code Modulation
[8] http://lomont.org/Software/Misc/FFT/LomontFFT.html
[9] http://naudio.codeplex.com/

einen lokal verwalteten Byte-Array kopiert. Anschließend werden die Daten der Warteschlange hinzugefügt. Ein *WaveOutDevice* kümmert sich um das Abspielen der Daten aus der Warteschlange. Das Streaming und Abspielen der Musik von Spotify stellte einen wichtigen Meilenstein, aber auch ein mögliches Problem im Bezug auf die Projektrealisierung dar. Da im Projektteam kaum Erfahrung im Bezug auf das Streaming und die erforderliche *Libspotify*-API vorhanden war und eine komplizierte Logik für die Playlist- und Prelisten-Funktion implementiert werden musste, wurde sehr früh mit der Arbeit an diesem Meilenstein begonnen.

5.6 Blake.NUI

Für die Erkennung einiger Gesten wurde die Bibliothek *Blake.NUI*[10] verwendet. Diese bietet die Möglichkeit Event-Handler für die Gesten Tap, DoubleTap und Hold anzulegen. Durch die Einbindung der Bibliothek lassen sich diese Gesten in den Code-behind Klassen der *Views* innerhalb des *Presentation* Layers abfangen. Von dort aus können nun Commands ausgelöst werden um die lose Koppelung des MVVM-Frameworks nicht zu verletzen.

5.7 Helpers

Das Projekt *Helpers* dient als Bibliothek für Applikations-unspezifische, wiederverwendbare Funktionen. So bietet es z.B. Unterstützung für Threading, String-Parsing und HTTP-Requests. Aber auch mathematische Funktionen finden im *Helpers*-Projekt Platz. Durch die Sammlung an Funktionen im *Helpers*-Projekt können häufig genutzte Funktionen von allen Projekten einbezogen werden und einfach wiederverwendet werden.

5.8 PieInTheSky

PieInTheSky ermöglicht die Darstellung der Kreismenüs für die Tangibles, wobei die ursprüngliche Bibliothek[11] an die Anforderungen von CTMS stark angepasst wurde, um u.a. eine Hervorhebung einzelner Elemente zu ermöglichen.

[10] http://blakenui.codeplex.com/

[11] http://www.codeproject.com/Articles/522343/A-Pie-Menu-for-WPF

5.9 Unit-Tests

Zuletzt ist noch das Projekt *Ctms.Applications.Test* zu erwähnen, welches Unit-Tests für Ctms.Applications beinhaltet. Diese sollte im Falle einer Wartung der Anwendung erweitert werden.

6 Informeller Usability Test

Zur Verifizierung der finalen Implementierung des Projekts wurde ein informeller Benutzertest mit vier Teilnehmern durchgeführt. Allen Teilnehmern des Usability Tests wurde zu Beginn eine kurze Einführung in das Projekt gegeben. Darüber hinaus wurden die Testpersonen (TP) über die Verwendung von Tangibles aufgeklärt. Es wurden keine konkreten Tasks formuliert, oder eine spezielle Reihenfolge vorgegeben. Zur Abdeckung des Funktionsumfangs des Projekts wurde bei Bedarf lediglich darauf hingewiesen, dass weitere Aktionen möglich sind, ohne dabei Hinweise auf die angedachte Vorgehensweise zur Erledigung dieser Aktionen zu geben. Hierdurch konnten die natürlichen Vorgehensweisen der Nutzer beobarchtet werden und es ergab sich ein harmonisches Gesamtbild zwischen Erkundung der Applikation durch den Nutzer und den Ausführungen, Fragen und Verbesserungsvorschlägen der Testpersonen.

Im ersten Test wurde eine Einzelperson getestet, im zweiten Test wurden drei Testpersonen gebeten kollaborativ die Applikation zu bedienen. Der zweite Test diente der Überprüfung der Kollaborationsfähigkeit des Projekts zu testen.

Beim Einzeltest fiel auf, dass die TP unsicher war, was nach dem Login und dem Laden einer Playliste zu tun sei. Nach kurzer Zeit kam die TP von selber darauf, dass das Menü geschlossen werden kann und es verschiedene Views gibt. Bei der erstmaligen Platzierung eines Tangibles versuchte die TP durch drehen des Tangibles die Schrift zu drehen. Nach dem Verschieben des Tangibles fiel der TP auf, dass die Schrift sich an der Mittelachse des Tisches ausrichtet um die Lesbarkeit von beiden Seiten des Tisches zu ermöglichen.

Positiv hervorgehoben wurde die Visualisierung um das Tangible. Insbesondere die Übersichtlichkeit und grafische Gestaltung gefiel der Testperson. Das Definieren eines Tangibles, sprich das Festlegen von Suchkriterien verlief problemlos.

Das Auslösen der Suche bereitete der TP Schwierigkeiten, da der Suchbutton im Zentrum des Tisches nicht als solcher erkannt wurde. Nach erkunden der Möglichkeiten betätigte die TP den Suchbutton letztlich und merkte nach Erscheinen der Suchergebnisse an, dass hinter dem Button "refresh" vermutet wurde. Unklar war der Testperson, dass die Ergebnisse durch eine "pinch"-Geste vergrößert werden können, woraufhin diese detaillierte Informationen über den Künstler anzeigen. Das Hinzufügen von Ergebnissen zur Playliste bereitete ebenso wie die gesamte Steuerung der Playliste keinerlei Probleme.

Das kombinieren mehrere Tangibles verlief problemlos, da die Testperson nach dem Anlegen zweier Tangibles diese zueinander schob und durch die Animation der Tangible-Kombination erkannte, dass diese zusammengehören.

Der kollaborative Test mit drei Versuchspersonen verdeutlichte mehrere Umstände. Einerseits zeigte es sich, dass die Versuchspersonen instinktiv natürlich miteinander Absprachen trafen während des Tests und sich gegenseitig anspornten. Andererseits kamen mehrere Verbesserungsvorschläge zu Tage. Die Interaktion mit Tangibles bereitete keinerlei Probleme, da sich die Versuchspersonen gegenseitig bei der Interaktion beobachteten und somit die Lernkurve extrem verkürzt wurde. Ebenfalls ergaben sich keinerlei Probleme bei der Definition von Tangibles durch die Menüführung und die Breadcrumbnavigation.

Auch im kollaborativen Test wurde der Search-Button nicht als solcher erkannt. Hierbei wurde ebenfalls das Icon bemängelt, welches eine falsche Assoziation beim Nutzer auslöst. Nichtsdestotrotz gelang den Testpersonen das Auslösen der Suche ohne Hinweise durch ausprobieren. Eine der Testpersonen machte die Anmerkung, dass das Anzeigen von mehreren Such-Buttons direkt bei den Tangibles wohl verständlicher für die Nutzer wäre.

Weiterhin wussten die TP nicht, dass durch Doppelklick auf Suchergebnisse ein kurzes Stück des angewählten Tracks angehört werden kann. Die Versuchspersonen vermuteten durch das Verschieben der Ergebnisse in das Zentrum des Tisches diese Abspielen zu können. Als Erklärung wurde diesbezüglich genannt, dass das Zentrum durch die Darstellung eines Lautsprechers zum markantesten Punkt wird. Auch hier wurden scheinbar falsche Assoziationen zu Tage gefördert. Hierzu kam der Vorschlag, alle Suchergebnisse mit einem Play-Button zu versehen. Beim weiteren Test des

Abspielens von Suchergebnissen viel auf, dass die Doppelklick-Geste öfters fehlschlug, da der erste Klick das Ergebnis leicht verschob, woraufhin der zweite Klick nicht mehr mit diesem in Zusammenhang gebracht wurde, weswegen letztlich das Abspielen fehlschlug.

Die Bedienung der Playliste und der Detailansicht der Artisten verlief ohne Probleme und wurde von den Testpersonen als gewohnt und eingänglich gelobt. Eine Testperson schlug vor, die angezeigten Songs einer Playliste mit weiteren Informationen, wie z.B. dem Genre, zu versehen, da diese Information in der Detailansicht nicht besonders hervorgehoben wird.

Ganz allgemein fiel beim kollaborativen Test auf, dass die Performance des Multitouch Tisches zeitweise nicht aussreichend zufriedenstellend ist. Vorrangig bei gleichzeitiger Interaktion der Nutzer und schnellen Wechsel der Views geriet die Darstellung ins Stocken.

7 Ausblick

Zu Beginn des Projekts war angedacht, für Neueinsteiger ein Tutorial anzubieten. Aufgrund des allgemeinen Funktionsumfangs wurde auf diese Hilfestellung verzichtet. Bei der Usability-Evaluation ergaben sich einige Probleme, die es zu beheben gilt. In den meisten Fällen handelt es sich dabei um die Darstellung des aktuellen Zustands der einzelnen Entitäten mit denen der Nutzer interagieren kann. Allerdings traten auch Probleme mit Gesten für bestimmte Funktionen und Anfangsschwierigkeiten bei der Interaktion mit Tangibles auf.

Ein weiteres Manko der Anwendung stellt die Performance dar. In Hinblick auf die Interaktion von mehreren Usern und der daraus resultierenden Fülle an zu verarbeitenden Handlungen tritt die Hardware schnell an ihre Grenzen. Dies ist vor allem bei der vielfachen Kombination von Tags zu beobachten, da hier ein Performanceproblem der Kugelanimation nicht behoben werden konnte. Eine intensivere Beschäftigung mit dem Thema Performance würde so eine ruckelfreiere Interaktion mit dem System bieten.

8 Anhang

Quellenangaben:

- Coding Sources (%\documentation\attachement\CodingSources.pdf)
- Image Sources (%\documentation\attachement\ImageSources.pdf)

Sonstiges:

- Video (https://www.dropbox.com/s/onfii7spp5xwqd9/Demo_Video_720.mp4)
- Screenshots (%\documentation\attachement\Screenshot_1.jpg,
  %\documentation\attachement\Screenshot_2.png