UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

FERRAMENTA PARA CRIAÇÃO DE COMPOSIÇÕES MUSICAIS PARA ANDROID

GUSTAVO GARCIA ALVARENGA

BLUMENAU
2013

2013/1-16

GUSTAVO GARCIA ALVARENGA

FERRAMENTA PARA CRIAÇÃO DE COMPOSIÇÕES MUSICAIS PARA ANDROID

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Aurélio Faustino Hoppe , Mestre - Orientador

BLUMENAU
2013

2013/1-16

FERRAMENTA PARA CRIAÇÃO DE COMPOSIÇÕES MUSICAIS PARA ANDROID

Por

GUSTAVO GARCIA ALVARENGA

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

	_____
Presidente:	Prof. Aurélio Faustino Hoppe, Mestre – Orientador, FURB
	_____
Membro:	Prof. Dalton Solano dos Reis, Mestre – FURB
	_____
Membro:	Prof. Roberto Heinzle, Doutor – FURB

Blumenau, 10 de julho de 2013

AGRADECIMENTOS

A minha família e amigos, pelo apoio (e insistência) para a realização deste trabalho.

Ao meu orientador, Aurélio Faustino Hoppe, por seu apoio, compreensão e paciência.

À música, por ter se tornado uma constante em minha vida.

Não há conhecimento que não seja poder.

Ralph Waldo Emerson

RESUMO

Este trabalho apresenta um aplicativo para a criação de composições musicais em dispositivos Android. O aplicativo desenvolvido visa fornecer a usuários leigos em composição musical os recursos necessários para criarem composições simples. Foi utilizado o componente de código aberto *HorizontalVariableListView* para a interface gráfica e testados os componentes *MediaPlayer* e *SoundPool* para a execução do áudio. A partir dos testes de usabilidade e experimentos realizados, concluiu-se que o sistema é suficientemente intuitivo e fácil de usar para usuários inexperientes.

Palavras-chave: Composição musical. Android.

ABSTRACT

This work presents an application for the creation of musical compositions on Android devices. The developed application seeks to provide users with no musical composition experience the necessary resources to create simple compositions. The open code component HorizontalVariableListView was used for the graphic interface and the components MediaPlayer and SoundPool were experimented for audio execution. From the usability tests and experiments performed, it was concluded that the application is sufficiently intuitive and easy to use for inexperienced users.

Keywords: Musical composition. Android.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

ADT - *Android Developer Tools*

GUI - *Graphic User Interface*

IHC - Interface Humano-Computador

JSON - *JavaScript Object Notation*

MIDI - *Musical Instrument Digital Interface*

MVC - *Model-View-Controller*

SDK - *Software Development Kit*

UML - *Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO

A música é a forma de expressão mais antiga da raça humana, tendo-se a voz como primeiro instrumento musical. Com ela, o homem aprendeu a produzir os mais diversos sons, e a agrupar estes sons, formando as primeiras linhas melódicas. Quase que simultaneamente, criaram-se os instrumentos musicais, que se multiplicaram e evoluíram ao longo da história (ANDRADE, 1977, p. 97). Muitos desses desapareceram com a evolução tecnológica. A música mudou muito em todo este tempo, mas o gosto do ser humano por ela permanece intacto.

Criaram-se vários instrumentos musicais para obter determinados tipos de sons, sempre visando a qualidade, o timbre e o modo de execução do mesmo. Para criação de diferentes tipos de músicas, ou músicas mais elaboradas, são necessários instrumentos musicais distintos, onde cada um destes instrumentos produz um determinado tipo de som (ANDRADE, 1977, p. 97-98).

Com a evolução da tecnologia da informação, começaram a surgir os primeiros instrumentos musicais eletrônicos, primeiramente instrumentos que produziam sons simples através da manipulação de ondas sonoras. Foram surgindo também instrumentos capazes de simular outros instrumentos, conhecidos como sintetizadores (GRIFFITHS, 1998, p. 145-150).

Juntamente com estes novos instrumentos eletrônicos, esta evolução da tecnologia da informação trouxe também a possibilidade da gravação e posterior reprodução de composições sonoras. Os avanços na computação não só permitiam a manipulação das gravações de áudio, como também traziam a possibilidade de auxiliar na composição musical (GRIFFITHS, 1998, p. 155-156).

Com o advento e popularização dos dispositivos móveis, a computação musical também passou a ser móvel. Aplicativos desenvolvidos para plataformas estacionárias foram sendo portados para dispositivos móveis e outros aplicativos surgiram tendo os dispositivos móveis como seus ambientes nativos.

Diante do exposto, desenvolveu-se um aplicativo para dispositivos móveis com sistema operacional Android capaz de criar diversas músicas em um único aplicativo. Com o desenvolvimento deste aplicativo pretende-se despertar os interesses de usuários leigos ao mundo da música e garantir que seus recursos sejam de fácil entendimento, tornando-a auto-suficiente para o aprendizado.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver um aplicativo que permita ao usuário criar uma música utilizando uma representação musical simplificada. Esta representação musical será apresentada ao usuário na forma de peças representando *samples*[1] que serão encaixadas em um *grid*.

Os objetivos específicos do trabalho são:

a) criar uma representação sonora compreensível para usuários sem domínio técnico em composição musical, permitindo que estes usuários possam criar suas próprias músicas;
b) disponibilizar ao usuário um banco de *samples* de variados instrumentos, para que o dispositivo execute a música de acordo com a representação definida pelo usuário.

1.2 ESTRUTURA DO TRABALHO

Este trabalho é apresentado em quatro capítulos, sendo neste primeiro capítulo apresentada uma breve introdução ao tema, os objetivos do trabalho e a estrutura o qual é apresentado.

O segundo capítulo contém a fundamentação teórica necessária para a compreensão do assunto e da solução implementada no aplicativo.

O terceiro capítulo apresenta todo o processo de especificação do aplicativo, onde são apresentados diagramas de caso de uso, de pacotes e de classes. Também são expostos detalhes da implementação além da operacionalidade do aplicativo do ponto de vista do usuário.

O último capítulo, por sua vez, apresenta as conclusões a respeito dos resultados obtidos além de sugestões para trabalhos futuros.

[1] Trechos de música curtos e simples, com duração de poucos segundos e apenas um instrumento.

2 FUNDAMENTAÇÃO TEÓRICA

A seção 2.1 descreve características para o desenvolvimento de aplicações de computação musical. A seção 2.2 explica sobre o desenvolvimento de aplicativos destinados à criação de música em tempo real para dispositivos móveis. Na seção 2.3 são descritas as principais características da plataforma Android. A seção 2.4 relaciona algumas técnicas de Interface Humano-Computador (IHC) para desenvolvimento de aplicações para dispositivos móveis. Por fim, na seção 2.5, são apresentados três trabalhos correlatos.

2.1 COMPUTAÇÃO MUSICAL

A computação musical é uma área da ciência da computação dedicada ao estudo das aplicações dos computadores a problemas musicais (MILETTO et al., 2004, p. 3). A computação musical investiga métodos, técnicas e algoritmos para processamento e geração de dados relacionados à música. Miletto et al. (2004, p. 3) afirma ainda que a computação musical visa resolver problemas relevantes como, por exemplo, controle a tempo real de dispositivos, interação com usuários possivelmente leigos em informática e representação da informação sonora.

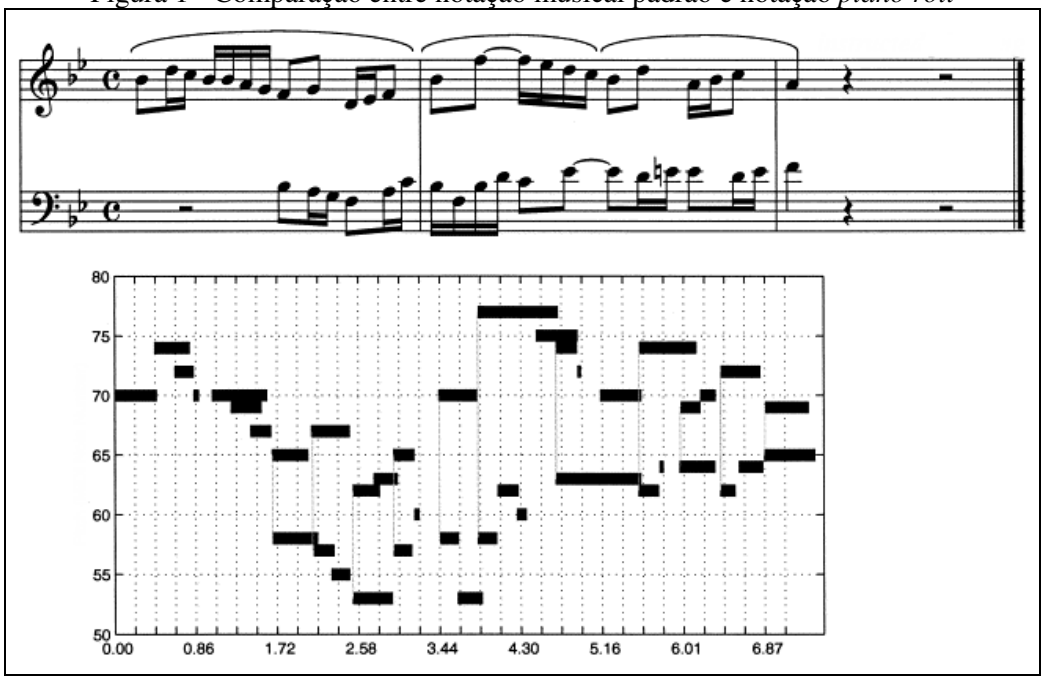
2.1.1 Representação de informação sonora: áudio digital

Pode-se representar o som usando um gráfico que mostre as variações que ocorrem na pressão do ar no decorrer do tempo. O mesmo gráfico serve também para representar as variações da tensão elétrica no tempo, pois estes sinais mantêm as mesmas características oscilatórias (MILETTO et al., 2004, p. 4). Com estas oscilações é possível fazer a digitalização do som.

Ao processo de representar numericamente o som dá-se o nome de digitalização de som. Somente com a digitalização é possível trazer a música para o ambiente computacional, e, assim, reproduzi-la ou alterá-la. O áudio digitalizado gera um grande volume de dados, porém contém apenas a informação sonora. Para que se possa manipular os sons no nível de elementos musicais, para alterar parâmetros da música ou gerar uma música nota por nota, é necessário a informação musical (MILLETO et al., 2004, p. 892).

Para se manipular uma informação musical é comumente utilizado um protocolo conhecido como *Musical Instrumente Digital Interface* (MIDI), representado normalmente em notação de *piano roll* (Figura 1). Em seu código há instruções como soar uma nota, silenciar uma nota, mudar o andamento ou o tom da música (HASS, 2010).

Figura 1 - Comparação entre notação musical padrão e notação *piano roll*



Fonte: Large e Palmer (2002, p. 17).

Na Figura 1 é feita uma comparação entre a notação musical padrão e a notação *piano roll*. Na notação musical padrão cada figura musical representa uma nota, onde sua duração é representada pela própria figura, sua altura é representada por sua posição na pauta em referência à nota definida pela clave utilizada e seu tempo é relativo à nota anterior. Na notação *piano roll* cada barra representa uma nota, onde sua duração é representada pelo comprimento da barra, sua altura é representada por sua posição vertical e seu tempo é representado por sua posição horizontal (HASS, 2010).

2.2 PRINCÍPIOS DE APLICATIVOS PARA DISPOSITIVOS MÓVEIS

Segundo B'far (2005, p. 3), sistemas para dispositivos móveis são aplicativos computacionais que podem ser facilmente deslocados fisicamente e cujas capacidades de computação podem ser usadas enquanto se deslocam. Recentemente, tem se difundido o uso de dispositivos móveis com sistemas operacionais especializados, como o iOS e o Android. Estes sistemas possuem características diferentes dos aplicativos que são desenvolvidos para dispositivos estacionários. Pode-se identificar as diferenças nas tarefas que se destinam a executar, a forma como são concebidos e a maneira em que são explorados (B'FAR, 2005, p. 3).

Além das características dos aplicativos para dispositivos móveis, existe a preocupação com as condições dos usuários que utilizam estes sistemas. Os usuários de dispositivos

móveis são essencialmente diferentes dos usuários de dispositivos estacionários (B'FAR, 2005, p. 22). Eles podem ser diferentes nos seguintes aspectos:

a) o usuário móvel está em movimento, pelo menos ocasionalmente, entre conhecidas ou desconhecidas localidades;
b) os usuários móveis normalmente não estão focados na tarefa de computação;
c) o usuário móvel frequentemente requer um alto grau de rapidez e capacidade de resposta do sistema;
d) o usuário móvel está mudando as tarefas com freqüência e/ou abruptamente;
e) o usuário móvel pode exigir o acesso ao sistema em qualquer lugar e a qualquer hora.

Estes aspectos são primordiais para o desenvolvimento de aplicações para dispositivos móveis. Elas tratam não apenas as condições físicas dos usuários móveis, mas também o estado mental do mesmo, ou seja, as suas expectativas e o estado de espírito (B'FAR, 2005, p. 22).

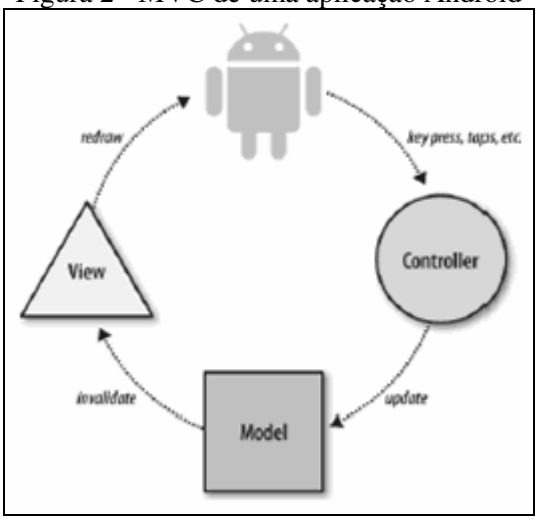
2.3 PLATAFORMA ANDROID

Trata-se de um sistema operacional criado para aparelhos móveis (ANDROID DEVELOPERS, 2012). A plataforma Android compreende não somente o sistema operacional, mas também um conjunto de ferramentas para desenvolvimento de aplicações para os *smartphones* e *tablets*. Trata-se do *Software Development Kit* (SDK) do Android.

O SDK dispõe de vários recursos para desenvolvimento da *Graphic User Interface* (GUI), depuração, testes, simulações e instalação destas aplicações, tendo como recomendação o ambiente de desenvolvimento Eclipse (ANDROID DEVELOPERS, 2012). Neste conjunto de recursos há, ainda, uma série de aplicações como: *e-mail*, programa de *Short Message Service* (SMS), calendário, mapas, *browser*, contatos e outros. Todos estes recursos do SDK fazem parte da arquitetura GUI do Android.

A GUI é um conjunto de ferramentas que fornece as classes necessárias para construir e gerenciar a interface do usuário. Ela fornece um objeto de aplicativo, manipulação de eventos, desenho de modelo, janelas, exibições e controles projetados especificamente para uma interface *touchscreen*. A estrutura da interface do Android é organizada segundo o padrão *Model-View-Controller* (MVC), como mostrado na Figura 2 (ROGERS et al., 2009, p. 157).

Figura 2 - MVC de uma aplicação Android



Fonte: Rogers et al. (2009, p. 158).

No Android o modelo MVC funciona, inicialmente, com a interação do usuário. Esta interação é feita pressionando teclas ou usando o toque em tela. O *Controller* é responsável por interpretar estes eventos. O *Model* processa o pedido do usuário e a *View* faz a renderização do resultado final na tela do dispositivo.

2.4 TÉCNICAS DE IHC PARA DISPOSITIVOS MÓVEIS

Uma boa interface de usuário baseia-se na forma como as pessoas pensam e trabalham, e não sobre as capacidades do dispositivo. Uma interface de usuário que é desinteressante, complicada ou ilógica pode fazer uma aplicação simples parecer uma tarefa complexa, mas uma interface bonita, intuitiva e atraente para o usuário aumenta a funcionalidade de um aplicativo e inspira um apego emocional positivo nos usuários (APPLE INC, 2012a, p. 19).

Segundo Apple Inc (2012a), para construir uma interface que atenda a estas características é necessário seguir algumas técnicas, como:

a) criar uma definição concisa e concreta da finalidade principal de um aplicativo e seu público-alvo;
b) desenhar o aplicativo para o dispositivo, respeitando o seu tamanho limitado;
c) equilibrar a personalização da interface gráfica e a clareza de propósito para manter a facilidade de uso dos recursos;
d) criar protótipos para testes com usuários. Mesmo com poucos usuários haverá um benefício de novas perspectivas sobre a funcionalidade da sua aplicação e a experiência do usuário.

Seguindo estas técnicas é possível refinar as idéias sobre os recursos que serão disponibilizados pela a aplicação e torná-la uma necessidade para o usuário (APPLE INC, 2012a, p. 22-29).

2.5 TRABALHOS CORRELATOS

Foram selecionados três trabalhos que possuem ligação com as funcionalidades propostas neste estudo: iSequence (BEEPSTREET INC, 2011), Melodica (CANDYCANE LLC, 2009) e GarageBand (APPLE INC, 2012b).

2.5.1 iSequence

O iSequence é uma ferramenta proprietária que opera sobre o sistema operacional iOS, exclusivamente para iPad (BEEPSTREET INC, 2011, p. 3), destinada à criação de música (Figura 3).

Figura 3 - Sequenciador principal do iSequence

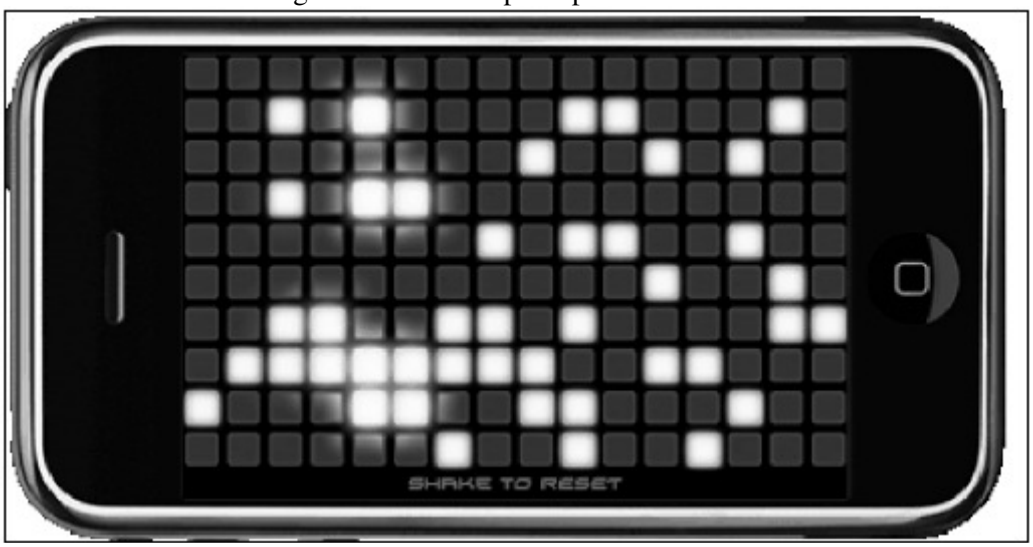
Fonte: BeepStreet Inc (2011).

A Figura 3 apresenta a interface principal do iSequence, dividida em três áreas. A área `A` contém um editor de música (lado direito) e o painel de transporte (lado esquerdo), onde é possível abrir outras telas como um *mixer*, gerenciador de música e lista de instrumentos. A área `B` possui um editor de sequência de notas musicais, onde todas as notas gravadas aparecem. Nesta área é possível inserir 32 notas em sequência e ajustar o volume de cada uma delas. A área `C` é reservada para o painel de instrumentos, como: piano, teclado e bateria. Estes instrumentos são utilizados na criação da música (BEEPSTREET INC, 2011, p. 5).

2.5.2 Melodica

Melodica é um sequenciador de música disponível para todos os dispositivos móveis que utilizam o sistema operacional iOS. Nele é possível definir os tipos de sons que serão executados em cada canal disponível em sua interface principal (CANDYCANE LLC, 2009). Como pode ser visto na figura 4, a aplicação utiliza efeito de luzes para cada som que é executado.

Figura 4 - Interface principal do Melodica



Fonte: Candycane LLC (2011).

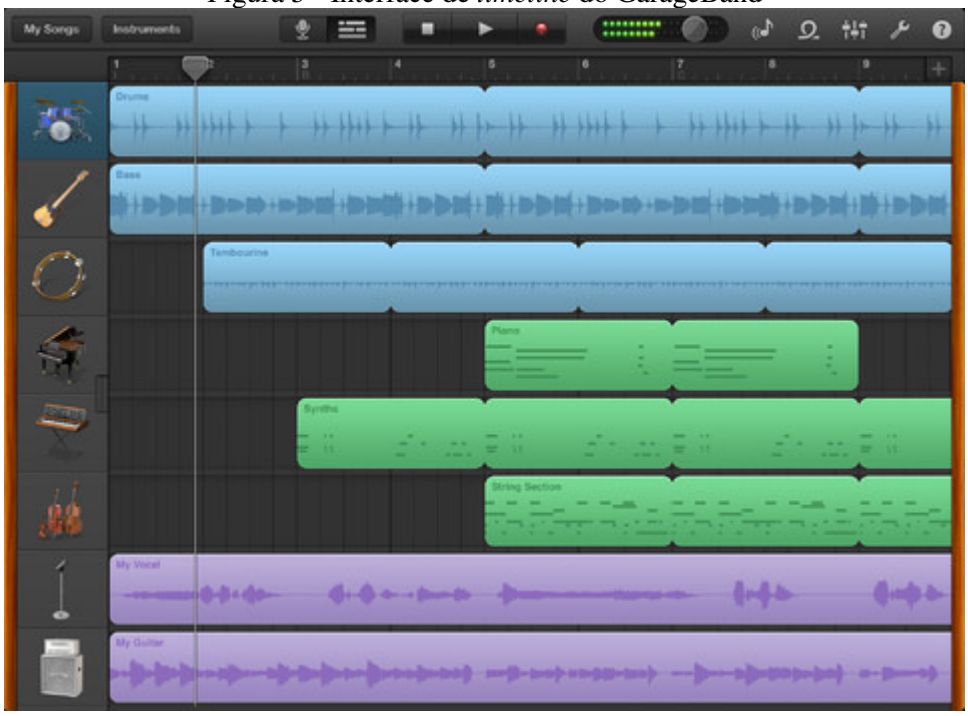
É possível escolher quatro temas de sons que acompanham o aplicativo, ou criar temas personalizados, onde o usuário pode escolher a escala do instrumento e o ritmo. Permite ainda salvar, carregar e compartilhar as músicas criadas.

A interface principal mostra a sequência de segmentos ou quadros luminosos para destacar em que momento cada canal será executado. Estes quadros são dispostos em uma matriz de 10 por 16 posições, e cada quadro é uma nota musical definida pelo usuário.

2.5.3 GarageBand

O GarageBand é uma ferramenta proprietária para composição musical disponível para os dispositivos que utilizam o sistema operacional iOS (há também uma versão para o sistema operacional MacOS, mas para este trabalho será considerada apenas sua versão para dispositivos móveis). A Figura 5 mostra sua interface.

Figura 5 - Interface de *timeline* do GarageBand



Fonte: Apple INC (2012b).

O objetivo do GarageBand é disponibilizar ferramentas para que qualquer pessoa, desde músicos experientes a leigos, possa criar e compartilhar músicas (APPLE INC, 2012b). O aplicativo contém instrumentos virtuais como pianos, teclados, guitarras e baterias, dando ao usuário a opção de utilizar acordes e ritmos pré-definidos ou criar as linhas musicais nota a nota; e ferramentas para editar, sincronizar e modificar as linhas criadas pelos instrumentos. A gravação de uma música pode ser feita tanto em tempo real, tocando-se as notas de cada instrumento com o acompanhamento de um metrônomo ou dos instrumentos já gravados, quanto editando-se as linhas musicais de modo estático, definindo manualmente as notas e tempos de cada instrumento.

2.5.4 Comparação entre os trabalhos correlatos

O Quadro 1 apresenta de forma comparativa algumas características dos trabalhos apresentados nesta seção.

Quadro 1 - Características dos trabalhos relacionados

características / trabalhos relacionados	iSequence (2011)	Melodica (2009)	GarageBand (2012)
tipo de usuário-alvo	avançados	leigos	leigos e avançados
foco musical	música eletrônica / ambiental	música eletrônica simples / *tone matrix*	música popular
apresenta a sequência dos elementos musicais em forma de *timeline*	sim	sim	sim
fornece *samples* para a criação das músicas	sim	não	sim
permite a edição das músicas nota a nota	sim	sim, obrigatoriamente	sim
permite o uso de múltiplos instrumentos	sim	não	sim
tempo de duração das músicas	loop	loop	início e fim definidos
permite salvar/exportar as músicas criadas	sim (formato próprio, MIDI, arquivo de áudio)	sim (formato próprio, arquivo de áudio)	sim (formato próprio, arquivo de áudio)
plataforma	iOS (tablet)	iOS (tablet / smartphone)	iOS (tablet / smartphone)

No Quadro 1, a primeira linha relaciona os trabalhos correlatos. A segunda linha informa o nível de conhecimento em composição musical esperado dos usuários de cada trabalho. A terceira linha informa o tipo de música que cada trabalho visa possibilitar. A quarta linha informa como a composição musical é visualmente apresenta ao usuário. A quinta linha informa se o trabalho fornece *samples* predefinidos para utilização pelo usuário. A sexta linha informa se o trabalho permite que cada nota de cada instrumento seja editada manualmente. A sétima linha informa se o trabalho permite o uso de múltiplos instrumentos em múltiplos canais de áudio. A oitava linha informa o modo de execução das músicas em relação a seu tempo de duração. A penúltima linha informa se o trabalho permite salvar as músicas para posterior edição e se permite que as músicas criadas sejam exportadas em formatos padrões de áudio. A última linha informa a plataforma à qual o trabalho se destina.

Comparando-se os trabalhos correlatos, pode-se notar que apesar de terem diferentes objetivos, há algumas semelhanças básicas entre eles. Todos apresentam a sequência dos elementos musicais em forma de *timeline*, e todos permitem que os usuários mais avançados editem as músicas no nível de elementos musicais fundamentais, editando cada nota de cada instrumento manualmente. Os aplicativos destinados a criação de músicas mais avançadas (iSequence, GarageBand) fornecem *samples* de instrumentos para facilitar e agilizar a criação das músicas. Os aplicativos destinados a criação de música eletrônica (iSequence, Melodica)

executam as músicas criadas continuamente em *loop*, fornecendo um tempo definido no qual os elementos musicas devem ser definidos; já o aplicativo GarageBand, destinado à criação de músicas populares, fornece um tempo com início e fim definidos, que pode ser expandido de acordo com a necessidade do usuário. Todos os aplicativos possibilitam o salvamento da música em formato próprio para posterior edição, e a exportação das músicas criadas em formatos comuns de áudio.

Todos os aplicativos relacionados destinam-se à plataforma iOS. Apesar de existirem aplicativos musicais para a plataforma Android, à qual se destina este trabalho, não foi encontrado durante este levantamento um aplicativo para esta plataforma que possuísse as características desejadas, especialmente no que diz respeito à usabilidade do aplicativo por usuários leigos.

3 DESENVOLVIMENTO DO APLICATIVO

Neste capítulo são abordadas as etapas envolvidas no desenvolvimento do aplicativo proposto. Na seção 3.1 são apresentados os principais requisitos do trabalho. Na seção 3.2 é exibida a especificação técnica do aplicativo. Na seção 3.3 é descrita a implementação do protótipo. Na seção 3.4 são detalhados os resultados alcançados.

3.1 REQUISITOS PRINCIPAIS DO APLICATIVO PROPOSTO

O aplicativo proposto deverá:

a) permitir a execução do áudio de acordo com as músicas criadas no mesmo (Requisito Funcional – RF);
b) permitir ao usuário configurar quais são os sons que serão executados em cada botão disponível (RF);
c) possibilidade de executar múltiplas faixas de áudio em canais diferentes (RF);
d) disponibilizar um *mixer* para controlar o volume de cada canal de áudio configurado (RF);
e) executar cada som configurado em tempo real, ou o mais próximo disso (RF);
f) disponibilizar uma interface simples e intuitiva ao usuário final, seguindo as técnicas de IHC para dispositivos móveis (Requisito Não Funcional – RNF);
g) ser implementada utilizando o ambiente de desenvolvimento Eclipse (RNF);
h) ser compatível com as versões do Android a partir da 3.0 (RNF).

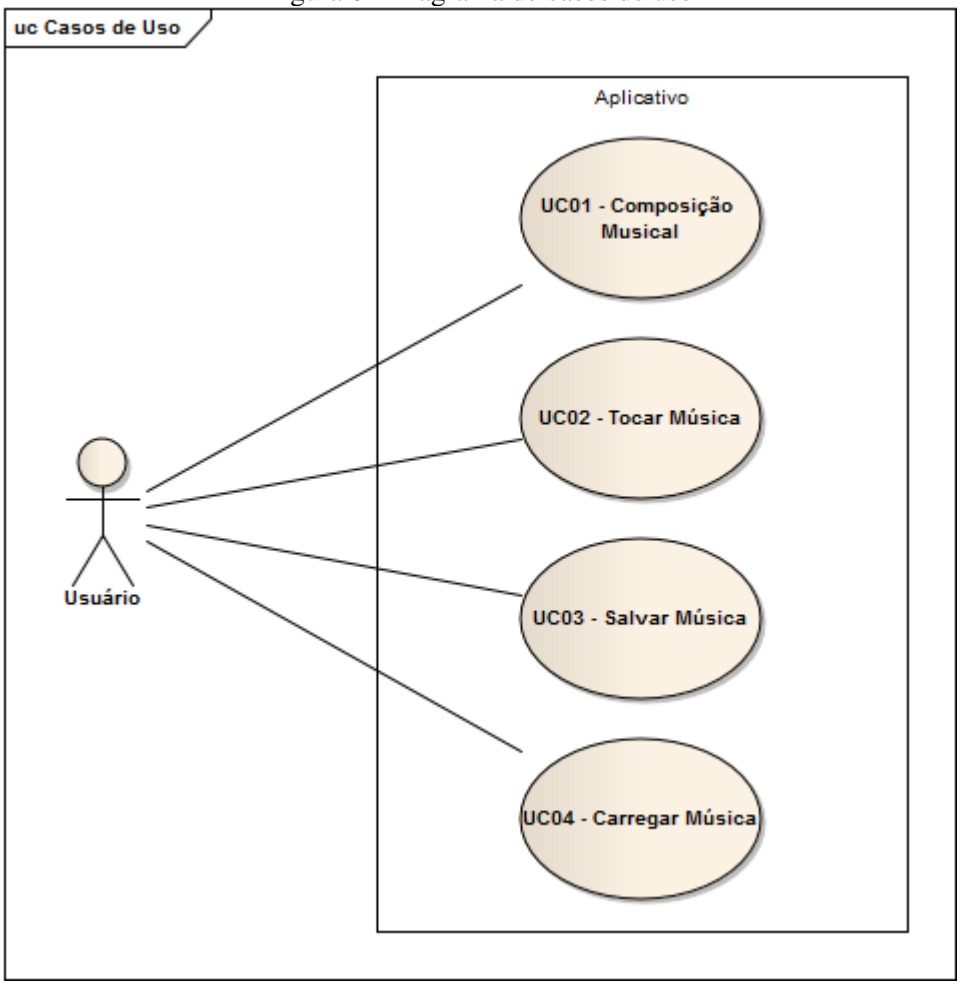
3.2 ESPECIFICAÇÃO

A especificação do protótipo apresentado neste trabalho foi feita utilizando a ferramenta Enterprise Architect versão 7.5. Através desta ferramenta foram desenvolvidos diagramas de caso de uso, de pacotes e de classes, que fazem parte da linguagem *Unified Modeling Language* (UML). Estes diagramas serão apresentados nas próximas seções.

3.2.1 Casos de uso

Nesta seção são descritos os casos de uso do protótipo, conforme apresentados na Figura 6. Foi identificado um ator relevante que interage com o protótipo, denominado `Usuário`.

Figura 6 - Diagrama de casos de uso



Os requisitos foram agrupados em quatro casos de uso, que serão descritos em detalhes nas próximas seções.

3.2.1.1 Composição Musical

O caso de uso de Composição Musical é a principal etapa na utilização do aplicativo de composição musical e é descrito no Quadro 2.

Quadro 2 – Caso de uso UC01

Número	01
Caso de uso	Composição Musical
Descrição	Caso de uso da funcionalidade que permite a composição de uma música no aplicativo.
Ator	`Usuário`
Pré-condições	Possuir o aplicativo de composição musical instalado no dispositivo móvel.
Pós-condições	Ter uma representação visual da música composta.
Cenário principal	1. O `Usuário` abre o aplicativo no dispositivo móvel; 2. O aplicativo exibe a interface principal; 3. O `Usuário` escolhe um `Sample` das listas de `Samples` disponíveis e o arrasta até a posição desejada na `Timeline`; 4. O passo 3 é repetido até o `Usuário` completar a música.

3.2.1.2 Tocar Música

O caso de uso de Tocar Música é utilizado durante o caso de uso de composição musical para verificar o andamento da composição e ao final deste caso de uso para ouvir a música composta. Este caso de uso é descrito no Quadro 3.

Quadro 3 – Caso de uso UC02

Número	02
Caso de uso	Tocar Música
Descrição	Caso de uso da funcionalidade que permite ao usuário ouvir músicas compostas no aplicativo.
Ator	`Usuário`
Pré-condições	Estar com o aplicativo aberto no dispositivo móvel e ter ao menos um `Sample` na `Timeline`.
Pós-condições	Ter executado a música de acordo com a composição aberta no aplicativo.
Cenário principal	1. O `Usuário` seleciona a opção `Tocar` no menu principal do aplicativo; 2. O aplicativo executa a música da primeira `Measure` da `Timeline` até a última `Measure` que contenha um `Sample`.
Cenário alternativo	Antes do passo 1, o `Usuário` pode selecionar uma `Measure` da `Timeline`. Nesse caso, no passo 2 o programa executará a música da `Measure` selecionada até a última `Measure` que contenha um `Sample`.

3.2.1.3 Salvar Música

O caso de uso de Salvar Música é usado para salvar o estado de uma composição em andamento ou uma composição completa e é descrito no Quadro 4.

Quadro 4 – Caso de uso UC03

Número	03
Caso de uso	Salvar Música
Descrição	Caso de uso da funcionalidade que permite salvar uma composição musical criada no aplicativo.
Ator	`Usuário`
Pré-condições	Estar com o aplicativo aberto no dispositivo móvel e ter ao menos um `Sample` na `Timeline`.
Pós-condições	Criar um arquivo com as informações da composição musical.
Cenário principal	1. O `Usuário` seleciona a opção `Salvar` no meu principal do aplicativo; 2. O aplicativo solicita o nome do arquivo a ser salvo com as informações da composição; 3. O `Usuário` informa o nome do arquivo; 4. O aplicativo salva as informações da composição musical no arquivo informado pelo `Usuário`.

3.2.1.4 Carregar Música

O caso de uso de Carregar Música é usado para carregar uma composição musical previamente salva no aplicativo e é descrito no Quadro 5.

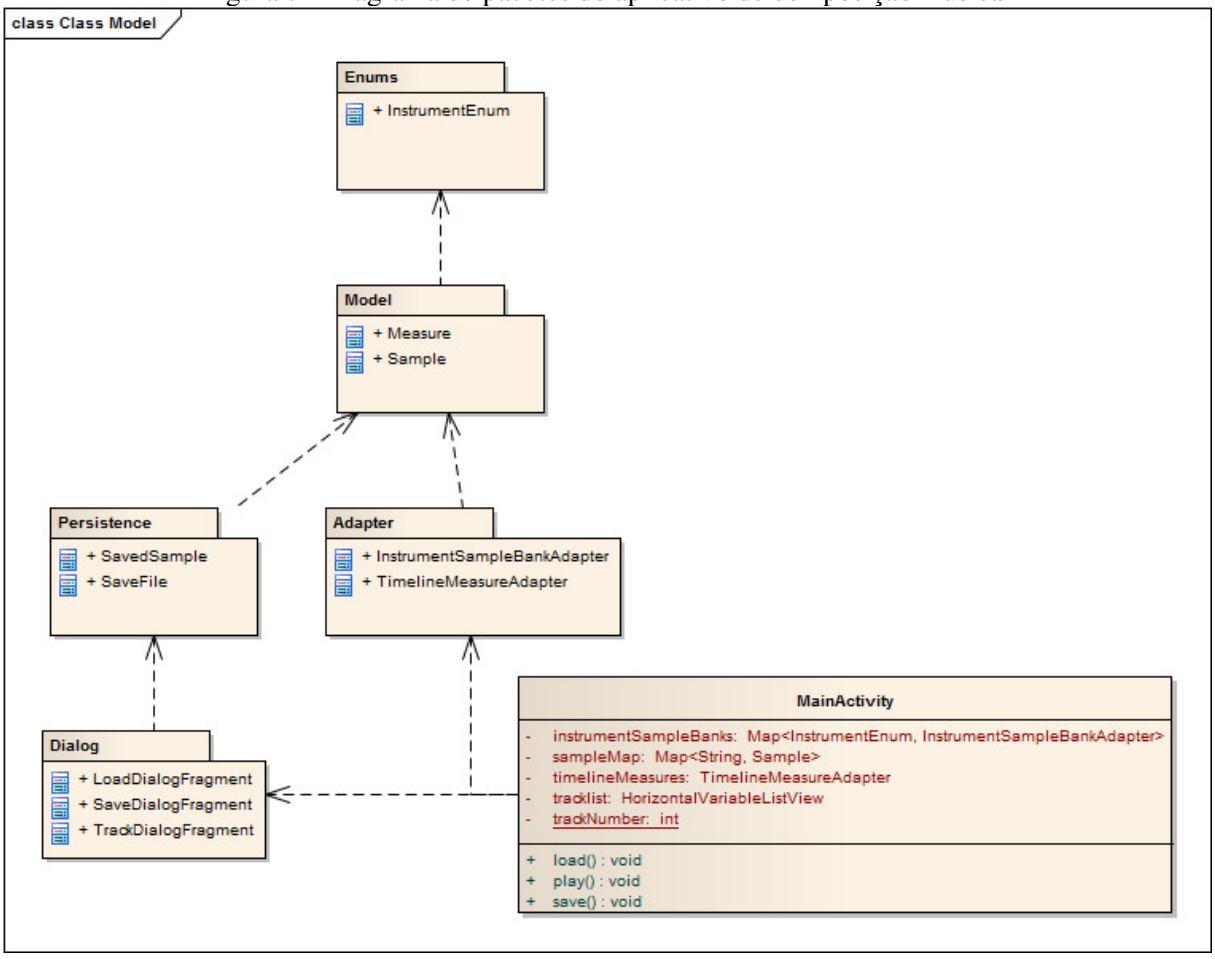
Quadro 5 – Caso de uso UC04

Número	04
Caso de uso	Carregar Música
Descrição	Caso de uso da funcionalidade que permite carregar uma composição musical do aplicativo.
Ator	`Usuário`
Pré-condições	Estar com o aplicativo aberto no dispositivo móvel e ter ao menos um arquivo de composição salva no dispositivo.
Pós-condições	Ter a composição musical aberta no aplicativo.
Cenário principal	1. O `Usuário` seleciona a opção `Abrir` no meu principal do aplicativo; 2. O aplicativo mostra uma lista com os arquivos disponíveis; 3. O `Usuário` escolhe um arquivo da lista; 4. O aplicativo carrega as informações do arquivo informado pelo `Usuário` e as exibe na `Timeline`.

3.2.2 Diagrama de classes

Nesta seção são descritos os relacionamentos e estrutura das classes desenvolvidas na implementação deste aplicativo. Na Figura 7 são demonstrados os agrupamentos lógicos das classes implementadas.

Figura 7 - Diagrama de pacotes do aplicativo de composição musical

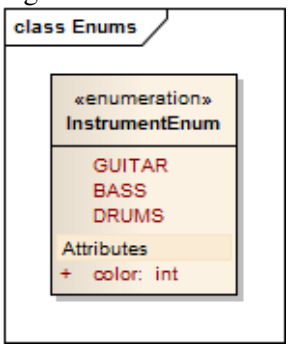


Nas próximas seções serão descritos em detalhes os pacotes do aplicativo e as classes que os compõem.

3.2.2.1 Pacote `enums`

O pacote `enums` contém a enumeração que determina o tipo de instrumento de cada `Sample`. Sua classe está representada na Figura 8.

Figura 8 - Pacote `enums`

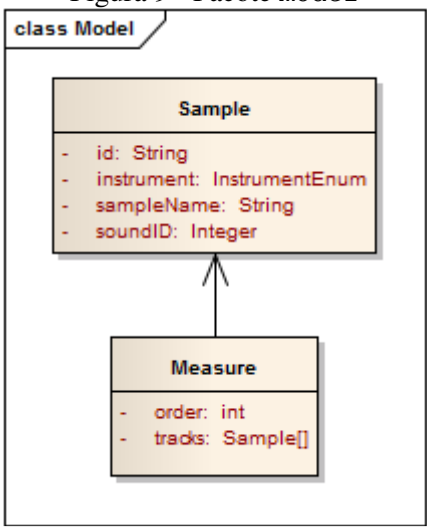


A enumeração `InstrumentEnum` identifica o tipo de instrumento de cada `Sample` e a cor representativa do instrumento na interface gráfica. Esta enumeração foi criada visando a criação futura de `Samples` de diferentes instrumentos.

3.2.2.2 Pacote `model`

O pacote `model` contém as classes que representam as abstrações dos conceitos referentes ao domínio da aplicação, sendo este a representação visual da composição musical, como pode ser observado na Figura 9.

Figura 9 - Pacote `model`



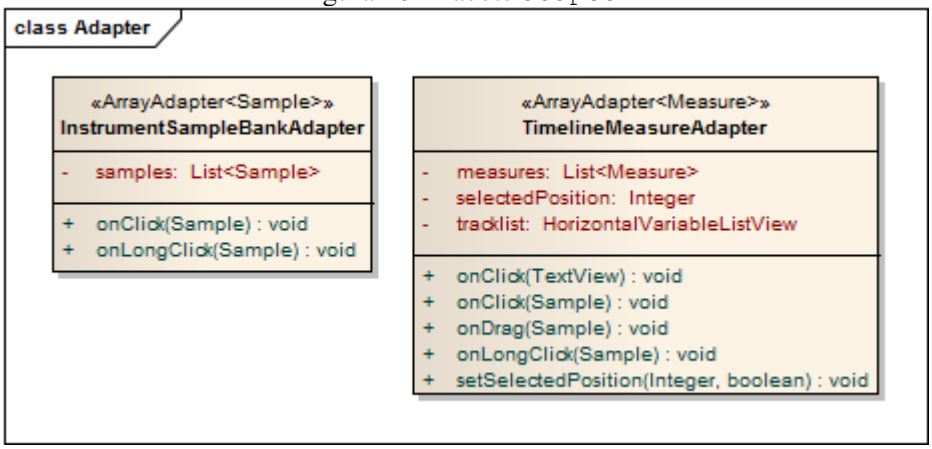
A classe `Sample` representa os sons de um determinado instrumento em um determinado espaço de tempo. O atributo `id` representa o identificador único de cada `Sample`, utilizado nas funcionalidades de persistência do aplicativo. O atributo `instrument` identifica o instrumento relacionado ao `Sample`. O atributo `sampleName` contém um nome representativo para o `Sample`, que é utilizado em conjunto com a cor definida pelo atributo `instrument` em sua representação visual na interface gráfica. O atributo `soundID` armazena o identificador do arquivo de áudio carregado em memória para este `Sample`.

A classe `Measure` representa o conjunto dos `Samples` que são executados ao mesmo tempo em trilhas de áudio diferentes. O atributo `order` representa a posição desta `Measure` na `Timeline` principal. O atributo `tracks` armazena os `Samples`, relacionando sua posição no `array` à sua trilha correspondente.

3.2.2.3 Pacote `adapter`

O pacote `adapter` contém as classes responsáveis pela exibição gráfica das estruturas internas de objetos das classes do pacote `model`, e pode ser observado na Figura 10.

Figura 10 - Pacote `adapter`



Todas as classes do pacote `adapter` extendem a classe `ArrayAdapter`, disponível na SDK do Android, que fornece as funcionalidades básicas do padrão de `Adapters` utilizado pelos componentes da SDK do Android.

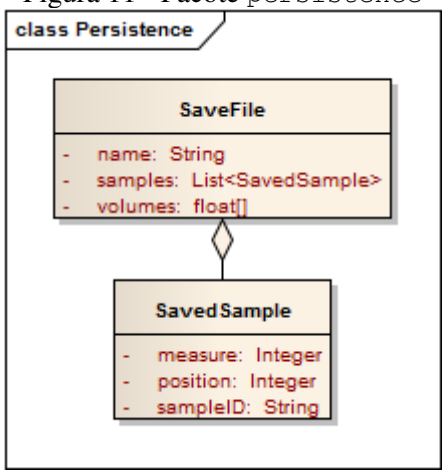
A classe `InstrumentSampleBankAdapter` representa o banco de `Samples` disponíveis para um determinado instrumento. O atributo `samples` armazena todos os `Samples` carregados no aplicativo para um determinado instrumento. O método `onClick` é chamado quando o `Usuário` realiza um toque em um `Sample` e executa o arquivo de áudio referente a este `Sample`. O método `onLongClick` é chamado quando o `Usuário` realiza um toque longo em um `Sample` e inicia a funcionalidade de *drag and drop* para a inserção de um novo `Sample` na `Timeline`.

A classe `TimelineMeasureAdapter` representa a `Timeline` principal do aplicativo e a lista de `Measures` que a compõe. O atributo `measures` armazena ordenadamente as `Measures` que compõe a música, e que são adicionadas e removidas dinamicamente à medida que o `Usuário` adiciona e remove `Samples` na `Timeline`. O atributo `selectedPosition` identifica a posição da `Measure` atualmente selecionada na `Timeline`. O atributo `tracklist` contém uma referência ao componente da interface gráfica que exibe a `Timeline`. O método `onClick(TextView)` é chamado quando o `Usuário` realiza um toque no indicador de tempo da `Measure` e define esta `Measure` como a seleção atual. O método `onClick(Sample)` é chamado quando o `Usuário` realiza um toque em um `Sample` e executa o arquivo de áudio referente a este `Sample`. O método `onDrag` é chamado quando o `Usuário` move um `Sample` até uma posição da `Timeline` e armazena este `Sample` nesta posição. O método `onLongClick` é chamado quando o `Usuário` realiza um toque longo em um `Sample` e inicia a funcionalidade de *drag and drop* para o reposicionamento ou remoção de um `Sample` na `Timeline`. O método `setSelectedPosition` é chamado pelo aplicativo para selecionar uma `Measure` e, de acordo com os parâmetros, posicioná-la no centro da visualização gráfica da `Timeline`.

3.2.2.4 Pacote `persistence`

O pacote `persistence` contém as estruturas utilizadas nas funcionalidades de persistência do aplicativo, e pode ser observado na Figura 11.

Figura 11 - Pacote `persistence`



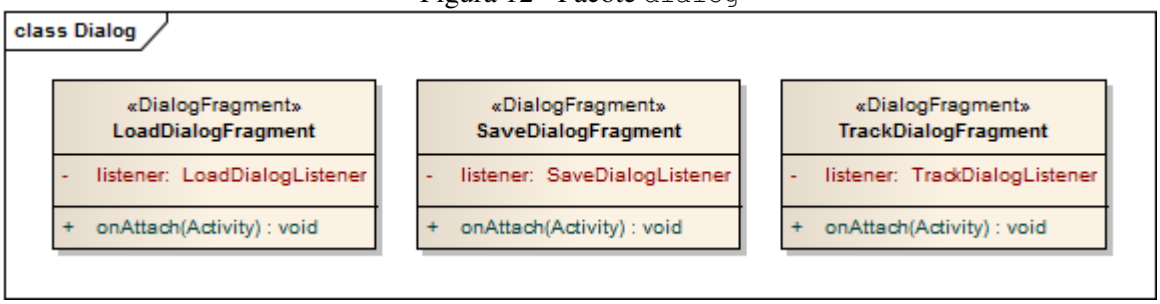
A classe `SavedSample` armazena as informações necessárias para a persistência de cada `Sample` que compõe a música. O atributo `measure` identifica o número da `Measure` na qual este `Sample` está armazenado. O atributo `position` identifica o número da trilha deste `Sample` dentro de sua `Measure`. O atributo `sampleID` armazena o identificador único do `Sample`, definido no atributo `id` da classe `Sample`.

A classe `SaveFile` agrupa os `SavedSamples` que são armazenados em um único arquivo. O atributo `name` contém o nome do arquivo. O atributo `samples` armazena todos os `SavedSamples` que compõe uma música do aplicativo. O atributo `volumes` armazena o nível de volume de cada trilha da música.

3.2.2.5 Pacote `dialog`

O pacote `dialog` contém as classes responsáveis pela exibição das janelas de diálogo customizadas do aplicativo, e pode ser observado na Figura 12.

Figura 12 - Pacote `dialog`



Todas as classes deste pacote extendem a classe `DialogFragment`, disponível na SDK do Android, que fornece as funcionalidades básicas de exibição de janelas de diálogo no

Android. O atributo `listener` destas classes armazena a referência a um `Listener` que definirá o comportamento do aplicativo referente às escolhas informadas nos diálogos, e o método `onAttach` garante que o `listener` seja definido no momento da exibição do diálogo.

A classe `LoadDialogFragment` exibe um diálogo com uma lista de todos os arquivos válidos encontrados no diretório de armazenamento padrão do dispositivo Android. A classe `SaveDialogFragment` exibe um diálogo com um campo de texto para que seja informado o nome do arquivo a ser salvo. A classe `TrackDialogFragment` exibe os controles de volume de cada trilha da música.

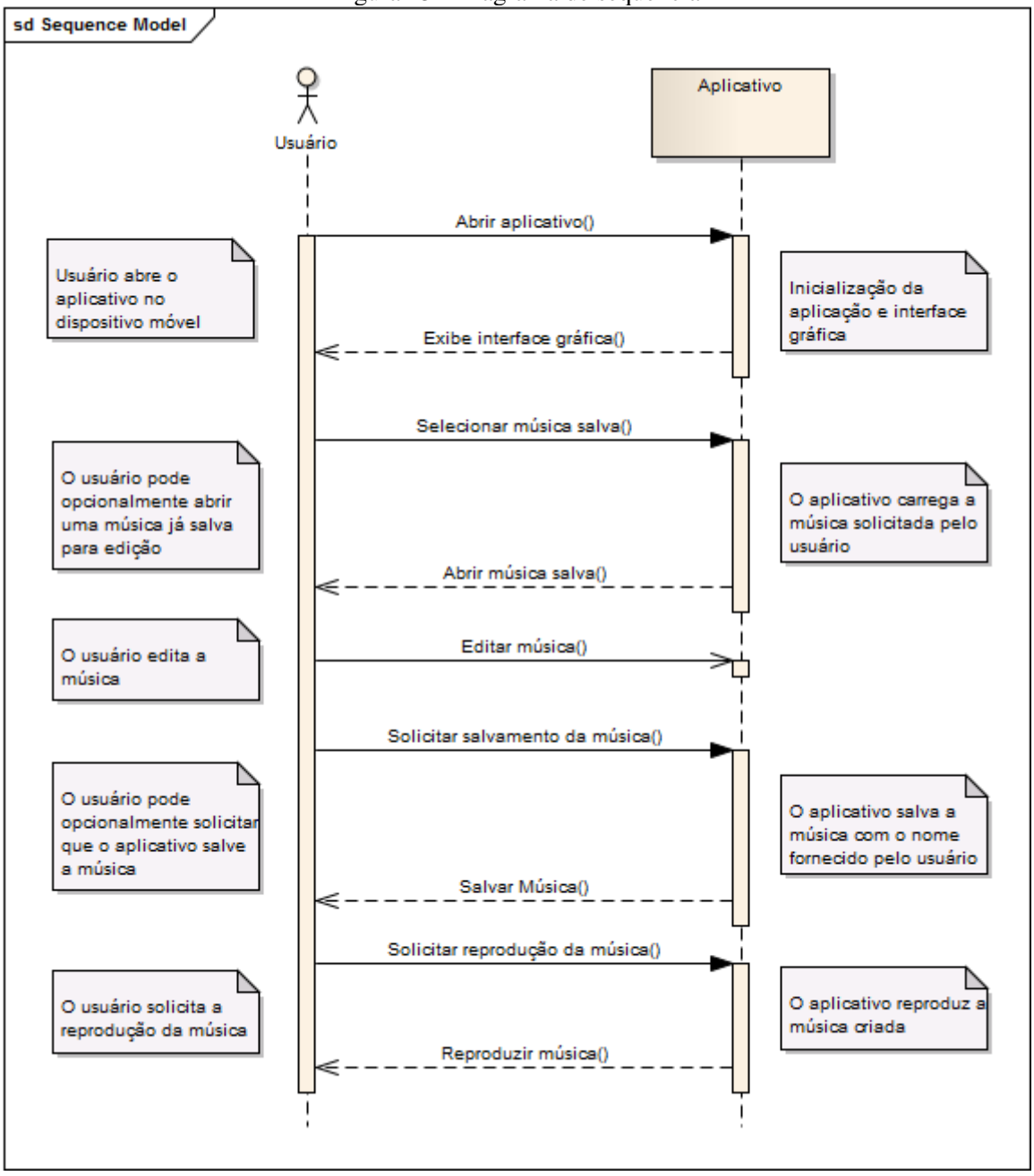
3.2.2.6 Classe `MainActivity`

A classe `MainActivity` representa a `Activity` principal do aplicativo, responsável por toda sua interface humano-computador. O atributo `instrumentSampleBanks` mapeia um `InstrumentSampleBankAdapter` para cada instrumento disponível na enumeração `InstrumentEnum`. O atributo `sampleMap` mapeia todos os `Samples` carregados pelo aplicativo a seus respectivos `id`. O atributo `timelineMeasures` referencia o `TimelineMeasureAdapter` que representa a `Timeline` principal do aplicativo. O atributo `tracklist` contém o componente que realiza a exibição gráfica da `Timeline`. O atributo estático `trackNumber` contém o número de trilhas disponíveis em cada `Measure`. O método `play` executa a música aberta no aplicativo. O método `save` exibe um `SaveDialogFragment` e salva as informações da música no arquivo informado. O método `load` exibe um `LoadDialogFragment` e carrega as informações da música do arquivo selecionado.

3.2.3 Diagrama de Sequência

O diagrama de sequência apresentado na Figura 13 contempla um cenário no qual o usuário passa por todos os casos de uso implementados no aplicativo.

Figura 13 - Diagrama de sequência



Na sequência apresentada na Figura 13, no primeiro passo o usuário abre o aplicativo no dispositivo móvel, que é inicializado e exibe sua interface gráfica. Em seguida, o usuário pode escolher carregar uma música já salva, nesse caso o programa carrega as informações do arquivo selecionado. Tendo o usuário selecionado uma música salva ou decidido iniciar uma nova, ele parte então para a etapa de edição da música. Após terminar a edição da música, ou a qualquer momento durante a edição, o usuário pode escolher salvar o progresso atual da música. Por último, o usuário solicita ao programa a execução da música, que então é reproduzida pelo programa de acordo com as definições do usuário.

3.3 IMPLEMENTAÇÃO

Nesta seção são apresentados os detalhes da implementação do protótipo. Na seção 3.3.1 são mostradas as técnicas e ferramentas utilizadas no desenvolvimento do aplicativo implementado. Na seção 3.3.2 são descritas as etapas da implementação. Na seção 3.3.3. é demonstrada a operacionalidade do aplicativo.

3.3.1 Técnicas e ferramentas utilizadas

Para o desenvolvimento do aplicativo foi utilizada a linguagem de programação Java. O desenvolvimento foi efetuado utilizando a ferramenta *Android Developer Tools* (ADT) 21.1. O ADT é um pacote que contém o ambiente de desenvolvimento Eclipse 3.7.2 com os plugins necessários para o desenvolvimento de aplicativos Android.

A *timeline* principal do aplicativo foi criada usando-se uma versão levemente modificada do componente `HorizontalVariableListView`.

3.3.2 Etapas da implementação

O funcionamento do aplicativo foi divido nesta seção em quatro etapas: inicialização, edição, persistência e reprodução.

Na seção 3.3.2.1 é descrita a etapa de inicialização, que trata da configuração inicial do aplicativo. Na seção 3.3.2.2 é descrita a etapa de edição, na qual o usuário edita os samples e configura as trilhas que compõe a música. Na seção 3.3.2.3 é descrita a etapa de persistência consistente de duas sub-etapas, salvar música e carregar música, que podem ser executadas pelo usuário a qualquer momento durante a etapa de edição. Por último, na seção 3.3.2.4 é descrita a etapa de reprodução, na qual é reproduzida a música criada pelo usuário.

3.3.2.1 Inicialização

Na etapa de inicialização, o aplicativo carrega em memória todos os *samples* e inicializa os bancos de *samples* dos instrumentos e a *timeline* do aplicativo.

O primeiro passo da etapa de inicialização é o carregamento em memória dos *samples* e inicialização dos bancos de *samples* dos instrumentos. Para exemplificar este processo o código é demonstrado no Quadro 6.

Quadro 6 - Exemplo de carregamento dos *samples* para a memória

```
1.  soundPool = new SoundPool(trackNumber, AudioManager.STREAM_MUSIC, 0);
2.
3.  sampleMap = new HashMap<String, Sample>();
4.  Sample sample;
5.
6.  List<Sample> guitarList = new ArrayList<Sample>();
7.  sample = new Sample("G1", "Em", InstrumentEnum.GUITAR,
      soundPool.load(this, R.raw.guitar_em, 0));
8.  sampleMap.put(sample.getId(), sample);
9.  guitarList.add(sample);
10.
11. sample = new Sample("G2", "A", InstrumentEnum.GUITAR,
      soundPool.load(this, R.raw.guitar_a, 0));
12. sampleMap.put(sample.getId(), sample);
13. guitarList.add(sample);
14.
15. sample = new Sample("G3", "D", InstrumentEnum.GUITAR,
      soundPool.load(this, R.raw.guitar_d, 0));
16. sampleMap.put(sample.getId(), sample);
17. guitarList.add(sample);
18.
19. sample = new Sample("G4", "G", InstrumentEnum.GUITAR,
      soundPool.load(this, R.raw.guitar_g, 0));
20. sampleMap.put(sample.getId(), sample);
21. guitarList.add(sample);
22.
23. guitarAdapter = new InstrumentSampleBankAdapter(this, guitarList);
```

No Quadro 6, pode-se ver na linha 1 a criação de um objeto da classe `SoundPool`, que é responsável por armazenar em memória os *samples* de áudio e executá-los quando necessário. Nas linhas 7, 11, 15 e 19 pode-se ver o modo como um *sample* de áudio é carregado em memória pelo `SoundPool`, passando-se a referência do recurso que contém o *sample* na função `SoundPool.load`. Esta função por sua vez retorna o *id* do *sample* carregado no `SoundPool`, que é armazenado em um novo objeto da classe `Sample`. Os `Samples` criados durante este processo são adicionados a uma lista de `Samples`, que é utilizada na inicialização de um objeto da classe `InstrumentSampleBankAdapter`, como pode ser visto na linha 23. Por fim, o mapa de `Samples` inicializado na linha 3 tem o objetivo de armazenar as referências de todos os `Samples` carregados no aplicativo, para ser utilizado posteriormente durante a etapa de persistência.

Por fim, a *timeline* do aplicativo é inicializada com um número determinado de `Measures` vazias, confome pode ser visto no Quadro 7.

Quadro 7 - Inicialização da *timeline* do aplicativo

```
1.  private List<Measure> createInitialMeasures() {
2.      List<Measure> list = new ArrayList<Measure>();
3.      createEmptyMeasures(list, 0, 9);
4.      return list;
5.  }
6.
7.  private void createEmptyMeasures(List<Measure> list,
                                     int start,
                                     int end) {
8.      Sample[] samples;
9.      for (int i = start; i <= end; i++) {
10.         samples = new Sample[trackNumber];
11.         for (int j = 0; j < trackNumber; j++) {
12.             samples[j] = null;
13.         }
14.
15.         list.add(new Measure(i, samples));
16.     }
17. }
```

O Quadro 7 mostra na linha 7 a função responsável por criar grupos de `Measures` vazias, recebendo como parâmetro a posição inicial e final do grupo de `Measures` na *timeline*. Esta mesma função é utilizada posteriormente para criar as `Measures` durante o processo de carregamento da etapa de persistência.

3.3.2.2 Edição

Na etapa de edição, o usuário edita a sequência de *samples* da música e configura o volume de cada trilha da música.

A edição é feita através de uma funcionalidade de *drag and drop*, arrastando-se os *samples* da área dos bancos de *samples* de instrumentos até a posição desejada na *timeline*. O Quadro 8 demonstra o código de criação do *listener* de toque longo, onde é configurado o comportamento do início do *drag and drop*.

Quadro 8 - Criação e configuração dos *samples* do banco de *samples* de instrumentos

```
1. public boolean onLongClick(View v) {
2.     Intent intent = new Intent();
3.     intent.putExtra("sample", item);
4.
5.     v.startDrag(ClipData.newIntent(item.getSampleName(), intent), new
           DragShadowBuilder(v), null, 0);
6.     Toast.makeText(getContext(), R.string.drag_tip,
           Toast.LENGTH_SHORT).show();
7.     return true;
8. }
```

No Quadro 8, na linha 2 é criado um objeto da classe `Intent`, uma classe padrão do Android que define a descrição de uma operação abstrata a ser executada. Na linha 3 é carregado como parâmetro neste `Intent` um objeto da classe `Sample`. Na linha 5 é iniciado o evento de *drag*. Na linha 6 é exibida uma mensagem ao usuário para avisá-lo do início do processo de *drag and drop*.

O processo de *drag and drop* é finalizado quando o usuário solta o *sample* em cima de uma das posições da *timeline*, conforme o código exibido no Quadro 9.

Quadro 9 - Processo de finalização do *drag and drop*

```
1.  case DragEvent.ACTION_DROP:
2.    Item dragItem = event.getClipData().getItemAt(0);
3.    Sample sample = (Sample)
            dragItem.getIntent().getExtras().getSerializable("sample");
4.
5.    Sample[] tracks = measure.getTracks();
6.    tracks[position] = sample;
7.
8.    if (measure.getOrder() + 9 > list.size()) {
9.          Sample[] samples;
10.         for (int i = list.size(); i < measure.getOrder() + 9; i++) {
11.               samples = new Sample[MainActivity.getTrackNumber()];
12.               for (int j = 0; j < MainActivity.getTrackNumber(); j++) {
13.                     samples[j] = null;
14.               }
15.
16.               list.add(new Measure(i, samples));
17.         }
18.   }
19.
20.   TimelineMeasureAdapter.this.notifyDataSetChanged();
21.
22.   return true;
```

No Quadro 9 é demonstrada a parte do código que realiza o tratamento do evento de *drop*. Na linha 3, é extraído do `Intent` recebido o `Sample` passado como parâmetro na inicialização do *drag and drop*. Nas linhas 5 e 6, este `Sample` é armazenado na estrutura de trilhas da `Measure` apropriada. Nas linhas 8 a 18, são inseridos novas `Measures` ao final da lista, caso seja necessário para que o usuário continue a edição da música. Por fim, na linha 20 é enviada uma notificação ao objeto da classe `TimelineMeasureAdapter` solicitando a atualização da interface gráfica.

A configuração do volume das trilhas da música é realizada em uma janela de diálogo contendo controles de volume separados para cada trilha. O código de criação desta janela de diálogo é exibido no Quadro 10.

Quadro 10 - Criação do diálogo de configuração de volumes

```
1.  public Dialog onCreateDialog(Bundle savedInstanceState) {
2.    AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
3.
4.    float[] volumes = this.getArguments()
                              .getFloatArray(MainActivity.PARAM_VOLUME);
5.
6.    final View view = getActivity().getLayoutInflater()
                                     .inflate(R.layout.tracks_dialog, null);
7.
8.    ((SeekBar) view.findViewById(R.id.trackvol0)).setProgress((int)
            (volumes[0] * 100));
9.    ((SeekBar) view.findViewById(R.id.trackvol1)).setProgress((int)
            (volumes[1] * 100));
10.   ((SeekBar) view.findViewById(R.id.trackvol2)).setProgress((int)
            (volumes[2] * 100));
11.   ((SeekBar) view.findViewById(R.id.trackvol3)).setProgress((int)
            (volumes[3] * 100));
12.
13.   builder.setTitle(R.string.volume).setView(view)
                              .setPositiveButton(R.string.ok,
       new DialogInterface.OnClickListener() {
14.         public void onClick(DialogInterface dialog, int id) {
15.               float[] volumes = new
                              float[MainActivity.getTrackNumber()];
16.
17.               volumes[0] = ((SeekBar)
            view.findViewById(R.id.trackvol0)).getProgress() / 100f;
18.               volumes[1] = ((SeekBar)
            view.findViewById(R.id.trackvol1)).getProgress() / 100f;
19.               volumes[2] = ((SeekBar)
            view.findViewById(R.id.trackvol2)).getProgress() / 100f;
20.               volumes[3] = ((SeekBar)
            view.findViewById(R.id.trackvol3)).getProgress() / 100f;
21.
22.               listener.onDialogTracksClick(TracksDialogFragment.this,
                                                 volumes);
23.         }
24.   });
25.
26.   return builder.create();
27. }
```

No Quadro 10 é demonstrada a criação da janela de diálogo de configuração de volumes. Na linha 4 são carregados os valores atuais de volume passados como parâmetro da `Activity` principal do aplicativo. Nas linhas 8 a 11, estes valores de volume são configurados nos elementos da interface gráfica. Nas linhas 14 a 24, é definido o *listener* responsável por retornar os novos valores de volume para a `Activity` principal. Nas linhas 17 a 20 são carregados os novos valores da interface gráfica, e na linha 22 estes valores são retornados à `Activity` principal.

3.3.2.3 Persistência

Na etapa de persistência, são realizados o salvamento e carregamento de informações de músicas.

Quando o usuário solicita o salvamento da música atual, fornecendo ao aplicativo um nome para o arquivo, o aplicativo salva as informações da música conforme o código demonstrado no Quadro 11.

Quadro 11 - Salvamento de uma música em arquivo

```
1.  public void onDialogSaveClick(DialogFragment dialog, String filename) {
2.    Gson gson = new Gson();
3.
4.    SaveFile save = new SaveFile(filename);
5.    save.setVolumes(this.trackVolumes);
6.    List<SavedSample> samples = new ArrayList<SavedSample>();
7.
8.    for (Measure measureDecorator : this.measureAdapter.getMeasureList())
                                                                        {
9.          Sample[] tracks = measureDecorator.getTracks();
10.
11.         for (int i = 0; i < tracks.length; i++) {
12.               if (tracks[i] != null) {
13                      samples.add(
      new SavedSample(measureDecorator.getOrder(), i, tracks[i].getId()));
14.               }
15.         }
16.   }
17.   save.setSamples(samples);
18.
19.   String json = gson.toJson(save);
20.
21.   File savedir = new File(Environment.getExternalStorageDirectory(),
                                                          "musicmaker");
22.
23.   if (savedir.mkdir() || savedir.isDirectory()) {
24.         File savefile = new File(savedir, filename);
25.         try {
26.               FileWriter writer = new FileWriter(savefile + ".txt");
27.               writer.write(json);
28.               writer.flush();
29.               writer.close();
30.
31.               Toast.makeText(this, "Música salva!",
                                          Toast.LENGTH_SHORT).show();
32.         } catch (IOException e) {
33.               e.printStackTrace();
34.         }
35.   }
36. }
```

No Quadro 11 é demonstrado o salvamento de uma música em um arquivo. Na linha 4 é criado um objeto da classe `SaveFile`, que guarda todas as informações necessárias ao salvamento da música. Nas linhas 8 a 16, as informações dos `Samples` inseridos na *timeline* são armazenadas. Na linha 8 são iteradas as `Measures` da *timeline*, e na linha 11 são iteradas as trilhas de cada `Measure`. Na linha 13, as informações necessárias de cada `Sample` encontrado são armazenadas em objetos da classe `SavedSample`. Após reunir todas as informações necessárias, o aplicativo converte o objeto `SaveFile` para o padrão *JavaScript Object Notation* (JSON) na linha 19 e salva o *stream* JSON em arquivo nas linhas 24 a 29.

Quando o usuário solicita a abertura de uma música salva, informando ao aplicativo o arquivo que contém a música, o aplicativo carrega as informações do arquivo conforme o código demonstrado no Quadro 12.

Quadro 12 - Carregamento de informações de uma música salva

```
1.  public void onDialogLoadClick(DialogFragment dialog, File loadfile) {
2.    String texto = new String();
3.
4.    try {
5.          BufferedReader reader = new BufferedReader(new
                                                    FileReader(loadfile));
6.          String linha;
7.          while ((linha = reader.readLine()) != null) {
8.                texto += linha;
9.          }
10.   } catch (IOException e) {
11.         e.printStackTrace();
12.   }
13.
14.   if (!texto.isEmpty()) {
15.         Gson gson = new Gson();
16.         SaveFile saveFile = gson.fromJson(texto, SaveFile.class);
17.
18.         this.trackVolumes = saveFile.getVolumes();
19.
20.         List<Measure> list = new ArrayList<Measure>();
21.
22.         for (SavedSample savedSample : saveFile.getSamples()) {
23.               if (list.size() <= savedSample.getMeasure()) {
24.                     createEmptyMeasures(list, list.size(),
                                          savedSample.getMeasure());
25.               }
26.
27.               Measure measureDecorator =
                                    list.get(savedSample.getMeasure());
28.               measureDecorator.getTracks()[savedSample.getTrack()] =
                                 sampleMap.get(savedSample.getSampleID());
29.         }
30.
31.         createEmptyMeasures(list, list.size(), list.size() + 9);
32.
33.         this.measureAdapter = new TimelineMeasureAdapter(this, list,
                                                        this.tracklist);
34.         this.tracklist.setAdapter(measureAdapter);
35.   }
36. }
```

No Quadro 12 é demonstrado o carregamento de informações de uma música salva em um arquivo. Nas linhas 4 a 12 é realizada a leitura do arquivo selecionado. Na linha 16, o *stream* JSON salvo no arquivo é convertido em um objeto da classe `SaveFile`. Nas linhas 18 a 34 são carregadas para a interface gráfica as informações recuperadas do arquivo.

3.3.2.4 Reprodução

Na etapa de reprodução, o aplicativo reproduz a música conforme editada pelo usuário. Esta etapa é realizada em duas partes, iniciando-se pela função demonstrada no Quadro 13.

Quadro 13 - Preparação para a reprodução da música

```
1.  private void play() {
2.    int lastMeasure = -1;
3.    for (int i = 0; i < measureAdapter.getCount(); i++) {
4.          Measure measure = measureAdapter.getItem(i);
5.          for (Sample sampleDecorator : measure.getTracks()) {
6.                if (sampleDecorator != null) {
7.                      lastMeasure = i;
8.                      break;
9.                }
10.         }
11.   }
12.
13.   if (lastMeasure > -1) {
14.         Integer startMeasure = measureAdapter.getSelectedPosition();
15.         if (startMeasure == null) {
16.               startMeasure = 0;
17.         }
18.
19.         play(startMeasure, lastMeasure);
20.   } else {
21.         Toast.makeText(this, R.string.empty_song,
                                                Toast.LENGTH_SHORT).show();
22.   }
23. }
```

No Quadro 13 é demonstrado o código da função chamada no momento que o usuário solicita a reprodução da música. Nas linhas 2 a 11 é realizada uma verificação para definir a última `Measure` da *timeline* que contém `Samples`. Caso não hajam `Samples` a serem executados, o aplicativo exibe ao usuário a mensagem definida na linha 21 avisando que a música está vazia. Se houver ao menos um `Sample` a ser executado, o aplicativo chama na linha 19 a função responsável por reproduzir os `Samples`, demonstrada no Quadro 14.

No Quadro 14 é demonstrado o código da função responsável por reproduzir a música criada pelo usuário. Esta função reproduz cada `Measure` da música de forma recursiva até não houver mais `Measures` com `Samples` a serem reproduzidos. Na linha 10 é solicitado ao `SoundPool` que execute o *sample* de áudio relacionado a um objeto `Sample` que se encontra em uma das trilhas da `Measure` em reprodução. Os parâmetros passados ao `SoundPool` são o ID recebido do `SoundPool` no momento em que o *sample* de áudio é carregado em memória, os volumes das saídas de áudio esquerda e direita, a prioridade de execução do *sample*, a quantidade de *loops* do *sample* e a taxa (velocidade) de reprodução. Depois de solicitar ao `SoundPool` a execução de todos os `Samples` da `Measure` atual, o programa verifica se ainda há `Measures` a serem executadas. Caso sim, o programa agenda nas linhas 15 a 21 a execução da próxima `Measure` ao momento do término da `Measure` atual. O acompanhamento da reprodução da música na *timeline* é realizada na linha 3 através da seleção da `Measure` em execução na timeline, e esta seleção é removida ao final da reprodução pelo agendamento realizado nas linhas 23 a 29.

Quadro 14 - Reprodução dos Samples da música

```
1.  private void play(final int measure, final int lastMeasure) {
2.    Measure measureDecorator = measureAdapter.getItem(measure);
3.    measureAdapter.setSelectedPosition(measure, true);
4.
5.    for (int track = 0; track < trackNumber; track++) {
6.          Sample sampleDecorator = measureDecorator.getTracks()[track];
7.
8.          if (sampleDecorator != null && sampleDecorator.getSoundID() !=
                                                                    null) {
9.                float volume = trackVolumes[track];
10.               soundPool.play(sampleDecorator.getSoundID(), volume,
                                                  volume, 1, 0, 1f);
11.         }
12.   }
13.
14.   if (lastMeasure > measure) {
15.         new Handler().postDelayed(new Runnable() {
16.
17.               @Override
18.               public void run() {
19.                     play(measure + 1, lastMeasure);
20.               }
21.         }, 2000);
22.   } else {
23.         new Handler().postDelayed(new Runnable() {
24.
25.               @Override
26.               public void run() {
27.                     measureAdapter.setSelectedPosition(null, true);
28.               }
29.         }, 2000);
30.   }
31. }
```

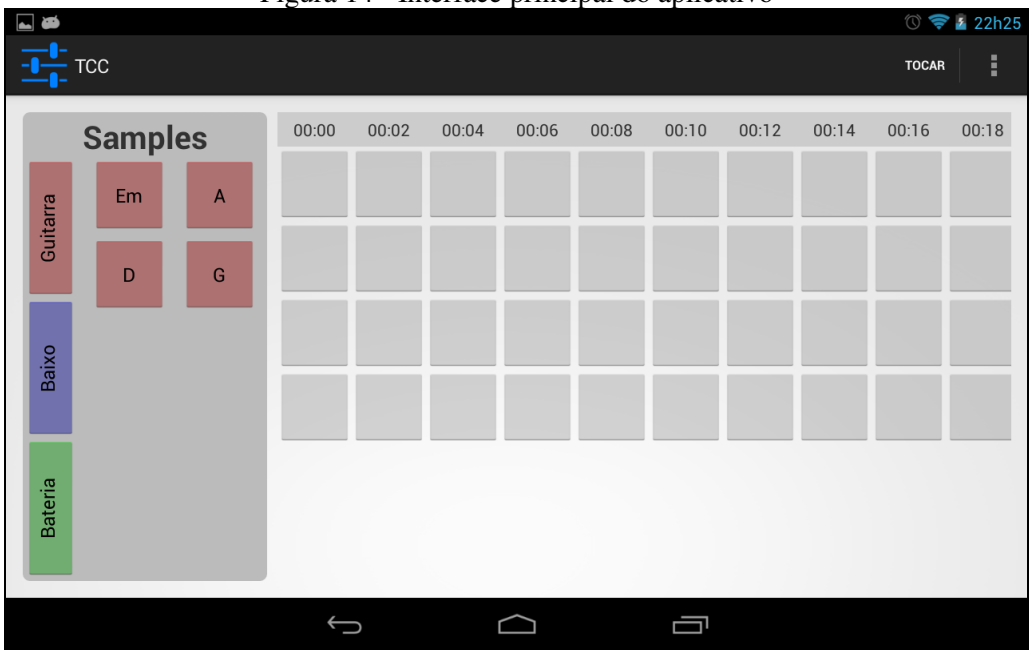
3.3.3 Operacionalidade da implementação

Esta seção apresenta a operacionalidade da implementação do ponto de vista do usuário e foi dividida em cinco seções de acordo com a ordem estabelecida no diagrama de sequência apresentado na Figura 13.

3.3.3.1 Abrir aplicativo

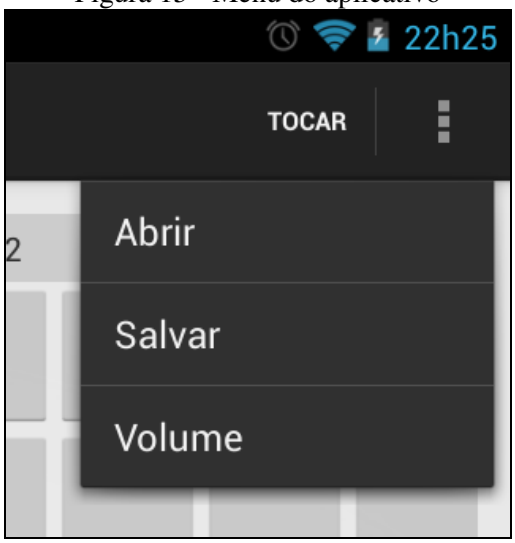
O aplicativo não requer nenhuma configuração inicial, bastando executar o arquivo de instalação. Após instalado, o ícone do aplicativo é exibido na lista de aplicativos do Android. Ao tocar-se no ícone, o programa é inicializado e exibe sua interface principal, conforme demonstrado na Figura 14.

Figura 14 - Interface principal do aplicativo



Na Figura 14 é exibida a interface principal do aplicativo. No quadro à esquerda encontram-se os bancos de *samples* por instrumento. No espaço à direita encontra-se a *timeline* do aplicativo. No menu padrão de aplicativos do Android encontram-se as opções do aplicativo, exibidas em mais detalhes na Figura 15.

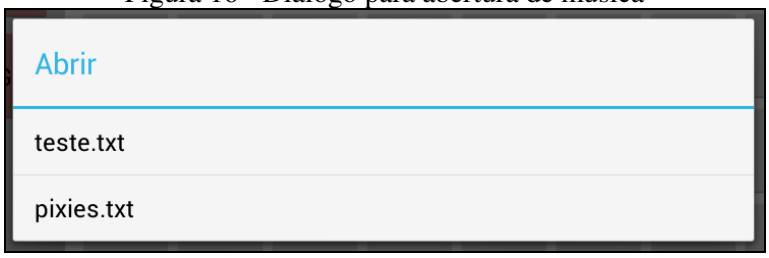
Figura 15 - Menu do aplicativo



3.3.3.2 Abrir música

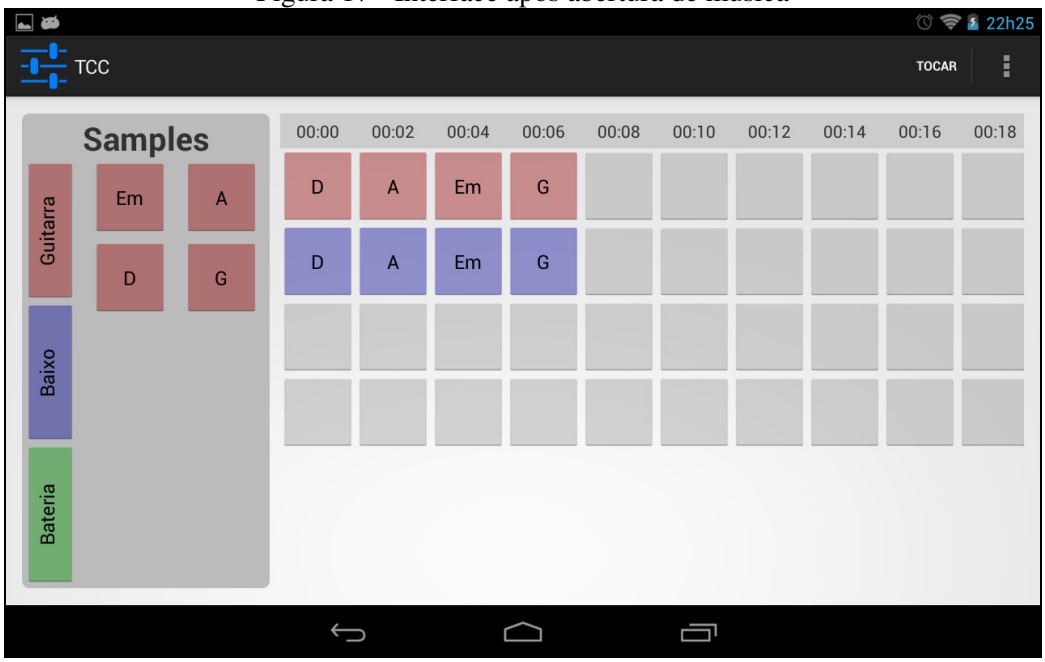
Caso o usuário deseje carregar uma música previamente salva no aplicativo, deve selecionar a opção Abrir do menu principal. Ao ser selecionada esta opção, o aplicativo exibe a janela de diálogo para escolha do arquivo a ser carregado, conforme demonstrado na Figura 16.

Figura 16 - Diálogo para abertura de música



Após o usuário selecionar um arquivo no diálogo exibido na Figura 16, o aplicativo carrega as informações do arquivo e atualiza a interface principal. Um exemplo de música carregada pode ser visto na Figura 17.

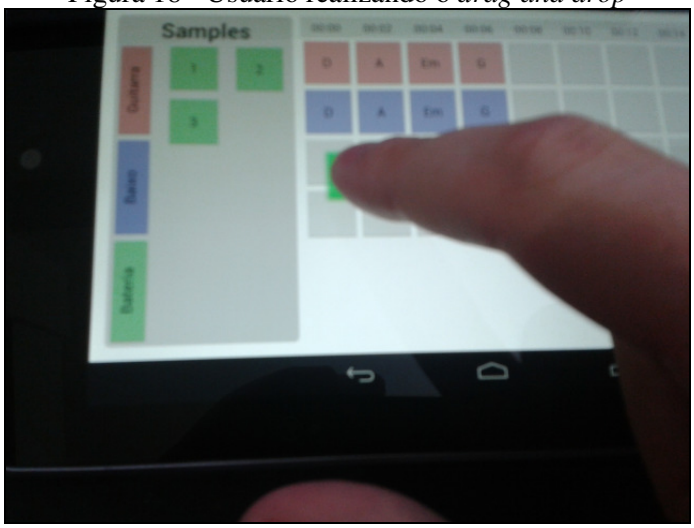
Figura 17 - Interface após abertura de música



3.3.3.3 Editar música

Para adicionar um *sample* à música o usuário primeiro deve escolher um instrumento e tocar no respectivo botão para que o aplicativo exiba o banco de *samples* do instrumento escolhido. Para verificar o conteúdo dos *samples* disponíveis o usuário pode realizar um toque simples no botão correspondente a um *sample* para que o aplicativo o reproduza. Após escolher o *sample* desejado o usuário deve realizar um toque longo no botão correspondente ao *sample* para iniciar a funcionalidade de *drag and drop*, arrastar o *sample* até a posição desejada na *timeline* e soltá-lo, conforme demonstrado na Figura 18.

Figura 18 - Usuário realizando o *drag and drop*



Para alterar a posição de um *sample* já inserido na *timeline*, o usuário deve realizar um toque longo no botão correspondente e arrastar o *sample* até a nova posição desejada. Para remover um *sample* da *timeline*, o usuário deve realizar um toque longo no botão correspondente e arrastar o *sample* para fora da *timeline*.

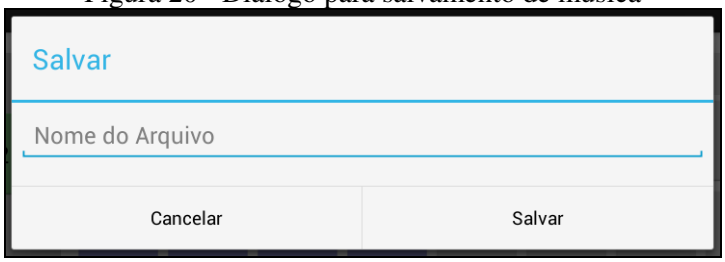
O volume de execução de cada trilha de áudio pode ser ajustado em separado. Para isso, o usuário deve escolher a opção `Volume` do menu principal, que abrirá a janela de diálogo exibida na Figura 19.

Figura 19 - Diálogo para ajuste de volumes

3.3.3.4 Salvar música

Para salvar em um arquivo as informações da música em edição o usuário deve selecionar a opção Salvar do menu principal, que abrirá a janela de diálogo exibida na Figura 20.

Figura 20 - Diálogo para salvamento de música



Após o usuário informar o nome do arquivo desejado e tocar no botão Salvar, o aplicativo salvará as informações da música em edição em um arquivo com o nome informado no diretório reservado ao aplicativo no dispositivo Android.

3.3.3.5 Reproduzir música

Para reproduzir a música em edição o usuário deve selecionar a opção Tocar do menu principal, que reproduzirá a música conforme demonstrado na Figura 21.

Figura 21 - Música em execução no aplicativo

Na Figura 21 pode ser vista a seleção de um tempo da *timeline*, que acompanha os *samples* sendo executados no momento. O usuário também pode escolher um tempo inicial para a execução da música selecionando o tempo desejado antes de selecionar a opção Tocar.

3.4 RESULTADOS E DISCUSSÃO

Nesta seção é detalhado a metodologia de avaliação, perfil dos usuários, aplicação do teste, análise e interpretação dos dados coletados e comparação entre as funcionalidades do aplicativo desenvolvido e os trabalhos correlatos.

Na seção 3.4.1 é descrito o experimento de usabilidade, no qual se avaliou a usabilidade do aplicativo pelo usuário final. Na seção 3.4.2 é descrito o experimento de compatibilidade, no qual se avaliou a compatibilidade do aplicativo com diversos dispositivos. Na seção 3.4.3 é feita uma comparação deste trabalho com seus trabalhos correlatos. Por fim, na seção 3.4.4 são discutidos alguns componentes utilizados neste trabalho.

3.4.1 Experimento de usabilidade

O experimento de usabilidade foi realizado com 10 usuários de diferentes perfis compatíveis com o público-alvo do aplicativo para avaliar a aceitação e eficiência das funcionalidades presentes no aplicativo.

3.4.1.1 Metodologia

O experimento foi realizado no mês de junho por meio de teste individual com os usuários. A cada usuário foi fornecido um dispositivo móvel com o aplicativo, um questionário de perfil, uma folha de instruções com o roteiro de testes e um questionário de usabilidade (Apêndice A).

Segundo Nielsen (1993), para se obter resultados satisfatórios em testes de usabilidade, são necessários apenas cinco usuários, alegando que, a partir do quinto usuário, se desperdiça tempo em observar praticamente os mesmos problemas repetidamente, sem encontrar novos erros. Porém, este experimento foi realizado com dez usuários para uma maior abrangência de perfis de usuário.

3.4.1.2 Aplicação do teste

Inicialmente cada participante foi recebido e orientado sobre os objetivos do teste de usabilidade. Antes de iniciar o experimento foi solicitado o preenchimento do questionário de perfil do usuário e foi fornecido ao usuário a folha de instruções.

As tarefas constadas no roteiro de testes procuravam abranger todas as atividades desempenhadas durante a utilização do aplicativo. Isso inclui desde a abertura do aplicativo, execução dos *samples*, edição da música, configuração dos volumes por trilha e salvamento e abertura de músicas.

Ao término das tarefas, foi apresentado aos usuários um formulário com questões abertas e fechadas. O questionário fechado foi composto por 8 perguntas que tinham como intuito captar a percepção quantitativa do usuário em relação à eficiência do sistema, sua

usabilidade e funcionalidades, e a receptividade do usuário em relação às extensões propostas ao aplicativo. Já os comentários do usuário e as perguntas abertas buscam instituir o contexto (percepção qualitativa) e permitir ao usuário sugerir alterações e novas extensões para o aplicativo.

As sessões de experimentos duraram em média entre 5 e 10 minutos, sendo realizadas utilizando um dispositivo Nexus 7. Os resultados deste experimento podem ser vistos na próxima seção.

3.4.1.3 Análise e interpretação dos dados coletados

A análise dos dados foi iniciada a partir dos dados coletados pelo questionário de perfil do usuário. A distribuição dos perfis dos usuários escolhidos para o experimento é exibida no Quadro 15.

Quadro 15 - Distribuição dos perfis de usuário

sexo	80% masculino 20% feminino
idade	20% menos de 18 anos 40% entre 18 e 25 anos 40% entre 25 e 35 anos
nível de escolaridade	10% ensino fundamental completo 20% ensino médio incompleto 10% ensino médio completo 10% ensino superior incompleto 50% ensino superior completo
possui dispositivo móvel	80% sim 20% não
grau de familiaridade com música	10% leigo em música 20% escuta música casualmente 30% apreciador de música 30% músico amador 10% músico profissional

Após análise do perfil do usuário, iniciou-se a interpretação e apresentação dos resultados obtidos a partir do questionário de usabilidade. O Quadro 16 mostra os resultados obtidos a partir de perguntas objetivas (questionário fechado).

Quadro 16 - Respostas da avaliação do sistema

Perguntas / Respostas	**Sim**	**Não**
1. Você conseguiu seguir os passos das instruções de uso sem dificuldades?	100%	
2. Você achou o aplicativo intuitivo e fácil de usar?	100%	
3. Dentro do escopo limitado do protótipo, você conseguiu criar uma música que considera satisfatória?	90%	10%
4. Você conseguiu identificar os diferentes *samples* sem dificuldades?	100%	
5. Você gostaria que fossem disponibilizados novos *samples* e instrumentos no aplicativo?	100%	
6. Você gostaria que fosse possível compartilhar uma música com outros usuários através do aplicativo, para criar e editar músicas colaborativamente?	90%	10%
7. Você gostaria que o aplicativo armazenasse o histórico de versões de cada música?	80%	20%
8. Você gostaria que o aplicativo exportasse as músicas criadas em formatos de áudio (ex. MP3)?	90%	10%

A partir dos resultados das questões 1 e 2 do Quadro 16 percebeu-se que os usuários não tiveram dificuldades na execução das tarefas solicitadas e que consideram o aplicativo intuitivo e fácil de usar.

Através da questão 3 identificou-se que a maioria dos usuários ficou satisfeita com as músicas que criaram utilizando o aplicativo.

Na questão 4 pode-se perceber que todos os usuários conseguiram identificar os *samples* utilizados no aplicativo.

Com os resultados das questões 5 a 8 nota-se uma grande aceitação às extensões propostas, sendo unânime o desejo por uma maior variedade de samples e instrumentos.

Além dos resultados obtidos através do questionário de usabilidade, foram coletadas opiniões sobre o ambiente por meio de questões abertas. Dentre os comentários feitos pelos usuários ao final do experimento ou ao responder o questionário aberto, destacam-se os itens a seguir:

a) apesar de não terem dificuldades para usar o aplicativo, alguns usuários sugeriram mudanças no modo de edição da *timeline*, como iniciar o *drag and drop* imediatamente com um toque no botão ao invés de um toque longo, ou apenas tocar em um *sample* para selecioná-lo e depois tocar no local desejado da *timeline*;
b) foi sugerido que fosse possível ao usuário adicionar seus próprios *samples*, além de um controle de andamento e tempo dos mesmos;
c) o usuário que identificou-se como leigo em música comentou que a representação

utilizada para os *samples*, apesar de suficientemente clara para o escopo atual do protótipo, pode tornar-se confusa com uma quantidade maior de *samples* para cada instrumento;

d) alguns usuários perceberam uma leve demora entre a execução de *samples* em tempos diferentes, que ocorria de modo aleatório mas tornava-se mais comum de acordo com o número de *samples* em um mesmo tempo.

Com os resultados obtidos no experimento realizado é possível concluir que o sistema não apresentou grandes problemas em sua utilização.

3.4.2 Experimento de compatibilidade

O experimento de compatibilidade foi realizado com 4 dispositivos Android para verificar a compatibilidade do aplicativo com variados *hardwares* e versões do Android. Os dispositivos utilizados são listados no Quadro 17.

Quadro 17 - Dispositivos utilizados no experimento de compatibilidade

Dispositivo	Versão do Android	Tamanho da tela
Google Nexus 7 (dispositivo de desenvolvimento)	4.2.2	7"
Motorola Xoom	4.0.4	10.1"
Acer Iconia	4.0.3	10.1"
Samsung Galaxy Tab 8.9	3.1	8.9"

Em todos os dispositivos testados a interface foi exibida corretamente e todas as funcionalidades do aplicativo funcionaram sem problemas. Porém, no dispositivo Acer Iconia os *samples* de bateria não foram executados corretamente, sendo substituídos por outro instrumento. Tendo em vista que o problema não ocorreu nos outros dispositivos testados, conclui-se que o mesmo foi causado por uma falha na implementação do padrão MIDI no dispositivo, não tendo relação direta com o aplicativo testado.

Não foi possível testar o aplicativo em um dispositivo do tipo *smartphone*, mas acredita-se que o aplicativo não seja compatível com tais dispositivos. Embora a única restrição técnica ao uso do aplicativo seja a presença do Android 3.0, o *layout* criado para o aplicativo é otimizado para *tablets* e não deve ser exibido corretamente em dispositivos com telas menores, dificultando sua utilização.

3.4.3 Comparação com trabalhos correlatos

Após a análise dos resultados obtidos, reformulou-se o quadro das características dos trabalhos correlatos (Quadro 1 da seção 2.5.4) adicionando o aplicativo desenvolvido, para fins de comparação. O Quadro 18 apresenta a análise comparativa entre as características do aplicativo desenvolvido com as características existentes nos trabalhos relacionados.

Quadro 18 - Comparação com os trabalhos correlatos

características / trabalhos relacionados	iSequence (2011)	Melodica (2009)	GarageBand (2012)	Aplicativo proposto
tipo de usuário-alvo	avançados	leigos	leigos e avançados	leigos
foco musical	música eletrônica / ambiental	música eletrônica simples / *tone matrix*	música popular	música popular
apresenta a sequência dos elementos musicais em forma de *timeline*	sim	sim	sim	sim
fornece *samples* para a criação das músicas	sim	não	sim	sim
permite a edição das músicas nota a nota	sim	sim, obrigatoriamente	sim	não
permite o uso de múltiplos instrumentos	sim	não	sim	sim
tempo de duração das músicas	loop	loop	início e fim definidos	início e fim definidos
permite salvar/exportar as músicas criadas	sim (formato próprio, MIDI, arquivo de áudio)	sim (formato próprio, arquivo de áudio)	sim (formato próprio, arquivo de áudio)	sim (formato próprio)
plataforma	iOS (tablet)	iOS (tablet / smartphone)	iOS (tablet / smartphone)	Android (tablet)

A partir do Quadro 17, pode-se observar que o aplicativo proposto contempla boa parte das características existentes nos trabalhos relacionados (vide seção 2.5.4), sendo que as diferenças do aplicativo desenvolvido são o seu foco em criação de música popular por leigos e o uso da plataforma Android.

3.4.4 Discussão sobre os componentes utilizados

Nesta seção são discutidos alguns componentes utilizados neste trabalho. Na seção 3.4.4.1 é discutido o componente utilizado para a exibição da *timeline*. Na seção 3.4.4.2 são discutidos os componentes utilizados para a execução dos *samples*.

3.4.4.1 Exibição da *timeline*

Para a exibição da *timeline*, foi utilizada a estrutura padrão de Adapters do Android, que trouxe consigo algumas dificuldades e limitações.

Sua maior dificuldade foi a de não haver na SDK padrão do Android um componente que exibisse o conteúdo de um Adapter verticalmente. Após pesquisa, decidiu-se utilizar o componente de código aberto HorizontalVariableListView, ao qual foram realizadas algumas modificações para permitir uma melhor manipulação do componente. Também foi necessário desenvolver um Adapter próprio para a exibição da *timeline*.

Sua maior limitação é o fato de, por agrupar os *samples* em colunas com tempos definidos, torna-se necessário que os *samples* utilizados tenham todos o mesmo tempo definido para as colunas.

3.4.4.2 Execução dos *samples*

Para a execução dos *samples*, usou-se primeiramente o componente MediaPlayer da SDK do Android, que permite a execução de variados formatos de mídia. Porém, verificou-se durante o desenvolvimento da *timeline* que este componente causava uma latência considerável entre as execuções de cada tempo de *samples*, o que deve-se ao fato de ser necessário criar uma instância de MediaPlayer para cada *sample* executado e carregá-lo para a memória no momento de sua execução.

Após pesquisas e testes, foi decidido então utilizar o componente SoundPool da SDK do Android. Este componente possui uma única instância que é criada durante a inicialização do aplicativo e que mantém em memória todos os samples carregados, permitindo assim um acesso mais rápido no momento da execução dos *samples*.

4 CONCLUSÕES

Este trabalho abordou a construção de um aplicativo para composição de músicas em dispositivos móveis voltado a usuários sem experiência.

Tendo em vista os resultados obtidos através de testes e experimentos, pode-se concluir que este trabalho alcançou seu principal objetivo de um modo satisfatório, permitindo que usuários leigos na composição musical criem músicas com o auxílio do aplicativo.

Quanto aos objetivos específicos, foi possível criar uma representação sonora compreensível o suficiente para a diferenciação dos *samples* durante a composição musical. Porém, foi apontado por usuários leigos em música que a representação escolhida pode tornar-se confusa com um número mais elevado de *samples* para cada instrumento. Também foi possível disponibilizar aos usuários um banco de *samples* para a criação das músicas, de modo que este banco possa ser facilmente extendido com novos *samples* e instrumentos.

A principal limitação atual da implementação se encontra no fato de que os samples utilizados no aplicativo devem obrigatoriamente ter a mesma duração. Um dos maiores desafios durante a implementação deste aplicativo foi o sincronismo na execução dos *samples* que, como evidenciado no experimento com os usuários, pode em alguns casos causar um atraso breve mas perceptível entre a execução dos *samples* de tempos consecutivos.

4.1 EXTENSÕES

Como extensões para este trabalho, sugerem-se:

a) otimizar o processo de reprodução da música para eliminar os atrasos ocasionais entre a execução dos *samples* de tempos consecutivos;
b) criar uma nova representação musical baseada em imagens ou símbolos que sejam mais intuitivas para os usuários leigos;
c) criar um *layout* de interface otimizado para uso em *smartphones*;
d) alterar a estrutura de dados do aplicativo para possibilitar o uso de *samples* com diferentes durações;
e) permitir a criação de *samples* pelo usuário no próprio aplicativo;
f) criar um sistema de compartilhamento e edição colaborativa de músicas integrado ao aplicativo;
g) alterar a funcionalidade de persistência do aplicativo para salvar as versões anteriores das músicas;

h) exportar a música criada em formato MIDI ou formatos comuns de áudio;
i) adicionar novos modos de edição da música, como os solicitados pelos usuários nos experimentos realizados (seção 3.4.1.3).

REFERÊNCIAS BIBLIOGRÁFICAS

ANDRADE, Mario de. **Pequena história da música**. 8. ed. São Paulo: Martins, 1977.

ANDROID DEVELOPERS. [S.l.], 2012. Disponível em: <http://developer.android.com>. Acesso em: 06 mar. 2013.

APPLE INC. **iOS human interface guidelines**. [S.l.], 2012a. Disponível em: <https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/MobileHIG.pdf>. Acesso em: 06 mar. 2013.

_____. **GarageBand**. [S.l.], 2012b. Disponível em: <http://www.apple.com/br/apps/garageband/>. Acesso em: 06 mar. 2013.

BEEPSTREET INC. **iSequence for iPad 2.1:** user manual. [S.l.], 2011. Disponível em: <http://www.beepstreet.com/iseqhd/iSequence%20for%20iPad%20user%20manual%20printable.pdf>. Acesso em: 06 mar. 2013.

B'FAR, Reza. **Mobile computing principles:** designing and developing mobile. New York: Cambridge University Press, 2005.

CANDYCANE LLC. **Melodica**. [S.l.], 2009. Disponível em: <http://www.candycaneapps.com/melodica/>. Acesso em: 06 mar. 2013.

GRIFFITHS, Paul. **A música moderna:** uma história concisa e ilustrada de Debussy a Boulez. Tradução Clóvis Marques. Rio de Janeiro: Jorge Zahar, 1998.

HASS, Jeffrey. **How MIDI works**. [S.l.], 2010. Disponível em: <http://www.indiana.edu/~emusic/etext/MIDI/chapter3_MIDI.shtml>. Acesso em: 26 mar. 2013.

LARGE, Edward W.; PALMER, Caroline. Perceiving temporal regularity in music. **Cognitive Science**, [S.l.], v. 26, n. 1, p. 1-37, Jan./Fev. 2002. Disponível em: <http://dx.doi.org/10.1016/S0364-0213(01)00057-X>. Acesso em: 29 mar. 2013.

MILETTO, Evandro M. et al. Educação musical auxiliada por computador: algumas considerações e experiências. In: CICLO DE PALESTRAS NOVAS TECNOLOGIAS NA EDUCAÇÃO, 3., 2004, Porto Alegre. **Anais**... Porto Alegre: UFRGS, 2004. Disponível em: <http://hdl.handle.net/10183/549>. Acesso em: 06 mar. 2013.

NIELSEN, Jakob. **Usability Engineering.** San Franciso: Morgan Kaufmann Publishing, 1993.

ROGERS, Rick et al. **Desenvolvimento de aplicações Android**. Tradução Lia Gabriele Regius. São Paulo: Novatec, 2009.

APÊNDICE A – Roteiro e Questionários de Avaliação de Usabilidade

Neste apêndice estão o roteiro e os questionários apresentados aos usuários durante os testes de usabilidade descritos na seção 3.4.1. O Quadro 19 apresenta o questionário de perfil de usuário. O Quadro 20 apresenta o roteiro de testes do aplicativo. O Quadro 21 apresenta o questionário de usabilidade do aplicativo.

Quadro 19 - Questionário de perfil de usuário

PERFIL DE USUÁRIO

Observação: as informações recebidas abaixo serão mantidas de forma confidencial.

Sexo: ( ) Masculino ( ) Feminino

Idade:
( ) Tenho menos de 18 anos
( ) Tenho entre 18 e 25 anos
( ) Tenho entre 25 e 35 anos
( ) Tenho mais de 35 anos

Nível de escolaridade:
( ) Ensino fundamental incompleto
( ) Ensino fundamental completo - 1º grau
( ) Ensino médio incompleto
( ) Ensino médio completo - 2º grau
( ) Ensino superior incompleto
( ) Ensino superior completo

Você possui algum dispositivo móvel (smartphone/tablet)?
( ) Sim ( ) Não

Indique seu grau de familiaridade com música:
( ) Sou leigo em música
( ) Escuto música casualmente
( ) Sou apreciador de música, consigo identificar ritmos e instrumentos
( ) Sou músico amador, sei tocar um instrumento
( ) Sou músico experiente, sei tocar um ou mais instrumentos e compor músicas para eles

Quadro 20 - Roteiro de testes do aplicativo

INSTRUÇÕES

Neste experimento estamos interessados em obter um melhor entendimento acerca da usabilidade do protótipo de aplicativo para composições musicais.

Este estudo visa especificamente coletar as impressões dos usuários ao utilizar o aplicativo, para que possamos adequá-lo à necessidade dos usuários. Portanto, gostaríamos que utilize o aplicativo seguindo as instruções abaixo e responda ao questionário da próxima página.

Instruções de uso do aplicativo:

Estas instruções visam fornecer ao usuário um passo-a-passo da utilização do aplicativo, desde sua abertura até a reprodução da música.

Se é sua primeira vez utilizando o aplicativo, recomendamos que siga as seguintes instruções:

1) Abra o aplicativo e familiarize-se com a interface. À esquerda encontram-se os bancos de *samples* dos instrumentos, e à direita encontra-se a *timeline* na qual será montada a música.

2) Toque os botões com os nomes dos instrumentos para visualizar seus respectivos bancos de *samples*. Toque os *samples* que se encontram dentro da tabela para ouvir cada um.

3) Toque e segure em um *sample* para entrar em modo de edição. Você será avisado que entrou em modo de edição por uma mensagem e texto e, caso seu dispositivo tenha suporte, uma leve vibração. Arraste o *sample* até uma posição da *timeline* e solte-o. Repita este processo algumas vezes, colocando *samples* diferentes em algumas trilhas e tempos diferentes.

4) Toque a opção de menu "Tocar" para ouvir o resultado de sua música. Depois, acesse o menu do programa e escolha a opção "Volume". Altere os volumes das trilhas e toque novamente a opção "Tocar" para ouvir a diferença.

5) Acesse o menu do programa e escolha a opção "Salvar". Dê um nome à sua música e toque em "Salvar".

6) Acesse novamente o menu do programa e escolha a opção "Abrir". Abra o arquivo de exemplo "pixies" e toque a opção "Tocar". Escolha novamente a opção "Abrir" e abra o arquivo que você salvou.

Quadro 21 - Questionário de usabilidade do aplicativo

QUESTIONÁRIO

Você conseguiu seguir os passos das instruções de uso sem dificuldades?
( ) Sim ( ) Não

Você achou o aplicativo intuitivo e fácil de usar?
( ) Sim ( ) Não

Dentro do escopo limitado do protótipo, você conseguiu criar uma música que considera satisfatória?
( ) Sim ( ) Não

Você conseguiu identificar os diferentes *samples* sem dificuldades?
( ) Sim ( ) Não

Você gostaria que fossem disponibilizados novos *samples* e instrumentos no aplicativo?
( ) Sim ( ) Não

Você gostaria que fosse possível compartilhar uma música com outros usuários através do aplicativo, para criar e editar músicas colaborativamente?
( ) Sim ( ) Não

Você gostaria que o aplicativo armazenasse o histórico de versões de cada música?
( ) Sim ( ) Não

Você gostaria que o aplicativo exportasse as músicas criadas em formatos de áudio (ex. MP3)?
( ) Sim ( ) Não

Você achou prática a representação visual dos *samples*? Se não, como acha que os *samples* deveriam ser representados?

_____
_____
_____
_____
_____

Se tiver alguma observação a fazer quanto ao protótipo (dificuldades, sugestões, etc.), escreva abaixo:

_____
_____
_____
_____
_____

Obrigado por sua participação!