A touch-based music player interface optimized for tablet-sized devices

Bachelor thesis, Yannick Stucki, ETH Zurich

June 2011

Preface

I would like to thank my advisors Samuel Welten and Michael Kuhn for all the vivid discussions about music players and interfaces, their help and their input. I would also like to thank Professor Roger Wattenhofer for providing me with a Motorola Xoom. Without the Xoom I could not have programmed *Jukefox for Tablets*, which will hopefully soon appear in the Android market. I also hope that *Jukefox for Tablets* will have many female users, even if the user I am talking about in this thesis will always be male.

Contents

1 Introduction **1**

2 The history of mobile music players interfaces **3**

3 Tablet interfaces **8**
- 3.1 Landscape use 8
- 3.2 Thumb navigation 8
  - 3.2.1 The tablet is placed 8
  - 3.2.2 The tablet is held in one hand, while interactions are done with the other hand 9
  - 3.2.3 The tablet is held with both hands, while interaction is done with both thumbs 9
- 3.3 Avoid full screen transitions 10

4 Design principles and goal **11**
- 4.1 The goal of decoupling the playback experience from the interface experience 12
  - 4.1.1 Finding specific songs 13
  - 4.1.2 Finding non-specific songs 13
  - 4.1.3 The user should be able to plan ahead 14
  - 4.1.4 The user should be able to act spontaneously 14
  - 4.1.5 Never interrupt the playing song, unless the user *really* wants to 14
  - 4.1.6 Avoid starting the playback of music the user does not want. 15
- 4.2 Explicit principles for this project 15
  - 4.2.1 Keep in mind that there is no intuitive interface 15
  - 4.2.2 The experience should be visually attractive and fun to use 16

5 Design choices **18**
- 5.1 The play queue 18
  - 5.1.1 The play controls should always be visible 19
  - 5.1.2 Do not implement a *now playing screen* 19
  - 5.1.3 Upcoming songs are more important than already played songs 21
  - 5.1.4 Gestures allow for precise and spontaneous control 21
  - 5.1.5 The clear button only affects the upcoming songs 22
  - 5.1.6 One play mode is enough 23

5.1.7 Auto refill for smart shuffle 25
5.2 Navigation within the music library and selection of songs 26
5.2.1 Supporting several selection views 26
5.2.2 Means of achieving consistency across all selection views . 27
5.2.3 Put the play queue on the right side 30
5.2.4 Sacrifice genres (for now) 31
5.3 Explore 31
5.3.1 Allow for hierarchical navigation 32
5.3.2 Encourage flat and linked navigation 32
5.4 Map 35
5.5 Search 36
5.6 Playlist 38

6 Conclusion and outlook **40**

List of Figures

2.1 The first Discman, the Sony D50. 4
2.2 Two models of the Apple iPod 4
2.3 The *now playing screen* of the Apple iPhone 5
2.4 The android player ^{3} (cubed) with 3D animations for albums (left) and the Map of Music from *Jukefox* (right). 6
2.5 The Music Rainbow. 6
2.6 The *now playing screen* on the default player for android on tablets. 7

5.1 The *Jukefox for Tablets* interface displaying all albums. 18
5.2 Rearranging through dragging (left) and removing through swiping (right) 22
5.3 An album being dragged into the queue. 27
5.4 An album's overlay. 28
5.5 An album's overlay with selected songs. 30
5.6 An overlay before sliding (left) and after sliding (right). 31
5.7 An artist's exploration view. On top (left); scrolled down (right). 34
5.8 A tag's exploration view. 35
5.9 A list to select an artist to explore. 36
5.10 The map view. 37
5.11 The map while the user is drawing a region with his finger. . . . 38
5.12 The search view. 39

Abstract

Mobile music players have changed drastically over the last couple of years and there is vivid experimentation going on in the field of music retrieval interfaces. However, the rest of the music player interface, mainly the part responsible for playback controls, has stayed largely the same as the one the Discman introduced in 1984.

To analyze playback interfaces, we identify a trade-off between what we call *playback* and *interface experience*, where the user can enhance one at cost of the other. We define a possible goal of music playback interfaces as *decoupling* the playback and interface experience, such that their negative coupling becomes weaker than in previous music playback interfaces. Such a weaker coupling improves the overall experience, as the trade-off becomes less severe.

Furthermore, we introduce our music player *Jukefox for Tablets* for the Android Honeycomb platform and explain how we made design choices based on our previously defined goal. We compare interface elements from *Jukefox for Tablets* to interface elements found on other players and show how we achieve a higher decoupling between the playback and the interface experience.

Chapter 1

Introduction

Like many other habits, the way we listen to music has changed drastically over the last few decades due to new technologies. One part which evolved probably more than any other is the field of mobile music players. While the number of songs that could be listened to on a portable music player used to be limited by the amount of cassettes or CDs a user could carry, a user can now carry around thousands of songs stored on a device or even stream from a collection of millions of songs using the Internet.

Traditionally, songs have been accessed by hierarchically structured meta data such as artists, album and song names. In the field of music retrieval interfaces, the question is how to design interfaces which can go beyond this traditional approach such that we can scale music retrieval to the size of today's music collections.

While the question of how to make a good music retrieval interface is of great importance for any music player, its central functionality still remains that of playing music. Little research has been going into the question of what a good interface for a player should look like, besides the interface for retrieving the music. In the contrary, the part of the interface commonly called the *now playing screen* is present in almost any player and added by developers with barely a second thought.

Our contribution is one possible way to define the question of how to make a good music playback interface, i.e. what property a good music playback interface should optimize. Furthermore, we implemented a music player interface for tablets running on Android 3.0 Honeycomb version or later. It is aimed at optimizing music playback as introduced in our definition.

In the next chapter we look at how mobile music players interfaces evolved, how they changed and in what aspects they have remained the same.

In chapter 3 we discuss a few aspects which shape and influence the tablet interface experience and make designing for the tablet different from other platforms such as the desktop or the mobile phone.

In chapter 4 we create the foundation of our application. We define our goal of music playback interfaces and its consequences. Furthermore, we explicitly highlight some principles which we deemed to be especially important for the realization of a music player interface on a tablet.

In chapter 5 we present our music player for tablets running on the Android platform. It is our attempt to achieve optimized music playback as defined in chapter 5. Furthermore, we highlight a set of design choices which lead to create the application the way it is. For each design choice we outline the rational behind the decision, while also explaining how it helps us optimizing our player, or even is necessary, to fulfill the requirements and adhere to the principles specified in 5.

In chapter 6 we conclude by summarizing our work and giving an outlook to what the next steps of our project will be.

Chapter 2

The history of mobile music players interfaces

In this chapter we do not try to give a complete history of mobile music players and their interfaces, but highlight a few important historic developments and analyze how two parts of the interface, the one for music retrieval and the one for music playback, have changed over time.

One of the first portable audio players was Sony's Walkman in 1979 [3]. For casette players such as the Walkman, the music retrieval interface was a user's casette collection and a whole casette is the smallest quantity that could be retrieved. The playback interface featured the play and pause buttons we still have today, but otherwise such players had only rewind and fast forward options to linearly seek through the casette. There is not really a distiction between playback and music retrieval interface, as seeking can be viewed at as belonging to both.

In 1984, Sony introduced the Discman which replaced cassettes by Compact Discs [13]. The CDs used in the Discman allowed to jump to a specific track instead of having to seek like when using cassettes. It is clear that this greatly improved the interface. In figure 2.1 we see the picture of the first Discman, the Sony D50 [14]. It already featured controls such as play/pause, previous/next track and displayed information such as track number and playback time.

As with the walkman, finding the right data carrier was part of the music retrieval interface.

The next step in the evolution of mobile music players where digital audio players (DAP), which had local storage that could be loaded with the users songs. Although experimented with before, the digital audio players started getting into the mass market at around 1998 [12].

DAPs changed the music retrieval interface dramatically, as the user now did not have to carry around external storage anymore. The interface for retrieving music was quite trivial at first, since the number of songs on such players was very limited at the beginning. The music playback interface remained very similar at first, since the DAPs initially only had tiny screens which could display a track name or a few letters at best. The most successful DAP is undoubtedly the iPod, which was first introduced by Apple in 2001 [16]. Although there is an enoumerous amount of digital audio players, we will analyze the iPod as

Figure 2.1: The first Discman, the Sony D50.

an example, not only due to its popularity, but also because it shaped and influenced the industry for years and still does.

The iPod (see figure 2.2) introduced distinct interfaces for playback (left) and music retrieval (rigth). The playback interface was the so called *now playing screen*. In this interface, the user basically had the same controls as when controlling a Discman: play/pause, previous/next track, change of volume, linear seeking within the song or changing the play mode to shuffle or repeat (in later versions). Only the ability to rate a song was not present on the Discman. Furthermore, the iPod's display allowed to show an album art (which would be on the CD case if one were using a Discman) and the name of the title, album, and song.

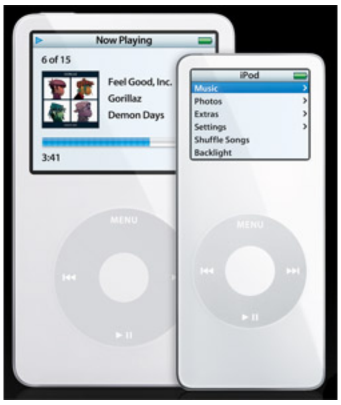


Figure 2.2: Two models of the Apple iPod

The retrieval mode offered a choice of hierarchical lists based on meta data of the songs. For example, a user could find a song by navigation from the list of artists to albums and then to the specific song.

When a song is clicked it is instantly played. If a user is in a special play mode called the *on-the-go* playlist, limited enqueing functionality, i.e. the functionality to put a song into a play queue to be played later, can be achieved. However this functionality is not a first class citizen, as it is not in the main part of the interface.

When the iPhone was introduced in 2007 by Apple, it was said to include an iPod and its music application has been called iPod ever since [15]. Clearly, Apple wanted to leverage the familiarity with iPods among millions of users. Thus, the music retrieval still operates the same, except that users now use a touch based list instead of a click wheel. The playback interface also remained the same, except that the album art got scaled up so that there was no more space to be used than before on the iPod with a smaller screen. Of course there where a few details that changed, but the concept clearly stayed the same as can be seen in figure 2.3.

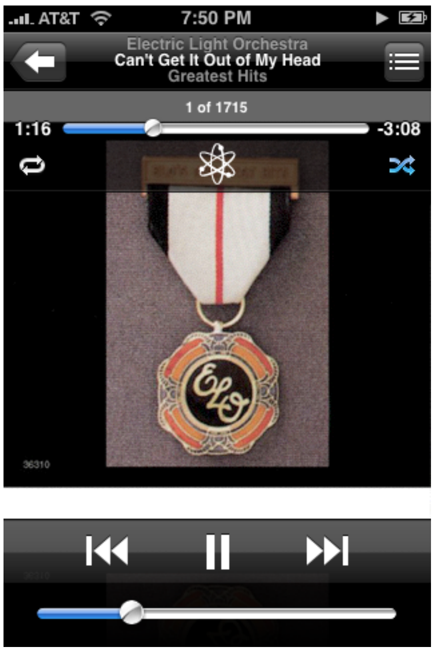


Figure 2.3: The *now playing screen* of the Apple iPhone

Since the iPod and iPhone, the vast majority of music players use an interface with the same two parts: A hierarchical meta data based music retrieval mode and a playback interface with the power of the Discman. Having seen this interface everywhere, users probably expect this and no major phone manufacturer dared to ship a radically different player on their phone.

The differentiation started happening on another level: design and style. Ever more special ways to animate albums in 3D found their way into music players, a good example being [3] (cubed) seen in figure 2.4(a) on the left, an android music player with a cube of albums.

However, not all progress halted. In the last couple of years researchers started looking into new ways of creating music retrieval interfaces, since they

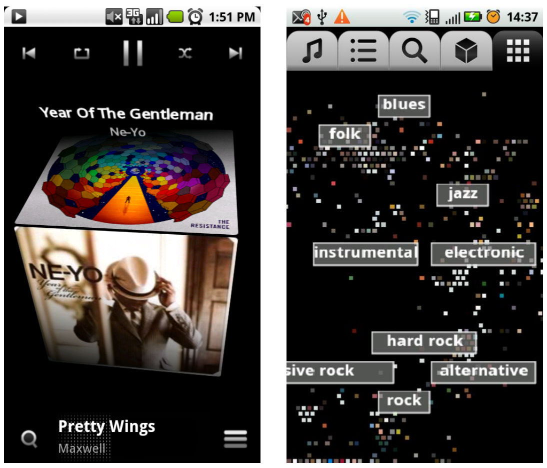


Figure 2.4: The android player [3] (cubed) with 3D animations for albums (left) and the Map of Music from *Jukefox* (right).

realized that the hierarchical meta data oriented approaches would not scale. Examples of research into new kind of interfaces include the Music Rainbow[7] as seen in figure 2 or the Map of Music from *Jukefox* seen in figure 2.4(a) on the right, the player upon which our tablet interface will be based[6].

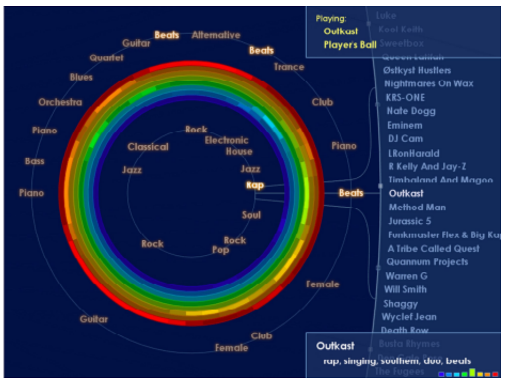


Figure 2.5: The Music Rainbow.

The latest generation of mobile devices are tablets. While tablets have existed long before the iPad or the Motorola Xoom, the new generation is the first generation of tablets to be mainly inspired by phones. This is why the music players on those devices are also heavily inspired by the music players on phone if not simply the same. The default music player on Android for example differs only by replacing the list of albums by a grid of albums with album art, its *now playing screen* having only the usual functionality as seen in figure 2. Also Apple's iPad interface is quite similar to the one found on the iPhone, having only a few tweaks borrowed from Apple's desktop player iTunes, but none of which would break already discussed paradigm.

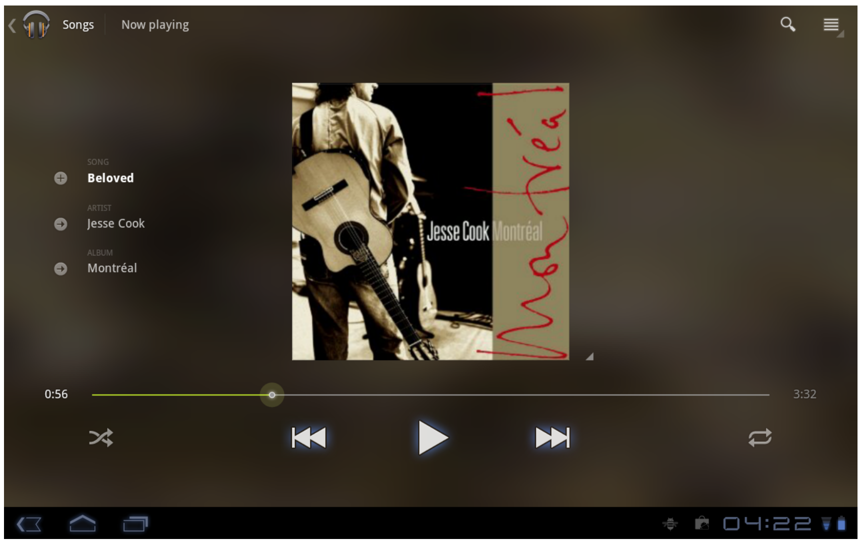


Figure 2.6: The *now playing screen* on the default player for android on tablets.

While this chapter does not cover anything and did not touch interfaces of non-mobile music players at all, it tries to explain how it is possible that the latest popular technology gadgets of 2011 have a music playback interface which is largely equivalent to that of the Discman in 1984.

Chapter 3

Tablet interfaces

In this section we want to name and explain a few of the tablet's characteristics which have to be kept in mind while designing an interface for such a device.

3.1 Landscape use

While phones usually have portrait as their default orientation, tablets are commonly held in landscape orientation. The Android developer guide says that landscape is the default orientation[1]. Although not everyone seems to agree with this view[2], we found that landscape was better suited than portrait in our particular case, especially when displaying two views next to each other horizontally with a the device having a widescreen aspect ratio. Although not all future tablets might share this ratio, landscape will still work well and thus our player is optimized for landscape, although it of course is also functional in portrait.

3.2 Thumb navigation

With mobile devices such as phones and tablets, different placements and holding positions influence the positions where the user is able to comfortably touch the tablet and in what manner. Furthermore, those positions also influence which part of the screen might be hidden because of the user's fingers. Three common ways to hold or position a tablet come in mind, while all of them can be happen in landscape or portrait.

3.2.1 The tablet is placed

If the tablet is placed on a table, a special tablet holder or a person's lap we usually do not have any problems with the interface. The person will probably be able to use both hands freely, but might be forced to use one hand for supporting the tablet if the ground is not stable enough. With at least one free hand, the user is able to click and drag & drop any item on the interface easily while most probably using the index finger. Multitouch operation such as pinch to zoom are no problem either by additionally using the thumb. Therefore, this mode of operation does not restrict the positioning of important UI elements or our mode of touchscreen interaction.

3.2.2 The tablet is held in one hand, while interactions are done with the other hand

In this mode of navigation the user is almost as flexible as when the tablet is placed (except maybe for typing, but that is not our problem to solve). We should note however, that tablets have varying weights and balance points and since users will mostly use their stronger hand for navigation, it may not be comfortable to remain in this mode for too long depending on the actual device. Being largely equivalent to the previous mode, this mode does also not restrict the positioning of important UI elements.

3.2.3 The tablet is held with both hands, while interaction is done with both thumbs

This is probably the most common mode of navigation, being much more comfortable than holding the tablet with one hand only. It is also the mode of navigation we have to be the most careful about. First of all, multitouch is not possible in this mode (except maybe if the whole screen is one mulittouch surface and the both thumbs can be used). Thus, one should provide additional single touch alternatives for any view using multitouch.

Secondly, if operated in landscape, the area in the middle is not reachable or uncomfortable to reach (depending on the size of the tablet and the user's thumb). Therefore, one should try to place most of the common navigation controls at the sides. Also, thumbs will cover up the content if one has to reach inwards with them. For example, if a list is placed on the left of the screen, it can be easily scrolled by using the left hand. However, if it also features a fast scroll mode where the scrollbar can be dragged (commonly found on the right of a list), the user's thumb will cover up the list while scrolling. The solution is to put the scrollbar on the left, such that no screen space is covered up when it is used. Furthermore, the user can now scroll with the thumb placed right at the edge of the screen, which is usually still touch sensitive. Therefore, like on desktop computers with the mouse pointers, the edges and especially the corners are very easy and comfortable to hit correctly. Thus, having buttons in corners or sliding fingers along vertical edges is a good choice.

A last detail which we need to be careful about is swiping vertically: As this will happen with the thumbs we should be careful to not have a too great swipe distance. Furthermore, it is probably save to say that pulling a thumb inwards (i.e. swiping right to left on the left side and swiping left to right on the right side of the screen) is more comfortable for the user than pushing the thumb outward.

There are probably even more details to which one can pay attention while keeping the tablet form factor and the positioning of hands in mind, but we mainly listed those which came to our attention while designing the UI for our application.

3.3 Avoid full screen transitions

One property of mobile phone UIs is that one screen is usually devoted to one task due to the confined space available. Thus, if the user is switching between different tasks, the phone usually transitions the full screen, i.e. displays a totally different view. This is a necessity on the phone, as there is simply not enough space to display two parts of an interface at once while still being finger-friendly and uncluttered. On the downside, navigation is connected with much more state as usually. It may be unclear to a user in what context a view was opened or how to navigate to a particular view he has seen before.

On tablets however, the situation is quite the opposite: Often it makes more sense to display two views next to each other instead of scaling one view up to the full size[4]. A common use case could for example be a navigation bar on the left side for an email program, in which the user can select an email message that will then be displayed on the right side. On the phone, the same views would have been hierarchically linked through screen transitions. Furthermore, having both views side by side the application can provide functionality which it could not have provided with one view at a time, such as drag and drop across those views.

Another strategy is to use an overlay, for example for a selection menu or to display additional information. Instead of launching a new fullscreen view, the application can just show a smaller overlay with the relevant information. Note though that this is not like pop ups on traditional computers that require action on the user's side and block all the other operations. On the contrary, the overlays can be implemented in such a way that clicking anywhere outside of their area closes them. Thus, overlays should only be implemented if they can be dismissed without having chosen a specific action. When implementing an overlay, it is a nice idea to overlay the background half transparently. This way the user does not lose the context in which the overlay is embedded, as the background is still partially visible.

Chapter 4

Design principles and goal

Any user interface has to prove its value in real life and we cannot attest its quality by solely reasoning about it. User studies are a way to emulate a real life situation, but reliable and unbiased studies are difficult to conduct. Performing an extensive user studies was beyond the scope of this thesis and thus we cannot show anything we claim by citing such studies.

Now that the empirical method has been ruled out, we are left with the possibility to reason and argue about our interface. While user interface design is clearly not an exact science, we try to argue as precisely as possible why we we made certain choices in our design process.

There is also no *best* or *right* way of how to design an interface and this is why we need to define what we actually want to achieve: We based our design process on one goal and a set of design principles and choices. The goal is to have an application with playing music as its central functionality. What we exactly mean by this will be defined and presented in the next section.

While we try to achieve our goal, we always have to adhere to a set of design principles. We already outlined a few properties of tablet devices with regard to interface design in the previous chapter. Additionally, we define a set of principles which we want to mention explicitly as they are of great importance to this specific project. The principles are that we have to keep in mind that there is no intuitive interface, to create an efficient user interface and to have an experience which is visually attractive and fun to use. This are just two principles which we would like to highlight in the context of a music player on the tablet and they do by no means cover all the principles and guidelines that should be kept in mind while designing an interface.

By the end of this chapter we have established our goal and principles. They become our guidance for the rest of the thesis. In the next chapter we present our player as a way to achieve our goal while adhering to our principles.

Before we come to discuss our goal we have to mention what kind of player we are talking about: Our player should allow sequential playback of locally stored tracks. A suitable player for a DJ or more experimental interaction with sound such as scratching is not the target of our player[18]. We want a player for casual music listeners, be it alone or in groups. Furthermore, we do also not consider streaming or Internet radio and thus, if we talk about music to which a user wants to listen, we restrict this to the music to which the user wants to listen and which is also locally available. For the sake of simplicity we will also disregard the gapless playback feature, since it briefly interleaves two songs, but in general still plays songs sequentially.

4.1 The goal of decoupling the playback experience from the interface experience

When a user is interacting with an application to listen to music, the application is giving him an experience. We define the *playback experience* as the part of the overall experience which is caused by the user hearing the sound which is produced by the music application (and its output device). In our player we are focused on optimizing this playback experience and make it our first priority.

There can be other positive (i.e. desired by the user) experiences such as looking at animations, music videos, lyrics or artist biographies. This also includes browsing the music library out of curiosity or entertainment. While such experiences certainly enhance a music player, we believe that they should only be implemented or optimized if they do not distract from our main task of optimizing the playback experience. This is why we will not look at such features for now and we assume that none are present.

We now look at the interface of a music player after defining that none of the above experiences are present: Its sole purpose remains that of shaping the playback experience. We define the experience for the user by interacting with the application's interface the *interface experience*. It consists of the interaction the user has to perform in order to influence his playback experience. Since we removed all the user-desired experiences from the interface by definition, the interface experience becomes the better, the less interaction a user has to perform. Furthermore, faster and easier to perform interactions also enhance the interface experience. Moreover, navigating to the player, be it within the operating system interface or physically moving to the player if one was away and has to turn the screen on again is also part of the interface experience.

In general, the playback experience and the interface experience are negatively coupled. If a user wants a better playback experience, he needs to do more interaction to precisely shape it the way he wants. For example, a user can get a great playback experience with almost any music player by spending the whole time interacting with the interface. An extreme case is a DJ, who provides an almost perfect playback experience (for himself, as he decides what to listen to), but who also has to interact constantly with the interface to do so.

On the other hand, one can just set the music player to shuffle and not interact with it at all. The interface experience is perfect, while the playback experience is probably not so good.

The user cares about the overall experience instead of either the playback or interface experience. Different users (and different situations) give different weight to the experiences: Some users interact more with an interface to make sure they get a great playback experience, while others simply want to listen to some music as long as they do not have to bother with the interface too much. Thus, we have a trade-off situation which is individual for every user.

We define the goal of music playback interfaces to be the decoupling of the playback and the interface experience. We further define that the more one of the two experiences decreases in quality when the other increases in quality, the greater is the coupling between them and vice versa. The larger the coupling, the harder it is to achieve quality for the overall experience. Ideally, the two experiences are orthogonal: Shaping the playback experience would then only depend on the user, as he could never choose a worse playback experience only because the better playback experience would have required a worse interface experience. At this stage, the trade-off vanishes.

Obviously, a total decoupling of the two experiences is impossible. Thus, it is also impossible to absolutely measure how *good* our interface is at decoupling the playback from the interface experience. However, we can compare our interface to existing interfaces and paradigms of music players. For example, if our player allows for an at least as good playback experience as another player while having an at most as bad interface experience, then our player can be viewed as having a less coupled playback and interface experience.

Note that we did not argue that one player's playback or interface experience is necessarily better than another's, as a user can always choose to play songs he does not actually want or click somewhere meaningless to destroy his interface experience by not using the interface properly. By talking about a decoupling of the two experiences we avoid this situation, as it leaves the decision to effectively utilize an interface up to the user.

We now discuss a couple of consequences and resulting requirements that need to be followed if we want to decouple the playback and the interface experience.

4.1.1 Finding specific songs

The interface should make it as easy as possible for the user to find specific songs that he has in mind. By *finding* we mean locating in the user interface in such a way that they can be accessed and used for various functionalities such as playback or whatever else the player has to offer. This requirement influences our music retrieval interface and is met by most traditional players, as they usually do quite well in helping to find specific songs (given their hardware limitations). Players such as the iPod are mainly optimized for this task.

If it is harder to find a specific song in one interface than the other, the trade-off between having a good interface experience and having a good playback experience is bigger in the first interface compared to the second. Thus, if the user wishes to find a specific song, the interface experienced is downgraded more severely in the interface where it is hard to find a specific song and thus there exists a tighter coupling.

4.1.2 Finding non-specific songs

The interface should make it as easy as possible for a user to find songs he would like to listen to, even if he has no specific song in mind. If the user does not specifically know which song he wants to listen to right now, we have to make it as easy and quick as possible to find a fitting song, as otherwise the user might settle with a song he does not really want. Therefore, a good song-discovery interface can enhance the playback experience for the user, while not decreasing the interface experience.

4.1.3 The user should be able to plan ahead

If a user already has ideas what to play for the next few songs, he should be able to easy schedule those songs, i.e. not require a large downgrade in interface experience to do so. On the other hand, if the user is not able to schedule songs, he will have to interact at the end of every song to get the same playback experience as if he had scheduled those songs. Interacting after each song would clearly be a worse interface experience than scheduling the songs once in advance.

4.1.4 The user should be able to act spontaneously

A user should be able to change any current or future playback related item spontaneously, i.e. at any point in time, as otherwise he cannot achieve the playback experience he wants to. This means that the interface is always readily available providing its full functionality.

4.1.5 Never interrupt the playing song, unless the user *really* wants to

We already discussed that music playback is fundamentally about what comes out of the speakers at what time. This is the most important thing for our player and this is what should not be messed up. After all, it is all about enjoying the music that is produced. This is why the currently playing song is the most important part of our interface at any moment. If the user interface (by its design) makes the user mess up some upcoming songs, it is not that big of a problem as music playback has not been affected as long as the user can fix the problem in time. However, if we interrupt a currently playing song without the user *really* wanting to do so we do not only downgrade the user interface experience, but also the playback experience itself.

But what do we mean by *really* wanting to interrupt the playing song. If a user hits pause he clearly wants to interrupt the current song. If a user tells the interface to play another song right away, he also clearly wants to interrupt the current song. Except if he had no viable alternative: A good example to illustrate this is the iPod Classic. Let us assume that a user is listening to a song with two minutes of the song left and is not in a special feature called the *on-the-go playlist*. The user has a great idea to what he should listen next and starts navigating the library. When the user finds the song about 30 seconds later he can either wait for 90 seconds and play the song then, or just play the song right now. In the latter case, the user did not plan to interrupt the playing song: There is a trade off between waiting 90 seconds and interrupting the current song. But that means that the user did not *really* want to interrupt the current song. Instead, the interface made it so tedious for the user to perform this basic functionality, that the user preferred a worse playback experience in order to have a better interface experience.

4.1.6 Avoid starting the playback of music the user does not want.

If the player starts to play music the user never wanted in the first place, he will try to interrupt this song with another song as soon as possible. Then, interrupting the song is desired, but only because something went wrong: The other song should never have started playing in the first place. Hearing a song for a few seconds and then skipping it produces definitely a worse playback experience than if this had not happened. If our interface prevents that situation without providing a worse interface experience, we loosen the coupling between the two experiences as the playback experiences was enhanced without making the interface experience worse.

4.2 Explicit principles for this project

In the rest of this chapter we present two design principles which we want to highlight explicitly among the myriads of principles and guidelines one could formulate. As we want to deviate as freely as needed from already established interfaces, we need to be especially careful about those principles to make sure our users understand our player and will continue using it after trying it out.

4.2.1 Keep in mind that there is no intuitive interface

A famous quote, which is often attributed to Bruce Edinger [5], states that

> The only intuitive interface is the nipple. After that it's all learned.

Some people even go one step further and claim that even the nipple is learned. The author of the quote wants to tell us that no interface we create can be intuitive, as everything needs to be learned at some point, maybe up to a few basic instincts, which surely are not enough to use an electronic interface. Jeff Raskin proposes to use the term *familiar* instead of intuitive in [8]. If a concept of an interface is familiar to the user, he might be inclined to describe it as intuitive. In fact though, he means that the concept is familiar and therefore he can reuse acquired knowledge. According to Raskin, intuitive would have meant that the user seems *to suddenly understand [a concept] without any apparent effort or previous exposure to the idea.* As a highly-regarded interface expert [17], he said that his newer and better interfaces were often discarded, since users performed better on older or more traditional interfaces as those were more familiar.

We do not want to limit ourselves by building a familiar interface, as building a novel interface is one of the main points of this project. However, we will use already established concepts where they make sense. So instead of going the extreme of either copying more or less existing interfaces or creating something radically almost entirely new, we try to go a balanced way somewhere in between.

Since we can neither run an expensive ad campaign nor is it our intention to write a manual which nobody is going to read, we need another strategy to make our users understand our interface. This is why we choose to have the strategy of discoverable and explanatory controls together with a consistent user interface.

Use discoverable controls and explain the user what happens once they are used

The way how we are trying to make the user understand our interface is by letting him try it out, see what happens and understand what exactly and why it happened. By letting the user experiment and then explain (in whatever way) what is happening, the user will be able to understand the interface without having to read a manual or view some instruction video.

To achieve this we first need discoverable controls. Everything which the user can click should look clickable; everything the user can drag should look draggable. Once the user clicks or starts dragging, the application should immediately respond visually and it should be clear that something happened.

Moreover, we need to explain what actually happens: Having icons without description is usually not enough and this is why we will add text when needed to really explain what the next action should be.

If we use gestures, we can explain what is happening by making the target object of the gesture respond to the gesture. For example, if we swipe a control to the right, the control should also move to the right with the finger. The user can then understand that swiping the control actually moved the control.

Consistency

We will try to make our interface as consistent as possible. Parts of the interface which look the same should behave the same. And parts which behave the same should look the same. This way, a user can learn a concept once and apply the knowledge again in another part of the interface where the same concept is used.

Furthermore, if possible, there should be as many parts of the user interface which look and behave the same, so that this reuse of knowledge can actually happen.

4.2.2 The experience should be visually attractive and fun to use

Since the iPhone came to the world of mobile phones, style has become an omnipresent requirement for all personal devices. Apple's success and the fact that many other companies now release more stylish and better looking products than before show that users really value having good looking devices, especially when they are as personal as a mobile phone. The current wave of tablets is inspired by the recent developments in mobile phones and thus it comes at no surprise that style and looks has the same value as with phones. Also the fun aspect, mostly achieved with nice graphics or engaging gestures, is very important to both modern phone and tablet platforms.

By developing a music application, we develop an entertainment application. Therefore it is even more crucial to make it visually attractive and fun to use than with most other applications. On top, music playing functionality has always been one of the iPhone's primary functionalities, as it evolved from the iPod, and helped to make Apple devices almost synonymous with playing music for many people.

All indications show that one cannot succeed on a large scale on tablets and phones with a music application that is not good looking or no fun to use. With so many applications available, the attention devoted to trying out a single one is quite limited and many users would stop trying out the application before they would discover new and improved features.

However, we are not designers and thus it is not our goal to make the best looking application where every color is chosen nicely, every graphic is beautiful and every spacing has the right value. The goal is to make a visually attractive enough and mostly fun applications so that users will enjoy it. The idea is that any decent designer could make it a great looking application without changing the way the interface works or is laid out, i.e. that the interface ideas where right, but the graphical execution not perfect.

Of course we also have to discuss how this fits into our idea of only looking at the interface experience without any fun or visually attractive features. We will still adhere to this principle and we will not make decisions purely on fun or looks. After all, every interface, be it as fun or good-looking as it gets, becomes boring after a while if the user has the goal of enhancing his playback experience.

Thus we will try to first design the functionality of our interfaces according to all the other principles mentioned before. Then, we will try to make it good-looking and fun without decreasing any functionality, clarity or getting in the way of our goal.

Chapter 5

Design choices

In this chapter we introduce our player *Jukefox for Tablets* running Android 3.0 Honeycomb. We will also often refer to the mobile phone player *Jukefox*, which is the code basis for *Jukefox for Tablets*. Before we look at the interface in detail we present a screenshot as a first overview in figure 5.1

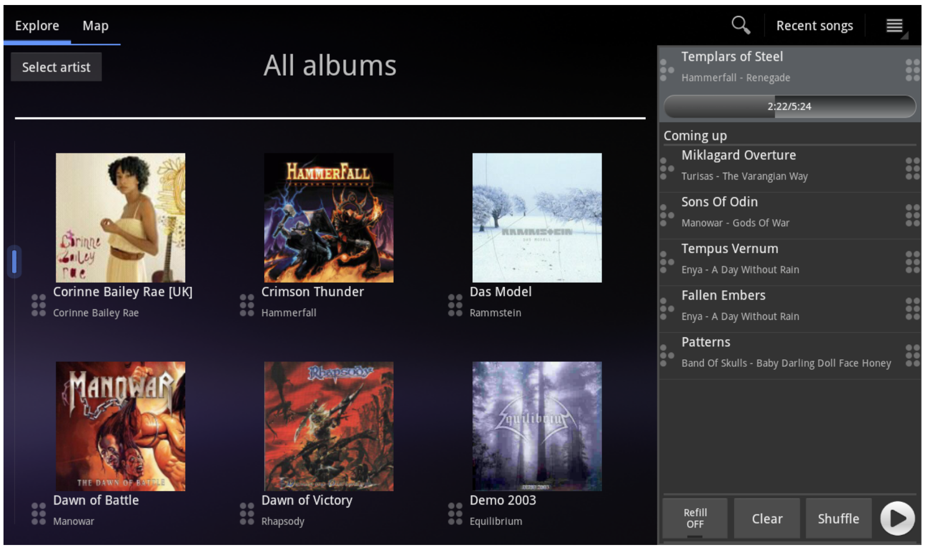


Figure 5.1: The *Jukefox for Tablets* interface displaying all albums.

5.1 The play queue

A crucial piece of *Jukefox for Tablets*' interface is the play queue; the gray area on the right as seen in figure 5.1. It consists of the currently playing song, the progress bar, the upcoming songs and the playback controls. We used color to group all the parts of the play queue such that it appears as one.

The play queue works as follows: The playing song is always on top. If the song on top is changed, then this new song will start playing. Upcoming songs are displayed in a list underneath the playing song. Even though the top song is not actually part of the list (as it does not scroll), it behaves exactly the same way as the other songs: A song can be removed by dragging the arrow on the left side to the right and it can be moved within the play queue (also to the top) by dragging it on the drag handle on the right side. When a song finished playing, it is removed from the play queue and every song moves up one position.

We made several conscious choices which lead us to use the play queue as it is:

5.1.1 The play controls should always be visible

There are plenty of reasons why users should always have access to the play controls. For example they have to mute the music because someone is calling them on another device or they suddenly get tired of the playing song and want to advance to the next one. If the player is not currently the application being run in the foreground, a user can still have access to features such as play next or play/pause via a headset or the a notification icon. The notification icon can also help the user get quickly back to the application. Another possibility is to use widgets on the home screen to also be able to control basic functionalities from there. An application should provide such controls out of the regular interface.

The same and additional functionalities should be always accessible if the user currently runs our application in the foreground. If the user for example has to leave a view within his application to browse his music collection just to hit pause or advance to the next song, this would be inconvenient. Furthermore, it would take longer to actually get to the controls and thus he would have to for example listen longer to a song than he wanted, which decreases the playback experience. In section 4.1.4 we talked about how the user should be able to change his plans for playback at any time: the sooner, the better. Moreover, if the user has to navigate through an interface to get to the play controls and abandon a current task (such as finding a specific song), he is faced with a trade off between interface and playback experience.

This is why the play queue is always visible at exactly the same spot while we are in our interface. Even if the keyboard is shown for typing a search query, the play/pause button is visible and other controls are moved up as seen in figure 5.12. Otherwise, toggling the play/pause button would require the user to temporarily close the keyboard and then open it again and thus he might choose to first finish his search which decreases his playback experience.

5.1.2 Do not implement a *now playing screen*

Almost every mobile music players features a dedicated *now playing screen*; a screen which shows the controls found on a Discman among maybe a few others and an album art. Please refer to chapter 2 for more information. Firstly, we need to look at what this *now playing screen* is. The *now playing screen* is a screen which simply features the currently playing song along with a few basic controls to such as previous/next song, play/pause and toggles for shuffle or repeat song. In short, while the user is in the now playing screen he is experiencing the Discman interface. The large screen space which is not used by the few present controls is used up by a prominent album art. Some players also expose some playlist/playqueue behavior which makes a bit more use of the available space. Navigation to the *now playing screen* often happens automatically: In most players, clicking a song in the library opens the *now playing screen*. Furthermore, being idle for some time also opens the *now playing screen*.

So why did we choose to not implement a *now playing screen*. The advantage of the *now playing screen* are that it can display a big album art of the current song, since in any other screen there will not be enough space for such a large picture. Everything else, i.e. having all the controls mentioned above, can be done in any other screen. So all the *now playing screen* gives is a good looking screen where users can enjoy a large picture and familiarity, since users know this concept from almost every mobile music player since the Discman. However, familiarity to the Discman interface is by no means an advantage, as we do not want to inspire ourselves at interfaces created for primitive devices according to today's standards. On the downside, the *now playing screen* can introduce a plethora of inconveniences: As argued in a previous point, the play controls should always be visible. Thus, having a *now playing screen* we can potentially duplicate those controls in some other place and thus break consistency. On "Music" on Android, the play/pause and previous/next controls practically at the same spot in any screen, since there is a thin strip at the bottom of each screen (except the *now playing screen*), displaying some of the *now playing screen* functionality. So in this example the developers where able to avoid such an inconsistency. On the iPad's music player however, those controls shift from the top center to the top right once the *now playing screen* is opened. The second inconvenience is that the *now playing screen* opens when the user do not want it or do not expect it. As mentioned above, most players open the *now playing screen* once song is clicked to play it. This works fine in players where clicking a song makes it play directly, yet it would be a highly annoying thing in a player which is built around having a play queue where it is unknown whether the user does not want to continue enqueueing songs. The other method of displaying the *now playing screen* would be after a certain period of user idleness. But also there it is hard to judge when or whether that is welcomed by the user. If a user comes back to a player after being idle for a while, then he most likely wants to change or add some new songs. It is probable that he already has to turn on the screen again, which is inconvenient enough, so we do not have to further annoy our users by showing them the *now playing screen*.

The last option would be to just display the *now playing screen* upon a click on some control, but then again, we are just talking about displaying a large picture so there is no usefulness involved towards achieving our goal of playing music and thus it is unclear whether users would click this, just to dismiss it later once they want functionality again. Basically, such a button would just be a toggle between a function and a non-functional screen.

To sum up, the*now playing screen* does not improve the playback experience and can only worsen the interface experience. Having an attractive interface is desirable as argued in section 4.2.2, however our primary goal is to decouple the interface and the playback experience and only make the interface more attractive if it does not introduce a stronger coupling of those experiences. There are enough ways to make an interface attractive without a *now playing screen*.

5.1.3 Upcoming songs are more important than already played songs

It is nice to have quick access to already played songs. For example, one might to check what exactly has just been played before or maybe one wants to replay a recently played song.

The back button found in almost any player is a bad tool for this. First of all, the back button usually does two things: If the song has been played for longer than a certain threshold of a few seconds, the song is restarted from the beginning. If the song has not played longer than those few seconds, the previous song is being played. While this duality of functionality was certainly a good idea for the Discman there is no reason why we should still have this on a 10 inch touchscreen. Before we do cover our alternative, we would like to point out that in places such as widgets, headphones and notification system those controls are still viable, as there the interface options are constrained.

Some players such as *Jukefox* feature a playlist in the *now playing screen*. When a song has finished playing, it still remains in the visible playlist. However, this approach has drawbacks. First of all, the playlist is monotonically increasing as long as nobody cleans up the already played songs. Then, a player provides either a way how to clean up songs explicitly (which no user would want to do) or the player cleans up the playlist when the user performs some other action, where this action then either has two meanings or the clean up of the already played songs as side effects (in which case the user loses his functionality to quickly look at already played songs).

Of course one could also just let this list grow, but then the playlist would become more and more complicated to handle as one could get lost in a huge scrolling list and would have to make sure not to lose track where the currently playing song is.

We cannot identify another reason to keep the recently played songs of a playlist on the screen, except when one mixes the concept of the list of songs which is actually playing with the concept of playlists which are a way of storing a group of songs persistently. For a discussion on such playlists please refer to section 5.6.

In *Jukefox for Tablets* we provide a solution to the problem of finding recently played songs by having a button which shows us recently played songs on click. Furthermore, this button could also be used for recent songs that were in the play queue but then deleted, however it is not clear whether it makes sense to have those songs quickly available again.

Having the upcoming songs in the queue however is very important, as it allows to plan ahead as requested in section 4.1.3

5.1.4 Gestures allow for precise and spontaneous control

Section 4.1.3 requires us to allow the user to plan and schedule playback ahead. This is why our play queue features upcoming songs as explained in section 5.1.3. While adding songs or groups of songs to the play queue, they are often not in the desired order right away. For example, we might encounter a song which we want to listen, but we are actually on the way of finding a song for the queue which we want to listen to sooner than this encountered song. If the queue did not allow for rearranging, planning ahead would be possible, but also much more tedious. A user would think twice if he wants to exactly plan ahead or maybe tolerate a few small changes in his plan in order to do less interaction with the interface. Thus, the queue needs an easy way to change the order of the upcoming songs, such that the rearrangement of songs will not cause a great decrease in interface experience. We achieve this by allowing to drag any song (even the top song) to any other song position as demonstrated in figure 5.2 on the left. For this, the user grabs the drag indicator on the right.

For similar reasons, the removal of a song in the queue should also be allowed, as it is for example sometimes easier for the user to put a whole album into a queue and so that he can then remove some specific songs he does not want to listen to. This removal is demonstrated in 5.2 on the right, where the user drags the arrow drag indicator on the left and swipes the song to the right.

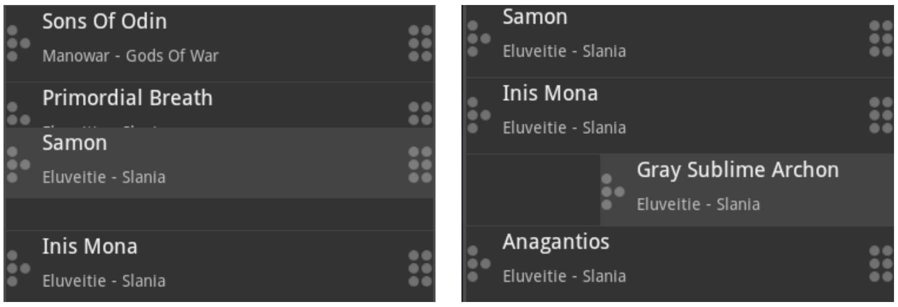


Figure 5.2: Rearranging through dragging (left) and removing through swiping (right)

Gestures are not only great for planning, but also quite important for spontaneously changing the scheduled songs as requested in Section 4.1.4. If the user wants to make only a small change spontaneously, for example the swapping of two neighboring songs, the required interface interaction is minimal and thus the user will very likely take this only slight decrease in interface experience to increase his playback experience.

5.1.5 The clear button only affects the upcoming songs

Section 4.1.4 requires that we can change all our future plans spontaneously and replace them with something else. We have seen in the previous section how little changes can be performed easily with the queue's gesture support. However, if the user wants to change the future plans totally, it is the easiest if he can simply clear the whole play queue.

However, we should not clear the currently playing song with this functionality. In section 4.1.5 we argue why the playing song should only be interrupted if the user *really* wants to do it. If clear only clears the upcoming song, but the user also wants to change the current song, then this requires one additional steps. However, if the user only wants to clear the upcoming songs, but wants the current song to continue he can either clear all the songs manually, which is tedious and decreases the interface experience, or clear all the songs regardless to have a worse playback experience. Thus we chose that clear only removes the upcoming songs from the play queue.

5.1.6 One play mode is enough

Some of the controls which are a relict from the Discman are toggles to enable shuffle or the repetition of songs. The latter often allows to toggle the repetition of a group of songs or of a single song. The repetition of a single song or album can be easily done in our play queue: Simply enqueue the respective item several times. While this will not provide a repetition forever, one can get enough repetitions in most cases. This is a quite rare use case and was included in earlier players that had no queue functionality at all. Of course this mode could be implemented and toggled somewhere in our interface, but then the queue would not function as a queue anymore and instead stay static once this mode is activated. While it could be made clear that the queue is in repetition mode, it makes the player more complicated or might even confuse people (which might forget that this mode is on). In short, it adds another dimension of state to the player, and the more state it has, the more possibilities there are to be confused, to be in the wrong state or to have to make an extra effort to switch states.

Having decided that it is probably desirable to have a single play mode, we explore how other possible play modes fit into our queue model. Shuffle is a play mode found on virtually every player and it can come in many flavors. One of them is shuffling a playlist, album or just any selected set of songs. There are two common ways how to implement this: Either, the position of the playing song jumps around randomly in the list of songs, or the list is randomly shuffled and then played from top to bottom. Having the play queue going from top to bottom anyway, it makes sense that we pick the latter approach. This way we do not need a play mode for shuffle, but can simply provide a shuffle button to shuffle our upcoming songs. Moreover, this version of shuffle comes with additional advantages on top of preserving the consistency of our play queue: The play queue is still fully functional, such that the user can first perform a shuffle and then tweak the order to his liking or enqueue additional songs. As required in section 4.1.5, we do never change the currently playing song except when the user really wants to. If the user starts a shuffle process, he is probably still happy with finishing the current song. He simply wants a randomization of what is coming next. Otherwise, he can always drag the current song somewhere into the queue after shuffling. Another benefit is that if new songs are enqueued after shuffling, another shuffling will lead to the desired result, which is that the new songs are shuffled within the other upcoming songs. Note that the songs which where previously shuffled but already played are not in the upcoming list anymore, and thus do not appear again in the shuffled list. This shows that the concept of removing the already played songs from the queue perfectly plays together with the concept of the shuffle button. The only potential disadvantage of the shuffle button instead of the randomly jumping around song position is if the user does not want to know beforehand in what order the songs will be played, as he wants to be surprised after each song. However, our interface was built around giving people control and knowledge about the playback of their music. Thus, the only advice we can give in this situation is to hit shuffle, quickly look away and enjoy the music. This should be enough, as once again it would not be worth adding some extra state for such a special need.

Another version of shuffle is shuffling the whole library. Again, having a new play mode is superfluous: While there are upcoming songs in the queue, they are played. If the user suddenly does not want them anymore, he can clear the list. If there are no upcoming song in the list *Jukefox for Tablets* will start doing a random shuffling of the whole library. This way, telling the player specifically what to play and getting random songs can be effortlessly interleaved. When one talks about randomness, one also has to specify its probability distribution. At first, players used the uniform distribution for such functionality. However, modern music players provide ways so called *smart* shuffle functionalities, which learn what songs the user is more likely to want at this moment or what songs fit well to the recently played songs.

In *Jukefox* , smart shuffle is one of its primary features and in *Jukefox for Tablets* we reuse the same smart shuffling capabilities. In *Jukefox* , among other players, the user still has the possibility to do a uniform shuffling. But why would a user want to listen to songs uniformly distributed among all possible probability distributions? In a more confined setting, e.g. where the user chooses to shuffle a group of songs such as an album, the uniform distribution makes sense as the user has already applied a selection among a larger set of songs. However, if our player should give the user random songs among all the songs, we see no reason why not to use the smarter version in any case. After all the playback experience will most likely be better (there will probably be less skips) if we have a good smart shuffle implementation. In a study published in [6], with 128 data logs over a period of each more than five days, the authors found that smart shuffle triggered 25 times as many songs to be played as the regular shuffle.

Of course one could argue once more that in some special cases someone might want explicitly this uniform distribution, but once again we would introduce more state to the player and in this case, pretty hard to detect state. If the player was running in the wrong mode, it would often be quite hard to find it out just by listening to the music, depending on the quality and configuration of the smart shuffle and the diversity of the music library. In *Jukefox for Tablets* a user could still enqueue the whole library (an item representing all the songs can be found as the first item in the grid of all the albums in the exploration view) and then hit shuffle, if he really wants it.

If we look at *Jukefox* , we see that there are two more modes that we have not covered yet. One mode plays a playlist once and then stops. While this might be desirable in rare cases (e.g. while falling asleep), it again introduces state where detecting that one is in the wrong state would be annoying. We also think that continuing with smart shuffle (which just learned from the previous playback) is almost always the better playback experience as no playback at all. This is why we did not include such a play mode.

The last mode we are going to talk about is the *play similar* song mode from *Jukefox* . It takes the currently playing song and then only plays similar songs to this song. We are not going to reiterate the added state and potential confusion that might arise from being in the wrong state. In *Jukefox for Tablets*, the interface will enable the user to obtain similar songs from any song and then simply enqueue them while still having all the benefits of the queue. In general there can be different interesting modes of smart shuffle or modes which play similar songs. Interface wise it is probably best to decide on one and try to substitute the other ones through other means. For example, one could add a button to an interface which suggests songs based on recent playback and based on some specialized and chosen smart shuffle mode. If we use an overlay to display those songs as discussed in section 5.2.2, we can not only use those songs for playback, but also to navigate to even further songs.

In conclusion, play modes are obsolete. They are a confined and limited set of ways to play the user's music and any set of play modes will be incomplete. They were introduced on players that do not allow for proper play control. Our solution of the interactive queue together with smart shuffle once the queue is empty and a shuffle upcoming song button. We are able to cover almost all the relevant and commonly found play modes while still preserving the flexibility of the queue and not adding additional state. In the future, further interesting and smart versions of play modes could be added not as actual play modes, but as parts of the interfaces making song suggestions to the user.

5.1.7 Auto refill for smart shuffle

In the previous section we discussed how we want to eliminate all possible play mode state and can provide this functionality explicitly or implicitly with our queue model. There is however one *play mode* we did not cover: It is the auto refill mode from *Music Queue*[9].*Music Queue* is an Android music player which served as a big inspiration for *Jukefox for Tablets*, especially *Jukefox for Tablets*'s play queue. In *Music Queue*, the queue consists always of five visible songs: If the user removes a song, a new one will be filled-in automatically. The idea is to improve the way people listen to music randomly.

If a user listens to random music from his whole collection, he will only listen to a certain percentage of the songs. And even if this percentage is as high as 50%, the user will still skip every second song. This is certainly not a nice playback experience, but the interface experience is quite good as the user needs to only skip occasionally, which is easy to do, and thus this mode of listening is quite popular.

While popular, this method of music listening can still be improved. In section 4.1.6 we argue that an interface should not start playing unwanted songs in the first place. If we present a couple of songs chosen by our smart shuffle instead of just the currently playing song, we can make our decision which songs to skip before those songs actually start playing. On the one hand, the playback experience is quite smooth, as there is no skipping through songs. On the other hand, the user has to do a little bit work rarely, when adjusting those next five songs. However, having set up those couple of songs, the user will not have to skip anymore for quite a while and thus not only the playback experience will increase, but maybe also the interface experience. After all, if a user has no time to adjust the next couple of songs and does not look at them, the interface will behave exactly the same as the other one, and thus can never be worse.

Another advantage is that the smart shuffling becomes much more information in a much quicker way when using this mode. While in the traditional mode the smart shuffle only becomes the played and skipped songs as input, the smart shuffle in this auto refill mode already can become information from the songs that are not even played yet. If for example a user listens to about every third song and there is an auto refill of size five, the user will in expectation go through about 15 songs until those five songs are adjusted. Going through those 15 songs can be done very quickly, as it just requires ten quick swipes. Within a less than a minute (a swipe should take less than six seconds), the smart shuffler already has the information about around 15 songs.

We chose to enable a refill mode as it clearly can provide an improved way of listening to random music and fits well into our queue interface. On the downside, we need to have an additional toggle in the queue interface. The toggle's state should indicate the activation of the mode, but ideally some more visual feature like a colored line up to where the songs are filled would be helpful. Furthermore, we have not yet been able to fully optimize smart shuffle for this mode, as we focused on interface design first, but this would surely be an interesting topic for the future.

5.2 Navigation within the music library and selection of songs

In the previous section we established the play queue as an always visible way of controlling and supervising the playback of music. This leaves the rest of the space to have an interface for navigating ones music library and selecting songs for playback.

This is where the strength of the additional screen space for the tablet comes into play. The larger area allows for much more display of information and in a more visual way than just a boring list (QUOTE GOOGLE IO). For example, the default tablet music player on Android and the player on the iPad both make use of the larger screen space by having a larger album art in the *now playing screen*. Furthermore, they also have views where they display a grids of all the albums showing their album art. Thus, they both make user of the larger screen space. However, they just feature the same interface as their counterparts on the phone in a bigger way.

While this makes navigation easier and more beautiful, it falls short of the possible enhancements a bigger screen would allow. In particular, we can utilize the fact that we can display two screens for separate tasks at once. Thus, instead of merely making the interface for song selection bigger, we make it directly work with the queue, for example via drag and drop.

5.2.1 Supporting several selection views

A *selection view* is a view which enables the user to select songs for the queue. For example, one possible selection view is showing all the albums in figure 5.1. In this section we will try to analyze how many different selection views there could or should be.

On the one hand, we do not want to restrict ourselves to one selection view only, as there is a range of possible ways to browse music and probably not one mode which fits all those possible ways. On the other hand, we also want to make sure that those selection views do not overlap significantly in what they try to achieve or in the way they function. During development we cut down the number of selection views to only two: *explore* and *map*. We also provide search, however, it is a bit different and we will cover it separately.

The concept for the selection views are that *explore* acts as a primary selection view. It should be the selection view which provides the basic functionality together with a few interesting concepts. Then, *map* acts an auxiliary navigation mode to complete *explore* with a more visual selection view. We will talk about both of those navigation modes in their own section.

Moreover, did not want to limit future versions to only two modes and wanted to keep the whole concept exdendable so that in the future novel and creative ways of selection can be tested while keeping the main views intact. Thus, the natural choice was to have a tab for each selection view. This way the selection views are on the same level in the hierarchy and can be used independently.

5.2.2 Means of achieving consistency across all selection views

Although the selection views function quite differently and can work independently from each other we have to make sure to adhere the consistency principle we defined in the previous chapter. This way the user will be able to transfer knowledge gained in one selection view to the other and the whole experience will be consistent across the whole application.

Drag and drop is possible if and only if the item has a drag indicator

The drag indicator introduced in the play queue is also used in the rest of the interface. While the drag indicator lets the user change the position of a song in the queue, it lets the user drag an item from a selection view right into the queue and drop it at any position. A drag operation can be started by either touching the indicator directly or long pressing the object the indicator is a part of; thus in exactly the same way as it works in the queue.

Almost any item which represents music, be it a name of an artist, an album art representing an artist or of course a song features a drag indicator and can thus be directly played without being opened first.

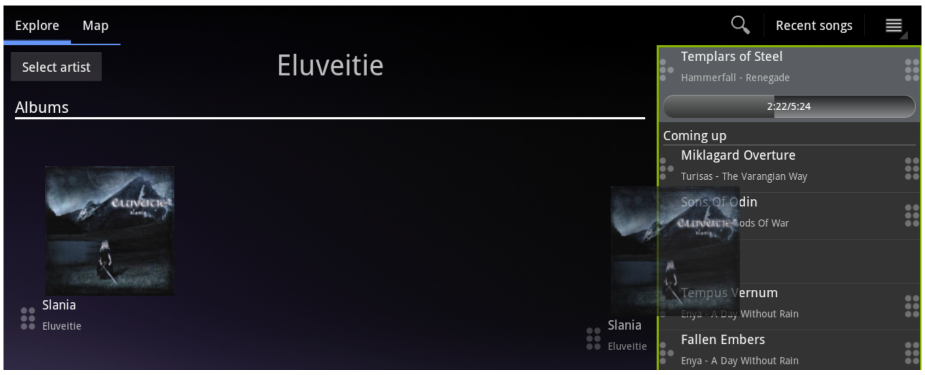


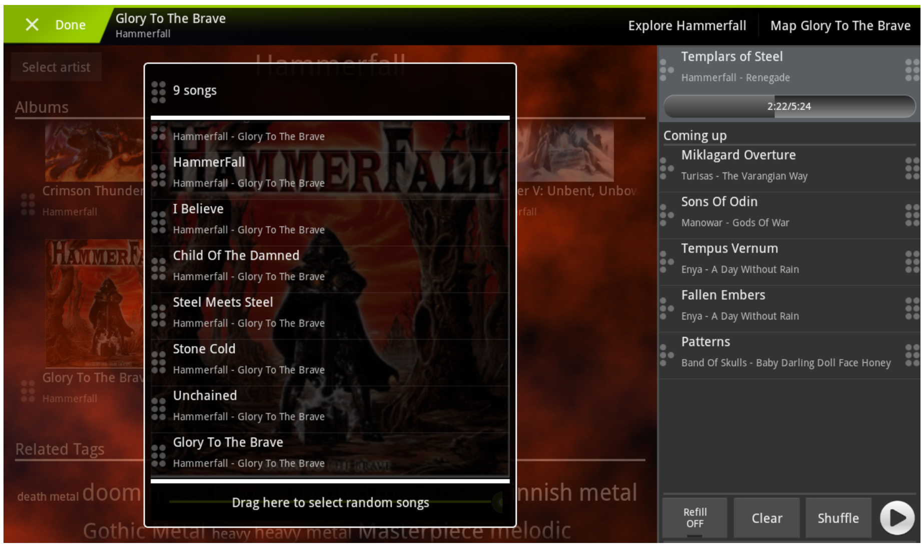
Figure 5.3: An album being dragged into the queue.

The only exception are the albums in the map and the tags in the explore mode: Adding a drag indicator there would have cluttered the view too much. In the map it would have been also problematic, since the user does many gestures with his fingers already and it would have maybe caused too many accidental drags. Only allowing drag and drop then via long press is not an option, since this way consistency would be broken and the feature would not be discoverable in the first place.

To also ensure that there is no confusion between the dragging of objects into the queue and the dragging of objects only inside the queue we limit the movement if a song is dragged in a queue to only be vertical. This way it is clear that the movement is only meant to rearrange and the song cannot leave the queue in this way. In addition, the queue is highlighted with a green frame as soon as an object is picked up in a selection view to indicate that it is the area where the song can be dropped. We do not add any further drag and drop gestures which would represent any other action than managing songs to be played in the queue. This way we really ensure consistency throughout the concept of drag and drop. As soon as the user discovers that drag and drop is possible since the drag indicators are quite visible and invite to be pressed or held, he will realize quickly how to use drag and drop in any situation. It is also not a bit problem that some users will probably never notice that long press also leads to drag and drop (although they could, as it would feel quite natural on the Android platform), since it is only an added convenience next to the drag indicator which does the same thing.

The song overlay as the only way to display songs

The song overlay is the only view (in a selection view) in which the user will ever encounter a song. The song overlay is a square view which pops over the current view and triggers an action view. It is used to display a list of songs.



Done
Glory To The Brave
Hammerfall
Explore Hammerfall
Map Glory To The Brave
Select artist
Albums
Crimson Thunder
Hammerfall
Glory To The Brave
Hammerfall
Related Tags
death metal doom
Gothic Metal heavy heavy metal Masterpiece melodic
nnish metal
9 songs
Hammerfall - Glory To The Brave
HammerFall
Hammerfall - Glory To The Brave
I Believe
Hammerfall - Glory To The Brave
Child Of The Damned
Hammerfall - Glory To The Brave
Steel Meets Steel
Hammerfall - Glory To The Brave
Stone Cold
Hammerfall - Glory To The Brave
Unchained
Hammerfall - Glory To The Brave
Glory To The Brave
Hammerfall - Glory To The Brave
Drag here to select random songs
Templars of Steel
Hammerfall - Renegade
2:22/5:24
Coming up
Miklagard Overture
Turisas - The Varangian Way
Sons Of Odin
Manowar - Gods Of War
Tempus Vernum
Enya - A Day Without Rain
Fallen Embers
Enya - A Day Without Rain
Patterns
Band Of Skulls - Baby Darling Doll Face Honey
Refill OFF
Clear
Shuffle

Figure 5.4: An album's overlay.

But before we look in detail at the song overlay we look how this view is opened, i.e. what the user has to do that it shows up and overlays the current view. There are a few special ways how an overlay of songs is displayed, but we can state the following implication: If the song overlay is opened by clicking on a view inside a selection view, this view must have had a square shape. One example are albums which are displayed as an album art. If the user clicks this album art, the song overlay will appear as a zoomed in overlaid version. In this mode, when the overlay displays a single album, we are also able to add some nice looking effects such as displaying the album in the background of the list of songs or to display some cloud-like effect around the overlay with a color used in the album. The benefit of only allowing square items to cause an overlay is that the user will learn to expect an overlay once he sees such an item, especially since the view causing the overlay and the overlay share the same shape and background.

This decision to only have only square items which represent a list of songs opening up overlays which actually display songs leads the interface in a direction where it will consist mostly of items representing albums. A single song will never be in the main selection view interface. One might think that this introduces an inefficiency, as nobody will ever see a song without first clicking on an album or a square item representing all songs of a particular artist. We take this as a small inefficiency trade-off. However, tapping outside of the overlay on the transparent cloud-like structure will take the user back to the screen underneath again. Thus, transitioning back is quick and since the overlay is not a full screen transition due to its transparent border the user does not lose context because of some hierarchical navigation.

Having songs always in the song overlay we can now think about enhancing it with some common features that will be desirable everywhere: We have talked about how any item with a drag indicator can be placed into the queue by drag and drop. While drag and drop is certainly enough to put albums an artists into the queue (after all, users will not be doing too often as one drag operation already provides quite a long time of music), this would probably end up tedious if it were the only possibility to play a song. After all a drag operation requires more time than clicking and furthermore makes the user extend his thumb out of the comfort zone or hold the tablet in one hand. This is why users can select songs in the song overlay. There is also an item on the top representing all the songs (and displaying with a number how many there are). Clicking it will toggle all the songs. As long as a song is selected three areas appear on the left side of the overlay easily reachable by the left thumb. They also appear if a user starts dragging a song in the overlay mode to serve as a drop target. If songs are selected they can be clicked to perform an action on all selected songs. The three actions are play now, play next and enqueue. Instead of using symbols, we use words to describe what they do, as there is enough space and we argued in our principles that we want to be descriptive. This is an example how we explain a user what he should do, after he clicked or started dragging a song, i.e. after he curiously tried out what is possible.

Those three actions are probably the most common actions which a user might want to perform with songs. Additionally, a user can always perform a rearrangement with his right hand in the queue after the left hand performed one of those actions, if this is more comfortable for the specific user than directly dragging and dropping the song. While play next and enqueue insert songs on the top and on the bottom of the upcoming list respectively, play now replaces the currently playing song (and inserts the the rest of the songs on the upcoming list if the action was performed on several songs). This way the action is not quite the same as if the user dragged the song or songs to the top of the queue, as then the currently playing song would be pushed down. But in the case of dragging, this is made clear during the animation and thus removing the currently playing song would not make sense. Also, the previously playing song

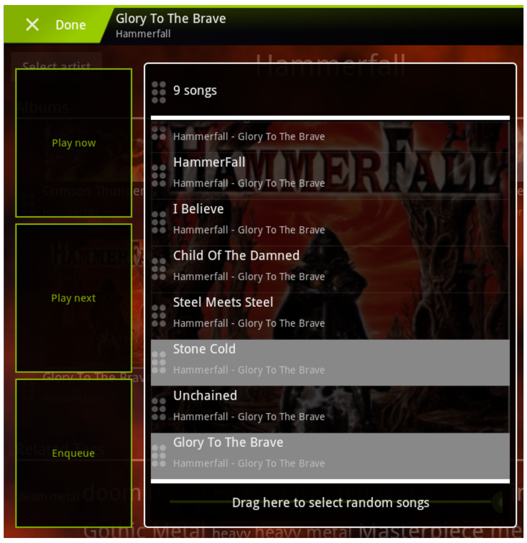

Done
Glory To The Brave
Hammerfall
Play now
Play next
Enqueue
9 songs
Hammerfall - Glory To The Brave
HammerFall
Hammerfall - Glory To The Brave
I Believe
Hammerfall - Glory To The Brave
Child Of The Damned
Hammerfall - Glory To The Brave
Steel Meets Steel
Hammerfall - Glory To The Brave
Stone Cold
Hammerfall - Glory To The Brave
Unchained
Hammerfall - Glory To The Brave
Glory To The Brave
Hammerfall - Glory To The Brave
Drag here to select random songs

Figure 5.5: An album's overlay with selected songs.

can be easily removed in this case if it needed, since the finger is already in the right place as it just dropped songs. In the case of pressing the *play now* button however, the semantics is that the user has heard enough of the current song and wants to play another song right away. We think that this situation is more common and do not expect any confusion arising from the way we handle this.

Another functionality of the song overlay is the ability to navigate to any song's artist or album in the form of the explore and map mode respectively. If all the songs are from the same album or artist, the respective controls are displayed at the right top in the action bar (Android Honeycomb's name for the bar on top), which is Android's preferred place for context related actions.

A last feature present in any overlay is the slider on the bottom. To make clear what it does and encourage the user to try it out, we add a descriptive text on top as suggested in section 4.2.1. When the user is moving the slider, the overlay will randomly select and permute a subset of songs which will be displayed instead of all the songs, the relative number of songs selected being proportional to the position of the slider. This way, one can create a random and smaller playlist out of every set of songs and there is no need to add any kind of playlist generation tool in any other place. An open question is what to do when the overlay only displays one or two songs to begin with: For now the slider remains, but we might remove it in the future.

5.2.3 Put the play queue on the right side

For a long time during the development the play queue was placed on the left side, without giving it much initial thought. The queue is a bit more complex to operate than the other modes and since more than 70%-95% of the population are right handed[11], placing the queue on the right makes our player easier to operate for a majority.

Another advantage is that the content is selected from the left side and

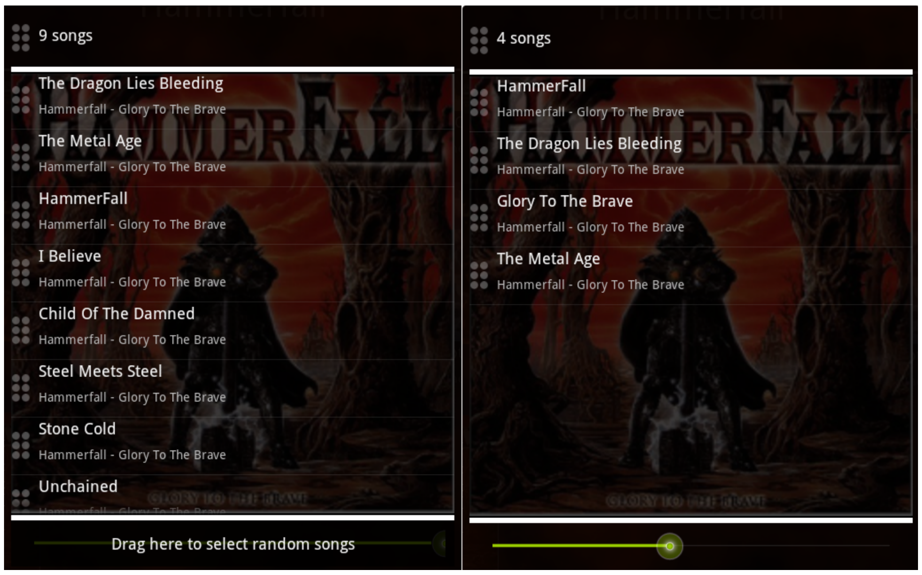

9 songs
The Dragon Lies Bleeding
Hammerfall - Glory To The Brave
The Metal Age
Hammerfall - Glory To The Brave
HammerFall
Hammerfall - Glory To The Brave
I Believe
Hammerfall - Glory To The Brave
Child Of The Damned
Hammerfall - Glory To The Brave
Steel Meets Steel
Hammerfall - Glory To The Brave
Stone Cold
Hammerfall - Glory To The Brave
Unchained
Drag here to select random songs
4 songs
HammerFall
Hammerfall - Glory To The Brave
The Dragon Lies Bleeding
Hammerfall - Glory To The Brave
Glory To The Brave
Hammerfall - Glory To The Brave
The Metal Age
Hammerfall - Glory To The Brave

Figure 5.6: An overlay before sliding (left) and after sliding (right).

dragged to the right side, which seems to be the more natural in a culture where we read and write from left to right. The same can be applied to the swipe removal, which is also left to right if we place the queue on the right, as discussed in section 3.2. The drag controls are then on the right side to allow easy dragging along the vertical edge of the tablet.

If one where to internationalize this application, one would maybe prefer (for similar reasons) to put the queue on the left side.

5.2.4 Sacrifice genres (for now)

Unfortunately we have not yet found a satisfying way to implement genres, since we already prominently feature tags and having both systems next to each other might be confusing. Maybe there is an elegant way how to combine the two concepts, however we will also have to see how users react to their omission. Maybe the combination of linked navigation by tags, similar albums of the map and smart shuffling will take on the roles that genres might have had before. Thus we sacrifice the concept of genres for now in order to keep the concepts simpler by not having two competing paradigms such as tags and genres.

5.3 Explore

One advantage of the tablet over the phone is the ability to flatten navigation (as we have already discussed?). At one stage of the development for *Jukefox for Tablets*, we considered having three lists next to each other as a way of navigation: artists, albums and songs. The artist list would always display all the artists, while at the top one could also select ”all artists”. Then, depending on which artist was selected in the artist list, the album list would display this artist's albums. Of course, if "all artists" was selected, we would then display all albums. At the top of the album list, analogously to artists, there was an all albums button and then in the same way all the songs of this album where displayed in the song list. The default selections for artists and albums were always those, which selected all artists or albums respectively. This hierarchical navigation was very efficient: Initially it displays all the artists, all the albums and all the songs. If the user restricts the navigation to a specific artists, he immediately sees all of his albums, and in the third list also all of his songs as "all albums" was the default selection in the second column. On a phone, those three lists would have been three different screens, and navigation between them required moving up and down levels. Having all three next to each other was more efficient and in most of the cases it also allowed to skip the album level as it had a sensible default selection (which on the phone would need to be pressed first).

However, there were numerous drawbacks which lead us to discard this interface: Firstly, it was not thumb friendly, as the lists in the middle of the screen were hard to scroll (see section 3.2). Then, it was one of the least attractive interfaces imaginable. Together with the play queue, our interface had four scroll lists next to each other without a single image or graphic. A visual chaos for any user. According to section 4.2.2 the interface should be attractive and fun to use. Thus, just using the tablet's larger screen to flatten a hierarchy was not sufficient to create a good interface. Having a good interface experience is not all about efficiency, but also about how comfortable or convenient it is to use this interface. Furthermore, we were already experimenting with a non-hierarchical navigation mode at this time and having two different modes which where too similar made the interface too complicated and thus also would have hindered efficiency.

We will now discuss the design choices made for our default navigation mode: the exploration mode.

5.3.1 Allow for hierarchical navigation

Music is, in some sense, hierarchical. Artists create albums and albums contain songs. Furthermore, this kind of information is usually known by the users and hierarchical navigation is usually a core functionality in music players as discussed in chapter 2.

While it has been suggested that hierarchical navigation has hit its limit in [6], not everyone has an enormous music library. Moreover, also in huge libraries it is sometimes useful to for example select a particular artist from a list of all artists. This is why we decided to keep some form of hierarchical navigation, albeit not exactly in the traditional way. But before we explain how hierarchical navigation fits in, we talk about what navigation we primarily want.

5.3.2 Encourage flat and linked navigation

Since our application is built on top of *Jukefox* , we can make use of high dimensional Euclidean music similarity space (from now on called *music space*) which is described in [6]. In particular, the music space allows for both a global view of the music (where songs can be assigned a global coordinate), as also for a local view (where each pair of songs can be compared by a similarity measure to each other). In the exploration view, we are going to make use of the latter. We are not going to explain the music space in detail and how to use it as this is done for *Jukefox* in [10], but try to give an overview of what we need for our purposes. In the music space, the similarity between two songs is given by their distance in space. Furthermore, we also use social tags which are part of the music space in form of direction vectors. The vector's direction influences the tag's style, and a vector's length influences the tag's popularity. Songs are points in space, and the further they lie in the direction of a tag, the more probable it is that this tag is assigned to this particular song. Therefore, the scaler product between a tag and a song gives us a measure. This is what is used in *Jukefox* to allow playlist creation based on tags.

We extended this idea by also calculating similarities between tags, as the scalar product between two tags indicates their similarity. In this case, however, we chose to use the angle instead of the scaler product to also include less popular tags.

This way we have similarity measures between all possible combinations of songs and tags (in particular also between any two songs and any two tags). With this, we have a viable alternative to the hierarchical nature of music given by meta data. It enables us to provide flat navigation via links, so that users can browse their music more like they browse the Internet, instead navigating it as if it were a file system.

However, we did not want through songs, tags, album and artists but have some more given structure: Our solution is to permit the exploration of artists and tags. If the user explores an artist, we display all the artist's albums, a collection of tags and a set of related albums. We also include a special collage of albums representing all songs of an artist and one representing all related songs of an artist.

Having no special view to explore a particular song or album, the user cannot perform exploration up or down the hierarchy: The exploration page already shows everything of the artist; songs can be seen by opening the corresponding overlay. Furthermore, every overlay becomes a navigation hub as described in section 5.2.2 and allows us to explore either the artist or map the album of the songs in the overlay. On top, playlist generation, for example creating a playlist out of all related songs, is built into the overlay as well. An additional benefit is that we only display a tag cloud and grid of albums, so the whole experience is very visual and we get to improve our application with respect to section 4.2.2

If a user clicks on a tag, we show related tags of this specific tag along with related albums of this tag. In the current version, the tags are arranged in a tag cloud alphabetically, where the size of the tag indicates how much it is related to the tag or artist we are currently exploring. This is not optimal though. If a user sees a tag in the cloud, even if only with a small font, which does not seem to fit at all, he might think that our tag system is not very good. On the other hand, we do not only want to display the very close tags, but also some tags further away. If we are only displaying the closest couple of tags, the user will never (or only very slowly, depending on how dense the tags are distributed) get away from those tags. The perfect navigation allows not only to navigate closely to related areas, but also to reach every other area in a small, constant amount of clicks. Otherwise our related navigation alone is not sufficient, and the user will have to rely on the hierarchical navigation again. It is okay if the user uses hierarchical navigation, after all we provide it, but he should not have to use it

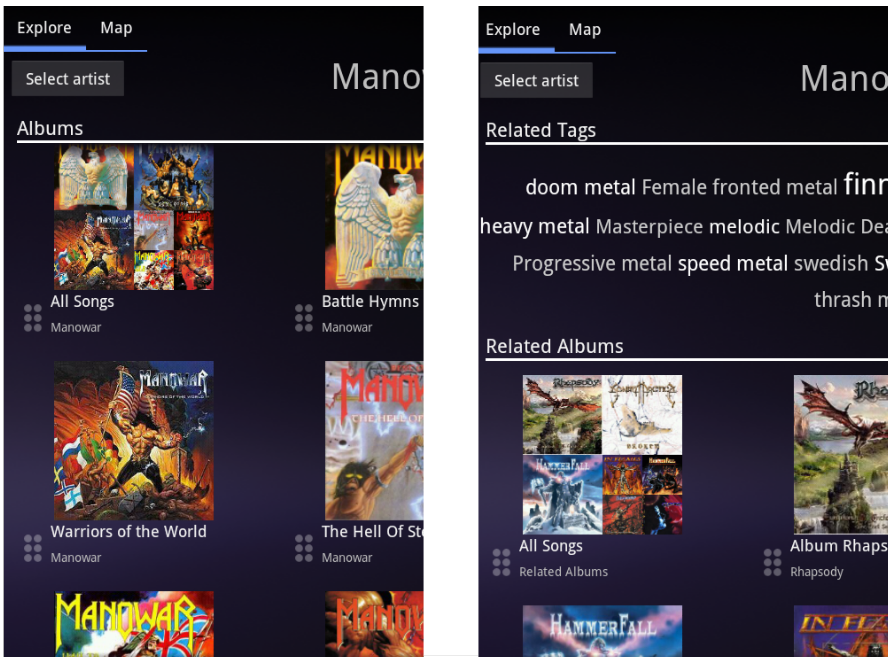


Figure 5.7: An artist's exploration view. On top (left); scrolled down (right).

only because our linked navigation system is not satisfactory. We have planned to implement a solution where we also consciously display not as close tags, but instead of a tag cloud use a more spatial approach where the related tags will be in the middle and the not so related tags further away at the boundaries.

A last question is to look at how we can build in hierarchical navigation. A large part is already provided: If we explore an artist, we can see all the artist's songs and albums. We also add a button to select a specific artist to explore from a list which gets overlaid on the left. To avoid full screen transitions and suggested in section 3.3 we only display a list on the left side of the screen and darken the rest out transparently. Also, the list is very comfortable to navigate as requested in section 3.2, with the scrollbar for faster scrolling on the left edge of the screen. Furthermore, there is also an option to select all artists and thus to display a grid of all albums, which we use as a default view when the application is opened. Also, the first item of this grid is a collage of albums which represents all songs, so one can also find the list with all songs.

In conclusion, the exploration view offers a mixture between hierarchical and linked navigation between related music. In the design, we did not want to force the user to use the linked approach, but instead tried to make it as attractive as possible, while blending in the hierarchical navigation so that one does not have to navigate multiple levels of hierarchical screens.

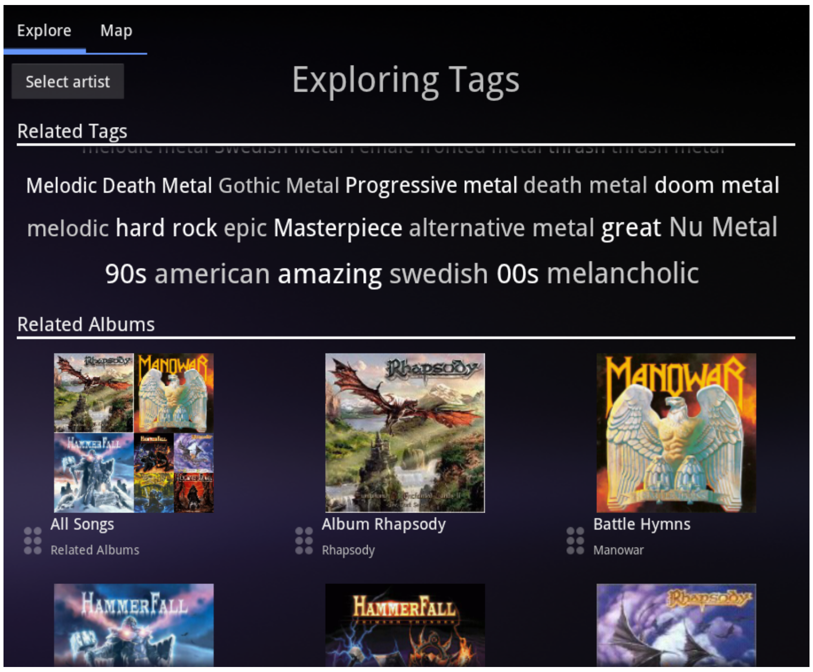

Explore
Map
Select artist
Exploring Tags
Related Tags
Melodic Death Metal Gothic Metal Progressive metal death metal doom metal
melodic hard rock epic Masterpiece alternative metal great Nu Metal
90s american amazing swedish 00s melancholic
Related Albums
All Songs
Related Albums
Album Rhapsody
Rhapsody
Battle Hymns
Manowar

Figure 5.8: A tag's exploration view.

5.4 Map

The map was not developed as part of this project but is one of the feature of the *Jukefox* player. We already discussed that during the development process we cut down on the number of modes that songs could be selected by merging the explore and hierarchical selection view. The map, however, fits into the concept as a secondary navigation mode: It offers the global view of the music space described in the previous section. As exploration is built on top of hierarchical meta data and local similarity measures instead of the global similarity view, there is little overlap with the music map and thus it makes sense to support both modes. Furthermore, the music map is a fun and playful way and helps us to make the player visually impressive in order to satisfy section 4.2.2.

The map needed almost no adaption from the phone version, scaling automatically to fit all available space, yet there are a couple things worth mentioning.

The ability to select a region of albums to create a playlist is hidden on the phone as one needs to press the trackball to activate it. On tablets, the larger screen space made it no problem to have space for a button at the left bottom where it is easily clickable. Furthermore, we made it semi-transparent and round so it looks unimposing on the map of albums, yet visible enough on the black background if zoomed out completely. The button uses an icon of a convex polygon to hint at the area of selection that the user draws to select a set of albums. While this might remind users that have tried the feature before, it does not tell a novel user what will happen. As described in section 4.2.1, we try to utilize the curiosity of users to our advantage and instruct them in the

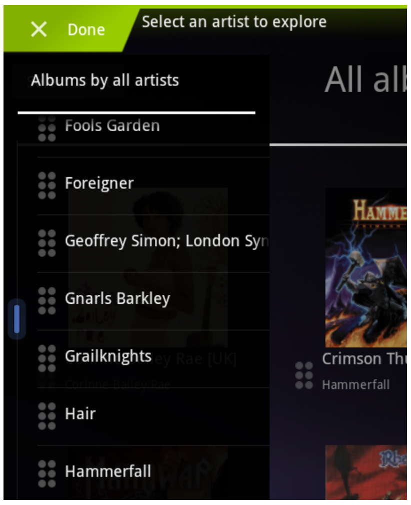


Figure 5.9: A list to select an artist to explore.

bar at the top with what they have to do.

Once the selection happened, we display an overlay of all the songs of all the selected albums. There is nothing more we have to do: Random playlist creation and navigation options are already provided as described in section 5.2.2.

5.5 Search

The Android developer guide states:

> Search is a core user feature on Android. Users should be able to search any data that is available to them, whether the content is located on the device or the Internet.

Over the last few years, search has become an almost ubiquitous paradigm and it comes as no surprise that Google as Android's primary developer has put great emphasis on search for its mobile platform. From the perspective of a mobile music player it makes sense to provide search (especially together with search-as-you-type) since typing a few characters yield faster results than list scrolling depending on the size of the user's library, even if only a software keyboard is available.

To make search worthwhile for the user, it has to be as efficient and painless as possible, as otherwise scrolling through lists will be faster and more comfortable.

First of all, we note that search-as-you-type is a must, especially on devices with soft keyboards where typing is anything but optimal. The interface experience will be better in any case.

Otherwise, there a few different ways how search can be done since we have different types of data such as artists, albums and songs. A first way is to let the user select what kind of data should be searched. However, selecting what

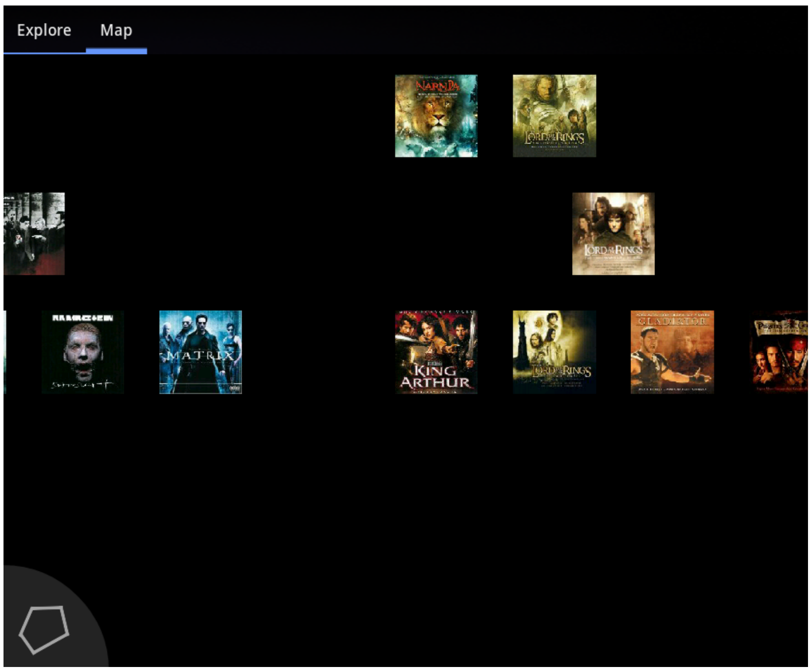


Figure 5.10: The map view.

to search is not necessary if the search query already implies what the user are looking for.

Another way would be to let the user have a search box without options, but to display data items of different types at the same time either mixed or grouped by type.

A third way is to only display one type of data type (e.g. songs), but search for all data types in the background. For example, if one searched for some album then all songs in this album would match the search term.

We chose to implement the last option for a couple of reasons: Songs are displayed in the same fashion across the whole application: In popovers which allow enqueueing, random selection and navigation functionality. So if we are to display songs in search results (which would be the most obvious type to display), we also should display them in such a popover for consistency reasons. But then, we cannot display artists or albums since there will be no space. However, that is not necessary, since navigation functionality is built into the popover already. If the user wants to explore an artist or map an album by search, he can simply search for the artist or the album respectively, click on the first matching song and user the navigation buttons on the right top. And if the user wants to search for an artist or an album just to enqueue some or all of its songs, he does not even need to leave the search interface as all the matching songs are already displayed. In this case, he can also just search for another term without leaving the search interface. This would not be possible if we had chosen to display artists and albums, since clicking them would open another view.

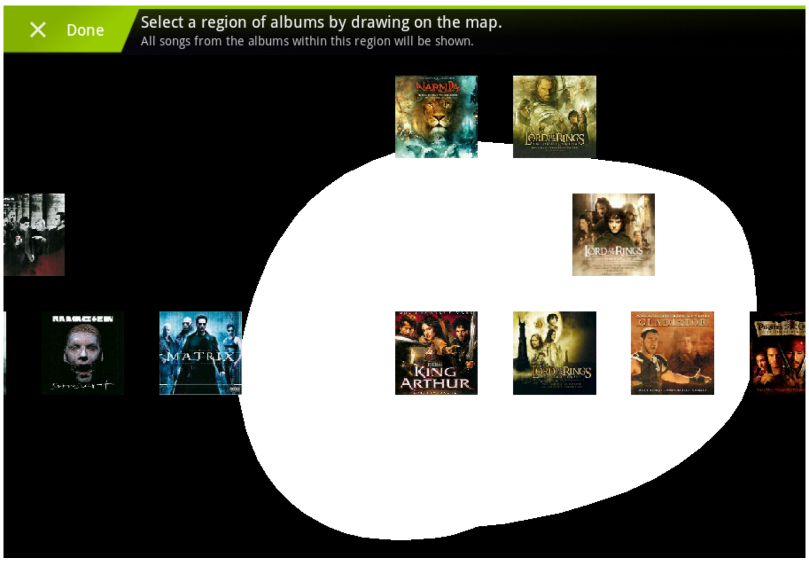

Done
Select a region of albums by drawing on the map.
All songs from the albums within this region will be shown.

Figure 5.11: The map while the user is drawing a region with his finger.

In conclusion, the approach to only display songs with search term matching song, album or artist name and display them in the omnipresent popover gives the user the most consistent, efficient solution and functional solution.

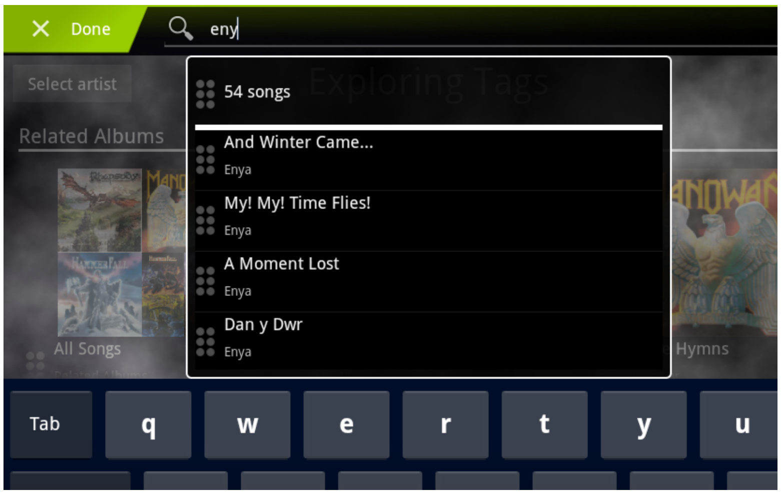
5.6 Playlist

Playlists are a way of saving time and work that the user put into grouping and arranging songs that he wants to listen to often or together. They are away of reusing already invested interface interactions, as it the interface experience can improve if we can do repetitive actions only once.

Many players however mix concepts when using playlists. For example, a player might use playlists as the only mean to allow for planning as required in 4.1.3. However, playlists are usually not first-class citizens in music player interfaces, as usually playback happens through other means. For example, in players like the default player on Android or iOS, scheduling songs can only happen via playlists. If the user is not currently playing songs from a playlist, he cannot start scheduling songs without interrupting the playing song, which is undesirable according to section 4.1.5. Furthermore, if a user wants to use playlists for dynamic scheduling, he has to create a persistent object as a side effect. Then, either the list of playlists will be full of useless playlists or he will have to delete them from time to time, investing energy for maintenance.

Another undesirable interface choice coming from the use of playlists where they should not be used is the display of already played songs, which is discussed in section 5.1.3.

When developing *Jukefox for Tablets*, we consciously decided to not implement playlists until the very end. This way, they are not able to interfere with the rest of our design and we will not mix them up with other concepts. So far, we have not yet have time and that is why there is unfortunately no persis-



54 songs

And Winter Came...
Enya

My! My! Time Flies!
Enya

A Moment Lost
Enya

Dan y Dwr
Enya

Figure 5.12: The search view.

tent mechanism to store a group of songs. This is surely a desired feature for the future, but of course we will be required to first think the concept through instead of just implementing and old paradigm.

Chapter 6

Conclusion and outlook

The journey to the optimized music player is far from over. Our aim of trying to decouple the playback experience from the interface experience clearly made the goal of making a great music player into an optimization problem, which is impossible to ever be maximized. With our possible definition of what the goal of a music player actually is, we have built a foundation upon which music players can be compared. We hope that also others will realize that more emphasis has to be put into the playback interface itself, instead of only looking at new music retrieval interfaces. Those parts of the interface are both of great importance, then only with both together can a music player truly shine.

Our music player is a solid first attempt at getting closer to the goal of an optimized music player in the sense we have and we were able to show by many examples, how our choices achieve a greater decoupling of the playback experience and the interface experience than in other players.

But much more important than theoretical discussions is the implementation and distribution of such ideas. As of this writing, *Jukefox for Tablets* is not yet ready to be released on the Android market. To really make a compelling product, one needs to take care of countless more details than already mentioned. We hope that we will soon be able to launch a first version and hear from the users what they think about our ideas. Furthermore, it would be great to have some usage data to be able to verify whether the users engage with our application as intended, or whether they even employ new usage patterns not yet imagined.

We have already mentioned certain points in chapter 5 which are so far missing or not optimal, such as the absence of anything comparable to playlists and it would be great to add or improve those functionalities. As a long term goal, it would be our dream to take this concept to other platforms as well, first and foremost to the mobile phone, where many of the ideas and almost all of the source code could be reused.

Bibliography

[1] Open Handset Alliance. *Android Developer Guide: Optimizing Apps for Android 3.0.* 2011. URL: http://developer.android.com/guide/practices/optimizing-for-3.0.html.

[2] Tim Bray. *Tall and Narrow.* May 2011. URL: http://www.tbray.org/ongoing/When/201x/2011/05/22/Portrait-Mode.

[3] *History of the Portable Audio Player.* 2008. URL: http://gadgets.softpedia.com/news/History-of-the-Portable-Audio-Player-046-01.html.

[4] Apple Inc. *iOS Human Interface Guidelines.* 2011. URL: http://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/MobileHIG.pdf.

[5] Richard Kettlewell. *The Only Intuitive Interface Is The Nipple.* 2001. URL: http://www.greenend.org.uk/rjk/2002/08/nipple.html.

[6] Michael Kuhn, Roger Wattenhofer, and Samuel Welten. "Social Audio Features for Advanced Music Retrieval Interfaces". In: *Human Factors* (2010), pp. 411–420. URL: http://portal.acm.org/citation.cfm?id=1874007.

[7] Elias Pampalk and M Goto. "Musicrainbow: A new user interface to discover artists using audio-based similarity and web-based labeling". In: *Proceedings of the 7th International Conference on Music Information Retrieval Victoria BC Canada.* Ed. by Kjell Lemström, Adam Tindale, and RogerEditors Dannenberg. ISMIR, 2006, pp. 367–370. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.140.9090&rep=rep1&type=pdf.

[8] J Raskin. "Viewpoint: Intuitive equals familiar". In: *Communications of the ACM* 37.9 (1994), pp. 17–18. URL: http://dialnet.unirioja.es/servlet/articulo?codigo=388848.

[9] Yannick Stucki. *Music Queue.* 2009. URL: http://yannickstucki.com/musicqueue.html.

[10] Samuel Welten. *Personalized Organization of Music on Mobile Devices.* 2009. URL: http://www.disco.ethz.ch/theses/fs09/report_swelten.pdf.

[11] *Why are more people right handed?* 2001. URL: http://www.scientificamerican.com/article.cfm?id=why-are-more-people-right.

[12] *Wikipedia Digital Audio Player*. 2011. URL: http://en.wikipedia.org/wiki/Digital_audio_player.

[13] *Wikipedia Discman*. 2011. URL: http://en.wikipedia.org/wiki/Discman.

[14] *Wikipedia Discman D50*. 2005. URL: http://en.wikipedia.org/wiki/File:Sony_Discman_D_50.jpg.

[15] *Wikipedia iphone*. 2011. URL: http://en.wikipedia.org/wiki/IPhone.

[16] *Wikipedia iPod*. 2011. URL: http://en.wikipedia.org/wiki/IPod.

[17] *Wikipedia Jef Raskin*. 2011. URL: http://en.wikipedia.org/wiki/Jef_Raskin.

[18] *Wikipedia Scratching*. May 2011. URL: http://en.wikipedia.org/wiki/Scratching.