Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Extending the Personal Music Collection by Free Music

Bachelor's Thesis

Dominic Langenegger

dominicl@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Samuel Welten
Prof. Dr. Roger Wattenhofer

June 10, 2012

Acknowledgements

My thanks go to my supervisor *Samuel Welten* who made it possible for me to work on the Android music player *jukefox*[1] and extend it with some work of my own.

Many other people helped me during the development and writing process of this thesis with their expert knowledge, support and feedback. I would like to thank all of them.

During the development of this extension, I used some libraries of others to simplify my work. I'd specially like to thank the developers of the following openly licensed libraries:

(i) Google Gson, a simple and fast JSON serializer and deserializer. [2]

(ii) SAX, the *Simple API for XML*, and its Java version provide an interface for very fast and efficient event based XML parsing. [3]

(iii) MySQL Connector/J, an uncomplicated MySQL driver that handles JDBC (Java Database Connectivity) calls to access a MySQL database. [4]

[1] http://www.jukefox.org/
[2] http://code.google.com/p/google-gson/
[3] http://www.saxproject.org/
[4] http://dev.mysql.com/usingmysql/java/

Abstract

Today, people listen to music on their mobile phone or other playback devices. *Jukefox* is a special music player running on *Google's* smart phone operating system *Android*. It's capable of recognizing similarities between multiple songs, albums and artists using data based on a huge music similarity map [1] which classifies songs into a high-dimensional space. Songs with small distance are similar, while widely separated coordinates mean, that two songs are very different.

This thesis shows how the service of *jukefox* can be extended with the functionality to explore new music that is similar to the one the user already has in his own music collection. It describes, how artists and songs that fit into the current mood and the overall music taste of a user, can be found based on the already available music similarity map and how a user can listen to them without the need of downloading every single piece of music manually.

As sources for new music, services like *Jamendo*[5], the *Free Music Archive*[6], *ccMixter*[7] or *SoundCloud*[8] with music licensed under the *Creative Commons* perfectly qualify. In this thesis a proof-of-concept implementation for *Jamendo* as music source is introduced, while the architecture and design allows for simple extension by other sources.

Keywords: music taste, mobile, android, streaming, creative commons, extending music collection, jukefox

[5] http://www.jamendo.org
[6] http://freemusicarchive.org/
[7] http://ccmixter.org
[8] http://www.soundcloud.com/

Contents

Chapter 1

Introduction

This chapter introduces the background of this work. It gives a short overview on the idea of *Free Music* and its impacts. Furthermore it shows some related works, mainly from commercial applications that already exist.

1.1 Motivation

Over the last years, the market for mobile smart phones has grown rapidly. This opened many new possibilities for users. Many people are listening to their music in digital form on their smart phone.

At the same time, the music industry has changed a lot. When users were buying physical compact disks to get the songs of their favorite band 10 years ago, the trend nowadays goes towards downloading and streaming over the internet (legally or illegally).

With the popularization of the internet, the spreading of new music has become a lot easier for artists. Many artists over the whole world have chosen to make their music freely available under an open license such as a *Creative Commons License*. However, it remains very difficult for users, to keep track of the huge amount of music that is available. And it's even harder to find new music that pleases the user's taste and mood.

This thesis introduces an extension of the *Android* music player *jukefox*, that opens the possibility to explore new music which is both freely available and fitting in the music taste of the user.

Jukefox classifies music tracks into a high-dimensional coordinate system where small euclidean distances stand for high similarity of songs. This so called *Music Similarity Map* provides the perfect footing for finding new songs that are similar to those, a user already has in his collection. If there are some - to the user yet unknown - songs near to an accumulation of songs in the user's music collection, they are potentially fitting very well into the user's music taste.

The aim of this thesis is to use this situation to suggest new music titles to users and let them obtain them for use in their own music collection.

1.2 Free Music

This section introduces the idea of free music. Contrary to what the words *Free Music* suggest, the word "free" does not refer to price but to freedom of copying, distributing and modifying for personal and noncommercial purposes. [1]

However for our purpose, we obviously prefer *Free Music* that is readily available without any cost to the user. This allows for unrestricted use for streaming and downloading whenever a user feels like listening to new music.

There exist several copyright licenses for *Free Music*. The most common ones are the *Creative Commons Licenses*. [2] The creative commons organization provides six basic licenses that differ in small points like commercial distribution or redistribution under the same license. Their goal is to make it easier for people to share and build upon the work of others.

Many media portals over the whole internet offer artists the possibility to share their work under creative commons licenses. Some of them are listed here:

Jamendo
Jamendo has a collection of about 350,000 songs in various genres under creative commons licenses.
http://www.jamendo.com/

SoundCloud
SoundCloud is - according to their own website - "the world's leading social sound platform where anyone can create sounds and share them everywhere." Many artists that offer their music on SoundCloud license it under one of the Creative Commons Licenses.
http://www.soundcloud.com/

Free Music Archive
The *Free Music Archive* offers free high-quality music downloads. It tries to offer a similar service to what public Radio always did and still does - free access to new music - in a way that fits into the age of the internet.
http://freemusicarchive.org/

ccMixter
The "cc" in *ccMixter* stands for *Creative Commons* and that's exactly the

[1] The Free Music Philosophy v1.4
Available at: http://www.ram.org/ramblings/philosophy/fmp.html [Accessed June 30, 2012]
[2] Creative Commons Website
Available at: http://creativecommons.org/ [Accessed June 1, 2012]

dedication of that project. Its another platform where users can upload their creations and retrieve those of others - all under Creative Commons Licenses.
http://ccmixter.org/

In the extent of this thesis, *Jamendo* served as the source for free music in the proof-of-concept implementation.

1.3 Related Work

This section covers some preexisting services that offer a similar functionality what this work adds to *jukefox*. The main aspect here is, that a compared service offers the possibility to find new music that is similar to the collection one already possesses.

iTunes Genius [3]

With *iTunes*, *Apple* has a widely used music player for both *Windows* and *Mac* operating systems. It's feature *Genius* is capable of automatically generating a playlist of songs from the users music collection which are similar to a selected song. The same feature is also used to make recommendations for purchases in the *iTunes Store*, where the user can directly buy and obtain new music.

Last.fm [4]

Last.fm's whole concept is, to recommend users new music, that most likely fits into their music taste. It learns about the music taste of a user with so-called "scrobbles", which contain information about a listened song and are sent from a music player that has "scrobbling"-support or their own radio services to *last.fm*. On the website, a user can then see recommendations of songs, albums and artists.

Spotify [5]

Spotify offers digitally restricted music streaming of multiple millions of songs from different record labels all over the world for a monthly fee - or for free with usage time limitations and ads. The service offers multiple recommendation methods. One is a dynamic radio station that plays songs similar to those of a selected artist or fitting into a picked genre.

[3] iTunes Genius on the iTunes features website
Available at: http://www.apple.com/itunes/features/#genius [Accessed June 2, 2012]

[4] Last.fm website
Available at: http://www.last.fm/ [Accessed June 2, 2012]

[5] Spotify website
Available at: http://www.spotify.com/ [Accessed June 2, 2012]

The main difference between most of these highlighted similar services and the one introduced with this thesis, is that they don't provide freely available music. They either require a user to already possess songs or to buy the right to listen to them.

Chapter 2

Jukefox

This chapter introduces *jukefox*, a music player that was developed at the *Swiss Federal Institute of Technology Zurich*[1] (ETH Zürich) for *Google's* smart phone operating system *Android*. Furthermore a brief overview of *Android* and the possibilities of application development for it is provided.

2.1 Android

In 2008, *Google* [2] introduced *Android* [3], an operating system for mobile devices like smart phones and tablets. It is based on a *Linux* kernel and developed by the *Open Handset Alliance*[4]. Since the system is widely open for modifications and customizations (released under the *Apache License* [5]), it's a popular choice for smart phone manufacturers and an often chosen target system for applications by third party developers. Applications for the *Android* system are primarily written in *Java*[6].

Developers can take advantage of the huge *Software Development Kit* [7] (SDK) *Google* offers for *Android*. Many functionalities to access the devices resources or do often used tasks, like music streaming, are integrated into the operating system. Therefore a developer can move on a relatively high level of abstraction and is able to focus on his own project instead of having to deal with low level details.

[1] http://www.ethz.ch
[2] http://www.google.com/
[3] The philosophy of Android on their web page
Available at: http://source.android.com/about/philosophy.html [Accessed June 2, 2012]
[4] http://www.openhandsetalliance.com/
[5] http://www.apache.org/licenses/
[6] http://www.java.com/
[7] Android SDK overview on their web site
Available at: http://developer.android.com/sdk/ [Accessed June 2, 2012]

2.2 Music Similarity Map

One of the nearly 320,000 applications (as of September 2011 [2]) that are available in Google's *Play Store* [8], the official online market place for *Android* applications, is *jukefox*.

Jukefox is not only a standard music player. The workings of it, are based on a huge, so called *Music Similarity Map*, which offers coordinates in a 32 dimensional euclidean space for over 1.1 million songs and thousands of artists. These coordinates were built using probabilistic latent semantic analysis (PLSA) on social tags and listening behavior of songs on *last.fm*. [1] Other works like [3, 4] or [5] are solely based on acoustic measures for music classification and miss the social aspect used for *jukefox*.

Out of this work resulted a music similarity map, that classifies songs into an euclidean space where points that are in small distance to each other, suggest a high probability that they are similar. Based on this, it's possible to do similarity searching by finding near(est) neighbors to some given point.

2.3 Music Taste

Ongoing research tries to classify and describe a users music taste in some mathematical way. With a music similarity map, like the one *jukefox* offers, at hand, an approach using a k-means algorithm (like the one by Stuart Lloyd from 1982 [6]) is one possible choice. This method performs cluster analysis by trying to classify a number of given objects, here songs or artist with their coordinates, into k groups.

Thanks to the work of my supervisor Samuel Welten, *jukefox* is capable of doing exactly this for a given set of weighted coordinates. The so called *SimpleMusicTaste* that results out of this analysis, contains multiple class centers and radii, building up multiple hyperspheres embracing clusters that are representative for the given set of points.

When initiated with song or artist coordinates of a users music collection, the result is a reasonable description of the users music taste. To rate a song or artist in comparison to the calculated music taste, the relative distance of it to the nearest class center in comparison to the class centers radius, is a fair measure. This is used in this thesis to rate the similarity of possible artists for suggestion (see section 5.1.2).

[8] https://play.google.com

Chapter 3

Jamendo

Jamendo [1] is an online platform that offers artists on the whole world a place to share their creations. A major aspect of the platform's philosophy is free music, namely the *Creative Commons Licenses* (See section 1.2), under which all work available on *Jamendo* is licensed. The following sections give a brief overview of the music collection *Jamendo* offers and their interface for third party services to gather information from their database.

3.1 Music Collection

In table 3.1 an overview of *Jamendo's* music collection and *jukefox'* data set about music, is shown. One of the main criteria when selecting a music provider for this work, was the cardinality of the available data set. Since the music suggestion methods used are based on artists rather than tracks (see section 5.1), the critical value is the number of artists available on *Jamendo*, that already have coordinates in the *jukefox* database. As of June 2012 this number is 8,108 which is about 30% of all *Jamendo* artists.

	Jamendo	Jukefox	Jukefox with Coords	Both
# Artists	27,061	2,703,315	1,246,901	**8,108**
# Albums	55,545	2,390,874	n/a	n/a
# Tracks	346,351	12,036,631	1,119,450	n/k[2]

Table 3.1: Comparison of Music Collections as of June 2012

[1] Offical Jamendo Website
Available at: http://www.jamendo.com/ [Accessed June 2, 2012]

[2] Not known since not determined

3.2 The Free Music API

To retrieve detailed information about tracks, albums and artists, *Jamendo* offers a simple *Application Programming Interface* (API) with support for many different output formats like *JSON*, *XML* or even plain text. It is internally called "The Free Music API" [3] and represents a direct interface to their backend database. Queries are very similar to a standard *SQL SELECT* and include a list of fields, tables, optional joins and filters together with the return format.

This provides a huge opportunity and a simple usage inside of the *jukefox* mobile application. As explained in section 5.2.1, "The Free Music API" is used to gather information about available albums and tracks for suggestion to the user.

[3] Overview of the jamendo.get2 API
Available at: http://developer.jamendo.com/de/wiki/Musiclist2Api/ [Accessed June 4, 2012]

Chapter 4

Goals

The Goal of this thesis, is to extend the *Android* application *jukefox* with a feature that can suggest new, free music to a user based on his music taste. This extension should be easy to handle and provide a clear user interface.

The user should be able to see a list of suggested albums from which he can select the ones he would like to explore. When he is indicating that he wants to listen to some of them, *jukefox* should automatically generate a playlist and begin streaming the music over the internet. If a song or album pleases the taste of the user, he should be able to either listen to more of the same artist or directly download whole albums into his music collection for later offline use.

With this extension, *jukefox* should not change into a streaming player that offers extensive search and exploring features. This means, it should not act as a client to some streaming portal(s), but rather as a special music player, that is capable of suggesting new music to the user that he might like.

Chapter 5

Realization

The following sections provide insight in the functionality and methods of this *jukefox* extension. The first sections explain details of the tasks done on a web server and the following ones cover client modifications.

5.1 Server

Since the *jukefox* application itself doesn't have the whole coordinate data set locally available, a server has to provide some services to enable a good recommendation system. Additionally a server can take care of calculations that would require a lot of performance on a smart phone, and would therefore drain the battery excessively.

The basic task of the server for this extension is, to know about all free music that could possibly be interesting for any kind of user and could directly be streamed to a device. When a user provides his music taste to the server, it should then find good matches of free music.

The next subsections will introduce the methods used to enable *jukefox* with this functionality.

In a first phase of planning we decided to base the recommendation system on artists rather than songs. Songs would most likely have allowed for more precise matches but artists provide a bigger source of data once one is chosen. Additionally most artist write most of their songs in a similar genre.

5.1.1 Database Import

Jukefox already provides a huge data set of coordinates for more than a million songs and artists. For external artists to be comparable to a music taste of a user, their coordinates need to be known. Therefore only artists whose coordinates in the music similarity map are known are interesting for our application.

As a first step of development we have to learn about available free music. This contains the following tasks:

(i) gather data about what music is available for free

(ii) see what subset of this music is present in the *jukefox* database with coordinates

(iii) store this subset for later use

The following sections describe the details of each task and how the whole process can be extended to easily support for incremental updates.

Data gathering

Luckily the selected music provider *Jamendo* has full database dumps in *xml* format ready for download. Those even get updated every work day.

We implemented a small application to automatically retrieve the newest dump and process it using a Java SAX library.

If there was no such complete source of data available, data gathering could have been much harder. However many other providers of free music have full API support, which could be used to get an idea of which artists are offering their music for free. In a worst case scenario, a web crawler could be used to scan the web interface of a music provider for available music.

Data processing and storage

The main task of data processing is, to select the subset of artists that are already known to *jukefox* and have an own place in the music similarity map. Artists that fulfil this requirement can then directly be stored into a database.

For so called *external artists* we chose to only store their id in the music provider database and a unique identifier for the music provider together with a link to the artist entry that is already present in the *jukefox* database. This saves storage capacity and still offers the possibility to uniquely identify him for later use.

Data updating

Since Artists may close their account or new artists may provide new music, the *Jamendo* database may change very frequently. Therefore we included a method to update our data set.

One method to update the *Jamendo* data, was to always keep the last version of the xml dump until a new one is processed. With the SAX parser, it's possible to build a fast data structure like a hashset containing all artist ids, in very short time. When processing the new dump, one can simply determine if an artist was already in the last dump and therefore no action has to be taken; or if he wasn't present and has to be inserted. With a loop through all artists of the old dump version, it's easy to see what artists aren't present anymore and therefore have to be removed from the database.

A benefit of this approach, is that it's not database intensive and therefore doesn't affect real-time and critical applications. It allows for regular execution by a job scheduler like cron[1] or runwhen[2] to keep the database up to date.

5.1.2 Application Programming Interface

For *jukefox* to retrieve a list of similar artists, the server has to provide an interface which accepts special queries. We decided to use the *Java Script Object Notation* (JSON) as the data format for all queries and responses.

For the implementation a Java HTTP-Servlet [3] was deployed on a Tomcat[4] web server.

Request

The android phone sends a query containing the user's music taste (see section 2.3) and a blacklist of artist identifiers that are already present in the user's music collection and therefore should be excluded from the result set.

Data Processing

A reasonable approach to find the best-fitting artists for a submitted music taste would be to do a k-nearest neighbor search on the class centers of the music taste with all available artist coordinates as data set. Some thoughts about k-nearest neighbor search and similar artist finding are listed here:

[1] Crontab reference
Available at: http://pubs.opengroup.org/onlinepubs/9699919799/utilities/crontab.html [Accessed June 5, 2012]

[2] The runwhen website
Available at: http://code.dogmap.org/runwhen/ [Accessed June 5, 2012]

[3] Oracle about Java Servlets
Available at: http://www.oracle.com/technetwork/java/overview-137084.html [Accessed June 2, 2012]

[4] Apache Tomcat Website
Available at: http://tomcat.apache.org/ [Accessed June 2, 2012]

(a) The performance of exact k-nearest neighbor algorithms usually decreases very fast with increasing feature set dimension count. Since *jukefox* uses a 32 dimensional coordinate system, solutions using, for example, kd-trees or R-trees are likely to degenerate to a complexity of $O(n)$ with n being the number of data points. [7] However some work using a VA$^+$-file technique from 2004 by Ferhatosmanoglu, that seems to be capable of doing more efficient nearest neighbor search in high dimensions, exists.[8]

(b) Many data structures that allow for fast nearest neighbor searches are memory intensive. Each data point has to be stored in memory together with some additional overhead for the data structure itself. Especially methods like locality sensitive hashing [9, 10], which would be very fast even for high dimensional spaces, have big space requirements.

(c) Even if performance and space measures are completely ignored, a result set should include some sort of randomness to provide a client with new data on every new query.

The final approach taken, was to use an artist cache containing only a limited number of external artists with their coordinates. From this random subset, the best fitting artists (about 20 - 50) are chosen using the distance from an artist to the nearest class center of the submitted music taste. The selection process prefers artists whose music was more often listened to, according to the *jukefox* play logs, over those that have a smaller listen count. By periodically replacing the artists in the cache by new ones, a certain degree of randomness is provided.

The current implementation uses a greedy algorithm on the data set in the cache. Since the cache is rather small (200-500 artists) a query is still fast (multiple measurements showed processing time of < 50 ms, on an Intel Core 2 Quad system at 2.3 GHz, per query). Other approaches may result in better performance and would be capable to run on bigger data sets, but the detailed evaluation of the methods on-hand exceed the scope of this work and is still an interesting and ongoing research area.

Response

Out of the selected artists, only a unique identifier for their data provider, their name and their rating according to their listen count and the music taste of the user are present in the response. All further work is done by the client application.

5.2 Client

The following sections explain the tasks of the *jukefox Android* application in order to provide the user with new free music. The client has to deal with *a*) calculating the users music taste and querying the server API for similar artists; *b*) finding albums of these artists and presenting them to the user; *c*) providing the possibility to directly stream the music; and *d*) downloading and integrating music in the music collection of the user.

5.2.1 Finding similar Artists

Since the suggested music should not only fit into the users basic music taste but also his current mood, *jukefox* by default uses the last 100 listened songs to calculate a music taste. This way only the songs that were recently played and reflect the current mood of the user are considered. Additionally it automatically generates a small variation between multiple queries if music was listened to in between. This music taste is then used to retrieve a list of artists via the server API (see section 5.1.2).

5.2.2 Result Presentation

After getting a set of artists that fit the music taste of a user, they have to be preprocessed for presentation to the user.

The built recommendation system supplies a list of albums. They are presentable in a good way by including their album cover. Out of this list, the user can select those albums that seem promising to him and let *jukefox* generate a random playlist out of the songs of all selected albums.

A possible appearance of this list of albums can be seen in figure 5.1.

Good Music

A major aspect of the album list presentation is the gathering of good albums to present. As a base only the list of artists and their rating from the API call is available.

A possible approach is to get a list of albums for every artist and select a limited number of them for display. *Jukefox* uses exactly this method while taking advantage of *Jamendo's* own rating system, based on user interaction, that is directly accessible over API calls. This opens the possibility to only select the albums with the highest *Jamendo* rating.

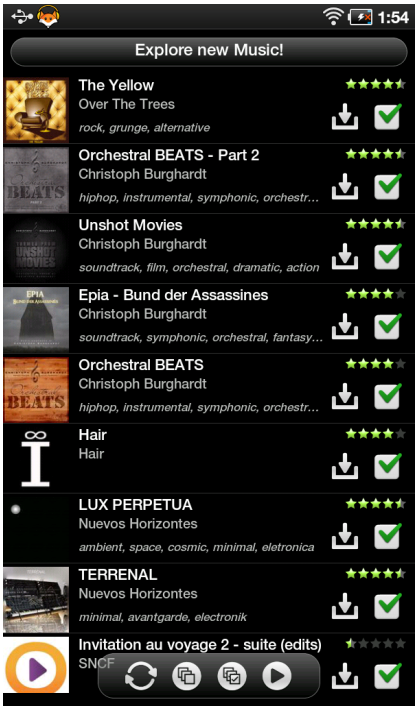


Figure 5.1: The album list containing suggested albums.

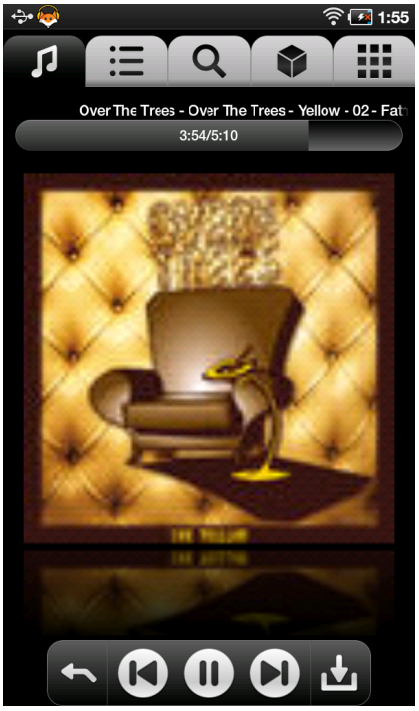


Figure 5.2: The jukefox music streaming view.

In a huge music set like the one *Jamendo* offers, there is always music that isn't of very good quality. It's a promising approach to use user data to determine which music could be prosperous. That's why we chose to rely on this external rating system to increase the quality of the albums *jukefox* suggests to users.

Together with the basic information like album name and artist name, an album cover, a list of social tags, the album rating and a download button (more about downloading of albums in section 5.2.4) is displayed for each album in the list. All album covers are downloaded asynchronously and cached on the phones memory card for recurring use.

5.2.3 Music Streaming

With the overview of suggested albums, the user is only one step away from listening to some new music. With a click on the play button in the control elements, *jukefox* automatically starts generating a playlist out of the best tracks of all the selected albums. With *Jamendo* as its data source, it again uses their provided song ratings. As soon as the playlist is generated, the player changes to the special music streaming view and starts playing the streamed songs. An example of such a playlist can be seen in figure 5.3.

Figure 5.2 shows the player with two special control elements to both sides of the track jump buttons. The left one allows to abort exploring music and returns to the standard player while clearing the playlist. The right button opens a download dialog like in figure figure 5.4.

5.2.4 Music Download and Integration

If a song pleases the taste of the user, he may want to get more of the same kind. With a click on the download button a dialog as seen in figure 5.4 shows up. This opens the possibility to either directly download the whole album into the own music collection, listen to the whole album the song originates from or see a list of all work the writer of the current song did. An example of such a list can be seen in figure 5.5.

The downloading process takes place in the background. Nevertheless the user is constantly informed about the current progress over a notification in *Android's* notification center. A download in progress may look as shown in figure 5.6

To complete the integration of downloaded songs into the own local music library, *jukefox* needs to scan the files and import them as it did with all the music that is already in the collection. This includes retrieving detailed data like coordinates for every song and can possibly take some time.

Figure 5.3: Example of generated playlist.

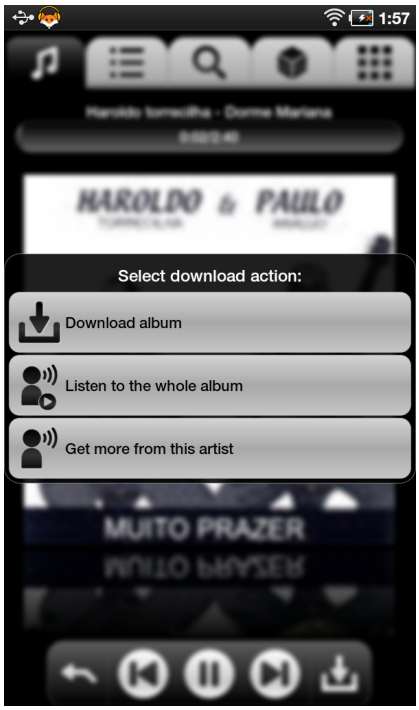


Figure 5.4: The download dialog.

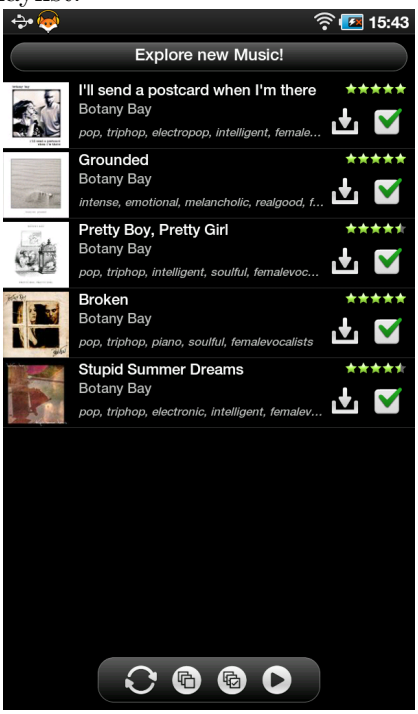


Figure 5.5: List of albums by one chosen artist.

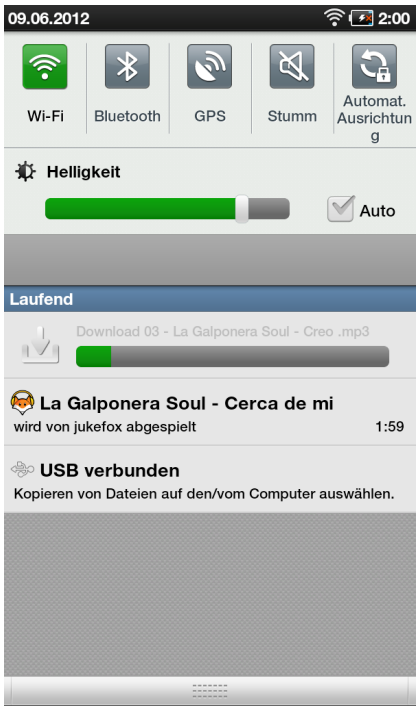
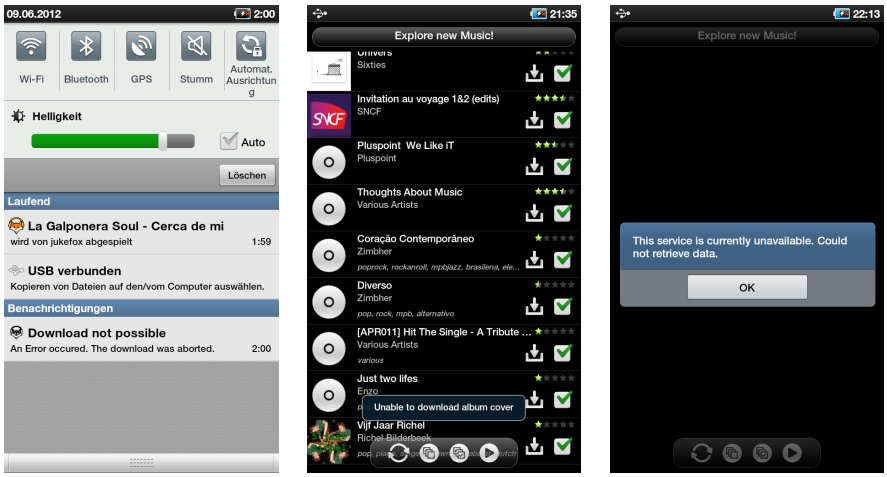


Figure 5.6: An album download in progress.

5.2.5 Error Handling

For special situations like the loss of the internet connection or if a server is not reachable, some error messages are provided to keep the user informed about what is happening. Examples are visible in figure 5.7.



(a) Error notification after a failed download.

(b) Error fetching album covers.

(c) API call failed.

Figure 5.7: Examples of error messages in subfigures (a), (b) and (c).

Chapter 6

Future Work

Jukefox is a ongoing research project that additionally pleases thousands of users over the whole world. This work adds the basic functionality to let a user explore new songs over the internet. In this chapter, some possible improvements and extensions are discussed.

6.1 Possible Improvements

6.1.1 Improve quality of suggested albums

The current approach of finding similar artist uses a greedy algorithm on a reduced data set. This is not optimal and could be improved with some novel techniques such as locality sensitive hashing. The application of locality sensitive hashing in the huge data set *jukefox* offers, could open an interesting research project. Especially a combination with algorithms to rate the quality of a given resource is promising and could evolve into a qualitatively good similarity feature.

Even when *jukefox* got a list of artists that are similar to the present ones, it's a hard task to find the best subset of available albums and songs for these artists that should be suggested to the user. This functionality currently relies on a rating provided by an external source. It would be interesting to see if the data *jukefox* itself has available - or an intelligent combination - could result in better results. Especially if other music providers would be included, some sort of common measure should be implemented.

6.1.2 View updating

In some cases, the album list view fails to correctly update when either the refresh button is clicked or all check boxes are checked or unchecked via the respective buttons. Unfortunately this behavior could not be tracked down by the time, this thesis was submitted.

6.2 Extensions

Apart from possible improvements, the introduced music exploring feature of *jukefox* could be extended in various places. Some possible future extensions are described here.

6.2.1 Include more data sources

With the current implementation, *jukefox* only uses data from *Jamendo* to find and offer similar music. Like indicated in section 1.2, there are various providers of free music available. Some offer very good music in certain genres. Together they could complement each other to offer a big and solid set of music for suggestions to the user.

A possible extension would be to include more of these services into *jukefox* and therefore offer better and more extensive music suggestions.

6.2.2 More possibilities for similar music finding

It would be a novel feature, if a user could request similar free music for every song, album or artist he has in his music collection. This could be implemented by initializing the *SimpleMusicTaste* with only a certain set of coordinates that correspond to the users selection.

6.2.3 Artist blacklisting

If a user stumbles upon a song he doesn't like at all, he might prefer to never be suggested songs of the same artist again. Therefore the current implementation is lacking a custom blacklisting feature for external artists, which would be easy to add and might save users some pain.

6.2.4 Quality testing

At the moment there is no measure used to rate the quality of the suggested songs according to their real similarity. Mathematically most of them are similar according to the 32 dimensional music similarity map, but it would be interesting to have real people rate the similarity of songs in a study. This could allow for improvement of the similar music finder or even the music map itself.

Chapter 7

Conclusion

Jukefox already offers a lot of functionality and is continuously extended. This extension was implemented to serve as a proof-of-concept that *jukefox'* underlying logic, especially the music similarity map, is suited to suggest new and free music to the user.

With this extended music player, it's now possible for a user to listen to new music with just a few clicks and no big effort. He doesn't have to browse around in the web and search for songs he might like. This whole service is available on *Android* smart phones with internet connection. As hours of testing during the development phase have shown, the music - in most cases - even is a good match to the users music taste.

This is a valid contribution, since few services exist that can offer a similar feature for free.

Bibliography

[1] Kuhn, M., Wattenhofer, R., Welten, S.: Social Audio Features for Advanced Music Retrieval Interfaces. In: ACM Multimedia, Florence, Italy. (October 2010)

[2] research2guidance: Android Market Insights September 2011. In: research2guidance Android Market Insights. (September 2011)

[3] Tsunoo, E., Tzanetakis, G., Ono, N., Sagayama, S.: Audio genre classification using percussive pattern clustering combined with timbral features. In: ICME. (June 2009)

[4] Slaney, M., Weinberger, K., White, W.: Learning a Metric for Music Similarity. In: International Conference of Music Information Retrieval (ISMIR). (September 2008)

[5] Pohle, T., Schnitzer, D., Schedl, M., Knees, P., Widmer, G.: On Rhytm and General Music Similarity. In: International Conference of Music Information Retrieval (ISMIR). (September 2009)

[6] Lloyd, S.: Least squares quantization in PCM. In: IEEE Transactions on Information Theory. (March 1982)

[7] Indyk, P., Goodman, J.E., O'Rourke, J.: Nearest Neighbors In High-Dimensional Spaces. In: Handbook of Discrete and Computational Geometry. (April 2004)

[8] Ferhatosmanoglu, H., Tuncel, E., Agrawal, D., Abbadi, A.E.: High Dimensional Nearest Neighbor Searching. In: Information Systems Journal. (December 2004)

[9] Paulevé, L., Jégou, H., Amsaleg, L.: Locality sensitive hashing: a comparison of hash function types and querying mechanisms. In: Pattern Recognition Letters. Volume 31., Elsevier (August 2010)

[10] Gionis, A., Indyk, P., Motwani, R.: Similarity Search in High Dimensions via Hashing. In: 25th VLDB Conference. (June 1999)

Appendix A

Appendix Chapter

A.1 Database Structure

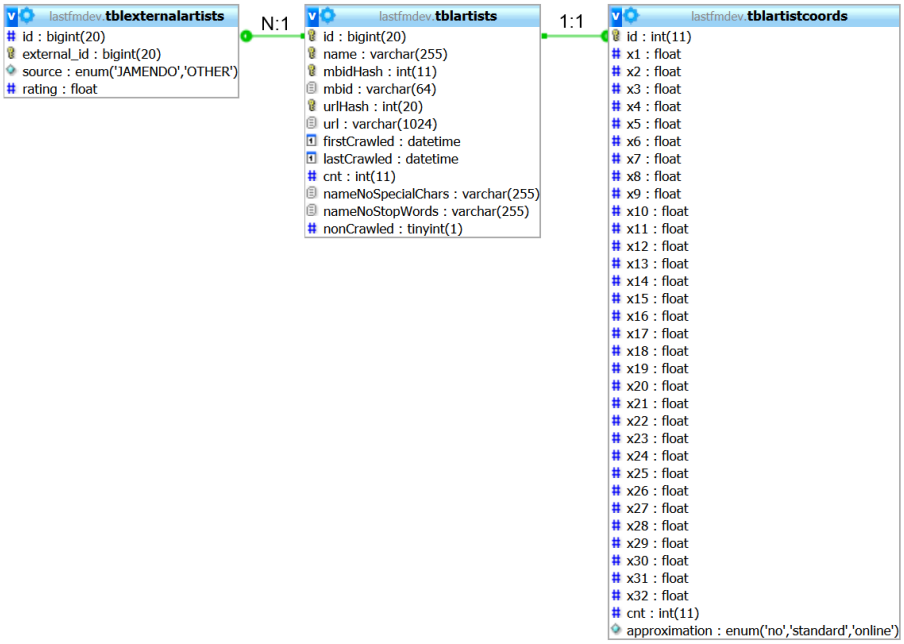


Figure A.1: Database structure on the server showing the relation between the new external artist table and the preexisting artist tables