Alexandre Braga Saldanha
Nathália Marques de Oliveira

Recomendação dinâmica de músicas baseada em agrupamentos

São Paulo
2015

Alexandre Braga Saldanha
Nathália Marques de Oliveira

Recomendação dinâmica de músicas baseada em agrupamentos

Trabalho apresentado para conclusão da graduação no curso de Engenharia da Computação da Escola Politécnica da Universidade de São Paulo.

São Paulo
2015

Alexandre Braga Saldanha
Nathália Marques de Oliveira

Recomendação dinâmica de músicas baseada em agrupamentos

Trabalho apresentado para conclusão da graduação no curso de Engenharia da Computação da Escola Politécnica da Universidade de São Paulo.

Área de Concentração:
Engenharia da Computação

Orientador: Prof. Dr. Jaime Simão Sichman
Coorientadora: Profª Drª Katti Faceli

São Paulo
2015

Resumo

Os serviços de recomendação musical mais utilizados atualmente têm como foco proporcionar ao usuário uma forma de descoberta de novas músicas, utilizando informações extraídas da sua base de usuários, como as músicas favoritas de pessoas com gostos semelhantes e as músicas mais populares entre seus amigos. No entanto, ainda existem muitas pessoas que mantêm grandes coleções musicais em seus dispositivos pessoais e não possuem formas automatizadas de organizar essas coleções ou receber recomendações adequadas ao seu perfil de uso. Este trabalho pretende atender a essas pessoas, utilizando técnicas de agrupamento e o *feedback* do usuário durante a reprodução para sugerir ao usuário o que ouvir em seguida.

Palavras-chave: Recomendação de músicas, geração de playlists, music information retrieval.

Abstract

Current popular music recommendation services help the user to discover new songs, based mainly on social information, i.e. favorite songs of people with similar tastes, or friends' favorite songs. However, there are still many people that keep large song collections on their personal devices and they can't organize automatically these collections or get recommendations suited to certain listening profiles. This project aims to help those people, using song similarity, clustering and implicit feedback during playback to suggest to the user what to hear next.

Keywords: Music recommendation, playlist generation, music information retrieval.

Lista de ilustrações

Sumário

1 Introdução

Com a crescente disponibilidade de arquivos de música em formato digital na internet, a área de pesquisa em Recuperação de Informações Musicais (MIR - *Music Information Retrieval*) têm se desenvolvido rapidamente. Diversos algoritmos e ferramentas têm sido propostos para organização de coleções musicais, recomendação de músicas e geração de *playlists*, classificação e agrupamento das músicas, transcrição de letras, busca por trechos, entre outras tarefas relacionadas. Essas tarefas envolvem não só algoritmos e técnicas computacionais, mas também processamento de sinais de áudio, teoria musical e estudos psicológicos, o que torna a área de pesquisa complexa e interdisciplinar (TZANETAKIS, 2015).

Há diversos serviços comerciais que utilizam técnicas de MIR para oferecer, principalmente, recomendações musicais aos usuários e possibilitar a descoberta de novas músicas. Dentre os serviços mais populares, podem ser citados Spotify[1], Pandora[2] e Last.fm[3] (LEE; WATERMAN, 2012). Os serviços de *streaming* têm se popularizado principalmente por funcionarem sob demanda, possibilitando que o usuário ouça as músicas que desejar no momento em que quiser, sem a necessidade de armazená-las previamente, desde que possua conexão à internet (LEE; WATERMAN, 2012).

Apesar da crescente popularidade de serviços de *streaming*, os *downloads* continuam sendo a principal fonte de lucro digital para gravadoras, somando milhões em vendas nas lojas digitais, como iTunes Store e Google Play (IFPI, 2015). Isso indica que, além dos usuários que já construíam suas coleções musicais digitais antes dessa popularização, muitos usuários continuam armazenando suas músicas localmente e construindo grandes coleções.

Com essas grandes coleções, surge o problema de explorá-las e organizá-las adequadamente. A tarefa manual de seleção de faixas e criação de *playlists* demanda muito tempo e dedicação por parte do usuário. Por outro lado, a função de *shuffle* dos *players* de música, aplicada à totalidade das músicas do usuário, por ser baseada em aleatoriedade, gera sequências geralmente desprovidas de relação entre as faixas, o que faz com que o usuário "pule" diversas músicas enquanto ouve a sua coleção (PAMPALK; POHLE; WIDMER, 2005).

Diversos sistemas já foram propostos para auxiliar o usuário nessa tarefa, tanto para coleções locais (KING, 2014; PAMPALK; POHLE; WIDMER, 2005; PAMPALK; GASSER; TOMITSCH, 2007), quanto para grandes coleções em nuvem (Spotify, Pandora).

[1] <spotify.com>
[2] <pandora.com>
[3] <last.fm>

Como descrito em (SCHEDL et al., 2012), há diversos tipos de atributos que podem ser utilizados para quantificar a semelhança entre músicas, bem como diversas formas de combiná-los para tentar realizar uma previsão do que o usuário poderia desejar ouvir em seguida. No entanto, gostos pessoais são de difícil modelagem e uma alternativa ótima ainda não foi encontrada.

Este trabalho pretende fornecer uma nova alternativa, tendo como base os trabalhos já realizados na área. O objetivo geral do trabalho é desenvolver um novo sistema de recomendação musical que forneça músicas adequadas, isto é, que minimize o número de músicas rejeitadas pelo usuário. De maneira específica, pretende-se utilizar neste sistema agrupamentos de músicas, para determinar músicas semelhantes, e o *feedback* implícito do usuário, para que a recomendação se adapte ao usuário e ao mesmo tempo seja transparente a ele.

O restante deste documento está dividido da seguinte forma:

- O Capítulo 2 apresenta os aspectos teóricos que são base para este trabalho, incluindo a revisão de trabalhos semelhantes;
- O capítulo 3 apresenta a especificação do sistema;
- O capítulo 4 apresenta a metodologia utilizada em todas as fases do trabalho, incluindo o projeto e implementação;
- O capítulo 5 inclui maiores detalhes técnicos sobre a implementação, como os desafios encontrados e as soluções utilizadas;
- O capítulo 6 apresenta os resultados dos testes e avaliação do sistema implementado;
- No capítulo 7 são apresentadas as conclusões sobre o trabalho, bem como perspectivas de continuação.

2 Revisão bibliográfica

A partir da pesquisa de trabalhos relacionados na literatura, foram identificados três tópicos de maior relevância para este trabalho, sendo eles:

1. **Atributos musicais:** seleção e extração dos atributos mais relevantes para cada aplicação, como características de áudio ou metadados;
2. **Organização musical:** diferentes formas de classificação e agrupamento das faixas com base nos atributos selecionados;
3. **Recomendação musical:** estratégias de recomendação musical e geração automática de *playlists*.

Neste capítulo, serão detalhados cada um destes tópicos e como foram abordados nos trabalhos analisados.

2.1 Atributos musicais

Segundo (SCHEDL et al., 2012), um dos grandes desafios da área de MIR é o desenvolvimento de características computacionais que codifiquem conhecimento sobre a forma como as pessoas percebem uma música. Essa percepção é influenciada por diversos aspectos, como a preferência musical do ouvinte, bem como sua cultura e seu conhecimento musical. Além disso, informações relevantes sobre faixas musicais podem ser extraídas de várias fontes, como o próprio sinal de áudio, partituras, letras ou até mesmo capas de álbuns. Os autores categorizam as diversas características computacionais em três classes:

- **Conteúdo musical:** esta classe engloba características obtidas diretamente da mídia que contém a faixa musical, como o próprio sinal de áudio, partituras ou videoclipes.

  As características obtidas a partir do sinal de áudio são as mais utilizadas na área de MIR e são comumente obtidas a partir da aplicação de técnicas de processamento de sinais. Essas técnicas geram características de baixo nível, relativas a frequências e amplitudes do sinal, que, embora facilmente manipuláveis computacionalmente, não possuem muito significado para a maioria dos ouvintes. Essas características podem ser combinadas para a geração de descritores de nível de abstração mais alto, como timbre, ritmo, tom ou até mesmo humor e gênero;

- **Contexto musical:** os autores descrevem o contexto musical como toda informação relevante ao item musical em questão que não pode ser extraída diretamente da

mídia que o contém. Como exemplos dessas informações podem ser citados o país de origem do autor e a letra da música.

A extração dessas características é fortemente relacionada à mineração de dados da Web, por exemplo em perfis de artistas, *playlists* geradas por usuários e *tags* colaborativas;

- **Contexto do usuário:** para aplicações com foco nos usuários, informações de contexto podem ser relevantes. Exemplos dessas características são as atividades realizadas pelo usuário, aspectos fisiológicos (como pressão arterial e frequência cardíaca), contexto social, estado emocional e contexto espaço-temporal (como localização, horário e clima).

Nos trabalhos pesquisados, as características utilizadas foram predominantemente da classe de conteúdo musical.

Em (POHLE; PAMPALK; WIDMER, 2005) foram computados os coeficientes Mel-Cepstrais (MFCC) das músicas para o uso em um modelo de mistura Gaussiano. Espectros e padrões de flutuações descritos no trabalho realizado em (PAMPALK; FLEXER; WIDMER, 2005) são utilizados em (PAMPALK; POHLE; WIDMER, 2005; PAMPALK; GASSER; TOMITSCH, 2007) para uso em um modelo de agrupamento semelhante ao utilizado por *Self-Organizing Maps* (SOM) (KOHONEN, 1998).

Atributos relacionados a timbre (espectro), intensidade (raiz da média quadrática do sinal do áudio) e ritmo (força das batidas, regularidade e tempo) foram utilizados em (LIU; LU; ZHANG, 2003) para identificar o humor que as músicas causam.

Valores relacionados à sensação de volume e modulação do volume são utilizadas por (PAMPALK, 2006) para o treinamento de uma SOM.

Em (KING, 2014), foram utilizadas diversas características de áudio, como descritores do espectro, quantidade de *zero-crossing* e coeficientes Mel-Cepstrais.

Frameworks de extração de características de áudio

A extração de características de áudio é um assunto amplamente discutido e estudado na área de recuperação de informações musicais. Seu estudo é tão amplo, que há diversas bibliotecas e *frameworks* de licença aberta para utilização em projetos e pesquisas relacionadas ao tema; um exemplo pode ser visto em (KING, 2014), que utiliza o jAudio. Alguns dos *frameworks* existentes são:

- Essentia: biblioteca de licença aberta, escrita em C++, para análise de áudio e extração de informações musicais (BOGDANOV et al., 2013). Não é considerado um *framework* propriamente dito, mas sim um conjunto de algoritmos para a extração

de características do áudio. Sua implementação foi feita para computação de grandes bibliotecas de áudio. Os algoritmos presentes na biblioteca incluem o manuseio de arquivos de áudio, análise de áudio digital, filtros, algoritmos genéricos estatísticos para caracterização, e descritores para características de baixo nível, tal como o espectro. A biblioteca pode ser obtida no site <http://essentia.upf.edu>;

- Marsyas: *framework* de licença aberta, escrito em C++, para processamento de áudio com ênfase em aplicações de recuperação de informações musicais (TZANETAKIS; COOK, 2000). O objetivo principal do *framework* é prover uma arquitetura geral, extensiva e flexível para uso em pesquisas e experimentos de ferramentos para análise de áudio. Ele oferece diversas tarefas relacionadas ao manuseio de arquivos de áudio, tais como entrada e saída de áudio e arquivos de áudio, processamento de sinais e módulos de aprendizado de máquina. O *framework* pode ser obtido no site <http://marsyas.info/>;

- MIRtoolbox: conjunto de funções escritas em Matlab para a extração de características a partir de arquivos de áudio (LARTILLOT; TOIVIAINEN; EEROLA, 2008). Ele tem por objetivo a computação de uma grande quantidade de características de bibliotecas de áudio, que podem ser aplicados para análises estatísticas. Taxa de *zero-crossing*, espectro e tom são algumas das características que podem ser extraídas com a ferramenta. Ela pode ser obtida no site <https://www.jyu.fi/hum/laitokset/musiikki/en/research/coe/materials/mirtoolbox>;

- jAudio: pacote de software utilizado para a extração de características a partir de arquivos de áudio. Seu funcionamento é simples: fornecendo uma lista de arquivos de áudio como entrada, é retornada uma lista com os valores reais das características de cada um deles. O pacote pode realizar a extração de todas as características mencionadas na descrição dos *frameworks* anteriores. Ele pode ser obtido no site <http://sourceforge.net/projects/jmir/files/>.

2.2 Organização musical

Diversos métodos de organização musical podem ser empregados utilizando os atributos musicais mencionados anteriormente.

Uma das aplicações mais comuns destes atributos é a classificação das músicas em categorias, como gênero (TZANETAKIS, 2015). A detecção de características musicais de alto nível como humor ou até mesmo compositor e álbum também são amplamente estudadas e existe um *framework* especializado para avaliar tais sistemas[1].

[1] <http://www.music-ir.org/mirex/wiki/MIREX_HOME>

Em (WATSON; MANDRYK, 2012) são utilizadas características de áudio e informações contextuais do usuário para criar um modelo de humor musical. Em (LIU; LU; ZHANG, 2003), as características de timbre, intensidade e ritmo são utilizadas para classificar o humor de uma música conforme o plano bidimensional de Thayer, que define o humor musical nas escalas de valência (feliz/triste) e energia (calma/agitada).

Outra forma de organização utilizada é a medição de similaridades entre músicas com base em seus atributos. Em (PAMPALK; POHLE; WIDMER, 2005; PAMPALK; GASSER; TOMITSCH, 2007), a similaridade entre as músicas é medida através da distância Euclidiana entre os conjuntos de características extraídas e é utilizada uma abordagem semelhante à utilizada em SOMs para o agrupamento de músicas semelhantes.

Outro método de organização encontrado é a geração de *clusters* a partir dos atributos extraídos. Os sistemas Islands of Music (PAMPALK, 2006), BeatlesExplorer (STOBER; NÜRNBERGER, 2008) e SoniXplorer (LÜBBERS; JARKE, 2009) são sistemas que traduzem os agrupamentos musicais para representações visuais que simulam localizações geográficas. Tanto BeatlesExplorer quanto SoniXplorer permitem que o usuário altere a localização das faixas, alterando os pesos das características adequadamente. Em (KING, 2014) são gerados agrupamentos hierárquicos de músicas; adicionalmente, um algoritmo de *Reinforcement Learning* é utilizado para aprender as probabilidades de transição entre os *clusters*, a fim de selecionar as músicas com maior probabilidade de serem aceitas pelo usuário durante a reprodução ou na geração automática de *playlists*.

2.3 Recomendação de músicas

Diversos trabalhos na área também estão relacionados com uma forma inteligente e/ou automática de recomendação de músicas para o usuário.

Em (POHLE; PAMPALK; WIDMER, 2005) foi desenvolvido um trabalho em que o problema de se gerar *playlists* de músicas similares foi mapeado em um problema do caixeiro viajante. Assim, de acordo com esse mapeamento, temos que os vértices do grafo são as músicas na coleção, e que a similaridade entre elas são as arestas. Estas similaridades são calculadas a partir dos atributos extraídos das músicas, como explicado anteriormente. Para resolver o problema, três algoritmos foram propostos: um algoritmo guloso, que monta um caminho no grafo incrementalmente, examinando as arestas em ordem crescente de comprimento; um algoritmo de árvore de extensão mínima, onde, a partir da árvore geradora mínima encontrada, é realizada uma busca em profundidade para montar um caminho conectando os vértices na ordem em que eles são visitados; e uma versão otimizada do algoritmo de Lin-Kernighan, proposta em 1971. Validações foram feitas para cada um dos algoritmos propostos, verificando a eficiência e a qualidade das *playlists* geradas, bem como uma análise subjetiva dos mesmos, através da disponibilização

de um *applet* Java para que os usuários pudessem navegar pelas listas geradas e avaliá-las. Ressalta-se que, mantendo-se a mesma coleção, os algoritmos sempre retornaram a mesma *playlist*, isto é, a *playlist* gerada é estática.

Em (PAMPALK; POHLE; WIDMER, 2005) são apresentadas propostas sobre geração de *playlists* baseadas na similaridade entre as músicas. Foram propostas quatro heurísticas para a geração de uma sequência de músicas:

1. Dada uma música semente (escolhida pelo usuário ou de maneira aleatória), os n vizinhos mais próximos da semente são reproduzidos, sendo n a soma do número de músicas aceitas e rejeitadas pelo usuário;

2. A música candidata (música ainda não reproduzida) mais próxima da última música aceita pelo usuário é a próxima a ser reproduzida. Esta heurística é similar à anterior, com a diferença de que na heurística anterior a música semente é sempre a última música aceita;

3. A música candidata mais próxima de todas as outras músicas aceitas é a próxima a ser reproduzida;

4. Para cada música candidata, seja d_a a distância para a música aceita mais próxima, e seja d_s a distância para a música rejeitada mais próxima. Se $d_a < d_s$, adiciona-se a música candidata ao conjunto S. A partir de S, é reproduzida a música com menor d_a; se S for vazio, é reproduzida a música com menor relação d_a/d_s.

Com estas heurísticas em mãos, testes com casos de uso hipotéticos foram realizados para comprovar a eficiência de cada uma delas, mostrando que a eficiência aumenta da heurística 1 para a 4.

A partir deste trabalho, (PAMPALK; GASSER; TOMITSCH, 2007) propôs um trabalho baseado na heurística 4. Ao contrário do trabalho anterior, o usuário pode avaliar músicas e artistas. Uma variação da heurística 4, anteriormente explicada, foi proposta. Esta nova heurística recomenda as músicas mais próximas de todas as músicas avaliadas positivamente e mais distantes das músicas avaliadas negativamente. O funcionamento dessa heurística está ilustrado no algoritmo 1, empregando a seguinte notação:

- S_{pos}/S_{neg} - o conjunto de músicas avaliadas positivamente e negativamente, respectivamente;

- A_{pos}/A_{neg} - o conjunto de artistas avaliados positivamente e negativamente, respectivamente;

- S_{cand} - o conjunto de músicas candidatas, isto é, músicas ainda não reproduzidas;

- $songs(A)$ - busca de todos as músicas do artista A;
- $\Delta(s_1, s_2)$ - distância entre a música s_1 e a música s_2;
- $closest\_song(S, a)$ - encontra a música $s \in S$ com o menor valor $\Delta(s, a)$.

Algorithm 1 Gerador de *playlists*(PAMPALK; POHLE; WIDMER, 2005)

$A \leftarrow S_{pos} \cup (songs(A_{pos}) - S_{pos})$
$B \leftarrow S_{neg} \cup (songs(A_{neg}) - S_{pos})$
$C \leftarrow S_{cand} - B$
for all s in C **do**
    $d_a(s) \leftarrow \Delta(s, closest_song(A, s))$
    $d_b(s) \leftarrow \Delta(s, closest_song(B, s))$
    if $d_b(s) < d_a(s)$ **then**
        $C \leftarrow C - s$
if empty(C) **then**
    return $argmin(d_a(s)/d_b(s))$
else
    return $argmin(d_a(s))$

A validação do sistema proposto, diferentemente de (PAMPALK; POHLE; WIDMER, 2005), teve a participação de usuários reais, os quais testaram um *player* que implementava a heurística do algoritmo 1. Foi realizada a validação da usabilidade do sistema e da qualidade das *playlists* geradas.

King (KING, 2014) apresentou um sistema de geração dinâmica de *playlists* utilizando aprendizado por reforço (*Reinforcement Learning*). A ideia é que um agente receba recompensas baseadas em suas ações. Em seu trabalho, o autor utilizou o algoritmo Q-learning (WATKINS; DAYAN, 1992). O sistema proposto trabalha com um cálculo de recompensa r, baseado no tempo que o usuário ouviu a música corrente, onde $-1 < r < 1$. Com isso, o cálculo pode ser realizado em cinco cenários diferentes, de acordo com o *feedback* implícito do usuário e o tempo de execução de uma faixa musical:

Faixa de música terminada

Quando uma música é executada até o fim sem nenhuma interrupção do usuário, uma recompensa positiva é atribuída;

Música pulada

Quanto mais cedo uma faixa de música foi pulada pelo usuário, menor será o valor negativo de recompensa. Portanto, tomando como $r = -1.0$ como o menor valor negativo de recompensa quando a música é pulada com 0 segundos, e $r = 1.0$ como o maior valor positivo de recompensa quando a música não é pulada, um cálculo

linear entre estes dois valores é realizado de acordo com o instante em que a música foi pulada;

Escolha de uma nova música

Um caso similar ocorre quando o usuário escolhe outra música para reproduzir. Neste caso, considera-se que o usuário pulou a música que estava sendo reproduzida, e é realizado o mesmo cálculo de recompensa descrito no item anterior. Além disso, é atribuída uma grande recompensa positiva à música escolhida pelo usuário, pois o autor considera que este é o *feedback* implícito mais poderoso que o usuário pode fornecer;

Adição à fila

O *player* apresentado pelo autor permite que se adicione músicas à fila de execução. Neste caso, diferentemente do caso anterior, o usuário não interrompe a reprodução da faixa atual; dessa forma, há a atribuição de dois valores positivos de recompensa: um para a faixa que está sendo reproduzida e não foi pulada, e outro para a faixa que foi adicionada à fila;

Playlists

Outra funcionalidade que o *player* do autor apresenta é a importação, criação e modificação de *playlists* feitas manualmente pelo usuário. Neste caso, um pequeno valor positivo de recompensa é atribuído para todas as músicas presentes na lista, e um valor positivo maior é atribuído a músicas consecutivas da lista.

O sistema utiliza *clusters* hierárquicos de músicas, gerados a partir de características extraídas do áudio. O algoritmo de aprendizado utiliza as recompensas descritas acima para aprender as probabilidades de transição entre os *clusters*. Assim, durante a reprodução no modo chamado pelo autor de "smart mode" ou na criação de *playlists*, a próxima música a ser reproduzida ou incluída na lista é selecionada da seguinte forma: navega-se pelos *clusters* de maior probabilidade a partir do caminho até a música atual; ao chegar num *cluster* que possui apenas folhas (músicas), utiliza-se uma função heurística para escolha da música.

Essa função heurística incorpora: (i) a distância Euclidiana entre a música sendo analisada e a última música tocada; (ii) as músicas mais tocadas; (iii) uma lista do histórico de reprodução imediato. Cada uma dessas parcelas é somada aos "votos" de uma faixa e a faixa com maior valor de votos é escolhida.

Para validação do sistema, o autor realizou testes com usuários e testes para medição dos *clusters* gerados. No primeiro caso, ao disponibilizar um aplicativo contendo o sistema para um certo número de usuários, foram coletados dados de uso para determinar o número de vezes que o usuário pula uma música conforme o uso do aplicativo, bem

como uma pesquisa de satisfação sobre o sistema. Já para o segundo caso, experimentos de validação no próprio sistema foram realizados para verificar se o mesmo realizava suas tarefas corretamente.

3 Especificação

Neste capítulo, será apresentada a proposta deste trabalho e sua especificação. Primeiramente, na seção 3.1, será apresentada uma visão geral do sistema e de suas principais funcionalidades; nas seções 3.2 e 3.3 serão listados os requisitos funcionais e não funcionais levantados; na seção 3.4 serão descritos os casos de uso; na seção 3.5, será apresentada a visão de alto nível da arquitetura proposta; na seção 3.6, será mostrado o diagrama de classes do sistema juntamente com uma breve explicação das classes definidas; na seção 3.7 será apresentado o algoritmo de recomendação utilizado; na seção 3.8 serão apresentados os *wireframes* da interface gráfica; por fim, na seção 3.9, será apresentada uma tabela com as escolhas de tecnologias feitas para o projeto, acompanhada de uma breve justificativa.

As alterações realizadas em relação à especificação original apresentada no primeiro semestre serão explicitadas em cada uma das seções, quando houver.

3.1 Visão geral

O sistema utilizará agrupamentos baseados em características de áudio bem como o *feedback* implícito do usuário para sugerir músicas durante a reprodução.

Em relação à especificação apresentada anteriormente, foi removida a etapa de adaptação dos *clusters*.

De maneira geral, o sistema terá o seguinte funcionamento, também ilustrado na Figura 1:

- Inicialmente, os arquivos de áudio serão carregados no sistema.

- Suas características serão extraídas, produzindo um conjunto de características para cada faixa.

  As características utilizadas serão características de conteúdo musical, conforme categorias descritas na seção 2.1. Para seleção de quais características a serem extraídas, foram utilizadas as referências mencionadas na mesma seção. Para extração, foi utilizada uma biblioteca já existente, como descrito na seção 3.9.

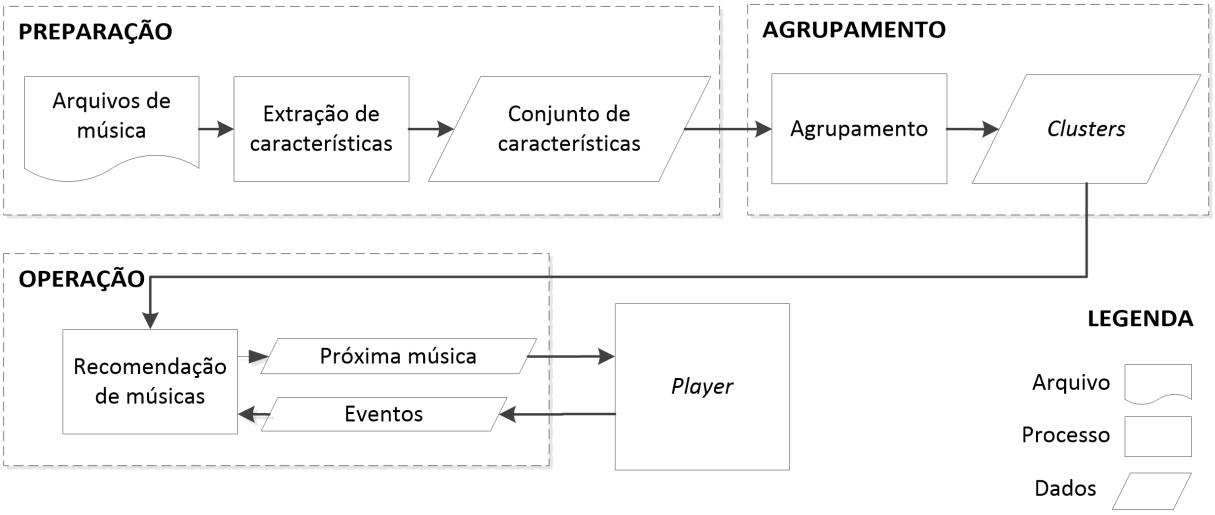
- Esses conjuntos serão enviados para o algoritmo de agrupamento, que produzirá os *clusters* correspondentes.

Como descrito na seção 3.9, inicialmente será utilizado o algoritmo k-médias (RUSSEL; NORVIG, 2013). Outros trabalhos que utilizam *clustering* são mencionados na seção 2.2.

- Estes *clusters* serão utilizados pelo algoritmo de sugestão de música durante a reprodução, bem como o *feedback* implícito do usuário.

  Foi proposto um novo algoritmo de sugestão, com base nos algoritmos pesquisados, descrito na seção 3.7.

Figura 1 – Diagrama de funcionamento do sistema



3.2 Requisitos funcionais

Os requisitos funcionais do sistema são listados abaixo.

Em relação à especificação apresentada originalmente, o requisito 2 foi alterado para a inclusão do *player* no sistema a ser desenvolvido, ao invés da comunicação com um *player* existente. Além disso, foi removido o requisito referente à adaptação dos *clusters* gerados.

RF1. O sistema deverá aceitar arquivos de música no formato mp3;

RF2. O sistema deverá reproduzir arquivos de música adicionados a ele e possuir interface para receber do usuário os comandos de *play*, *pause*, *skip* e *repeat*.

RF3. O sistema deverá indexar todas as músicas adicionadas ao *player* que estejam no formato suportado;

RF4. O sistema deverá extrair e armazenar características de áudio para todas as músicas indexadas;

RF5. O sistema deverá gerar *clusters* de músicas a partir das características extraídas e armazenadas;

RF6. O sistema deverá enviar ao *player* a nova música a ser tocada ao receber eventos de término de uma música, pulo de uma música ou início da reprodução;

RF7. O sistema deverá armazenar a sequência de músicas tocadas e eventos recebidos;

RF8. A nova música a ser tocada deverá ser produzida como função dos *clusters* de músicas, da sequência de músicas tocadas e da sequência de eventos recebidos.

3.3 Requisitos não funcionais

São apresentados a seguir os requisitos não funcionais do sistema. Os valores utilizados para os intervalos de tempo são estimativas de valores de espera aceitáveis para o usuário em cada um dos casos. O tamanho da coleção e a duração média das faixas são restrições impostas para atingir o desempenho desejado.

Em relação à especificação apresentada originalmente, foram removidos os requisitos relacionados à adaptação dos *clusters* e ao desempenho da rotina de inicialização.

RNF1. O intervalo entre o recebimento dos eventos e o envio da próxima música ao *player* deve ser inferior a 20 segundos;

RNF2. A interação do sistema com o *player* não deve interferir na qualidade da reprodução das faixas;

3.4 Casos de Uso

Nas tabelas a seguir são descritos os casos de uso do sistema.

Em relação à especificação apresentada originalmente, foi adicionado o caso de uso 3 e removido o caso de uso relacionado à adaptação dos *clusters*.

Tabela 1 – UC01

Código	UC01
Nome	Adicionar músicas
Atores	Usuário
Descrição	Usuário adiciona músicas ao *player*.
Requisitos relacionados	RF1, RF3, RF4, RF5
Pré-Condições	Nenhuma
Fluxo básico de eventos	
Ações do ator	**Ações do sistema**
1. Usuário seleciona comando de adicionar músicas	
2. Usuário seleciona pastas de músicas para serem adicionadas	3. Sistema obtém das pastas selecionadas as músicas nos formatos suportados
	4. Sistema extrai características das músicas em segundo plano
	5. Ao término da extração, sistema gera os *clusters* a partir das características extraídas
	6. Ao ser terminado, sistema armazena as informações em disco
7. Fim do UC	

Tabela 2 – UC02

Código	UC02
Nome	Recomendar nova música
Atores	Nenhum
Descrição	Sistema recomenda uma nova música para o usuário.
Requisitos relacionados	RF6, RF8
Pré-Condições	Nenhuma
Fluxo básico de eventos	
Ações do ator	**Ações do sistema**
	1. Sistema recebe um evento de próxima música
	2. Sistema executa o algoritmo para determinar a próxima música
	3. Sistema passa ao *player* a próxima música a ser reproduzida
4. Fim do UC	

Tabela 3 – UC03

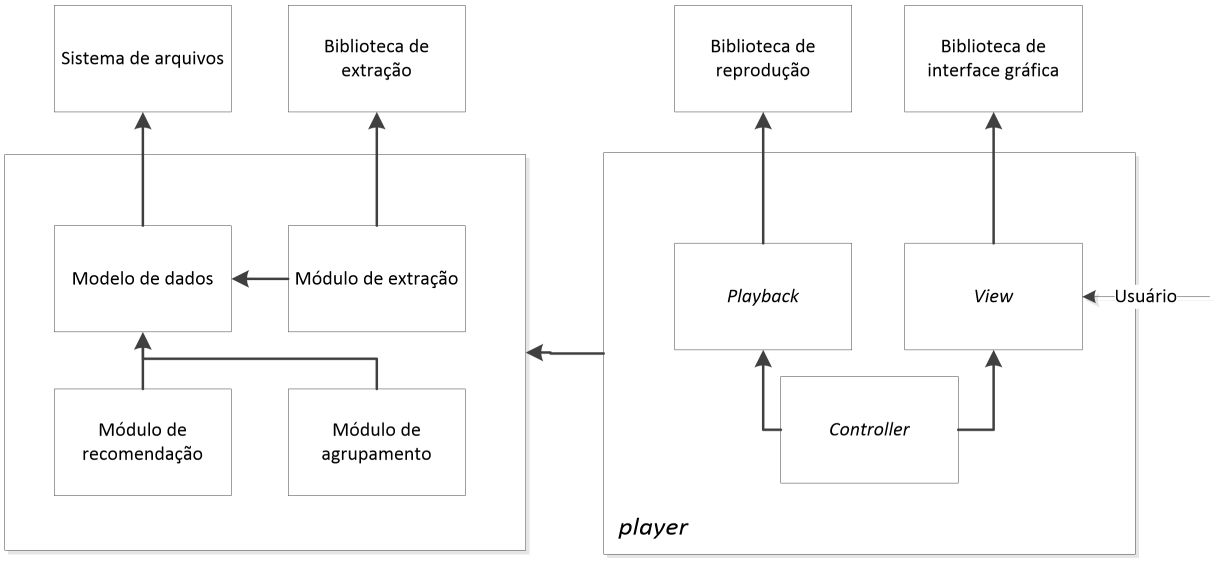
Código	UC03
Nome	Reproduzir
Atores	Usuário
Descrição	Sistema permite que o usuário reproduza, pause, pule ou repita uma música.
Requisitos relacionados	RF2 e RF7
Pré-Condições	A música deve ter sido adicionada ao sistema.
Fluxo básico de eventos	
Ações do ator	**Ações do sistema**
1. Usuário seleciona um dos comandos de reprodução na interface do sistema.	2. Sistema efetua o comando recebido.
	3. Sistema armazena o evento recebido.
4. Fim do UC	

3.5 Arquitetura

O sistema possui dois grandes componentes: o *player* e o componente de lógica da recomendação. A arquitetura do *player* foi baseada no padrão MVC (*Model-View-Controller*). O componente de lógica da recomendação foi dividido em três módulos lógicos, sendo eles o módulo de extração, o módulo de agrupamento e o módulo de recomendação. Todos os módulos dependem do modelo de dados, que representa os dados do sistema e lida com o armazenamento e carregamento desses dados no sistema de arquivos.

O diagrama abaixo representa a visão de alto-nível da arquitetura do sistema.

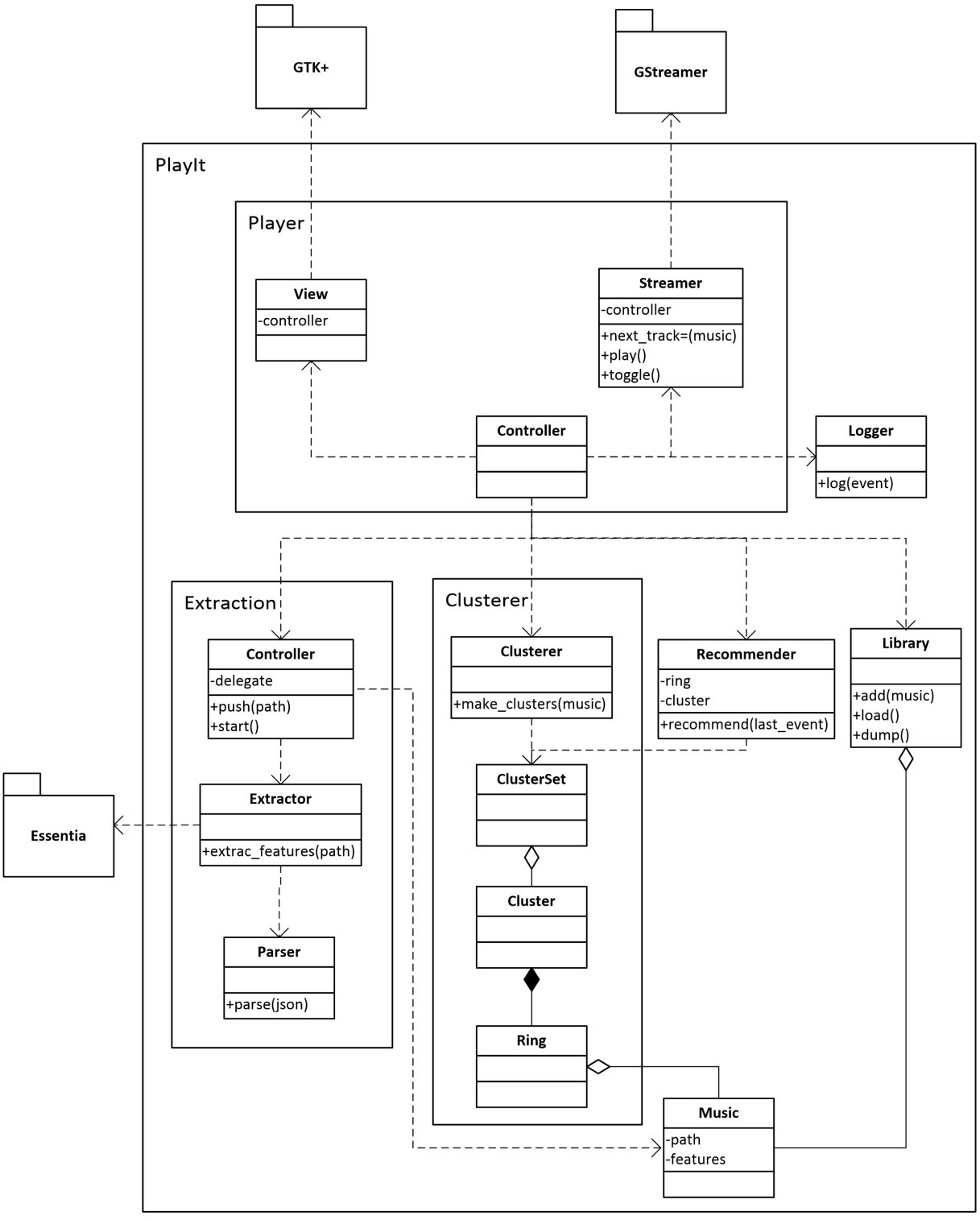
Figura 2 – Arquitetura do sistema



3.6 Diagrama de Classes

Na Figura 3 está representado o diagrama de classes do sistema, atualizado em relação à especificação original. Nele estão representados apenas os atributos e métodos mais relevantes. Também não estão representados tipos de parâmetros ou retornos.

Figura 3 – Diagrama de classes



A seguir serão descritas de maneira simplificada as responsabilidades das classes definidas no diagrama da Figura 3. As classes controladoras possuem de maneira geral a responsabilidade de coordenar as tarefas executadas pelas classes às quais se relacionam, por isso não estão descritas na lista abaixo.

View
Classe responsável pela interface gráfica do sistema;

Streamer
Classe responsável pelo *playback* de músicas;

Extractor
Classe responsável pela extração dos atributos das músicas;

ExtractionParser
Classe responsável pela adaptação da saída da extração;

Library
Classe responsável pelo armazenamento da lista atual de músicas;

Music
Classe que representa uma música no sistema;

Clusterer
Classe responsável pela geração dos agrupamentos de músicas;

ClusterSet
Classe que armazena o conjunto *clusters* gerados;

Cluster
Classe que representa um *cluster*;

Ring
Classe que representa um anel de um *cluster* (conforme definido na seção 3.7);

Recommender
Classe responsável por gerar a próxima música a ser reproduzida pelo *player* com base no algoritmo de recomendação;

Logger
Classe com a responsabilidade de armazenar eventos ocorridos.

No capítulo 5, onde detalhe-se a implementação do sistema, encontra-se identificada a relação entre cada uma dessas classes e os módulos apresentados na figura 2.

3.7 Algoritmo de recomendação

O algoritmo de recomendação desenvolvido neste trabalho baseia-se nas seguintes ideias:

- Recomendar músicas similares às que o usuário aceitou mais recentemente;
- Introduzir aleatoriedade para que as sequências geradas não sejam monótonas ou repetitivas;
- Transitar entre *clusters* de maneira gradual;
- Permitir a recomendação de músicas similares, porém um pouco mais distantes, para acompanhar transições graduais de interesse do usuário.

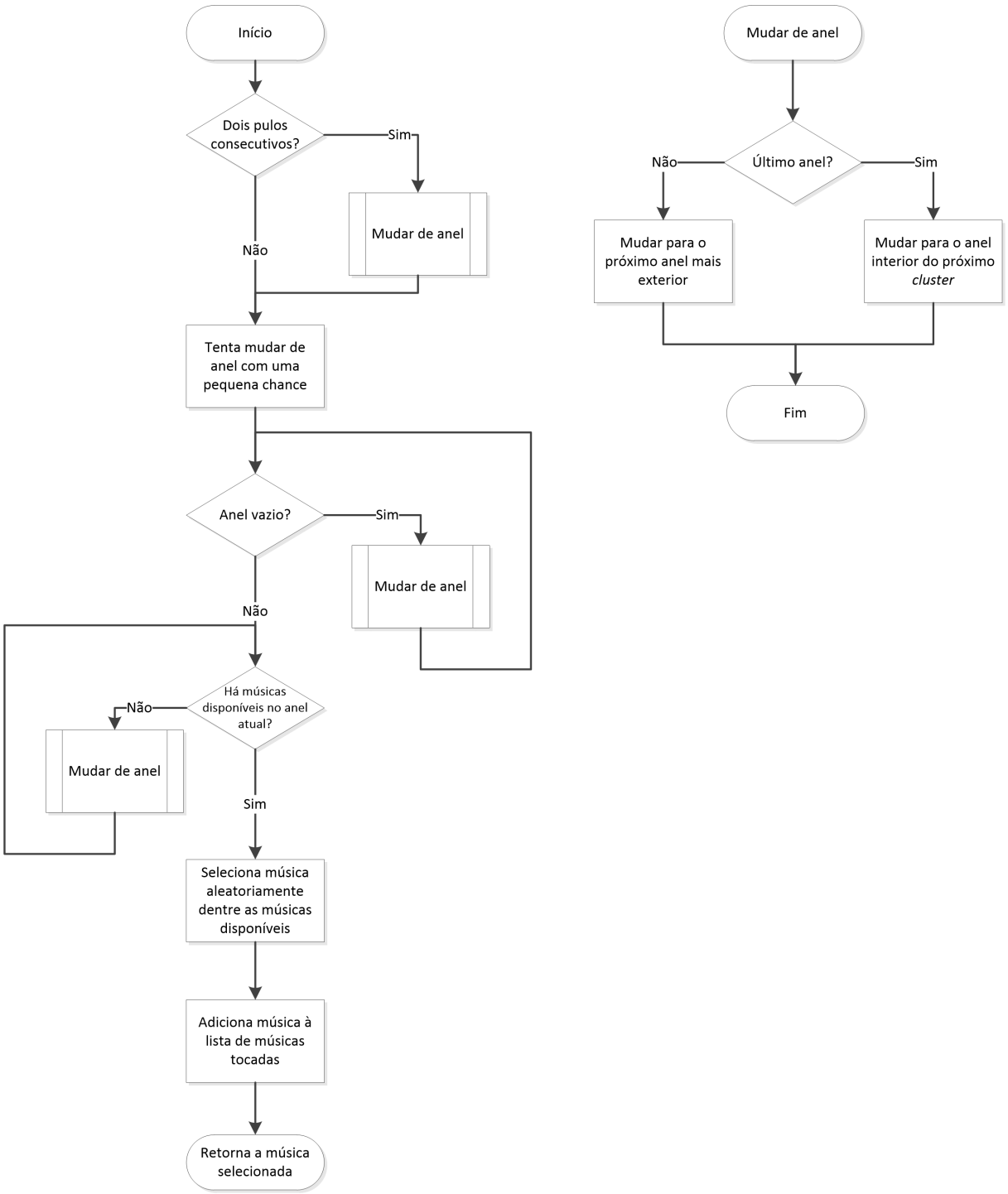
Com base nessas ideias, foi introduzido o conceito de anéis dentro dos *clusters* gerados. Esses anéis são gerados com base no raio do *cluster*, que é a maior distância entre uma música desse *cluster* e seu centroide. Todas as músicas com distância ao centroide menor que um terço do raio são alocadas no anel mais interno; as músicas com distância entre um e dois terços são alocadas no anel intermediário e as demais são alocadas no anel mais externo.

A reprodução é iniciada no anel mais interno de um *cluster* selecionado aleatoriamente. Enquanto o usuário aceita as músicas desse anel, são recomendadas aleatoriamente outras músicas dentre as músicas disponíveis, que são as músicas do anel atual que não fazem parte das últimas músicas tocadas. Para isso, o algoritmo mantém a lista de músicas tocadas. Caso o conjunto de músicas disponíveis seja vazio, troca-se de anel.

Caso o usuário rejeite duas músicas consecutivas, transita-se para o próximo anel, distanciando-se do centroide e, eventualmente, trocando de *cluster*. Foi introduzida também uma pequena chance (5%) de mudança espontânea de anel, para acompanhar transições de interesse do usuário. Essa mudança pode acontecer em ambos os casos.

O diagrama da figura 4 representa o algoritmo desenvolvido.

Figura 4 – Algoritmo de recomendação



3.8 Interface gráfica

A partir da decisão de implementar um *player* ao invés de integrar o sistema a um *player* existente como um *plugin*, como havia sido decidido anteriormente, foi feito o projeto de uma interface gráfica para o *player*. Essa interface foi projetada para ser mínima, pois um dos objetivos do projeto é que a eficiência da recomendação diminua a necessidade de interação do usuário com o *player*, permitindo que ele se concentre em suas músicas. Assim, a interface projetada possui apenas uma área de texto para que o usuário seja informado do progresso das tarefas de extração e agrupamento e os controles necessários para o funcionamento do *player*: *play/pause*, *skip*, *repeat* e um comando para adicionar músicas, representado pelo símbolo de adição.

Figura 5 – Esquema da interface gráfica

3.9 Tecnologias

Na tabela 4 estão listadas as escolhas realizadas para o desenvolvimento do projeto em relação a tecnologias e bibliotecas utilizadas. Essas escolhas foram alteradas em relação à especificação apresentada originalmente. As justificativas estão apresentadas abaixo.

Tabela 4 – Tecnologias a serem utilizadas no projeto

Item	**Escolha**
Plataforma	Linux (Elementary/Ubuntu Trusty)
Linguagem	Ruby 2.2.3
Framework de extração	Essentia
Biblioteca de agrupamento	*gem* kmeans-clusterer [1]

[1] <https://rubygems.org/gems/kmeans-clusterer>

A linguagem Ruby foi escolhida para o desenvolvimento por ser uma linguagem orientada a objetos e ser a linguagem com a qual os autores possuem maior familiaridade, possibilitando o foco nos objetivos do projeto diretamente, sem a necessidade do aprendizado ou revisão da linguagem.

A plataforma Linux foi escolhida por possuir maior facilidade e ferramentas de desenvolvimento, além de ser gratuita. A distribuição utilizada pelos autores para desenvolvimento é o Elementary OS [2], baseado no sistema Ubuntu 14.04.

O algoritmo de agrupamento utilizado será inicialmente o k-médias, por ser um dos algoritmos de agrupamento mais tradicionais e ser amplamente utilizado. A implementação utilizada será a fornecida pela *gem* (biblioteca da linguagem Ruby) kmeans-clusterer, devido à simplicidade de utilização e bom desempenho.

A biblioteca de extração selecionada foi a Essentia, devido ao suporte nativo ao formato mp3.

[2] <http://elementary.io>

4 Metodologia

O desenvolvimento deste trabalho foi divido em dois semestres. Durante o primeiro semestre, foi priorizado o trabalho de pesquisa bibliográfica e concepção do projeto. No segundo semestre, foi realizada a implementação, seguindo uma abordagem iterativa e incremental.

A seguir são apresentadas as etapas nas quais foi dividido o desenvolvimento deste trabalho e como foram desenvolvidas cada uma delas.

Etapa 1: Pesquisa bibliográfica

A primeira parte do trabalho consistiu na pesquisa bibliográfica. Foram consultadas diversas fontes acadêmicas, especialmente artigos publicados na área e trabalhos apresentados em conferências, como por exemplo as realizadas pela ISMIR [1].

Etapa 2: Análise

Com o objetivo de delinear a funcionalidade do sistema, foi realizada uma etapa de análise para levantamento de requisitos funcionais e casos de uso.

As informações levantadas nesta etapa serviram como base para o desenvolvimento do projeto, porém foram aprimoradas durante a etapa de implementação.

Etapa 3: *Design*

Nesta etapa, foi determinada a plataforma de implantação do sistema e foi realizada a modelagem do sistema, através da definição da arquitetura e da estrutura interna dos componentes.

Da mesma forma que na etapa de análise, a modelagem realizada nesta etapa foi aprimorada durante a implementação.

Etapa 4: Implementação

Na etapa de implementação, foram desenvolvidos e testados cada um dos componentes definidos para o sistema de forma iterativa e incremental. Cada funcionalidade do sistema foi implementada através de ciclos de modelagem, implementação e testes, até que o comportamento desejado fosse atingido. Foram utilizados testes unitários e o *framework* de testes automatizados RSpec[2].

O desenvolvimento dos componentes foi iniciado pelo modelo de dados. Na etapa posterior, os módulos de extração e agrupamento foram implementados paralelamente

[1] *International Society for Music Information Retrieval* (<http://www.ismir.net/>)
[2] <http://rspec.info/>

ao *player*, com seus três componentes (*playback*, *controller* e *view*). Em seguida, o *player* foi integrado a esses módulos paralelamente ao desenvolvimento do algoritmo de recomendação e, finalmente, foi integrado a ele.

Ao término desta etapa, foram realizados testes de integração.

Etapa 5: Testes e avaliação

O sistema foi testado a partir de uma avaliação do número de pulos resultantes da simulação de alguns casos de uso, de maneira similar aos testes realizados em (PAMPALK; POHLE; WIDMER, 2005). Os resultados também foram comparados aos obtidos neste trabalho.

Etapa 6: Documentação

Os resultados obtidos foram avaliados e registrados na documentação final do projeto.

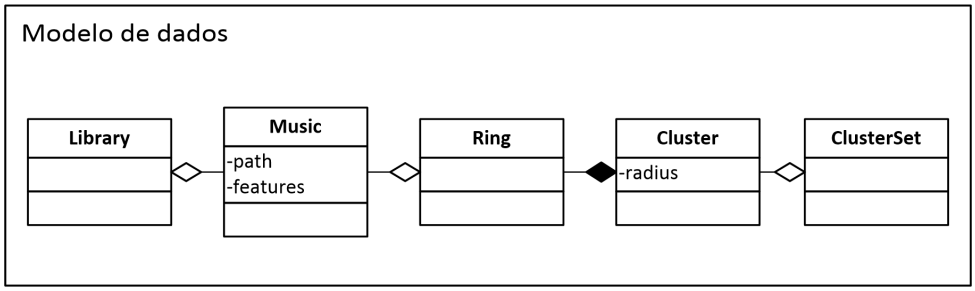
5 Implementação

Neste capítulo, serão apresentados detalhes da implementação de cada módulo, bem como os desafios encontrados e soluções utilizadas. A seção 5.2 descreve o módulo de extração; a seção 5.3 descreve o módulo de agrupamento, incluindo as modificações introduzidas na estrutura de dados dos *clusters*; a seção 5.4 descreve a implementação do *player*, incluindo *playback*, interface gráfica e integração com o restante do sistema; por fim, a seção 5.5 descreve a implementação da recomendação.

Cada uma das seções está ilustrada por um diagrama de classes esquemático do componente descrito. No entanto, esses diagramas não são completos. O diagrama completo encontra-se na figura 3. O código fonte deste projeto pode ser encontrado no seu repositório em <https://github.com/play-it/play-it>.

5.1 Modelo de dados

Figura 6 – Modelo de dados

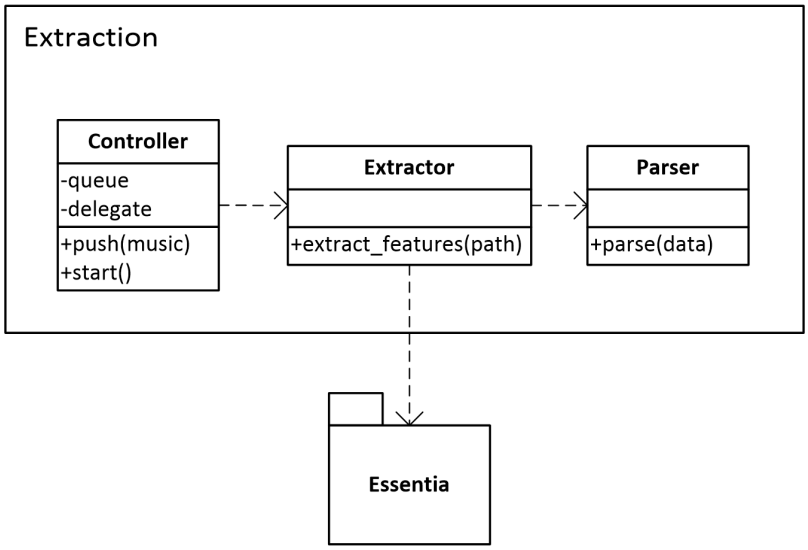


O modelo de dados implementado é constituído das classes que representam os dados do sistema, relacionados a músicas e *clusters*. As classes `Library` e `ClusterSet` implementam métodos para armazenamento e carregamento de seus dados a partir do sistema de arquivos.

5.2 Extração

Como mencionado na seção 3.9, foi utilizada a biblioteca Essentia para extração das características de áudio. Para isso, foi utilizado um arquivo binário precompilado para a plataforma escolhida, que é invocado pela classe `Extractor` através de chamadas do sistema. As características a serem extraídas são configuradas através de um arquivo de perfil, e a saída é gravada em formato JSON num arquivo temporário; os dados deste arquivo são então transformado num objeto em formato adequado pela classe `Parser`.

Figura 7 – Módulo de extração

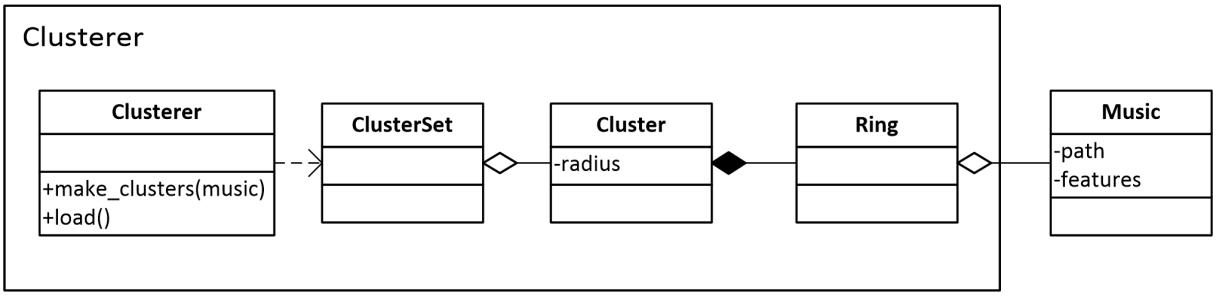


Devido a problemas de desempenho da extração, esse processo foi transferido para o segundo plano através do uso de uma *thread* secundária e uma fila de extração. Estas são controlados pelo `Controller`. Isso permite que novas músicas sejam processadas enquanto o *player* mantém seu funcionamento.

Devido a esse funcionamento assíncrono, foi utilizado o padrão de *design* de delegação para que o `Controller` informe ao solicitante o progresso da extração. O protocolo definido consiste em dois métodos: um invocado quando há alteração no tamanho da fila e outro invocado quando a extração é concluída.

5.3 Agrupamento

Figura 8 – Módulo de agrupamento



Como descrito na seção 3.9, o algoritmo de agrupamento utilizado foi o k-médias, implementado pela *gem* kmeans-clusterer.

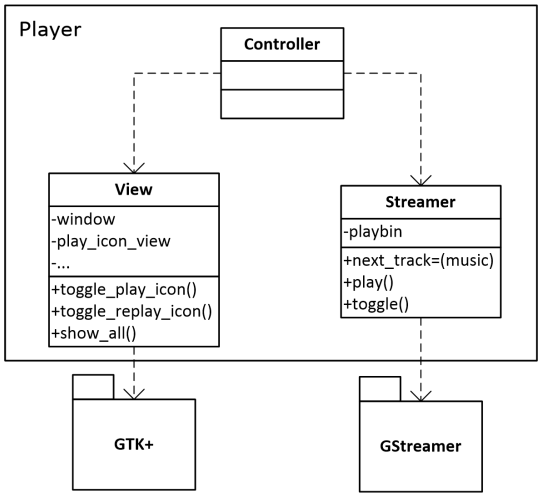
Para a geração dos *clusters*, é utilizado o método de avaliação da soma do erro quadrático, a fim de determinar o melhor conjunto de *clusters* gerados. Logo, o agrupamento

é executado com diferentes valores de k, em um intervalo de dois até a raiz do número de músicas, e selecionado o conjunto com menor soma do erro quadrático. Para garantir a convergência, para cada valor de k, o algoritmo de agrupamento é executado um número de vezes calculado a partir do número de músicas a serem agrupadas.

Para dar suporte ao algoritmo de recomendação, foi introduzida a estrutura de anéis nos *clusters* gerados. Esses anéis são regiões dos *clusters* ao redor de seu centroide, determinadas com base no raio do *cluster*.

5.4 *Player*

Figura 9 – Módulo *player*



O *player* possui três componentes: *controller*, responsável pelo controle e integração dos demais componentes, bem como a comunicação com o restante do sistema, *view*, responsável pela interface gráfica, e *playback*, responsável pela reprodução de áudio.

A *view* foi implementada através da biblioteca GTK+[1], utilizando *builders* para definir o *layout* através de XML e também utilizando folhas de estilo (CSS) para alguns efeitos da interface. Os ícones foram obtidos gratuitamente da coleção Material Design do Google, disponível no site flaticon[2].

O *playback* foi implementado através da biblioteca GStreamer[3], utilizando especificamente o *plugin playbin*[4]. Isso foi encapsulado na classe `Streamer`.

O controle foi realizado com base no padrão de delegação, como mencionado na seção 5.2. `View` e `Streamer` possuem uma referência para o `Controller` e enviam

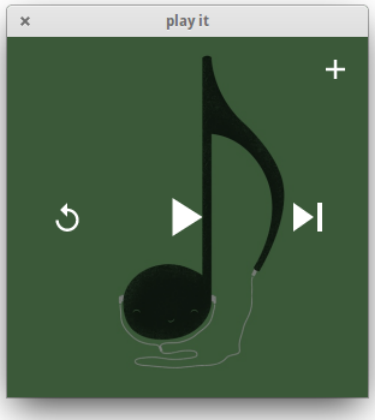
[1] <http://www.gtk.org/>
[2] <http://www.flaticon.com/packs/material-design>
[3] <http://gstreamer.freedesktop.org/>
[4] <http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-base-plugins/html/gst-plugins-base-plugins-playbin.html>

mensagens a ele quando capturam eventos através de um protocolo predefinido. O trecho de código abaixo ilustra os métodos delegados da `View` para o `Controller`.

```
1 def_delegators :@controller,
2                :on_play_icon_view_item_activated,
3                :on_replay_icon_view_item_activated,
4                :on_skip_icon_view_item_activated
```

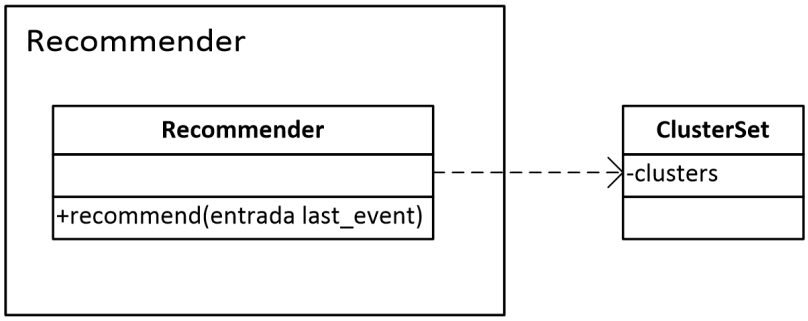
A figura 10 apresenta uma captura de tela da interface do *player*.

Figura 10 – Captura de tela da interface do *player*



5.5 Recomendação

Figura 11 – Classe `Recommender`



A recomendação foi implementada através da classe `Recommender`, que possui como atributo o `ClusterSet` de todas as músicas indexadas e recebe um evento para recomendar a próxima música. Essa classe possui também diversos outros atributos para manter o seu estado, como o índice do *cluster* e do anel em que se encontra, e uma lista de músicas já tocadas para evitar recomendações muito frequentes da mesma música.

A *gem* probability[5] foi utilizada para implementar as chances de troca espontânea de anéis ou *clusters*.

[5] <http://ariejan.github.io/probability/>

6 Testes e avaliação

Neste capítulo, são apresentados os testes realizados e seus resultados. A seção 6.1 descreve os testes realizados; a seção 6.2 apresenta os resultados obtidos e a seção 6.3 apresenta a análise dos resultados.

6.1 Descrição

Para realização dos testes, foi utilizado o *dataset* de trechos de músicas disponibilizado pela Universidade de Dortmund[1]. O *dataset* contém 1886 trechos de músicas de diferentes gêneros em formato mp3. A tabela 5 apresenta a quantidade de trechos para cada gênero.

Tabela 5 – Quantidade de trechos de cada gênero no *dataset* utilizado

Gênero	**Quantidade de músicas**	%
alternative	145	7,7
blues	120	6,4
electronic	113	6,0
folkcountry	222	11,8
funksoulrnb	47	2,5
jazz	319	16,9
pop	116	6,1
raphiphop	300	15,9
rock	504	26,7
total	1886	100,0

O processo de extração de características e agrupamento das músicas desse *dataset* resultou em 43 *clusters*.

Foram realizados dois tipos de testes. O primeiro tipo, denominado objetivo, consistiu em automaticamente aceitar ou pular uma música sugerida pelo sistema com base em seu gênero. Foram realizados dois testes desse tipo a partir da simulação de casos de uso, como realizado em (PAMPALK; POHLE; WIDMER, 2005).

Para a simulação dos casos de uso, foi assumido que o usuário gostaria de escutar uma hora de música, o que são aproximadamente 20 músicas aceitas. Os casos simulados foram:

1. São aceitas apenas músicas do mesmo gênero;

[1] <http://www-ai.cs.uni-dortmund.de/audio.html>

2. São aceitas inicialmente músicas de um gênero A, em seguida músicas tanto do gênero A quanto de um gênero B similar e, por fim, são aceitas apenas músicas do gênero B (transição de gêneros).

Como base de comparação, a heurística A apresentada em (PAMPALK; POHLE; WIDMER, 2005) foi implementada. Um pequeno módulo de seleção aleatória (*shuffle*) também foi implementado para comparação de resultados.

O segundo tipo de testes, denominado de subjetivo, envolveu a participação de dois usuários. Eles foram apresentados a uma música inicial e foram instruídos a aceitar ou pular as músicas subsequentes sugeridas pelo sistema com base na similaridade percebida por eles entre a música sugerida e a música inicial. Foram realizados quatro testes, com músicas iniciais de diferentes gêneros.

6.2 Resultados

Para o primeiro teste objetivo, era esperado que a taxa de aceitação do *shuffle* fosse aproximadamente igual à proporção de músicas do gênero sendo testado, como mostrado na tabela 5. As taxas médias de aceitação para cada um dos gêneros e algoritmos comparados neste caso de uso estão apresentadas na tabela 6.

Tabela 6 – Taxas de aceitação por gênero para o primeiro teste objetivo (%)

Gênero	**Algoritmo desenvolvido**	**Heurística de referência**	***Shuffle***
alternative	6,8	41,7	7,4
blues	5,8	10,3	6,6
electronic	7,8	11,2	5,3
folkcountry	12,1	10,1	11,4
funksoulrnb	2,1	23,3	2,6
jazz	27,0	52,6	20,2
pop	5,4	11,5	7,4
raphiphop	9,0	37,0	15,8
rock	31,8	28,3	25,9
todos	6,9	16,8	7,3

Para o segundo teste objetivo, os gêneros de origem e destino (A e B) foram escolhidos manualmente. A tabela 7 mostra os resultados.

Para os testes subjetivos, apenas o algoritmo desenvolvido neste trabalho foi testado. As taxas médias de aceitação para cada um dos gêneros testados estão apresentadas na tabela 8.

Tabela 7 – Taxas de aceitação por gênero para o segundo teste objetivo (%)

De	**Para**	**Algoritmo desenvolvido**	**Heurística de referência**	***Shuffle***
jazz	blues	14,4	12,9	12,6
alternative	rock	19,5	16,0	18,8
rock	raphiphop	30,6	16,4	29,7
blues	pop	7,0	20,1	8,2
funksoulrnb	electronic	4,7	12,6	4,8
todos		9,8	11,5	10,1

Tabela 8 – Taxas de aceitação por gênero para os testes subjetivos

Gênero	**Taxa de aceitação (%)**
rock	92,3
raphiphop	42,9
electronic	48,0
funksoulrnb	60,0
todos	55,8

6.3 Análise dos resultados

Pode-se observar, no primeiro teste objetivo, que a taxa de aceitação do algoritmo proposto foi inferior em relação à heurística de referência na maioria dos gêneros. Pode-se notar uma taxa de aceitação muito mais baixa em gêneros com os menores números de músicas, como observado para o gênero *funksoulrnb* (2,5% do total de músicas), em que obtivemos uma taxa de aceitação de apenas 2,1%, sendo muito inferior aos 23,3% da heurística de referência, e comparável aos resultados do *shuffle*, que obteve 2,6% de aceitação. Em comparação, o gênero *rock* (26,6% do total de músicas) obteve melhores resultados que as outras duas referências.

Já no segundo teste objetivo, o algoritmo proposto obteve resultados similares aos outros algoritmos, mas ainda com taxas de aceitação baixas quando são envolvidos gêneros com quantidades baixas de músicas, como é visto na transição de *blues* para *pop*, e de *funksoulrnb* para *electronic*.

Em contrapartida, os testes subjetivos obtiveram resultados satisfatórios, mesmo quando o teste foi iniciado com um gênero de poucas músicas, como mostra a taxa de aceitação de 60% quando iniciado o teste com uma música do gênero *funksoulrnb*, que representa apenas 2,5% do total de músicas.

7 Conclusões

Como demonstrado pelos testes realizados, o desempenho do algoritmo de recomendação não foi satisfatório nos testes objetivos. Isso pode mostrar uma possível deficiência na heurística proposta, dados os resultados comparáveis à seleção aleatória, ou ainda a má escolha de características para extração.

Por outro lado, o algoritmo de recomendação mostrou-se satisfatório a partir dos testes subjetivos realizados, o que pode indicar que os gêneros não são os melhores categorizadores para as músicas, ou que a estratégia de agrupamento não foi bem sucedida. Contudo, uma quantidade maior de testes precisa ser efetuada, aplicada a um grupo maior de usuários.

Foi observado durante os testes que a heurística proposta não leva em consideração as músicas aceitas pelo usuário. Isso faz com que, ao mudar de *cluster*, o algoritmo demore a voltar para um *cluster* semelhante, causando muitas rejeições. Para melhoria do algoritmo, pode ser implementado um mecanismo mais sofisticado de troca de *cluster*, que leve em consideração as músicas já aceitas pelo usuário.

Outra continuação possível para o projeto é a implementação da estratégia de adaptação dos *clusters*, como planejado originalmente, a partir dos dados de uso. Isso permitiria que a percepção de similaridade do usuário influenciasse diretamente nos *clusters* gerados e, consequentemente, na recomendação de músicas.

Referências

BOGDANOV, D. et al. ESSENTIA: an audio analysis library for music information retrieval. In: *Proceedings of International Society for Music Information Retrieval Conference*. Curitiba, Brasil: [s.n.], 2013. p. 493–498. Citado na página 16.

IFPI. *Digital Music Report 2015*. [S.l.], 2015. Citado na página 13.

KING, J. V. *Generating Music Playlists with Reinforcement Learning*. Dissertação (Mestrado) — University of Cambridge, Cambridge, United Kingdom, 2014. Citado 4 vezes nas páginas 13, 16, 18 e 20.

KOHONEN, T. The self-organizing map. *Neurocomputing*, Elsevier, v. 21, n. 1, p. 1–6, 1998. Citado na página 16.

LARTILLOT, O.; TOIVIAINEN, P.; EEROLA, T. A Matlab toolbox for music information retrieval. In: PREISACH, C. et al. (Ed.). *Data Analysis, Machine Learning and Applications*. Springer Berlin Heidelberg, 2008, (Studies in Classification, Data Analysis, and Knowledge Organization). p. 261–268. ISBN 978-3-540-78239-1. Disponível em: <http://dx.doi.org/10.1007/978-3-540-78246-9_31>. Citado na página 17.

LEE, J. H.; WATERMAN, N. M. Understanding user requirements for music information services. In: *Proceedings of the 13th International Society for Music Information Retrieval Conference, ISMIR 2012, Mosteiro S.Bento Da Vitória, Porto, Portugal, October 8-12, 2012*. [s.n.], 2012. p. 253–258. Disponível em: <http://ismir2012.ismir.net/event/papers/253-ismir-2012.pdf>. Citado na página 13.

LIU, D.; LU, L.; ZHANG, H. Automatic mood detection from acoustic music data. In: *Proceedings of ISMIR 2003, 4th International Conference on Music Information Retrieval, Baltimore, Maryland, USA, October 27-30, 2003*. [s.n.], 2003. Disponível em: <http://ismir2003.ismir.net/papers/Liu.PDF>. Citado 2 vezes nas páginas 16 e 18.

LÜBBERS, D.; JARKE, M. Adaptive multimodal exploration of music collections. In: *Proceedings of the 10th International Society for Music Information Retrieval Conference, ISMIR 2009, Kobe International Conference Center, Kobe, Japan, October 26-30, 2009*. [s.n.], 2009. p. 195–200. Disponível em: <http://ismir2009.ismir.net/proceedings/PS2-1.pdf>. Citado na página 18.

PAMPALK, E. *Computational Models of Music Similarity and their Application in Music Information Retrieval*. Tese (Doutorado) — Universidade Técnica de Viena, 2006. Citado 2 vezes nas páginas 16 e 18.

PAMPALK, E.; FLEXER, A.; WIDMER, G. Improvements of audio-based music similarity and genre classificaton. In: *Proceedings of ISMIR 2005, 6th International Conference on Music Information Retrieval, London, UK, 11-15 September 2005*. [s.n.], 2005. p. 628–633. Disponível em: <http://ismir2005.ismir.net/proceedings/1053.pdf>. Citado na página 16.

PAMPALK, E.; GASSER, M.; TOMITSCH, M. A content-based user-feedback driven playlist generator and its evaluation in a real-world scenario. In: *Proceedings of Audio Mostly Conference*. [S.l.: s.n.], 2007. Citado 4 vezes nas páginas 13, 16, 18 e 19.

PAMPALK, E.; POHLE, T.; WIDMER, G. Dynamic playlist generation based on skipping behavior. In: *Proceedings of ISMIR 2005, 6th International Conference on Music Information Retrieval, London, UK, 11-15 September 2005*. [s.n.], 2005. p. 634–637. Disponível em: <http://ismir2005.ismir.net/proceedings/2072.pdf>. Citado 8 vezes nas páginas 13, 16, 18, 19, 20, 36, 43 e 44.

POHLE, T.; PAMPALK, E.; WIDMER, G. Generating similarity-based playlists using traveling salesman algorithms. In: *Proceedings of the 8th International Conference on Digital Audio Effects (DAFx-05)*. [S.l.: s.n.], 2005. p. 220–225. Citado 2 vezes nas páginas 16 e 18.

RUSSEL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. 3. ed. [S.l.]: Pearson, 2013. 529 p. Citado na página 24.

SCHEDL, M. et al. User-aware music retrieval. In: *Multimodal Music Processing*. [s.n.], 2012. p. 135–156. Disponível em: <http://dx.doi.org/10.4230/DFU.Vol3.11041.135>. Citado 2 vezes nas páginas 14 e 15.

STOBER, S.; NÜRNBERGER, A. Towards user-adaptive structuring and organization of music collections. In: *Proceedings of Adaptive Multimedia Retrieval: Identifying, Summarizing, and Recommending Image and Music, 6th International Workshop, AMR 2008, Berlin, Germany, June 26-27, 2008*. [s.n.], 2008. p. 53–65. Disponível em: <http://dx.doi.org/10.1007/978-3-642-14758-6_5>. Citado na página 18.

TZANETAKIS, G. Music Information Retrieval. Supporting material for the Music Information Retrieval course offered at the University of Victoria. 2015. Citado 2 vezes nas páginas 13 e 17.

TZANETAKIS, G.; COOK, P. MARSYAS: a framework for audio analysis. *Organised sound*, Cambridge University Press, v. 4, n. 03, p. 169–175, 2000. Citado na página 17.

WATKINS, C.; DAYAN, P. Q-learning. *Machine Learning*, Kluwer Academic Publishers, v. 8, n. 3-4, p. 279–292, 1992. ISSN 0885-6125. Disponível em: <http://dx.doi.org/10.1007/BF00992698>. Citado na página 20.

WATSON, D.; MANDRYK, R. L. Modeling musical mood from audio features and listening context on an in-situ data set. In: *Proceedings of the 13th International Society for Music Information Retrieval Conference, ISMIR 2012, Mosteiro S.Bento Da Vitória, Porto, Portugal, October 8-12, 2012*. [s.n.], 2012. p. 31–36. Disponível em: <http://ismir2012.ismir.net/event/papers/031-ismir-2012.pdf>. Citado na página 18.