Desktop-Client with Smart Synchronization

Bachelor's Thesis

Sandro Affentranger

sandroaf@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Samuel Welten, Tobias Langner
Prof. Dr. Roger Wattenhofer

July 1, 2013

Acknowledgements

There are many people who supported me while I have worked on this thesis. Unfortunately I cannot thank all of them here.

First of all I want to thank my supervisors Samuel Welten and Tobias Langner for their support. During our weekly meetings they gave me useful feedback and helped me with their knowledge if I was stuck.

I also want to thank Dr. Roger Wattenhofer for giving me the chance to make my bachelor's thesis at the Distributed Computing Group and working on this unique and interesting project.

Furthermore I also want to thank my roommate. Our usually rather long discussions during our morning coffee gave me often yet another view on problems.

And last but not least I want to say thank you to my girlfriend. Her moral support was very important for me, especially during the more stressful phases.

Abstract

Nowadays people use their smartphones for listening to music on their way. Unfortunately most smartphones do not have enough memory to store a whole music collection on them. The user has to decide which songs he wants to have on his phone and which ones not. If he buys new albums or his music taste changes over time, he has to synchronize his smartphone once again.

This thesis shows how to develop an extension for *jukefox*, a smart music player, which enables the music synchronization between the desktop jukefox version also developed during this thesis and the android application. The extension allows the user to browse his whole music collection on the smartphone, mark songs to add or to delete and synchronize them without being connected to the computer where the music is stored.

In addition a smart synchronization mode was implemented which analyzes the music listening behavior of the user in order to automate the synchronization process. It takes into account which songs were listened often and finds similar songs in the music collection of the user which are not yet on the phone. It regards also songs which were not listened at all. Afterwards it makes suggestions to the user who can then either accept or discard them.

Keywords: jukefox, android, desktop jukefox application, music synchronization, smart synchronization

Contents

Chapter 1
Introduction

1.1 Motivation

Today people use their smartphones to listen to their music and no longer MP3 players. After all they just need to carry around one device with themself instead of two. The downside is that the new smartphones often have insufficent storage space for the whole music collection. In the future it will get even worse, since several smartphone manufacteres have announced that their future devices will no longer have a SD card slot anymore. Even Google[1] tries to restrict removable storage.[2]

The music industry tries to bypass this issue with the concept of music streaming. Instead of having music physically stored on the smartphone, you could just stream it through the internet. Usually for a small monthly fee. The advantage of this is that music streaming providers like Spotify[3] offer millions of songs and the users are not bound to just their music collection.

But there is one big drawback of this new trend. It requires a good and fast internet connection to stream songs effectively. But even today there are some regions where the mobile reception is rather bad or not existent. And even if you live in a region with a good mobile reception, you have to pay for the network traffic caused by streaming. Most mobile service providers do not allow unlimited traffic in their current mobile subscriptions.

A good compromise would be if you could access your whole music collection and download songs on demand. If this would be done when connected via WLAN, it can be done without affecting the used traffic of ones mobile subscription. Still, with a cordless synchronization like this, the synchronization itself is a rather tedious work. It should be possible to exploit some common synchronization behavior in order to automate it. Songs that are often skipped could

[1]https://www.google.com/ [25.06.2013]

[2]Why Nexus devices have no SD card: http://www.androidcentral.com/why-nexus-devices-have-no-sd-card [25.06.2013]

[3]https://www.spotify.com/ [17.6.2013]

get deleted from the smartphone, while similar songs to often listened songs get added to it.

1.2 Related Work

There exist several applications which provide similar functionalities as synchronization via WLAN or creating smart playlists. They are briefly introduced in this section.

Winamp:[4]

The android version of *Winamp* enables the synchronization via USB or WLAN. The music can be downloaded to the smartphone from the *Winamp Desktop Player* as long as the smartphone and the computer are in the same network.

iTunes Genius:[5]

With *Genius* it is possible to create smart playlists in *iTunes*. Smart means here that the songs harmonize well together. These playlists can be synchronized with an *iPhone* together with all contained songs.

iSyncr:[6]

iTunes only supports the synchronization with *iPhones*. For android users exists therefore *iSyncr* which enables it nevertheless. The user can browse his songs and playlists of *iTunes* on his smartphone and select them for synchronization. The synchronization works via USB or WiFi even if the smartphone is not in the same network as the computer.

Rocket Music Player:[7]

The *Rocket Music Player* is a music player from the same team as *iSyncr*. It can be combined with *iSyncr* in order to synchronize music and playlists directly into the music player.

Although the synchronization of music via WiFi is supported by several android applications, it exists no application that automates it.

If *iTunes Genius* gets combined with both the *Rocket Music Player* and *iSyncr*, one could create smart playlists and synchronize them directly into the music player. This could relieve the synchronization process by enabling the user

[4]https://play.google.com/store/apps/details?id=com.nullsoft.winamp [19.06.2013]
[5]http://www.apple.com/itunes/ [24.06.2013]
[6]http://www.jrtstudio.com/iSyncr-iTunes-for-Android [19.06.2013]
[7]https://play.google.com/store/apps/details?id=com.jrtstudio.AnotherMusicPlayer [19.06.2013]

to synchronize well matched songs in one go. However would still be complicated and require some work of the user.

What we will tackle in this thesis is to enable an easy synchronization directly into *jukefox* without a complicated configuration. Furthermore we do not stop with an easy synchronization but try to facilitate it by helping the user with the decision which songs he should synchronize.

1.3 Goals

The goals of this thesis are on the one hand to develop a desktop version of *jukefox* with a graphical user interface, which should also act as server and provide music. And on the other hand to extend the android application of *jukefox* in order to allow a synchronization with the desktop version.

Once registered, the user should be able to browse his whole music collection on the smartphone and see which songs are currently on the smartphone and which are just stored on the jukefox server. He also should be able to download music from the server which then will be added to the smartphone.

The synchronization should also be possible from the server, where the user could also manage all registered smartphones. He should be able to add and delete songs, albums and artists.

Furthermore a smart synchronization should be developed, which takes over the rather exhausting manually synchronization. It should analyze the listening behavior of the user and suggest which songs he could delete from his smartphone and which songs could be added to it. Since the storage space for the music on a smartphone is usually very limited and has to be shared with other things like photos and applications, the user should be able to specify how big his music collection on the smartphone may maximally be. The smart music synchronization process should regard this then and take care that this upper bound will not be exceeded.

Chapter 2

Jukefox

Jukefox, a smart music player, is an ongoing research project at the *Swiss Federal Institute of Technology Zurich*[1]. It was originally developed particularly for *Google's* operating system *Android*[2]. In this chapter we introduce *jukefox* and give a short overview of *Android*.

2.1 Android

Android has become really fast very popular as an operating system for mobile devices. In the first quarter of 2013 about 75%[3] of all world-wide shipped smartphones run on *Android*.

Applications for *Android* are programmed in most cases in a customized version of *Java*[4]. This eases the programming of a communication protocol between an *Android* application and computer program written in *Java*.

2.2 Music Similarity

These days music collections grow bigger and bigger in such a way that the conventional list-based user interfaces for browsing music reach their limits. *Jukefox* offers several smart user interfaces based on a *Music Similarity Map* to give the user a new way to interact with his music collection.

Over one million songs are placed in a 32 dimensional euclidean space. Their coordinates are computed using Probabilistic Latent Semantic Analysis (PLSA) on social tags and listening behavior of songs on *last.fm*[5]. This combines the

[1]http://www.ethz.ch/ [29.06.2013]
[2]http://www.android.com/ [25.06.2013]
[3]http://www.idc.com/getdoc.jsp?containerId=prUS24108913 [25.06.2013]
[4]http://java.com/ [25.06.2013]
[5]http://www.lastfm.de/ [30.06.2013]

advantages of objective audio-features and subjective social tags.[6]

The resulting *Music Similarity Map* can be used to find similar songs to a given song. If two songs have a small distance in the 32 dimensional euclidean space to each other, they are with high probability similar.

2.3 Jukefox CLI Player

Although *jukefox* was programmed for the *Android* platform, the core of it is by now completely platform-independent and written in native *Java*. Besides the core of *jukefox* exist PC-specific classes which can be used to run it on a computer.

Jukefox is already available for computers, but just as a command line interface (CLI). Of course it is not intended to be utilized by normal PC users but was rather a proof-of-concept, that the core of *jukefox* is platform-independent and can also be run on a computer.

The *Jukefox CLI Player* has implemented the basic music player commands like PLAY, STOP, NEXT, etc. and the ability to load and save playlists. Even the smart playlists functions of *jukefox* are available. It is needless to say that due the absence of a graphical user interface (GUI) it misses the smart user interfaces of *jukefox*.

2.4 Library Import Manager

Until now *jukefox* imported every music file it found on a smartphone. Apart from directories which the user had blacklisted. There were three different situations where the library import manager scanned all files and searched for changes:

i) on startup, if the automatic import is activated

ii) after the android media scanner found some changes (also only if the automatic import is activated)

iii) if the user started the import manually by clicking on 'Manual Import' in the options menu

All needed information about songs, albums and artists are fetched from the *Jukefox Webserver*. The ids are assigned incrementally.

[6]Social Audio Features for Advanced Music Retrieval Interfaces:
http://www.disco.ethz.ch/publications/mmfat11301-kuhn_221.pdf [30.06.2013]

Therefore the existing library import and the synchronization cannot be activated at the same time. Otherwise conflicts could occur, i.e. they would not agree on the ids of songs, albums and artists. It could be possible that the *library import manager* imports a song which only exists on the smartphone and gives it a certain id which is already used by the server.

Thus we need to disable the normal library import in the situations that are mentioned above. This can be done easily by checking in the settings if the synchronization is activated or not.

Chapter 3

Implementation

This chapter is about the realization of the goals that we have specified in section 1.3. We will introduce the concepts that we have developed and explain how we designed the system. For that we will need some terms which are defined below.

(Jukefox) Server: Refers to the server part of the *desktop jukefox application*

Server Database: The database of the *desktop jukefox application*

Client: An *Android* smartphone running *jukefox*

3.1 System Design

The system is designed based on the *client-server model*[1]. The *desktop jukefox application* acts as server and the *Android jukefox* application acts as client. Clients can request music from the server. The overall structure of the system can be seen in figure 3.1.

Before a client can use the service provided by server it has to register itself at the server. The user has to confirm the registration at the *desktop jukefox application*. Afterwards the server assigns the client an id and an authentication number. These two numbers are used for authenticate the further communication between client and server.

3.1.1 Establishing a Connection between Server and Client

If a client tries to connect to the server where it is registered, it searches at first the local network by broadcasting a message. All running *jukefox* servers in the same network respond then with their hostname and their version. The client can then check if the right server is under the received responses. If the server is not in the local network, the client uses then the stored *jukefox* server address to establish a connection.

[1]http://en.wikipedia.org/wiki/Client%E2%80%93server_model

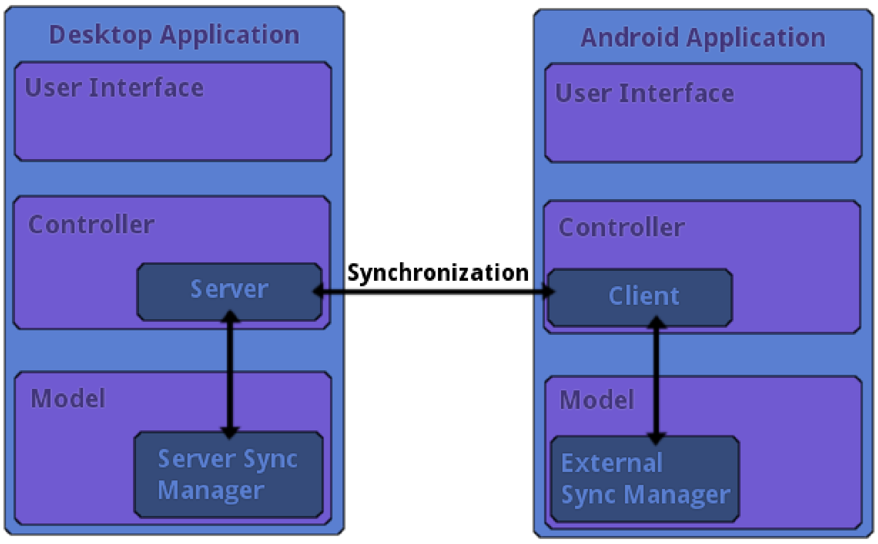


Figure 3.1: Overall structure of the system

If the *jukefox* server tries to connect to a client, it only searches the local network for it. Each client which is registered at this server then responds with its id that it got from the server. When the server receives a response from the sought-after client, the server can retrieve its address from the response.

3.1.2 Communication between Server and Client

The communication protocol consists of several request and respond messages, which are represented in the *JSON*[2] format. For sending files we cannot use *JSON* because it is not designed for serializing binary data. So we just send the binary representation of the file through the socket. Each message is authenticated by the id of the client and its authentication number. We handle a request only when its authentication is valid.

The communication protocol consists of requests and responses of the following message types:

REGISTER:

The request contains the name of the phone and its serial number. If the synchronization is allowed, the response contains its id and authentication number which is used for further communication.

[2]http://www.json.org/

UPDATE SERVER DATABASE:

The request contains current version and timestamp of server database on the client. If the client did not received the database from the server yet, the timestamp and the version are 0. The response contains as result the answer if database is up to date and contains otherwise new version and timestamp of the database which is sent afterwards. Furthermore the response also contains a list of songs, which the server assumes are on the client.

ADD SONG:

Since both client use the same ids for the same songs, the request contains only the id of the song which should be added.

DELETE SONG:

Like ADD SONG

GET FILE:

The request contains the path of requested file.

SYNCHRONIZE:

The request contains two lists. A list of songs which should be added and a list of songs which should be deleted.

If not mentioned differently, a response message contains only a boolean value which indicates if an error occurred or not. And if an error occurred, the response contains furthermore the error message.

A sample message exchange during the registration is illustrated below. The registration request from the client is represented in listing 3.1 and the response from the server in listing 3.2.

Listing 3.1: Registration Request from Client

```
{
        "type":"REGISTER_REQUEST",
        "deviceId":0,
        "authentication":0,
        "arguments":
        {
                "deviceName":"SAMSUNG GT-I9100",
                "deviceSerialNumber":"000903b604e29f"
        }
}
```

Listing 3.2: Registration Response from Server

```
{
        "type":"REGISTER_RESPONSE",
        "error":false,
        "results":
        {
                "deviceId":28,
                "authentication":2141083527
        }
}
```

3.2 Desktop Jukefox Application

The *desktop jukefox application* should be a fully music player which includes the features that make *jukefox* special. Namely the knowledge about music similarity and the extraordinary user interfaces for browsing a music collection.

In addition to a normal music player the *desktop jukefox application* is also a music synchronization server which provides clients access to its whole music collection.

3.2.1 Graphical User Interface

We have designed the GUI so that users of the *Android* application find their way immediately. For that reason the GUI should offer the user the same look & feel as he is used to from the *Android* application while we take advantage of the benefits of a PC. Therefore we have kept the navigation through tabs.

Since the screen of a computer is by far bigger we can display more information at the same time. Like most other music players, we display the playback buttons, information about the currently playing song, the current playlist and an user interface at the same time.

Structure of the GUI

The GUI is divided into several containers as shown in figure 3.2. They all are JPanels[3], which are arranged with a GridBagLayout[4].

All containers beside the content container do not change over time. In the the content panel we will display the different user interfaces. The user can navigate through them by clicking on the tabs on the left hand side.

[3]http://docs.oracle.com/javase/6/docs/api/javax/swing/JPanel.html [26.06.2013]
[4]http://docs.oracle.com/javase/6/docs/api/java/awt/GridBagLayout.html [26.06.2013]

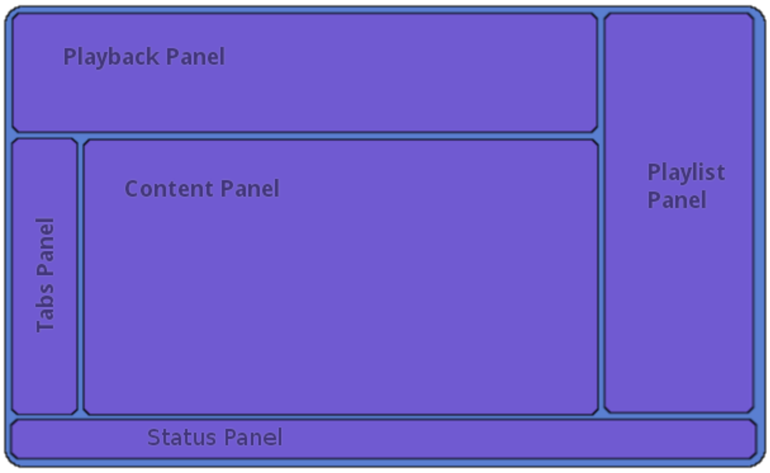


Figure 3.2: Structure of the GUI

We implemented also a common music user interface which uses three lists for artists, albums and songs. They use the typical music hierarchy navigation. This means that the content of song list depends on the selected albums in the album list whose content depends on the selected artists in the artist list.

For the user interface of the synchronization we have decided to create yet another visualization of the user's music collection. The synchronization interface consists of a *JTabbedPane*[5] with a tab for each client who is registered at the *jukefox* server. For each client exists a *Music Tree* which not only shows the whole music collection but also indicates which songs are on the client, respectively are marked for adding.

Music Tree

The component to administrate the songs which are stored on a client is based on Santosh Kumar's *JTree with CheckBoxes*[6] and can be seen in figure 3.3.

The checkboxes on the left side of the songs, albums and artists show if they are on the client, respectively will be after the synchronization. Through these checkboxes the user can change mark them for adding or deleting.

The click on a checkbox does not start the synchronization. This has to be done manually by the user.

[5]http://docs.oracle.com/javase/6/docs/api/javax/swing/JTabbedPane.html [26.06.2013]
[6]http://www.jroller.com/santhosh/entry/jtree_with_checkboxes [18.6.2013]

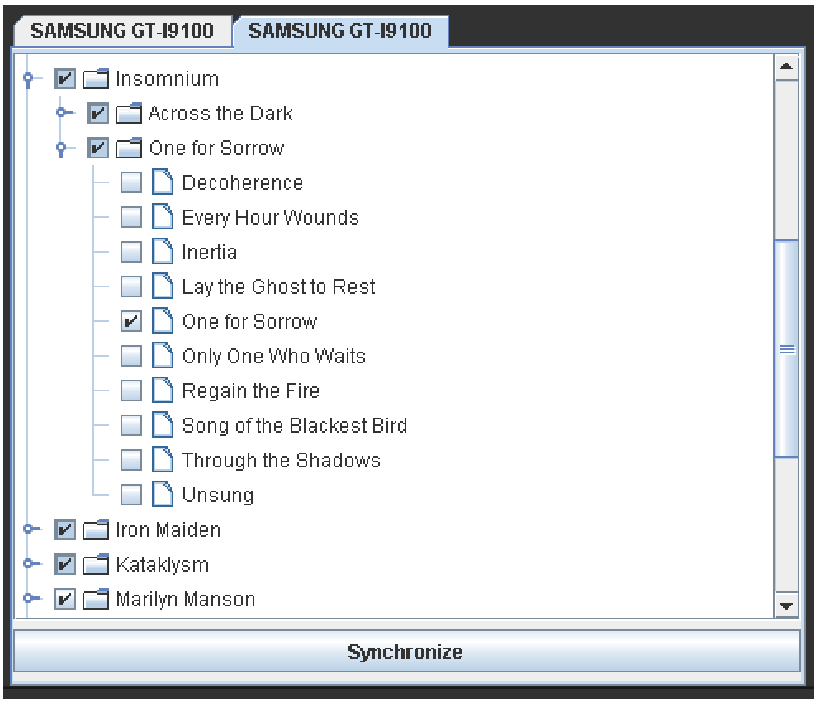


Figure 3.3: Synchronization User Interface with Music Tree

3.2.2 Server Synchronization Manager

The *Server Synchronization Manager* provides access to the state of the server. It has stored the data of the user's whole music collection and holds information about which songs are stored on which clients.

3.3 Android Jukefox Application

The *Android* app of *jukefox* already offers several user interfaces for browsing the music collection on the phone. But they are only designed for displaying songs that are on the phone. It would have been possible to extend them so that they could also indicate if a song is on the phone or not. But we have decided to make a clear separation between browsing only the songs on the phone and browsing all songs.

The user can activate the user interface for browsing the music collection the same as he can start the normal library import. By clicking on 'Manual Import' in the options menu. If the synchronization is activated, this entry would have otherwise been useless.

3.3.1 Synchronization User Interface

The user interface for browsing the music collection of the *jukefox* server is designed as a hierarchical list. Initially it shows all the artists which are currently on the server. If the user clicks on a certain artist, the list gets updated and shows then all albums of this artist. Clicking again on an album shows the songs of that album.

Each item of the list visualizes the synchronization state of itself. To the left of the name is a tristate checkbox which shows if the item should be on the client or not. For a song this is simply yes or no. But for an artist or album this also can be partially. Rightmost is indicated how many songs have to be added and deleted during the synchronization to reach the wished state.

The synchronization has to be initiated by clicking on the synchronization button.

3.3.2 External Synchronization Manager

The *External Synchronization Manager* provides access to both the database on the phone and to the database of the server. It is used during the whole synchronization process.

Server Database

We can retrieve all necessary information about the music which is currently on the *jukefox* server from the server database. Therefore we have decided that all clients have a copy of the server database. This has the following advantages:

+ For browsing the music collection of the *jukefox* server we can get all artists, albums and songs by querying the server database.
+ While synchronizing we only have to send files and no information about songs.
+ For the smart synchronization process we do not have to communicate with the *jukefox* server. We have all needed information on the client.

Nevertheless there are also one disadvantage:

- A principle of *jukefox* is that it works independent of underlaying the database management system. This means that the *jukefox* server and the client do not necessarily use the same database management system.
  At the moment we transfer the server database by simply sending the database file from the *jukefox* server to the client. This works because they use both

SQLite[7]. However this violates the principle. Therefore the transfer of the server database has to be altered by sending INSERT-statements instead of the database file. Like this we can create on the client a copy of the server database which uses the client's database management system.
A plus factor would be, that we can omit unneeded information such as the playlog of the *jukefox* server.

Synchronization Tables

The server database besides the database of the client is not enough. For some functions we would need queries over both databases. Although *SQLite* supports an ATTACH-statement[8] which makes this possible, this is not the general case. Especially it is not part of the *SQL-92 standard*[9].

Most functionalities we could implement by querying one database and use the result to query the second database. But unfortunately this does not hold for some functions.

For example computing the states of an album or artist would be very difficult and slow this way. So we have decided to maintain three additional tables (synchronization tables) which have a corresponding entry for every artist, album and song in the server database.

With this database scheme computing the states of an artist or album is much simpler and can be done by querying only one database.

Updating the Server Database

The server database is updated when before the synchronization user interface is shown. If the server database is outdated, we replace it with the new one. But this is not enough. Since each artist, album and song has an entry in the synchronization table, we have to update them as well. Otherwise it could happen that the user sees a song in the synchronization user interface which is no longer on the server or new songs would not been displayed.

3.4 Synchronization

The synchronization status of each song is described by two states. On the one hand by the current status which indicates whether the song is on a phone or

[7]http://sqlite.org/ [27.06.2013]
[8]SQLite Syntax: ATTACH-statement of SQLite: http://www.sqlite.org/lang_attach.html [15.6.2013]
[9]SQL-92 standard: http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt [18.6.2013]

not. And on the other hand by the target status which indicates if whether the song should be on the phone or not.

If the current status of a song is equal to the target status of it, then we do not have to change anything. Otherwise we either need to add or delete it to achieve the desired status.

3.4.1 Consistency

The consistency between the views of the server and the client is not critical. Therefore we decided to use an optimistic synchronization. With optimistic we mean that even if we do not get a response from the server we assume that the synchronization worked correctly (i.e. that server and client agree on the current states of all songs).

And after updating the server database we check for possible inconsistencies which may have occurred anyway. If we find one, we try to resolve it. Therefore we assume that the client is right. For example the server thinks a song is on the client but it is not, then we send a message to the server that he should delete it.

For finding inconsistencies we need to know the view of the server. But the server database is only updated if it is outdated and a change in the synchronization tables of the server database does not yield to a new timestamp. Therefore we added a list of the songs which the server thinks are on the client to every UPDATE DATABASE response. Then we can use them to find inconsistencies.

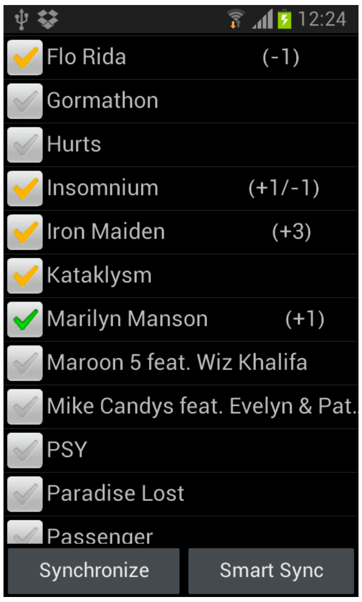
3.4.2 Client-side synchronization

The client can initiate two kinds of synchronization. Either the normal synchronization or the smart synchronization (as described in section 3.5). In both ways the requested changes are determined by looking at the current and target states of songs and select the song ids where the states differ.

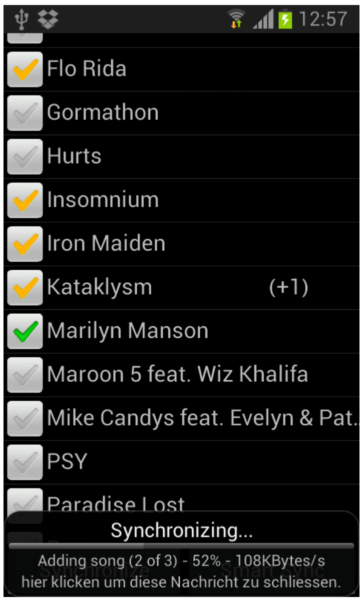
The synchronization is executed in three steps. Adding songs, deleting songs and removing obsolete data.

Adding Songs

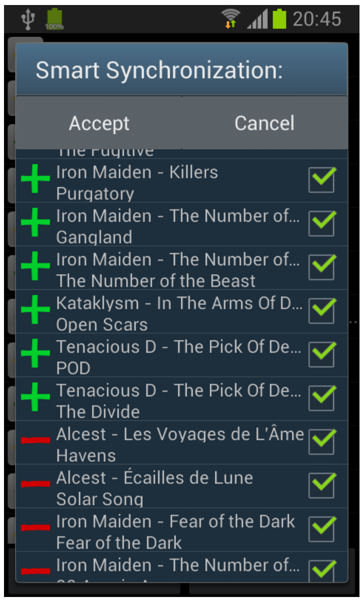
At first the necessary data (about song, album, artist and genre) is fetched from the server database. Then a add song request is sent to the server and if the response has no error, we can get the song file. The field *data* in the server database gives the path of the song on the server. This is sent with a get file request to the server which then sends the file. The same can we do for requesting the covers of the album of the song.


12:24
Flo Rida (-1)
Gormathon
Hurts
Insomnium (+1/-1)
Iron Maiden (+3)
Kataklysm
Marilyn Manson (+1)
Maroon 5 feat. Wiz Khalifa
Mike Candys feat. Evelyn & Pat.
PSY
Paradise Lost
Passenger
Synchronize
Smart Sync

(a) The new synchronization user interface in the android app


12:57
Flo Rida
Gormathon
Hurts
Insomnium
Iron Maiden
Kataklysm (+1)
Marilyn Manson
Maroon 5 feat. Wiz Khalifa
Mike Candys feat. Evelyn & Pat.
PSY
Paradise Lost
Synchronizing...
Adding song (2 of 3) - 52% - 108KBytes/s
hier klicken um diese Nachricht zu schliessen.

(b) Status information about the synchronizing process


20:45
Smart Synchronization:
Accept
Cancel
The Fugitive
+ Iron Maiden - Killers
Purgatory ✓
+ Iron Maiden - The Number of...
Gangland ✓
+ Iron Maiden - The Number of...
The Number of the Beast ✓
+ Kataklysm - In The Arms Of D...
Open Scars ✓
+ Tenacious D - The Pick Of De...
POD ✓
+ Tenacious D - The Pick Of De...
The Divide ✓
− Alcest - Les Voyages de L'Âme
Havens ✓
− Alcest - Écailles de Lune
Solar Song ✓
− Iron Maiden - Fear of the Dark
Fear of the Dark ✓
− Iron Maiden - The Number of... ✓

(c) Suggestions of the smart synchronization

Figure 3.4: Application screenshots

After that we have everything we need and can move the files (previously stored in a temporary directory) to the music collection directory and the cover directory. Then we have to change the gathered data accordingly to the new paths of the files, before we add it to the database of the client.

If all went without problems or errors, we can update the current status of the song, and the current states of the album and the artist of the song.

Deleting Songs

Before deleting the song from the database, we inform the *jukefox* server about it. The *jukefox* server then changes the current status of the song and sends a response to the client. After that the client can remove the song from the database and update the current status of it.

Removing Obsolete Database Entries

If all songs of an album or an artist are deleted, they still appear when the user browse his music collection. To avoid this, we have to remove all obsolete information from the database.

3.4.3 Server-side synchronization

The synchronization initiated from the server differs only slightly from the synchronization initiated of the client. The *jukefox* server sends a synchronization request to the client which contains all changes, i.e. two lists containing the ids of songs to add and to delete.

If the client receives this request, it updates the target states of the songs which should be changed accordingly to the request of the server. Then it starts the normal synchronization.

3.5 Smart Synchronization

Jukefox already provides a playlog, which stores when the user listened which songs, if they were skipped and much more. During the smart synchronization we analyze this data and then suggest songs which could be added or deleted. The user can view the list of suggestions and ignore some of it. If he accepts the suggestions, the target states of the songs which were not ignored are updated accordingly to the suggestions. After that we just start the normal synchronization.

3.5.1 Smart Synchronization Manager

We only consider the playlog entries which are not older than when the user used the last time smart synchronization. But because the smart synchronization gets better if we have more data about the listening behavior of the user, we consider at least the data of the last two weeks.

If we call the functions for getting songs to add or delete, we can specify the maximal number (maxNum) of songs we want to get.

Adding Songs

For determine which songs could be added, it is necessary to first gather the needed data from the playlog of the phone and then use this data for querying the server database. We use three methods which use different approaches to find songs which could be added:

i) similar songs to often listened songs

ii) songs which were recently added to the server

iii) some random songs which are currently not on the phone

Of each method we add $num = weight_{method} * maxNum$ songs to the resulting list of suggestions.

Simalar songs to often played:

First we determine the most often played songs ($songId$) and how often they were played ($numPlayed$). Then we use $\frac{numPlayed_i}{\sum_i numPlayed_i}$ as a weight to determine how many similar songs we want to suggest for $songId_i$.

Recently added songs:

We just the last added songs on the server (sorted with descending timestamp) and check that they are not yet on the phone.

Random songs:

We just get some songs of the server which are not on the phone yet.

Since its possible that two or even all three methods suggest the same song. In order to prevent duplicates, we check if a song already exists in the result before we add it to it. But since this would make it possible that we have less songs from the method, we decided to get $maxNum$ songs of each method and add only as many songs as wanted.

Deleting Songs

For determining which songs should be deleted, it is sufficient to regard only the playlog of the smartphone. We have decided to concentrate on two different ways to find songs to delete:

i) often skipped songs

ii) songs which were not listened to

Just like at finding songs for adding, we add $num = weight_{method} * maxNum$ songs of each method to the resulting list of suggestions. And we use the same procedure to prevent duplicates.

3.5.2 How to determine the Amount of Changes

The maximal size of the user's music collection can be either defined by himself or otherwise we decided to use 75% of available space where the music is stored. The remaining space should get filled up with as many songs as possible.

Because the size of a song is not stored in the database and we did not want to extend it if it is not really necessary, we worked with the assumption that a average song is about 10MB big. The assumption should be good enough to working with.

There are two different cases which are considered when we determine the amount of change:
Free space available: If the maximal music collection size is not yet reached, songs are only added and not deleted.
No free space: If the maximal music collection size is almost reached, we have to delete songs in order to get space for adding new songs. Then we change about 10% of the music collection.

But since the user has to accept the suggestions of the smart synchronization, we set an upper bound of 15 for the number of songs to add and delete. Otherwise the dialog would get cramped and the user had no longer a good overview of the suggestions.

Chapter 4

Future Work

The time for writing this thesis was limited and we had to make several compromises. Apart from some possible improvements there are also several ways how the new synchronization feature of jukefox could possibly be extended in order to make it more usable for the users.

4.1 Possible Improvements

Complete the Desktop Jukefox Application: The current version of the *desktop jukefox application* is only a prototype which just supports some minimalistic music player functionalities. This also depends on the API which is used for music playback. It does not support features, which were not essential for the *jukefox cli player* but are absolutely mandatory for the desktop application, like volume regulation for example.
Other missing functionalities like generating and managing playlists could be easily adopted from the android application without greater adaptions. More jukefox-specific features like the song map or the tag cloud have to be ported from the android application to pure java code.

Automatic Synchronization: A nice feature would be, if the synchronization does not have to be started manually any longer, but starts automatically as soon as the smartphone established a successful connction via WLAN.

Improve the smart synchronization: The smart synchronization could get improved by allowing the user to manipulate the weights for the different methods to suggest songs. Furthermore more methods could be implemented which exploit different synchronization behavior.
In addition it should be possible to improve the smart synchronization by developing agents that autonomous alter the weights.

4.2 Possible Extensions

Use of Cloud Storage: Most users will not have a server at home on which they could run the *desktop jukefox application*. But it is only possible to synchronize when the *desktop jukefox application* runs. Most users will want to shutdown their PC before the go out of the house and like this they could not make use of the possibility to download songs on the way.
For that reason one possible extension would be to allow the user to store his music collection in the cloud. Therefore it would be necessary to move some functions of the jukefox server to the android application and distinguish between whether the smartphone is synchronized with a jukefox server or the music is just stored in the cloud.

Run library import and synchronization together: Another extension would be to enable both the normal library import and the synchronization at the same time. This would require that the library import manager would consult the jukefox server if it finds a new song on the phone and if the song would not yet exist on the server, send the song to it. Like this songs that were on the phone in the beginning would be also considered.

Chapter 5

Conclusion

The synchronization of music is a challenge for the user. Making the decision of choosing a subset of his music collection to store on a phone is complicated and subject to permanent change.

Jukefox already has knowledge about music similarity. So far that knowledge was used to create smart playlists which match the mood of the listener. Now we used it to propose songs which could be added or deleted.

To conclude this thesis we can say that the developed music synchronization is a fair extension for *jukefox* and the smart synchronization feature shows that we can facilitate the synchronization process by evaluating the music listening behavior of the user. The new smart synchronization feature of is the first step towards the automating of the synchronization.

Appendix A

Appendix

A.1 How to set up the Jukefox Server

1. Install the *desktop jukefox application* on a computer.
2. Add the paths of your music library in the settings and start the library import.
3. Redirect the ports 8001 and 8002 to this computer.
4. Ensure that the computer has either a static IP for it or make use of *Dynamic DNS*.

A.2 How to use the new Synchronization Feature

1. At the first start of the application select 'Synchronization' as library method. You can activate the synchronization later in the settings. WARNING: toggling between classical library import and synchronization deletes your database on the phone!
2. Before continuing ensure that either you are in the same network as the server or that you have set the server address in the settings.
3. Click on 'Manual Import' in the options menu and follow the steps for the registration. You have to permit the registration at the server.
4. If you click now again on 'Manual Import' you will see the new synchronization user interface. Click on the checkboxes to the left to add or delete songs or whole albums and artists. By clicking on 'Synchronize' you start the synchronization. If you click on 'Smart Sync' you will see a dialog with a list of suggestions based on your listening behavior. You can exclude some of thesuggestions. When you accept the suggestions, the synchronization will start.