FAKULTÄT FÜR !NFORMATIK

Collaborative music selection with mobile devices

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Magister der Sozial- und Wirtschaftswissenschaften

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Markus Toth
Matrikelnummer 0025690

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuer: Univ. Prof. Dr. Wolfgang Klas
Mitwirkung: Mag. Stefan Leitich

Wien, 23. Juli 2008

[Unterschrift Verfasser/in] [Unterschrift Betreuer/in]

Technische Universität Wien

A-1040 Wien Karlsplatz 13 Tel. +43/(0)1/58801-0 http://www.tuwien.ac.at

Zusammenfassung

Musik ist ein wichtiges Werkzeug um eine bestimmte Atmosphäre für soziale Treffpunkte in öffentlichen Räumen (wie Cafes, Bars oder Tanzlokale) zu erreichen. Die richtige Musik wird normalerweise vom Personal des Lokals oder von einem professionellen 'Musikauswähler', einem Disc-Jockey (DJ), ausgewählt.

Persönliche Medienbibliotheken werden immer mobiler. Tragbare Musikwiedergabegeräte sind mittlerweile alltäglich und für viele Menschen ist es längst normal die Musik des eigenen Musikgeschmacks immer mit sich zu führen. Üblicherweise bestimmt eine einzelne Person welche Musik in einem Lokal gespielt wird und dadurch sind alle aus der Zuhörerschaft nur passive Konsumenten der Musik. Die Möglichkeit eines jeden Zuhörers mit der eigens mitgebrachten Musikbibliothek Einfluss auf den Musikauswahlprozess zu haben würde in einer komplett neuen Art von interaktiver Erfahrung resultieren.

In dieser Arbeit stelle ich ein neues Konzept und die prototypischen Implementierung für die Musikauswahl in öffentlichen Räumen vor. Das Konzept von PublicDJ basiert auf einem rundenbasierenden Spiel für mehrere Spieler bei dem jeder Teilnehmer Lieder zu einem Server schicken kann. Der Server analysiert die von den Spielern ausgewählte Musik und wählt das am besten passendste Lied (basierend auf Kriterien die vor dem Start verkündet wurden) für die nächste Runde aus. Es kann entweder auf die im voraus händisch hinzugefügten Metadaten (wie Künstler, Genre, Veröffentlichungsjahr) oder Metadaten, die mit Hilfe von Techniken der Audio-Merkmals-Extraktion aus den reinen Audiodaten neu erzeugt werden, für die Anwendung der Auswahlkriterien zurückgegriffen werden. Dadurch ist es möglich Aufgaben wie 'Wählt Lieder aus dem gleichen Genre!' oder 'Wählt Lieder von demselben Künstler' den Teilnehmern zu stellen. Die Aufgaben sind dabei ein indirektes Werkzeug um die Musik zu steuern.

Dieser Prototyp für die gemeinschaftliche Musikauswahl in öffentlichen Räumen erhöht die Interaktion und die Beteiligung des Zuhörers durch die Möglichkeit aktiv an einem bisher komplett passiv erfahrenen Prozess teilzunehmen.

Abstract

Music is an important tool to achieve a certain atmosphere in public spaces (e.g., Cafes, Pubs, Clubs) for social gatherings. The selection of proper music tracks is usually either done by a staff member of such a location or a professional music selector (a disc-jockey (DJ)).

The mobility of personal media libraries is increasing. A portable music player has become an everyday item and people are used to have their favorite music with them at any time. In an usual setting a single person is determining, what a whole auditory is listening to, degrading the listeners to passive consumers. Creating the possibility to allow every person in the auditory getting involved in the music selection process, by using their portable music library, would result in a totally new kind of interactive listening experience in public spaces.

In this thesis i present a new concept for music selection in public spaces as multiplayer game - PublicDJ, and its prototypical implementation. The concept is based upon a round based multiplayer game, where each player can submit music tracks to a server. The server analyses submitted tracks and selects the best matching track, based on a former announced criteria for playback. Selection criterias can range from high-level manually annotated audio metadata to low level-audio metadata extracted from audio using music information retrieval techniques. This allows the implementation of tasks like for example 'Submit songs of the same genre!' or 'Submit songs of the same artist!', which the users have to fulfill and can be used as a steering instrument for the music played.

This prototype for collaborative music selection in public spaces as multiplayer game, increases interaction and involvement of listeners by providing the possibility of active participation in a previously completely passive experienced procedure.

Acknowledgments

I thank my parents for giving me the opportunity to study at the university.
I want to thank my brother Alexander for his persistent motivation
and my brother Thomas and Susanne for proofreading this work.
I thank my friends, especially Christian and Wolfgang
who gave their best to sidetrack me.

I wish to thank Stefan Leitich for allowing me to work
on Audio Playlist Generation and his continuous support.
Thanks go also to Thomas Lidy for providing me with the
audio feature extraction source code and the support he gave me.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

The MP3 audio file format paved the way for building up big personal music collections. These collections got mobile with the success of MP3 players. Everyone can listen to his or her preferred music on the go. These facts definitely have changed how we consume music. But what if we go to a club? There you can not affect selection of music unless the DJ in charge listens to your opinion (which seldom is the case).

PublicDJ is a prototypical implementation of a distributed round based multiplayer game, where the length of each round is given by the length of the played music track. Each participant can submit one music track each round and all music tracks that arrive in time get analyzed by the server. Based on an initially announced selection criteria which may range from high-level manually annotated audio metadata to low-level audio metadata extracted using music information retrieval techniques, the server chooses the best matching track and queues it in the playlist. Several mechanisms have been developed to cope with the limitations of time and resources and are described in greater detail in Chapter II.

1.2 Outline

This thesis gives an overview about existing work in the field of playlist generation and presents PublicDJ. by describing all the problems that were encountered and the solutions that were used to solve them. Various techniques from fields ranging from audio feature extraction over playlist generation to software engineering and user interface design were applied to overcome these problems. The main goal was to develop a usable proof of concept that solves this challenge. The family of applications called PublicDJ is a new approach for generating playlists the collaborative way.

This diploma thesis is split into two major parts, the theoretical part I and the practical part II. The development of PublicDJ was the main goal of the practical part of this thesis. The theoretical part deals with all research topics that PublicDJ touched while it was designed, implemented and tested. The thesis concludes with a few words about the future of PublicDJ.

1.2.1 Theoretical Part

Chapter 2 deals with the ID3-Tag which is used in the popular MP3 format. The basic layout of the different ID3-versions is explained and the evolution of this Tag-format is described. PublicDJ was designed to analyze the metadata of MP3 files, so it was mandatory to get a deep understanding of the stored metadata itself. It was necessary to understand the process for extracting it properly before it was possible to use the metadata for any playlist generation tasks.

In ***Chapter 3*** the research field of playlist generation is described from a top-down-perspective. It describes playlist generation categories called *Content-based Playlist Generation*, *Metadata-based Playlist Generation* and the self invented *Metadata and Content independent Playlist Generation* and classifies the various approaches by the method of generating playlists.

Chapter 4 is an excursion to the techniques of audio-feature-retrieval which were used in the content-based part of PublicDJ. It is described how *Rhythm patterns*, *Statistical spectrum descriptors* and *Rhythm histograms* are calculated and examples are shown which visualize the signatures of two music files of different genres. Based on the created signatures it is possible to classify music into different genres automatically.

Chapter 5 gives a description of the applied techniques for normalization and distance measurement.

1.2.2 Practical Part

Chapter 6 introduces to the practical part and covers the requirements that were specified before PublicDJ was designed at all. The general design of PublicDJ, the round design and the use case diagrams for visualization are presented in subchapters.

Chapter 7 presents the implementation details about the solutions for various encountered issues as used in PublicDJ.

Chapter 8 describes how to setup the program environment on a Personal Digital Assistant (PublicDJ Client and Admin), in this specific case a Dell Axim x51v, and on the normal workstation (PublicDJ Master Server and Analysis Server).

Chapter 9 documents the processes/threads of the PublicDJ applications and shows UML sequence diagrams of the interaction between the processes/threads in important usecases.

1.2.3 Conclusion

Chapter 10 outlines the current state and the possible future of PublicDJ.

Part I

THEORETICAL PART

Chapter 2

Metadata (ID3)

Metadata is data about data. Specifically, the term refers to data used to identify, describe, or locate information resources, whether these resources are physical or electronic. While structured metadata processed by computers is relatively new, the basic concept of metadata has been used for many years in helping manage and use large collections of information. Library card catalogs are a familiar example of such metadata. ([5])

An item of computer-processed metadata may describe a date, a content item or a collection of data including multiple content items. It always depends on what kind of data the metadata describes, it also sometimes happens that metadata describes other metadata and in that case the described data is just seen as data and not as metadata anymore.

Audio metadata which is stored together in the same file as the actual audio-data makes it possible to display the name/artist/creation date of the currently played track on the playing device (not only the filename). It allows to filter out music-tracks from a big music collection by using defined constraints (like genre, year, artist,..). Missing audio metadata is one of the main problems anyone writing applications for automatic playlist generation has to face.

On the one hand there are several ways of storing audio metadata, on the other hand there are several different formats. The most popular format so far is ID3 ([31],[30]) together with the MPEG-1 Audio Layer 3 (MP3) format. ID3 was developed as an unofficial add-on for MP3 out of necessity of storing supplementary information together with the audio data. MP3s are very small compared to WAV-files and therefore they are far better usable for transfer over the Internet. That was another reason why MP3/ID3 got a boost of popularity as the format was used for sharing audio files all over the world via various file sharing tools. Later several other formats emerged (like OGG-Vorbis [45] which is another popular format mostly adopted by the open source world) but MP3/ID3 still is the most popular file format for audio files.

As the set of applications coded for this thesis had to deal mainly with the MP3 Format and ID3 Metadata, ID3-Metatags are described more detailed in the following section.

2.1 ID3

The audio formats MPEG layer I, layer II and layer III (MP3) do not have a native way of storing information about the contents except some neglectable simple yes/no parameters like 'private', 'copyrighted' and 'original home' (meaning this is the original file and not a copy). A solution was introduced in 1996 by adding a small amount of extra data at the end of the file which made it possible to get the MP3 file carry information about the audio data and not just the audio data itself [31].

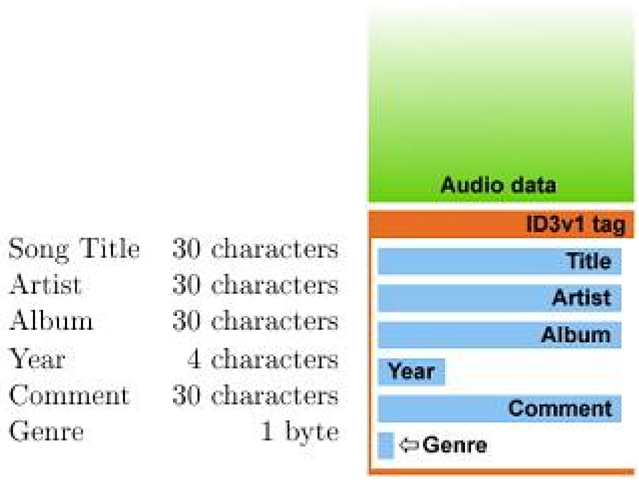
As there was far less chance to disturb decoders, the tag (as the data was called) was placed at the end of the file. To make it easier to detect a fixed size of 128 bytes was chosen and it was defined that the word 'TAG' starts an ID3 TAG, which meant that there are 125 bytes left for the actual data. So the easiest way to find a ID3v1/1.1 tag is to look for the word 'TAG' 128 bytes from the end of a file.

2.1.1 ID3v1

The ID3v1 tag has the following layout depicted in Figure 2.1.

Not every artist has a 30 character name so the bytes left after the artist name should be filled with the binary value 0. The genre field is only 1 byte in size as it was not designed to enter the full genre - it should only store a value that corresponds to a value in a predefined list. The initial list had 80 entries ranging from 0 to 79. [1]

[1]The programmers of Winamp, a very popular musicplayer on the Windows operating system, added 67



Song Title	30 characters
Artist	30 characters
Album	30 characters
Year	4 characters
Comment	30 characters
Genre	1 byte

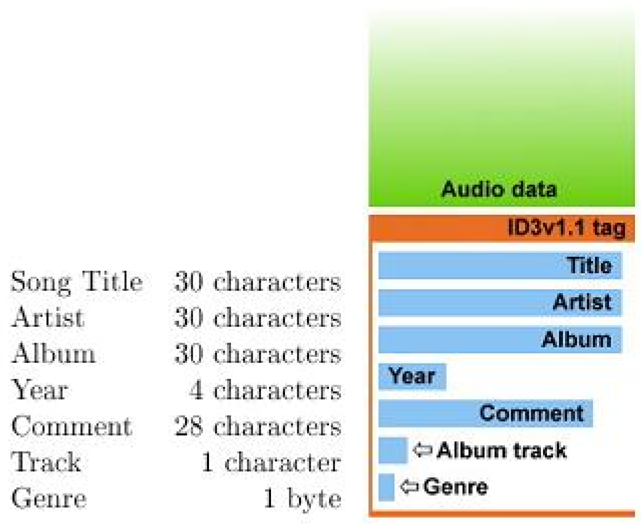
Figure 2.1: Metatags: ID3 - version 1, [9]

ID3v1.1

The initial version of ID3 was easy to implement for programmers but it was soon discovered that it was nearly impossible to enhance the tags to carry even more data because of their design. Backwards compatibility with existing software was important and there were no fields reserved for future use, so that ID3v1.1 only brought the minor improvement of an additional field for the tracknumber. As all unused fields must be filled with zeroed bytes (0x00) Michael Mutschler[2] made the assumption that all ID3v1 readers stop reading any field when a zeroed byte is read. By shortening the comments field by 2 bytes and zeroing the first byte there was one left for an additional field which was used to store the track number from the CD without breaking compatibility.

additional genres in 2 steps(45 genres were introduced with ID3v2.3 [27] and additional 22 later

[2]Author of MP3ext [25]



Song Title	30 characters
Artist	30 characters
Album	30 characters
Year	4 characters
Comment	28 characters
Track	1 character
Genre	1 byte

Figure 2.2: Metatags: ID3 - version 1.1 [10]

2.1.2 ID3v2

ID3v2 was created in 1998 not being backwards compatible like ID3v1.1. It was designed from scratch and should fix all the inflexibilities of ID3v1 and ID3v1.1. The ID3-Tag moved from the end of the file to the beginning to allow easier Internet streaming (for example for webradios). While the tags had a fixed size in ID3v1/ID3v1.1 they now had a variable size which reduced the size of the new tags as long as the fields were not all completely filled[3]. The tags consisted of so-called frames, each of them containing a certain kind of metadata. Each frame can have a length of up to 16MB and the total tag size can be up to 256MB. As shown in 2.3, ID3v2 also allows to store lyrics, pictures, general information and much more.

A big problem for ID3-Tags was internationalization. ID3v1 Tags only allowed the ISO/IEC 8859-1 charset ([12]) which was a problem for all users in foreign countries where the Latin alphabet is not used (which this charset is for). With ID3v2 the support for UTF-16([42]) was added to solve that issue.

[3]In case the fields were all full, the ID3v2 Tag was 56 bytes bigger

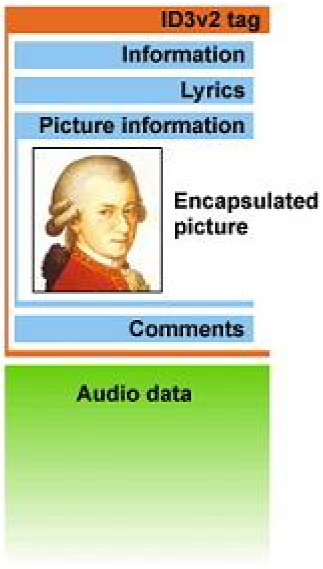


Figure 2.3: Metatags: ID3 - version 2 , [11]

Three different versions of ID3v2 have been developed so far.

- ID3v2.2 [28] was released in March 1998. Like ID3v1/ID3v1.1 it used 3 three character frame identifiers.
- ID3v2.3 [27] was made public in February 1999 and enhanced the frame identifier to four characters. Several frames were added and this new version introduced the feature that multiple values could be added into a single frame by separating them with a '/'-character.
- ID3v2.4 [29]was released in November 2000 and so far is the latest version. Support for UTF-8 [42] was introduced which extended the character set vastly. It also used a null byte to separate multiple values, so it was possible to use the character '/' in the textual data again.

So far ID3v2.3 is the most popular and therefore most widely-used version of ID3-Tags - not because ID3v2.3 is better than ID3v2.4, but because most applications for tagging MP3s where written before ID3v2.4. Most of the programs were able to treat ID3v2.3 correctly but could not handle ID3v2.4 properly. This is why most MP3s still use either ID3v2.2 or ID3v2.3 tags (only early encoded MP3s still use ID3v1 or ID3v1.1 tags).

ID3v2 Chapters

In December 2005 the so called ’ID3v2 Chapter Frame Addendum’([26]) was made public as an enhancement for ID3v2.3 and ID3v2.4 tags. It describes how it is possible to jump to different positions within the audio file and how synchronized slide shows of images and titles can be provided during playback. There is nearly no support in software for this informal standard so far.

Chapter 3

Playlist generation

Playlist Generation is a field of research where some progress was noticeable in the past years. Several papers emerged which had completely different approaches to the problem of creating a playlist automatically.
This section gives an overview of the different main approaches to playlist generation. When describing them in the relevant subsections some examples will be given and their concept briefly described.

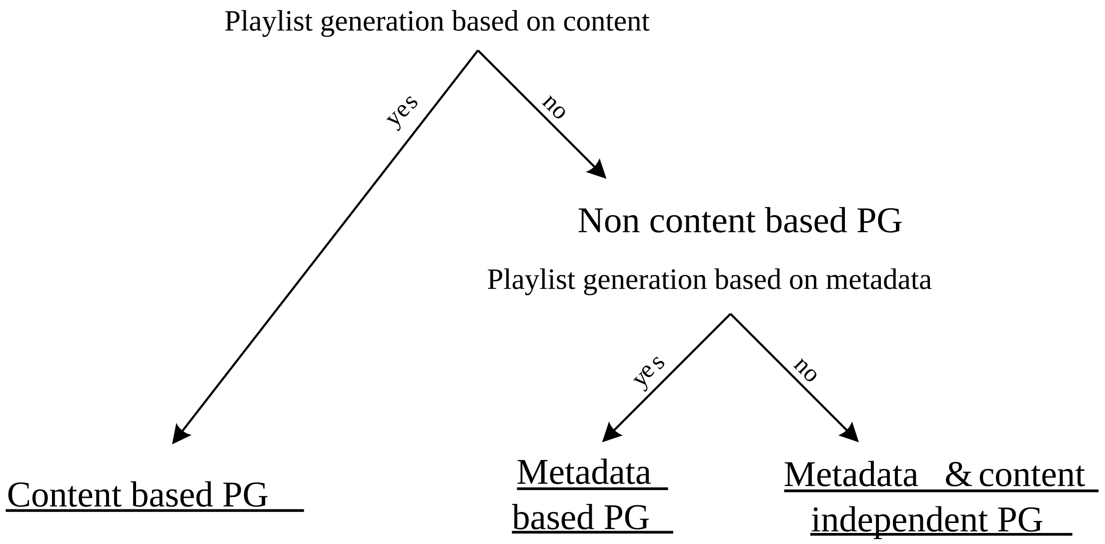

Playlist generation based on content
yes
no
Non content based PG
Playlist generation based on metadata
yes
no
Content based PG
Metadata based PG
Metadata & content independent PG

Figure 3.1: Playlist Generation: Different basic types

There are several different basic characteristics of playlist generators. The main distinguishing feature for playlist generation is the information method that is used for obtaining the information on which the playlists are based on. Systems analyze the contents of the music tracks, their metadata or use other information sources (3.1).

So we have three main categories for playlist generators:

- Content based playlist generation
- Metadata based playlist generation
- Content and metadata independent playlist generation

It is possible to combine these categories to improve the resulting playlists (see 3.4).

There are two additional attributes that, if applicable, can be used for distinguishing between systems (see Figure 3.2).

- Seed song vs no seed song
- User feedback vs no user feedback

Seed songs are reference songs that are often used as an initial input by the playlist generator. They can be used to overcome problems related to missing advance information about the user's listening habits as the user selects one or more songs that reflect his/her current mood.

User feedback is a way to improve playlist generators. By considering user feedback in the playlist generation process it is possible to customize the resulting playlist to fit to the user's musical preferences. Through user feedback the playlist generator can either be improved and trained or immediately affected.

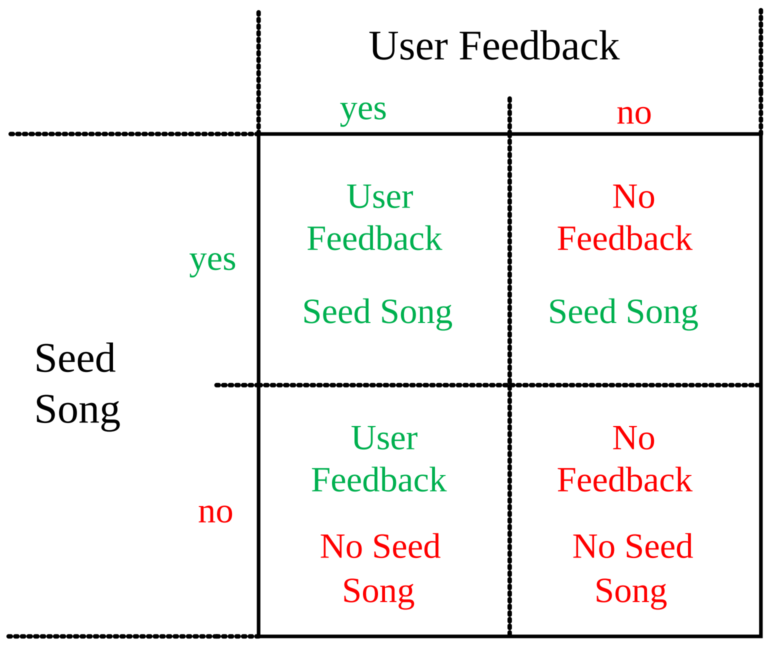


Figure 3.2: Playlist Generation: Additional variants

3.1 Content-based playlist generation

Content-based playlist generators do not use any metadata but create their own information by analyzing the raw data of the music tracks themselves. They obviously are much more CPU-intensive than non-content-based playlist generators as they have to process much more data. That is the reason why there are no gadgets like MP3-players or mobile phones with built-in content-based playlist generators available.

On the one hand higher CPU-consumption is a disadvantage (maybe only a temporary one as computing power increases steadily) on the other hand it is an advantage to be able to classify a song by its raw music data. Content-based playlist generators create their own set of metadata for the analyzed tracks and are therefore independent from the potentially wrong metadata that is stored within the music files.

The main problem to solve in content-based playlist generation is actually a problem closely related to audio-analysis. The raw music data itself does not tell directly how different the music track is compared to another in terms of genre, rhythm etc. Useful information has to be extracted and to be interpreted in order to compute some sort of representation of the audiofile. Depending on how this representation is used later it might be a simple value up to a complex vector containing several computed low and high level audio descriptors. The main questions here are how to concentrate the raw music data, how to extract information and how to interpret the gained information for generating playlists. When creating a content-based playlist generator one first has to define which information which the generator is based on. In a second step one has to find methods and heuristics for extracting the information from the music file.

The first time used in [20], a 'signature' is frequently computed for every music track involved in the playlist generation process. A signature is, compared to the raw music data, a representation of the analyzed music track on a higher level. How a signature is created depends totally on how it is used afterwards and its quality is highly dependent on the steps involved in creation.

Having a signature as a representation of every music track allows comparing the signatures. This enables an algorithm to compare how similar two music tracks are by applying a metric on the two respective signatures. The result is a distance, depending on the implementation or method ranging from a vector with multiple values to a simple value, representing the difference between the two music tracks. When several songs are compared to a reference song then the song with the smallest distance is, depending on the quality of the used process of audio feature retrieval, the most similar one regarding the compared feature or combination of features. Based on this information it depends on the implementation how playlists are actually arranged.

In [21] Beth Logan tried to enhance her previous work by adding graph based playlist generation and automated relevance feedback. Although the ideas of both extensions were promising she noticed that there had to be some problem with the distance measure as the playlists did not improve, they got even worse than without.

In [43] for example the Gaussian Mixture Model (GMM) is used as a signature. It uses a seed song as reference and transforms the problem of playlist generation into a traveling salesman problem. The songs are connected via edges, the length of the edges represent the difference between two songs. By solving the problem iteratively the algorithm will return a playlist containing n songs in a special order which fit to the seed song. No user feedback is employed here.

3.2 Metadata-based playlist generation

Metadata-based playlist generators work with metadata that is stored in the music file. There are several advantages of using metadata for playlist generation:

1. Accessing metadata causes low CPU load

2. Metadata might be very accurate regarding genre, author and year

3. Most audiofiles shared via filesharing tools are music files with metatags (mp3, ogg)

Metadata usually can be accessed very quickly without putting much load on the CPU. This makes it an option for implementing automatic playlist generators directly as applications that run on various mobile devices as these devices will be able to provide the required processing power.

Many digital audio players and several applications on mobile phones are already able to read the ID3-Tags (and sometimes even the Ogg Vorbis Tags) enabling them to access the metadata of the files and using it for filtering following some user defined constraints. This means that it is easier to find the music track the user is searching for.

The drawbacks of Metadata-based playlist generation are:

- Metadata can be missing

- Metadata can be wrong

It can never be assumed that metadata is available for every music track and therefore precautions have to be made to avoid that the playlist generator gets into a inconsistent state because of missing metadata. Either it has to be made sure that metadata is available for all possible tracks (several playlist generators use an external database for storing the metadata) or there should be a special treatment for all files that lack metadata, so that the playlist generation application does not lock up. If metadata is stored in the header of the music-file itself (like ID3 in MP3s) it is good if it is accessible at all. This can not be assumed as support for new technologies and new formats need to be added either natively or via external libraries to program languages before a playlist generator can be written in that language at all.

Another problem is when the metatags contain wrong data. Especially files acquired via filesharing tools over the Internet are likely to have missing, wrong or partly filled metatag. This problem can not be solved and sometimes may cause some confusion when using metadata based playlist generators. Files with wrong metadata mess up the resulting playlist as completely different songs than the chosen ones are added.

In [2] an approach was presented that should allow generating more efficient playlists with 'arbitrarily complex constraints', even with a vast number of titles. Basically the idea was to see the problem of creating a good playlist as an optimization problem. To solve it, cost functions are used as a representation for the problem of finding equivalent songs and adaptive search is applied to find the songs with the lowest total costs. Musical metadata, such as genre, tempo, duration, artist, year are used as constraints. The costs of the constraints represent how bad the constraint is satisfied, for a given assignment of variables. At first the constraints were defined, then an initial random playlist is created, then the initial costs for the constraints are calculated and the problem is solved iteratively like a cost optimization problem.

[38] presents AutoDJ - it is a playlist generation system that creates playlists based on one or more seed songs automatically. It uses metadata from the available music song pool as inputs it applies Gaussian Process Regression to create a function representing the user's preferences. Based on this function playlists are generated. When reviewing the playlists users can remove files from it and this has direct impact on this function and on future generated playlists. So AutoDJ uses seed songs together with user feedback.

The idea of PATS ([37]) is to create playlists by choosing a seed song from a pool of songs which have been analyzed in advance. Then all songs are grouped with dynamic clustering method for grouping songs based on their attribute similarity. The authors of PATS defined that the context-of-use is important as while the same track might fit perfectly to a party it might be totally inadequate when having a candlelight dinner. PATS selectively weights attribute-values, as not all attribute-values are equally important in every context-of-use. An inductive learning algorithm reveals the most important attribute-values for a context-of-use from preference feedback that the user gives over time. The link between the context-of-use and a playlist is established by choosing the seed song as it will represent the current emotional state of the user and therefore also the context-of-use. Implicit user feedback therefore is assumed to be given by the chosen seed song.

An approach where more than one person is involved in generating playlists is the software called audioscrobbler[1], used by the Internet radio station LastFM[2]. A profile is created for every user and it changes over time depending on the user's listening habits. Last.FM now additionally allows users to submit ratings for all songs[3] and therefore enables an additional way for the user to impact this profile via user feedback. A big database stores all the input and collaborative filtering is used to calculate relationships and recommendations based on the music that is listened to and rated by the users. These relationships are of a high quality as as an algorithm calculates these relations/recommendations individually for each user based on profiles and ratings from other users which share the same musical taste.

3.3 Metadata and content-independent playlist generation

Metadata and content independent playlist generators are usually creating playlists on the basis of user feedback. This feedback can range from a real scale-based rating over manually skipping of disliked music tracks to motion sensors which are tracking the percentage of the audience that is dancing.

The advantages of these approaches is that it is not necessary to extract any form of data from the music tracks (like in most metadata-based approaches) and that no deep analysis of the audio content is needed (like in all content-based approaches). But a new problem arises with this advantages: there is no advance information. Content-based and metadata-based approaches extract information from the analyzed music tracks in order to find common attributes by which the music tracks can be classified. Based on this information the playlists are generated and songs from within the same music class are selected for a playlist. Unluckily this causes bad initial playlists for nearly all metadata and content independent playlist generators. Information for basing the playlists on is collected over time, depending on the user's activities. When enough information is

[1] http://www.audioscrobbler.net/

[2] http://last.fm

[3] Last.FM uses the three buttons 'Love', 'Skip' and 'Ban' as inputs

available the generated playlists will be better in terms of how accurate the user's taste is met. When little information is present it is likely that the user's taste is not met at all. Therefore a training phase might be a way of overcoming this problem.

In [35] the author tried to generate dynamic playlists based on the skipping behavior of the user. The user's only possible method of interacting with the player is a skip-button; from his or her skipping behaviour new information about the user's music preferences is gained. All available songs are classified in advance, then music tracks similar to those that were skipped are removed from the playlist and songs similar to accepted ones are added. Because of its design the player does not create suitable playlists from the beginning as it first has to collect information about the user's listening habits.

A very similar approach was presented in [1], but instead of tracking the skipped songs it takes the complete playlist history into account and creates a 'listener model'(a profile of listening habits), which is then used for creating playlists that meet the user's demand. This can be helpful if there is no time for creating a playlist manually.

[8] presents a method to create ratings automatically for songs/artists in realtime by analyzing the request history of a webradio station. It assumes that it is possible to request a specific song or a specific artist on the webradio's webpage. The more often a song or an artist is requested the higher is the songs/artists rating. The higher the rating is the more likely it is that the specific song or a song of the specific artist is played on the webradio.

[39] uses audio streams that have been generated by professional DJs, which the author calls Expertly Authored Streams (EAS), like webradios as a source for relations between songs. The order of the songs from EAS are used as a source for an undirected graph where the songs are the nodes and the arcs connecting the nodes are the relationship between those two songs. The weight of the arc is defined by how often the two songs are played after each other and the author assumes that it represents similarity between those two songs. After this has been done for enough songs (the author uses several tens of thousands songs) a playlist can be constructed after choosing a seed song. Depending on the intended length of the playlist songs directly connected to the seed song are added to playlist with a likelihood proportional to the weight of the arc. The same procedure is repeated for the selected songs until the playlist is long enough.

3.4 Combinations

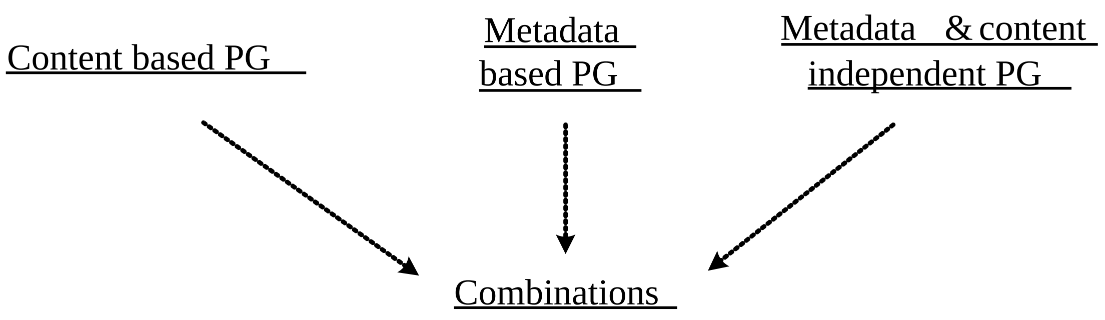


Figure 3.3: Playlist Generation: Combinations

There are several combinations of the different basic concepts which emerged because it was discovered that better playlists can be created easier when more than one basic concept is involved. Some basic concepts have clear disadvantages in areas where other concepts are very good. Content-Based and metadata-based playlist generators have the advantage of being able to obtain advance information from music tracks, content and metadata independent approaches basically use user feedback to improve the playlists over time with the goal of matching the user's taste better and better. So some approaches tried to create even better playlists by joining complementary concepts.

[15] is an extension of the former mentioned [43] as it combines the audio signal-based similarity with web-based musical artist similarity to accelerate the task of the automatic playlist generation. For every artist a Google query is executed and the 50 top-ranked web pages are retrieved, HTML tags and common terms are removed (so that only about 10.000 terms remain) and then all similar terms are counted. The result is one vector per artist that contains the term weights. After normalizing the term weights (so the vectors have the length 1) they are used for comparing the artists with each other. The similarity of artists is seen as an indicator for similarity of their songs. Only songs from similar artists are compared using the audio signal-based approach where playlist generation is seen as a traveling salesman problem. Following the evaluation of the paper this approach leads to improved quality and a significantly decreased amount of necessary calculations compared to the only audio signal-based method.

[40] shows an approach how to create a playlist interactively by clicking on an artist

map which is a representation of an entire music collection. The artist map itself was published in [41] and it is created based on metadata (artist, album, song name and publication year) which gets fetched from web-services and features (tempo and texture/spectral information) that are computed directly from the music itself. There are four different attributes (mood, genre, year and tempo) where the user can choose two to create a 2-dimensional artist map. The attributes label important positions on the map to provide context (tempo: very slow, slow, medium, fast, very fast; year: <1960,60-70,..90-00,2000+ ; genre: Rock, Popular, Dance, Alternative...) and the artists are represented as small dots. When zooming in the names of the artists get visible as well. The users can set playlist points on this artist map at any zoom-level and the application creates playlists by adding songs from the artists that are near to the playlist points. This way users can control on which way the playlist drifts from the first to the last song.

In [34] an approach was presented that combined both content-based playlist generation (using the music similarity measure described in [33] with information from fluctuation patterns [32]) with metadata and content independent playlist generation [35]. A java application was implemented to allow easier evaluation of Music Information Retrieval (MIR) technologies (especially content-based music similarity measures and playlist generation heuristics) in everyday music consumption.

Part II(Chapter 6 - 10) presents PublicDJ, a tool that implements both metadata based and content based playlist generation techniques. It also has the possibility of extensions which enables anyone to add support for other music analysis methods. Regarding content based playlist generation a very similar metrics was used with information of Rhythm Patterns (also called Fluctuation Patterns [32]), Statistical Spectrum Descriptors and Rhythm Histograms ([17],[18]) . PublicDJ is a tool for music selection in public spaces. It is a round based multiplayer game. It starts playing a track that was predefined by the Admin-application and then accepts music tracks transfered via WLAN from Personal Digital Assistants, analyzes the tracks and just before the currently track ends all submissions are compared. The music track with the lowest difference to the currently played track, regardless if metadata-based or content-based analysis was used, gets played next. Implicit user feedback is given through the collaborative preselection of the music tracks[4] each round because the pool of music tracks consists only of tracks that were submitted in that round.

[4]The music tracks were brought along by the participants themselves

Chapter 4

Excursion: Audio Feature Extraction

The family of applications from PublicDJ has several options how playlists are generated from submitted music, including both a metadata-based and a content-based approach. This chapter is dedicated to the content-based techniques that are used and where the Java sourcecode was provided by Dipl.-Ing. Thomas Lidy[1]. He also provided the implementation of Rhythm Patterns, Statistical Spectrum Descriptors, Rhythm Histograms and classes for distance measurement, his work can be seen as the underlying core regarding audio feature extraction. PublicDJ (or to be more exact the Analysis Server application) is a wrapper around this core.

To calculate Rhythm Patterns, Statistical Spectrum Descriptors or Rhythm Histograms some common tasks (Pre-Processing, Segmentation and S1 - S6 in Figure 4.1) have to be done.

[1] Dipl.-Ing. Thomas Lidy, Vienna University of Technology, Institute of Software Technology and Interactive Systems, Information and Software Engineering Group

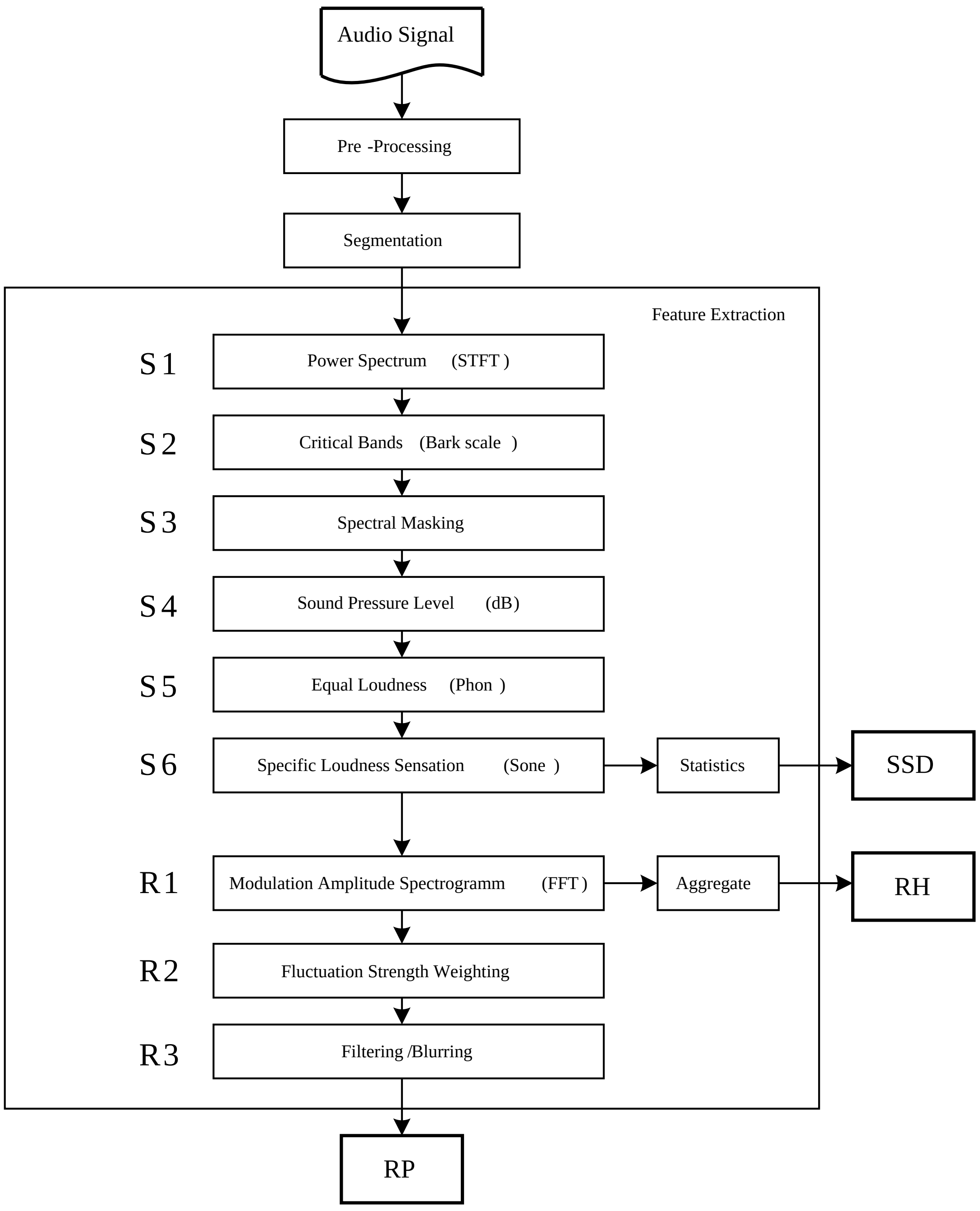


Figure 4.1: Feature extraction process for Statistical Spectrum Descriptors(SSD), Rhythm Histograms (RH) and Rhythm Patterns (RP)

Pre-Processing

The audiofile has to be a 44100 Hz / 22050 Hz / 11025 Hz WAV-File or an MP3 file with the same frequency. If it is something else it needs to be converted because the segment sizes of the following step are only defined for these three frequencies.

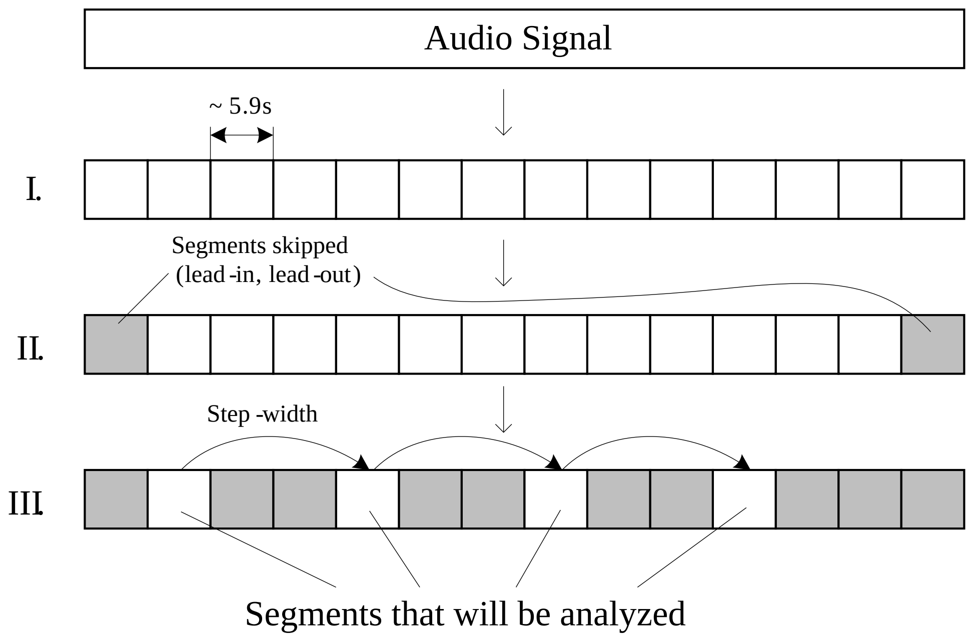


Figure 4.2: Segmentation of the audiofile and process for choosing which segments get analyzed later and which not.
I. Segmentation of the audiofile. The segment size depends on the sampling frequency. It is 2^{16} for 11 kHz, 2^{17} for 22 kHz, and 2^{18} samples with 44 kHz, which means it is always about 5.9 seconds
II. One or more segments are skipped at start and end (in the figure only 1 segment at start and end)
III. Segments are recurrently skipped with a preconfigured skip rate (in the figure this skip rate is 2)

Segmentation

At first the whole file is partitioned into short segments of about 6 seconds. Extracting the audio features from all segments would increase time and mandatory processing power but the net benefit would be relatively low as the analysis of a fraction of segments already leads to very good results. Especially segments at the start and at the end often contain silence which would worsen the results. Figure 4.2 illustrates the segmentation process.

S1 - Power Spectrum (STFT)

By using the short time Fast Fourier Transform (STFT) with a window size of 23 ms and an applied Hann-Window with 50 percent overlap a spectrogram is calculated for every segment. Because of the Hann-Window with 50 percent overlap this results in 511 small segments. The amount of samples of the 23 ms segments depend on the sampling frequency - 256 samples at 11 kHz, 512 samples at 22 kHz, 1024 samples at 44 kHz.

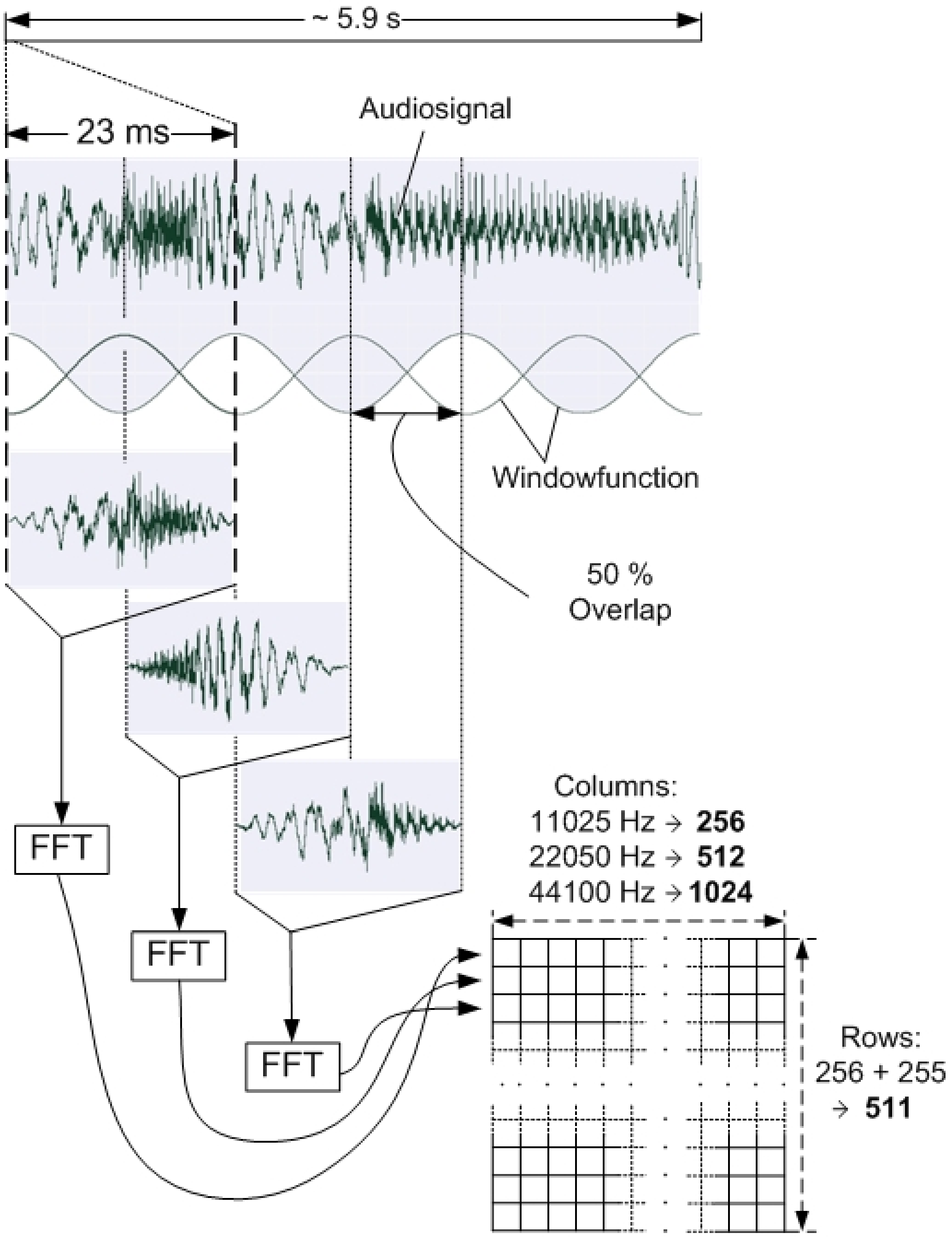
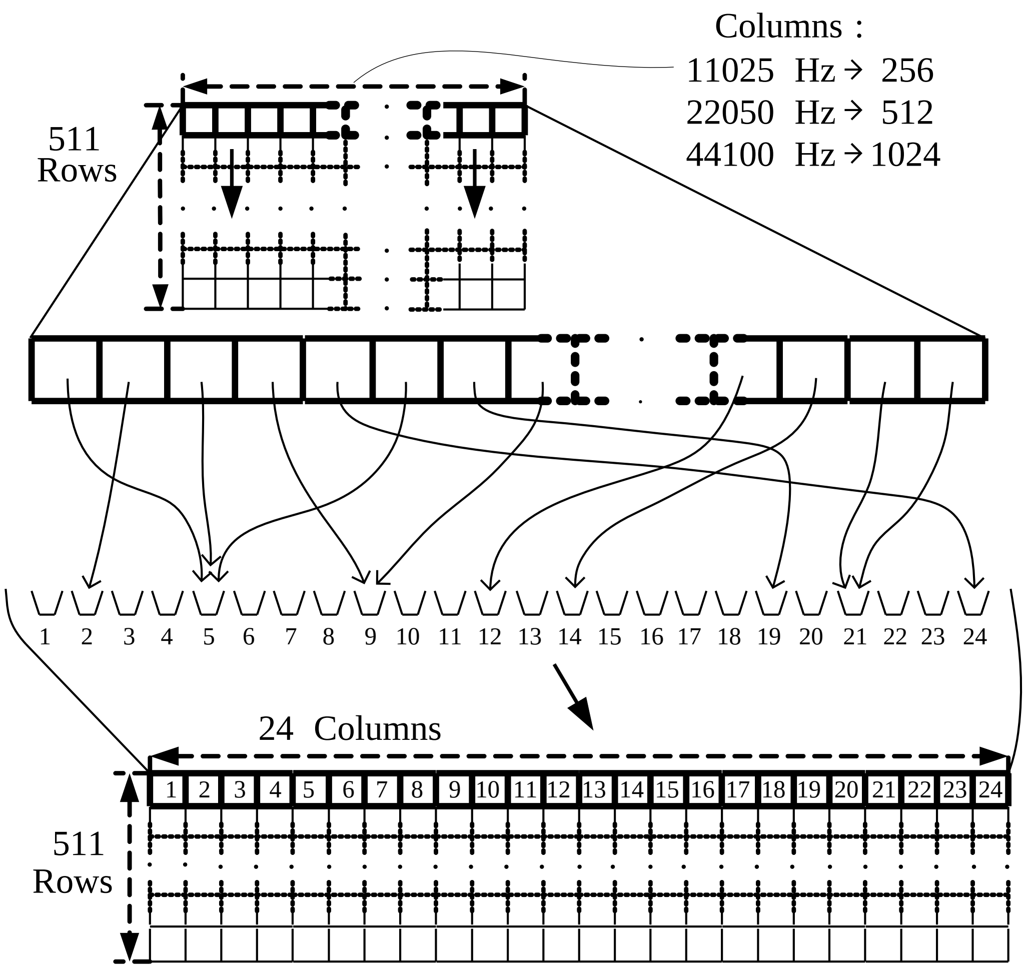


Figure 4.3: Illustration of a STFT applied on a 5.9ms segment. Window size is 23 ms and a Hann-Window with 50 Percent overlap is used. The STFT divides the segment further into 23 ms parts, modifies the amplitude of them following the Hann-Window, then Fast Fourier Transformations (FFT) are computed and finally the result gets stored in a vector with 511 rows and variable columns depending on the frequency of the analyzed file.

Therefore the result of this step is a vector with 511 rows and 256/512/1024 columns.

S2 - Critical Bands (Bark Scale) - [4]

The Bark scale is a psycho-acoustical scale which ranges from 1-24 and corresponds to



Bark Scale critical band ranges (in Hz)

1 : 0 – 100	9: 920 – 1080	17: 3150 - 3700
2: 100 – 200	10: 1080 – 1270	18: 3700 – 4400
3: 200 - 300	11: 1270 – 1480	19: 4400 – 5300
4: 300 – 400	12: 1480 – 1720	20: 5300 – 6400
5: 400 – 510	13: 1720 – 2000	21: 6400 – 7700
6: 510 – 630	14: 2000 – 2320	22: 7700 – 9500
7: 630 – 770	15: 2320 – 2700	23: 9500 – 12000
8: 770 – 920	16: 2700 – 3150	24: 12000 - 15500

Figure 4.4: Process of applying the Bark Scale and the Bark Scale Table

the 24 critical bands of hearing. By applying the Bark scale to the spectrogram 24 frequency bands are accumulated (the data vector per 23 ms segment is transformed from 511 rows and 256/512/1024 columns to 511 rows and 24 columns).

S3 - Spectral Masking - [4]

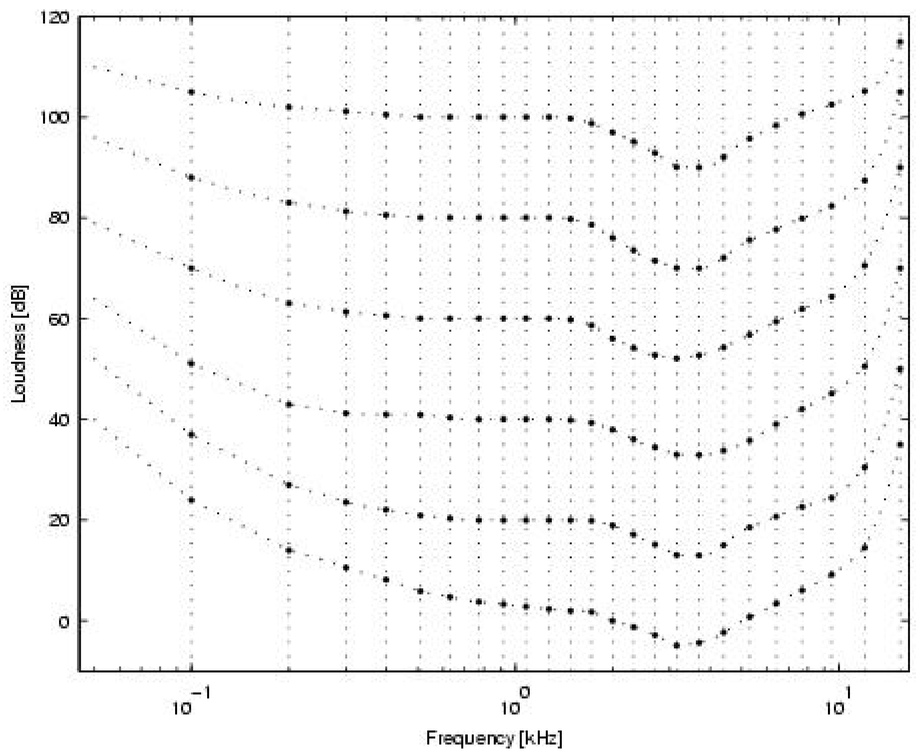


Figure 4.5: Equal loudness contours for 3, 20, 40, 60, 80 and 100 phon. [36]

Optionally a Spectral Masking spreading function is applied to the signal. The occlusion of a quiet sound by a louder sound during simultaneous presentation is called Spectral Masking.

S4 - Sound Pressure Level (dB) - [4]

The loudness is calculated in decibel relative to the threshold of the sense of hearing by transforming the spectrogram to the decibel scale.

S5 - Equal Loudness (Phon) - [4]

The equal loudness levels are calculated from the sound pressure levels using the unit Phon. The human ear does not directly translate the sound pressure level (in decibel) to hearing sensation (in phon), it is not a linear relationship. In Figure 4.5 you can see several equal loudness contours that show which frequencies our hearing sensation is most sensitive to. Notice the nearly equal ascent of all curves from 500 Hz and that the human ear is most sensitive to frequencies from 2kHz to 5 kHz.

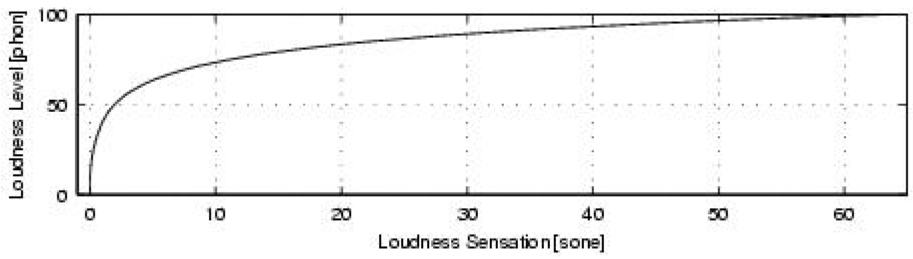


Figure 4.6: Relationship between Phon and Sone, [36]

S6 - Specific Loudness Sensation (Sone) - [4]

The loudness is calculated from Phon to Sone. The loudness of 1 kHz with a sound pressure level of 40 dB is defined as 1 Sone. If sound A is twice as loud as another sound B, sound A has twice the Sone value as sound B. The relationship between Phon and Sone can be seen in Figure 4.6.

All previous steps result in a so called Bark scale sonogram, a power spectrum which describes human loudness sensation.

R1 - Modulation Amplitude Spectrogram (FFT)

By using a Fast Fourier Transformation (FFT) the modulation amplitude of the loudness sensation is calculated of each critical band for each 6 second segment. This results in a Modulation Amplitude Spectrogram. As an FFT is only defined for a datasets with a length of a power of an additional cell is added and filled with zeros. Figure 4.7 illustrates how the FFT is applied on the data and Figure 4.9 shows which parts of the resulting matrix is used for further processing. In this step the size of the matrix changes from 24 x 511 to 24 x 60.

R2 - Fluctuation Strength Weighting - [4]

The effects of amplitude modulation on our sensation is depending on the frequency. Figure 4.8 shows the relationship between fluctuation strength and modulation frequency. The coefficients calculated through the FFT in step R1 are weighted according to this psychoacoustic model which results in a boost around 4 Hz.

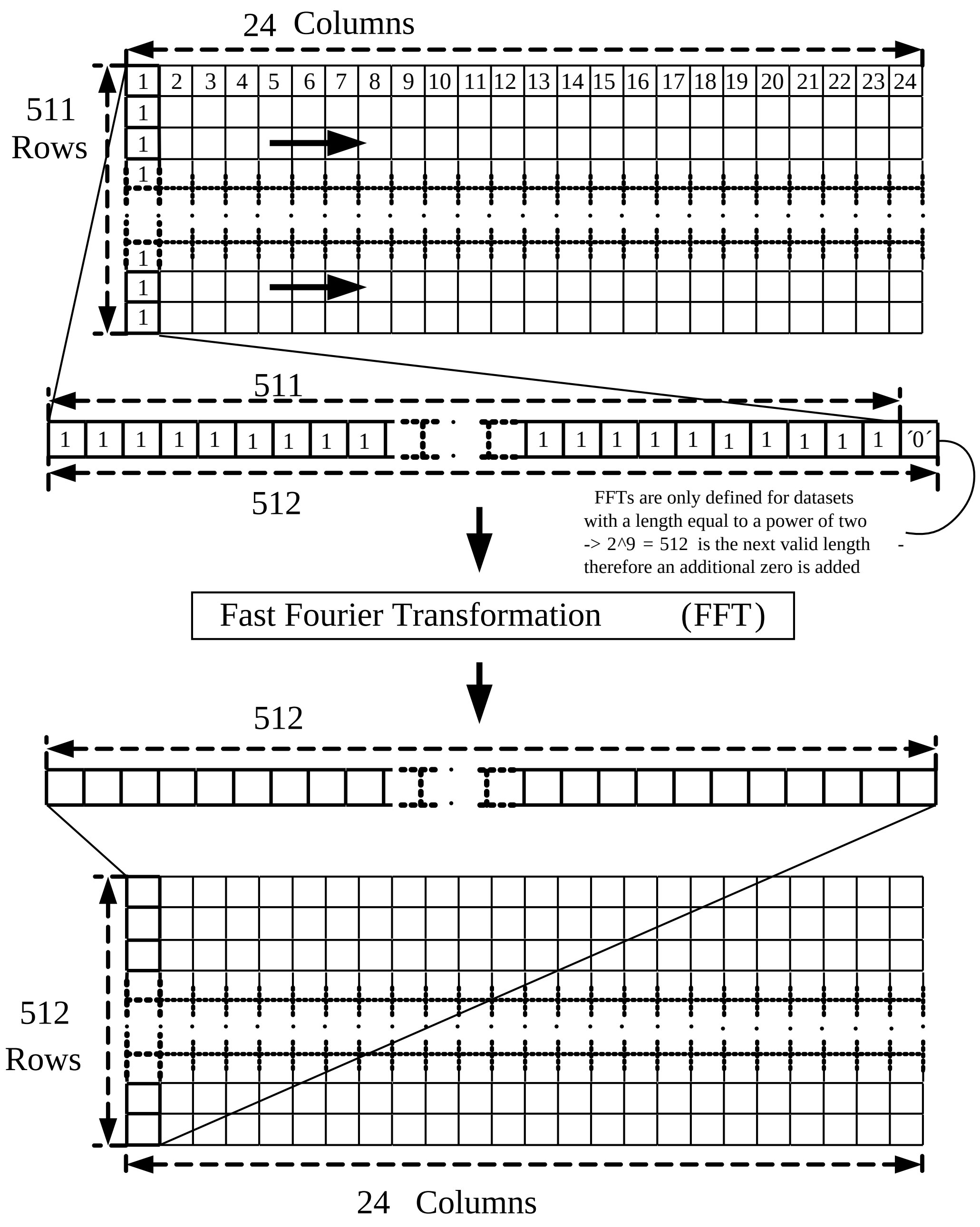

24 Columns
511 Rows
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
511
512
´0´
FFTs are only defined for datasets with a length equal to a power of two -> 2^9 = 512 is the next valid length therefore an additional zero is added
Fast Fourier Transformation (FFT)
512
512 Rows
24 Columns

Figure 4.7: Using the Fast Fourier Transformation to calculate the modulation amplitude of the loudness sensation for each critical band.

R3 - Filtering/Blurring - [36]

The last feature extraction step is the application of a gradient filter and Gaussian smoothing as it may improve likeness of the resulting Rhythm Patterns.

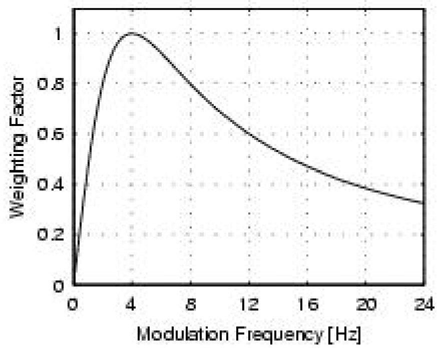


Figure 4.8: The relationship between fluctuation strength and the modulation frequency, [36]

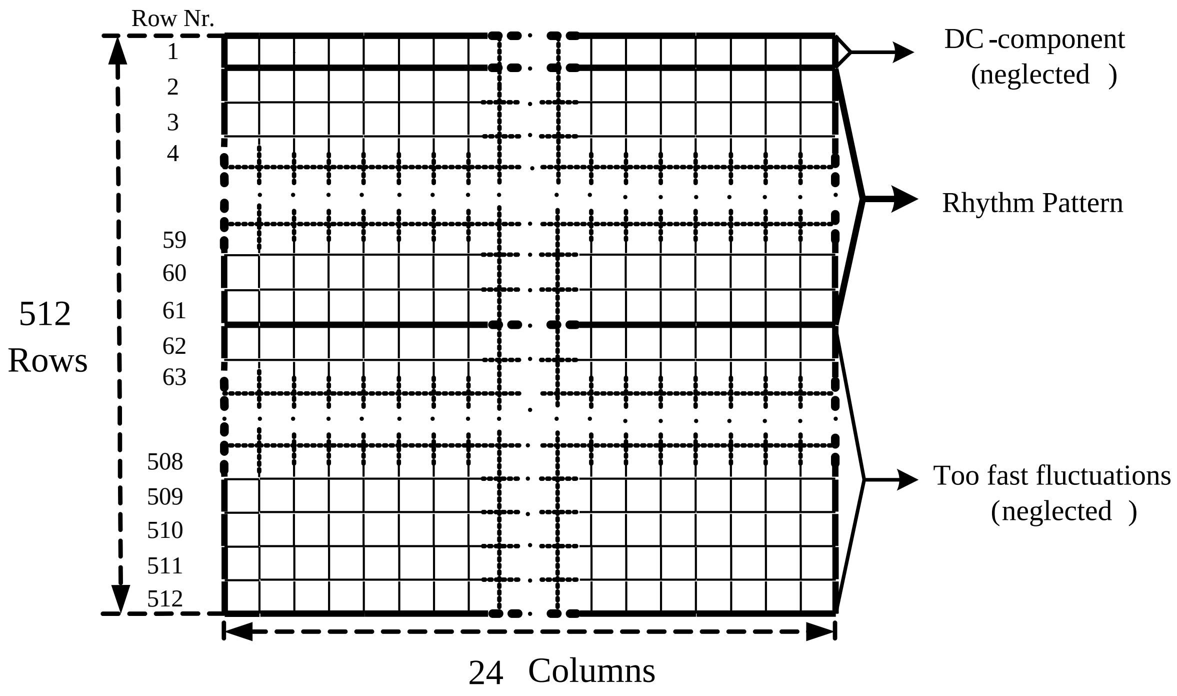
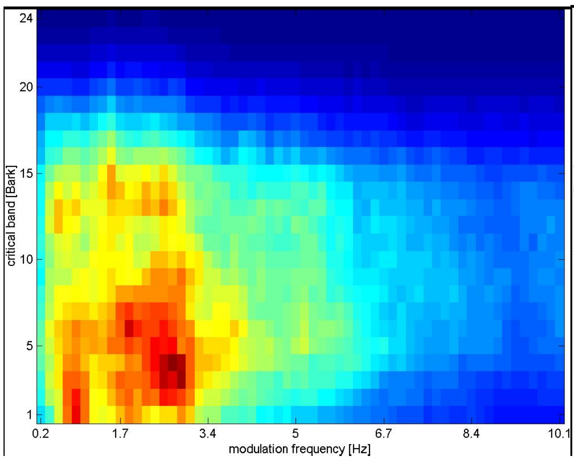


Figure 4.9: Only a part of the data is used when extracting Rhythm Patterns.
Row 1: This Row is the DC-component, which should be zero if an WAV-audiofile is centered around the zero line and therefore can be neglected.
Row 2 - 61: This rows are the actual Rhythm Patterns.
Row 62 - 512: All values over row 61 are neglected because they are seen as 'too fast fluctuations'.

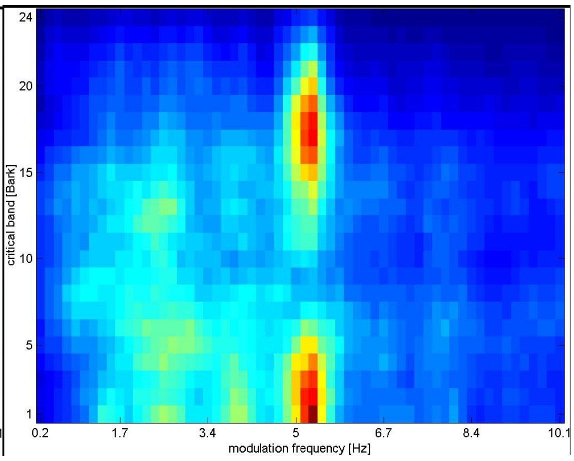
4.1 Rhythm Patterns (RP) - [36]

The rhythm pattern contains information on how strong and fast beats are played within the respective frequency bands. ([36])

Rhythm Patters (RP) are the result if all steps from the initial pre-processing steps to



(a) Classical: Johann Strauß



(b) Rock: Queens Of The Stone Age

Figure 4.10: Two examples for Rhythm Patterns, [18]

the end of R3 are applied (Figure 4.9).

Figure 4.10 shows two examples. The classical piece called 'Blue Danube Waltz' by Johann Strauss and rock piece named 'Go With The Flow' by The Queens Of The Stone Age are compared. The classical composition has a big red and yellow area in the area of low modulation frequencies and lower critical bands, something which is typical for classical music. The result for the rock song shows two distinctive small areas at about 5.3 Hz. The area at the lower bands represents the bass and the area in the higher bands represents e-guitars and percussions.

4.2 Statistical Spectrum Descriptors - (SSD) [19]

After applying all steps including S6 it is possible to calculate the Statistical Spectrum Descriptors (see Figure 4.11). To describe the fluctuations within the critical bands a number of statistical moments are calculated per critical band. The mean, median, variance, skewness, kurtosis, min- and max-value are calculated for each critical band and then the Statistical Spectrum Descriptor is extracted for each segment. The complete SSD feature vector is then generated by calculating the mean or the median of this descriptors over all segments. Regarding to evaluations in [19] it is possible for SSD features to surpass the performance of Rhythm Pattern features if used for music genre classifications depending on the used audio collection.[2]

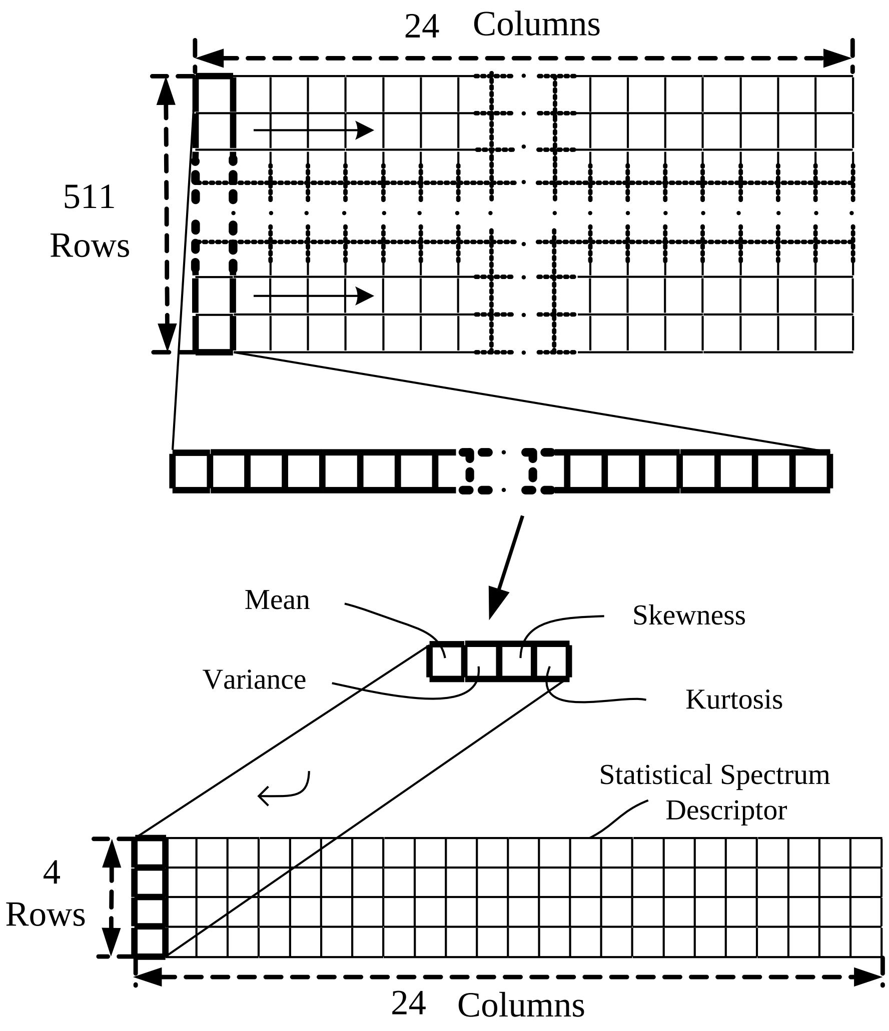


Figure 4.11: Mean, Variance, Skewness and Kurtosis are calculated for each bark band and the resulting 4 x 24 matrix is a Statistical Spectrum Descriptor. Min- and max-value were not used in Thomas Lidy's implementation.

4.3 Rhythm Histograms - (RH) [19]

Rhythm Histogram features are a descriptor for general rhythmic characteristics in a piece of audio. A modulation amplitude spectrum for critical bands according to the Bark scale is calculated, equally as for Rhythm Patterns. Subsequently, the magnitudes of each modulation frequency bin of all 24

[2]In [19] Statistical Spectrum Descriptors outperformed Rhythm Patterns when compared with the GTZAN and ISMIRgenre collections.

critical bands are summed up, to form a histogram of 'rhythmic energy' per modulation frequency (see Figure 4.12). ([18], page 35)

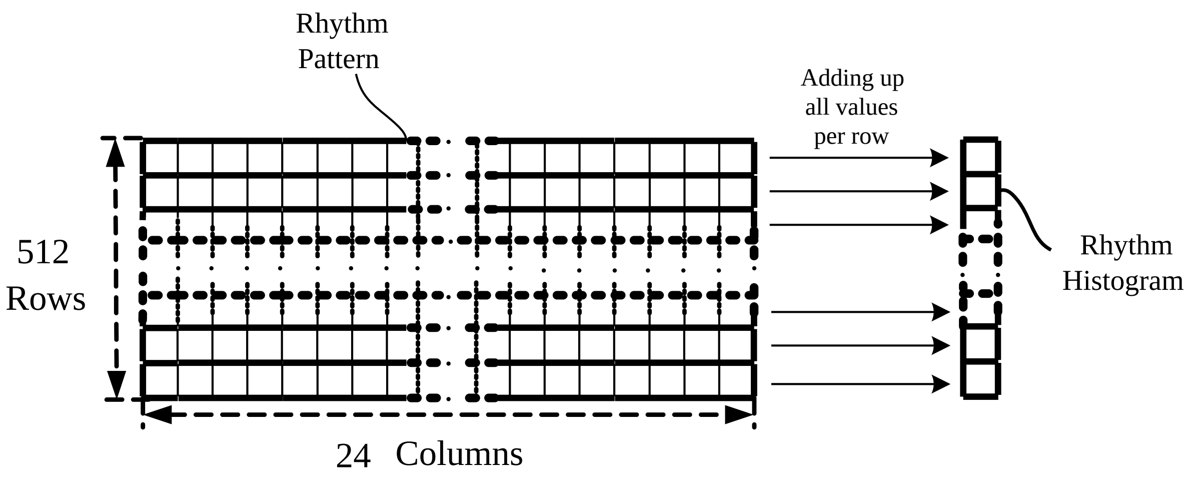
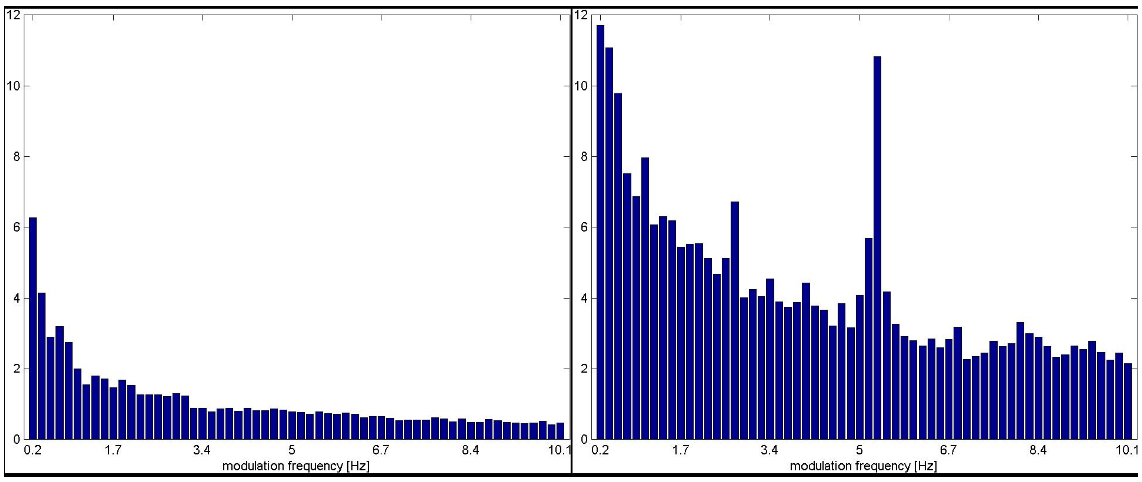


Figure 4.12: The Rhythm Histogram is created by summing up the magnitudes for each modulation frequency.

The Rhythm Histogram feature set is created by computing the median of the histograms of every processed segment. Figure 4.13 compares the Rhythm Histograms of a classical piece and a rock piece, the same music tracks as before). The classical piece generally has less energy and has most of it in the low modulation frequencies. The rock piece has a distinct peak at about 5.3 Hz.



(a) Classical: Johann Strauss (b) Rock: Queens Of The Stone Age

Figure 4.13: Rhythm Histograms, [18]

Chapter 5

Normalization and Distance Measurement

In PublicDJ (if music is compared on basis of content) all music tracks that are submitted each round are analysed and the result of each track is stored in a seperate vector. At the end of each round each vector is, after a normalisation step, compared with the vector of the song that was played in that round. Depending on the chosen mode one or more tracks are added to the playlist.

There are many algorithms for measuring distances available, but as only the **City Block Distance** and the **Euclidean Distance** are used in the application PublicDJ I only cover these two.

5.1 Normalization

In PublicDJ signatures are computed for every music track. Each signature is stored in its own vector containing values each representing the characteristics of a small part of the music track. All signature vectors have the same length and they contain real numbers. Before the distance between two signatures can be computed it is necessary to normalise all the signatures. If we look at the i-th field in all vectors and check in which range the values are and then compare it to a range from any other field it is noticeable that it is different. It is necessary that the values in all fields of the signatures are in the same range so all fields have the same maximum influence on the final distance value.

The applied approach for normalization was to compute the maximum value of all signatures for the i-th field, afterwards to divide every value in the i-th field by this maximum value (as absolute value) and do this for all fields. This results in vectors that all have values between -1 and 1 in every field.

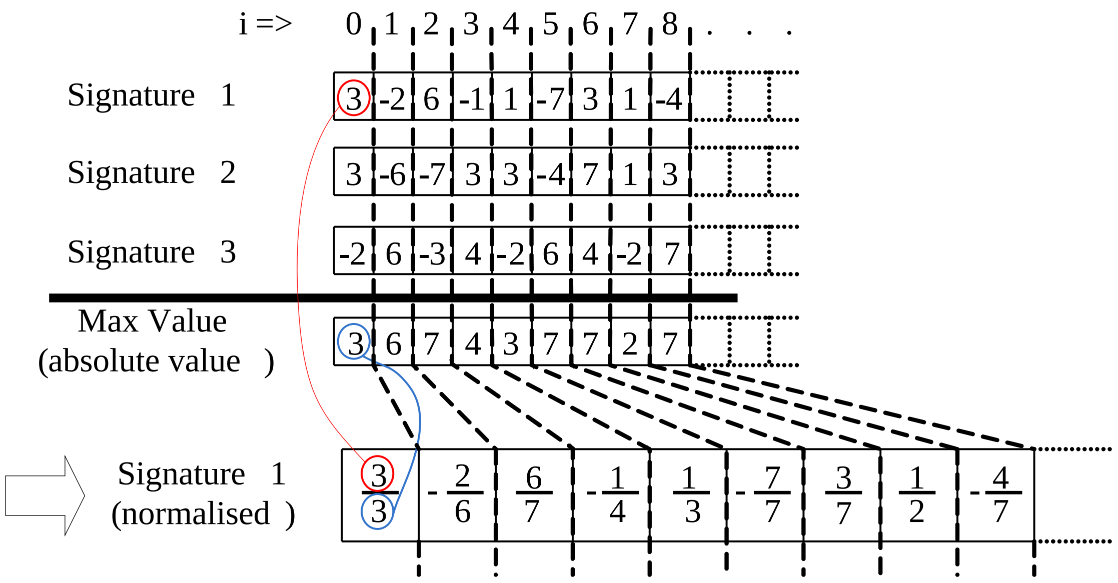


Figure 5.1: Normalization

5.2 Distance Measurement in Euclidean Space

In Euclidean Space the distance between two points is normally computed with the Euclidean Distance (or also called 2-norm distance and L_2 distance). Additionally the City Block Distance (also called 1-norm distance or L_1 distance) was used to provide a different distance computation for comparison.

The Minkowsky Distance[1] is defined as

$$d_{\rho-norm} = \left(\sum_{i=1}^{n} (p_i - q_i)^{\rho} \right)^{\frac{1}{\rho}} \Longrightarrow L_{\rho} \tag{5.1}$$

The City Block Distance (L_1, $\rho = 1$), the Euclidean Distance (L_2, $\rho = 2$) and the Chebyshev Distance (L_∞, $\rho = \infty$) are special variants of the Minkowski Distance.

$$d_{1-norm} = \sum_{i=1}^{n} |p_i - q_i| \Longrightarrow L_1 \tag{5.2}$$

$$d_{2-norm} = \left(\sum_{i=1}^{n} (p_i - q_i)^{2} \right)^{\frac{1}{2}} \Longrightarrow L_2 \tag{5.3}$$

$$d_{\infty-norm} = lim_{\rho \to \infty} \left(\sum_{i=1}^{n} (p_i - q_i)^{\rho} \right)^{\frac{1}{\rho}} \Longrightarrow L_{\infty} \tag{5.4}$$

[1]The mathematic formulas 5.1 - 5.4 are taken from [3]

5.2.1 City Block Distance (L1)

The City Block Distance is similar to the Euclidian Distance as it also is summing up the absolute distances between two vectors (p_i,q_i), the distinction is that it is doing it on a compound by compound basis. The difference is not squared and after summing up the square root is not calculated, which is why it can be seen as a linear variant of the Euclidian Distance as described afterwards.
It is also often called Rectilinear Distance, Manhatten Distance or L_1 distance.

$$d_{cb} = \sum_{i=1}^{n} |p_i - q_i| \tag{5.5}$$

For two 1-dimensional points ($n = 1$) $P = (p_x)$ and $Q = (q_x)$ (as used in PublicDJ) it means that you only have to calculate the difference between the 2 values:

$$d_{cb-1dim} = |p_x - q_x| \tag{5.6}$$

Considered in 2 dimensions ($n = 2$, $P = (p_x, p_y)$ and $Q = (q_x, q_y)$), the City Block Distance is like going from one point of a city to another but with the constraint that you can only walk on the street grid. So the actual City Block Distance would be

$$d_{cb-2dim} = |p_x - q_x| + |p_y - q_y| \tag{5.7}$$

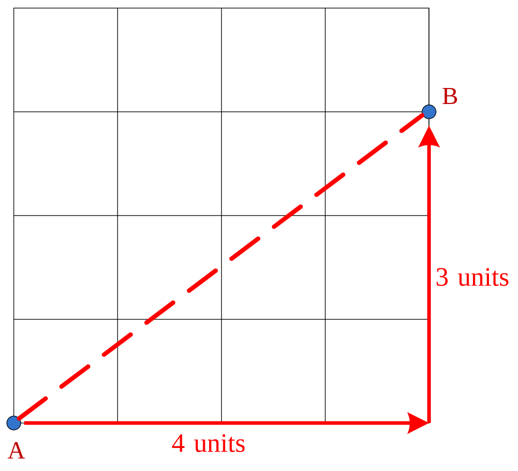


Figure 5.2: Distance Measurement: City Block Distance - 2 Dimensions

5.2.2 Euclidean Distance (L2)

The Euclidean Distance is a very commonly used for measuring distances, thats why it also is often called standard metric or L_2 metric. It can be considered to be the shortest distance between two points. In principle it is the square root of the sum of the squared distances of two corresponding vector values (p_n,q_n).

In an Euclidean n-space the distance between two points $P = (p_1, p_2, ..., p_n)$ and $Q = (q_1, q_2, ..., q_n)$ is defined as

$$d_e = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \tag{5.8}$$

For two one-dimensional points $P = (p_x)$ and $Q = (q_x)$ $(n = 1)$ (as used in PublicDJ) this means:

$$d_{e-1dim} = \sqrt{(p_x - q_x)^2} = |p_x - q_x| \tag{5.9}$$

The absolute value is used as distance is seen as a scalar value that is unsigned.

Considered in 2 dimensions ($n = 2$, $P = (p_x, p_y)$ and $Q = (q_x, q_y)$) it is basically the same as the Phytagorean theorem:

$$d_{e-2dim} = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2} \tag{5.10}$$

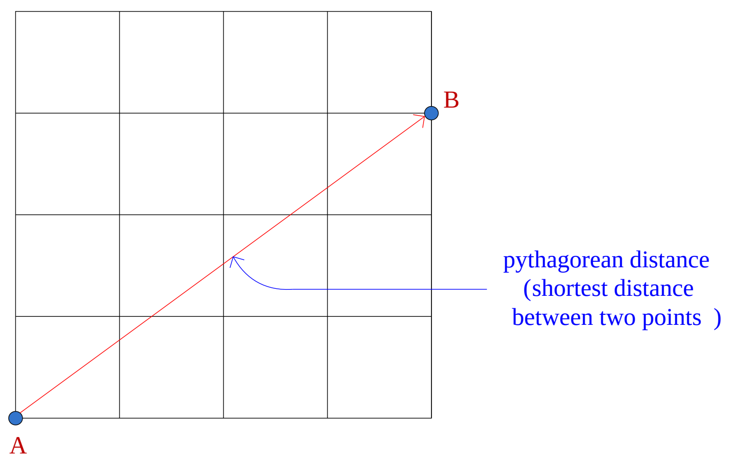


Figure 5.3: Distance Measurement: Euclidean Distance - 2 Dimensions

Part II

PRACTICAL PART

The practical part of this diploma thesis deals with a group of applications called PublicDJ which together form a distributed round based multiplayer game.

This following requirements had to be met:

- There should be a client for users on a Personal Digital Assistant (PDA) with an intuitive user interface.
- There should be a server where the clients could connect to and which implemented all of the application logic.
- Communication should take place over Wireless LAN.
- Users should be able to send MP3s to the server.
- The server should analyze all submitted tracks and map the result to a single signature which can be used for comparing the MP3s with the currently played track and with all other user submissions.
- The song which fits best to the currently played song should be played in the next round and become the next reference song.
- The PDA-client-applications should work on as many devices as possible.

New mobile devices are often hard to use at the beginning because the user has either to dig through manuals or the limited user interface was designed in a bad way. One of the main reasons for this is that often there is no universal operating system and software for the area of mobile devices. The programming language Java ([22]) provides support for platform independent programming. Because of this advantage Java was chosen for implementing all PublicDJ applications.

The following chapters cover the program design of PublicDJ (Chapter 7), the use-cases (Chapter 8), solutions of various encountered issues (Chapter 9) and a detailed documentation about all processes and threads as used in PublicDJ applications.

Chapter 6

Architectural View of PublicDJ

6.1 Design concept

The complete concept is similar to a Three-Tier architecture (see Figure 6.1)

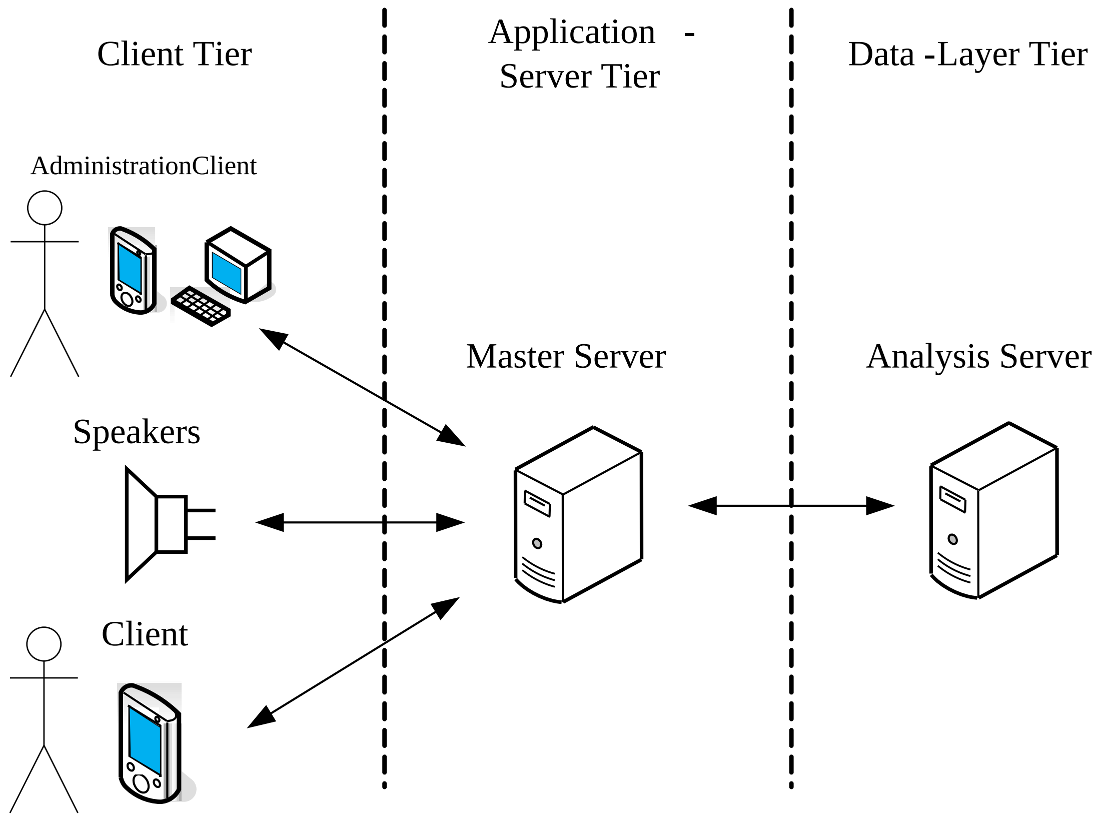


Figure 6.1: Design - Three-Tiers of PublicDJ

6.1.1 Client Tier

The client (called 'Client' later) runs on a Personal Digital Assistant (PDA) with Wireless LAN and a Java-Virtual-Machine (JVM). As Remote-Method-Invocation (RMI) is mandatory for the communication it must be supported by the JVM. The Client also runs on a normal computer with a proper JVM installed.

An additional administrative client (called 'Admin') was developed for managing the main server from a PDA as well. This application implements a user interface for managing the server, which runs without graphical user interface by default. The administrative client can run on the same or any other computer on the network.

6.1.2 Application-Server Tier

There is a central server (called 'Master Server') which implements the complete application logic that also controls the communication between all components. All PublicDJ components connect to and are controlled by the Master Server.

6.1.3 Data-Layer tier

The data-layer of PublicDJ implements the processing of music data. It is called 'Analysis Server'. Here all the algorithms for building signatures of music files are applied. The reason for calling this part data-layer is that this part pf PublicDJ is analyzing the audiofiles. It is the only component that actually accesses the raw audio data and uses it for computations. The Master Server in contrast only accepts the files and stores them, but does not process raw music data.

During the runtime of the application several music files are analyzed. The Server is computationally relieved by outsourcing the CPU-intensive task of analyzing music data to one or more separate servers, the Analysis Servers. They can run on the same physical machine but they can also run on different ones. It is even possible to use a pool of Analysis Servers that are located in the Internet

The hardware setup for this concept is visualized in Figure 6.2.

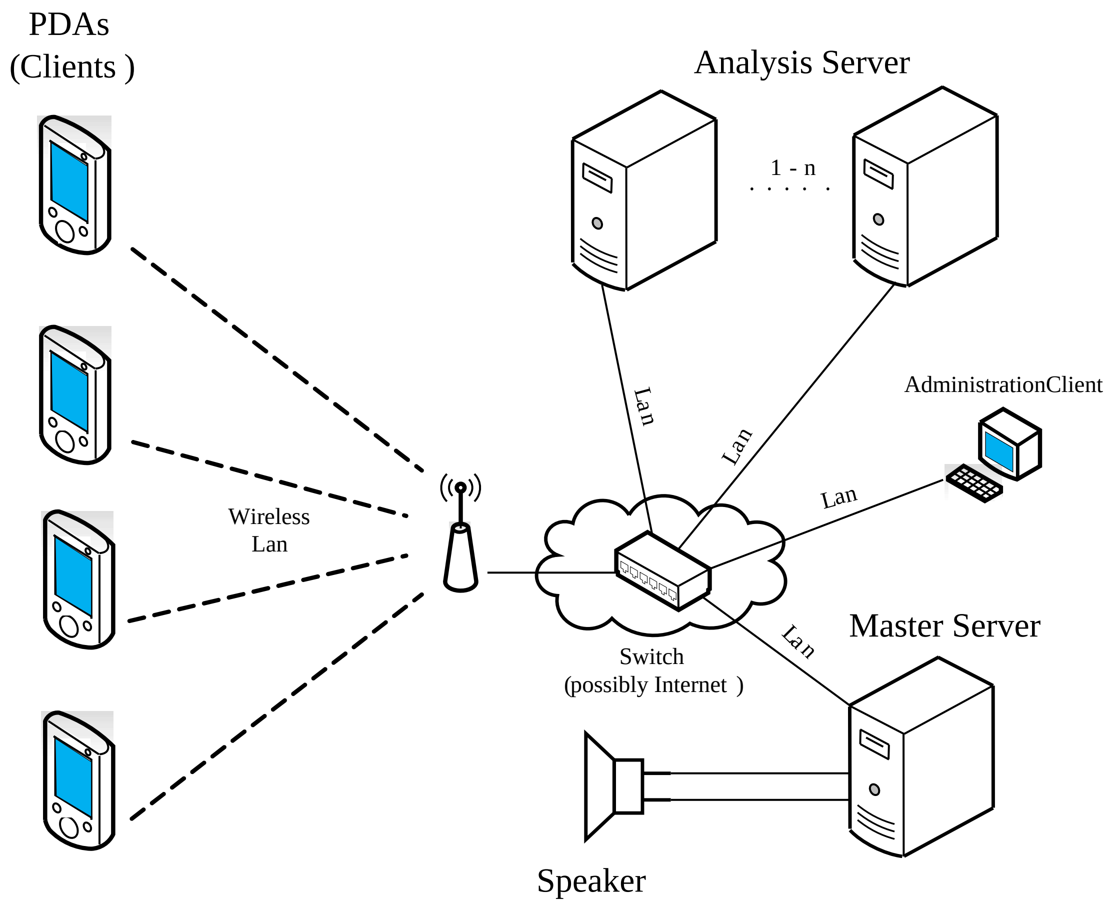


Figure 6.2: Concept: Hardware Setup

6.1.4 Interaction

There are four applications that interact with each other. Figure 6.3 visualizes the interaction of the individual PublicDJ components as well as users. All applications interact only with the Master Server.

Excursion: Remote Method Invocation (RMI)

In most programming languages it is possible to implement remote invocations between computer programs with Remote Procedure Calls (RPC). RPC is a paradigm for implementing the client-server model of distributed computing. By passing messages to a server application it is possible to trigger the execution of methods (in the past also called functions, subroutines or procedures) on a different computer. These messages include which procedure to execute and which parameters are passed to the procedure.

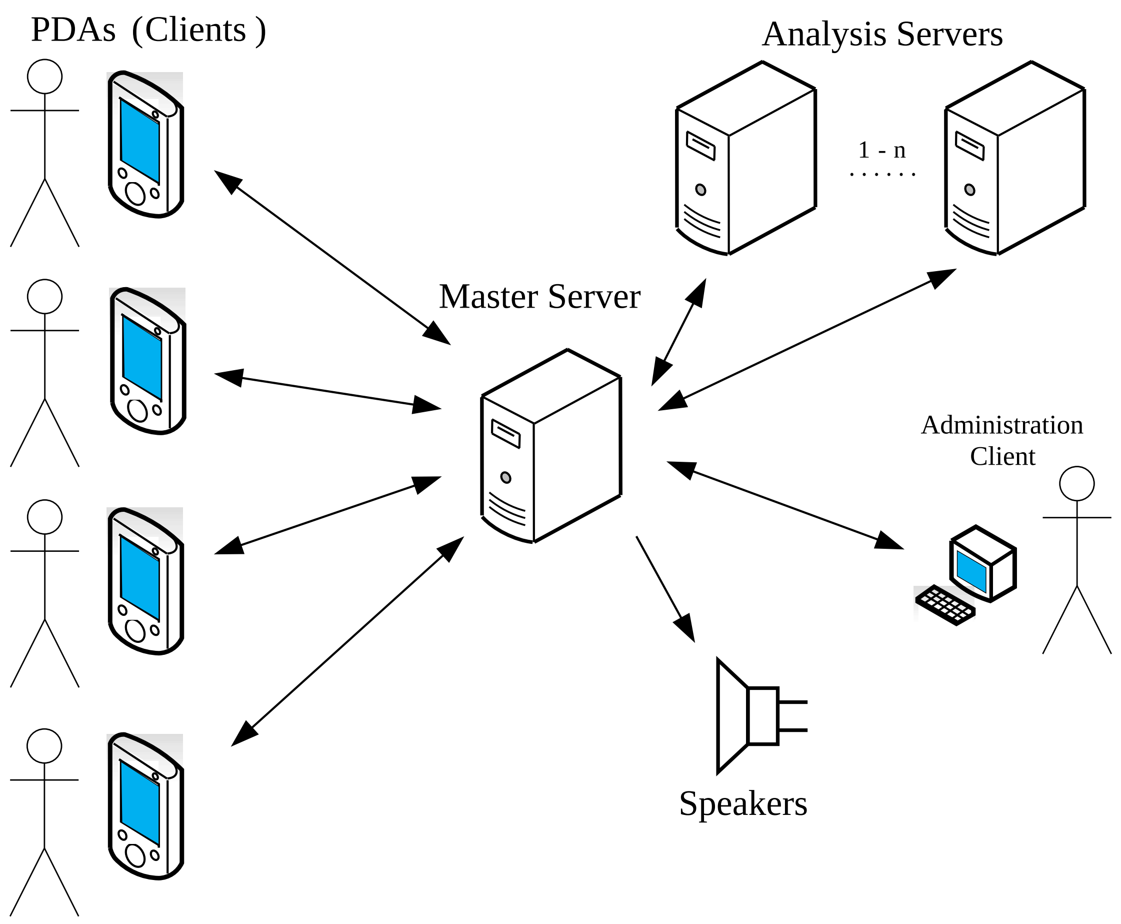


Figure 6.3: Interaction Concept

JAVA's Remote Procedure Call mechanism is called Remote Method Invocation (RMI). Additionally to calls of remote methods with primitive parameters like RPCs, RMI allows to pass objects as parameters. There is no difference in the treatment of locally created or remotely created objects it is possible to execute methods of remote objects. Therefore it is even possible to execute methods of remote objects locally.

RMI separates the definition of behaviour and the implementation of that behaviour. The code that defines the behaviour (defined in the interface) and the code that implements the behaviour are separated. RMI allows both to run on different network nodes.

In RMI a Java Interface is used to define a remote service and the implementation of this remote service is defined in a class. One of Java's main principles is that interfaces define behaviour and classes define implementation. RMI is consistent with that principle. Figure 6.4 shows this separation.

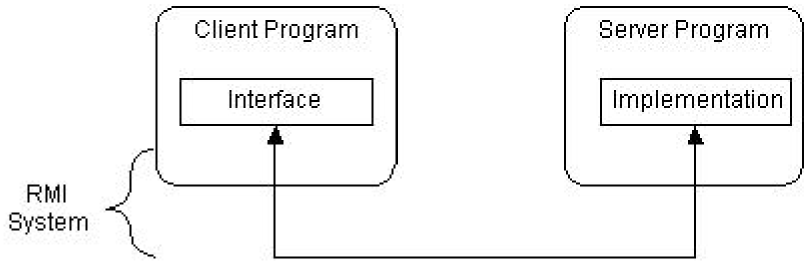


Figure 6.4: RMI: Simplified Architecture of RMI, [24]

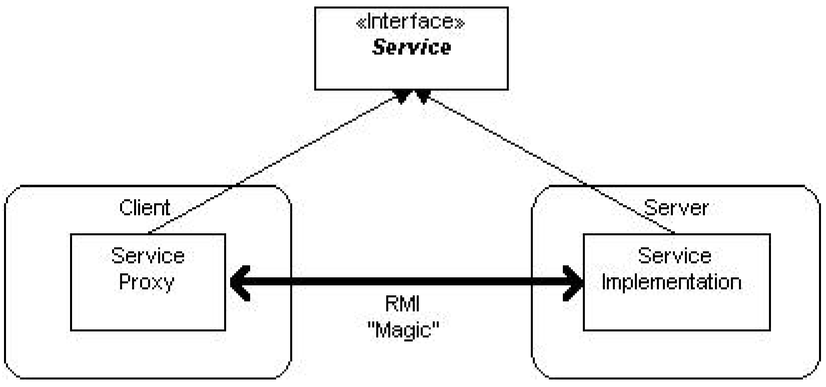


Figure 6.5: RMI: Interfaces, [23]

A Java interface itself does not contain executable code, it is only a definition of a service. Two classes are allowed to implement the same interface in RMI. The first one is the implementation of the behaviour which is executed on the server. The second one runs on the client and it acts as a proxy for the remote service (see figure 6.5). When a client makes a remote method call on the server it actually calls a method on the proxy object. RMI forwards the request to the remote JVM resulting in an execution at the server implementation. Return values provided by the remote service are returned to the proxy and then to the client's application.

Java's RMI consists of three abstraction layers. The first one is the Stub and Skeleton layer which catches all method calls made by the client to the interface and redirects these calls to a remote RMI service. The second layer interprets and manages all references made from clients to the remote service objects and is called Remote Reference Layer. The third one, called the Transport Layer provides basic connectivity and it is based on TCP/IP connections between computers in a network.

Clients find remote services by using a naming or directory service. RMI can use

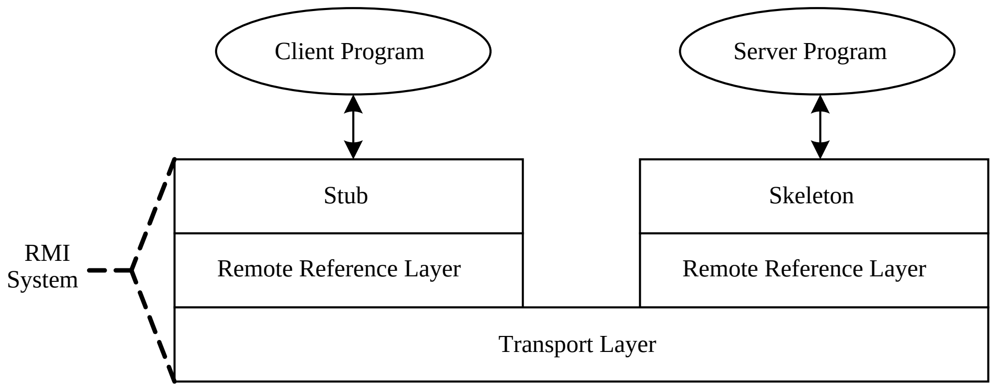


Figure 6.6: RMI: Architecture Layers

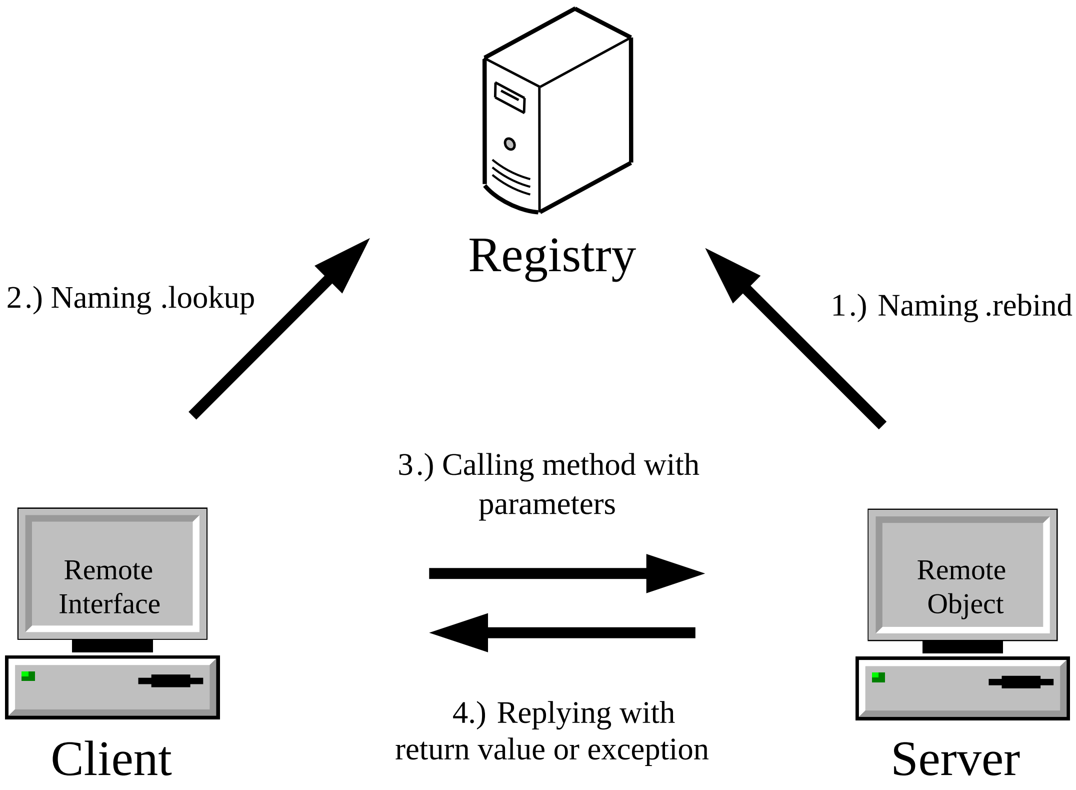


Figure 6.7: RMI: Service discovery (1,2) and remote method invocation (3,4)

different directory services like the Java Naming and Directory Interface (JNDI) or the Lightweight Directory Access Protocol (LDAP). A simple service called RMI Registry, rmiregistry, is included in RMI. Servers announce their service by registering a Remote Object at the RMI Registry with a unique name. The client then looks up this name at the RMI-Registry and receives a reference to the remote object which has to fit to the client's remote interface. By invoking a method of the remote interface a method of the corresponding remote object is executed on the server and finally either a return value or an exception is sent back. Figure 6.7 illustrates this process.

6.2 Design of a round

A round based design was chosen for enabling interactivity between PublicDJ and the participating people. With the start of each new round a new music track which was selected by the Master Server in the previous round is played. Only the first track needs to be chosen manually in most modes. The length of a round varies because it is equal to the length of the music track that is played in that round. From the start of the round until a certain point in time (which is chosen by the Master Server) participating people can transfer musictracks to the Master Server. Its possible that the Master Server cancels the transmission (because there is not enough time to finish the file transfer and to analyze the file). After the transfer of a music file is finished the users have to wait until the start of the next round to find out which submission was chosen.

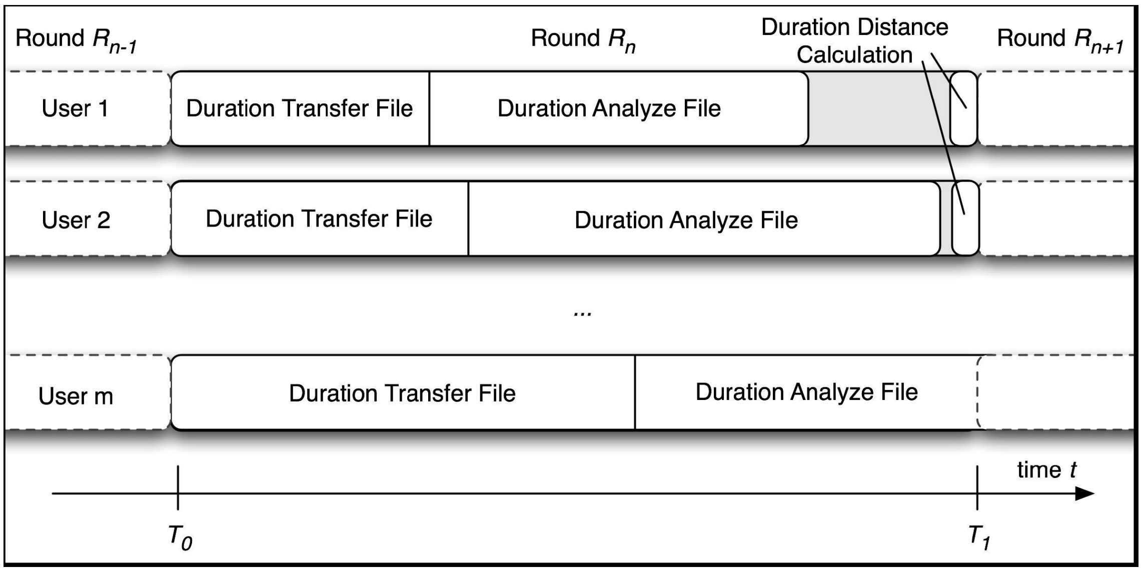


Figure 6.8: Round design, [16]

6.2.1 Description from the Master Server's perspective

Every new round a new musictrack (which was chosen during the round before) starts to play. This track can be seen as the reference track for all participating users in this round. Directly after a new song is started all clients are informed that the new round has started. It is now possible to submit a new track. After a music file is chosen on a Client it is transferred over the network to the Master Server. The Master Server regularly calculates the Expected Time of Arrival (ETA, this is the expected time when a process finishes) for transfers and in case it won't be possible to transfer and analyze a file in time it will cancel the respective transmission. As soon as a transfer is finished, the Master Server triggers the Analysis Server to analyze the music track. The Master Server again regularly requests the analysis progress for every analyzed file. In case it would not be possible to finish the analysis of a file it will cancel the analysis that was started at last. Just before the played track is finished all results of the completely analyzed music files are compared to each other and an algorithm chooses one or more songs which are queued into the playlist. Usually only one track is queued but in this special case of having two or more songs with the same low distance value it will queue all of them.[1]

6.2.2 Description from the Analysis Server's perspective

The Analysis Server is stateless. It simply performs the analysis-tasks that are deployed by the Master Server and reports back the results after analysis. Regularly the Master Server asks the Analysis Server about the progress (in percent) of all analyzed files. If the Master Server detects that an analysis will not finish the analysis before the end of the current round, the corresponding tasks are canceled. The only indication of the end of a round is that the computeAnalyseDifference method (Class AnalyseFile) is called for each completely analyzed file.

6.2.3 Description from the user's perspective (Client)

The user connects with a mobile device that satisfies the requirements of PublicDJ. The mobile device stores a private music collection of the user, from which he can submit songs. When the user starts the application, a short tutorial is displayed to the user that explains the handling of the Client application. When a round is started the user can choose a music track which he/she thinks is appropriate and instruct the Client application

[1] Freeplay mode 7.1 is an example for this behavior as it queues all songs into the playlist because all tracks get the same distance value assigned.

to send it to the Master Server by clicking on a button. After the transfer of the file is finished the user sees the instructions that he/she can only wait until the next round to see the outcome. If the submitted track fits best to the one being played by the Master Server (compared to the the other submissions) then this track will be played next. If this is not the case then the user can make his/her next try in this new round.

6.3 Usecases / Interfaces

The best way to visualize interactions is by modeling them with Unified Modeling Language (UML - [7]) usecase diagrams. This is a way to define possible actions, the different user-groups as well as their relationships (which actors are able to trigger which actions). The implementation of these actions and their effects are neglected on purpose in usecase diagrams for simplicity which leads to easier understanding of the occurring interactions. The users of the Admin and Client Applications are the living actors (and usecase diagrams are usually used only to visualize interaction between users and applications), but these usecase diagrams visualize the interactions between four applications: the Admin, Client, Master Server and Analysis Server. Therefore they become actors from this perspective. The main reason for doing this is that there is only few interaction between users and applications but much more between the applications themselves. In Java, usecases are translated in general into methods of interfaces. Methods of interfaces are only vaguely defined, the real implementation is done in one of the applications. The most important usecases are visualized in UML 2 sequence diagrams in 9.5 at the end of this document.

6.3.1 Admin-Master Server

The Admin was designed to manage the Master Server and all connected Analysis Servers. Therefore it needs to be able to login, modify some analysis settings, prechoose a song for the next round, start and stop rounds. In all modes except Freeplay-mode it is necessary to prechoose a song(as you always need a reference track for comparing the submitted songs to). The Master Server informs the Admin when a new user logged in or disconnected. AliveCheck() is an empty method that is only used for checking if the Admin Application is still connected or not, therefore this can be seen as a function for measuring the availability of the Admin.

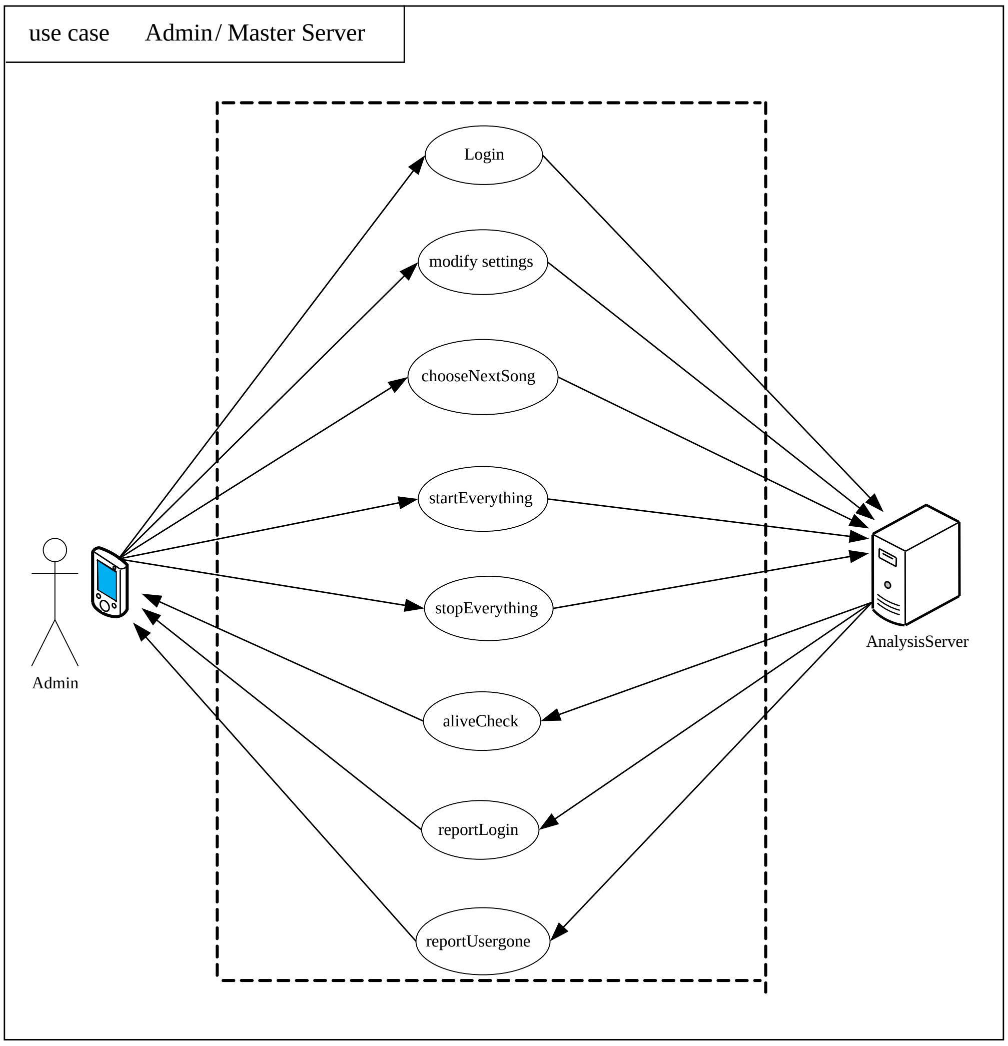


Figure 6.9: Usecase: Admin - Master Server

6.3.2 Client-Master Server

The Client Application has only two functions:

- login to the Master Server
- send a file to the Master Server

It was designed to make it as easy as possible for a new participant to use the Client Application, even if he is unfamiliar with it. The Master Server uses 'aliveCheck' to check if a user is still connected or not, 'newRoundStart' is needed every new round so the Client Application is informed and pushed into the new state. 'stop' is used for telling the Client Application that the current round was stopped. 'Transfer File' is used for transferring files from the Client to the Master Server.[2]

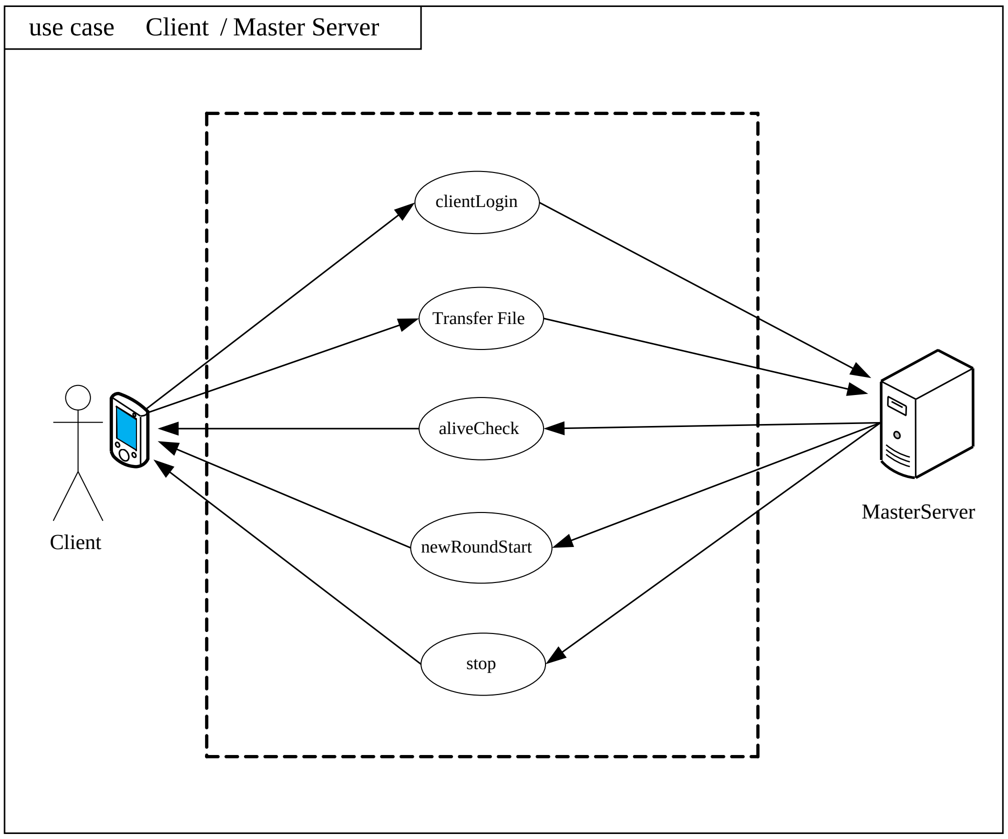


Figure 6.10: Usecase: Client - Master Server

[2]To allow multiple transfers the files are transferred in small packages with the size of 100 KB. If a package was received properly and the transfer was not canceled before the String 'ok' is returned to the Client meaning that the Master Server is waiting for additional packages, if a transfer was canceled before the string 'canceled' is returned.

6.3.3 Analysis Server-MasterServer

The Analysis Server itself can do nothing more but use the function of 'login' on the Master Server. Everything else is triggered by the Master Server. 'setAnalyseParameters' is used to change the settings either directly after login or after the settings were modified by the Admin-Application. All settings are needed for 'Analyse Musicfile', where the Master Server tells the Analysis Server to analyze a specific musicfile (which is the Analysis Server's main task). The Master Server can request the status of the Analysis Server including the analysis-progress for each file in percent. As always 'aliveCheck' is used for testing the availability.

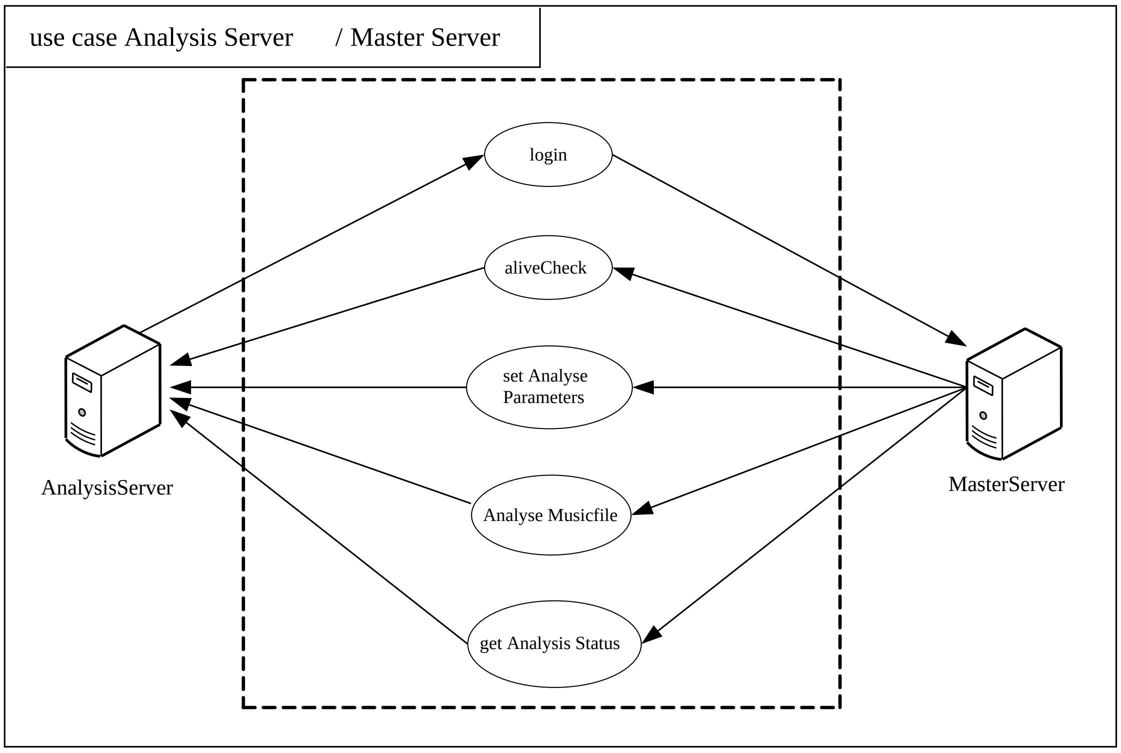


Figure 6.11: Usecase: Analysis Server-Master Server

Chapter 7

Implementation details

During the implementation the following issues had to be solved:

- How to select the next music track each round?

  At the end of every round at least one music track has to be added to the playlist of the Master Server so that a reference song exists for the next round.

- How to cope with limited resources (Wireless LAN bandwidth, processing power)?

  - How to load balance the analysis tasks?

    The architecture of PublicDJ allows to connect several Analysis Servers to the Master Server. As the Analysis Servers may consist of different computers and therefore may have different performance a mechanism was needed for load balancing the analysis tasks properly. The goal was to analyze all submitted music tracks as fast as possible by distributing the analysis tasks among all Analysis Servers so they can finish them in time.

  - How to compute ETAs for file transfers and analysis tasks?

    The length of a round is limited by the length of the current reference song. It takes some time to transfer a song via Wireless LAN and the analysis on the Analysis Server can even take longer depending on the used mode. To be able to manage file transfers and analysis tasks it is therefore mandatory to compute estimations for how long it takes until they finish.

  - When and by which criteria shall file transfers and analysis tasks be canceled?

    Wireless LAN is a shared medium and while several file transfers are taking place at the same time it can take a long time until all files are transferred completely. All these files need to be analyzed afterwards and (depending on the analysis mode and the Analysis Server's hardware) this probably takes even longer than the file transfers. The time for transferring and analyzing the files is limited by the length of the round, so that mechanisms are needed to ensure that as many songs as possible are successfully transferred and analyzed.

The following sections present the approaches for these challenges in PublicDJ.

7.1 Selection of the next music track

PublicDJ implements a playlist generation concept that is independent from the chosen mode. The different available modes only differ in the way signatures are created. While a reference song is playing, song submissions are accepted and signatures are computed for them depending on the selected mode. The end of the reference song causes the round to end as well. In case of content-based modes all signatures (which are effectively vectors with multiple values) are normalized following the concept presented in Chapter 5.1. In metadata-based modes there is no need for normalizing signatures as the they only consist of single values. In the next step distances between the reference song's signature and all other signatures are computed(in content-based modes following chapter 5.2, in metadata-based modes following the simple equations presented in this chapter) resulting in single values. Those values represent the distances between the signatures of the reference and the compared songs. The bigger the distance is the more different are the songs. The music track with the lowest distance gets queued into the playlist. If several tracks have the same lowest distance value then they all get queued. Figure 7.1 visualizes the process of song selection.

PublicDJ features several different modes for the selection of the next music track. As mentioned before these modes have influence on how the signatures for the music tracks are computed. In total there are three categories: Freeplay, metadata-based modes and content-based modes.

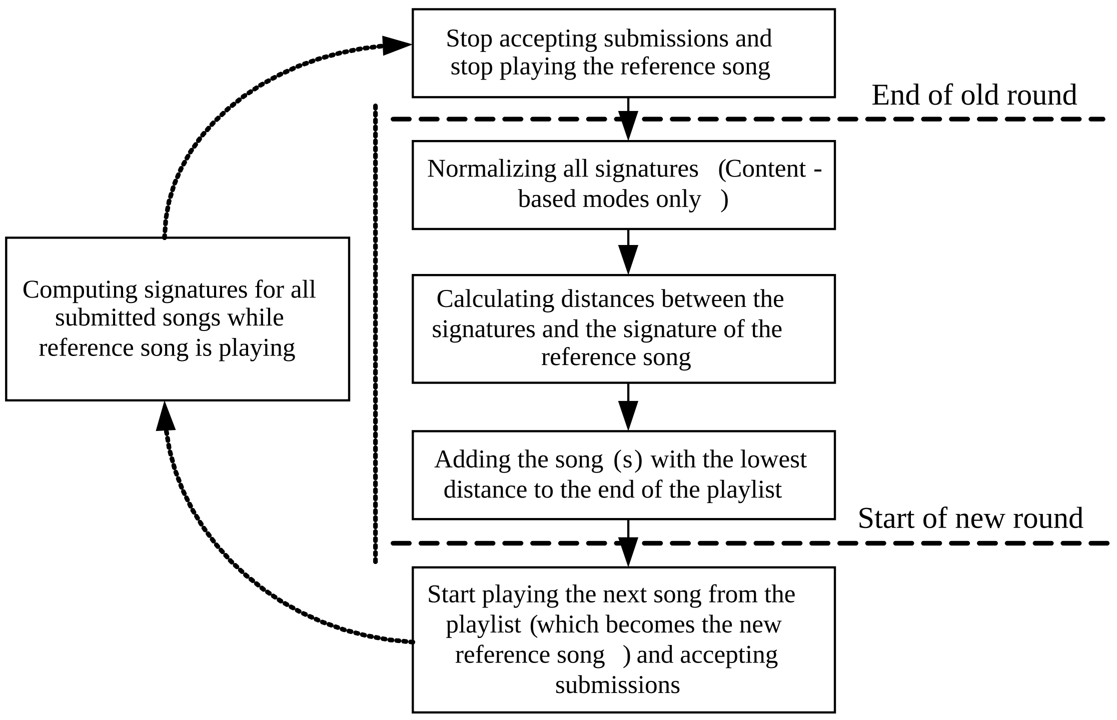

Stop accepting submissions and stop playing the reference song
End of old round
Normalizing all signatures (Content - based modes only)
Calculating distances between the signatures and the signature of the reference song
Adding the song (s) with the lowest distance to the end of the playlist
Start of new round
Start playing the next song from the playlist (which becomes the new reference song) and accepting submissions
Computing signatures for all submitted songs while reference song is playing

Figure 7.1: Song selection: Concept

- **Freeplay**

  In this mode all submitted tracks are not analyzed at all and therefore are treated exactly the same by assigning them the same default value. All tracks are just queued to the playlist.

- **Metadata based modes**

  In this mode the metadata of the MP3 files is extracted for calculating the distance between the music tracks. The difference variable is set to a very high value by default (which reflects a very big distance). It has three different submodes.

  - **Same_Genre**

    The genre field of the metadata is used for case-insensitive comparison. If the genre strings in 2 songs are exactly the same the difference variable is set to 0, if not it will return the default value.

  - **Similar_Genre**

    In this mode the difference-variable is set to 0 if the genre-string of the analyzed song is a substring of the genre of the playing one or vice versa.

  - **Year**

    Nearly all MP3s use the year field. If no year is set then the difference is left at default. If a year is set it is calculated by subtracting year2 from year1 and then computing the absolute value. In the following equation year1 and year2 represent the years of creation of two compared songs.

$$distance = |(year1 - year2)| \tag{7.1}$$

- **Content based modes**

  There are three different modes for content based playlist generation.

  - **Fluctuation Patterns**

    In this mode Fluctuation Patterns (also called Rhythm Patterns) are used for computing signatures for music tracks. Fluctuation Patterns/Rhythm Patterns are described in chapter 4.1.

  - **Statistical Spectrum Descriptors**

    In this mode Statistical Spectrum Descriptors are used for computing signatures for music tracks. Statistical Spectrum Descriptors are described in chapter 4.2.

  - **Rhythm Histograms**

    In this mode Rhythm Histograms are used for computing signatures for music tracks. Rhythm Histograms are described in chapter 4.3.

7.2 Resource Management

As depicted in Figure 6.8, a round is divided into several phases. Songs are submitted and analyzed afterwards. The extracted feature attributes are normalized, distances are calculated and the respective best matching song is selected. Those phases are not visible to the users, as they can start submitting their selection anytime during the round. If the submission can't be finished during the round, it is canceled by the Master Server and the user is suggested to resubmit the selected song during the next round. The same is valid for the analysis. If a song can not be analyzed totally till the end of the round, the analysis of the song is canceled and the submission is not considered in the selection process. This explains the demand for estimated time amount calculations on the Master Server for each transmission and analysis and a management of the limited resources.

PublicDJ had to manage the following resources:

- Network bandwidth for file transfers
- CPU cycles for analyzing the music

Both are limited by the time of the currently played song. All files are transferred via wireless LAN and this is a shared medium. It can be compared to network hubs that were used before network switches became common. Current technology supported by PDAs is the IEEE 802.11b[1] standard which is too slow for PublicDJ. Only the latest PDAs and smartphones feature the IEEE 802.11g[2] standard which enable these devices for PublicDJ.

If playlists should be generated with a content-based approach then the resources on the Analysis Server are critical. Audio feature extraction can be very processor intensive and it often takes about 20 seconds per music track. It is possible to connect several Analysis Servers concurrently to the Master Server and to load balance for distributing the load evenly among more computers.

PublicDJ was targeted to consider these critical issues and to implement an appropriate solutions as shown in the next sections.

7.2.1 Load Balancing

Load Balancing the analysis tasks can improve performance if there are multiple Analysis Servers. Every Analysis Server can itself analyze several files concurrently and it is configurable via the configuration file how many simultaneous analyses should be performed. The maximum amount of concurrent tasks reflect the performance of the used computer. A slow workstation can be configured for only one concurrent analysis, a fast one e.g.

[1]IEEE 802.11b is able to transfer about 11 Mb (1̃.3 MB) per second. Because of the protocol overhead caused by the headers of the network packages and the relatively small maximum size of the payload compared to Fast Ethernet the average transfer speed is significantly lower (about 0̃.76 MB per second, [6]). The average size of MP3s is about 4 MB and 5 or more users should be able to transfer concurrently (Transferring 20 MB takes about a minute)

[2]IEEE 802.11g is able to transfer about real 14 Mbit/s => 1.75 MB per second (the transfer of 6 music files with 4 MB each take about 13.7 seconds), [44].

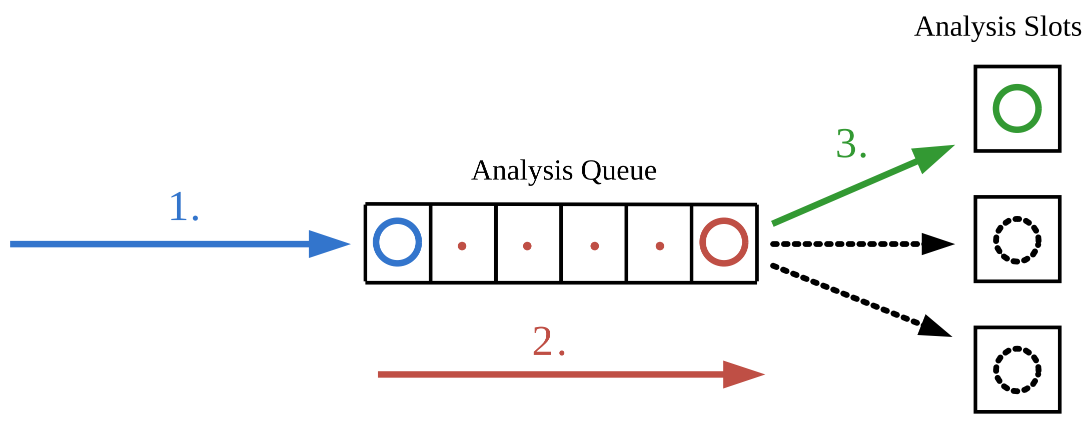


Figure 7.2: Load Balancing: Analysis Queue and Analysis Slots
1. Analysis-tasks are assigned to the Analysis Server by the Master Server and arrive at end of the queue where they are temporarily stored.
2. All tasks advance by one whenever a task from the analysis slots is successfully completed and the first one is fetched from the queue . The size of the queue decreases by one each time.
3. The first analysis-task is fetched from the queue, put into the analysis slot and started whenever an analysis-slot is free.

for four. Additionally to the analysis slots there is an analysis task queue implemented which temporarily stores all assigned analysis tasks in case all slots are in use. Figure 7.2 illustrates how the queue is working.

An algorithm for load balancing has been developed. Figure 7.3 shows the three assignment rules after which the Analysis Server is chosen for receiving and later processing the task.

7.2.2 ETA calculation

The file transfers and the analysis were implemented in such a way that it is always possible to request the progress of any analysis in percent. By requesting and storing all progresses from all Analysis Servers every few seconds it is possible to estimate when the analysis is going to end. Under the assumption that the load progress is constant/linear over time the last few saved points are used to compute an average progress per second in percent. x is the number of points used for computing the progress speed in 7.2 and 7.3.

$$timeWindow = time_n - time_{n-x} \tag{7.2}$$

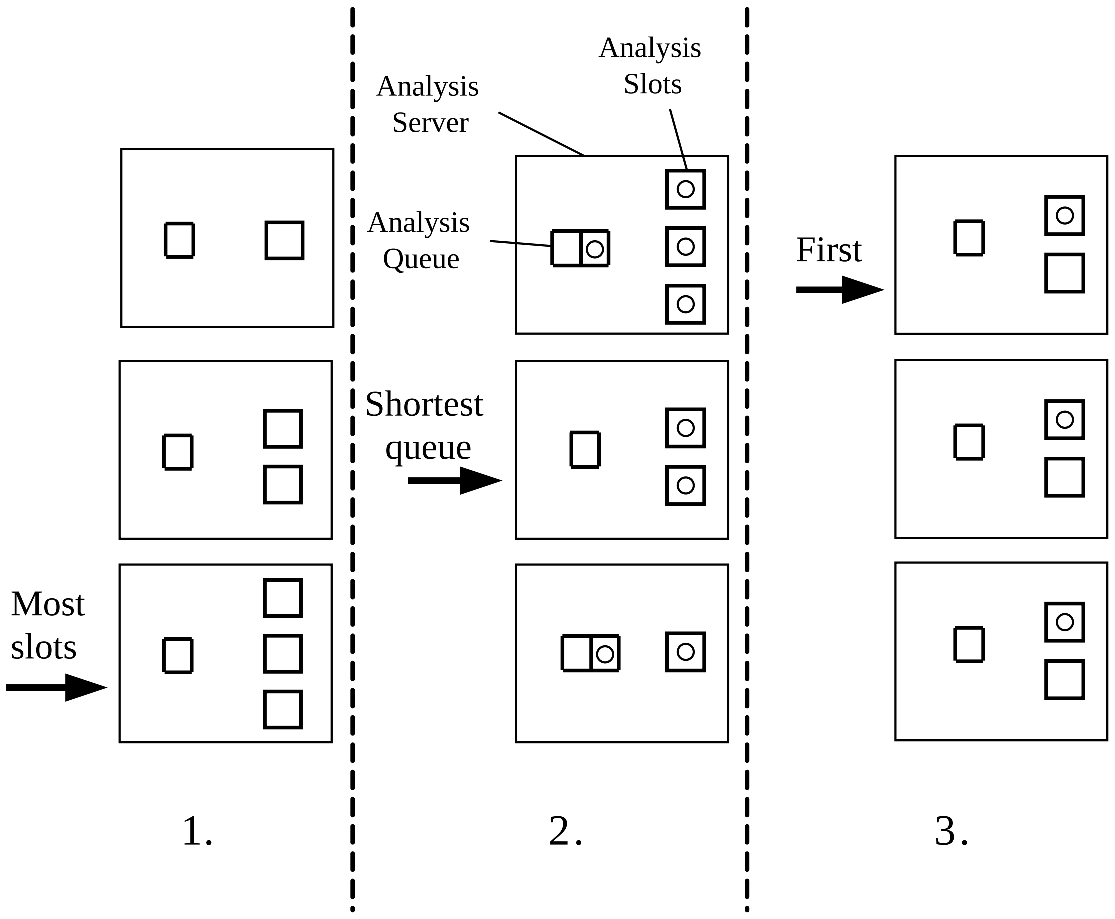


Figure 7.3: Load Balancing: Assignment Rules
1. The Analysis Server with the most free analysis-slots is chosen.
2. If all slots of all Analysis Server are full, then the shortest queue is the next criterion.
3. If all Analysis Server have the same amount of free slots or no free slots and a queue of the same length then simply choose the first one.

$$progressChange = progress_n - progress_{n-x} \tag{7.3}$$

If you divide the progress-change by the time-window you actually get the progressSpeed per second.

$$progressSpeed = \frac{progressChange}{timeWindow} \tag{7.4}$$

By dividing (1 - progress) by the progressSpeed a rough ETA is calculated.

$$ETA = \frac{1 - progress}{progressSpeed} \tag{7.5}$$

7.2.3 Transfer/Analysis Canceling Mechanism

Transfers need to be canceled when there is not enough time to finish the transfer of a song and the following analysis. Analyses need to be canceled if there is not enough time to finish an analysis before the round ends. ETAs for both the transfers and the analyses are calculated (as described in previous section, 7.2.2) every five seconds and checks are performed to find out which transfers and which analyses need to be canceled.

Compared to the analysis canceling mechanism the transfer canceling mechanism is simple. Transfers simply do not consume that big amount of CPU-time compared to audio feature extraction which is needed for content-based playlist generation - only network bandwidth is the bottleneck. The equation 7.6 describes the implemented check for transfers.

$$ETA_{filetransfer} + time_{analysis} > time_{remaining} \tag{7.6}$$

$ETA_{filetransfer}$ is the ETA for the file transfer and the $time_{remaining}$ simply is the time until the end of the current round. $time_{analysis}$ is not an ETA, it is the average time needed for analyzing a song based on former analyses. The Master Server stores the duration of the last finished analysis for each Analysis Server. Every time an Analysis Server finishes an analysis it reports this to the Master Server which updates the duration the Analysis Server needed to analyze the song. Initially this value is predefined via the Master Serverïs configuration file, but after a single analysis by every Analysis Server this value is quite accurate. In equation 7.7, $duration_i$ is the duration that Analysis Server i needed to finish its last analysis.

$$time_{analysis} = \sum_{i=1}^{numberofAnalysisServers} \frac{duration_i}{i} \tag{7.7}$$

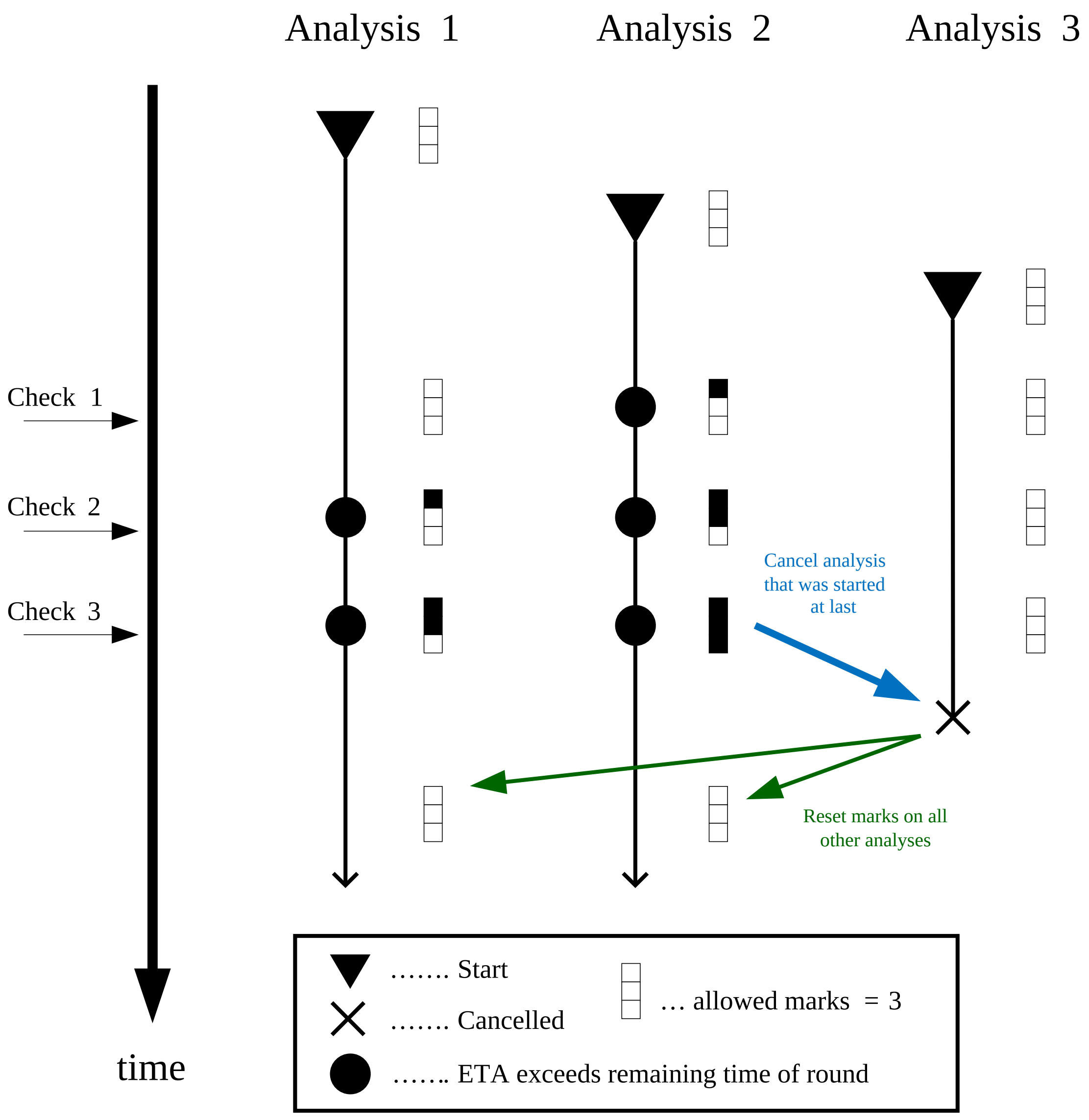
This implementation proofed to be sufficient for canceling transfers.

Managing the analysis-tasks is more complicated as it takes longer to finish them than transferring files. No task is canceled the first time the ETA is exceeding the remaining round time. It may happen that for several reasons one file is analyzed slower than all others over a short time and its progressSpeed is only temporarily very low. The main idea behind canceling analyses is that resources of the Analysis Servers get freed so that it is more likely that other concurrent analyses finish in time. Because of the assumption that analyses which were started earlier are more likely to finish the last started analysis is canceled whenever a problem occurs. Usually the additional load on the Analysis Server that is caused by the last started analysis is the reason why it cannot cope with all concurrent analyses anymore. Therefore it seems reasonable to cancel the last started analysis.

The basic idea of the implemented approach was that analyses can collect marks. Whenever the ETA of an analysis is longer than the remaining time of the current round a mark is added to that analysis. If an analyses collects a certain amount of marks it gets canceled. If the boundary for canceling for example is three marks then it takes three checks to cancel an analysis(which would take 15 seconds as the checks are performed every 5 seconds). It proofed to be sufficient to cancel only one analysis per check but this is configurable in the Master Server's configuration file. Figure 7.4 illustrates the basic mechanism.

These are the rules for the marking-system:

- If the ETA of an analysis exceeds the remaining time of the current round add a mark to that analysis.
- If an analysis has collected more marks than allowed cancel the last started analysis on that Analysis Server (but only if there were not too many other analyses canceled on that Analysis Server in this check)
- If an analysis was canceled the marks of all other analyses are reset after this check.



Analysis 1
Analysis 2
Analysis 3
Check 1
Check 2
Check 3
Cancel analysis that was started at last
Reset marks on all other analyses
time
……. Start
……. Cancelled
… allowed marks = 3
……. ETA exceeds remaining time of round

Figure 7.4: Analysis Canceling Mechanism

Chapter 8

Setup of the program environment

8.1 The Client - on a PocketPC

In this section it is described how to setup a Dell Axim x51v so that it supports all the given requirements.
The requirements are:

- Wireless LAN support
- A working Java VM
- RMI support for the Java VM

8.1.1 Installation of IBM J9 Java VM on the Dell Axim x51v

Not all mobile devices are the same. Only for a few a working Java VM exists. Its even more unlikely that this JavaVM supports RMI. At the time of the experiments several Java VMs like the Sun Java ME VM (together with the Java Wireless Toolkit 2.3 which included the needed classes for RMI), the J9 Java VM from IBM, CrEme JVM from NSICOM (not available anymore) and other JVMs were tested that claimed to support RMI. The only Java VM where RMI worked properly over wireless LAN on the Dell Axim x51v has been the J9 Java VM from IBM. Notice that the Dell Axim x51v uses Microsoft Windows CE 5.0 as an operating system and therefore it is only possible to access the Dell Axim x51v if you have ActiveSync installed. These instructions can be directly applied on a workstation running Microsoft Windows 2000/XP.

1. Make sure you have your mobile device already connected to the workstation and that ActiveSync[1] is already recognizing it properly

2. Download the J9 Java VM evaluation version provided by IBM. [2]

3. Install the Java VM on your workstation. ActiveSync installs the J9 runtime to the */Program Files/J9* directory on your PocketPC

4. Download the *Workplace Client Technology, Micro Edition 5.7 Integrated Package* from IBM[3] and install it on your workstation. Although usage of this application is limited to only 30 days we need it to get access to the RMI Optional Package.

5. After installation start the *Device Developer Enviroment* which is based on Eclipse 2.0. Go to the help menu and start the *Update Manager*. Click on *Sites to Visit* and select *IBM WebSphere Everyplace Device Developer Technologies/Technologies/WEME RMIOP* and install the package. It will be installed to each runtime environment on *C:/Program Files/IBM/DeviceDeveloper/wsdd5.0/ive-2.2/runtimes*.

6. Copy the new files to your Dell Axim x51v: since we are using the Personal Profile, copy the *rmip.jar* and *rmip_nl1.jar* (optional language pack), from *C:/Program Files/IBM/DeviceDeveloper/wsdd5.0/ive-2.2/runtimes/wm2003/arm/ppro10/lib/jclPPro10/ext* to the */Program Files/J9/PPRO10/lib/jclPPro10/ext* folder on your Dell Axim x51v. This adds the RMI support to the J9 environment. This version of RMI is compatible with the JDK1.3/JDK 1.4.2 environment and supports most of its features.

8.2 Installation of the required packages on the workstations

The user has to install a recent Java SDK on the workstations. The advantage of using a Java SDK of Version 1.4.2 is that it can generate 1.4.2-compatible code which directly

[1] ActiveSync 4.5 - http://www.microsoft.com/downloads/details.aspx?FamilyID=9e641c34-6f7f-404d-a04b-dc09f8141141\&displaylang=en

[2] To do this go to their webpage(http://www-306.ibm.com/software/wireless/weme/), click on *Trials and Demos* on the left sidebar, then download *WebSphere Everyplace Micro Environment 6.1.1 - CDC 1.0/Foundation 1.0/Personal Profile 1.0 for Windows Mobile 5.0/ARM* for PocketPC.

[3] Workplace Client Technology, Micro Edition 5.7 Integrated Package http://www-306.ibm.com/software/wireless/wctme/bundle.html

works on the Dell Axim x51v with the J9 Java VM installed. This is a clear advantage and therefore this version was used.

One of the biggest problems to solve was that the Sun Java SDK does not support the MP3(or OGG)-Format natively. Reimplementing this wheel was out of scope of this thesis as there was already a working jar-package from `http://www.javazoom.net/mp3spi/mp3spi.html` which could be used for enhancing the JRE/SDK. This package consists of 2 jar-files, which are used for accessing the ID3 Tags of MP3s and for getting direct access to the raw music-data (tritonus-package, jlayer package). Some more classes integrate them into the Java JDK/SDK via the so called Service Provider Interface.

Chapter 9

PublicDJ Documentation

All four applications (Master Server, Client, Admin, Analysis Server) were clustered in the package PublicDJ. There are two additional packages which are imported by every application package, interfaces and data structures. These two packages were created to avoid unintended errors and to reduce the code in total (instead of having the same code for the interfaces and data structures multiple times in the relevant sub-packages as they implement common functionality required by all applications). Nevertheless the lines of code (LOC) exceeded 9000 lines, including 896 comment lines (about 9 %). Figure 9.1 shows these structures.

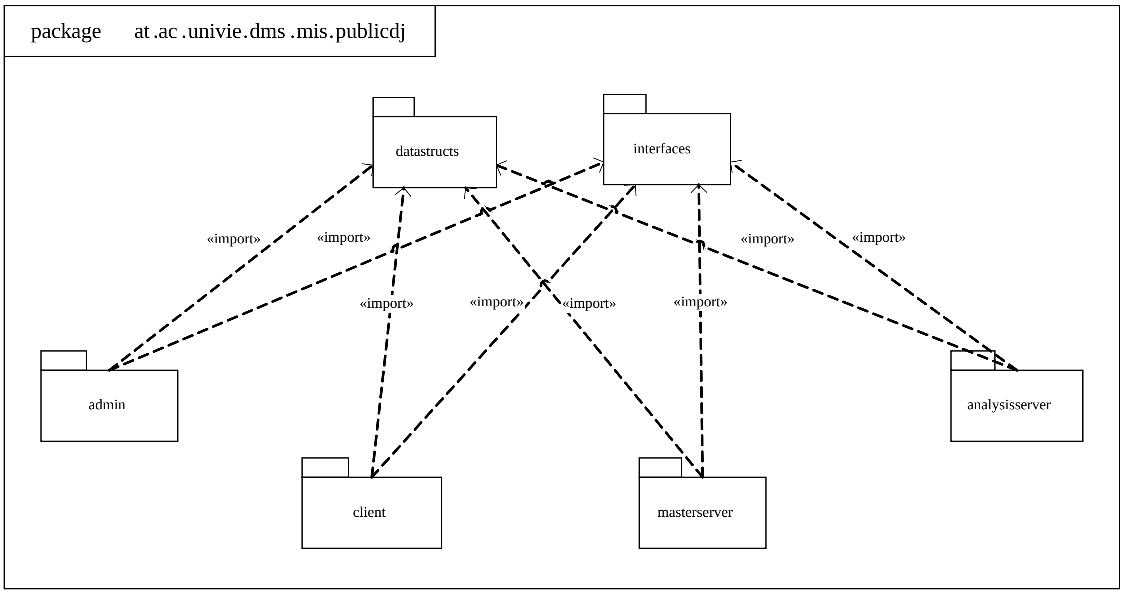


Figure 9.1: Package Diagram Overview

9.1 Master Server

All the other parts of PublicDJ are directly connected to the Master Server via RMI. The Master-Server-application is the most important component of PublicDJ as it implements the complete application logic.

The **Master Server** starts all other threads and is the RMI-Server for all external applications. It accepts file transfers from PublicDJ Clients and Admins and delegates successfully transferred files to the PublicDJ Analysis Servers to have them analyzed.

The Master Server consists of the following threads:

- The **MusicPlayerDaemon** Thread is responsible for playing music files. As round lengths are directly connected to the length of the music tracks this thread also triggers new rounds.
- The **AliveDaemon** Thread checks every second which User/Admin-Clients and Analysis Servers are connected. It cleans up the internal lists (by keeping only the alive devices in the data structures) which are used for 'RMI-multi-casting' by the Master Server.
- The **FinishedTransferMonitor** Thread periodically checks if any music files have been transferred completely. If this is the case this thread triggers the selection of an Analysis Server for each file.

The Master Server can be started or stopped by the Admin-Application. Every round state change needs to be sent to all other connected entities from the Master Server via RMI.

The Master Server itself also has several states (Figure 9.3). 'Default' is the initial state before the game is started. One Admin/several Clients/several Analysis servers can already connect in that state. Depending on the analysis-mode at least one Analysis Server has to connect (except Freeplay mode) because it is needed for all analysis tasks. In case the administrator of PublicDJ is not satisfied with the default options (Metadata Mode / Same Genre) he/she at least has to connect once to configure the options of the game.

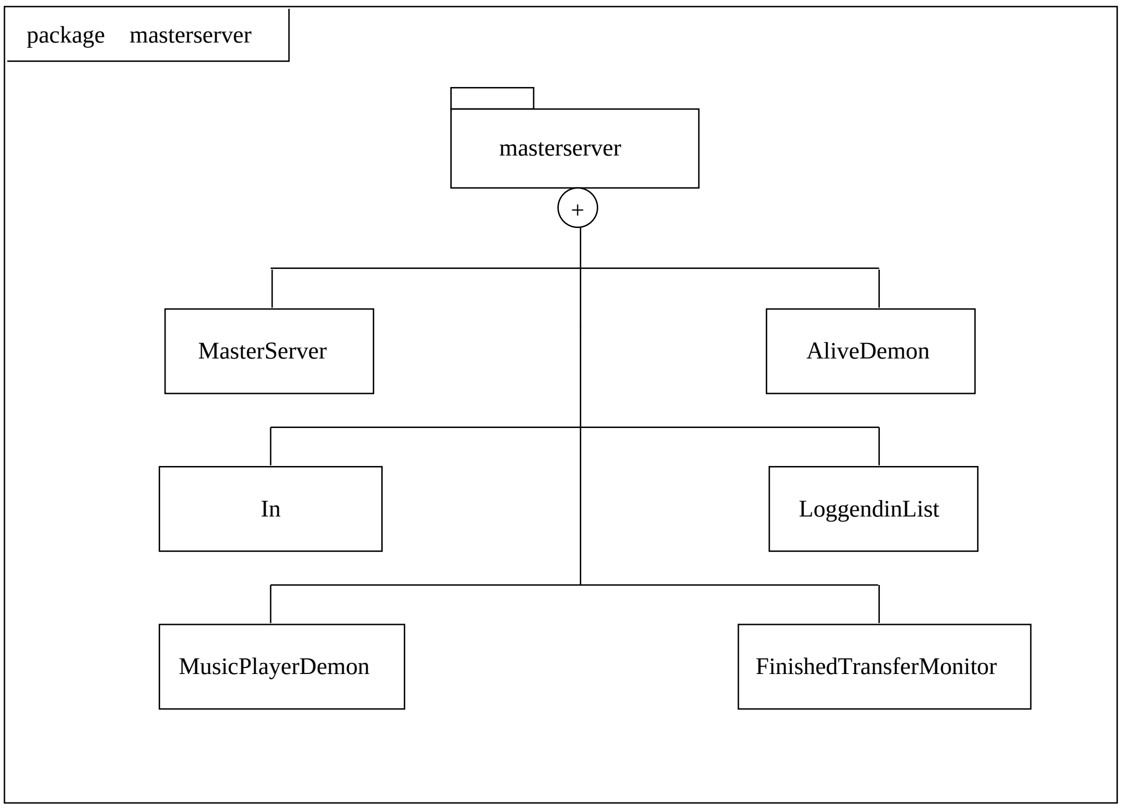


Figure 9.2: Package diagram Master Server

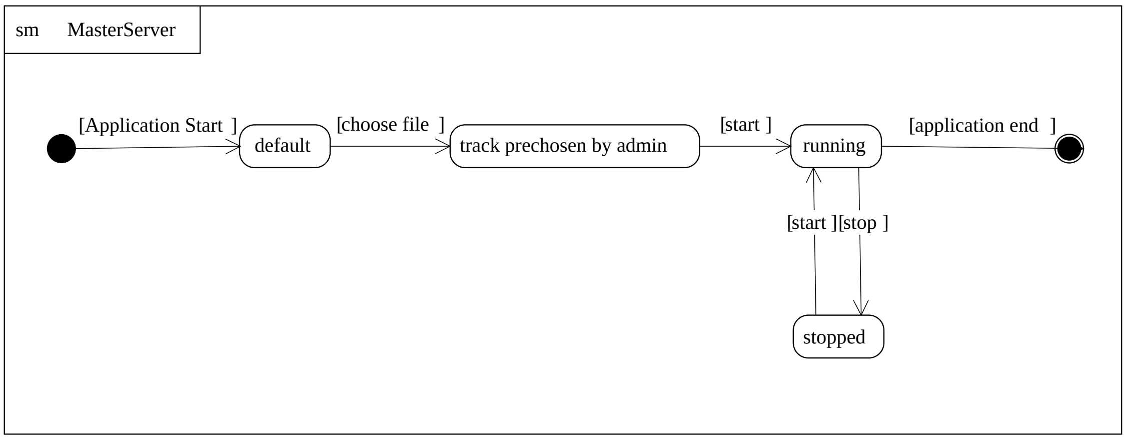


Figure 9.3: State machine diagram Master Server

If an Analysis Server connects it receives the info about the currently chosen mode for analysis. If the mode is modified it is always updated on all Analysis servers. Notice that up to *n* Analysis servers can be connected. If an Analysis server connects it can be directly used by the Master Server for analysis tasks.

If the Admin-Application connects it fetches the current settings and the current status of the Master Server and allows an administrator choosing a music track for the next round as well as starting and stopping the multiplayer game. Only one administrator can be connected at the same time.

If a Client connects it gets added to the LoggedInList. When the multiplayer game is started all Clients receive information about the play-mode, how many other users are connected and the name of the upcoming audio track. Notice that a Client can always connect to the Master Server, the user just has to wait until the next round until he/she can join the game.

After the game is started by the Admin the Master Server performs the following steps each round:

1. It starts to play the music track that was chosen last round
2. It informs all Clients about the start of the new round
3. It accepts $1 - m$ incoming transfers and load balances them
4. It checks if a completely transferred file already was submitted/analyzed before by computing and comparing MD5 sums
    (a) If the file was already analyzed it just uses the stored signature
    (b) otherwise it triggers an analysis at the Analysis Server to calculate the signature of this file
5. It periodically computes an ETA for each transfer, each analysis and cancels transfers/analyses if needed

9.1.1 Threads

MusicPlayerDaemon

This thread plays the music every round. For MP3 decoding it relies on the MP3SPI-package from [13]. The MusicPlayerDaemon can handle Ogg Vorbis Files as well through the VorbisSPI-package from [14]. [1]

[1]Both the MP3SPI and the VorbisSPI packages are Java **S**ervice **P**rovider **I**nterfaces in short SPI. Java does not have native support for OGG Vorbis and MP3 ID3 files so external libraries are needed. SPIs

Every round is bound to the length of the played music track and therefore it also triggers the 'multicast' of the new round via RMI from the Master Server to every connected Admin/User/Analysis Server.

AliveDaemon

This is the Garbage-Collection-Thread of the package. Every second this daemon executes an RMI-Method called alive() on every connected Client, every Admin and every Analysis Server. In case the remote component cannot be reached it is removed from the internal data-structures. This is important because when a new round is triggered by the MusicPlayerDaemon there should be no orphaned entries left or a wrong number of users is displayed on the remote devices temporarily.

FinishedTransferMonitor

All transfers are monitored periodically by this thread. The FinishedTransferMonitor Thread does nothing more than check every TransferData-object in the transferDataVector for finished transfers. If a transfer already finished it triggers a method on the Master Server that starts the selection of a Analysis Server for analyzing the file.

9.1.2 Other important classes

LoggedInList

All logged-in users are stored in a data structure called LoggedInList. This is a class that employs lists for storing the users and that offers several custom queries and operations for easier handling from the main Master-Server-class.

In

This is a helper class for accessing the Master Server's configuration files which are called MasterServerCONFIG.ini, AdminDB.ini and UserDB.ini. It deals with common tasks like

are general interfaces which allow everyone to create an external package for any audio format that is not supported. If a package is properly and completely implemented it is possible to use all methods on the audio file as if it was natively supported. Unfortunately both the VorbisSPI and the MP3SPI were not fully implemented which meant that it was needed to code several things in non standard way. That is the reason why OGG Vorbis files can only be submitted in modes which do not need a conversion of the audiodata.

reading the configuration files line by line and passing them over to the readConfigs()-method for interpretation of the default settings.

9.1.3 Configuration Files

The configuration file for the Master Server is called MasterServerCONFIG.ini. This is the default one:

```
###START###
//file where User/Pass-combinations are stored
USERDATABASE=UserDB.ini

//file where Admin/Pass-combinations are stored
ADMINDATABASE=AdminDB.ini

//Directory where incoming Files are stored (can be relative or absolute)
SHARED_DIRECTORY=FILES

//if you want a server-gui or not. (true/false)
GUI=true

//Prechosen Analysis Algorithm
//(Freeplay - No Analysis/Metadata/Fluctuation_Patterns/Stat_Spectrum_Descr/Rhythm_Histogram)
ANALYSIS_ALGORITHM=Metadata

* relevant only for all modes except Freeplay**
//Metadata modes
//(Same_Genre/Similar_Genre/Year), content-based modes (City block metric/Euclidean metric)
CHOOSE_MODE=Similar_Genre

* relevant only for fluctuation pattern-algorithm **
SKIPPED_LEADINFADEOUT=1
CONTINOUSLY_SKIPPED_6SECS=5

###END###
```

The UserDB.ini and the AdminDB.ini have this layout:

```
###START###
username1:password1
username2:password2
...
usernameN:passwordN
###END###
```

9.2 Analysis Server

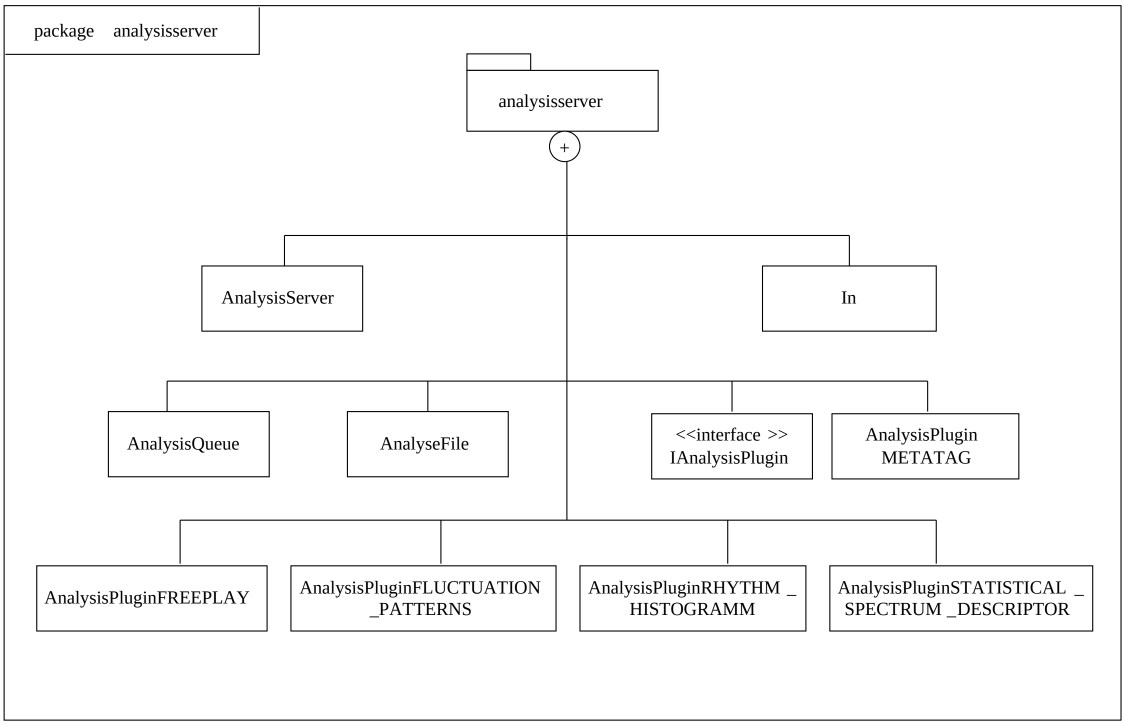

package analysisserver
analysisserver
+
AnalysisServer
In
AnalysisQueue
AnalyseFile
<<interface >> IAnalysisPlugin
AnalysisPlugin METATAG
AnalysisPluginFREEPLAY
AnalysisPluginFLUCTUATION _PATTERNS
AnalysisPluginRHYTHM _ HISTOGRAMM
AnalysisPluginSTATISTICAL _ SPECTRUM _DESCRIPTOR

Figure 9.4: Package diagram - Analysis Server

The Analysis Server's purpose is to unload the Master Server by performing the computation intensive tasks on different computers. This is necessary to guarantee low latency for users while they are interacting with the system and to avoid glitches while playing the music tracks. It is possible to connect $1 - n$ Analysis Servers at the same time and each one is able to perform several computations concurrently. All instances fetch their configuration (except the configuration of how many files can be analyzed concurrently) directly from the Master Server when connecting so it is possible to add some more when there is demand for it. Figure 9.5 illustrates how the Analysis Server logs in and figure 9.5 illustrates how a music file is analyzed. The AnalyseQueue was already presented in chapter 7.2.1 and the different modes were described in chapter 7.1.

Besides the existing modes (Freeplay, Metadata[Similar Genre, Same Genre, Year], Content-based [Rhythm Patterns, Statistical Spectrum Descriptor, Rhythm Histogram]) a plug-in mechanism was implemented. The main idea was to make it as easy as possible to add new plug-ins by defining the interface IAnalysisPlugin. To write a plug-in a developer needs to implement the methods of this Java interface properly and to add support for the mode in the Admin application so it can be configured from there.

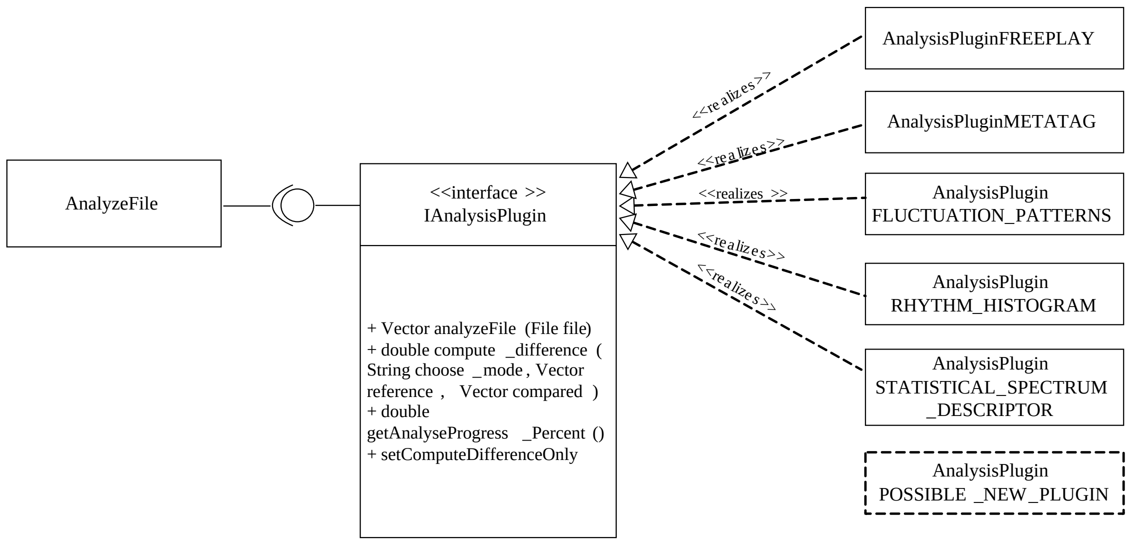

AnalyzeFile
<<interface >>
IAnalysisPlugin
+ Vector analyzeFile (File file)
+ double compute _difference (String choose _mode , Vector reference , Vector compared)
+ double getAnalyseProgress _Percent ()
+ setComputeDifferenceOnly
<<realizes>>
AnalysisPluginFREEPLAY
AnalysisPluginMETATAG
AnalysisPlugin FLUCTUATION_PATTERNS
AnalysisPlugin RHYTHM_HISTOGRAM
AnalysisPlugin STATISTICAL_SPECTRUM _DESCRIPTOR
AnalysisPlugin POSSIBLE _NEW_PLUGIN

Figure 9.5: Analysis Server Plug-in-Concept

9.2.1 Configuration File

The configuration file for the Analysis Server is called AnalysisServerCONFIG.ini. This is the default one:

```
###START###
SHARED_DIRECTORY=FILES
MAX_CONCURRENTLY_ANALYSED_FILES=3
###END###
```

9.3 Client-Application

The 'Client'-application is the program the end-users get to see. The graphical user interface was designed as simple as possible to make it easy and failsafe to use.

The Client consists of two classes (see Figure 9.6):

- 'Client' which implements the RMI-Interface that is used for exchanging information between the Master Server and the mobile device
- 'ClientGui' which implements the graphical user interface

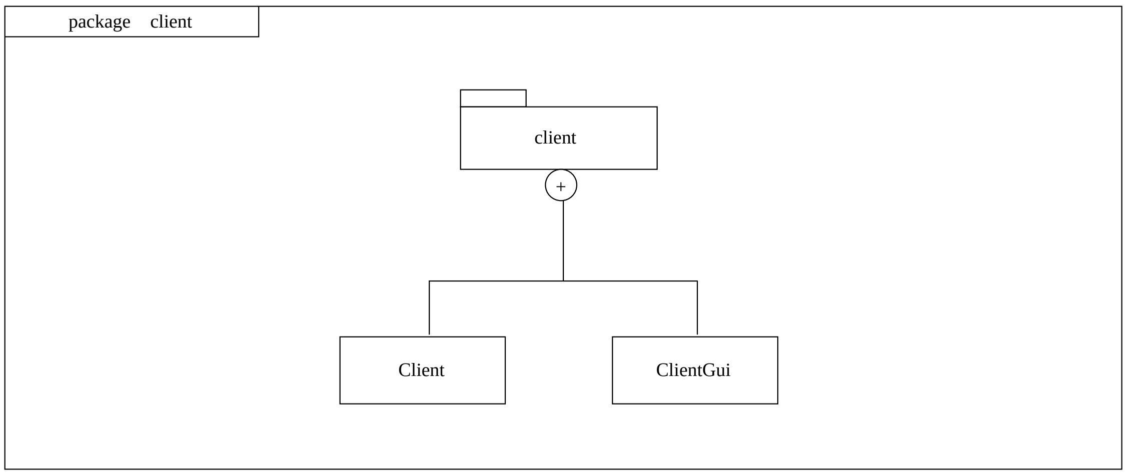


Figure 9.6: Package diagram Client-Application

The state machine diagram(9.7) shows the concept of the '**Client**'-application.

After the user has logged in the Client-Application is in the state **default** in which it waits until a round is started. When the Master Server starts a new round the Client-Application enters the state **newround**. Now the users can either choose and submit a music-track or can just quit. If a music-track is chosen for submission the state **transferFile** is reached - all buttons are now disabled. Now there are 3 cases that might occur.

- If a transfer error occurs the state **transferFile_error** is reached. The user is asked if he/she wants to transfer the file again.
- If the transfer got canceled by the Master Server (probably because it was submitted too late) then the state **fileTransfer_canceled** is entered. The user is informed that

the transfer got canceled and that he/she can try again the following round.

- If the transfer finishes successfully then the state **finishedTransfer** is reached. This of course means that the file was transferred successfully and that analysis will start soon.

Notice that regardless of the state in which the Client-application is, it will always 'recover' to the state **newRound** when a new round is started. Using this mechanism all clients are kept in a consistent state.

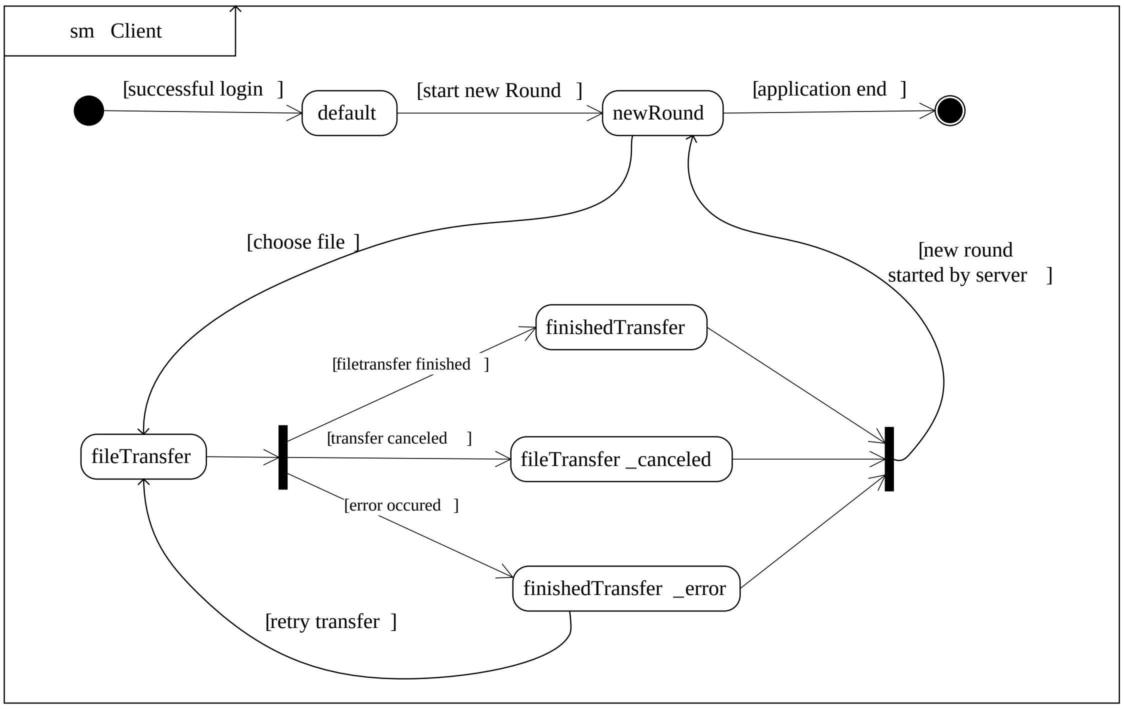
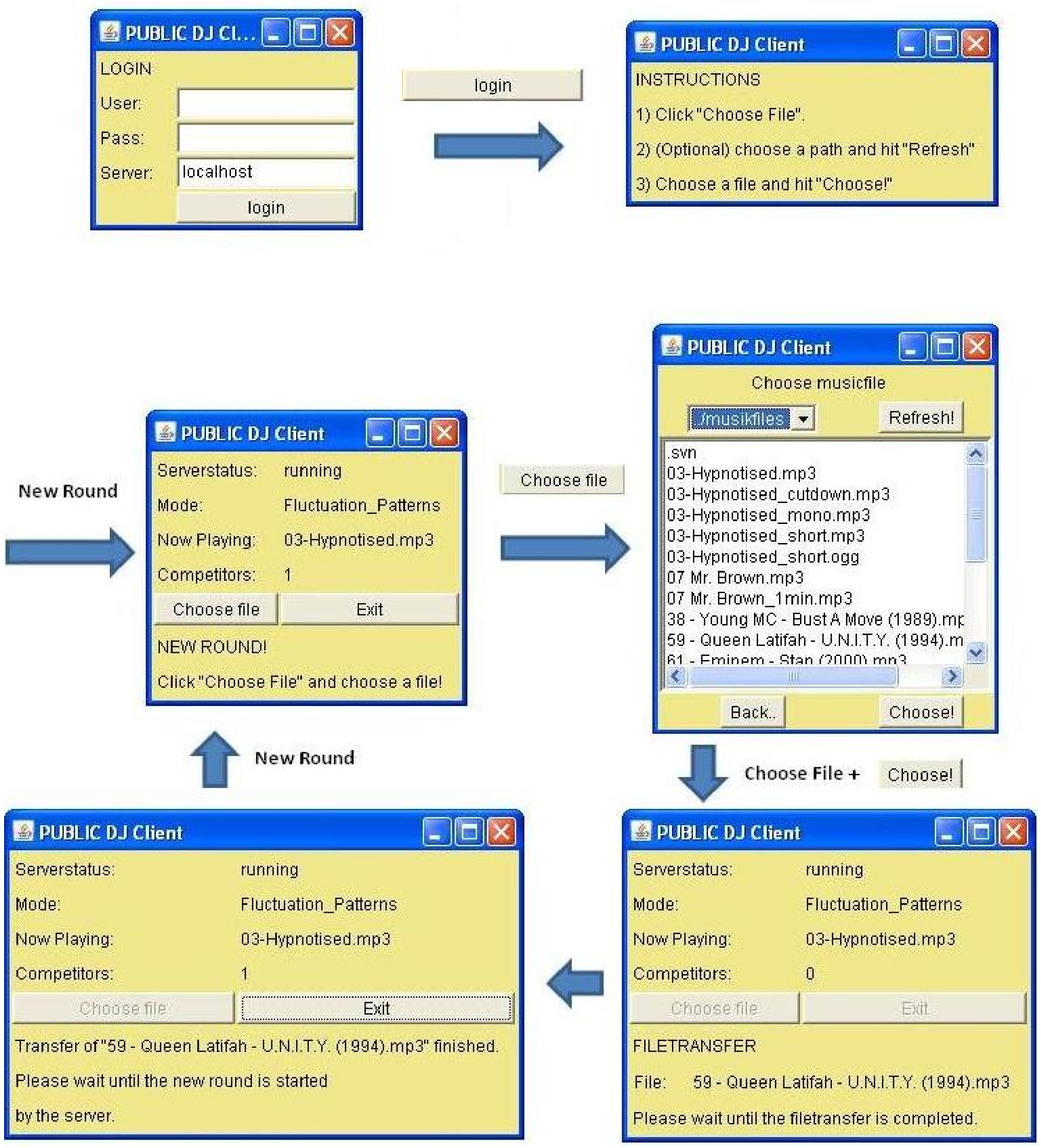


Figure 9.7: State Machine Diagram of the Client-Application

The screenshots (Figure 9.8) give an impression how this looks like for the actual user on the PDA.


PUBLIC DJ Cl...

LOGIN

User:

Pass:

Server: localhost

login

PUBLIC DJ Client

INSTRUCTIONS

1) Click "Choose File".

2) (Optional) choose a path and hit "Refresh"

3) Choose a file and hit "Choose!"

PUBLIC DJ Client

Serverstatus: running

Mode: Fluctuation_Patterns

Now Playing: 03-Hypnotised.mp3

Competitors: 1

Choose file | Exit

NEW ROUND!

Click "Choose File" and choose a file!

PUBLIC DJ Client

Choose musicfile

./musikfiles | Refresh!

```
.svn
03-Hypnotised.mp3
03-Hypnotised_cutdown.mp3
03-Hypnotised_mono.mp3
03-Hypnotised_short.mp3
03-Hypnotised_short.ogg
07 Mr. Brown.mp3
07 Mr. Brown_1min.mp3
38 - Young MC - Bust A Move (1989).mp
59 - Queen Latifah - U.N.I.T.Y. (1994).m
61 - Eminem - Stan (2000).mp3
```

Back.. | Choose!

PUBLIC DJ Client

Serverstatus: running

Mode: Fluctuation_Patterns

Now Playing: 03-Hypnotised.mp3

Competitors: 0

Choose file | Exit

FILETRANSFER

File: 59 - Queen Latifah - U.N.I.T.Y. (1994).mp3

Please wait until the filetransfer is completed.

PUBLIC DJ Client

Serverstatus: running

Mode: Fluctuation_Patterns

Now Playing: 03-Hypnotised.mp3

Competitors: 1

Choose file | Exit

Transfer of "59 - Queen Latifah - U.N.I.T.Y. (1994).mp3" finished.

Please wait until the new round is started

by the server.

Figure 9.8: Screenshot - Client - Login and sample round

9.4 Admin-Application

In contrast to the Client-Application and Master Server the 'Admin' application is stateless. It was a requirement to be able to login and logout anytime, so Master Server/Analysis Server/Clients do not need a connected Admin-Application to work. It is only mandatory that the end-user configures the Master Server and Analysis Server parameters, chooses the first track and starts the round. Then the Admin-Application can be detached and attached whenever needed for stopping the currently played song or for overruling the next song.

The Admin consists of two classes (see Figure 9.9):

- 'Admin' which implements the RMI-Interface for communicating with the Master Server
- 'AdminGui' which implements the graphical user interface

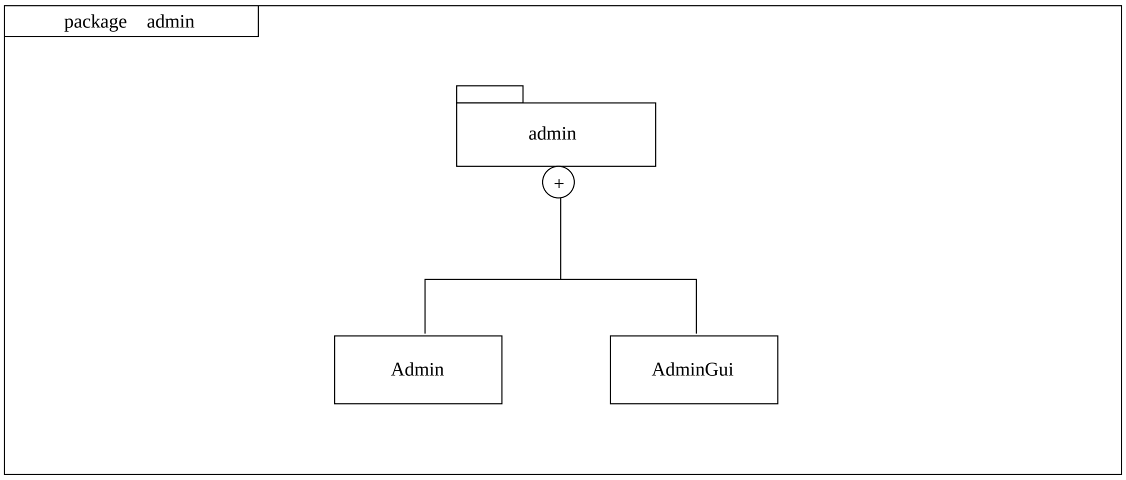


Figure 9.9: Package Diagram - Admin-Application

Figure 9.10 shows some screenshots of the GUI before and after login in to the application.

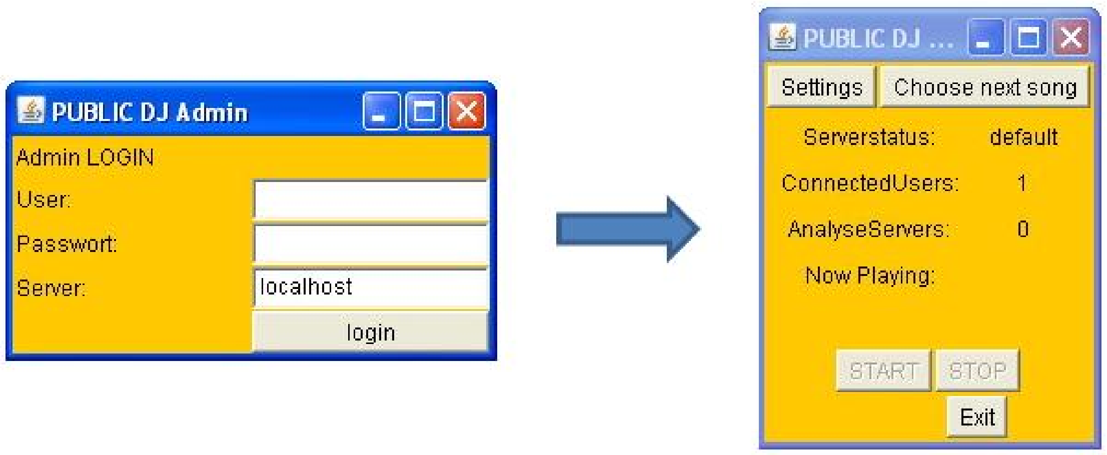
Figure 9.11 shows several screenshots which give an impression of the user interface. As mentioned before there are five different modes.

- Freeplay - No Analysis

  It has no further options. Every submitted track is accepted and queued to the playlist.

- Metadata

  You can define the CHOOSE METHOD which reflects by which constraints the songs have to be chosen. CHOOSE METHOD can be *Same Genre*, *Similar Genre*(currently played song's genre is a substring of the song or vice versa) and *year*.


PUBLIC DJ Admin

Admin LOGIN

User:

Passwort:

Server: localhost

login

PUBLIC DJ ...

Settings | Choose next song

Serverstatus: default

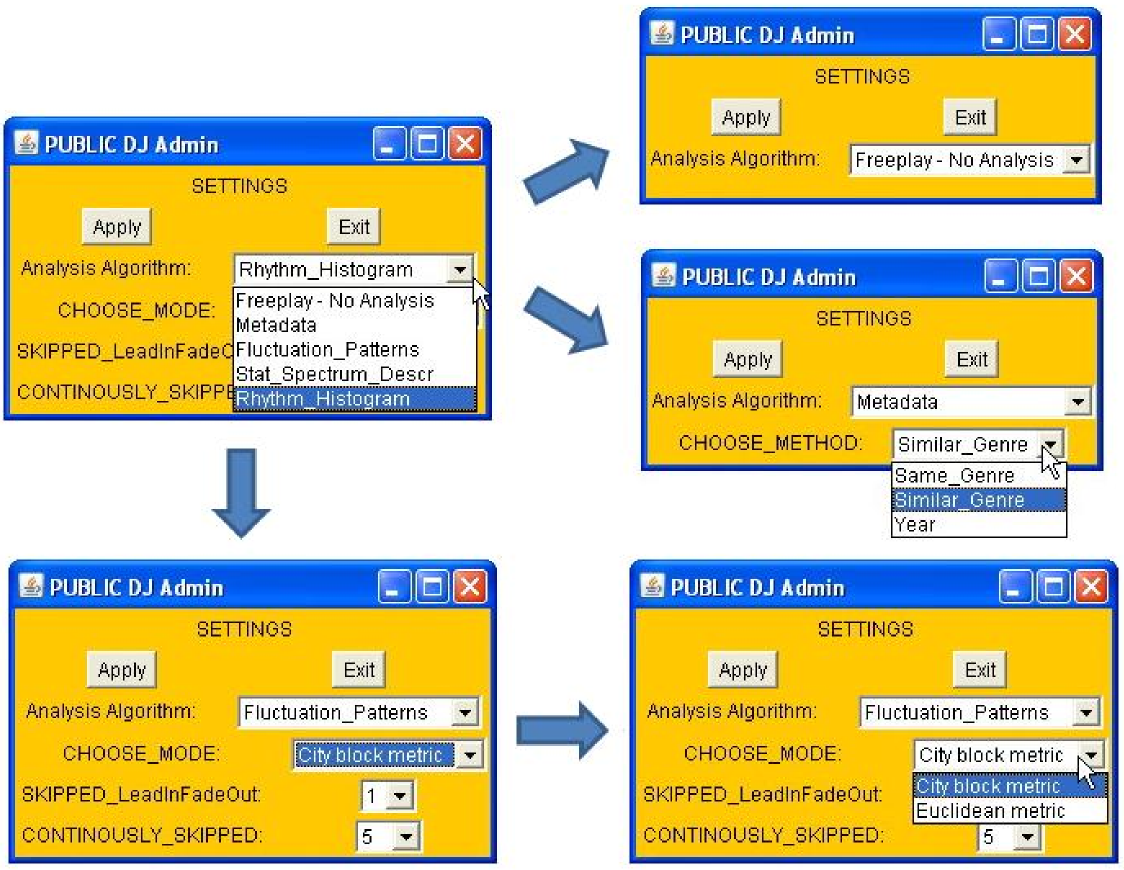
ConnectedUsers: 1

AnalyseServers: 0

Now Playing:

START | STOP | Exit

Figure 9.10: Screenshot - Admin - Login


PUBLIC DJ Admin

SETTINGS

Apply | Exit

Analysis Algorithm: Rhythm_Histogram

CHOOSE_MODE:

SKIPPED_LeadInFadeO

CONTINOUSLY_SKIPPE

Freeplay - No Analysis

Metadata

Fluctuation_Patterns

Stat_Spectrum_Descr

Rhythm_Histogram

PUBLIC DJ Admin

SETTINGS

Apply | Exit

Analysis Algorithm: Freeplay - No Analysis

PUBLIC DJ Admin

SETTINGS

Apply | Exit

Analysis Algorithm: Metadata

CHOOSE_METHOD: Similar_Genre

Same_Genre

Similar_Genre

Year

PUBLIC DJ Admin

SETTINGS

Apply | Exit

Analysis Algorithm: Fluctuation_Patterns

CHOOSE_MODE: City block metric

SKIPPED_LeadInFadeOut: 1

CONTINOUSLY_SKIPPED: 5

PUBLIC DJ Admin

SETTINGS

Apply | Exit

Analysis Algorithm: Fluctuation_Patterns

CHOOSE_MODE: City block metric

City block metric

Euclidean metric

SKIPPED_LeadInFadeOut:

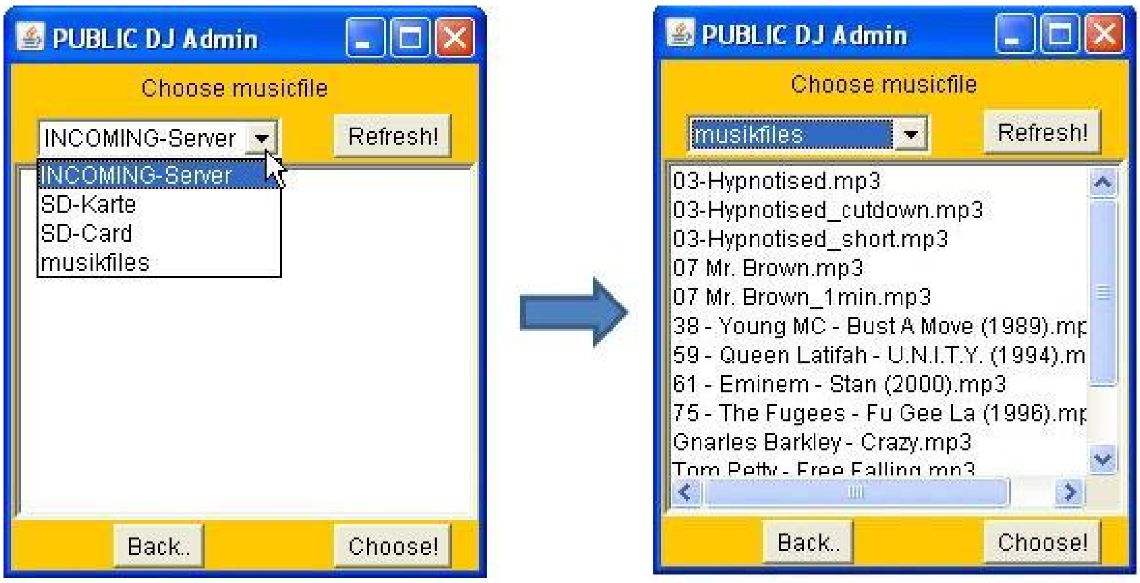
CONTINOUSLY_SKIPPED: 5

Figure 9.11: Screenshot - Admin - Settings

- Fluctuation Patterns
- Statistical Spectrum Descriptors

- Rhythm Histogram

The modes Fluctuation Patterns, Statistical Spectrum Descriptors and Rhythm Histograms all have the same options as they are all based on the same concept.

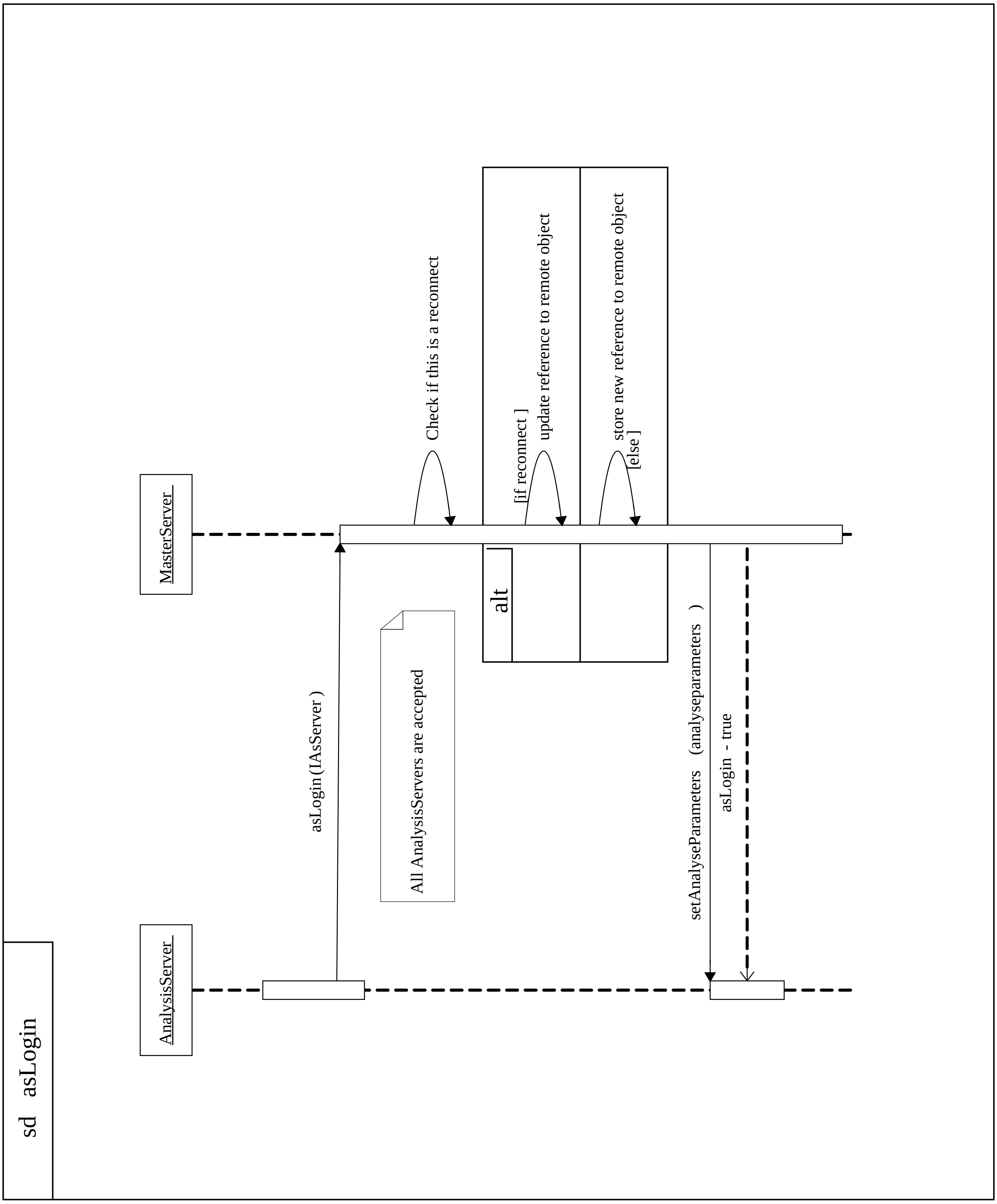


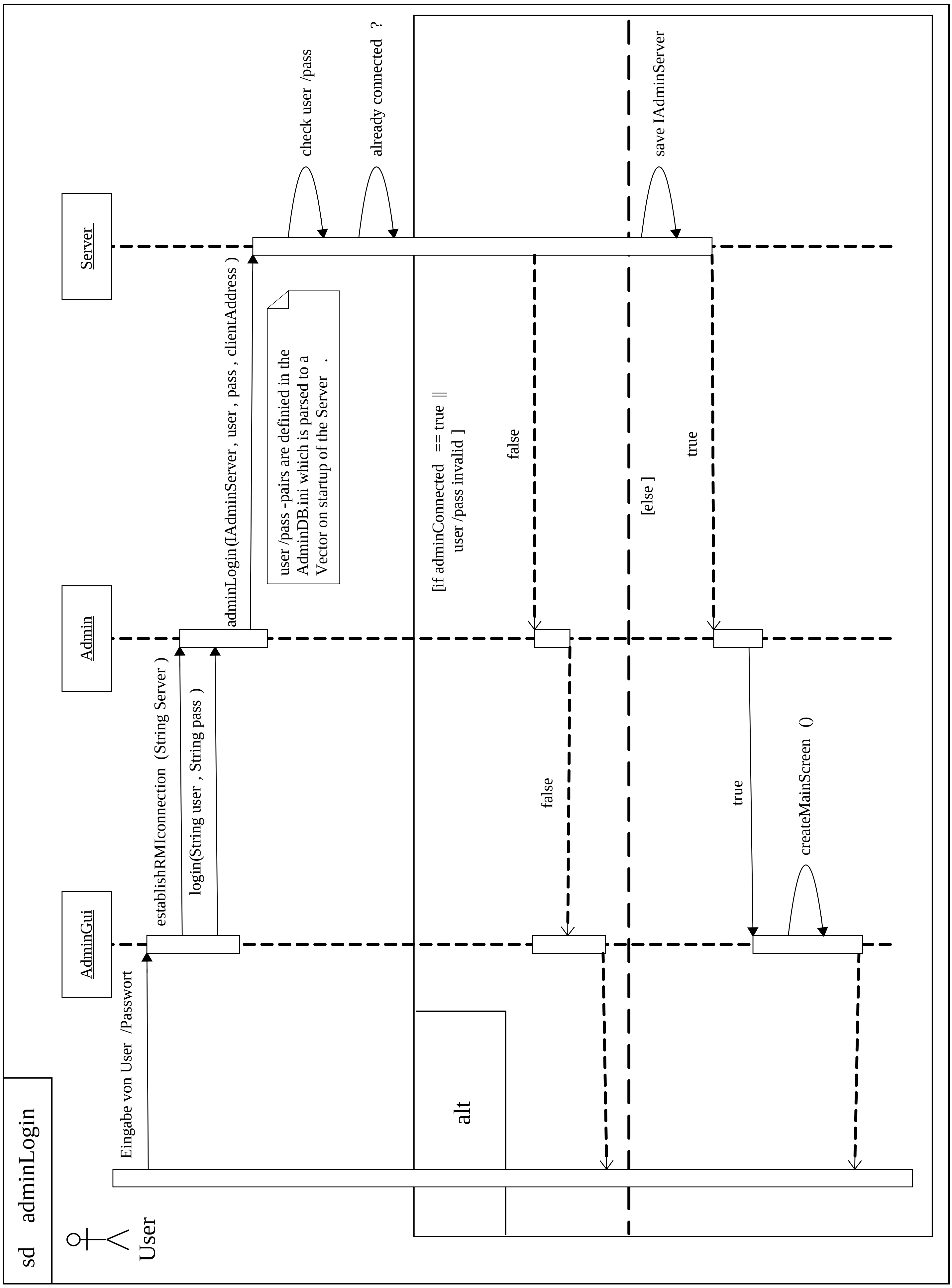
03-Hypnotised.mp3
03-Hypnotised_cutdown.mp3
03-Hypnotised_short.mp3
07 Mr. Brown.mp3
07 Mr. Brown_1min.mp3
38 - Young MC - Bust A Move (1989).mp
59 - Queen Latifah - U.N.I.T.Y. (1994).m
61 - Eminem - Stan (2000).mp3
75 - The Fugees - Fu Gee La (1996).mp
Gnarles Barkley - Crazy.mp3
Tom Petty - Free Falling.mp3

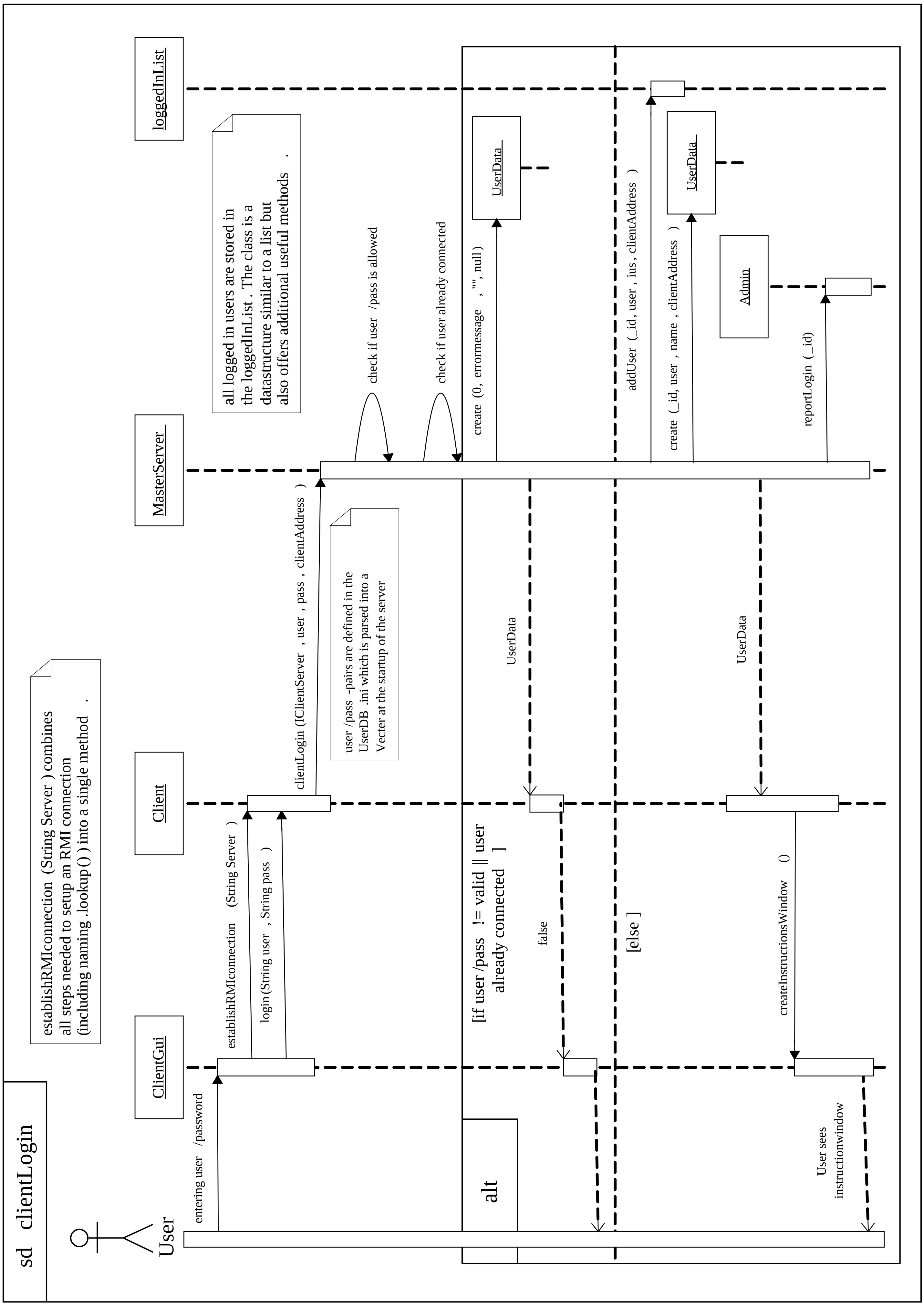
Figure 9.12: Screenshot - Admin - Prechoose File

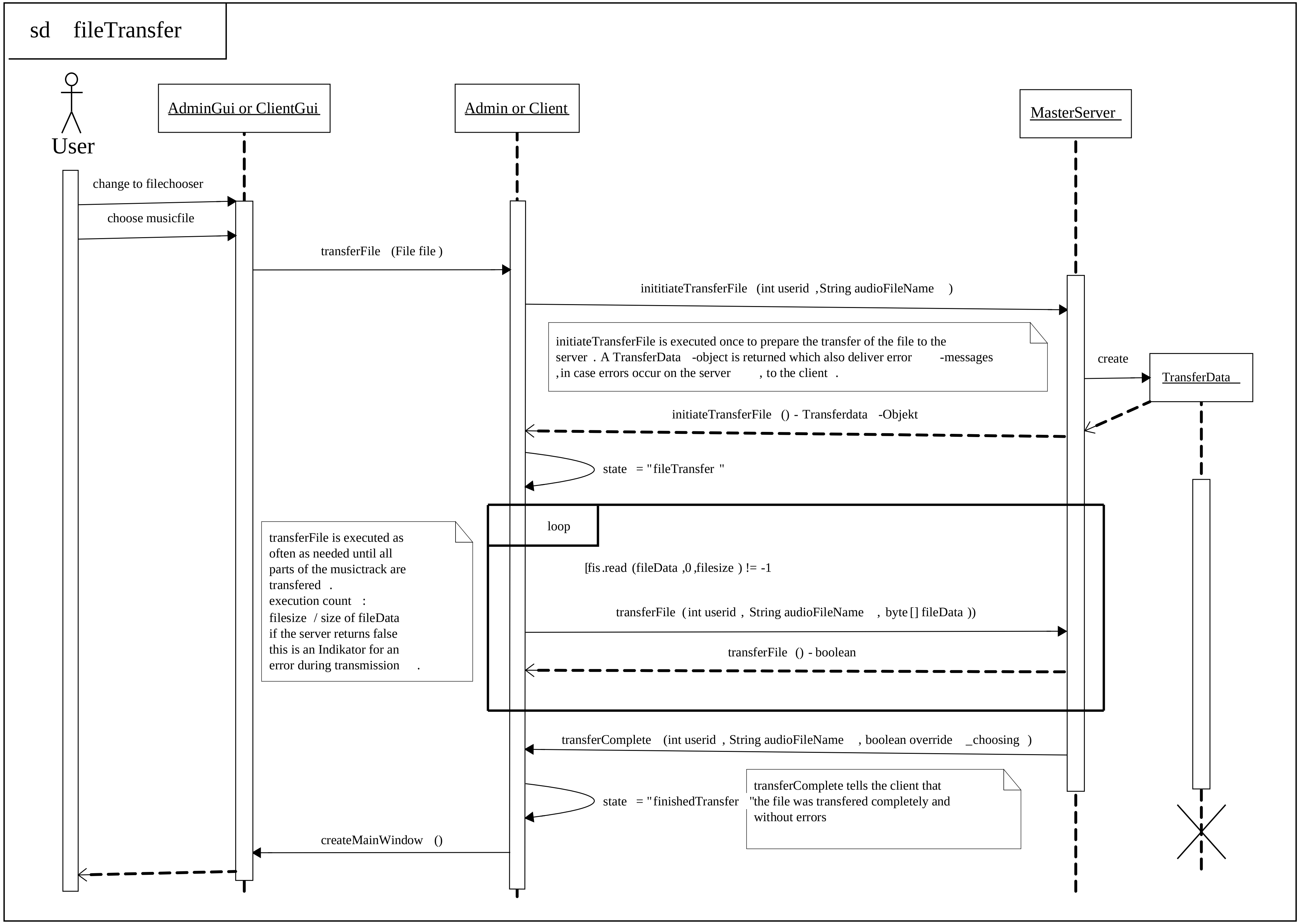
In Figure 9.12 you can see that it is possible to override the next played song by choosing any song from the INCOMING-directory on the Master Server.

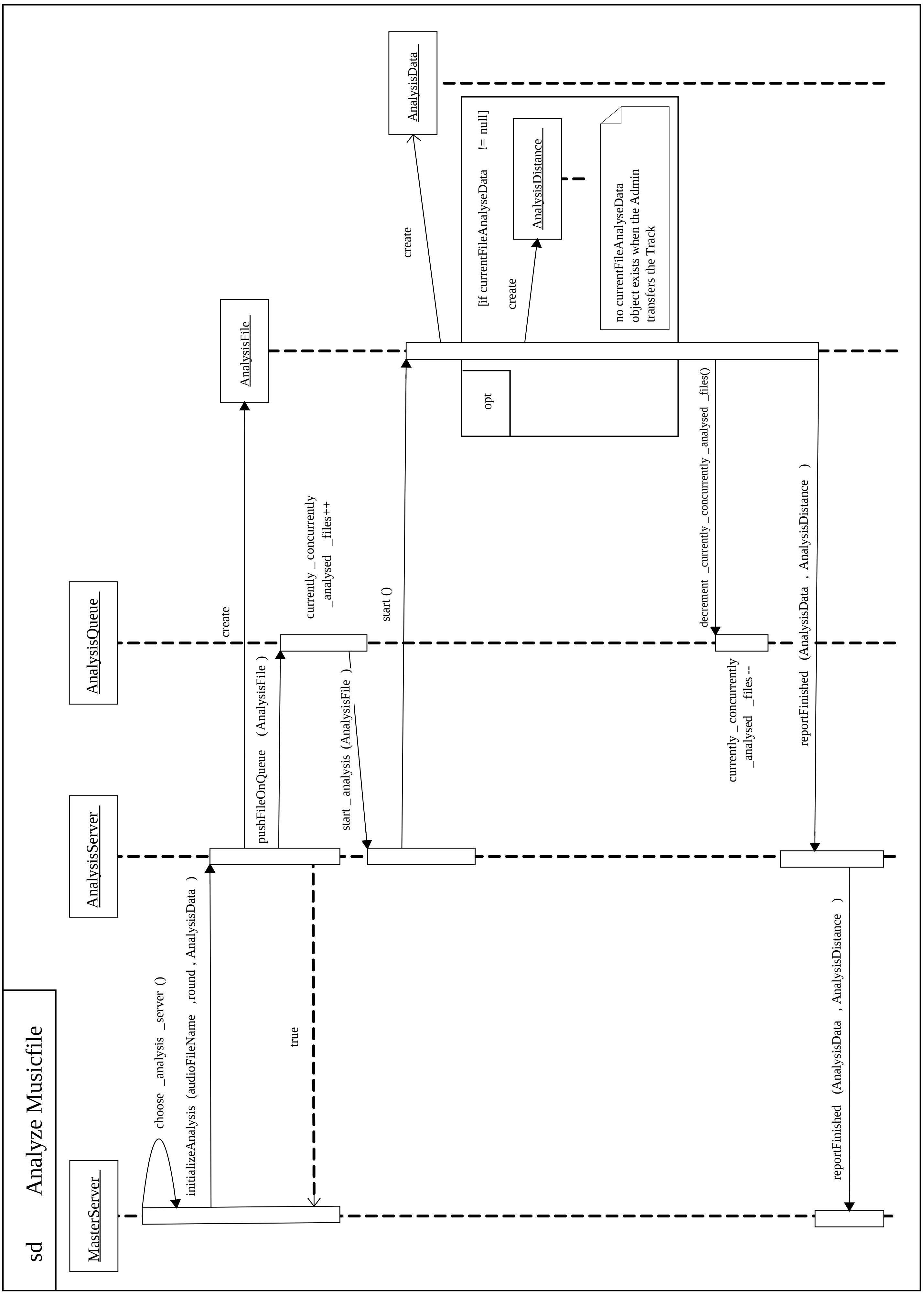
9.5 UML2 Sequence Diagrams for important Usecases


sd asLogin
AnalysisServer
MasterServer
asLogin(IAsServer)
All AnalysisServers are accepted
Check if this is a reconnect
alt
[if reconnect]
update reference to remote object
store new reference to remote object
[else]
setAnalyseParameters (analyseparameters)
asLogin - true


sd adminLogin
User
AdminGui
Admin
Server
Eingabe von User /Passwort
establishRMIconnection (String Server)
login(String user , String pass)
adminLogin(IAdminServer , user , pass , clientAddress)
user /pass -pairs are defined in the AdminDB.ini which is parsed to a Vector on startup of the Server .
check user /pass
already connected ?
alt
[if adminConnected == true || user /pass invalid]
false
false
[else]
save IAdminServer
true
true
createMainScreen ()


sd clientLogin
User
ClientGui
Client
MasterServer
loggedInList
establishRMIconnection (String Server) combines all steps needed to setup an RMI connection (including naming .lookup()) into a single method .
entering user /password
establishRMIconnection (String Server)
login (String user , String pass)
clientLogin (IClientServer , user , pass , clientAddress)
user /pass -pairs are defined in the UserDB .ini which is parsed into a Vecter at the startup of the server
all logged in users are stored in the loggedInList . The class is a datastructure similar to a list but also offers additional useful methods .
check if user /pass is allowed
check if user already connected
alt
[if user /pass != valid || user already connected]
create (0, errormessage , "", null)
UserData
UserData
false
[else]
addUser (_id, user , ius, clientAddress)
create (_id, user , name , clientAddress)
UserData
UserData
Admin
reportLogin (_id)
createInstructionsWindow ()
User sees instructionwindow


sd fileTransfer
User
AdminGui or ClientGui
Admin or Client
MasterServer
TransferData
change to filechooser
choose musicfile
transferFile (File file)
inititiateTransferFile (int userid ,String audioFileName)
initiateTransferFile is executed once to prepare the transfer of the file to the server . A TransferData -object is returned which also deliver error -messages ,in case errors occur on the server , to the client .
create
initiateTransferFile () - Transferdata -Objekt
state = "fileTransfer "
loop
[fis.read (fileData ,0 ,filesize) != -1
transferFile (int userid , String audioFileName , byte [] fileData))
transferFile () - boolean
transferFile is executed as often as needed until all parts of the musictrack are transfered .
execution count :
filesize / size of fileData
if the server returns false this is an Indikator for an error during transmission .
transferComplete (int userid , String audioFileName , boolean override _choosing)
state = "finishedTransfer "
transferComplete tells the client that "the file was transfered completely and without errors
createMainWindow ()


sd Analyze Musicfile
MasterServer
AnalysisServer
AnalysisQueue
AnalysisFile
AnalysisData
choose _analysis _server ()
initializeAnalysis (audioFileName ,round , AnalysisData)
true
create
pushFileOnQueue (AnalysisFile)
currently _concurrently _analysed _files++
start _analysis (AnalysisFile)
start ()
create
opt
[if currentFileAnalyseData != null]
create
AnalysisDistance
no currentFileAnalyseData object exists when the Admin transfers the Track
decrement _currently _concurrently _analysed _files()
currently _concurrently _analysed _files--
reportFinished (AnalysisData , AnalysisDistance)
reportFinished (AnalysisData , AnalysisDistance)

Chapter 10

Summary

PublicDJ is a new approach for collaborative playlist generation with mobile devices. The work presented in this thesis resulted in ([16]) at the Audio Mostly 2007. PublicDJ is implemented as a distributed round based multiplayer game that supports both content-based and metadata-based playlist generation modes. User feedback is given by all players as they submit only the music they would like to hear in that moment. While implementing PublicDJ different mobile devices and different Java-VMs have been experimentally evaluated. It turned out that only a small number of devices currently support WLAN and that only a few of the available Java VMs support RMI.

Currently the problem prevails that WLAN enabled mobile devices are rare. The vision about a crowd that acts as a joint DJ stays and falls with the spreading of WLAN enabled mobile devices that everyone carries for daily usage. The mobile device actually does not even have to be a phone or a PDA as the functionality of many mobile devices like MP3 players and PDAs are merged into the next generation smartphones. The iPhone™(developed and trademarked by Apple Inc.) for example is one of the first smartphones featuring WLAN together with a Java Virtual Machine that supports Java's RMI. The PublicDJ Client and Admin-Applications have been tested successfully on this device. The time for a common use of PublicDJ has not come yet but there are good chances that it might get commonly used in the future.

It is planned to release the sourcecode of all parts of PublicDJ that have been written by me (which excludes third-party source code) under an open source licence. Other developers then can review the sourcecode and improve PublicDJ by adding new functionality. Improving the plugin-mechanism and adding new analysis-modes are the most obvious tasks. If the Client was modified to work in the browser (the Google Web Toolkit[1] probably would be the best way to achieve this) then it would be possible to host a webradio instead of playing it locally.

[1] http://code.google.com/webtoolkit/

Bibliography

[1] Andreja Andric and Goffredo Haus. Automatic playlist generation based on tracking user's listening habits. *Multimedia Tools Appl.*, 29(2):127–151, 2006.

[2] J.J. Aucouturier and F. Pachet. Scaling up music playlist generation. In *Proceedings of IEEE International Conference on Multimedia and Expo (ICME)*. Lausanne, Switzerland, August 2002.

[3] Musiol Bronstein, Semendjajew and Mühlig, editors. *Taschenbuch der Mathematik*. Verlag Harri Deutsch, 1977.

[4] Hugo Fastl and Eberhard Zwicker. *Psychoacoustics: Facts and Models*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[5] Eric Miller Frank Manola. Rdf primer - w3c recommendation. http://www.w3.org/TR/REC-rdf-syntax/, February 2004.

[6] M. Garg, S. Kappes. An experimental study of throughput for udp and voip traffic in ieee 802.11b networks. In *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE*, March 2003.

[7] Object Management Group. Uml® resource page. http://www.uml.org/, January 2008.

[8] J.C. Hauver, D.B. French. Flycasting: using collaborative filtering to generate a playlist for online radio. In *Web Delivering of Music, 2001. Proceedings. First International Conference on*, November 2001.

[9] http://id3.org. Illustration of id3 version 1. http://id3.org/ID3v1?action=AttachFile\&do=get\&target=id3v1_blocks.gif. 7th May 2008.

[10] http://id3.org. Illustration of id3 version 1.1. http://id3.org/ID3v1?action=AttachFile\&do=get\&target=id3v1.1_blocks.gif. 7th May 2008.

[11] http://id3.org. Illustration of id3 version 2. http://id3.org/ID3v2Easy?action=AttachFile\&do=get\&target=id3v2_blocks.gif. 7th May 2008.

[12] ISO/IEC. Final text of dis 8859-1, 8-bit single-byte coded graphic character sets – part 1: Latin alphabet no.1, 1998.

[13] JavaZoom.net-Community. http://www.javazoom.net/mp3spi/mp3spi.html.

[14] JavaZoom.net-Community. http://www.javazoom.net/vorbisspi/vorbisspi.html.

[15] Peter Knees, Tim Pohle, Markus Schedl, and Gerhard Widmer. Combining audio-based similarity with web-based data to accelerate automatic music playlist generation. In *MIR '06: Proceedings of the 8th ACM international workshop on Multimedia information retrieval*, pages 147–154, New York, NY, USA, 2006. ACM.

[16] Stefan Leitich and Markus Toth. Publicdj - music selection in public spaces as multiplayer game. In *Audio Mostly 2007*, Ilmenau, Germany, September 2007.

[17] T. Lidy and A. Rauber. Combined fluctuation features for music genre classification. In *MIREX 2005*, 2005.

[18] Thomas Lidy. Evaluation of new audio features and their utilization in novel music retrieval applications. Master's thesis, Vienna University of Technology, 2006.

[19] Thomas Lidy and Andreas Rauber. Evaluation of feature extractors and psycho-acoustic transformations for music genre classification. In *ISMIR 2005, 6th International Conference on Music Information*, pages 34–41, 2005.

[20] B. Logan and A. Salomon. A music similarity function based on signal analysis. In *Proceedings of the 2001 IEEE International Conference on Multimedia and Expo, ICME 2001, August 22-25, 2001, Tokyo, Japan.*, 2001.

[21] Beth Logan. Content-based playlist generation: Exploratory experiments. In *ISMIR 2002, 3rd International Conference on Music Information*, 2002.

[22] Sun Microsystems. Learn about java technology. http://java.com/en/about/.

[23] Sun Microsystems. Rmi interfaces. http://java.sun.com/developer/onlineTraining/rmi/images/RMIInterfaces.gif. 3rd June 2008.

[24] Sun Microsystems. Simplified illustration of rmi. http://java.sun.com/developer/onlineTraining/rmi/images/RMIArchitectureLayers_01.gif. 3rd June 2008.

[25] Martin Mutschler. Mp3ext. http://www.mutschler.de/mp3ext/.

[26] C. Newell. Id3v2 chapter frame addendum. http://id3.org/id3v2-chapters-1.0, December 2005.

[27] M. Nilsson. Id3 tag version 2.3.0. http://www.id3.org/id3v2.3.0, February 1999.

[28] Martin Nilsson. Id3 tag version 2. http://www.id3.org/d3v2.3.0, March 1998.

[29] Martin Nilsson. Id3 tag version 2.4.0 - main structure. http://www.id3.org/id3v2.4.0-structure, November 2000.

[30] Dan O'Neill. Id3v2 made easy. http://id3.org/ID3v2Easy. last time edited on the 17th December 2006.

[31] Dan O'Neill. What is id3 (v1)? http://id3.org/ID3v1. last time edited on the 29th October 2006.

[32] Elias Pampalk. Islands of music: Analysis, organisation, and visualisation of music archives. Master‘s thesis, Vienna University of Technology, 2001.

[33] Elias Pampalk. *Computational Models of Music Similarity and their Application in Music Information Retrieval*. Ph.d. dissertation, Vienna University of Technology, 2006.

[34] Elias Pampalk and Martin Gasser. An implementation of a simple playlist generator based on audio similarity measures and user feedback. In *ISMIR*, pages 389–390, 2006.

[35] Elias Pampalk, Tim Pohle, and Gerhard Widmer. Dynamic playlist generation based on skipping behavior. In *ISMIR 2005, 6th International Conference on Music Information*, pages 634–637, 2005.

[36] Elias Pampalk, Andreas Rauber, and Dieter Merkl. Content-based organization and visualization of music archives. In *MULTIMEDIA '02: Proceedings of the tenth ACM international conference on Multimedia*, pages 570–579, New York, NY, USA, 2002. ACM.

[37] Steffen Pauws and Berry Eggen. Pats: Realization and user evaluation of an automatic playlist generator. In *ISMIR 2002, 3rd International Conference on Music Information Retrieval, 2002*, 2002.

[38] John C. Platt, Christopher J. C. Burges, Steven Swenson, Christopher Weare, and Alice Zheng. Learning a gaussian process prior for automatically generating music playlists. In *Advances in Neural Information Processing Systems 14*, pages 1425–1432, 2002.

[39] C. Herley R. Ragno, C. J. C. Burges. Inferring similarity between music objects with application to playlist generation. In *Proceedings of the 7th ACM SIGMM international workshop on Multimedia information retrieval*, 2005.

[40] Fabio Vignoli Rob van Gulik. Visual playlist generation on the artist map. In *ISMIR 2005, 6th International Conference on Music Information Retrieval*, 2005.

[41] Huub van de Wetering Rob van Gulik, Fabio Vignoli. Mapping music in the palm of your hand, explore and discover your collection. In *ISMIR 2004, 5th International Conference on Music Information Retrieval,*, 2004.

[42] Markus Scherer. Utf-16 for processing. `http://www.unicode.org/notes/tn12/`, January 2004.

[43] Gerhard Widmer Tim Pohle, Elias Pampalk. Generating similarity-based playlists using traveling salesman algorithms. In *DAFx05*, 2005.

[44] Alexander L. Wijesinha, Yeong tae Song, Mahesh Krishnan, Vijita Mathur, Jin Ahn, and Vijay Shyamasundar. Throughput measurement for udp traffic in an ieee 802.11g wlan. In *SNPD-SAWN '05: Proceedings of the Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and*

Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Networks, pages 220–225, Washington, DC, USA, 2005. IEEE Computer Society.

[45] xiph.org. Vorbis.com faq. `http://www.vorbis.com/faq/`, October 2003.