

**FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO**



**FEUP** FACULDADE DE ENGENHARIA  
UNIVERSIDADE DO PORTO

# **Real-time interactive music remixing app**

**Pedro Richard Branco Barreira e Silva Kretschmann**

MASTER IN ELECTRICAL AND COMPUTERS ENGINEERING

Supervisor in FEUP: Rui Penha, PhD

Supervisor in INESC TEC: Matthew Davies, PhD

July 28, 2015

© Pedro Richard Branco Barreira e Silva Kretschmann, 2015

A Dissertação intitulada

“Real-Time Interactive Music Remixing App”

foi aprovada em provas realizadas em 28-07-2015

o júri

Presidente Professora Doutora Ana Cristina Costa Aguiar

Professora Auxiliar do Departamento de Engenharia Eletrotécnica e de  
Computadores da Faculdade de Engenharia da Universidade do Porto

Professor Doutor Rui Pedro Pinto Carvalho Paiva

Professor Auxiliar do Departamento de Engenharia Informática da Faculdade de  
Ciências e Tecnologia da Universidade de Coimbra

Professor Doutor Rui Luis Nogueira Penha

Professor Auxiliar Convidado do Departamento de Engenharia Informática da  
Faculdade de Engenharia da Universidade do Porto

O autor declara que a presente dissertação (ou relatório de projeto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extratos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são corretamente citados.

Autor - Pedro Richard Branco Barreira Silva Kretschmann

Faculdade de Engenharia da Universidade do Porto



# Abstract

This dissertation presents *rebeat*, an interactive music remixing application for handheld devices that enables non-expert users to experiment in real-time mashup creation. Due to recent advances in Music Information Retrieval (MIR) and the increase in the computational power of computer systems, academic researchers have been developing applications oriented towards interactive and real-time music mashup creation. Although these systems typically provide music matching functionality to aid a user's creative process, they rarely go beyond the proof of concept stage and are therefore not available for users to experiment with. Outside of the research community some commercial applications for user-guided mashup experimentation have been developed for smartphones and tablets. While these mobile applications often offer pleasant and user-friendly user interfaces they are quite limited in terms of the mashup possibilities, e.g., by restricting users to pre-stored music content only, without the opportunity to use their own music collections.

The proposed system, *rebeat*, was developed using the real-time programming environment *Pure Data*, wherein MIR techniques were used to develop and implement music understanding functions to allow user-guided music mashup manipulation, as well as higher-level algorithms, including harmonic measures to implement a model of transition compatibility for intelligent music sequencing. The system was designed to offer a quick and intuitive method of pattern creation – where patterns consist of groups of four beat slices spliced together. *rebeat* also allows for long-term sequencing of these patterns to incorporate higher level temporal structure into the created mashups. Using *MobMuPlat*, a standalone application that runs on Android and iOS mobile devices, the system was implemented with a friendly graphical user interface which allows intuitive touch control for all operations. For development and experimentation, a dataset of beat-synchronized music excerpts was prepared.

The *rebeat* system was evaluated via a listening experiment, where 38 participants rated examples of sequences of patterns generated by the different pattern creation algorithms. The results of the evaluation showed that users preferred the patterns created using the proposed intelligent sequencing methods rather than those made entirely at random.



# Agradecimentos

Em primeiro lugar quero agradecer à minha mãe, Maria Alzira Kretschmann. Por toda a paciência, apoio e ajuda desde sempre, dedico-te esta dissertação.

Estendo o agradecimento aos meus irmãos, à minha avó, aos meus tios e aos meus primos. Sempre se demonstraram solidários durante este percurso e nunca falharam com uma palavra amiga.

Aos orientadores desta dissertação, um agradecimento muito especial. Ao Matthew Davies, agradeço pela oportunidade de trabalhar numa área que realmente gosto e por todo o apoio incansável durante estes meses, foi sem dúvida uma peça fundamental na concretização deste projeto. Ao Rui Penha, agradeço pela prontidão em ajudar e por todo o apoio prestado ao longo dos meses.

A todos os elementos do Sound and Music Computing Group, agradeço por me fazerem sentir em casa e por toda a ajuda e conselhos partilhados ao longo destes meses. Agradeço em especial ao Gilberto Bernardes por basicamente me ter ensinado uma nova linguagem de programação.

Em especial por me terem apoiado durante a concretização deste projeto, agradeço ao Pedro Pacheco, Ricardo Rocha, João Machado, Hugo Melchior, Duarte Silva, Marta Rolo, Tiago Baldaia e Edgar Couto. Querendo evitar encher a página de nomes, agradeço a todos os meus amigos que me acompanharam durante todos estes anos. Nos bons e nos maus momentos, nas conquistas e nas derrotas, a festejar ou a trabalhar, fica o meu eterno agradecimento por todas as experiências que partilhamos. Estes foram os meus anos de ouro por vossa causa.

Obrigado por tudo.

Pedro Kretschmann



*“Music is the one incorporeal entrance into the higher world of knowledge  
which comprehends mankind but which mankind cannot comprehend.”*

Ludwig van Beethoven



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Objectives . . . . .	2
1.3	Dissertation Structure . . . . .	3
<b>2</b>	<b>Background and State of the Art</b>	<b>5</b>
2.1	Computer music and mixing . . . . .	5
2.1.1	Computer music . . . . .	5
2.1.2	The DJ as a performer . . . . .	5
2.1.3	Mashup technique in music creation . . . . .	6
2.1.4	Existing tools oriented for live development . . . . .	7
2.1.5	Existing tools oriented for studio development of mashups . . . . .	10
2.2	Music Analysis, processing and manipulation . . . . .	12
2.2.1	Programming languages for creative applications . . . . .	12
2.2.2	Music information retrieval . . . . .	13
2.2.3	Digital signal processing techniques . . . . .	15
2.3	References to develop a new application . . . . .	16
2.3.1	Existing systems oriented for real-time development . . . . .	16
2.3.2	Conclusions and motivation for project . . . . .	25
<b>3</b>	<b>Project Specification</b>	<b>27</b>
3.1	Overview . . . . .	27
3.2	Music collection pre-processing tools . . . . .	28
3.3	Framework development tools . . . . .	28
3.4	Graphical user interface design tools . . . . .	29
<b>4</b>	<b><i>rebeat</i> System</b>	<b>31</b>
4.1	Overview . . . . .	31
4.2	Music Dataset Preparation . . . . .	31
4.3	Methods of Pattern Creation . . . . .	33
4.3.1	Patterns . . . . .	33
4.3.2	Algorithms of Pattern Creation . . . . .	37
4.4	Interface . . . . .	44
4.4.1	Overview . . . . .	44
4.4.2	Load Panel . . . . .	45
4.4.3	Mashup Panel . . . . .	51

<b>5 Evaluation</b>	<b>59</b>
5.1 Pattern creation methods . . . . .	59
5.1.1 Experiment design . . . . .	59
5.1.2 Results . . . . .	64
<b>6 Conclusions</b>	<b>69</b>
6.1 Contributions . . . . .	69
6.2 Future Work . . . . .	70

# List of Figures

2.1	Pionner's audio player <i>Professional Multi Player CDJ-2000NXS</i> . . . . .	8
2.2	Pioneer's mixing board <i>Professional DJ Mixer DJM-2000NXS</i> . . . . .	8
2.3	Screenshot from Serato's DJ software <i>Serato DJ 1.7.5</i> . . . . .	9
2.4	Screenshot from Native Instruments DJ software <i>Traktor Pro 2</i> . . . . .	10
2.5	Screenshot from <i>FL Studio 11</i> , a DAW from Image-Line. . . . .	11
2.6	Screenshot from <i>Logic Pro X</i> , a DAW from Apple Inc. . . . .	11
2.7	Screenshot from Ableton's DAW, <i>Ableton Live 9</i> . . . . .	12
2.8	Screenshot of <i>Beat-Sync-Mash-Coder</i> user interface. . . . .	17
2.9	Screenshot of <i>AutoMashUpper</i> user interface. . . . .	18
2.10	Screenshot of <i>Improvasher</i> patch for <i>Max</i> . . . . .	19
2.11	Diagram of <i>Improvasher</i> real-time mashup system. . . . .	20
2.12	Screenshots of <i>Audio Mashup Pro</i> user interface – part 1. . . . .	21
2.13	Screenshots of <i>Audio Mashup Pro</i> user interface – part 2. . . . .	21
2.14	Screenshots of <i>Mashup</i> user interface. . . . .	22
2.15	Screenshots of <i>iMash</i> user interface. . . . .	23
2.16	Screenshots of <i>Mashupper</i> user interface – part 1. . . . .	24
2.17	Screenshots of <i>Mashupper</i> user interface – part 2. . . . .	24
3.1	System model . . . . .	27
3.2	Screenshot of work environment during a project in <i>FL Studio 11</i> . . . . .	28
3.3	Screenshot of an audio file sampler project in the programming environment of <i>Pd</i> . . . . .	29
3.4	Screenshot of <i>MobMuPlat Editor</i> work environment. . . . .	30
4.1	Screenshot of <i>FL Studio 11</i> where 4 different music samples are being spliced together resulting in a music excerpt with 16 bars length (32 seconds). . . . .	33
4.2	A pattern is composed by four beat slices spliced together from two different songs. Theoretically, these beat slices can be from anywhere in the song, however there are several strategies for choosing them. . . . .	34
4.3	Screenshot from the <i>rebeat Pd</i> framework: <i>Pd</i> compiler displaying information regarding the creation of one four beat long pattern. . . . .	35
4.4	Example of beat slicing and splicing method for patterns with 4 beats length. . . . .	36
4.5	Example of a pattern created with Algorithm 1 – Baseline Random Transition Model. . . . .	37
4.6	Example of a pattern created with Algorithm 2 – Metrical Structure Based Transition Model. . . . .	38
4.7	Screenshot from <i>rebeat's Pd</i> framework: implementation of Algorithm 2. . . . .	39
4.8	Screenshot from <i>rebeat's Pd</i> framework: implementation of the probability threshold slider. . . . .	40

4.9	The best match per beat is selected based is the minimum cosine distances between beats. . . . .	42
4.10	Example of a pattern created with Algorithm 3 - Chroma Based Transition Model. . . . .	43
4.11	Example of a pattern created with Algorithm 4 - Chroma Based Transition Model. . . . .	44
4.12	<i>rebeat</i> GUI: “Panel 1” or “Load Panel” . . . . .	45
4.13	Screenshot from Panel 1 – “OPEN”, “PLAY” and “STOP” buttons, and displays the calculated values for the “Duration” and “Beats”. . . . .	46
4.14	Screenshot from <i>rebeat</i> ’s <i>Pd</i> framework: “load section 1” with respective “Play” and “Stop” functions. . . . .	46
4.15	Screenshot from <i>rebeat</i> ’s <i>Pd</i> framework: list of the available audio files in “load station 1”. . . . .	47
4.16	Screenshot from <i>rebeat</i> ’s <i>Pd</i> framework: “VOLUME Master”, “VOLUME 1”, “VOLUME 2”, “Play Both” and “Stop Both”. . . . .	48
4.17	Screenshot from Panel 1 – “Filter”, “3-band Equalizer” and “Volume 1” sliders. In this example, the high and mid-range frequencies of the audio ouput are being attenuated. . . . .	49
4.18	Screenshot from <i>rebeat</i> ’s <i>Pd</i> framework: “DJ-style 3-band Equalizer 1”. . . . .	50
4.19	Screenshot from <i>rebeat</i> ’s <i>Pd</i> framework: biquadratic filters implementation in the “DJ-style 3-band Equalizer 1” . . . . .	50
4.20	Screenshot from <i>rebeat</i> ’s <i>Pd</i> framework: high-pass/low-pass filtering slider. . . . .	51
4.21	<i>rebeat</i> GUI: “Panel 2” or “Mashup Panel” . . . . .	52
4.22	Screenshot from Panel 2: “Create ALL” button. . . . .	53
4.23	Screenshot from Panel 2: “Final Mahup Grid” with a sequence ready to be played. . . . .	53
4.24	Screenshot from Panel 2: “NEW” and “PLAY” buttons, and “pattern size” buttons and display from “Pattern 1” section. . . . .	53
4.25	Screenshot from Panel 2: “Final Mashup Grid”. . . . .	54
4.26	Screenshot from Panel 2: “Pre-defined sequences”, “Clear All” and “PLAY” buttons. . . . .	54
4.27	Screenshots from Panel 2: “Final Mashup” grid with the “Pre-defined sequence” 1 (A), 2 (B) and 3 (C) selected. The “time indicator” flash indicates which pattern was playing when the screenshot was taken. . . . .	55
4.28	Illustration of how the system plays the sequenced patterns with the correspondent beat and time progression. . . . .	56
4.29	Screenshot from Panel 2: When pressing the “NEW” button in “Pattern 1”, its pattern colour (blue) gets highlighted. . . . .	57
4.30	Screenshot from Panel 2: “Equal Pattern Size” buttons and “patterns size” display. . . . .	57
4.31	Screenshot from Panel 2: “Activate Harmonic Transition Model” button. . . . .	58
4.32	Screenshot from Panel 2: “Probability Threshold” slider; “Music 1” and “Music” flash indicators. . . . .	58
5.1	Screenshot of <i>rebeat</i> mobile app with the above features selected. . . . .	61
5.2	Experimentation test: screenshot of the “Test” page on <i>rebeat</i> webpage. . . . .	62
5.3	Experimentation test: screenshot of the consent confirmation popup. . . . .	62
5.4	Experimentation test: screenshot of the musical background question. . . . .	63
5.5	Experimentation test: screenshot of a sequence rating question. . . . .	64
5.6	Experimentation test: average algorithms evaluation graphic. . . . .	65
5.7	Experimentation test: average evaluation per pairs of songs graphic. . . . .	67
5.8	Experimentation test: percentage of ratings per algorithm graphic. . . . .	68

# List of Tables

4.1	Relation between the dataset music excerpts reference values, assuming audio sampled at 44100 Hz. . . . .	32
4.2	Different pattern sizes, assuming audio sampled at 44100 Hz. . . . .	37
4.3	Relation between a natural beat progression through an audio file and the respective beat progression within each bar, assuming 120 BPM and audio sampled at 44100 Hz. . . . .	38
4.4	Example of a valid sequence, i.e., one pattern per time block (column). . . . .	54
4.5	Minimum and maximum durations available in the final mashup composition grid.	55
5.1	List of algorithms submitted to the experimentation test. . . . .	60
5.2	Experimentation test: pattern creation features. . . . .	60
5.3	Experimentation test: long term composition. . . . .	60
5.4	List of sound examples recorded with the parttern creation algorithms. . . . .	61
5.5	Experimentation test: weight attributed to each rating option. . . . .	64
5.6	Experimentation test: average algorithms evaluation results. . . . .	65
5.7	Experimentation test: average evaluation per pairs of songs. . . . .	66
5.8	Experimentation test: percentage of ratings per algorithm. . . . .	67



# Abbreviations

BPM	Beats Per Minute
CD	Compact Disc
DAW	Digital Audio Workstation
DJ	Disc Jockey
DSP	Digital Signal Processing
EDM	Electronic Dance Music
FFT	Fast Fourier Transform
GUI	Graphical User Interface
HPCP	Harmonic Pitch Class Profile
MIDI	Musical Instrument Digital Interface
MIR	Music Information Retrieval
OSC	Open Sound Control
<i>Pd</i>	Pure Data
STFT	Short Time Fourier Transform



# Chapter 1

## Introduction

### 1.1 Motivation

“A computer can be programmed to play “instrumental” music,  
to aid the composer, or to compose unaided.”

M. V. Mathews [8]

The continuous development of technology in music related software and hardware has made it easier for musicians and producers to perform precise music manipulation in the studio or even in live performance. With Digital Audio Workstations (DAW) it is possible to create an entire piece of music from scratch or manipulate existing songs in order to create new ones. The manipulation technique most relevant for the scope of this work is known as music *mashup* and it consists of locating desired points in two or more songs, and splicing or mixing them together. The live performance of this technique by disc jockeys (DJs) using digital/analog turntables and software has been socially and commercially accepted, aided by the continuous growth of electronic music enthusiasts and music producers. A good example is the artist Girl Talk<sup>1</sup>, an american DJ with background in engineering known for his work with mashup-style remixes and digital sampling.

The software (DAWs) and hardware systems (turntables and controllers) that allow mashup creations require some expertise to control the output considering that a coherent beat and harmonic synchronization is desired. Although it may seem easy for experienced performers, in many ways it is considered to be fairly hard for amateurs or first time users. In order to bring this new form of music creation closer to the common user, some systems have been developed to perform automated and real-time sequencing (cut and splice) to create mashups. Examples are the *Beat-Sync-Mash-Coder* [6], the *AutoMashUpper* [3] and the *Improvasher* [5], being systems where the user only needs to choose the desired songs or input to mashup and select some basic features like final tempo. These systems allow users, with little to no experience, to perform

---

<sup>1</sup><http://www.allmusic.com/artist/girl-talk-mn0000660155/biography>

creative real-time mashups but the output may not always be pleasing, considering that possible limitations in the models on musical compatibility might negatively impact the music results of the mashup.

It may even take years to know how to work with mashup related systems like DAWs and turntables in order to manipulate music with pleasing results, but nowadays, through music technology, we can give everyone access to the pleasure of music making, even to people who have a keen interest for music but didn't had the time or opportunity to study it or to pursue a musical career. Considering the increasing development of handheld systems in the past few years, people have developed a tendency of centering their time for recreational activities on their smartphones or tablets due to the large variety of applications offering all kinds of experiences, including music.

Currently there is no mashup tool which is easy to use and implements an intelligent model for mashup creation, and is available for handheld devices. With that in mind, this project proposes a system for mobile devices in which, one app with a friendly GUI (Graphical User Interface) can implement some DAW's functionalities and some logical guidance to perform pleasing mashups. This kind of app may help encourage inexperienced users to engage with and experiment in mashup creation and perhaps increase their personal musical interests.

## **1.2 Objectives**

This dissertation was developed within the Sound and Music Computing Group (SMC Group) at INESC TEC.

The main objective of this project was to develop an interactive music remixing app for handhelds that enables non-expert users to experiment in real-time mashup creation with music excerpts using a GUI.

In order to meet the main goal, the objectives set for this project were divided in two phases and described as follows.

### **Preparation Phase**

- Background learning and academic familiarization with:
  - Electronic music impact;
  - Mashup and remix culture;
  - Existing real-time and in-studio mashup systems;
  - Real-time Digital Signal Processing (DSP) and Music Information Retrieval (MIR) techniques .
- Identify open problems that justify the creation of a new system.

### **Development Phase**

- Develop a back-end algorithm to perform music analysis and to enable user-guided real-time mashup creation.

- Design and implement an interactive GUI to allow intuitive experimentation of the system with a mobile device.
- Develop higher-level functions to implement intelligent sequencing in pattern creation.
- Perform a user-driven evaluation of the system to determine its effectiveness and to retrieve information about the user's preferences.

### 1.3 Dissertation Structure

The remainder of this dissertation is organised as follows:

- Chapter 2: Background and State of Art  
Describes and analyses mashup culture and the software systems available nowadays. Refers also to relevant DSP and MIR techniques.
- Chapter 3: Project Specification  
Describes the tools, specifications and procedures that were used to develop and implement the proposed system.
- Chapter 4: *rebeat* System  
Presents all the decisions and implementations that resulted in the *rebeat* system. The main interaction possibilities and functionalities of the GUI are also described.
- Chapter 5: Evaluation  
Describes the listening experimentation test performed in order to evaluate the pattern creation algorithms.
- Chapter 6: Conclusions  
Presents the dissertation contributions and presents areas for future work.



# **Chapter 2**

## **Background and State of the Art**

### **2.1 Computer music and mixing**

#### **2.1.1 Computer music**

Computer music can be characterized by the application of computational technology into music composition and production [16]. The fast development in computer technology affected the way we compose and create music and, nowadays, computer music can be found in every artistic presentation like theatre, movies or television, and in any multimedia platform with audio content. Its popularity is mostly due to the developments and innovations in music manipulation techniques used in studio and in live performance [16].

While computer music has been performed in academic research for many years, most notably, Max Mathews, who started developing computer music in the 1950's [8], the reality of the 21<sup>st</sup> century is that the availability of accessible software music tools has given rise to a computer music culture outside these circles. New kinds of innovating experiences in electronic music manipulation are being made by self-taught artists in home studios all over the world [19].

#### **2.1.2 The DJ as a performer**

A DJ's primary job can be put in simple words: perform a selection of music and play it to a crowd using some kind of software and/or hardware [2].

When dance clubs began to spread and DJs made their ways into nightlife and parties, it was sufficient to simply play one song after another. The ability of the performer to read the crowd's reaction and choose the next song was enough to please the audience and this social acceptance began to shape the 21<sup>st</sup> century music panorama. Audiences began embracing a new performance realm that emerges from recording technology. The DJ therefore creates his own musical selection in his live performance, resulting in different audiences from different genres seeking DJs with similar music tastes or tendencies. [2]

Although the “choose-and-play” type of DJing was once enough, there were some branches that evolved into creating complex and expressive art forms. Evaluating the DJ performance, there are two types of DJs: the *scratch*<sup>1</sup> DJ and the mix DJ. *Scratch* DJs are recognized for employing a diverse repertoire of interactive techniques and skills and often use a cut and paste style of playing where songs switch abruptly through manual interaction. The live demonstration of these techniques were so acclaimed that championships have been disputed worldwide. An example is the *DMC World DJ Championship*<sup>2</sup>, an annual DJ competition hosted by Disco Mix Club (DMC) which began in 1985 and still endures today. In this competition DJs from all over the world show their skills of music selection of vinyl records using turntables and audio mixing boards.

On the other hand, mix DJs are more focused in sequencing different tracks through a creative and artful choice of material and expert, synchronized transitions [2]. This project focuses on one of these music mixing possibilities – *mashup* –, as described in the section 2.1.3.

### 2.1.3 Mashup technique in music creation

The art of combining different segments of music into a new piece of digital art is a phenomenon that has gained great interest in the music industry [6]. “Music mashups offer a way for users to re-engage with existing and familiar musical content by adding some extra, complementary musical components.” [3].

21<sup>st</sup> century producers and DJs are used to selecting and mixing music before the actual live performance. Using available DAWs they are able to cut, copy, splice, change, add or remove any kind of sound in any music, providing them a highly creative experience in which the electronic music composer or performer is usually known for mixing a particular signature genre or using specific sound tendencies. This new reality of digital music creation and manipulation made it possible for any person with a computer and a DAW to experiment with homemade creations. The number of electronic music producers began to increase in the last two decades and the music charts and online stores are now full of electronic music, especially sub-genres of Electronic Dance Music (EDM). Original compositions or manipulations of existing materials like remixes and mashups can be found everywhere. There are available websites like *Mashstix*<sup>3</sup>, designed to upload and share mashup creations, and websites like *CCMixter*<sup>4</sup> that provide multi-tracking recording for remixing music.

In electronic music mixing, one of the most practiced techniques is known as mashup. It can be made using a software like a DAW in a studio scenario, and it can be performed in live performances using for example CD players and a mixing board which provides sequencing tools that, along with the use of headphones allow the DJ to perform real-time mashups. More information about the currently available mixing tools in live performance can be found in section 2.1.4.

---

<sup>1</sup>Scratching is a technique used by DJs to produce distinct sounds by moving a vinyl record back and forward on a turntable while rapidly manipulating the crossfader on a mixing board.

<sup>2</sup><http://www.dmcchamps.com/about.php>

<sup>3</sup><http://www.mashstix.com/>

<sup>4</sup><http://ccmixter.org/>

In terms of studio-oriented mashup development (i.e., the use of software applications to perform music mashups), there are different strategies compared to live performance. Creating a mashup is a laborious and time-consuming creative process. The desired source material must be located, cut up, and spliced together into a new work of art. In order to create a mashup that can be defined as musically cohesive and pleasant, every piece or sound segment must be meticulously synchronized. There are actually two different methods to create studio-oriented mashups. The first consists in combining very small units of sounds into an automatic *audio collage* according to some user specified criteria. For example the user chooses the desired audio output, like an existing song, and the computer is responsible to choose samples and re-construct the waveform. This method offers limited control to the user and restricts the creative experience. The second method to create mashups is to have the user manually splice and layer the raw audio samples or music segments using a DAW. DAWs are powerful tools for music manipulation and to take full advantage of the system some experience and practice is required, considering that combining multiple tracks is a meticulous process with innumerable minuscule adjustments [6]. More information about the currently available mixing tools for studio-oriented development can be found in the section 2.1.5.

#### 2.1.4 Existing tools oriented for live development

Before the migration of music formats from vinyl records to compact discs (CD) and digital media (like MP3 or WAV), a DJ would require analogue tools to perform: vinyl records, a pair of turntables and an audio mixing board. This direct analog interface has survived until today, mainly because of its tight physical connection with music and the visual appeal that results from the DJ's interaction with the hardware [2]. In this reality DJs must invest a great amount of time and effort to perfect their skills with turntables.

In the 80s the CD reached the market and compared to vinyl format it was physically smaller and more practical, could store a greater amount of music or data and it was more immune to noise interference. The development of technology in audio compression and processing led to commercial developments in music's related systems. With the appearance of MP3 compression and USB storage devices (like flash drives), the audio players adapted and nowadays the digital oriented DJs have available a large variety of hardware systems like CD/MP3 players and digital mixing boards.

An example is the *CDJ*, a line of CD players developed by Pioneer Electronics since the 90s that allows analogue control of music from CDs or flash drives, usually using an emulated surface that simulates vinyl control. Their last model released, the *Professional Multi Player CDJ-2000NXS<sup>5</sup>* (see Figure 2.1) is one of the mainly used CD players in the market for DJ performances.

---

<sup>5</sup><http://pioneerdj.com/english/products/player/cdj-2000nxs.html>



Figure 2.1: Pionner's audio player *Professional Multi Player CDJ-2000NXS*.<sup>a</sup>

---

<sup>a</sup><http://www.pioneer.eu/eur/newsroom/news/CDJ-2000nexus/page.html>

Pioneer also develops mixing boards and their last model, the *Professional DJ Mixer DJM-2000NXS*<sup>6</sup> (see Figure 2.2), the ideal match for *CDJ-2000NXS* in DJ performance.



Figure 2.2: Pioneer's mixing board *Professional DJ Mixer DJM-2000NXS*.<sup>a</sup>

---

<sup>a</sup><http://www.pioneer.eu/pt/products/44/74/461/DJM-2000NXS/page.html>

In most EDM, the sub-genres (i.e., all kinds and variants of house, electro and drum and bass) are easily recognized for having specific beats per minute (BPM) that remain constant throughout

---

<sup>6</sup><http://pioneerdj.com/english/products/mixer/djm-2000nxs.html>

the entire song. Considering that most electronic music DJs are used to mixing a specific genre or group of sub-genres, they have the same flow of BPM during a performance which makes it less complex to obtain pleasant transitions between different songs. In these cases the DJ only is required to match the beats of the songs using the CD players/turntables or a software.

It has become common to see DJs using a laptop in live performances. With the evolution of computational capacity of computer systems it is possible to perform real-time music manipulation with software programs. “Laptop performance can be defined as live computer music using general-purpose computers, very often without the use of specialized hardware controllers. Its defining characteristic is an emphasis on the live use of software tools” [19].

Software like *Serato DJ*<sup>7</sup> (see Figure 2.3) or *Traktor Pro*<sup>8</sup> (see Figure 2.4) are good examples, considering their functionalities aim to aid the DJ perform tasks like music mixing, beat-matching or harmonic synchronization in order to perform pleasant and coherent transitions or mashups, all this in a live performance context.



Figure 2.3: Screenshot from Serato’s DJ software *Serato DJ 1.7.5*.<sup>a</sup>

<sup>a</sup><http://serato.com/dj/downloads>

<sup>7</sup><https://serato.com/about>

<sup>8</sup><http://www.native-instruments.com/en/products/traktor/dj-software/traktor-pro-2/software-features/>



Figure 2.4: Screenshot from Native Instruments DJ software *Traktor Pro 2*.<sup>a</sup>

<sup>a</sup><http://www.native-instruments.com/en/products/traktor/dj-software/traktor-pro-2/>

## 2.1.5 Existing tools oriented for studio development of mashups

The available music production software, also known as DAWs, are programs that allow the user to create new sounds using digital synthesizers, audio and Musical Instrument Digital Interface (MIDI) loops, or pre-recorded samples and packs. DAWs also allow manipulation of existing music with a large variety of cutting, pasting, sampling and sequencing tools. This way the user can create mashups with a variety of existing songs and record them to use in future live performances. The same is possible with remixes.

There are many DAWs available in the market sharing similar functionalities and performance. Software like *FL Studio 11*<sup>9</sup> (see Figure 2.5) from Image-Line, the iOS oriented *Logic Pro X*<sup>10</sup> (see Figure 2.6) from Apple Inc and *Ableton Live*<sup>11</sup> (see Figure 2.7) from Ableton are highly used.

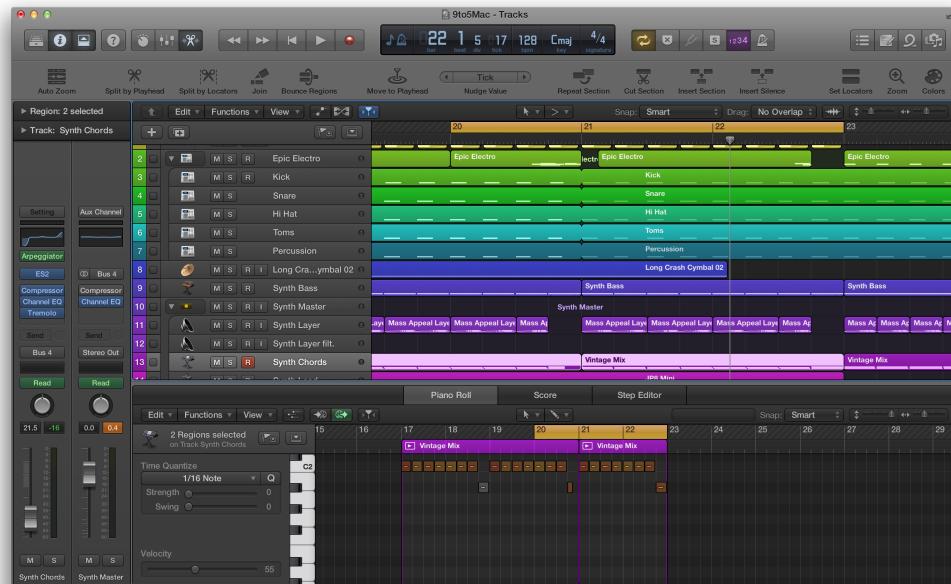
<sup>9</sup><http://www.image-line.com/index.html>

<sup>10</sup><https://www.apple.com/logic-pro/>

<sup>11</sup><https://www.ableton.com/en/live/>

Figure 2.5: Screenshot from *FL Studio 11*, a DAW from Image-Line.<sup>a</sup>

<sup>a</sup>[http://www.image-line.com/press/presskitblog.php?entry\\_id=1362048119&title=fl-studio-11](http://www.image-line.com/press/presskitblog.php?entry_id=1362048119&title=fl-studio-11)

Figure 2.6: Screenshot from *Logic Pro X*, a DAW from Apple Inc.<sup>a</sup>

<sup>a</sup><http://9to5mac.com/2013/07/26/logic-pro-x-review-powerful-new-features-a-simplified-ui-with-no-compromises-for-pros/>

Ableton is a music software company based in Berlin that develops and distributes the production and performance program *Ableton Live*<sup>12</sup> (see Figure 2.7) and a collection of related instru-

<sup>12</sup><https://www.ableton.com/en/live/>

ments and sample libraries. Ableton<sup>13</sup> was founded in 1999 and released the first version of *Live* in 2001 being the last version released *Live 9*. This software based in loops became quite famous due to its session view layout where the user could compose using cells in a vertical grid. This DAW established itself not just as a recording program for composers, but also as a performance instrument in itself, and nowadays is the main software for music creation and manipulation<sup>14</sup>.

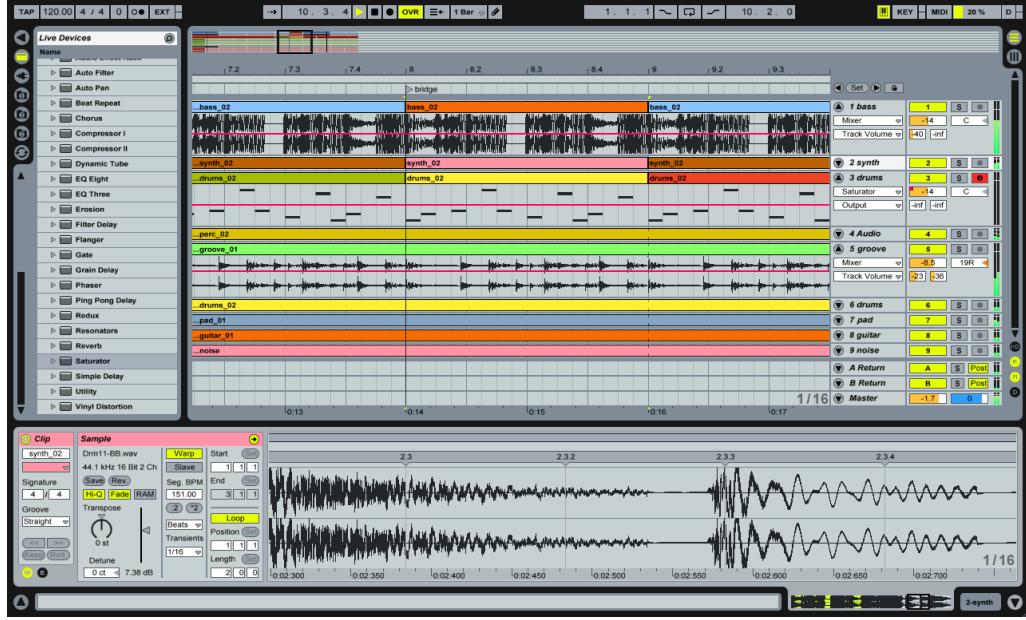


Figure 2.7: Screenshot from Ableton’s DAW, *Ableton Live 9*.<sup>a</sup>

<sup>a</sup><http://homestudio.com.br/cursos/live/>

## 2.2 Music Analysis, processing and manipulation

### 2.2.1 Programming languages for creative applications

During the last 20 years, a large variety of programming languages have been used to create modules or systems to perform real-time MIR and DSP. High-level languages can be used to retrieve audio information and enable algorithmic composition, in order to manipulate sound features. Some of the most relevant used programming languages are briefly described as follows.

*Matlab*<sup>15</sup> is a high-level programming language and interactive environment which allows to explore different areas including signal and image processing, communications, control systems, and computational finance.

*Python*<sup>16</sup> is another widely used general-purpose, high-level programming language, and it was developed in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum in

<sup>13</sup><https://www.ableton.com/en/about/>

<sup>14</sup><http://www.musicradar.com/tuition/tech/the-19-best-daw-software-apps-in-the-world-today-238905/20>

<sup>15</sup><http://www.mathworks.com/products/matlab/features.html>

<sup>16</sup><https://www.python.org/about/>

the Netherlands as a successor of a language called *ABC*. It is a multi-paradigm programming language that focuses on code readability.

*SuperCollider*<sup>17</sup> is a programming language for real-time audio synthesis and algorithmic composition. It was developed by James McCartney and originally released in 1996. He released it under the terms of the GNU General Public License in 2002 when he joined the Apple Core Audio team.

*Processing*<sup>18</sup> is a programming language, development environment, and online community. Since 2001, *Processing* has promoted software literacy within the visual arts and visual literacy within technology.

*Max*<sup>19</sup> and *Pure Data*<sup>20</sup> (*Pd*), are visual programming languages for music and multimedia, developed by Miller Puckette at IRCAM, then bought by Opcode and finally by Cycling'74. They allow researchers, multimedia performers, musicians and developers to create software graphically, without writing lines of code.

### 2.2.2 Music information retrieval

Music information retrieval (MIR) can be considered as “a field that covers all research topics involved in the understanding and modelling of music and that uses information processing methodologies” [18].

“We define musically relevant data as any type of machine-readable data that can be analysed by algorithms and that can give us relevant information for the development of musical applications. The main challenge is to gather musically relevant data of sufficient quantity and quality to enable music information research that respects the broad multi-modality of music.” [18]

During this project audio content analysis techniques resulting from MIR research will be used to extract musical concepts using algorithms applied to the audio signal. Some of the research topics and techniques within MIR relevant to this project are described in this section. Some of them can be considered as global audio features (e.g. tempo, beat or downbeat), which are used to perform beat-matching with techniques like time-stretching. Others can be considered as local features (e.g. chords or key signatures). Here follows a brief description of these components and estimation techniques.

Tempo and beat are very important in the perception of music (which can be defined as a time structured set of sound events) and carry important information that can be used in many applications: query by tempo, processing using tempo information (e.g. beat synchronous mixing, beat slicing, segmentation into beat units), musical analysis (i.e., interpretation) or, more generally, sound analysis. For this reason, tempo/beat estimation have been the subject of an increasing number of contributions in the last few years [12].

---

<sup>17</sup><http://supercollider.github.io/>

<sup>18</sup><https://processing.org/>

<sup>19</sup><https://cycling74.com/max7/>

<sup>20</sup><http://puredata.info/>

In music terminology, beats can be defined as the time instants at which human beings would tap their foot for rhythm of the music, and tempo is defined as the speed or a pace of a given song, and it is usually indicated in beats per minute (BPM). It refers to the pace of music measured by the number of beats occurring in one minute. The greater the number of BPM, the smaller the amount of time between successive beats, and therefore the faster the song. Particularly in EDM, accurate knowledge of a song's BPM is important to perform beat-matching. Tempo and beat estimation techniques are fundamental in automatic music processing and its purpose is to provide information about the BPM and temporal beat locations of the upcoming track in order to perform time-stretching to match the tempo of the current playing track [1].

Contrary to the many contributions available in the topic of beat-tracking research in recent years, less attention has been given to higher level metrical analysis, such as downbeat extraction/detection [4]. “The meter of a piece of music implies a counting mechanism for hierarchical stressed and unstressed beats within a measure. A downbeat is the first beat within a measure (or if counting beats, the one)” [7], i.e., the first beat of each bar.

Despite downbeat extraction being a topic yet to be fully studied in the same level as beat-tracking, there are many possible applications within MIR. Some examples are: to enable fully automated rhythmic pattern analysis for genre classification, to indicate likely temporal boundaries for spectral audio segmentation and improve the robustness of beat-tracking systems by applying higher level knowledge [4].

In tonal music, besides components like tempo and genre, the musical key is also an important feature [11]. “The main key of a musical work, and the sequence of keys through which the music passes, are fundamental to music analysis. The home key serves as an anchor: the chord sequences in the music may lie within the home key, or may contain notes that are not part of the home key and therefore pull away from the anchor, suggesting other keys” [10]. In the context of traditional tonal music, the main key provides the context according to which the expressive characteristics of modulation and tonicization are assessed.

Many approaches have been proposed to perform key estimation on symbolic data (MIDI or notated music). Some examples include looking for key-establishing harmonic aspects, using the tonal hierarchy or implement harmonic analysis [11].

In traditional tonal music, a chord is defined as the simultaneous sounding of three or more different notes. Chord progressions are frequently used in modern music and any chord can, through inventive voice-leading, be followed by any other chord, although certain patterns of chords have been accepted as the main progressions in common-practice harmony.

Many approaches and methods for chord extraction are based in a low-level feature called the Harmonic Pitch Class Profile (HPCP) or chroma vector, which is a 12-dimensional vector of real numbers representing the energy or salience of the pitch classes. The sequence of chroma vectors

that describe the pitch class content of an audio signal over time is called a chromagram. To calculate chromagrams some approaches have been proposed like automatic tuning to the reference frequency, median smoothing, removal of harmonics or noise attenuation [9]. Some of its applications are key estimation [11], chord extraction [9] or melody and bass line estimation [15].

In the field of MIR, automatic estimation of musical key or of chord progression over time for a music track, has received much attention in the recent years due to its many possible applications. For example it allows search/query music databases, automatic playlists generation and automatic accompaniment [13].

### 2.2.3 Digital signal processing techniques

Through the analysis of the information retrieved with the MIR techniques mentioned in section 2.2.2, it is possible to develop algorithms to manipulate digital music signals and perform the necessary adjustments in audio content that may enable a system to create automated music mashups with musical coherence. Some of the most relevant technologies resulting from MIR and used in DSP are time-scaling (also known as time-stretching) and pitch-shifting techniques.

Pitch-shifting is a digital audio effect that changes the pitch of a sound or music signal maintaining its duration, i.e., all frequencies are scaled by a constant factor. Pitch-shifting is widely applied in electronic music creation or manipulation such as pitch correction of musical performances in the recording studio or transposing songs to a desired key.

A common approach to perform pitch-shifting consists in time-scaling the input signal, and then resampling the output to restore the signal's original time-base while having shifted its frequency content [17].

Time-stretching audio signals can be defined as the process of changing the signal's temporal scale without modifying its frequency content. It has many applications, for example, beat-matching. In general, time-stretching can be performed via time-domain or frequency-time methods. Time-domain methods allow large stretching ratios and are more suitable to speech or monophonic signals but they don't perform that well with polyphonic signals. In the time-frequency domain the phase-vocoder is the most common approach [14].

In terms of time-frequency approaches, the phase vocoder is a well-established tool for time-scaling and pitch-shifting speech and audio signals via modification of their short-time Fourier transforms (STFTs). The phase-vocoder is generally considered to yield high quality results although it may be less efficient computationally than time-domain approaches. [17]

To overcome some limitations of the phase-vocoder algorithms, others approaches to perform these DSP techniques have been proposed. An example is frequency-domain pitch-shifting based on the constant-Q transform [17].

## 2.3 References to develop a new application

### 2.3.1 Existing systems oriented for real-time development

In order to bring the exciting reality of music mashup experimentation closer to the inexperienced users, academic researchers have been developing systems that aid the user in the creation process. Due to the advances of MIR research areas and the increase in computational capability of computer systems, it is now possible to perform real-time DSP on consumer-grade systems [19]. Combining music manipulation techniques like beat trackers and phase vocoders it is possible to develop systems able to perform automated synchronization of audio clips. This allows us to replace the traditional audio editing paradigm of the DAW with an intuitive clip selection interface, where any user regardless his experience is able to experience mashup creation [6]. In the past few years some apps oriented for real-time and/or automated mashup have been developed. Due the computational advances of handheld systems some apps oriented to user-guided mashup were also developed for smartphones and tablets. Some of the most relevant apps of the past few years are mentioned below.

#### 2.3.1.1 Computer oriented systems

The *Beat-Sync-Mash-Coder*<sup>21</sup> [6] is a web application for real-time and semi-automated creation of beat-synchronous music mashups (see Figure 2.8).

---

<sup>21</sup><http://music.ece.drexel.edu/bsmc>

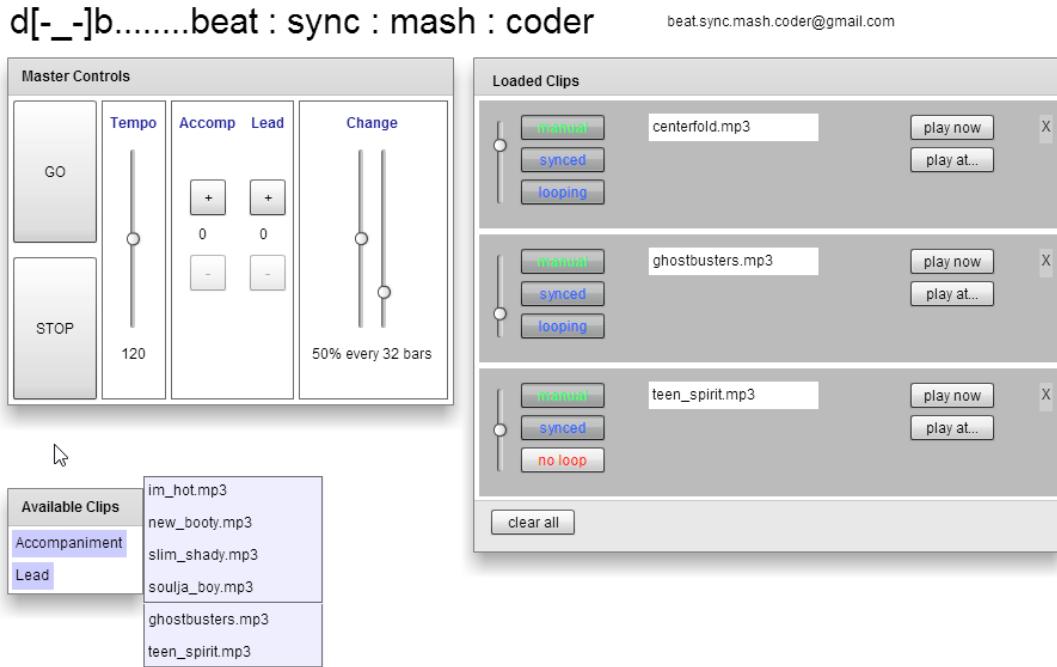


Figure 2.8: Screenshot of *Beat-Sync-Mash-Coder* user interface.<sup>a</sup>

---

<sup>a</sup><http://music.ece.drexel.edu/bsmc/>

The user only needs to choose clips between “lead parts” (solo vocals or instruments) and “accompaniment parts” (drum and bass or chordal instrument parts), and select the desired output tempo. During playback clips can be added or altered, resulting in new mashup creations. The system uses DSP modules (a phase vocoder and beat tracker) to automatically combine audio clips in a musically meaningful way. The beat tracker enables the system to detect the locations of beats in an audio clip and then the beats of every clip are synchronized to ensure that the overall output is perceived as musically coherent. The phase vocoder allows the system to change the tempo of each clip to match the user-specified target during playback, while minimally changing the user’s perception of the pitch and timbre. Due to the high computational demand of DSP algorithms, the framework used was *Alchemy* from Adobe, which allows C code to be compiled to optimized bytecode that can be executed in the cross-platform Flash environment. This choice provided the developers the power of C language without loss of platform independence, enabling the system to perform all DSP client-side.

The *AutoMashUpper*<sup>22</sup> [3], presented in 2014, is a system developed in Matlab that allows users to create mashups by mixing different songs at distinct regions of an input song (see Figure 2.9).

---

<sup>22</sup><http://telecom.inescporto.pt/~mdavies/automashupper/>

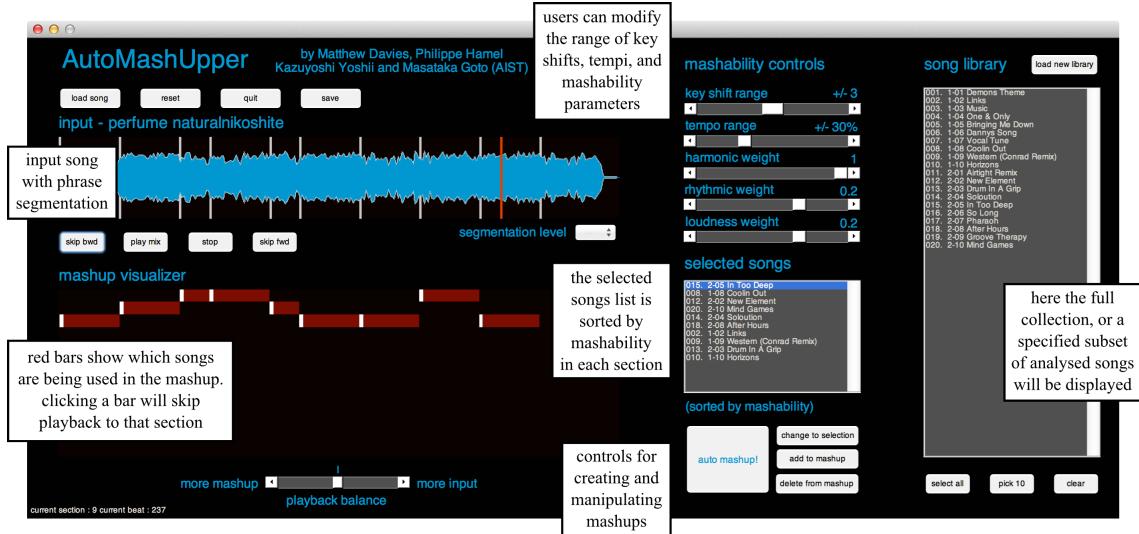


Figure 2.9: Screenshot of *AutoMashUpper* user interface.<sup>a</sup>

<sup>a</sup><https://staff.aist.go.jp/m.goto/PAPER/ISMIR2013davies.pdf>

The system measures the spectral balance between songs, as the harmonic and rhythmic similarity, in order to determine how elements of songs can be made fit together using key transposition and tempo modification. When the song with highest mashup compatibility for each section is found, the system performs time-stretching, pitch shifting and amplitude scaling to successfully match the mashup. Via the user interface the user is allowed to manipulate or alter the mashup and control the parameters in which the compatibility estimation between songs is calculated. To perform time-stretching and pitch-shifting operations the open-source library *Rubberband* was used whilst the *Replay Gain* method was used to perform loudness compensation.

The *Improvasher*<sup>23</sup> [5], presented in 2014, is a real-time musical accompaniment system which creates an automatic mashup to accompany live musical input (see Figure 2.10).

<sup>23</sup><https://sites.google.com/site/nime14improvasher/>

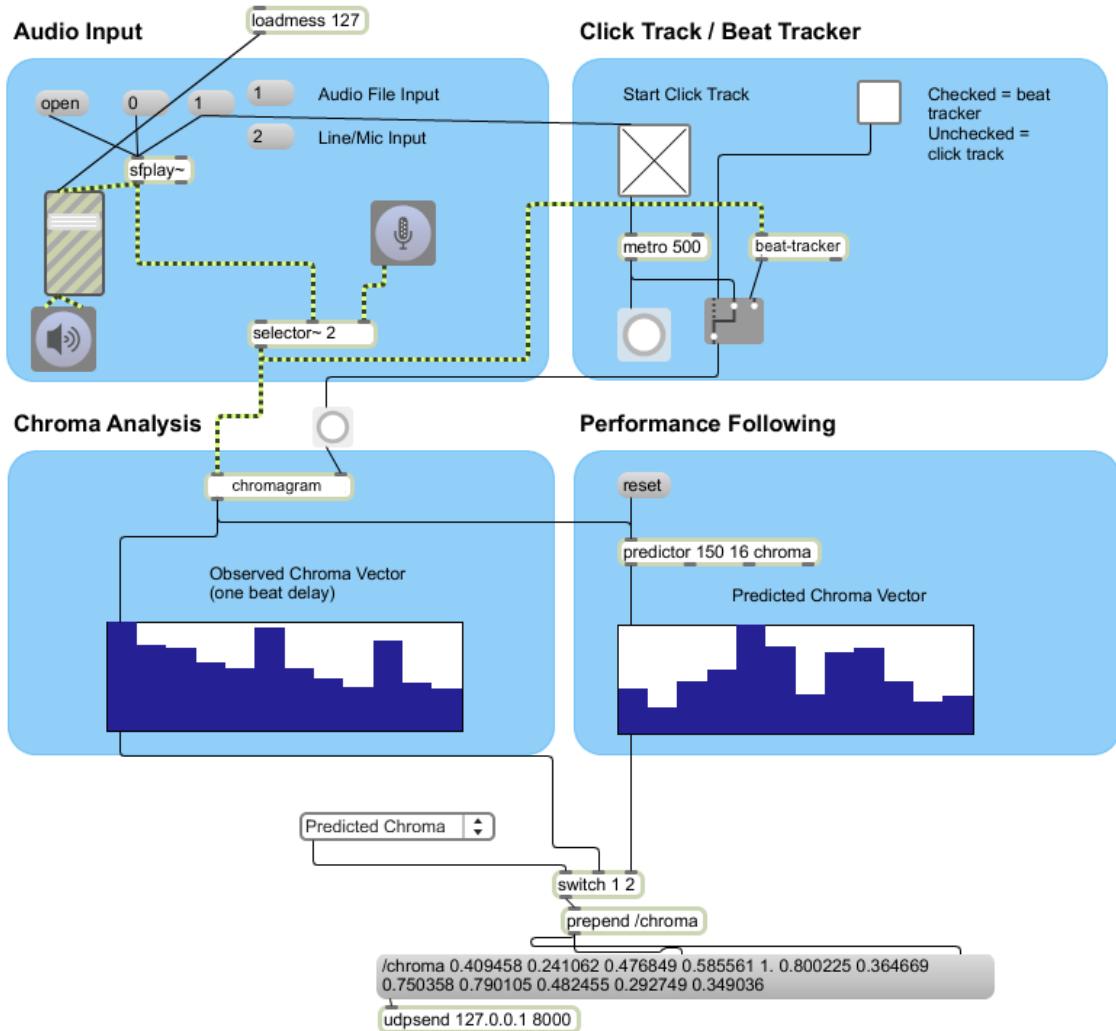


Figure 2.10: Screenshot of *Improvasher* patch for *Max*.<sup>a</sup>

<sup>a</sup>[http://nime2014.org/proceedings/papers/405\\_paper.pdf](http://nime2014.org/proceedings/papers/405_paper.pdf)

This application also makes use of DSP techniques resulting from MIR research and it was built around two modules: a performance following module and a music mashup module. “Given a live musical input (e.g. from a musician playing the guitar), we use the performance follower to predict the chroma (i.e., harmonic) content for the next beat in the live input signal, and send this chroma information to the mashup system to determine the best matching beat slice from a set of candidate songs. Once the beat slice with the highest compatibility has been found, the corresponding audio content is played back in real-time to accompany the live input. Proceeding in this way, beat by beat, our system Improvasher, can create a real-time mashup accompaniment for the live musician.” (see Figure 2.11).

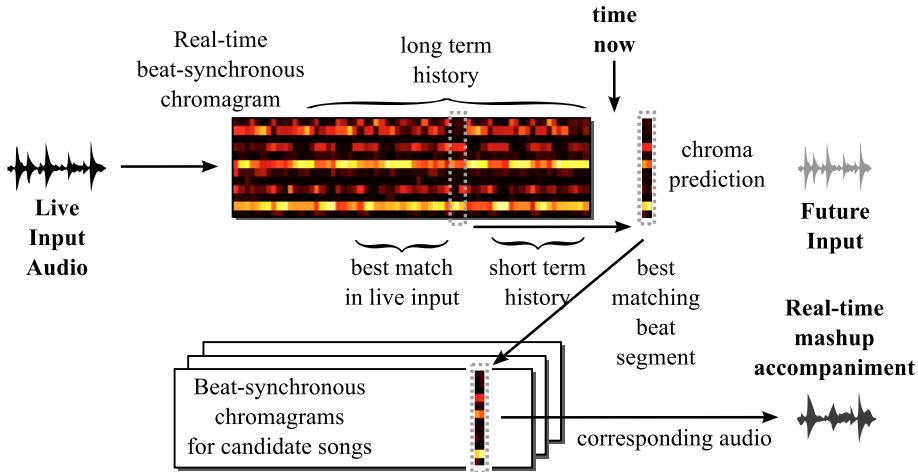


Figure 2.11: Diagram of *Improvasher* real-time mashup system.<sup>a</sup>

<sup>a</sup>[http://nime2014.org/proceedings/papers/405\\_paper.pdf](http://nime2014.org/proceedings/papers/405_paper.pdf)

The performance following module was implemented using the visual dataflow language *Max* and the mashup module was implemented using a standalone C++ application. The two modules were connected via Open Sound Control (OSC).

### 2.3.1.2 Handheld oriented systems

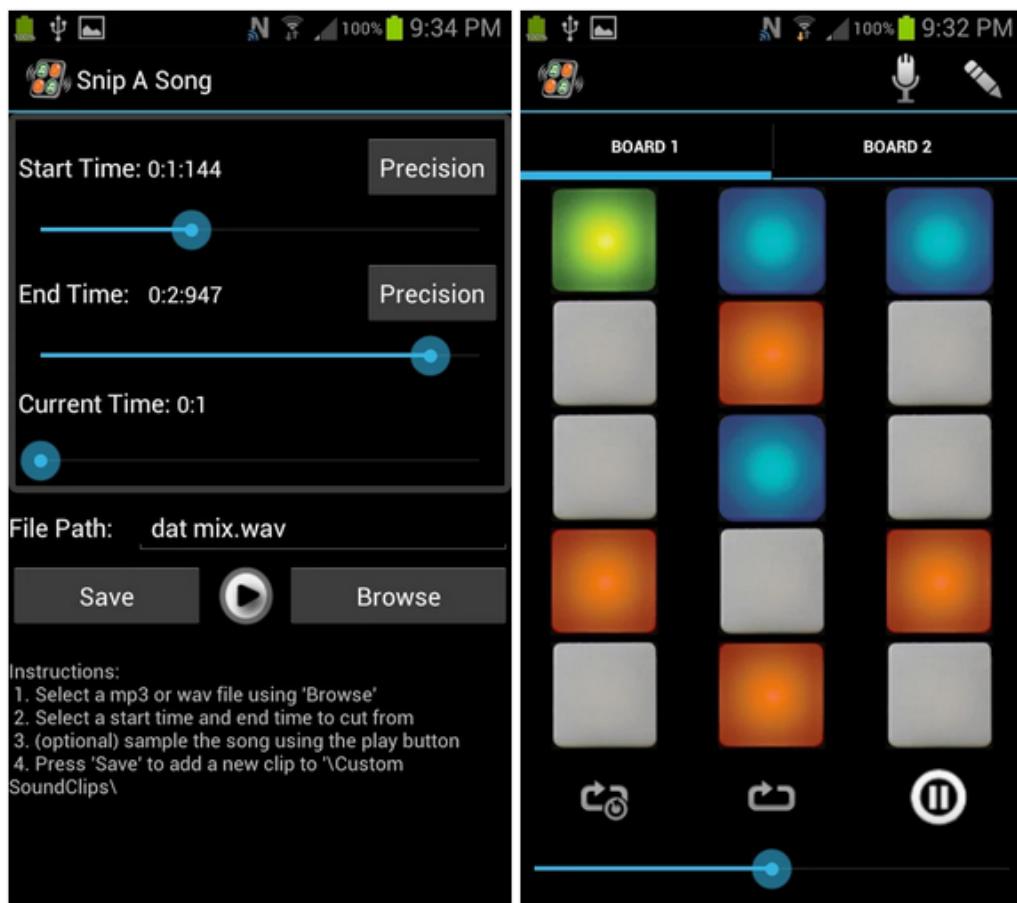
In the conventional mobile application online stores, such as Google Play Store<sup>24</sup> (for Android OS) and Apple Store<sup>25</sup> (for iOS), there are already available some apps that offer different levels of mashup creation experiences. A research in the app stores mentioned above revealed some mobile applications worth mentioning that are analysed below.

The paid application *Audio Mashup Pro*<sup>26</sup> (see Figures 2.12 and 2.13) was released on Google Play Store in November of 2012 by Jordan Brobyn. It's a simple application that allows the user play multiple sound clips or audio files (like MP3, WAV, M4a and Ogg) simultaneously. It can be used as a soundboard or to play composed music by combining different sounds to make music live.

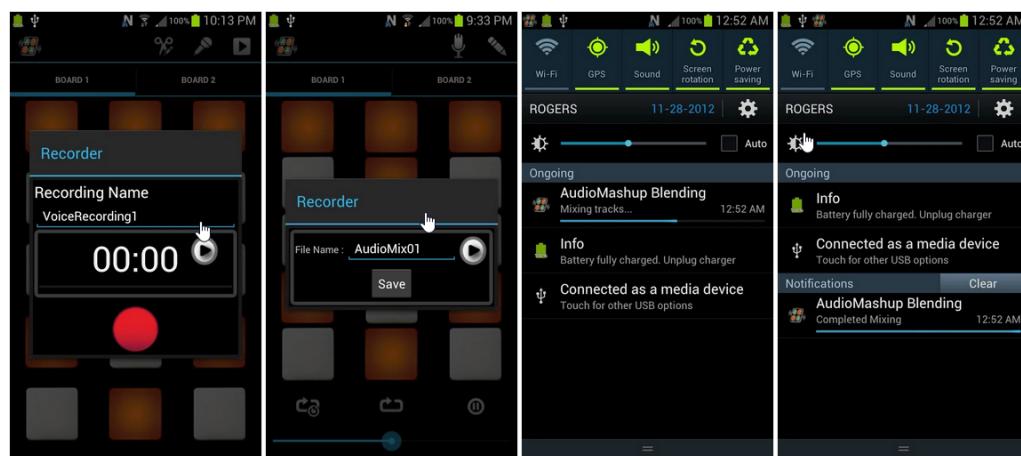
<sup>24</sup><https://play.google.com/store>

<sup>25</sup><https://itunes.apple.com/pt/genre/ios/id36?mt=8>

<sup>26</sup><https://play.google.com/store/apps/details?id=my.SoundBoard.Pro&hl=pt>

Figure 2.12: Screenshots of *Audio Mashup Pro* user interface – part 1.<sup>a</sup>

<sup>a</sup><https://play.google.com/store/apps/details?id=my.SoundBoard.Prohl=ptpT>

Figure 2.13: Screenshots of *Audio Mashup Pro* user interface – part 2.<sup>a</sup>

<sup>a</sup><https://play.google.com/store/apps/details?id=my.SoundBoard.Prohl=ptpT>

There is also a *Lite* version available free of charge but it has less features. In terms of usage, the user can upload desired songs from the device to *Auto Mashup Pro* and is allowed to select start

and stop points in each music to trim the desired section. Then it's possible to choose the desired segmented clips and play them simultaneously under the control of an interactive soundboard. This method requires a minimum of skill in audio perception of the beat and synchronization in order to slice the uploaded songs in appropriate points, otherwise it is difficult to obtain a musical coherent mashup.

*Mashup*<sup>27</sup> is a paid application developed by NeoMia and released on Google Play Store in December of 2013. *Mashup* (see Figure 2.14) is a simple system that allows the user to splice together segments of any music or audio files stored in the device. The mashup creation results from playing a group of segments defined by the user in a desired order. Similar to *Audio Mashup Pro*, the user must rely on his hearing abilities to cut and select the audio segments.



Figure 2.14: Screenshots of *Mashup* user interface.<sup>a</sup>

<sup>a</sup>[https://play.google.com/store/apps/details?id=com.neomobia.splicer&hl=pt\\_PT](https://play.google.com/store/apps/details?id=com.neomobia.splicer&hl=pt_PT)

*iMash*<sup>28</sup> is a paid application developed by Mixed In Key and released on Apple Store in August of 2014. It allows to mashup two songs uploaded from the user's collection. The interface shows the waveforms of the two songs and the user can scroll through them to select the desired matching point (see Figure 2.15).

<sup>27</sup><https://play.google.com/store/apps/details?id=com.neomobia.splicer>

<sup>28</sup><https://itunes.apple.com/us/app/imashup-mashup-remix-app/id503804151?mt=8>

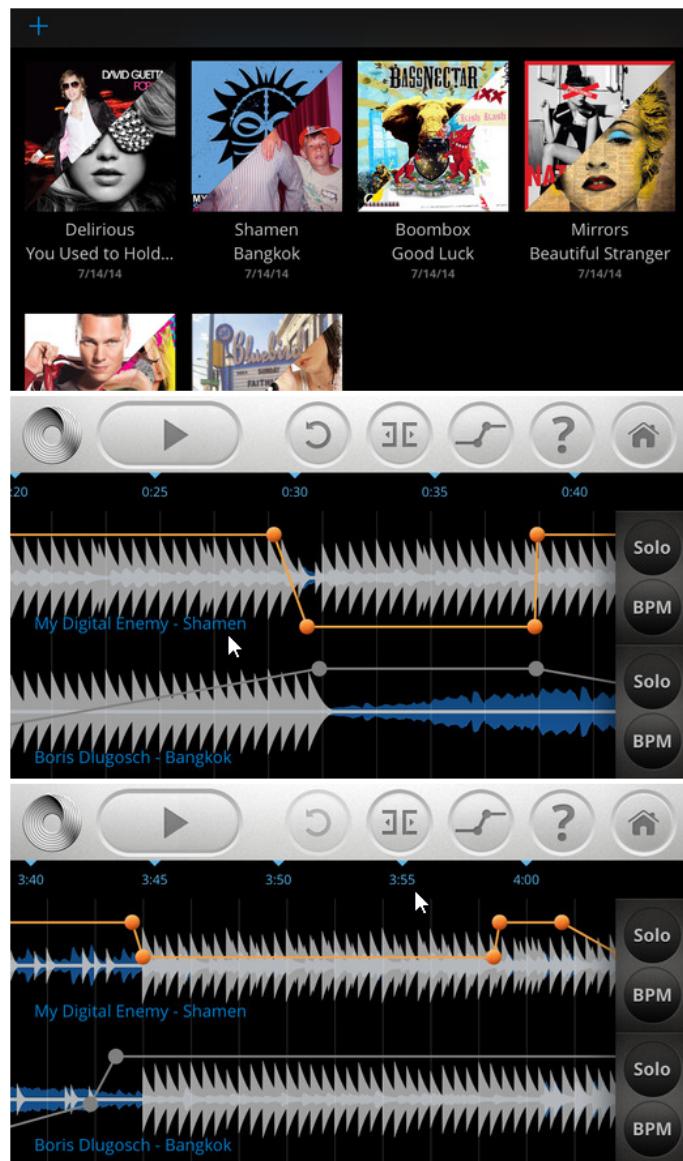


Figure 2.15: Screenshots of *iMash* user interface.<sup>a</sup>

---

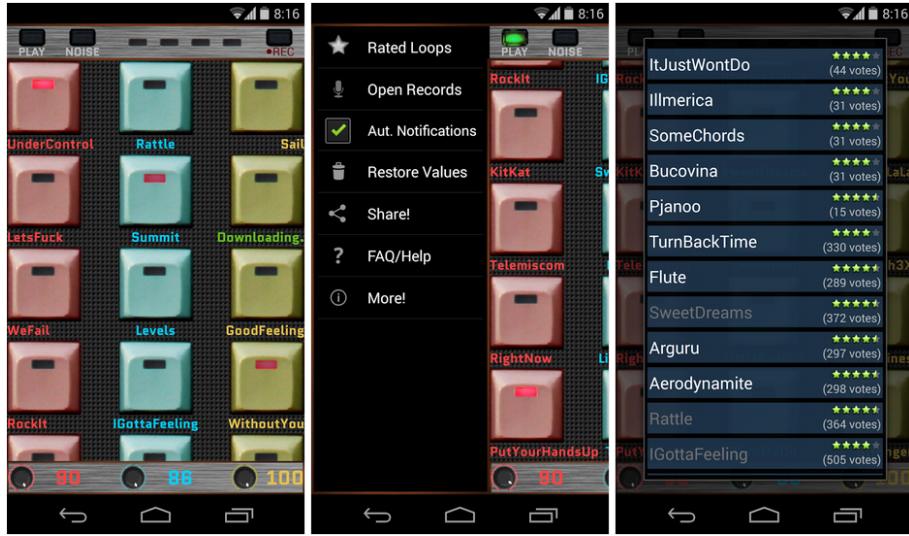
<sup>a</sup><https://itunes.apple.com/us/app/imashup-mashup-remix-app/id503804151?mt=8>

This system implements a beat-matching algorithm that aids the user when synchronizing the two songs which may improve the probability of obtaining a pleasant output, on the other hand the impossibility of using more than two song restricts some of the user's creativity.

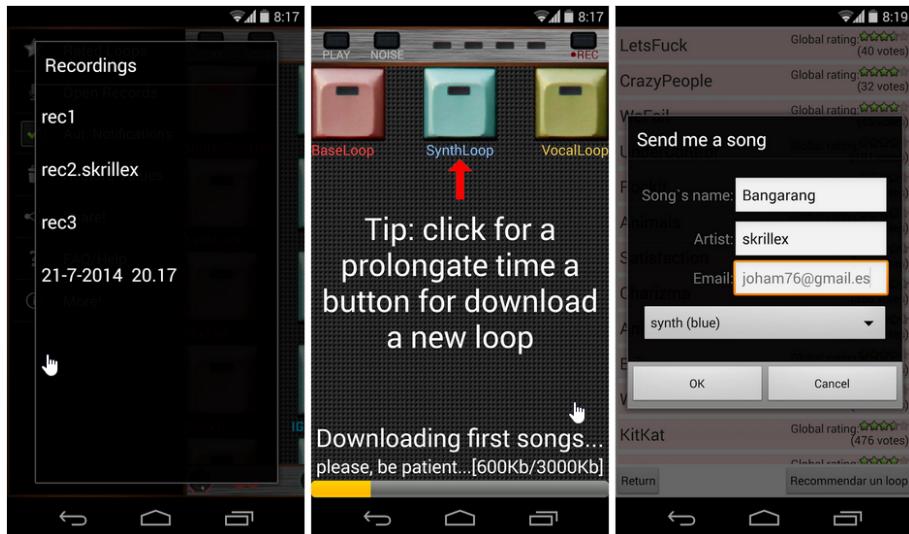
In December of 2014 the free application *Mashupper – Remix & Mashup Tool*<sup>29</sup> was released by Marzaise Labs on Google Play Store. Through a soundboard interface with interactive buttons (see Figures 2.16 and 2.17), the user only needs to combine the desired loops from three different types: base loops (“drum machine”), synth loops (“instrumental”) and vocal loops (“acapellas”).

---

<sup>29</sup><https://play.google.com/store/apps/details?id=com.marzaise2.mashup>

Figure 2.16: Screenshots of *Mashupper* user interface – part 1.<sup>a</sup>

<sup>a</sup><https://play.google.com/store/apps/details?id=com.marzaise2.mashuphl=ptPT>

Figure 2.17: Screenshots of *Mashupper* user interface – part 2.<sup>a</sup>

<sup>a</sup><https://play.google.com/store/apps/details?id=com.marzaise2.mashuphl=ptPT>

This system differs from *Audio Mashup Pro* and *Mashup* for two reasons: first only the pre-recorded audio files (loops) stored in the server can be used, the usage of personal music is not an option, and second the system is responsible for synchronization considering that only pre-recorded loops can be used. The mashup creation experience is less challenging but offers an easier approach to inexperienced users.

Another example is the also free application *Launchpad Mashup*<sup>30</sup>. Developed by Biltek Software and released on Google Play Store in December of 2014, this system is quite similar to *Mashupper – Remix & Mashup Tool* being that the user only is allowed to mix the existing pre-recorded loops.

### 2.3.2 Conclusions and motivation for project

Reviewing the music mashup oriented systems mentioned in 2.3.1. it's feasible to divide the available approaches in research/academic oriented systems and commercial oriented applications.

The research oriented systems are the result of the increasing growth in processing capacities of computer systems, improvements in programming languages and MIR techniques which have made it possible to implement real-time DSP operations in audio signals.

The computer oriented applications mentioned in section 2.3.1 offer different levels of mashup experimentation: real-time beat-synchronous mashups using pre-analysed audio clips [6], multi-song mashup creation by adding pre-processed audio clips to an input song [3] and real-time mashup creation of musical accompaniment for a live musical input [5].

These applications combine beat-tracker and phase-vocoder technology to perform time-stretching and beat-matching operations, relieving the user from the synchronization tasks required in mashup creation. This makes it easier for inexperienced users to create mashups considering that it's only required to choose between the provided audio clips and define some *mashability* parameters.

Some systems also use high-level DSP to perform estimations about harmonic parameters like key signature and chord progression: the *AutoMashUpper* with measures of harmonic and rhythmic similarity between songs and a measure of spectral balance, in order to allocate and splice audio segments using key transposition and tempo modification, and the *Improvasher* by predicting the harmonic content of the next beat in the live input signal.

Although these systems perform in some level estimation and matching operations to aid the user's creative process, they can only be defined as research prototypes oriented for proof of concept and therefore not oriented for the common and inexperienced user: the user interfaces developed are not as user-friendly and intuitive as they could be and some knowledge about musical parameters/features may be required to fully understand the systems.

The commercial oriented applications were developed to be used in mobile devices such as tablets or smartphones. The proliferation of handheld device's industry emphasized the proximity that people have with mobile technology and how it is related with leisure activities. With that in mind, these mobile applications mentioned in section 2.3.1, which can be easily acquired online in mobile app stores, can be considered as the best strategy to reach more people that could become interested in experimenting software designed to create and exploit music mashup possibilities.

---

<sup>30</sup><https://play.google.com/store/apps/details?id=com.LaunchpadMashup>

On the other hand, the mobile applications available are quite restrictive in mashup possibilities and at providing DSP operations. They all have a pleasant and user-friendly graphical interface but offer less complex mashup experiences. Most of the apps that allow multi-song mashup are limited to the usage of pre-processed samples available in the systems, which considerably narrows the creative possibilities. On the other hand, the apps that allow to upload songs require the user to manually splice the music segments do be used in the mashup and this reality makes it difficult to effectively synchronize the segments to obtain pleasant results.

Considering the analysis in this section, does not currently exist a music mashup system that can simultaneous implement MIR/DSP techniques while running in an intuitive and friendly GUI oriented for mobile devices. This way all the matching and synchronization operations would be done client-side leaving only a simple an intuitive GUI for the user to interact with. This approach may be a positive strategy to bring the reality of mashup music creation closer to common and inexperienced users.

# Chapter 3

## Project Specification

### 3.1 Overview

The proposed system is an interactive music remixing application for handhelds that enables non-expert users to experiment in real-time mashup creation. Using *Pd*, MIR techniques were used to develop and implement music understanding functions to allow user-guided music mashup manipulation, as well as higher-level algorithms, such as HPCP measures, to implement a model of transition compatibility to perform intelligent sequencing. Using *MobMuPlat*, a standalone application that runs on Android and iOS mobile devices, the system was implemented with a friendly GUI in order to allow intuitive and easy experimentation for inexperienced users. The dataset of music excerpts was prepared with *FL Studio*.

The Figure 3.1 illustrates the proposed system model. What follows is a description of the development tools used during this dissertation in order to implement the proposed system.

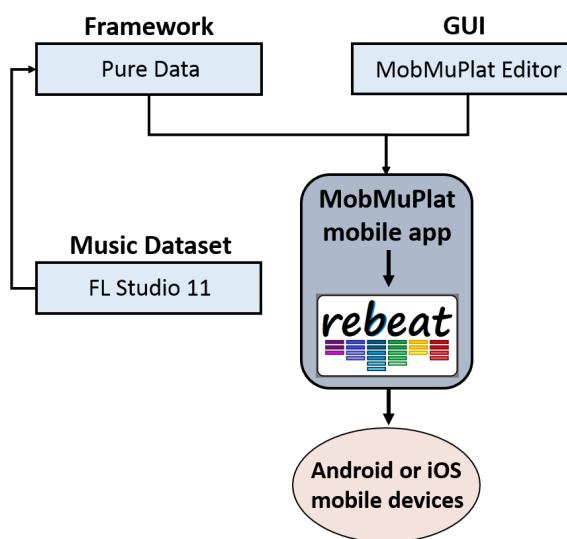


Figure 3.1: System model

### 3.2 Music collection pre-processing tools

The dataset collection needed to implement and test the DSP algorithms required specific features along with some previous analysis and processing. Existing music samples were selected and some music excerpts were composed and produced.

Mainly due to personal experience, the chosen software was *FL Studio 11*<sup>1</sup> from Image-Line, a powerful DAW known worldwide for its production and audio manipulation possibilities (see Figure 3.2). This software easily provides information about audio files like BPM, tempo or key signature and allows highly creative music experimentation. It will also be used to convert audio files into WAV format to obtain the highest quality possible in output mashups.



Figure 3.2: Screenshot of work environment during a project in *FL Studio 11*.

### 3.3 Framework development tools

To implement MIR techniques and develop both low-level or high-level DSP algorithms, the programming language *Pd* was used. As mentioned in section 2.2.1, *Pd* is a patcher programming language like *Max*, being that they share very similar interfaces, operations/functionalities and methods of code creation.

*Pd* is a so-called data flow programming language, where software called patches are developed graphically. Functions, messages, variables, tables, arrays, etc. are represented by objects and are connected together with chords. Data flows from one object to another allowing to perform from very low level mathematic operations to complex audio or video manipulations (see Figure 3.3).

<sup>1</sup><http://www.image-line.com/index.html>

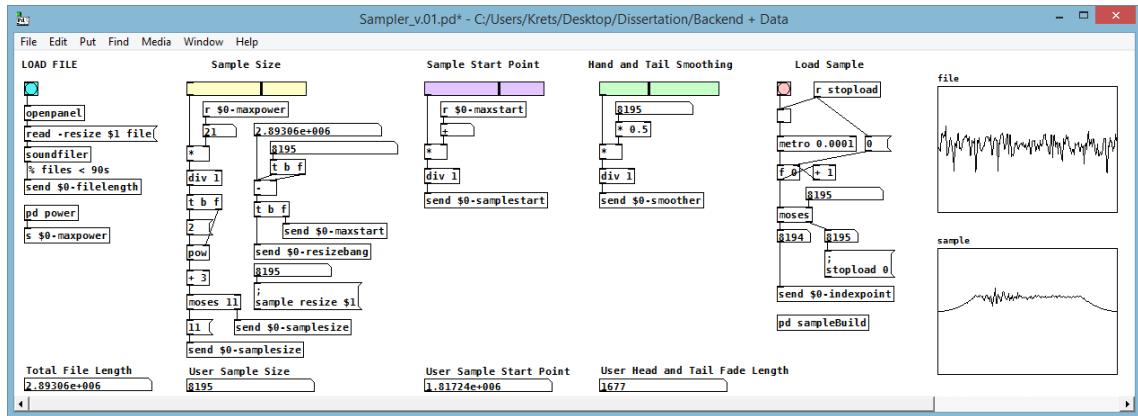


Figure 3.3: Screenshot of an audio file sampler project in the programming environment of *Pd*.

This audio-environment language allows implementations of *Pd* objects, *Pd* patches and objects developed in another programming language. Regarding this project, an important aspect was the possibility of implementing *Pd* functionalities using standalone applications to run in mobile devices as mentioned in section 3.4. Comparing with *Max*, *Pd* has the advantage of being available open-source which results in countless contributions and improvements of audio processing from developers and researches around the world.

## 3.4 Graphical user interface design tools

One of the main objectives of this dissertation was to design and develop a GUI to implement the proposed mashup system in handheld devices like tablets or smartphones. The interface needed to be intuitive and clear in order to enable first time user's experimentation without any problems.

This interface was designed using the tools from project *MobMuPlat*<sup>2</sup> (short for Mobile Music Platform), a standalone iOS+Android application developed by Iglesia Intermedia. It allows the creation of customized audio software, via the open-source audio environment *Pd*. *MobMuPlat* hosts a list of user-created documents, each of which defines a user interface and audio engine. Among other features, *MobMuPlat* can do synthesis, sampling, networking via local Wi-Fi, OSC messaging, set and query hardware characteristics, use device sensors, use MIDI and HID devices and display images or graphics. This standalone application made it possible to implement the user's functionalities developed in the audio engine with *Pd*, into the GUI designed with *MobMuPlat Editor* (see Figure 3.4).

*MobMuPlat* project can be found open-source on Github and the *MobMuPlat* mobile application is available in the mobile application stores.

---

<sup>2</sup><http://www.mobmuplat.com/>

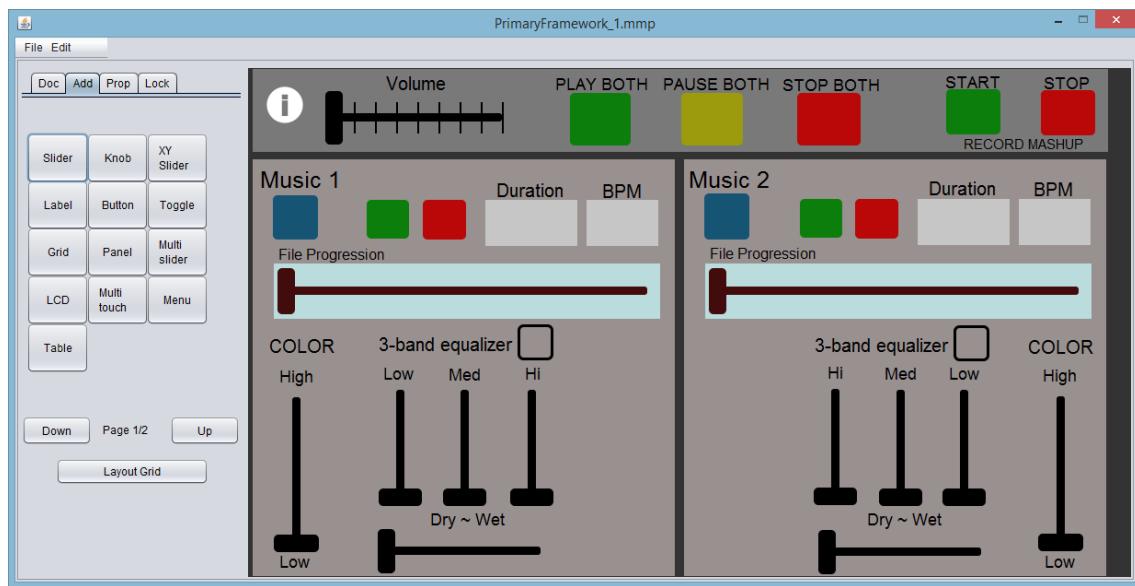


Figure 3.4: Screenshot of *MobMuPlat Editor* work environment.

# Chapter 4

## *rebeat* System

### 4.1 Overview

This chapter refers to the proposed approach for user-guided music remix, which resulted in the system *rebeat*, a mobile application oriented for music creation through guided experimentation.

Using the available songs stored in the backend, *rebeat* allows audio structure manipulation and guides the user into creating different music excerpts. The system was designed to offer a quick method of pattern creation and allows long term sequencing of these patterns in time.

The *rebeat* app is Android and iOS compatible and all user interactions can be performed in real-time, meaning that the MIR techniques implemented and all calculations run client-side and all values are updated in real-time.

### 4.2 Music Dataset Preparation

In terms of music mixing, one of the most essential techniques is beat synchronization. When *mashing* songs together or splicing music excerpts it is essential to match the sound files beat by beat in order to obtain a coherent musical structure through time. Most of the existing music mashup oriented mobile applications described in section 2.3.1.2, require the user to perform manual beat-synchronization (by selecting start and stop points to slice and choosing matching points) when adding new sounds to the mashup. In *rebeat*, beat-synchronisation is ensured because the available music excerpts which can be provided as input adhere to very specific properties, such as a fixed tempo and a fixed number of beats. This results in having beat-synchronization ensured, regardless any audio manipulation or action performed, allowing non-expert users to create perfectly beat-synchronized compositions.

In musical notation, a bar (or measure) is a segment of time corresponding to a specific number of beats. Dividing music into bars provides regular reference points to pinpoint locations within a piece of music. Considering a bar of 4 beats, the decided length for the music excerpts was 16 bars, resulting in 64 beats per music excerpt. In terms of tempo, an equal value of 120 BPM was

implemented, considering it is a typical tempo value of house music, a very common genre in DJ mixing. The result is that each beat has the duration of 500 milliseconds and therefore, each excerpt has the duration of 32 seconds. The relation between the music excerpts reference values is described in Table 4.1.

<b>120 beats per minute</b>			
	<b>Length</b> (beats)	<b>Length</b> (samples)	<b>Duration</b> (milliseconds)
1 beat	1	22050	500
1 bar	4	88200	2000
16 bars	64	1411200	32000

Table 4.1: Relation between the dataset music excerpts reference values, assuming audio sampled at 44100 Hz.

The methods of pattern creation presented in section 4.3.2 are based on splicing beats together from different positions in songs, and even from different songs, therefore it was necessary to implement a robust method for isolating exactly one beat in a piece of music.

Based on Table 4.1, a general equation can be written which returns the position (in samples) of a specific *beat*  $i$  within a specific *bar*  $k$  – see Equation 4.1.

$$beat(k, i) = B \cdot k + b \cdot i + 1 \text{ samples}, \quad \left\{ \begin{array}{l} \text{constant } B = 88200 \text{ samples} \\ \text{constant } b = 22050 \text{ samples} \\ \text{variable } k = \text{bar} \in \{0, 1, \dots, 15\} \\ \text{variable } i = \text{beat} \in \{0, 1, 2, 3\} \end{array} \right. \quad (4.1)$$

As mentioned before, which music excerpt has 16 bars (0,1,..., 15) wherein each bar has 4 beats (0,1,2,3), so, considering that the constants  $B$  and  $b$  represent, respectively, the distance (in samples) between bars and between beats, it is possible to isolate any beat desired within a music excerpt.

As mentioned in section 3.2, due to personal experience and familiarity, *FL Studio 11* was the DAW selected to perform the above mentioned dataset pre-processing and data preparation. The first eight excerpts were composed and produced by implementing acquired music production knowledge. The remaining four are the result of existing music samples spliced together out of shorter excerpts with the purpose of increasing the harmonic variety. As shown in Figure 4.1, each of these last excerpts is the result of 4 different 4 bars length (8 seconds) music samples spliced together, resulting in the 16 bars length (32 seconds) music excerpts.

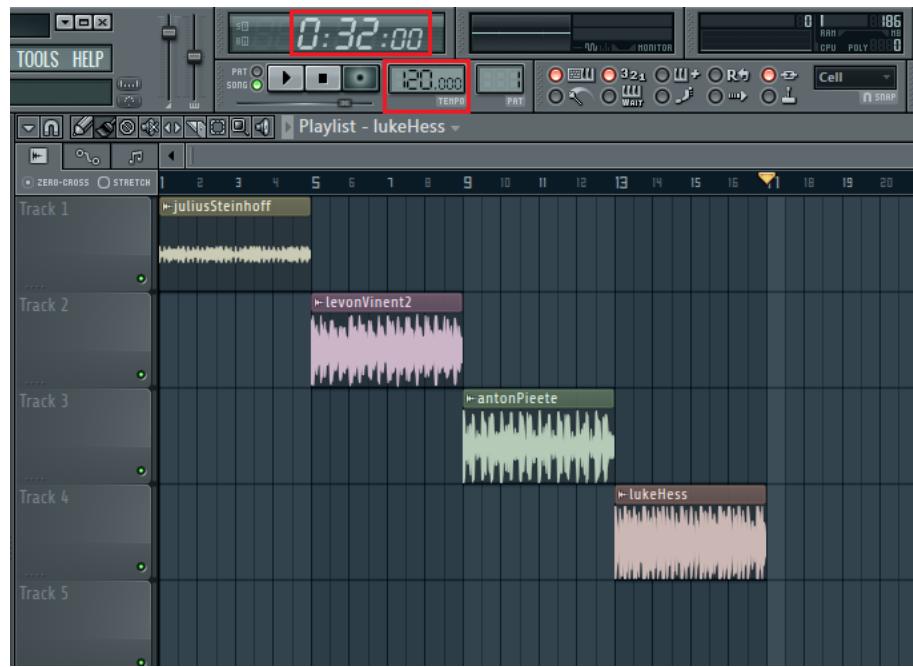


Figure 4.1: Screenshot of *FL Studio 11* where 4 different music samples are being spliced together resulting in a music excerpt with 16 bars length (32 seconds).

All music excerpts were stored as mono WAV files with 44100 Hz sampling rate, and although it increases the dataset size compared to using MP3s and hence the amount of storage space on the handheld device, the purpose is to ensure the maximum possible audio quality.

By specifying these temporal properties and pre-processing the music dataset used in this way, beat-synchronization is ensured. This means that all subsequent operations of pattern creation and sequencing can be simplified based on this structure.

## 4.3 Methods of Pattern Creation

### 4.3.1 Patterns

Regarding the scope of this dissertation, a pattern is defined as a group of beat slices spliced together that represent the first level of composition provided by the system. In *rebeat*, it's possible to create, manipulate and work with up to four different patterns, composed with beat slices from one or two songs. As shown in Figure 4.2, each pattern is composed by four beat slices from two different songs.

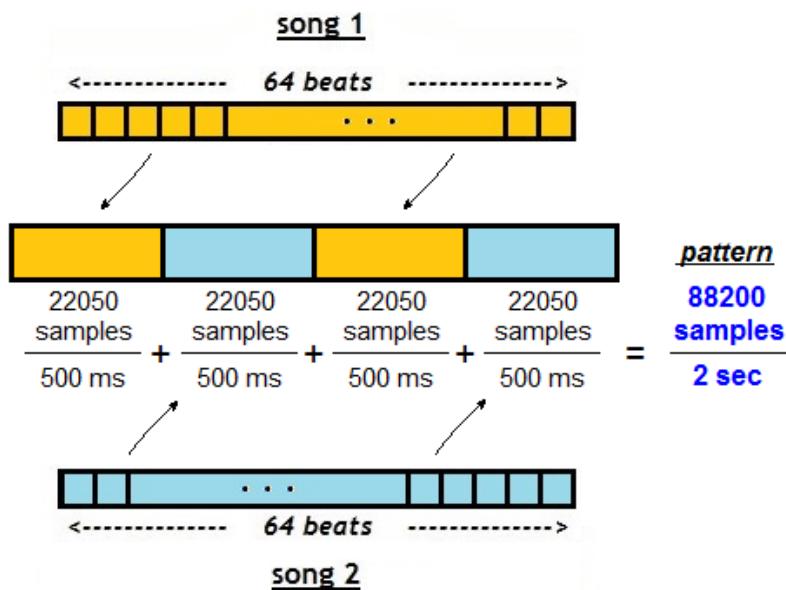


Figure 4.2: A pattern is composed by four beat slices spliced together from two different songs. Theoretically, these beat slices can be from anywhere in the song, however there are several strategies for choosing them.

With the intention of aiding the user in patterns creation, the system's framework implements algorithms based on musical structure features and algorithms based on information from HPCP (i.e. harmonic) measures from the audio files.

The pattern creation processes represent a high-level decision that is controlled by the system. The user is allowed several types of audio manipulations during the pattern creation process, but ultimately, the decision of which beats are selected to be spliced into a pattern is assigned to the pattern creation algorithm.

What follows is a description of how the patterns are created with illustrations of how they are stored and accessed in the framework.

#### • Pattern Creation:

Every time a pattern is created, changed or played, information stored in tables from the framework, is accessed and updated in real-time. The result is that all manipulations performed by the user can be heard in the audio output while being made.

Each pattern has information stored in different tables where each table corresponds to specific values from the features that can be controlled by the user.

When a pattern is created, four beat slices are selected from the two loaded music excerpts. The following information relative to each beat slice is stored in three tables:

- Song Number - Identifies from which song is the beat slice. If equal to "0", it means the beat slice is from music excerpt 1 (from channel 1); if equal to "1", means the beat slice is from music excerpt 2 (from channel 2);

- Pattern Samples - Refers to the number of samples of the beat slice, i.e., relates to the pattern size;
- Reading Window - Refers to the beat slice window (start point and end point) on which the selected music must be played in the beat slice. The corresponding beat slice window time duration in milliseconds is also stored.

Figure 4.3 shows an illustrative example regarding the creation of one pattern.

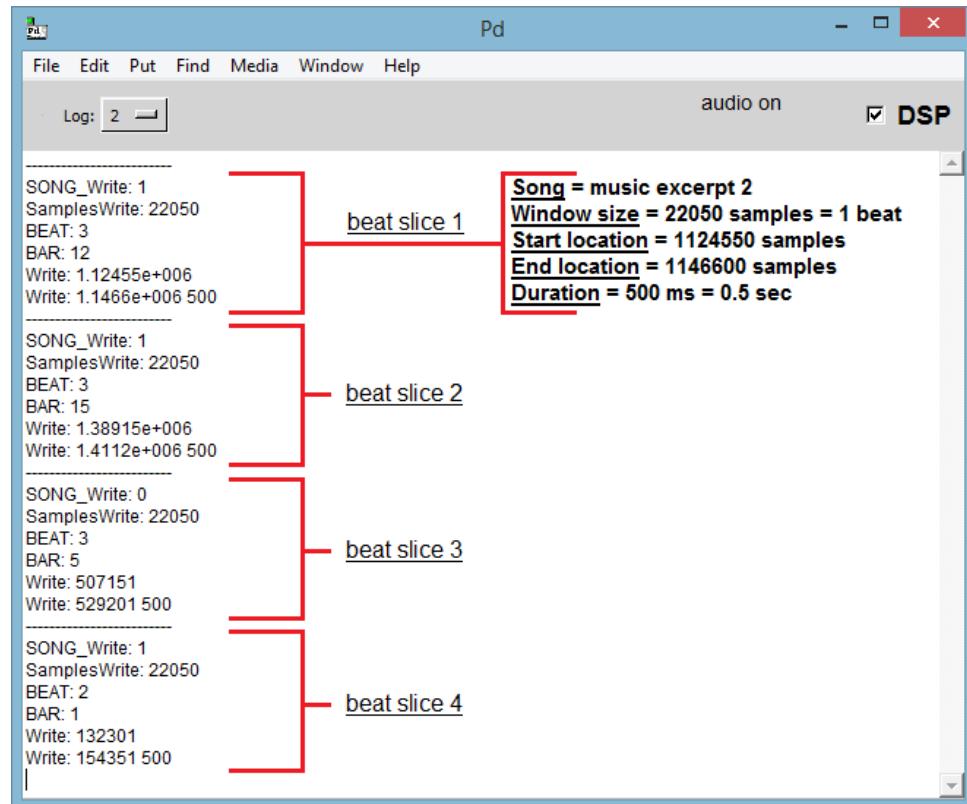


Figure 4.3: Screenshot from the *rebeat Pd* framework: *Pd* compiler displaying information regarding the creation of one four beat long pattern.

In this case, the default pattern size is four beats, meaning one beat per beat slice. On the other hand, the remaining of the parameters – song selection and beat selection (sample window start point) – were defined randomly. The location in samples of any beat within the music excerpts can be calculated using Equation 4.1. Applying it at “beat slice 1” from the pattern in Figure 4.3 and according to what it shows, the sample location is:

$$\text{beat}(12,3) = 88200 \times 12 + 22050 \times 3 + 1 = 1124551 \text{ samples}$$

In each one of the mentioned tables, the information relative to each pattern, is stored in the same position. In this way, to play a specific pattern the system only needs to read the

mentioned tables by looking for the index relative to the desired pattern's position. Every time a user plays a pattern, this information retrieved from the tables is sent to a *Pd* `line~` object, resulting in audio output. This means that each pattern can be played and changed as many times as desired, which has proved to be an important feature to implement long term patterns sequencing in the “Final Mashup Grid”.

- **Change Pattern Size:**

When the user opens *rebeat*, the pattern size of all patterns is defined by default as being equal to 4 beats. With the purpose of increasing the number of possible combinations of beat slices within each pattern, it was implemented the possibility of defining different pattern sizes than 4 beats length – 2, 8 or 16 beats. What follows is an explanation of how different pattern sizes results in different beat slices combinations.

Considering that the pattern size by default is four beats long and that each pattern is composed by four beat slices, this implies that each beat slice is 1 beat length. Figure 4.4 illustrates how the beats are sliced and spliced together in patterns with the default size. For this example, the beat sequence was randomly defined as 0, 1, 2 and 3, which according to the previous equation, results in the respective sample window starting points:

- $\text{beat}(0,0) = \underline{1 \text{ sample}}$
- $\text{beat}(0,1) = \underline{22051 \text{ samples}}$
- $\text{beat}(0,2) = \underline{44101 \text{ samples}}$
- $\text{beat}(0,3) = \underline{66151 \text{ samples}}$

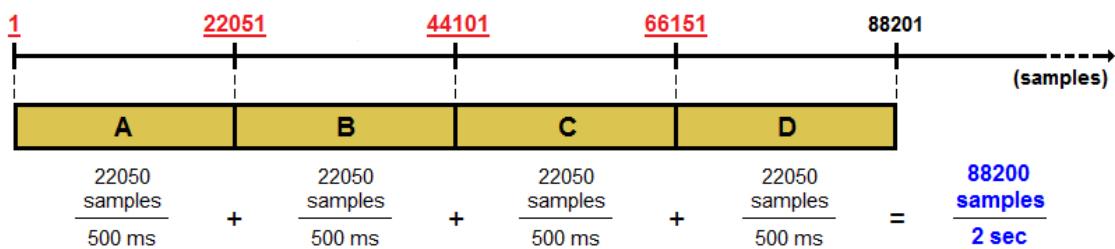


Figure 4.4: Example of beat slicing and splicing method for patterns with 4 beats length.

As shown in the figure, the result is a pattern with four beats length, composed by four samples with one beat length, which corresponds to a total duration of two seconds.

When the pattern size is different than four beats, the beat slices length will be different than one beat. Considering that, patterns can also be two beats long (using half-beat slices), eight beats long (using two-beat slices) or sixteen beats long (using four-beat slices) – see Table 4.2.

Pattern Sizes	Beat slices length (beats)	Pattern length (samples)	Pattern duration (seconds)
2 beats long	half beat	88200 samples	1 sec
4 beats long	1 beat	44100 samples	2 sec
8 beats long	2 beats	176400 samples	4 sec
16 beats long	4 beats	352800 samples	8 sec

Table 4.2: Different pattern sizes, assuming audio sampled at 44100 Hz.

### 4.3.2 Algorithms of Pattern Creation

#### 4.3.2.1 Algorithm 1 - Baseline Random Transition Model

The algorithm presented in this section was developed to serve as a reference of what the creation pattern process would generate without any musical guidance or structure implied, contrary the algorithms described in the following sections.

As mentioned before, each of the available music excerpts has the length of 64 beats, i.e., 16 bars (from bar 0 until bar 15), where which bar is composed by four beats (beat 0, beat 1, beat 2 and beat 3). This algorithm of pattern creation relies on the relation between bars and beats and with Equation 4.1, the system is able to calculate the location in samples of any beat within a music excerpt.

The algorithm first selects a random bar between the sixteen available, and then selects a random beat within four beats available in the bar. The resulting beat represents the first beat slice of the pattern. The remaining three beat slices of the pattern are selected the same way. An illustrative example is shown in Figure 4.5.

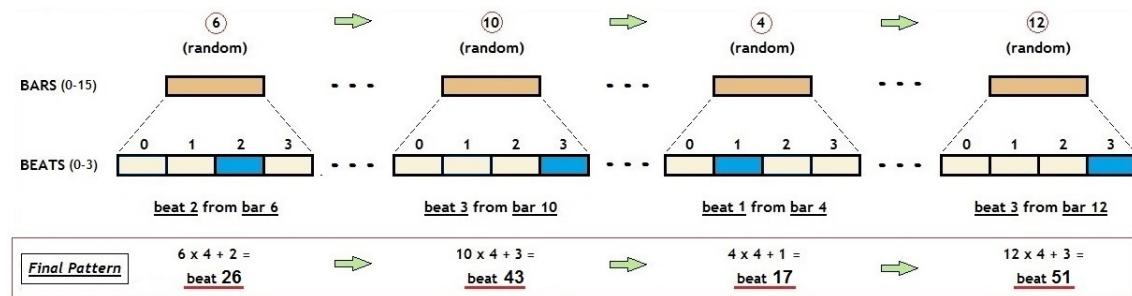


Figure 4.5: Example of a pattern created with Algorithm 1 – Baseline Random Transition Model.

Considering the fact that not only the bars are selected randomly, but also the beats within each bar, this algorithm generates patterns with unpredictable combinations of beats.

#### 4.3.2.2 Algorithm 2 - Metrical Structure Based Transition Model

The method of pattern creation presented in this section is based on musical structure characteristics and relies on the relation between bars and beats of Equation 4.1.

For example, if a music excerpt is played without any manipulation, the natural beat progression through the audio file would naturally be – beat 0, beat 1, beat 2, beat 3, beat 4, beat 5, beat 6,... beat 63. Comparing this sequence with the definition of beat location from Equation 4.1, results in the following respective beat indexs – beat(0,0), beat(0,1), beat (0,2), beat(0,3), beat(1,0), beat(1,1), beat (1,2),... beat (15,3) – as shown in Table 4.3.

Audio file progression						
Beats	0	1	2	3	4	5
Samples	1	22051	44101	66151	88201	110251
<b>beat(k,i)</b>	beat(0,0)	beat(0,1)	beat(0,2)	beat(0,3)	beat(1,0)	beat(1,1)

Table 4.3: Relation between a natural beat progression through an audio file and the respective beat progression within each bar, assuming 120 BPM and audio sampled at 44100 Hz.

It was decided to implement in *rebeat* framework a method for pattern creation where an algoritm selects randomly one bar (from the sixteen available) for each beat slice of the pattern, while the beat within each bar is selected in order to ensure the natural beat progression whithin each bar – beat 0, beat 1, beat 2 and beat 3 – see Figures 4.6 and 4.7.

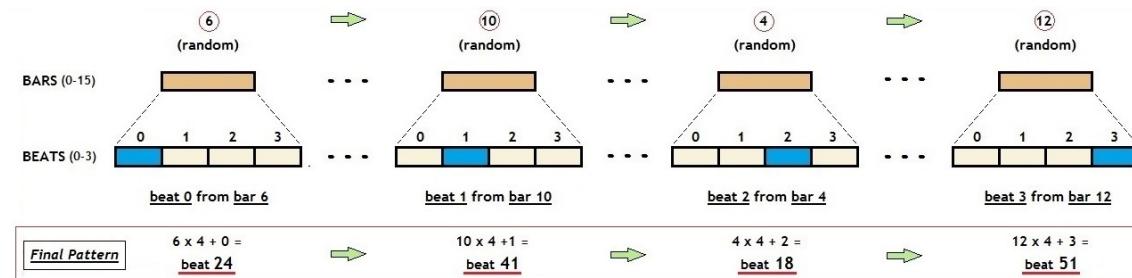


Figure 4.6: Example of a pattern created with Algorithm 2 – Metrical Structure Based Transition Model.

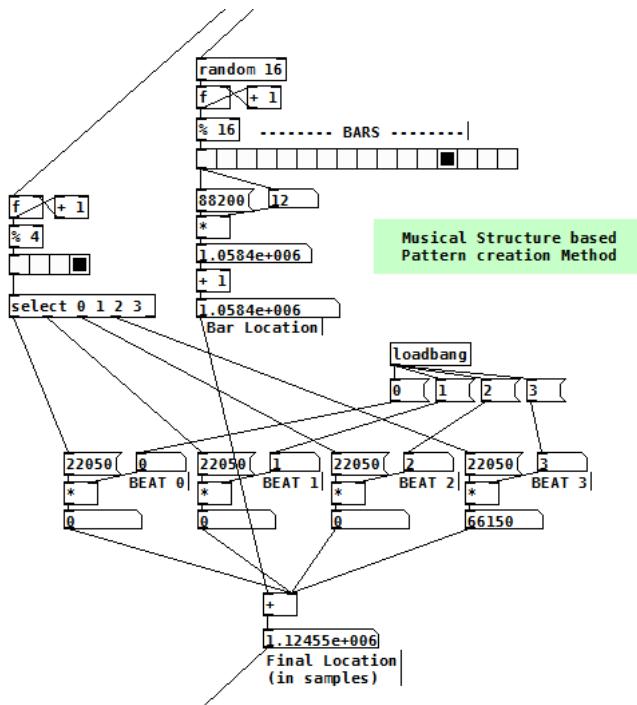


Figure 4.7: Screenshot from *rebeat*'s *Pd* framework: implementation of Algorithm 2.

- **Probability Threshold:**

The “Probability Threshold” (see Figure 4.8) is a slider that allows the user to change the probability of which music – “Music 1” or “Music 2” – will be selected in each of the four beat slices selection that compose each pattern.

The slider value directly changes in the pattern creation process. For each beat slice, the *Pd* object `random` generates a random number (between 0 and 1), and if that number is lower than the number defined by the “Probability Threshold”, the `moses` object will select the music excerpt 2 (from channel 2). On the other hand, if the number generated by the `random` object is higher than the slider’s value, the `moses` object will select the music excerpt 1 (from channel 1).

The slider has values limited between 0 (left) and 1 (right), which means that moving it to the right, will increase the probability of selecting music excerpt 2. To increase the probability of selection music excerpt 1, the slider must be moved to the left. By default, when the user opens *rebeat*, the “Probability Threshold” slider is 0.5, resulting in an equal probability of selecting both songs for each beat slice of each pattern.

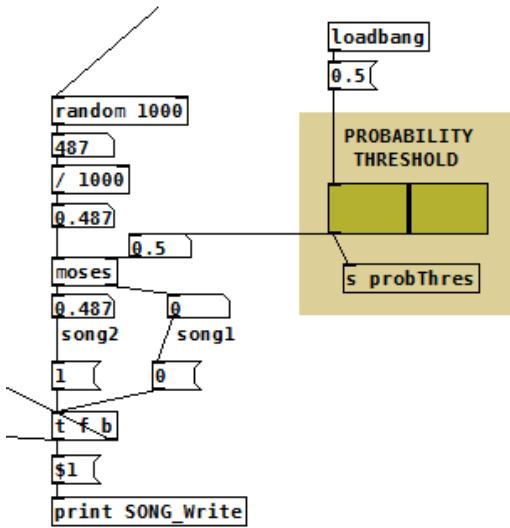


Figure 4.8: Screenshot from *rebeat's* *Pd* framework: implementation of the probability threshold slider.

#### 4.3.2.3 Algorithms 3 and 4 - Chroma Based Transition Model

This method is responsible for pattern creation where the beats selected to compose a pattern are chosen by measuring the similarity between the beats available, in order to preserve harmony during transitions in time within a pattern. The purpose is to create patterns composed of similar sounds, which should increase the chances of obtaining pleasant results. This similarity is measured by calculating the cosine distances between HPCP vectors calculated for each beat. At any time, there are 128 beats to create a pattern, i.e., two music excerpts (one loaded per channel) with 64 beats each, resulting in a great number of possible combinations within each pattern. This method selects the next beat to compose a pattern by looking for the best beat match available, i.e., the beat with the minimum cosine distance value associated. What follows is a description of how the method was designed and implemented in *rebeat's* framework.

- **Harmonic Pitch Class Profile:**

The HPCP, also known as chroma, is a low-level feature suitable to represent the pitch content of polyphonic music signals. “The chroma vector is a 12-dimensional vector of real numbers representing the twelve pitch classes (C,...,D), which amounts to considering pitch while supressing the height dimension. Much like a spectrogram describes the spectral content of a signal over time, the chromagram is a sequence of chroma vectors that describes the pitch class content of an audio signal over time.” [9]. Within MIR, it has been commonly used in areas like key estimation [11], chord extraction [9] or melody and bass line estimation [15].

With the purpose of providing the system the ability of calculating HPCP measures, a *Pd* patch previously developed by Gilberto Bernardes (researcher in the SMC Group at INESC TEC), was adapted and implemented in *rebeat*'s framework.

In order to calculate HPCP vectors referring to each beat of each song, the following operations are performed per beat each time a new music excerpt is loaded in the system:

1. Obtain the frequency components of the music signal through spectral analysis, using a Fast Fourier Transform (FFT) which converts the signal from the temporal domain into the frequency domain. The features of the FFT applied to each beat are as follows:
  - (a) Sample rate = 44100 Hz;
  - (b) Window size =  $2^{13} = 8192$  samples
  - (c) Hop size = 4096 samples ( i.e, overlap between windows equal to 50%)
  - (d) Frequency range is limited to 5000 Hz (rough approximation of the upper limit of instrumental tones)
2. Peak detection where only the local maximum values of the spectrum are considered. To that purpose the *Pd* object `sigmund~` is used with the number of peaks limited to 20.
3. Folding the 20 highest spectral peaks to a 12 bins distribution vector (each bin denotes a note of the equal-tempered scale, also referred to as pitch classes) by accumulating their energy in the respective bin calculated by the modulo 12 of the input frequency in MIDI note numbers.

The above operations result in a 12 bin HPCP vector per beat. In order to compare the similarity between beats, cosine distance calculations are performed using the HPCP vectors. The cosine distance equation is based in the cosine similarity and is defined as follows – Equation 4.2:

$$D_c(A, B) = 1 - S_c(A, B) = 1 - \frac{\sum_i^n A_i \cdot B_i}{\sqrt{\sum_i^n (A_i)^2} \times \sqrt{\sum_i^n (B_i)^2}} = \begin{cases} \approx 0 \rightarrow similar \\ \approx 1 \rightarrow distinct \end{cases} \quad (4.2)$$

The cosine similarity measures the angular cosine between two vectors (A and B) of an inner product. Because  $\cos(0^\circ)=1$  and less than one for other angles, the cosine distance gives the relative distance (0 to 1) between two HPCP vectors. The level of similarity refers to how close the distance value is to zero or to one, wherein results close to zero represent great similarity between vectors, and results close to one represent the opposite.

For each beat, the corresponding beat with the minimum cosine distance (closest to zero) is defined as being its best beat match (see Figure 4.31) and its location in samples is stored.

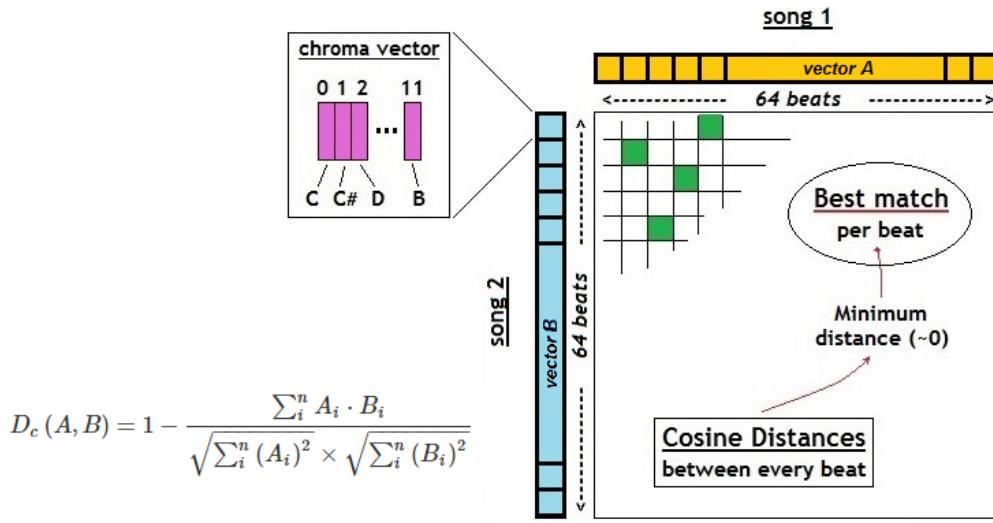


Figure 4.9: The best match per beat is selected based on the minimum cosine distances between beats.

The above figure refers to cosine distance calculations between song 1 (channel 1) and song 2 (channel). The illustration shows how the best matches per beat are found by measuring the similarity between each beat from one music excerpt and the 64 beats from the other. In order to give freedom to the algorithm to choose beats from both songs at any time, the cosine distance equation, was implemented to calculate the best beat matches between song 1 and itself, between song 1 and song 2, between song 2 and itself, and between song 2 and song 1.

- **Pattern creation with harmonic transition model**

Contrary to the music structure based algorithm of pattern creation presented in section 4.3.2.2, this method is based on HPCP measures from the implemented MIR techniques described above and is devoid of any implied music structure.

Concerning this method of harmonic transition, two slightly different algorithms were created, from now on, designated as Algorithm 3 and Algorithm 4. What follows is a description of how the four beat slices that compose a pattern are chosen in each of them:

- Algorithm 3

First the algorithm randomly selects one of the two music excerpts and one random beat (from the 64 available) within that excerpt. This beat represents the first beat slice of the pattern, in this case, beat 2 from music 1 (see Figure 4.10). At this point, the algorithm will search in both songs for the best match available to the previous beat. As the example shows, the best beat match, i.e., the minimum cosine distance, from music 1 is beat 12 and

from music 2 is beat 24. Considering that the cosine distance of beat 12 from music 1 is lower than in beat 24 from music 2, the second beat slice of the patterns is defined as being beat 12 from music 1. The third and fourth beat slices are selected the same way, resulting in the following final pattern: beat 2 (music 1) + beat 12 (music 1) + beat 27 (music 2) + beat 26 (music 2).

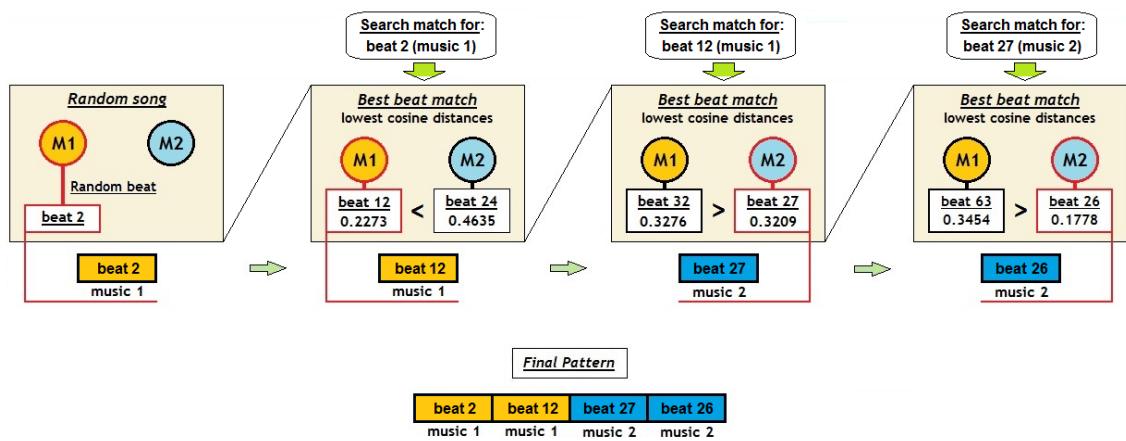


Figure 4.10: Example of a pattern created with Algorithm 3 - Chroma Based Transition Model.

#### - Algorithm 4

The pattern creation method with Algorithm 4 is identical to Algorithm 3, apart from one difference. In this algorithm the best match per beat, instead of being searched by comparing the cosine distances with the previous selected beat, it's searched by comparing the cosine distances with what would be the next beat after the previously selected one. As shown in Figure 4.11, the first beat slice of the pattern is beat 2 from music 1. The second beat slice to be selected is based on the best beat match calculated regarding  $\text{beat}(2 + 1) = \text{beat } 3$  from music 1. The rest of the pattern is created following the same logic.

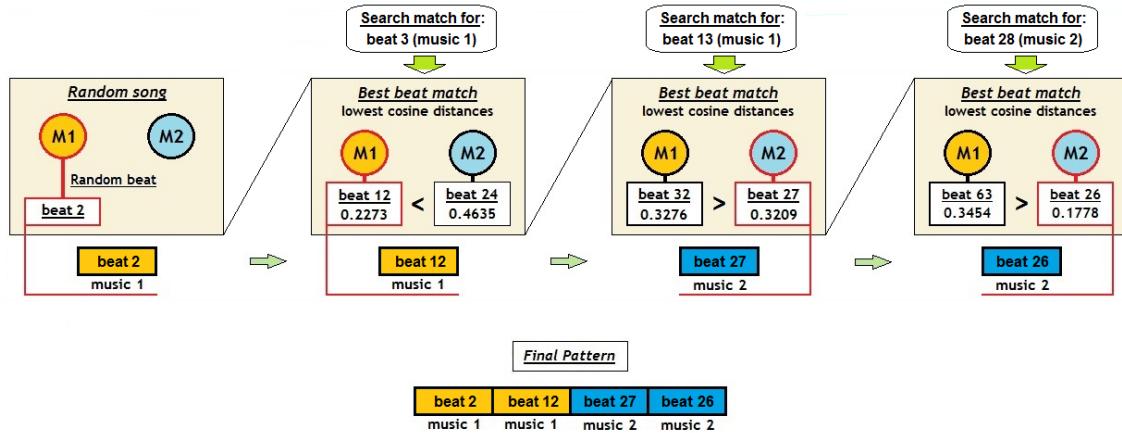


Figure 4.11: Example of a pattern created with Algorithm 4 - Chroma Based Transition Model.

#### • Probability Threshold:

Basically, as shown in Figures 4.10 and 4.11, in order to select the next beat to compose a pattern, the algorithm compares the best beat match available in each music and is inclined to select the beat with the best beat match value (i.e., lower cosine distance) from the two options available. This way, the slider purpose is to define the probability of selecting, at each moment, the best beat match from one music or the other. If the slider is completely to the left, the probability of selecting always the beat matches provided by the music from which the first beat slice of the pattern was chosen, will be equal to one, meaning that the entire pattern will be created with beats of the same music excerpt. In the other hand, if the slider is completely to the right, this means that the probability of selecting always the beat matches provided by the music from which the first beat slice of the pattern was chosen, will be equal to zero, meaning that any following beat to compose a pattern, will be selected from the opposite music excerpt selected in the beat slices before. This fact implies that the patterns will be composed by alternating beats from music 1 and music 2.

## 4.4 Interface

### 4.4.1 Overview

The goal of *rebeat* is provide users an easy way to experiment with real-time mashup creation. To this end, a GUI was developed using *Pd* and *MobMuPlat* to allow touch-based interaction on handhelds.

The GUI includes two panels, “Panel 1” and “Panel 2”, from now on referred respectively as “Load Panel” and “Mashup Panel”. The design of these panels is partially inspired by general designs incorporated in music manipulation software or hardware (see Figures 2.2, 2.5, 2.6 and 2.7) and also based on personal experimentation and interaction with music related systems. The

“Load Panel” was aimed to resemble with traditional mixing interfaces. Meanwhile, “Mashup Panel” is presented as a new kind of mashup oriented interface build around the creation of *patterns*.

In this section all of *rebeat*’s GUI functionalities and audio manipulation interactions available in both panels are fully presented.

#### 4.4.2 Load Panel

When a user opens *rebeat*, it presents the first GUI panel – “Load Panel” (see Figure 4.12). In this panel, through manual interaction with buttons and sliders, the user is allowed to perform different kinds of audio manipulation over the music excerpts available in the dataset.

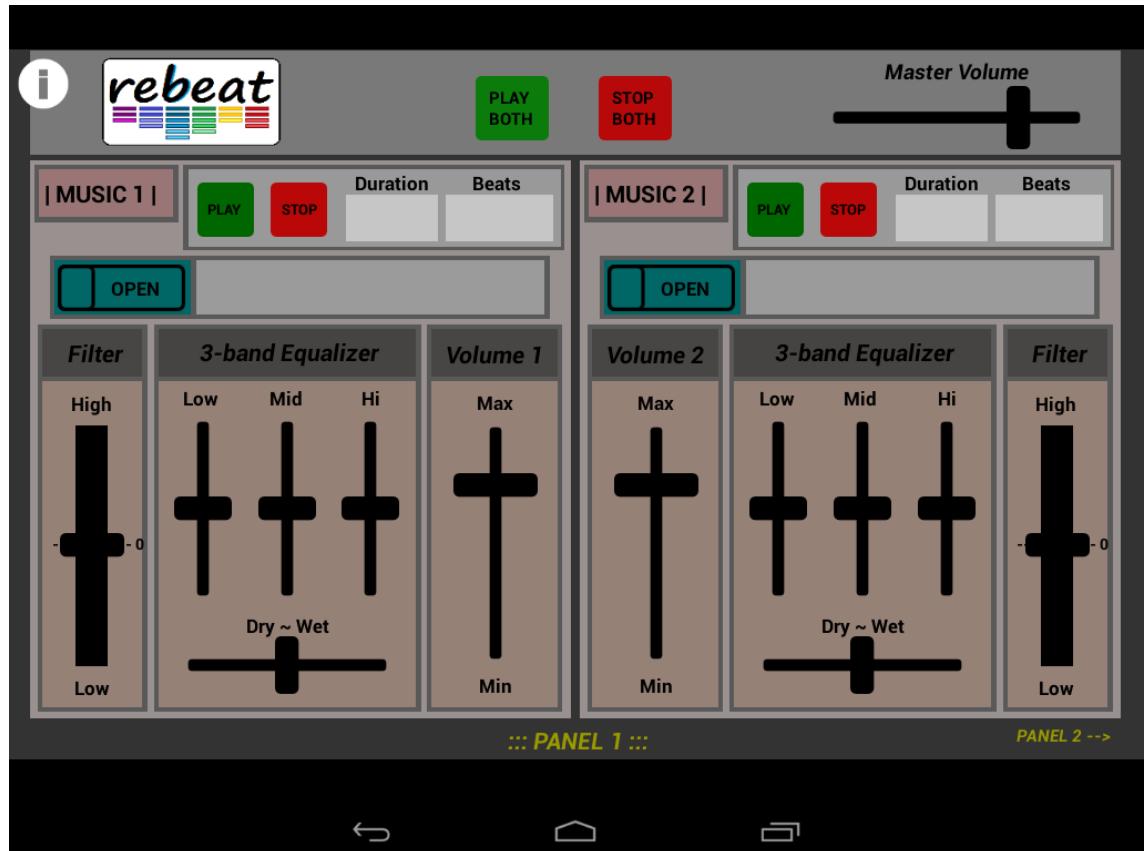


Figure 4.12: *rebeat* GUI: “Panel 1” or “Load Panel”

“Load Panel” is divided in two audio channels, entitled “Music 1” and “Music 2”, in which, per channel, it is possible to load one song at a time. What follows is a description of the interactive functions available.

#### 4.4.2.1 Play and Mix songs

- Choose and Open Song:

The “OPEN” button reveals a menu containing the list of available songs (pre-processed music excerpts) to load. By selecting one of the songs, the menu closes and the user is returned to “Panel 1”, where, the menu’s list index and name of the selected song appear in front of the “OPEN” button, and the displays the “Duration” and “Beats” values (see Figure 4.13).



Figure 4.13: Screenshot from Panel 1 – “OPEN”, “PLAY” and “STOP” buttons, and displays the calculated values for the “Duration” and “Beats”.

The backend *Pd* objects to allow the loading of separate audio files is shown in Figure 4.14).

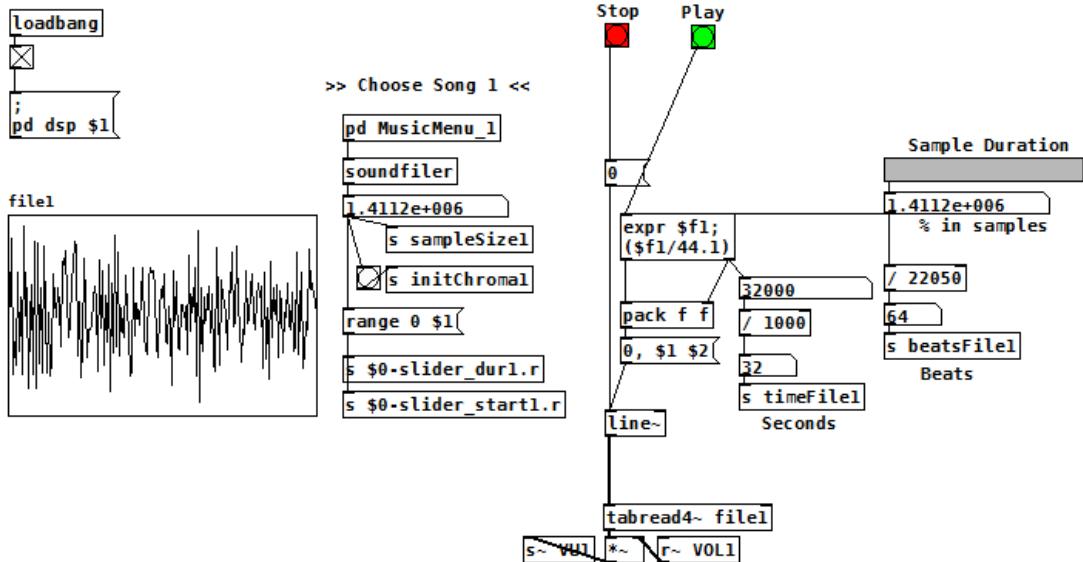


Figure 4.14: Screenshot from *rebeat*’s *Pd* framework: “load section 1” with respective “Play” and “Stop” functions.

The system has an available list of music excerpts (see Figure 4.15) to be used in the audio manipulation functionalities provided in *rebeat*’s framework. Both channels have access to the same music excerpts. When a file is selected, its respective waveform representation

is stored in a table. The *Pd* object `soundfiler`, provides a sample representation of the audio file total length, which can be used to calculate the respective beat size and duration (see Figure 4.14).

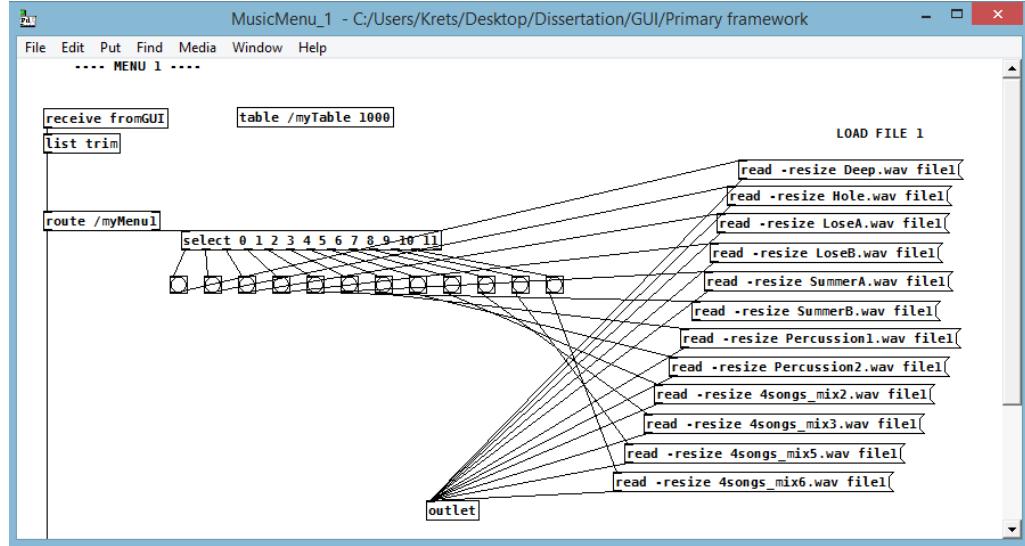


Figure 4.15: Screenshot from *rebeat*'s *Pd* framework: list of the available audio files in “load station 1”.

The `line~` object generates linear ramps whose levels and timing are determined by sending messages to it. These messages may be a target value in samples (causing the output to jump to the target) or a target and a time in milliseconds (to start a new ramp). In *rebeat*'s framework, when a song is loaded, the `line~` object indicates to the `tabread4~` object that the total length of the current loaded audio file, can be played. The `tabread4~` object will read from its table the values indicated by the `line~` object and send that information to a `dac~` object, responsible for providing real-time (stereo) audio output by converting the digital signal into analog audio output.

- **Play/Stop Songs:**

In each channel there are “PLAY” and “STOP” buttons (see Figure 4.13). By pressing the “PLAY” button, the system will play the entire length of the previously selected music excerpt in that channel. At any moment the user may press the “STOP” button to interrupt the song. When pressing the “PLAY” button again, the song will start playing again from the beginning.

There are also “PLAY BOTH” and “STOP BOTH” buttons (see Figures 4.12 and 4.16). When pressing the “PLAY BOTH” button, both songs from the two channels will play simultaneously. Considering that all music excerpts available have the same length (with

fixed tempo and the same number of beats), when pressing this button, both songs will be beat-synchronized, resulting in a mixed audio output, for which only synchronisation in time is enforced (i.e., no model of harmonic information). The “STOP BOTH” button works as a turn-off audio switch. Pressing this button will stop any song that is currently playing, either the output originated by channel “Music 1”, channel “Music 2” or both.

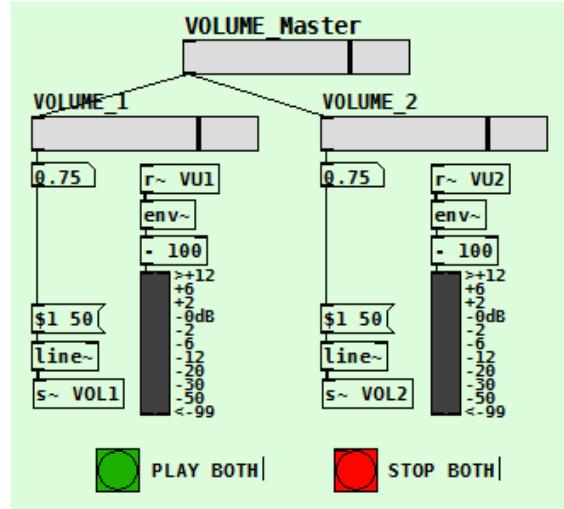


Figure 4.16: Screenshot from *rebeat*'s *Pd* framework: “VOLUME Master”, “VOLUME 1”, “VOLUME 2”, “Play Both” and “Stop Both”.

- **Volume:**

Each of the channels has an associated volume controller to control it’s audio output. At any time, while playing a song or not, the audio output volume of each channel can be increased or decreased, using the sliders “Volume 1” or “Volume 2” (see Figure 4.12).

Although each panel allows separate control of its output volume level, a “Master Volume” slider was implemented in both panels of *rebeat*'s GUI (see Figures 4.12 and 4.21), in case one requires to manipulate the volume of both songs (see Figure 4.16). This slider controls the volume values of the individual volume sliders, i.e., changing the value of this slider changes the individual sliders to the same volume value.

- **Go to Panel 2:**

Sliding the screen on top of the label “PANEL 2 ->” situated in the lower right corner of “Panel 1” (see Figure 4.12), leads to “Panel 2”, described in section ??.

#### 4.4.2.2 Audio Manipulations

In terms of interactive audio manipulations, it was decided to implement in each channel, two specific DJ oriented functionalities – a 3-band equalizer and a high/low filter control – that can be found in most comercial DJ mixing boards (see Figure 2.2).

- **3-band Equalizer:**

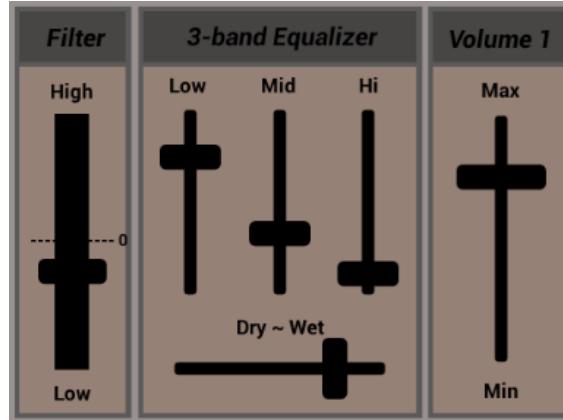


Figure 4.17: Screenshot from Panel 1 – “Filter”, “3-band Equalizer” and “Volume 1” sliders. In this example, the high and mid-range frequencies of the audio output are being attenuated.

A 3-band equalizer’s purpose is to attenuate desired frequency bands from an audio file in order to emphasize desired sound components in the audio output. Using the specific sliders – “Low”, “Mid” or “Hi” – it is possible to increase or decrease the attenuation of, respectively, low, medium and high frequencies. The “Dry-Wet” slider serves as an intensity control of the implemented filters, and works with the following response logic – “Wet” increases the filters intensity (slider to the right) and on the other hand, “Dry” inhibits the filters action (slider to the left). For example, if the user only wishes to listen to the lower frequencies (relative to drums and bass sounds) of a music excerpt, the procedure would be to increase the “Low” slider and decrease the “Mid” and “Hi” sliders (see Figure 4.17). By default, when a user opens *rebeat*, all the sliders are at half value – (see Figure 4.12).

In order to attenuate different frequency bands with desired intensity, the use of second-order recursive linear filters, specifically biquadratic filters (2 poles and 2 zeros), was required – see Equation 4.3.

$$\text{IIR Raw Biquad Filter : } H(z) = \frac{b_0 + b_1 \cdot z^{-1} + b_2 \cdot z^{-2}}{a_0 + a_1 \cdot z^{-1} + a_2 \cdot z^{-2}} \quad (4.3)$$

The coefficient values of the biquadratic filters, were adapted from a *Pd* patch of a 3-band equalizer developed by the media-artist Derek Holzer<sup>1</sup>. The resulting 3-band equalizer implemented in *rebeat*’s framework is shown in Figures 4.18 and 4.19).

---

<sup>1</sup><http://www.umatic.nl/>

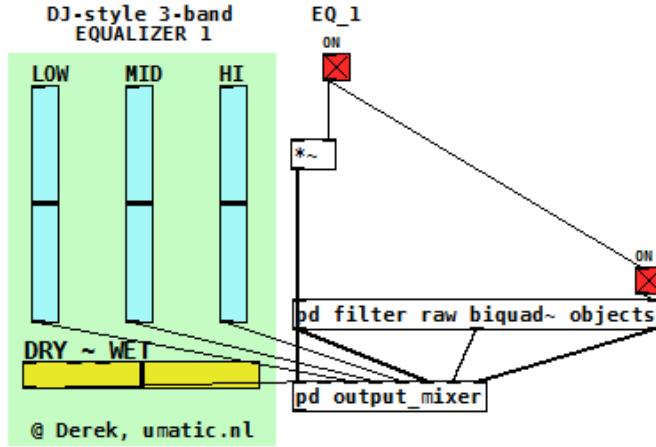


Figure 4.18: Screenshot from *rebeat*'s *Pd* framework: “DJ-style 3-band Equalizer 1”.

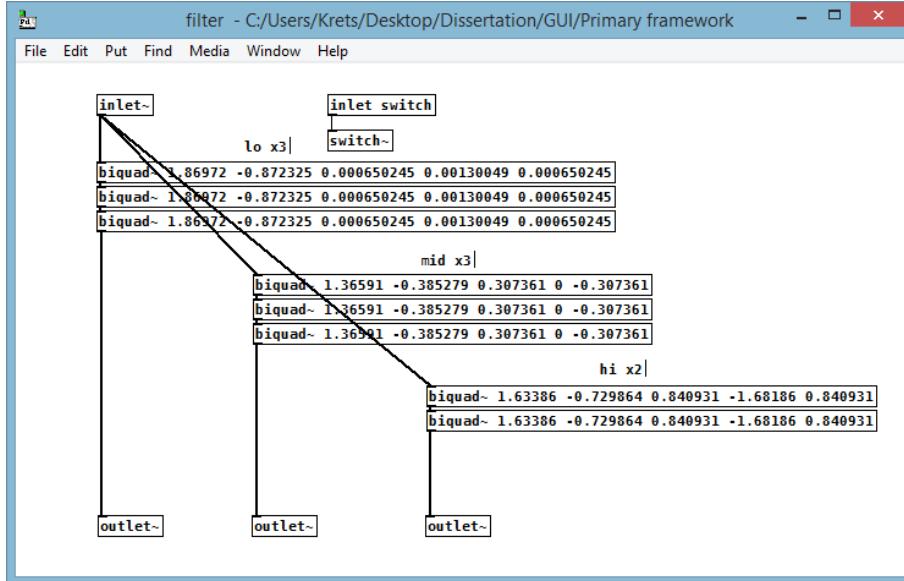


Figure 4.19: Screenshot from *rebeat*'s *Pd* framework: biquadratic filters implementation in the “DJ-style 3-band Equalizer 1”

#### • Filter:

Also inspired in commercial DJ mixing boards, each channel has a “Filter” slider. Using a single slider, the user is able to isolate high or low frequencies (see Figure 4.17). When a user opens *rebeat*, by default the slider is at half position, i.e., 0, meaning that no filter is applied in the audio output.

The purpose is to quickly attenuate high or low frequency components from the audio file, and it is ideal to perform transitions between different sounds or music excerpts. In commercial mixing boards, the controller is usually a knob where turning it to the left activates

low-pass filtering, and turning it to the write activates high-pass filtering. In *rebeat*'s framework, the filters were implemented using a single slider, as shown in Figure 4.20.

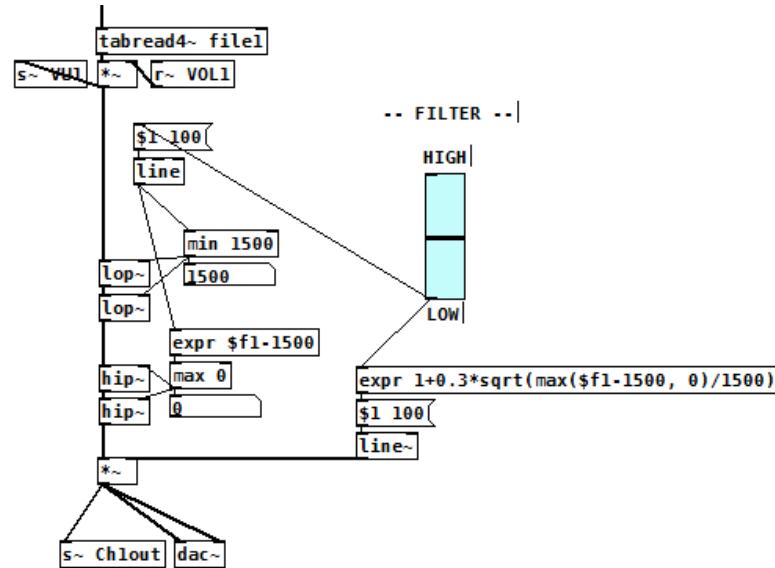


Figure 4.20: Screenshot from *rebeat*'s Pd framework: high-pass/low-pass filtering slider.

### 4.4.3 Mashup Panel

“Mashup Panel” (see Figure 4.21), includes the main novel technical contributions of the dissertation. It allows simple user interaction over the mashup creation possibilities implemented in *rebeat*'s framework.

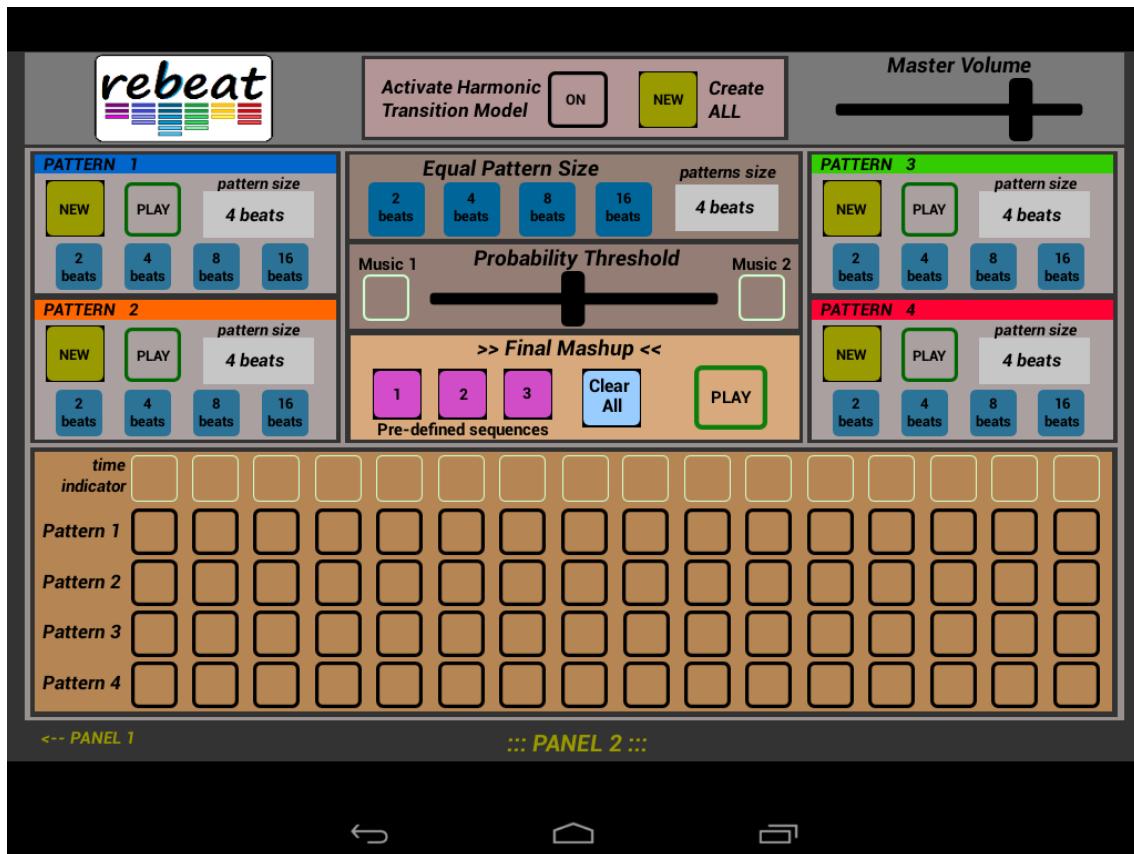


Figure 4.21: *rebeat* GUI: "Panel 2" or "Mashup Panel"

"Mashup Panel" is divided into different sections, where the user can work with up to four different patterns by manipulating functionalities like pattern creation, pattern features manipulation and long term pattern sequencing.

What follows is a description of the interactive functions available.

#### 4.4.3.1 Create and Play Mashup

- **Create ALL:**

In order to provide easy experimentation to first time users, when "Panel 2" is presented, the system allows for the creation of an entire mashup with a single touch in the interface. Each time the "Create ALL" button (see Figure 4.22) is pressed, the system creates and stores four patterns (different from each other) according to the pattern characteristics. Also, by pressing this button, a pre-defined sequence appears ready to be played in the "Final Mashup Grid" (see Figure 4.23).



Figure 4.22: Screenshot from Panel 2: “Create ALL” button.

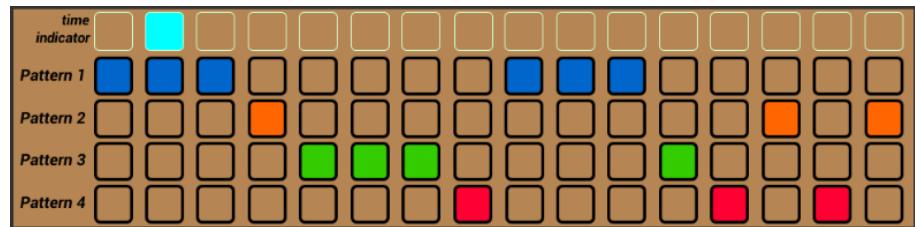


Figure 4.23: Screenshot from Panel 2: “Final Mashup Grid” with a sequence ready to be played.

- **Play/Stop Final Mashup:**

In order to play the mashup created (i.e., a sequence of patterns selected in the “Final Mashup Grid”) there is available a “PLAY” button in the “Final Mashup” section (see Figure 4.21). Pressing it will play the defined sequence of patterns. The “Final Mashup Grid” will play in loop (all sixteen blocks) until the “PLAY” button is pressed again to stop it.

The “time indicator” flashes in the “Final Mashup Grid” (see Figure 4.23) which was implemented to provide visual information about what is playing at each moment indicating the time progression of the pattern that is currently being played in that column.

- **Play/Stop Pattern:**

Each pattern, after being created, can be played at any time by pressing the “PLAY” button, which turns green when pressed (see Figure 4.24) and loops the respective pattern until the button is pressed again, turning the green highlight off.

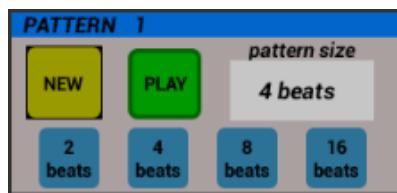


Figure 4.24: Screenshot from Panel 2: “NEW” and “PLAY” buttons, and “pattern size” buttons and display from “Pattern 1” section.

- **Long Term Pattern Sequencing:**

*rebeat*’s higher level of user interaction in terms of music creation is offered by the “Final Mashup Grid” (see Figure 4.25) in “Panel 2”. In order to give some creation responsibility

to the user, the system incorporates a composition grid to perform long term sequencing of patterns in time, being this action based solely on the user's decision.

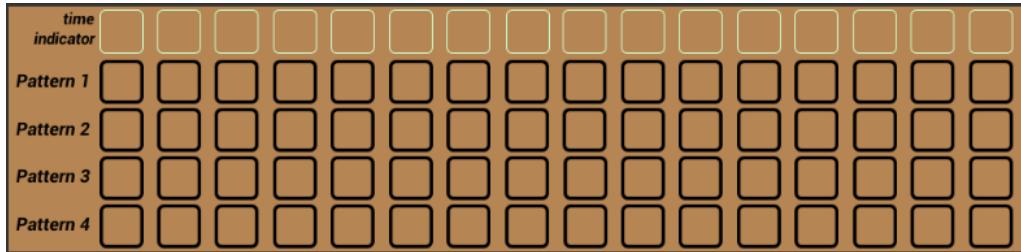


Figure 4.25: Screenshot from Panel 2: “Final Mashup Grid”.

The system allows sequencing in time of patterns along the 16 slot columns available. Each column (or time block) has four slots available at any time, corresponding to the four patterns. Any combination of patterns is possible, which provides a great number of possible sequences, but it is only possible to select one pattern (slot) per time block, meaning that in each moment there's only one pattern to be played (see Table 4.4).

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Pattern 1	x	x	x		x	x	x									
Pattern 2				x				x								
Pattern 3									x	x	x		x	x	x	
Pattern 4											x					x

Table 4.4: Example of a valid sequence, i.e., one pattern per time block (column).

With the purpose of providing some automatic sequencing possibilities, the system provides 3 different pre-defined sequences to use in the “Final Mashup Grid”. By pressing one of the buttons – “1”, “2” or “3” (see Figure 4.26) – available in the “Final Mashup” section, the respective sequence will appear ready to be played (see Figure 4.27).

In order to avoid the time consumption it would take to manually clear the cells from the “Final Mashup Grid”, a “Clear All” button was implemented (see Figure 4.26). Pressing it will automatically clear all 64 cells from the grid, and this way, it's possible to create new sequences faster.

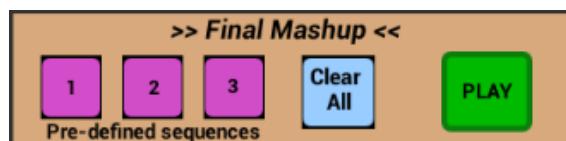


Figure 4.26: Screenshot from Panel 2: “Pre-defined sequences”, “Clear All” and “PLAY” buttons.

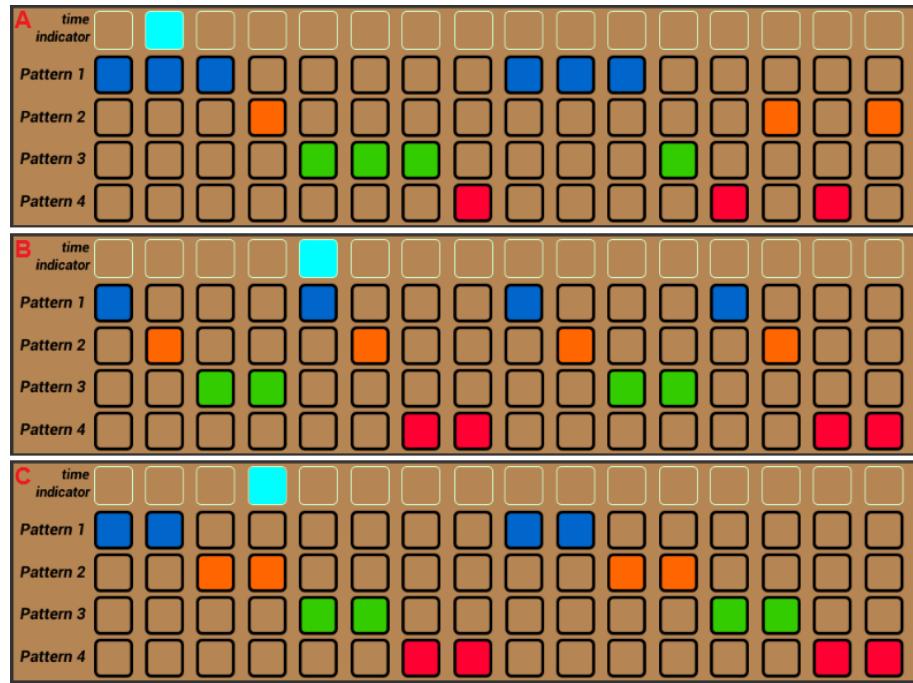


Figure 4.27: Screenshots from Panel 2: “Final Mashup” grid with the “Pre-defined sequence” 1 (A), 2 (B) and 3 (C) selected. The “time indicator” flash indicates which pattern was playing when the screenshot was taken.

After a complete sequence of patterns is defined, the system allows to play it in loop, i.e., the sixteen blocks (patterns) will be played one after the other, respecting the block progression, and that after playing the last pattern (block 16) the system starts playing again the first pattern (block 1), and so on.

The *rebeat* system allows the creation of patterns with different sizes and therefore, the mashup grid is prepared to play sequences of patterns with different sizes. This means that there’s a minimum and maximum composition length available (see Table 4.5). If all four patterns have two beats length, the final mashup length will be  $16 \times 2$  beats = 32 beats, the equivalent to 16 seconds at 120 BPM. On the other hand, if all four patterns have sixteen beats length, the final mashup length will be  $16 \times 16$  beats = 256 beats, i.e., 128 seconds at the same tempo.

	<b>Pattern Size (beats)</b>	<b>Composition Length (beats)</b>	<b>Composition Duration (seconds)</b>
<b>Minimum</b>	2	$16 \times 2 = 32$	16
<b>Maximum</b>	16	$16 \times 16 = 256$	128

Table 4.5: Minimum and maximum durations available in the final mashup composition grid.

### - Example of how the sequenced patterns are played

As an illustrative example, let's assume a smaller sequence (fewer time blocks to play), e.g. – Pattern 1, Pattern 2, Pattern 3, Pattern 1. In terms of pattern sizes, lets assume the following values:

- Pattern 1 = 4 beats
- Pattern 2 = 2 beats
- Pattern 3 = 8 beats

The Figure 4.28 represents an illustration of the mentioned situation. The first pattern to be played is “Pattern 1” with four beats, meaning that two seconds later, “Pattern 2” will be played. “Pattern 2” has two beats, so one second later is “Pattern 3” the one to be played. “Pattern 3” has eighth beats, so four seconds later begins “Pattern 1” again, finishing two seconds later. This way, the total number of beats played is equal to eighteen, which takes exactly nine seconds.

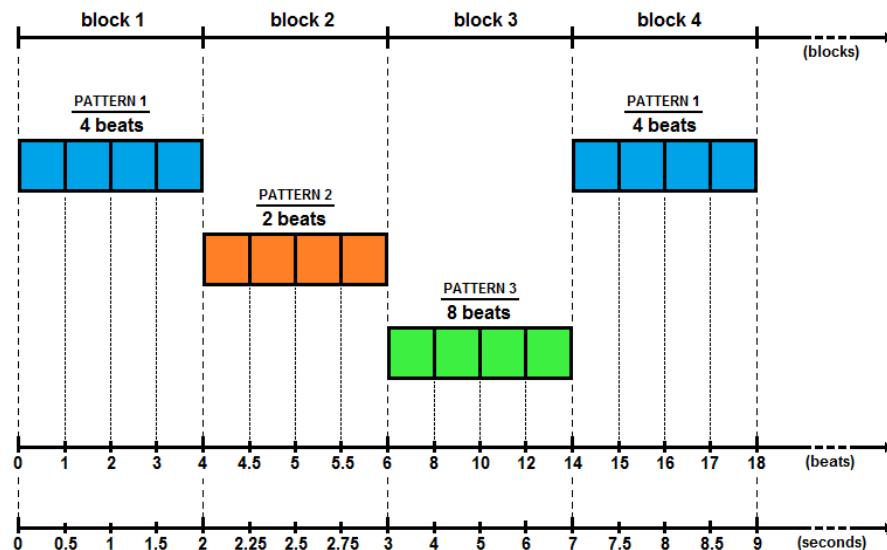


Figure 4.28: Illustration of how the system plays the sequenced patterns with the correspondent beat and time progression.

Another behaviour worth mentioning is the ability to read and write in real-time the patterns information stored in the backend tables (as described in section 4.4.3.2). This means that it is possible to have a complete sequence of patterns being played, while at the same time the patterns can be created or changed, meaning that any manipulation performed within a pattern will be updated so the changes are perceived in the final audio output as they are being made.

For example, if the user selects an equal size of four beats for all patterns, the final mashup length to be played would be 16 blocks times four beats resulting in 64 beats, equivalent to 32 seconds. Meanwhile, if the sequence is being played and the user selects an equal size

of two beats for all patterns, automatically the patterns are updated in the mashup sequence and played accordingly. The result is a final sequence of 16 blocks times two beats, equal to 32 beats (16 seconds), i.e., half length and duration than the previous situation.

#### 4.4.3.2 Change Patterns

- **New Pattern:**

In each pattern section of the interface, there is a “NEW” button (see Figure 4.24) that creates and stores a new pattern in the backend each time it’s pressed. The pattern is created according to the pattern characteristics selected, or, according to the default values of the system if none manipulation has been made.

To increase visual interaction and allow easy distinction between patterns, each pattern was associated with a different colour – blue for “Pattern 1”, orange for “Pattern 2”, green for “Pattern 3” and red for “Pattern 4”. This way, when the “NEW” button in each pattern is pressed, its respective pattern colour gets highlighted (see example in Figure 4.29).

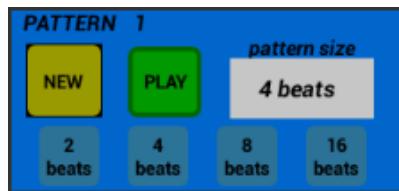


Figure 4.29: Screenshot from Panel 2: When pressing the “NEW” button in “Pattern 1”, its pattern colour (blue) gets highlighted.

- **Pattern Size:**

Pressing one of the buttons – “2 beats”, “4 beats”, “8 beats” or “16 beats” – will display in the respective pattern the selected size under “pattern size” label (see Figure 4.24).

In case the user wishes to select the same pattern size for the four patterns, there is the possibility of performing that interaction via one single touch on the interface. By pressing one of the buttons – “2 beats”, “4 beats”, “8 beats” or “16 beats” – in the “Equal Pattern Size” section (see Figure 4.30), all the patterns will be with the same size. The selected value will be displayed under the “patterns size” label and at the same time, the “pattern size” display of each pattern will update its value, matching the “patterns size” value from the “Equal Pattern Size” section.



Figure 4.30: Screenshot from Panel 2: “Equal Pattern Size” buttons and “patterns size” display.

- **Activate Harmonic Transition Model:**

When opening *rebeat*, the default method of pattern creation is based on musical structure features as described in section 4.3.2.2. However, the system also has another method of pattern creation, based on HPCP measures, as described in section 4.3.2.3. Pressing the “Activate Harmonic Transition Model” button (see Figure 4.31) available in “Panel 2”, will activate the referred method, and as long as the button is turned on, all pattern creation operations and patterns features, will be performed and changed according to that method’s decisions.

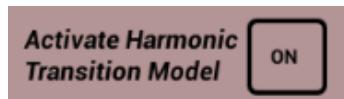


Figure 4.31: Screenshot from Panel 2: “Activate Harmonic Transition Model” button.

- **Probability Threshold:**

- Harmonic Transition Model Off

The “Probability Threshold” (see Figure 4.32), as described in section 4.3.2.2, is a slider that allows the user to change the probability of which music – “Music 1” or “Music 2” – will be selected in each of the four beat slices selection that compose each pattern.

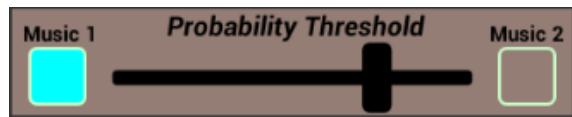


Figure 4.32: Screenshot from Panel 2: “Probability Threshold” slider; “Music 1” and “Music” flash indicators.

- Harmonic Transition Model On

When the “Activate Harmonic Transition Model” button is pressed, the purpose of the “Probability Threshold” slider (see Figure 4.32) available in “Panel 2” is to change the probability of being selected the beat from the best beat match in music 1 or the beat from the best beat match from music 2 (as described in section 4.3.2.3).

In the “Probability Threshold” section, there are also two flash indicators, “Music 1” and “Music 2” (see Figure 4.32). While a pattern or the “Final Mashup Grid” is playing, these indicators will flash in real-time indicating which music is being played. In this way, the user gets a visual interaction that provides some information about the source of the sounds he/she is listening.

# Chapter 5

## Evaluation

### 5.1 Pattern creation methods

In order to evaluate the performance of the pattern creation algorithms described in section 4.3.2, a listening experimentation test was designed. The main purpose was to collect data in order to evaluate the performance of the *rebeat* system possibilities without continuous experimentation. Since *rebeat* is a mobile application oriented for music mashup experimentation, the target audience goes from inexperienced users to professional musicians and electronic music enthusiasts/performers. With that in mind, the test was made publicly available and was closed with 38 participants, including inexperienced users, amateur performers and professional/experienced users.

The test audio files were stored in a *Soundcloud* account<sup>1</sup> created for that purpose. In order to have a rating based system, a survey was designed using the online tools of *Polldaddy*<sup>2</sup>, which allows the direct implementation of the survey on WordPress websites. The experimentation test was hosted on the *rebeat* webpage<sup>3</sup> and as mentioned before, anyone could take it. The results were collected and analysed.

#### 5.1.1 Experiment design

As presented in section 4.3.2, although the algorithms directly influence pattern creation, they are independent from the final mashup pattern sequencing. The pattern sequencing in time allows the user to create a great number of possible combinations with four patterns, meaning that it is technically possible to create musically incoherent or unpleasant pattern sequences.

It was decided to evaluate the performance of each one of the four different pattern creation algorithms presented in Chapter 4 – see Table 5.1).

To equally evaluate the four algorithms, all the sound examples were created while respecting equal features (see Table 5.2). Also, in terms of pattern sequencing in time, the same composition

---

<sup>1</sup><https://soundcloud.com/rebeatmobileapp>

<sup>2</sup><https://polldaddy.com/>

<sup>3</sup><https://rebeatmobileapp.wordpress.com/test/>

<b>Algorithm</b>		<b>Description</b>
<b>1</b>	Baseline Random Transition Model	Random bar and beat selection
<b>2</b>	Metrical Structure Based Transition Model	Random bar selection, maintaining beat progression
<b>3</b>	Chroma Based Transition Model 1	Beat-match calculated at current beat
<b>4</b>	Chroma Based Transition Model 2	Beat-match calculated at next beat

Table 5.1: List of algorithms submitted to the experimentation test.

structure was used in the creation of all test mashup sequences (see Table 5.3). All these options could be performed by any user with the *rebeat* mobile app (see Figure 5.1).

<b>Algorithm</b>	<b>Pattern Creation Features</b>		
	<b>Pattern Size</b>	<b>Probability Threshold</b>	<b>Harmonic Transition Model</b>
<b>1</b>	4 beats	50%	OFF
<b>2</b>	4 beats	50%	OFF
<b>3</b>	4 beats	50%	ON
<b>4</b>	4 beats	50%	ON

Table 5.2: Experimentation test: pattern creation features.

<b>Algorithm</b>	<b>Long Term Composition</b>			
	<b>Pattern Size</b>	<b>Pattern Duration</b>	<b>Pattern Sequence</b>	<b>Sequence Duration</b>
<b>1</b>	4 beats	2 sec	<b>A - A - B - B</b>	8 sec
<b>2</b>	4 beats	2 sec	<b>A - A - B - B</b>	8 sec
<b>3</b>	4 beats	2 sec	<b>A - A - B - B</b>	8 sec
<b>4</b>	4 beats	2 sec	<b>A - A - B - B</b>	8 sec

Table 5.3: Experimentation test: long term composition.

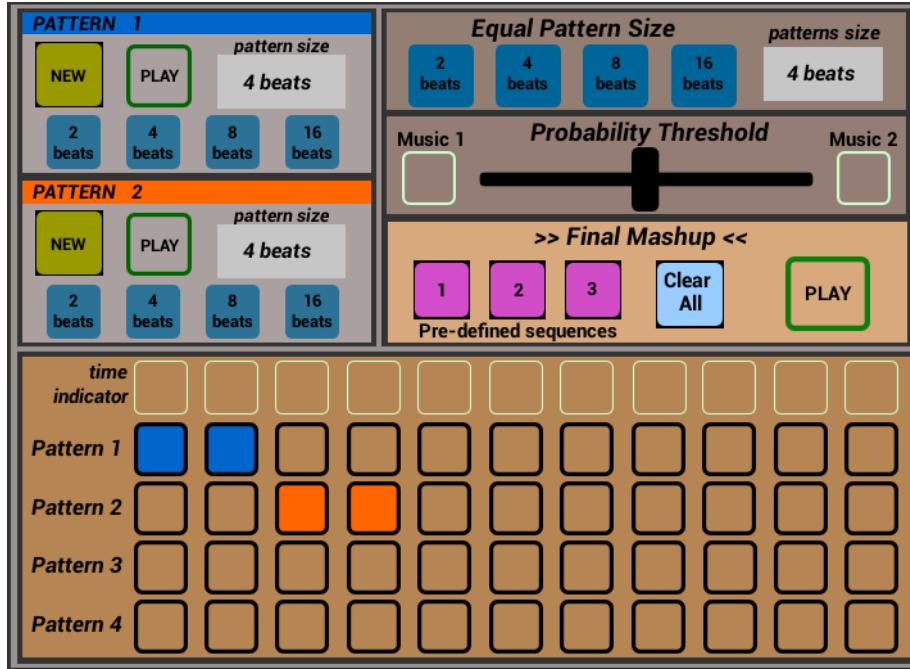


Figure 5.1: Screenshot of *rebeat* mobile app with the above features selected.

Within the *rebeat* dataset (see section 4.2), 10 music excerpts were randomly selected in pairs. Within each pair, two audio examples were created using each algorithm. The result is 40 test sound examples (T1, T2 … T40) with the same long term composition structure and duration of eight seconds each, generated by four different algorithms (see Table 5.4). The aim of the evaluation is to replicate the first impressions of a user without continued experimentation.

Pair	Tracks		Algorithm 1		Algorithm 2		Algorithm 3		Algorithm 4	
1	#0	Deep	T1	T2	T11	T12	T21	T22	T31	T32
	#4	SummerA								
2	#1	Hole	T3	T4	T13	T14	T23	T24	T33	T34
	#13	4songs_mix6								
3	#2	LoseA	T5	T6	T15	T16	T25	T26	T35	T36
	#5	SummerB								
4	#3	LoseB	T7	T8	T17	T18	T27	T28	T37	T38
	#9	4songs_mix2								
5	#10	4songs_mix3	T9	T10	T19	T20	T29	T30	T39	T40
	#12	4songs_mix5								

Table 5.4: List of sound examples recorded with the parttern creation algorithms.

When accessing the experimentation test on the *rebeat* webpage, the participants were presented with some guidance information and instructions (see Figure 5.2).

## TEST

One of the main features of *rebeat* is the possibility of real-time user-guided pattern creation. The system incorporates different algorithms which apply different methods of pattern creation. In order to test the algorithms performance, some patterns were recorded and assembled into sequences to design the following experimentation test.

**:: IMPORTANT ::** Please read all instructions carefully before starting the experiment.

- In the following experiment, you will be presented small audio examples and will be asked to rate them in terms of how pleasing they sound.
- Each example lasts for 8 seconds. In total there are 40 examples and the experiment should take no more than 10-12 minutes to complete.
- Your participation in this experiment is completely voluntary. If you wish to stop participating, you may do so at any time.
- Please take the experiment in a quiet environment with the use of headphones.
- To begin the experiment please click the "Take Test!" button below.

**Take Test!**

For help support please contact one of the [authors](#).

Figure 5.2: Experimentation test: screenshot of the “Test” page on *rebeat* webpage.

After reading the above information and clicking in the “Take test!” button available, a popup would appear to the participant notifying him that the resulting rating data would be anonymously used when reporting experimental results (see Figure 5.3).



Figure 5.3: Experimentation test: screenshot of the consent confirmation popup.

By clicking the “Start Test” button the first question would appear where the participant was

asked to indicate his/her level of musical background by selecting one of the three available categories (see Figure 5.4).

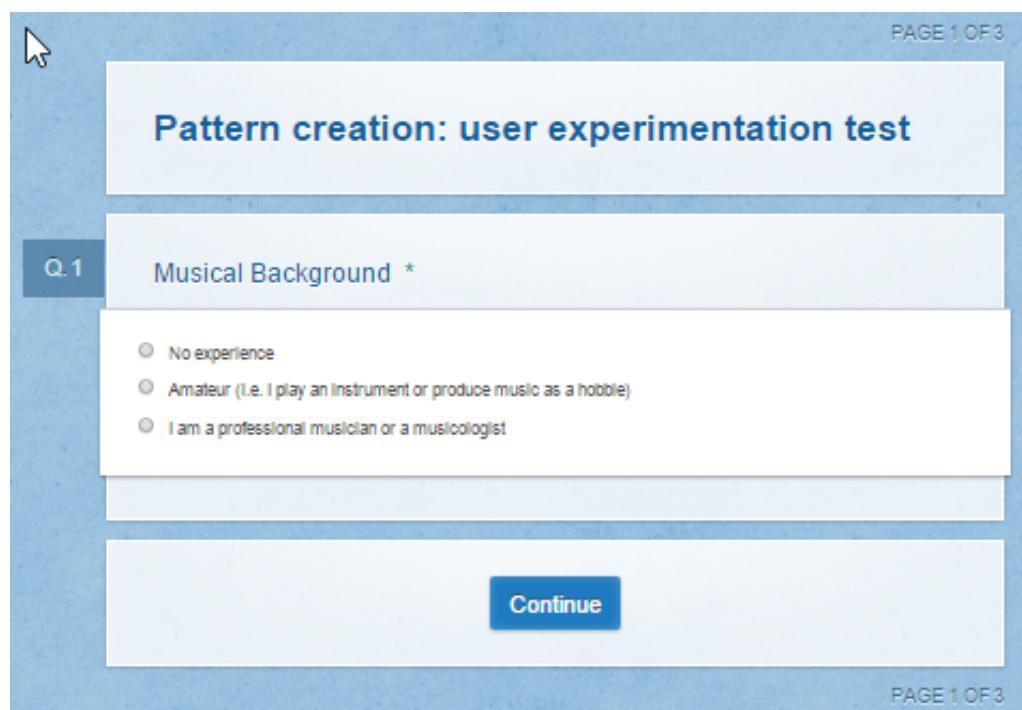


Figure 5.4: Experimentation test: screenshot of the musical background question.

Then the 40 test sequences were presented in a different random order for each participant, to prevent any dependence between different excerpts. The participant was able to listen to the sequences as many times as desired but it was mandatory to give ratings to all of them. As mentioned before, each sequence was rated in terms of how pleasing they sounded by choosing one of the available options – “Bad”, “Poor”, “Acceptable”, “Good” and “Very Good” (see Figure 5.5).

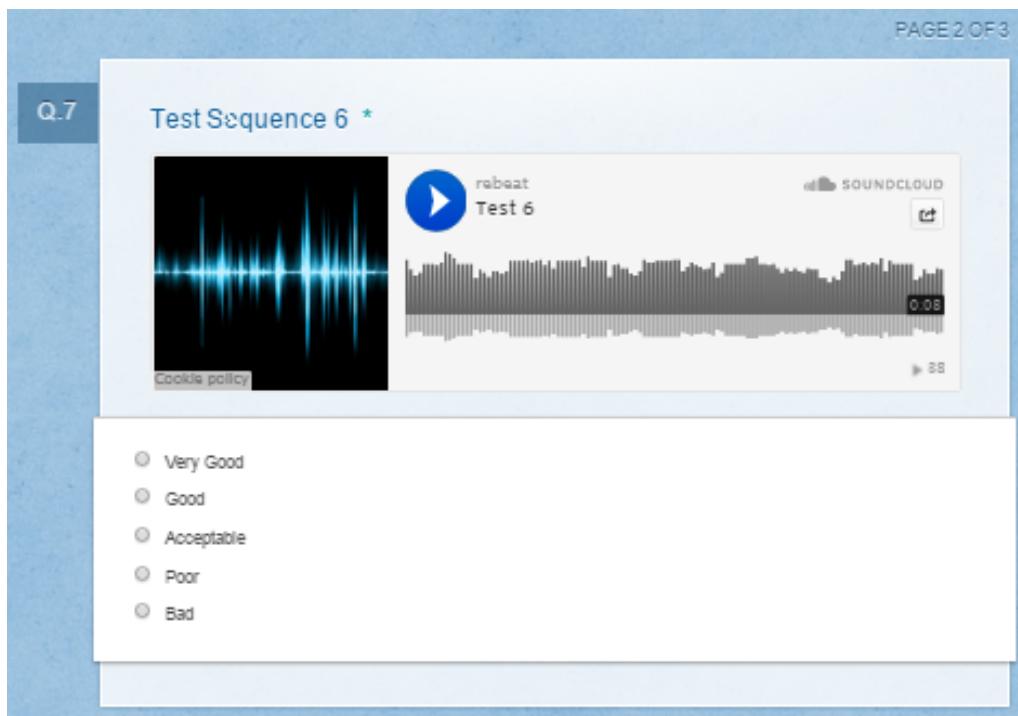


Figure 5.5: Experimentation test: screenshot of a sequence rating question.

### 5.1.2 Results

The ratings results were gathered and used to generate evaluations from different perspectives, based on a specific weight that was associated to each rating as shown in Table 5.5.

	Weight
<b>Very Good</b>	5
<b>Good</b>	4
<b>Acceptable</b>	3
<b>Poor</b>	2
<b>Bad</b>	1

Table 5.5: Experimentation test: weight attributed to each rating option.

The percentage of ratings per algorithm, allowed to generate a final average algorithms evaluation, as shown in Table 5.6 and Figure 5.8.

	Evaluation (0 to 5)
Algorithm 1	<b>2.85</b>
Algorithm 2	<b>3.13</b>
Algorithm 3	<b>3.17</b>
Algorithm 4	<b>3.22</b>

Table 5.6: Experimentation test: average algorithms evaluation results.

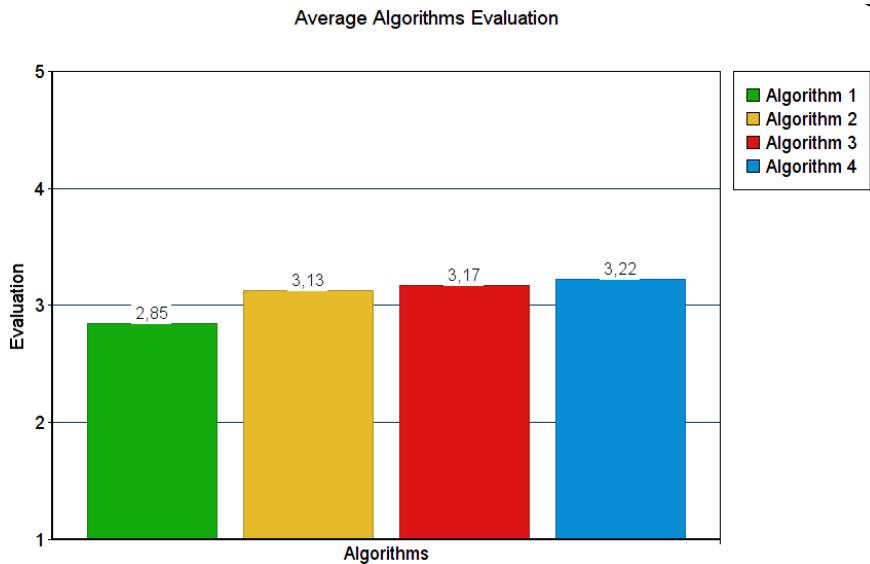


Figure 5.6: Experimentation test: average algorithms evaluation graphic.

Based on these final results there are some conclusions that can be made:

- Algorithm 1 presents a lower overall rating than the others. Considering that this algorithm performs a completely random pattern creation (see section 4.3.2.1), this result was expected. That being said, analysing the values from Table 5.6, it's noticeable that not all test examples generated by Algorithm 1, were perceived and evaluated as unpleasant. Actually, regarding “Pair 3”, all algorithms were evaluated with lower ratings than the other pairs and the evaluation of Algorithm 1 is quite similar with the others. This leads to the conclusion that, even without any transition model applied, it is possible to obtain, acceptable or even good results with random pattern creation, with the same probability of obtaining bad or really incoherent results. Also, it can be observed that although Algorithm 2 had slightly lower ratings than Algorithm 3 and 4, it shows a noticeable improvement over Algorithm 1. Considering that it was intended to demonstrate that music structure provide acceptable results in terms of pleasantness, it can be concluded that the algorithm performance is as

expected.

In this way, regarding the algorithms of pattern creation that aren't based in the harmonic transition model (i.e. Algorithm 1 and Algorithm 2), it was decided to implement Algorithm 2 in *rebeat*'s framework, considering it offers higher probability of obtaining pleasant patterns when comparing with Algorithm 1.

- Algorithm 3 and Algorithm 4 were evaluated with the highest ratings so it is fair to conclude that they are more likely to provide better results than the other algorithms. As described in section 4.3.2.3, these two algorithms are both based on the same harmonic transition model, wherein the only difference lies in the beat index where the next best matches per beat will be calculated. That being said, it was expected to obtain very similar results between these two algorithms, as shown in Figure 5.6. By comparing them directly, it is noticeable that Algorithm 4 presents only slightly higher ratings than Algorithm 3, so, although in general the difference between the performance of both of them is not meaningful, it was decided to implement Algorithm 4 in *rebeat* system.

The participants evaluation of each test example from Table 5.4 is shown in Table 5.7. The bold and italic values refer to, respectively, maximum and minimum evaluations per pair of songs. Analysing individually each pair, it was noted that the Algorithm 1 had the worst evaluation in four of the five pairs. On the other hand, Algorithm 4 had the best evaluation in three of the five pairs, but also having the worst evaluation of “Pair 3”. In terms of average ratings per pairs of songs, the graphic on Figure 5.7 can be consulted.

Pair	Tracks		Evaluation (0 to 5)							
			Algorithm 1		Algorithm 2		Algorithm 3		Algorithm 4	
1	#0	Deep	T1	T2	T11	T12	T21	T22	T31	T32
	#4	SummerA	3.29	2.79	<b>3.39</b>	3.18	3.00	3.21	3.24	3.32
2	#1	Hole	T3	T4	T13	T14	T23	T24	T33	T34
	#13	4songs_mix6	2.95	3.03	3.05	2.97	<b>3.84</b>	2.97	3.21	3.61
3	#2	LoseA	T5	T6	T15	T16	T25	T26	T35	T36
	#5	SummerB	2.66	2.45	2.58	<b>2.76</b>	2.68	2.58	2.74	2.37
4	#3	LoseB	T7	T8	T17	T18	T27	T28	T37	T38
	#9	4songs_mix2	2.37	2.79	3.05	<b>3.61</b>	3.58	3.11	<b>3.61</b>	3.13
5	#10	4songs_mix3	T9	T10	T19	T20	T29	T30	T39	T40
	#12	4songs_mix5	3.08	3.13	3.26	3.45	3.42	3.29	<b>3.87</b>	3.13

Table 5.7: Experimentation test: average evaluation per pairs of songs.

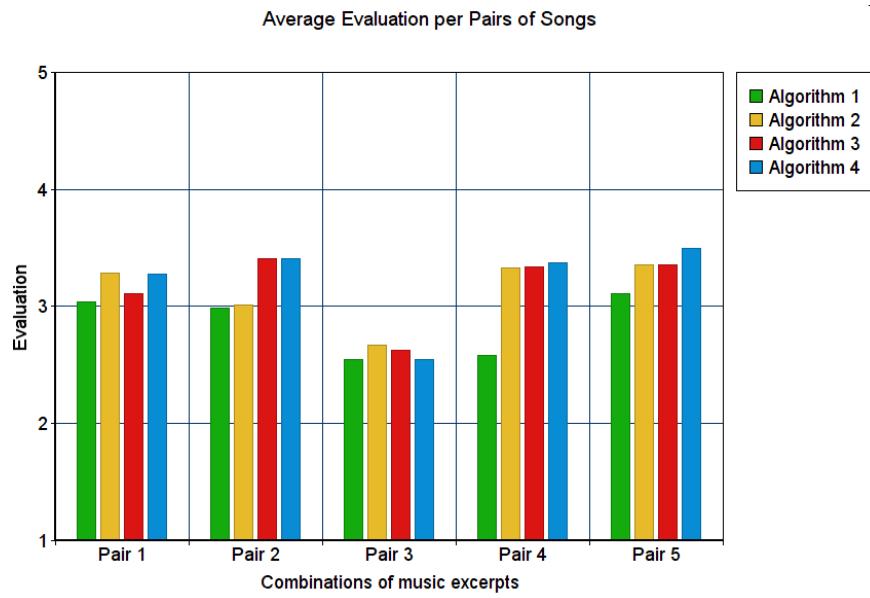


Figure 5.7: Experimentation test: average evaluation per pairs of songs graphic.

In the above graphic can be observed that “Pair 3”, has the worst average evaluation regardless the algorithm. Concerning the other pairs, Algorithm 3 and Algorithm 4 gather the higher evaluations, being closely followed by Algorithm 2.

Considering that the average evaluation is positive (above 2.5 out of 5), this results leads to believe that the evaluation of a algorithm doesn't solely rely on its performance but also depends on the music excerpts chosen. This means, as expected, that not all combinations of musics can be used to perform mashups. There will always be pairs of songs that match well together and pairs that not so much.

In another perspective, the test results can also be analysed in terms of percentage of ratings per algorithm (see Table 5.8 and Figure 5.8).

	<b>Algorithm 1</b> (T1 to T10)	<b>Algorithm 2</b> (T11 to T20)	<b>Algorithm 3</b> (T21 to T30)	<b>Algorithm 4</b> (T31 to T40)
<b>Very Good</b>	9.2	12.6	<b>13.9</b>	12.9
<b>Good</b>	16.8	23.2	23.4	<b>28.2</b>
<b>Acceptable</b>	34.2	<b>35.0</b>	34.2	34.5
<b>Poor</b>	<b>29.5</b>	23.2	22.4	17.1
<b>Bad</b>	<b>10.3</b>	6.1	6.1	7.4

Table 5.8: Experimentation test: percentage of ratings per algorithm.

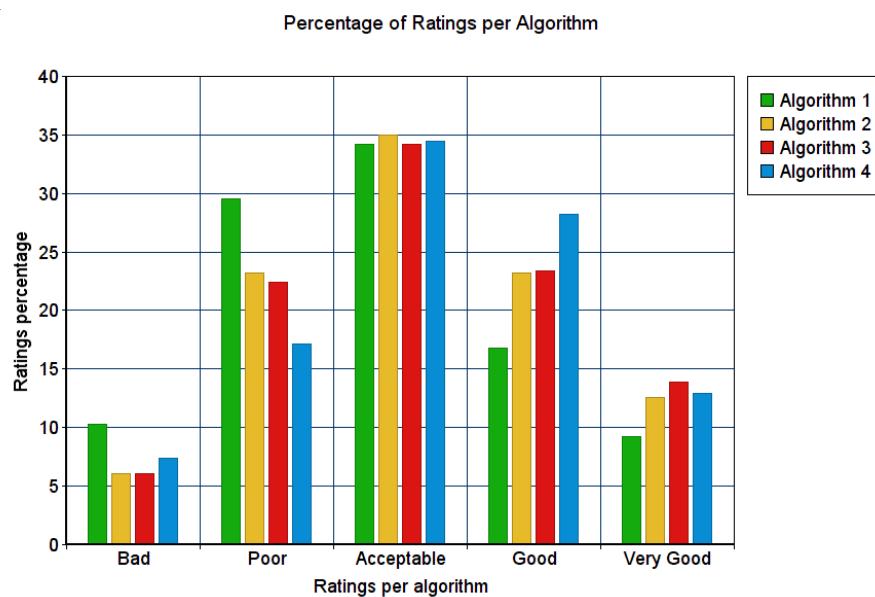


Figure 5.8: Experimentation test: percentage of ratings per algorithm graphic.

Analysing the above values, Algorithm 1 has the highest percentage of “Bad” and “Poor” ratings, as expected – considering that this algorithm performs completely random pattern creation. In terms of “Acceptable” ratings, there is not a meaningful difference between any of the algorithms. Algorithm 4 had the highest percentage of “Good” ratings and Algorithm 3 the highest percentage of “Very Good” ratings.

These results provide a more clear distinction of the algorithms in terms of user preference. The algorithms based on MIR techniques – such as Algorithm 3 and Algorithm 4 (see section 4.3.2.3) – were shown to provide a higher level of pleasantness than Algorithm 1, devoid of any kind of harmonic or structure features. Also as expected, the algorithm based on music structure characteristics – Algorithm 2 (see section 4.3.2.2) –, was often evaluated as “Acceptable” and “Good”.

# Chapter 6

## Conclusions

### 6.1 Contributions

The aim of the *rebeat* system is to offer non-expert users, music mashup experimentation with high-level user-guidance functions, via a GUI that runs on mobile devices. Through a user-friendly GUI, the user is allowed to load music excerpts from the available dataset, manipulate them and experiment in different levels of mashup creation. The first level refers to patterns creation, where the user interacts with the system by guiding it into real-time pattern creation. The second level of mashup creation relies on the user's decision, where he/she is allowed to perform long term sequencing of patterns through an interactive composition grid. Every user interaction provided by the system implies several operations and MIR calculations that are performed client-side in real-time, where any manipulation performed will update the system so the changes can be heard in the final audio output as they are being made.

Concerning the listening experimentation test submitted to the pattern creation algorithms, it shows that the transition models implemented in *rebeat* system increase the general pleasantness level of created patterns, comparing to the completely random pattern creation algorithm. The audio examples generated with the harmonic transition model based in HPCP measures were evaluated with the highest level of pleasantness, which demonstrates how useful systems implementing MIR techniques can be by providing guidance in music composition and audio manipulation.

The test also revealed that even with completely random pattern creation, the general pleasantness of the generated patterns, was perceived and evaluated as acceptable. However, since the random composition of patterns has no transition model, good performance cannot be guaranteed. In summary, the overall evaluation of *rebeat* system is positive, which is reinforced due the implementation of methods to aid in pattern creation.

## 6.2 Future Work

While developing this dissertation, the number of new possibilities and different approaches emerging was remarkable. Developing a mobile application offers countless possibilities in terms of user-interaction, functionalities and visual interface design. During the development process of each component of *rebeat* system, mainly due to time restrictions, decisions had to be made, goals were defined and therefore, many possible improvements and ideas for new features were held back. What follows is a brief presentation of possible improvements to be made in each of *rebeat*'s components.

- **Music dataset features**

The music excerpts available in the dataset were prepared in order to share equal features such as a defined length of 16 bars and a specific tempo of 120 BPM. A valid future work improvement is to allow the system to use music excerpts with different lengths and tempo. One solution could be the implementation of real-time time-stretching operations, with the purpose of changing the BPM of any music excerpt to a desired value without changing its harmonic content. In order for the system to continue to provide client-side beat-synchronization, the BPM of both songs would still be required to be the same for both, but with this implementation it would be possible to experiment in mashup creations in different tempos, which increases the possibilities within the creation process by making different music genres and styles available.

Another relevant restriction of *rebeat* is the impossibility for the user to upload music from their personal music collection. This is justified by the fact that the user interface implemented with *MobMuPlat* standalone app only allows direct communication with the *Pd* framework implemented. The incapability of *MobMuPlat* to interact with the mobile device, currently prevents the chance to use different music content. Hence, it's also not possible to record and store in the device the mashups created. Possible solutions could be to implement the framework using another audio oriented programming language or perhaps the adoption of a different kind of GUI design tool, or wait for new features in *MobMuPlat* that may be presented in the future. If *rebeat* were to allow the upload of any music excerpt from the device and allow to store the created projects for further listening or manipulation, the experimentation possibilities and gratification achieved through user interaction would probably increase.

- **Framework**

According to the listening experimentation test results performed on the pattern creation algorithms, it was fair to conclude that in fact the implementation of high-level harmonic transition functions based in MIR techniques, increased the quality and pleasantness of possible patterns created. Also, the results regarding the music structure based algorithm were positive, being evaluated almost at the same level as the MIR based algorithm. A possible future improvement could be the implementation of a pattern creation algorithm based

both in music structure rules and in transition techniques resulting from MIR functions. Considering the large variety of possible audio features manipulations resulting from MIR techniques, it's also plausible to assume that if more high-level guidance functions were to be implemented in the system, the possible results from creative processes would be richer in terms of musical coherence and pleasantness. For example, pitch-shifting operations could also be implemented, in order to increase ways in which harmonic transitions can be found.

- **User-interface**

The GUI was designed with *MobMuplat* tools and the decision to use this standalone application was justified by the fact that it provides continuous OSC messaging between the interface and the framework, allowing to perform the functionalities implemented in *Pd* through real-time touch interactions in a mobile device. The final design and performance of the GUI was not submitted to an user experimentation test since it was considered a priority to evaluate the pattern creation aspect of *rebeat*. By allowing first-time users to experiment with *rebeat* mobile app, important information could be gathered and analysed in order to perform specific improvements in the GUI design.

The final GUI implemented was divided in two panels – “Load Panel” and “Mashup Panel” –, offering, via user interaction, several audio manipulation and mashup creation possibilities. A useful improvement to be made in the future would be to implement a third panel – “Help Panel” – in the final GUI, that could present simple descriptions/explanations about the available user-oriented functions of *rebeat*. This panel would provide a simple and direct assistance to first-time users in case of any doubts that may emerge while experimenting with *rebeat* application.

The GUI was designed for non-expert users so, the user-interactions implemented were designed in order to allow easy and intuitive experimentation. That being said, a possible development for the future could be to reinvent the system's approach in terms of user interaction and functionalities with the purpose of offering even easier experimentation through basic functions performed via touch interaction. If the GUI were to be designed on a different platform that could provide richer visual contents and interactive functions (such as multi-touch, drag-and-drop, gyroscope sensor, etc.) the target market for this system would probably increase, if for example even children were inclined to use it.



# Bibliography

- [1] Miguel Alonso, Bertrand David, and Gaël Richard. Tempo and beat estimation of musical signals. In *Proceedings of ISMIR*, pages 158–163, 2004.
- [2] Timothy Beamish, Karon Maclean, and Sidney Fels. Manipulating music: multimodal interaction for DJs. *Conference on Human Factors in Computing Systems*, 6(1):327–334, 2004.
- [3] Matthew E. P. Davies, Philippe Hamel, Kazuyoshi Yoshii, and Masataka Goto. AutoMashUpper: Automatic Creation of Multi-Song Music Mashups. *IEEE/ACM Transactions on Audio Speech and Language Processing*, 22(12):1726–1737, 2014.
- [4] Matthew E. P. Davies and Mark D. Plumbley. A Spectral Difference Approach to Downbeat Extraction in Musical Audio. In *Proceedings of EUSIPCO*, 2006.
- [5] Matthew E. P. Davies, Adam Stark, Fabien Gouyon, and Masataka Goto. Improvasher: A Real-Time Mashup System for Live Musical Input. *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 541–544, 2014.
- [6] Garth Griffin, Youngmoo E. Kim, and Douglas R. Turnbull. Beat-Synchronous Music Mashups. In *Proceedings of ICASSP*, pages 437–440, 2010.
- [7] Jason Hockman, Matthew E. P. Davies, and Ichiro Fujinaga. One in the Jungle: Downbeat Detection in Hardcore, Jungle, and Drum and Bass. In *Proceedings of ISMIR*, pages 169–174, 2012.
- [8] Max V. Mathews. The Digital Computer as a Musical Instrument. *Science*, 142(3592):553–557, 1963.
- [9] Matthias Mauch and Simon Dixon. Simultaneous estimation of chords and musical context from audio. *IEEE Transactions on Audio, Speech and Language Processing*, 18(6):1280–1289, 2010.
- [10] Katy Noland and Mark B. Sandler. “Key Estimation Using a Hidden Markov Model”. In *Proceedings of ISMIR*, pages 121–126, 2006.
- [11] Steffen Pauws. “Musical key extraction from audio.”. In *Proceedings of ISMIR*, pages 96–99, 2004.

- [12] Geoffroy Peeters. “Time variable tempo detection and beat marking”. In *Proceedings of ICMC*, 2005.
- [13] Geoffroy Peeters. “Chroma-based estimation of musical key from audio-signal analysis”. In *Proceedings of ISMIR*, pages 155–120, 2006.
- [14] Emmanuel Ravelli, Mark Sandler, and Juan P. Bello. “Fast implementation for non-linear time-scaling of stereo signals”. In *Proceedings of DAFX*, pages 182–185, 2005.
- [15] Justin Salamon and Emilia Gómez. “A Chroma-based Salience Function for Melody and Bass Line Estimation from Music Audio Signals”. In *Proceedings of SMC*, pages 331 – 336, 2009.
- [16] João Santos. “Interpretação em tempo real sobre material sonoro pré-gravado”. Master’s thesis, University of Porto, 2014.
- [17] Christian Schörkhuber, Anssi Klapuri, and Alois Sontacchi. “Audio pitch shifting using the constant-Q transform”. *Journal of the Audio Engineering Society*, 61(7):562–572, 2013.
- [18] Xavier Serra, Michela Magas, Emmanouil Benetos, Magdalena Chudy, S Dixon, Arthur Flexer, Emilia Gómez, F Gouyon, P Herrera, S Jordà, Oscar Paytuvi, G Peeters, Jan Schlüter, H Vinet, and G Widmer. *Roadmap for Music Information Research*. 2013.
- [19] Mark Zadel and Gary Scavone. “Laptop performance: Techniques, tools, and a new interface design”. In *Proceedings of ICMC*, pages 643–648, 2006.