Title

Andreas Berre Eriksen, Lars Emil Eriksen
Jens Emil Gydesen, Mikkel Alexander Madsen
Nichlas Bo Nielsen & Ulf Gaarde Simonsen
D301E12@cs.aau.dk

December 20th - 2012

TITLEPAGE

PREFACE (TODO)

The following is a list of words used in the report, that might not be well known.

App/Application An application (or app for short) is a piece of software run by a smartphone (although it may also be used for a program on other platforms)

Webservice A webservice is an application hosted by a remote system, often executable through a web browser

Bookmarklet A bookmarklet is a small application stored by a bookmark in a web browser

Signatures

Andreas Berre Eriksen

Lars Emil Eriksen

Jens Emil Gydesen
jgydes11@student.aau.dk

Mikkel Alexander Madsen

Nichlas Bo Nielsen

Ulf Gaarde Simonsen
usimon11@student.aau.dk

Contents

List of Figures

List of Tables

Chapter 1

Problem Analysis

In this chapter we will first describe our initializing problem in Section 1.1, as an introduction to our report. Since there's already some existing services dealing with our problem, we will analyze these to see what they do well, and where they can be improved, which we will do in Section 1.2. To find a solution to our problem, we need to know who our target group is, which we will describe in Section 1.3, as well who our informants are.

1.1 Initializing problem

Music is an huge part of many peoples lives. There are very few people, if any, in the world who doesn't know any music that they like. The amount of different music genres is enormous, and how each genre sounds can differ dramatically. Because of this people's tastes in music are often very different and this can cause problems. When a lot of people with different music tastes are gathered, it can often be very difficult to decide which music should be played. This can lead to the music changing a lot, even before songs are finished. It can also lead to songs playing that only a few people wants to hear. It is also annoying for the host of the party to be responsible for the music, and have to spend a lot of time to make sure that there is always music playing. To get a better understanding of this problem, e will now analyze some existing services which already try to counter some of these problems.

1.2 Existing services

In this section, we will look at existing services and analyze their functionality as a social playlist service. Each service will be briefly described and analyzed in a pros/cons table.

1.2.1 TuneTug

TuneTug as described from the website:

> "TuneTug is a new service that lets everyone be the DJ at parties and events. TuneTug allows your guests to "Tug" up and down songs with their mobile phone that they want to hear. To set up TuneTug you need an internet connected iPhone, iPod or iPad - WiFi is suggested but not required. Users on most any type of mobile device can vote and see the playlist live. TuneTug Voting has been tested on recent smartphone devices including: iPhone, Android and Blackberry." [5]

TuneTug as an application is only available on the iOS platform. This gives some limits. The playing of music can only be done on the application, with the built-in music player, or through Spotify, with a premium account for 99DKK [4]. The playlist is either run through Spotify, iTunes library (locally) or by combining these two into a new playlist. TuneTug also has a webservice which can be run on a smartphone with a browser. You can log in through Facebook, or create a quick name. You can "Tug down" or "Tug up" a song to push this song on the dynamic playlist. Tugs neutralize each other. This means, that 1 "Tug up" +1 "Tug down" = 0. The one with highest rating is the next song to be played. The UI is basically a playlist, where you can see the song played at the top, and two buttons "Tug up" and "Tug down", as seen by Figure 1.1 and figreffig:ttweb.

In the iOS application the following can be configured:

- Party name
- Tugs per user (standard is 10), these are votes
- Update party location via GPS
- Limit song plays
- Clear votes
- Participants must register (yes or no)
- Anyone nearby can join party (yes or no)

We will now shortly analyze the pros and cons of TuneTug. An overview of the pros and cons can be seen in Table 1.1.

You're able to vote through the web application, which allows all users to vote, and not just those with iOS devices. The host can control the privacy of the party. He can make the party public, giving easy access to everyone nearby, or lock it with an ID, which the guests needs to get before they can vote. He can also choose whether or not the guests have to register first. If the host has a premium Spotify account, he can stream the music through that, giving the guests a lot of music to choose from. This does, however, cost 99DKK a month to have, otherwise he will have to own the music himself. The host will also have to own an iOS device himself, to use the native application.

Pros	Cons
Web application for voting	Platform dependent
Parties can be public or locked with an ID	Have to own the music or pay
No registering required	
Easily set up	
Spotify connectible	

Table 1.1: TuneTug pros & cons

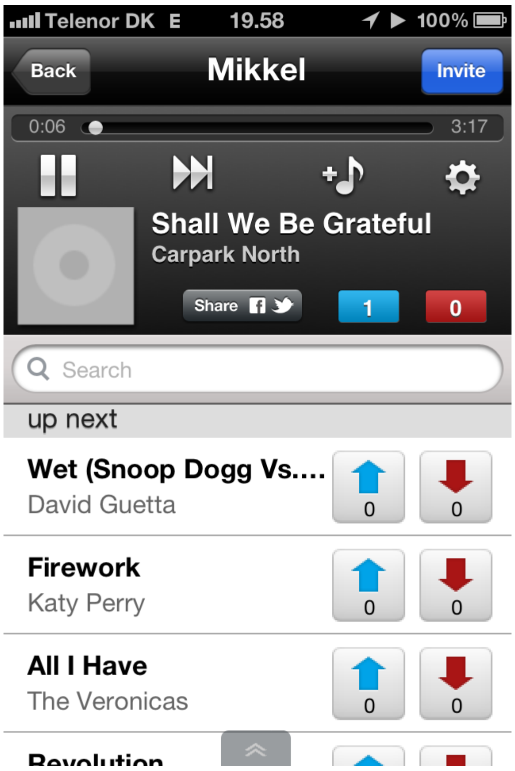

Figure 1.1: TuneTug native application

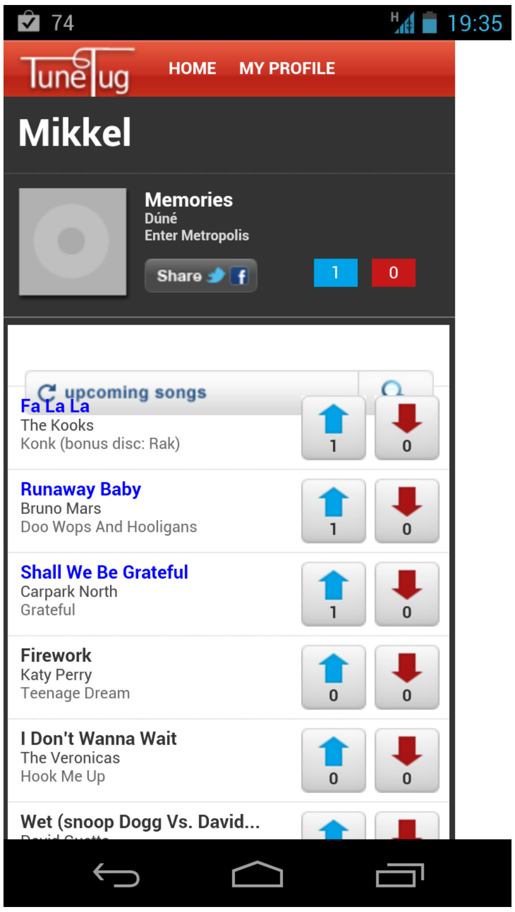

Figure 1.2: TuneTug web application

1.2.2 djtxt

djtxt about their service:

> ”Throwing a party? With djtxt you can set up a crowdsourced party playlist in one click. Your party guests send in song requests by text message, and djtxt plays their tunes. You'll see who added each song, and you can look at a recap of the playlist after the party.” [2]

djtxt is a webservice where the musical choice part of their service is done through either Twitter, email or SMS. Instead of voting, songs are played if requested by an attendee. djtxt uses Grooveshark, a webservice used to stream music from a large collection of music. To start streaming music from Grooveshark, the host has to use a unique bookmarklet for his party (created by djtxt), which creates an overlay on Grooveshark's website. The users can then send messages to djtxt, which then handles song requests and adds the requested songs to a Grooveshark playlist. If the playlist is empty at some point, djtxt automatically adds a random song, to make sure that there's always music playing.

The pros and cons of djtxt will now be analyzed, and an overview can be seen in Table 1.2.

Since djtxt can be controlled through SMS, email and Twitter messages, it's very platform independent. However, to use SMS, the party must be a premium party, which costs \$2 an hour, and does only work in the United States. Since the service uses Grooveshark to play the music, they get access to the complete Grooveshark library, which holds a lot of music. However, the service will also suffer from whatever problems Grooveshark might have. Grooveshark have been under legal problems, and has even been blocked by many ISPs (Internet Service Provider), even in Denmark [3]. This means that if Grooveshark has been blocked, djtxt will not work.

Pros	Cons
Platform independent	Requested songs are always played
Requested songs are always played	Basically Grooveshark
All of Grooveshark's music can be played	Songs requests can be misinterpreted by service
Everyone can participate	
Easy to use	

Table 1.2: djtxt pros & cons

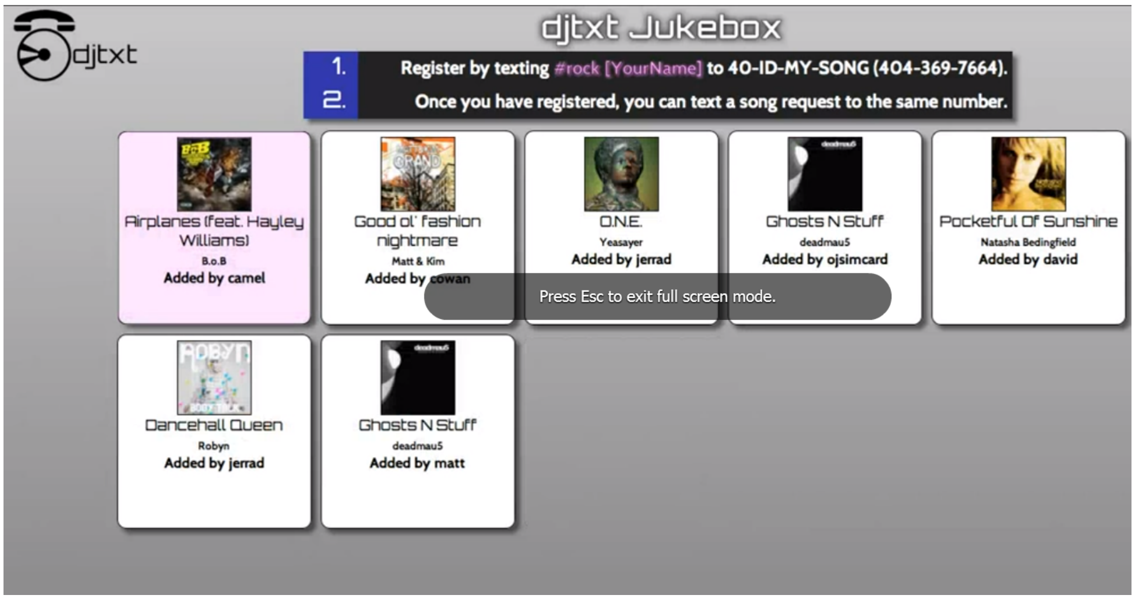

Figure 1.3: djtxt Grooveshark overlay [2]

1.3 Target Group

In this section we will shortly describe our target group, and then continue on describing the informants we have used for our project.

The target group for our service will primarily be young people, between the age of 15 and 40. We aim for this group, as they are the age group with the most smartphones [1], and the group which is most often found at parties (we assume). It would mainly be private party arrangers who would use our service, however these parties should not involve a DJ (Disc Jockey), as the application wouldn't be of any use to a DJ. We will use informants who are in our target group to help us develop a proper solution. We decided not to use quantitative questionnaire as we preferred having detailed answers from a few people, rather than short answers from a lot of people.

1.3.1 Informants

We will now describe the informants we used to help us develop a solution. We will describe each informant by their demographics and by their needs. We used their needs to help determine which area is most problematic for our informants, and hence our target group. As our target group is rather broad, we decided to go with one from each end, that is a teenager/young adult, and an adult. Besides using these informants to help determine the problem, we interviewed each informant several times, each time testing a new version of our solution. The result of these tests can be found in Chapter **??**. The following information about the informants was gathered at the first test.

Ida Gaarde Simonsen

- Name: Ida Gaarde Simonsen
- Age: 16
- Party frequency: 1-2 times a month
- Party size: 30 to several hundred, typically more than 30

Our first informant is Ida. Ida is a teenage girl, attending a Danish high school. She attends parties a couple of times a month, most of them are high school parties. These parties are usually larger than 30 people, and sometimes up to a few hundred people. The music is played by a computer using Spotify, where they usually make a playlist beforehand and then plays this offline playlist. She rarely experiences people changing song before the current song is over. She doesn't see it as a big problem. She liked the idea of not having to get up (and go to the computer) to change song.

Rikke something

- Name: Rikke something...
- Age: 30+
- Party frequency: Maximum once a month
- Party size: 12-15

Our second informant is Rikke. Rikke is an adult, working as a volunteer at Skråen, which is an music association that arranges concerts, shows and events. She doesn't go to parties as frequently as most young people, but she still goes rather often. At the parties she attends they are 12-15 people attending. At these parties they often use iPod or iPad connected to speakers to play the music, and sometimes they use playlists on PCs. She often experiences people changing music while the current song is playing, and finds it annoying, especially when it's a song she wants to hear that is being skipped. An interesting note from this interview is that she said if the music stops, the party stops. In order for a party to take place, a constant flow of music is needed.

1.4 Research Question

After researching the problem in context and looking at existing services, and which methods that are used to solve this, we will define a research question. The interviews with the informants showed that there was an interest in such a system being developed, and that the specific problems were that the music was changed in the middle of a song, and also that the music was not in the favor of the majority.

We can then define the research question:

> *How can we develop a system which can help people at private parties listen to the music that they want to hear, without interruptions, in an easy way?*

Chapter 2

The System

In this chapter we will describe our system through various methods. We will start with a system definition in Section 2.1, to get a quick overview of what functions the system has, who is involved, the conditions for the system, what technology have been used to develop and use it, and what the purpose of the system is.

The system is a party music player, with a dynamically generated music playlist, where each attendee can vote for their favourite music, and save each song to a favourite list for easy access later. This is all done through a mobile application, while a webservice will administrate the music, and an external webservice will play the music through the website. The system is to be used by users with very different experience in using mobile applications and websites. We will elaborate more on the architecture of the system in Section 2.2.

2.1 System specifications

In this section we define the specifications for our system, and elaborate as to why these were chosen. Our specifications are defined by *hard constraints* and *soft constraints*. Hard constraints are constraints that must be complied, as the system would not function without them. The soft constrains are luxuries, however they still matter a great deal for the product's functionality. This is a subjective matter, however this is what we feel is important.

Hard constraints:

- Music must not stop/be interrupted
- Have a library of music
- Have a playlist for music
- Add music to playlist
- Being able to vote up music
- Being able to play music
- The highest voted music is played first

The very most important constraint, which is the basic idea of the research, is to develop a system which keeps the music playing without interruptions, implying that the majority will get to listen to the kind of music they like. We feel that if we are to produce a system, these are the hard constrains which are a must for the system to have any end user value.

Having a library of music allows, users of the system the ability to find music they like, same principle as going to a library and picking out the book you want. Having a playlist is also important. The playlist's function is to serve as a list of music that will be played in some near future, perhaps not in the order of which the music numbers are listed, however playing the highest voted song first. Thus the functionality of playing, voting and having the highest voted music first is necessary.
Soft constraints:

- User uniqueness functionality (unique user login)
- A favorites list
- A list of previously played songs
- Connection with social media
- Party check-in
- Administrator permissions e.g. play/stop/pause/skip music
- Platform independent

User uniqueness functionality is very useful. It provides us with the ability to keep track of each user and have our system act differently depending on the user. The idea of a favorite list provides an arbitrary user the ability to quickly add his favorite songs to a playlist, while the idea of the previously played songs list is a way to make sure a given user can see the songs which have been played, and possibly favorite them. Social media connectivity is a neat feature, given how popular social services are in this era, e.g. Facebook, Twitter, Spotify etc. The system could for example be integrated with Facebook's events, giving a Facebook user the ability to post his party information on Facebook. Administrator permissions is rather contradictory with the first hard constraint of "Music must not stop/be interrupted" however, we think it is necessary for the administrator of the party being able to skip music. After all it's his party, and stopping the music may be needed. Having a platform independent system would mean that we could reach a larger amount of people, allowing everyone at a party to use the system, and not just a few.

2.2 Architecture

In this section we will show and elaborate on the basic architecture of the system.

The system is split into 5 different components:

- Internet
- Web server
- Smartphone
- External music library
- Computer

Figure 2.1 is a visual network diagram representation of the system and its components. A component in square, implies that there can be multiple units of it in the system. The database is a subcomponent of the web server, i.e. the web server hosts the database. The speakers are also a subcomponent of each computer, it is not interesting for this project, but it's important to note its existence in the system. We will now go through every component and talk about its use in the system.

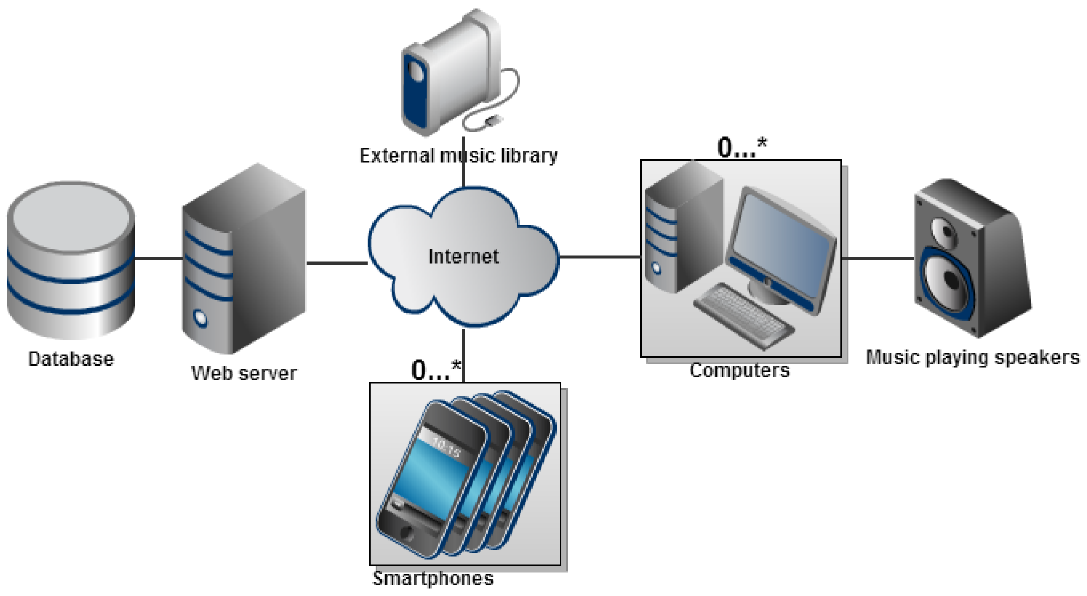

Figure 2.1: System architecture

Internet

The Internet is a very abstract component and is not very interesting for our system. The component allows for easy interaction in between components without direct connection, and because of this it's a very attractive connection interface.

Web server

The web server hosts a website and the subcomponent; the database. The website is used by the host to take care of creating parties and setting up the party and it's music player. The database is the storaging entity of the system. In the database all information is stored e.g. party information, music playlists. The database hosts information for each party.

Smartphone

The smartphone is an important part of the system. Party attendees can manipulate and interact with the party's music directly through the smartphone. The smartphone is mainly used for adding and voting up songs, it also has other less important features. This component uses internet connection and has a direct interaction with the webserver. The smartphone's internet connection is mainly used to change or receive database data. The adding feature of the smartphone searches in the external music library and adds the song picked by the user to the party's playlist.

External music library

The external music library is quite essential for the system. The library provides us with an almost complete music library and easy to usi API solution. Using and external library allows us to focus on what we think is the more important parts of the system. As seen on Figure 2.1 the external music library is connected to through the internet, allowing easy interaction with the system.

Computer The computers' simple task, is taking care of playing the music. The computer should probably only be used to set up the music player and the system will, take care of playing the chosen music. The computer plays the music from the external music library.

2.3 Problem-Domain Analysis

In the section, we analyze the problem-domain, to look for requirements for our system. We do this to model our problem domain. We start by identifying classes in Section 2.3.1 which will result in an event table for our problem domain. In Section 2.3.2 we describe relations between the classes in our system in a class diagram. We then end our problem-domain analysis by describing the behavior and attributes of objects and classes in Section 2.3.3.

2.3.1 Classes

To model our problem-domain, we start by identifying classes and events in our system. We define a class to be a collection of objects. These objects share attributes, behavior and structure. An event is an incident involving one or more of these objects. We put our classes and events in an *event table*, to get an overview of the classes and their events. The result of this can be seen in Table 2.1. An X in the table indicates that the objects from this class are involved in the corresponding event.

	Classes					
Events	Host	Party	Attendee	Playlist	Song	Vote
User created	X		X			
Logged in	X		X			
Party created	X	X				
Party joined		X	X			
Music added	X		X	X	X	
Music voted	X		X		X	X
Music played				X	X	
Playlist sorted				X	X	
Party ended	X	X				

Table 2.1: Event table

To create a better understanding of the event table, we will now describe the connection between the classes and events.

User created
Whenever a user is create, he/she can be either a host for a party, or an attendee of a party

Logged in
A user can either log in as host or attendee

Party created
Whenever a host creates a party, a new party object is made

Party joined
Attendee can join a party created by a host

Music added
Both the host and the attendee can add a song to a playlist

Music voted
: Both the host and the attendee can add a vote to a song

Music played
: Whenever a song is done playing on the playlist, a new song will be played, and the previous played song will be added to another playlist containing already played songs

Playlist sorted
: When a song is about to end, the playlist will be sorted to get the most popular song to be played next

Part ended
: A host can end a party

2.3.2 Structure

We will now take the classes from Section 2.3.1, and add structural relations between them. We will make an *object-oriented structure*, and describe the relation between the classes with three different structures: *Generalization*, *aggregation* or *association.*

Generalization The generalization structure describes a relation between two or more specialized classes, and a super class. The specialized classes (subclasses) inherit properties from the superclass. We show a generalization relation by a $\triangle$, and we express the relation as a "is-a" relation.

Aggregation The aggregation structure is a relation between two or more objects. We show multiplicity by showing a number next to the object. A number shows the amount of objects each object is related to. An * means that we have "many" objects related to it. "..." means that we have *x to y* objects. For example "0..*" describes zero to many. The aggregation relation expresses a "has-a", "is-part-of" or "is-owned-by" relation, and we show this relation with a $\Diamond$.

Association The association structure describes a relation between objects, much like aggregation, but does not define any properties of the objects. We express the association relation by "knows" or "associated-with", and we show this by a simple line. As with the aggregation, we can also have multiplicity between the related objects.

The result of these structural relations between the classes, can be seen in Figure 2.2.

As you can see in Figure 2.2, the classes are almost identical to the ones from Table 2.1, except that we have an *abstract* class "User". The User class is written in italic, to show that it's an abstract class, and cannot be initiated. The users we create are either of the class Host, or Attendee. We do this to show that Host and Attendee shares the association with Song and Vote. The User class, and hereby also the Host and Attendee classes, have an association with the Song and Vote classes. This is because a User is able to vote on a song, and can also add songs. The Song class aggregates from the Playlist class, as a playlist is just a list of songs. At the top we have the Party class, which is a associated with a host, has zero-to-many users and one playlist. The playlist has zero-to-many songs in it, and each of these songs has one-to-many votes.

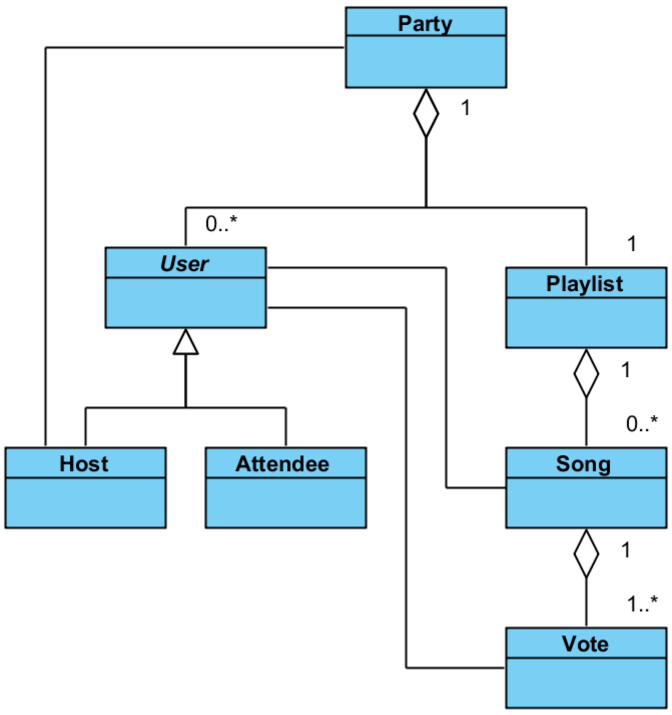

Figure 2.2: Class Diagram

2.3.3 Behavior

To describe the behaviour of key classes in more detail, we need to make a behaviour pattern analysis of each key class from Section 2.3.1, which we have done with statechart diagrams.

The first class to be analyzed is the Party class as seen in Figure 2.3. In Figure 2.3

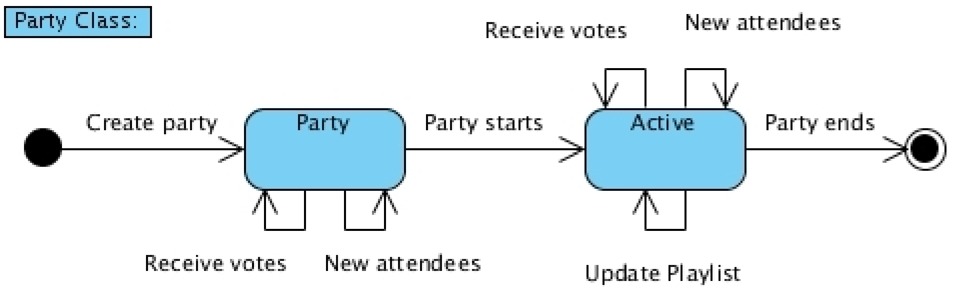

Figure 2.3: Party Class

the Party starts with being created. The party then continually receives votes from the attendees, and registers newly added guests for the party. When the party starts, the Party class changes state, as it becomes active. In this state the party class still receive votes from attendees and register newly added guests, but now it also sends the information about the votes to the playlist, thus making the playlist active. All of these events happens continually until the party is ended.

The next class to be analyzed is the Playlist class as seen in Figure 2.4. In Figure 2.4 the Party starts with becoming active, thus changing the state of the Playlist class to active. While the Playlist class is active, votes are received from the users, and also adds

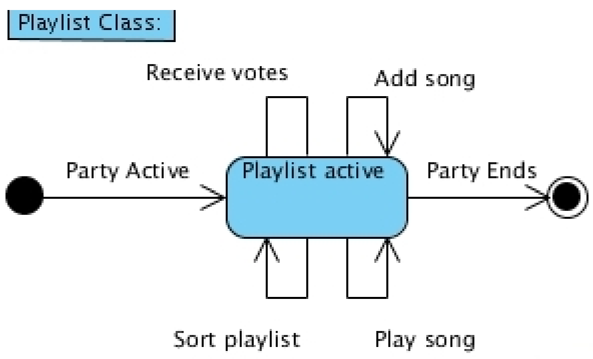

Figure 2.4: Playlist Class

new songs to the playlist. The playlist is also being sorted, based on how many votes each song has. The song with the highest amount of votes, when the previous song ends, is the next song to be played. All of these events happens continually until the party is ended.

Another important class in our system to be analyzed is the User class as seen in Figure 2.5. In Figure 2.5 the User starts with being created and then becomes active.

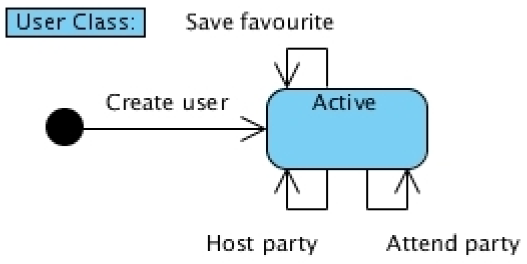

Figure 2.5: User Class

When active, the user can either attend or host a party, and add songs to their favourite list. All these event can occur as many times as the user wants, performing the events continually.

2.4 Application-Domain Analysis

2.4.1 Usage

2.4.2 Functions

2.4.3 Interfaces

2.5 Architectural Design

2.5.1 Criteria

2.5.2 Components

2.5.3 Processes

Bibliography

[1] Anson Alexander. Smartphone usage statistics 2012. Website, January 2012. http://ansonalex.com/infographics/smartphone-usage-statistics-2012-infographic/. 4

[2] djtxt. http://djtxt.me/]. vi, 3, 4

[3] Andrew Orlowski. Grooveshark blocked in denmark by copyright warriors. Online Article, February 2012. http://www.theregister.co.uk/2012/02/23/grooveshark_blocked_in_denmark/. 4

[4] Spotify. Website. http://www.spotify.com/dk/get-spotify/premium/. 2

[5] TuneTug. Website. http://tunetug.com/about.html. 1