NELTON VINÍCIUS MATIOLI SANTOS

MOBILE EAR TRAINER: SISTEMA MÓVEL DE TREINAMENTO AUDITIVO PARA MÚSICOS

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

LAVRAS
MINAS GERAIS – BRASIL
2007

NELTON VINÍCIUS MATIOLI SANTOS

MOBILE EAR TRAINER: SISTEMA MÓVEL DE TREINAMENTO AUDITIVO PARA MÚSICOS

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Área de Concentração:
Engenharia de Software

Orientador:
Ahmed Ali Abdalla Esmin

LAVRAS
MINAS GERAIS – BRASIL
2007

NELTON VINICIUS MATIOLI SANTOS

MOBILE EAR TRAINER: SISTEMA MÓVEL DE TREINAMENTO AUDITIVO PARA MÚSICOS

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Aprovada em 19 de março de 2007

_____

Prof. Ricardo Martins de Abreu Silva

_____

Prof. Cristiano Leite de Castro

_____

Prof. Ahmed Ali Abdalla Esmin
(Orientador)

LAVRAS
MINAS GERAIS – BRASIL

RESUMO

Este trabalho mostra o *Mobile Ear Trainer* que é um sistema móvel de treinamento auditivo para músicos. Ele possibilita treinar o reconhecimento de intervalos, acordes e escalas. As modalidades de treinamento são: intervalos harmônicos ou melódicos, ascendentes e descendentes. Tríades e tétrades tocadas simultaneamente ou arpejadas. Escalas: maior, menor, menor harmônica, menor melódica e os modos.

Palavras-Chave: Computação Musical, Computação Móvel, Treinamento Auditivo.

ABSTRACT

This work shows the *Mobile Ear Trainer* which is a mobile system for musicians ear training. It allows to exercise intervals, chords and scales recognition. The types of training are: harmonic intervals, ascending and descending melodic intervals. Triads and seventh chords played simultaneously or arpeggiated. Scales: major, minor, harmonic minor, melodic minor and the modes.

Keywords: Musical Computation, Mobile Computation, Ear Training

SUMÁRIO

LISTA DE FIGURAS

1. INTRODUÇÃO

1.1 Contextualização e Motivação

Para um músico é muito importante que ele seja capaz de reconhecer: ritmos, distâncias entre notas, escalas e acordes, através apenas da audição dos mesmos, pois assim ele consegue compreender melhor a música como um todo. Esta capacidade é alcançada através do treinamento auditivo. O músico aprende a reconhecer estes sons através da repetição da audição destes. Além de proporcionar um melhor entendimento da música ao escutá-la, o treinamento auditivo apresenta muitos outros benefícios. Através dele o músico consegue tocar a música que pretende tocar. Consegue ouvir a música “na sua cabeça” enquanto a lê. Consegue improvisar melhor. Desenvolve a habilidade de interagir melhor com outros músicos. Possibilita ao músico arranjar ou compor uma música mais facilmente. Como se pode notar o treinamento auditivo é algo essencial a qualquer músico. Por ser uma tarefa repetitiva, este treinamento já foi implementado em diversos *softwares* feitos para computadores pessoais.

Porém, a sociedade atual passa por uma urgência cada vez maior, o homem sente a necessidade de poder realizar ações com a possibilidade de se locomover, não desperdiçando nenhum momento do seu tempo. Deste modo, os aparelhos móveis vêm se tornando ferramentas indispensáveis ao homem moderno. Principalmente os celulares. Com isso, muitos *softwares* antes projetados apenas para computadores pessoais, hoje estão sendo transportados para *palms*, *pdas* e celulares.

O músico como ser humano inserido nesta sociedade também tem esta necessidade. Portanto, é vantajoso para ele poder exercitar o seu ouvido, algo essencial a sua profissão, a qualquer momento. Não sendo preciso estar à frente de um computador para realizá-lo ou com um instrumento musical em mãos. Visto que grande parte da população usa celular, seria interessante a existência de um *software* para o treinamento auditivo no celular.

1.2 Objetivos e Estrutura do Trabalho

Este trabalho tem por objetivo a implementação de um *software*, para celulares, de treinamento auditivo para músicos. O *software* aborda as partes de reconhecimento de intervalos, tanto melódicos quanto harmônicos; reconhecimento de acordes, tanto as tríades

como as tétrades, tocados simultaneamente ou arpejados e reconhecimento das escalas, maior, menor, menor harmônica, menor melódica e os modos.

O trabalho está estruturado da seguinte forma. O capítulo 2 é o Referencial Teórico, onde serão apresentados aspectos teóricos de textos sobre o treinamento auditivo para músicos, as tecnologias de aprendizagem de música, o desenvolvimento de aplicações móveis, a tecnologia J2ME e a *Mobile Media* API. O capítulo 3 é a metodologia, onde são apresentados o tipo de pesquisa e os procedimentos. O capítulo 4 trata dos resultados e discussão destes, neste capítulo é apresentado uma descrição do sistema desenvolvido e os testes realizados. O capítulo 5 são as considerações finais, onde é apresentado a conclusão e sugestões de trabalhos futuros. E o capítulo 6 é a bibliografia citada.

2. REFERENCIAL TEÓRICO

2.1 Treinamento Auditivo para Músicos

O Treinamento auditivo é que capacita o músico a conseguir compreender melhor a música como um todo. Esta está dividida em três elementos básicos: ritmo, melodia e harmonia. Lacerda (1961), define que ritmo é a maneira como se sucedem os valores na música. A melodia é definida como uma sucessão de sons de alturas e valores diferentes, que obedece a um sentido musical. A harmonia é definida por Sadie (1994) como a combinação de notas soando simultaneamente, para produzir acordes, e sua utilização sucessiva para produzir progressões de acordes.

Através do treinamento o músico pode adquirir o ouvido relativo que Sadie (1994) afirma ser a capacidade de identificar intervalos “de ouvido”, sem entretanto identificar alturas isoladas. Pode-se também desenvolver o ouvido absoluto que seria a capacidade de identificar a altura de uma nota ou de entoá-la sem referência a uma nota anteriormente tocada. É chamado às vezes de “ouvido perfeito”. Este tipo de ouvido é mais difícil de ser desenvolvido.

Um dos modos de se alcançar esta melhor percepção da música é através do ditado musical. Segundo Pozzoli (1977), o ensino do ditado pode ser dividido em três ramos: ditados rítmico, melódico e harmônico. No ditado rítmico o aluno estudará as combinações das durações dos sons (ritmo); no ditado melódico estudará as relações existentes entre os sons sucedendo-se, e no ditado harmônico estudará as relações existentes entre os sons que são produzidos simultaneamente.

Dentro do ditado melódico são feitos exercícios de identificação de intervalos e escalas. Intervalo é a diferença de altura entre dois sons. Os intervalos podem ser classificados em melódico e harmônico. O intervalo melódico é aquele em que as notas soam sucessivamente. O intervalo harmônico é aquele em que as notas soam simultaneamente.

Escala é uma seqüência de notas em ordem de altura ascendente ou descendente. É longa o suficiente para definir sem ambigüidades um modo ou tonalidade, e começa ou termina na nota fundamental daquele modo ou tonalidade. As escalas geralmente utilizadas no ditado são: a maior, menor natural, menor harmônica, menor melódica e os modos: jônio, dórico, frígio, lídio, mixolídio, eólio, lócrio. A figura 2.1 mostra estas escalas e modos.

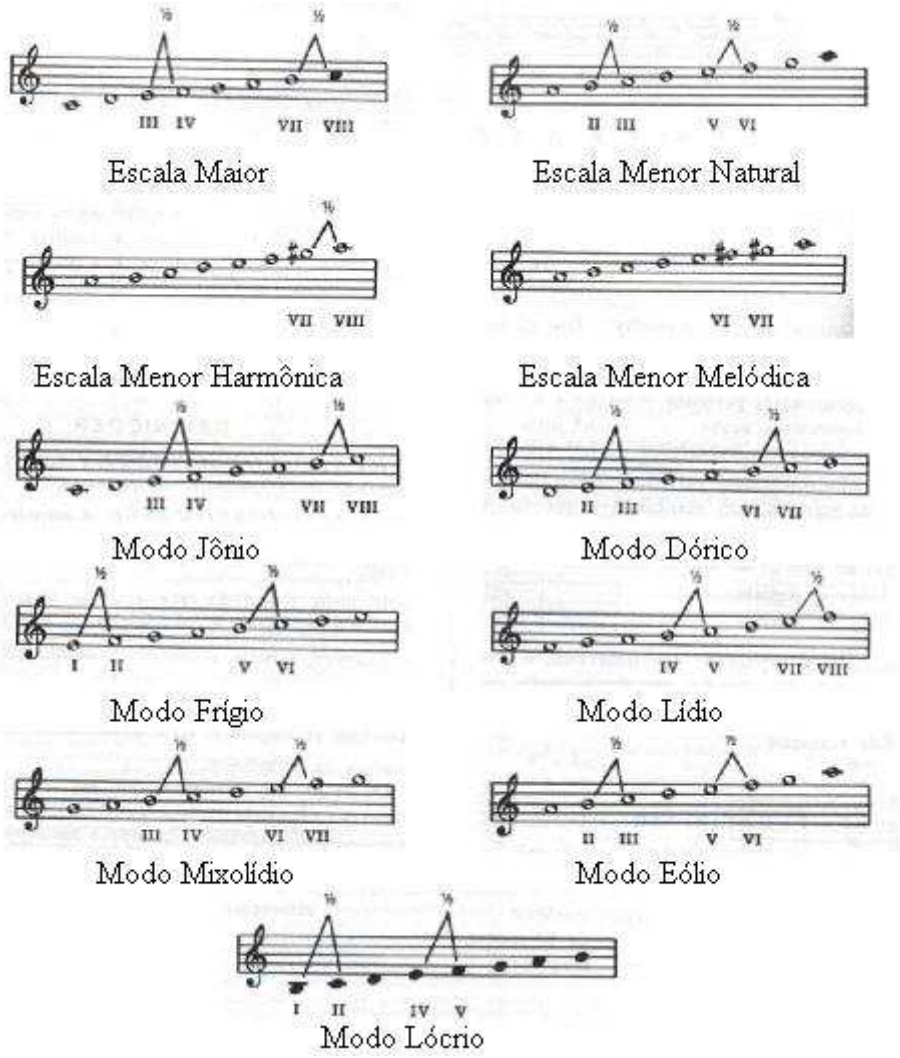

Figura 2.1 Escalas e Modos (Fonte: (Lacerda, 1961))

Dentro do ditado harmônico são feitos exercícios de identificação de tríades, tétrades e progressões harmônicas. Segundo Sadie (1994) tríade é um acorde consistindo de três notas que podem ser organizadas para formarem duas terças sobrepostas. Se a terça inferior é maior e a terça superior menor, diz-se que a tríade é maior (dó-mi-sol); Se a terça inferior é menor e a superior maior, é uma tríade menor (dó-mib-sol). Se ambas as terças são maiores, a tríade é aumentada (dó-mi-sol#); Se ambas as terças são menores, a tríade é diminuta (dó-mib-solb).

Segundo Guest (1996), tétrade é o acorde de quatro notas, separadas por terças. Se tivermos uma tríade maior acrescida de uma terça superior maior, é um acorde maior com sétima maior (dó-mi-sol-si). Se for uma tríade maior acrescida de uma terça superior menor, é um acorde maior com sétima (dó-mi-sol-sib). Se for uma tríade menor acrescida de uma terça superior menor, é um acorde menor com sétima (dó-mib-sol-sib). Se tivermos uma tríade diminuta acrescida de uma terça superior maior, é um acorde menor com sétima

e quinta diminuta (dó-mib-solb-sib). Se tivermos uma tríade diminuta acrescida de uma terça superior menor, é um acorde diminuto (dó-mib-solb-sibb). Se ocorrer uma tríade maior com quinta diminuta acrescida de uma terça superior maior, é um acorde maior com sétima e quinta diminuta (dó-mi-solb-sib). Se for uma tríade aumentada acrescida de uma terça superior diminuta, é um acorde maior com sétima e quinta aumentada (dó-mi-sol#-sib). Sendo uma tríade aumentada acrescida de uma terça superior menor, é um acorde maior com sétima maior e quinta aumentada (dó-mi-sol#-si). E por fim, se é uma tríade menor acrescida de uma terça superior maior, é um acorde menor com sétima maior (dó-mib-sol-si).

2.2 Tecnologia para a Aprendizagem da Música

Ficheman et al. (2003) afirma que a pesquisa de recursos computacionais para educação musical é desafiadora, pois para que possam suprir as necessidades da área precisam ser especificadas questões sobre produção sonora, autenticidade dos timbres, coordenação/simultaneidade entre áudio e seu respectivo texto (em formas de notação musical ou outra) e a velocidade de transmissão das informações para execução e apreciação em tempo real. Enquanto em outras áreas tais elementos podem ser considerados adicionais ou especiais - “efeitos sonoros” ou premiações - em educação musical eles são a matéria prima.

Para Kruger (2000), um *software* educativo musical pode ser considerado de qualidade quando apresenta as seguintes características: Parâmetros pedagógicos; Interações sociais; e Informática & Educação Musical.

De acordo com Hentschke (1993), toda prática educacional está direta ou indiretamente ligada a uma teoria de aprendizagem. Estas teorias fundamentam as concepções da educação musical. Um *software* com que se deseja propiciar experiências no campo da música tem que conter alguma concepção musical.

Para Fritsch et al. (2001) há um certo desconhecimento da fundamentação pedagógico-musical por parte de alguns pesquisadores de *software* educacional para música. Tal situação torna-se aparente quando:

- Os *softwares* de educação musical são desenvolvidos em projetos com ênfase computacional, sem um equilíbrio com os pressupostos educacionais;

• As limitações do *software* de autoria escolhido cerceiam a liberdade criativa quanto à implementação de rotinas computacionais pedagogicamente inovadoras;

• Não são realizadas avaliações sistemáticas constantes durante a implementação. Como resultado, podem ser obtidos *softwares* com alta qualidade em programação visual e sonora, embora sem características pedagógicas que satisfaçam as necessidades de estudantes e professores, e que sejam pertinentes ao contexto sócio-cultural brasileiro.

Conforme Miletto et al. (2004), apesar dos músicos reconhecerem a importância da sua formação através de uma sólida educação musical, surpreendentemente muito pouco *software* é desenvolvido especialmente para isto.

Dentre os *softwares* educacionais para a área musical destacam-se os feitos para treinamento auditivo. Isto se explica pelo fato desta atividade ser repetitiva e como Araújo (2002) afirma, com o uso de *software* educacional, torna-se possível a repetição seguida de uma mesma tarefa e sua efetiva assimilação.

Ainda segundo Araújo (2002), os aplicativos para treinamento auditivo são programas destinados a desenvolver as habilidades auditivas do aluno de música. Em geral, apresentam exercícios divididos em modalidades tais como:

• Intervalos melódicos: o computador toca duas notas em seqüência e o aluno deve indicar a distância intervalar entre as notas;

• Intervalos harmônicos: semelhantes aos intervalos melódicos, porém as notas são tocadas simultaneamente;

• Acordes: o computador toca um acorde e o aluno deve dizer que notas foram tocadas e/ou classificar o acorde e/ou indicar o nome do acorde. Pode, ainda, ser apresentado um acorde arpejado (uma nota de cada vez) ou harmônico (todas as notas de uma só vez);

• Ritmos: o computador toca um ritmo e o aluno deve reproduzi-lo ou escrevê-lo;

• Escalas: é tocada uma escala e o aluno deve dizer qual a escala tocada.

Exemplos destes *softwares* seriam o EarMaster e o Solfege. O EarMaster é um *software* proprietário, que apresenta 12 áreas de exercícios. Comparação de intervalos, identificação de intervalos, cantar intervalos, identificação de acordes, inversão de acordes, progressão de acordes, identificação de escalas, leitura rítmica, imitação rítmica, ditado rítmico, correção rítmica e ditado melódico.

O Solfege é um *software* livre, que roda em plataforma Windows e Linux. O Solfege apresenta os seguintes exercícios: identificação de intervalos, cantar intervalos, identificação de ritmos, cantar acordes, identificação de escalas e identificação de progressões harmônicas.

2.3 Desenvolvimento de Aplicações Móveis

De acordo com Forman e Zahorjan (1994), a popularidade dos dispositivos móveis produz uma demanda de desenvolvimento de aplicações específicas para eles. Dispositivos móveis, porém, têm características diferentes daquelas encontradas em computadores pessoais. A CPU tem um menor poder de processamento. A memória tem menor capacidade. A fonte de energia, em geral baterias ou pilhas, faz com que o uso destes dispositivos seja restrito. A tela é muito menor. Os mecanismos de entrada de dados são diferentes. A conexão de rede tem uma largura de banda reduzida, latência e taxa de erros maiores quando comparados com redes cabeadas. Além disso, as desconexões podem ser freqüentes. Este conjunto de características faz com que o desenvolvimento de aplicações para dispositivos móveis não seja feito da mesma forma e nem com as mesmas ferramentas utilizadas em aplicações para computadores pessoais ou *mainframes*.

Courdec e Kermarrec (1999) afirmam que a construção de aplicações para o ambiente móvel deve levar em conta suas restrições e tentar diminuir seu efeito. Para tal, devem ser projetadas, em todos seus aspectos, com mobilidade e adaptabilidade em mente.

Segundo Topley (2002) existem algumas diretrizes úteis durante o processo de desenvolvimento de aplicações para dispositivos móveis. São elas:

- Ambiente: Fazer uma pesquisa sobre o ambiente em que a aplicação será utilizada, antes de começar o desenvolvimento da aplicação. Deve-se primeiro saber as necessidades dos usuários em potenciais, os requisitos impostos pelas redes e em que plataformas o aplicativo funcionará (telefones celulares, *palmtops*, etc);
- Dividir as Tarefas da Aplicação: É preciso pensar cuidadosamente quanto à decisão de quais operações devem ser feitas no servidor e quais devem ser feitas no dispositivo móvel. MIDlets permitem localizar muitas das funcionalidades do aplicativo no dispositivo, podem recuperar dados do servidor, processá-los e exibi-los na tela do dispositivo localmente. Esta abordagem reduz a interação com a rede sem fio, conseqüentemente diminuindo o tráfego da rede;

• Representação dos dados: Os dados podem ser representados de muitas formas, uns mais compactos do que outros. Deve-se considerar representações disponíveis e escolher aquelas que exigem menos bits para serem transmitidos.

• Latência da Mensagem: Em algumas aplicações, pode ser possível fazer outras tarefas enquanto uma mensagem está sendo processada. Se a demora é considerável, é importante manter o usuário informado do progresso da transmissão.

• Simplicidade da Interface: Manter a interface do aplicativo simples e intuitiva, de forma que o usuário raramente precise consultar o manual do usuário para fazer uma tarefa. É importante reduzir a quantidade de informações exibidas no dispositivo; apresentar as seqüências das entradas do usuário com um número mínimo de acionamento de botão ou teclas; oferecer sempre que possível listas de seleção para o usuário.

Segundo Lee et al (2005) a arquitetura para aplicações móveis utiliza camadas e filas. A divisão em camadas descreve a divisão do código da aplicação em uma única máquina. As camadas são freqüentemente módulos de código colocados em pastas diferentes ou diretórios no cliente ou no servidor.

A camada de código que interage mais perto do usuário é comumente chamada de camada de apresentação. A segunda camada é freqüentemente denominada camada de negócios, pois em geral ele trata da lógica comercial. A terceira camada é freqüentemente denominada camada de acesso a dados. Em geral, ela trata da comunicação com o banco de dados ou com a origem de dados.

As filas descrevem a divisão de trabalho de código da aplicação em diversas máquinas. A segmentação em filas geralmente envolve a colocação de módulos de código em máquinas diferentes num ambiente de servidores distribuídos. Se o código da aplicação já estiver em camadas, o processo de segmentação em filas ficará muito mais simples. As filas descrevem arquiteturas de servidor, e em geral não contamos dispositivos de cliente como uma fila. Embora isso seja possível, não é uma convenção comum.

Basicamente os dispositivos móveis podem ser classificados em dois tipos de clientes: clientes magros e clientes gordos. Os clientes magros não possuem código da aplicação personalizado e dependem completamente do servidor para sua funcionalidade. Portanto, não dependem tanto do sistema operacional como ocorre com os clientes gordos. Estes, em geral possuem de uma a três camadas de código da aplicação e podem operar

independentemente de um servidor por certo período. Clientes gordos geralmente são mais úteis em situações em que não há garantia de comunicação permanente entre o cliente e o servidor. No entanto, os clientes gordos, como foi dito anteriormente, dependem muito do sistema operacional e do dispositivo móvel, por isso pode ser difícil liberar e distribuir o código.

Falando sobre o desenvolvimento de uma aplicação móvel, Lee et al (2005) afirma que qualquer aplicação móvel deve tratar de vários requisitos. Os tipos mais comuns e importantes são os requisitos de usuário, de negócios, funcionais, de operação e de sistema.

As aplicações móveis devem ser projetadas como qualquer outra aplicação corporativa complexa. Deve-se desenvolver uma arquitetura do projeto, um conjunto de casos de uso, fluxogramas, modelos de projeto, modelos de dados e todos outros artefatos geralmente associados com o projeto técnico. É extremamente necessário escolher tecnologias e linguagens antes de os requisitos e o projeto terem sido completados.

2.4 A Tecnologia J2ME

Segundo Amorim (2005), a plataforma J2ME (*Java 2 Platform, Micro Edition*) é a edição da linguagem Java que foi projetada para dispositivos com memória, vídeo e poder de processamento limitados, variando desde máquinas ligadas à TV até telefones celulares. Antes do surgimento da tecnologia J2ME as aplicações tinham que ser escritas na linguagem nativa de cada dispositivo usando bibliotecas proprietárias, o que as tornavam incompatíveis com dispositivos diferentes.

Conforme Leal (2004) e Soares (2004) a tecnologia J2ME proporciona o melhor custo /benefício na medida que:

- Profissionais da linguagem Java podem ser aproveitados;
- Amplamente adotada pelos fabricantes e operadoras de telefonia móvel;
- Compatibilidade de plataformas: aplicativos Java são escritos uma vez e rodam em diferentes dispositivos com sistemas operacionais diferentes;

Segundo a Sun (2006), a arquitetura J2ME é modular e escalável. Esta modularidade e escalabilidade é definida pela tecnologia J2ME em um modelo de três camadas embutidas sobre o sistema operacional do dispositivo: a camada de perfil (*Profile*), a camada de configuração (*Configuration*) e a camada representada pela máquina virtual *Java* (*Java Virtual Machine)*, juntamente com o sistema operacional presente no

dispositivo (*Host Operation System*). A aplicação encontra-se logo acima da camada de perfil. Estas camadas podem ser observadas na figura 2.2.

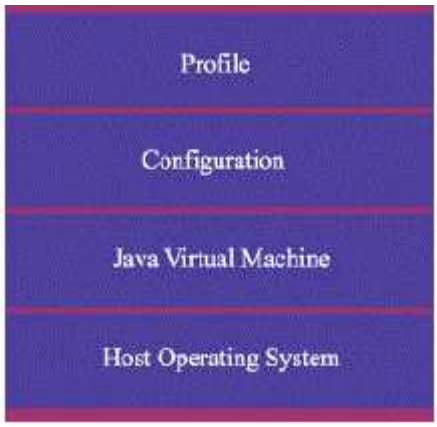

Figura 2.2 Camadas da Arquitetura J2ME (Fonte: (Muchow, 2004))

De acordo com Miranda (2002), uma configuração é uma JVM (*Java Virtual Machine)* e API (*Application Programming Interface*) que contém os requisitos mínimos de funcionamento (a API pode ser um subconjunto de J2SE), destinada a uma larga faixa de dispositivos, com capacidade de processamento e memória semelhantes.

Segundo Muchow (2004), as configurações estão divididas em:

• *Connected Device Configuration* (CDC): conjunto de APIs para dispositivos "fixos", como um computador ligado à televisão;

• *Connected Limited Device Configuration* (CLDC): conjunto de APIs para dispositivos com poder de processamento, vídeo e memória limitados, geralmente móveis.

O CDC contém o CLDC. O CDC possui todas as classes definidas pelo CLDC, incluindo quaisquer novas não inclusas no J2SE (Java 2 *Standard Edition*).

Segundo Pereira (2006), a CDC é focada em dispositivos com maior poder de processamento, mais recursos de rede, mais memória e interface com o usuário mais sofisticada. *Gateways* residenciais, *settop-boxes* de tv digital etc. Fazem parte desta configuração. CDC especifica como máquina virtual a JVM, a máquina virtual utilizada na versão padrão do Java, J2SE.

Corbera (2003) afirma que para a configuração CLDC os aparelhos devem apresentar as seguintes características:

• *Hardware*: o dispositivo deve ter no mínimo 160Kb para Java, um processador de no mínimo 16 bits com baixo consumo (adequado a baterias típicas de um celular) e conexão de rede (neste caso *wireless* – 9.6Kbps, 144Kbps ou 2Mbps);

• *Software*: o *software* deve incluir suporte a um subconjunto da linguagem Java e a um subconjunto da máquina virtual Java que definam um núcleo de funções que permitam o desenvolvimento de aplicações móveis.

Segundo Depiné (2002), existem algumas limitações na configuração CLDC em relação à edição J2SE. As principais são:

• Não há suporte para números de ponto flutuante, portanto os tipos *float* e *double* não podem ser usados. Isso ocorre principalmente por causa do *hardware* que não suporta, pois operações com ponto flutuante requerem um maior processamento;

• Sem suporte a finalização (object.finalize());

• Suporte limitado a erros. Existem apenas três classes de erros: java.lang.Error, java.lang.OutOfMemoryError, e java.lang.VirtualMachineError. Erros de *Runtime* são manipulados de maneira dependente de implementação, que pode terminar a aplicação ou efetuar um *reset* no dispositivo. Exceções de aplicação são utilizadas normalmente, como no J2SE.

• Não há suporte a Java *Native Interface* (JNI), por questões de segurança e performance.

Muchow (2004) afirma que a K *Virtual Machine* (KVM) é a implementação da Sun de uma máquina virtual Java que se encaixa nas especificações do CLDC. A KVM foi projetada para ser tão menor e eficiente quanto fosse possível. Segundo Kuhnen (2003), a KVM foi desenhada para uso de memória estática entre 40 a 80Kb. A quantidade mínima de memória requerida pela KVM é 128Kb, incluindo o interpretador, um mínimo de bibliotecas especificadas pela configuração e algum espaço para rodar aplicativos.

De acordo com Paludo (2003), mesmo com a divisão do J2ME entre as configurações, a variedade entre dispositivos ainda é muito grande. Para resolver isso a Sun criou extensões das configurações chamadas *Profiles*. Os perfis (*Profiles*) definem um conjunto de APIs de alto nível, que quando combinadas com a API básica das configurações, fornecem um ambiente Java completo, contendo modelo de ciclo de vida da aplicação, interface com o usuário, acesso à propriedades do dispositivo, etc. Alguns dos perfis existentes são: *Mobile Information Device Profile* (MIDP), RMI *Profile*, *Foundation Profile* (FP), *Information Module Profile* (IMP), *Personal Profile* (PP), *Personal Basis Profile* (PBP), etc. A seguir são explanados alguns deles, com enfoque maior ao perfil MIDP por ser o mais utilizado atualmente e o mais aceito no mercado.

Conforme Pereira (2006), o *Foundation Profile* (FP) é o nível mais baixo de perfil do CDC. Ele fornece uma implementação de rede do CDC que pode ser usada para construir aplicações sem interface com o usuário. Ele pode também ser combinado com o PBP e o PP para dispositivos que necessitem de uma interface com o usuário.

O *Information Module Profile* (IMP) quando utilizado com a CLDC fornece um ambiente para sistemas embarcados conectados que não possuem rica capacidade gráfica, ou que seus recursos são limitados de alguma outra forma. Caixas de chamada de emergência, parquímetros, módulos sem fio em sistemas de alarme residenciais, são exemplos de dispositivos atendidos por este perfil.

O *Personal Profile* (PP) é o perfil CDC utilizado em dispositivos que necessitam de um suporte completo para interface, como *PDAs* e consoles para jogos. Ele inclui a biblioteca AWT completa e é fiel ao ambiente *web*.

O *Personal Basis Profile* (PBP) é uma divisão do PP que fornece um ambiente para dispositivos conectados que suportem um nível básico de apresentação gráfica ou necessitem do uso de *toolkits* específicos para aplicações. Ambos, PP e PBP, são as camadas superiores do CDC e FP.

Com relação ao *Mobile Information Device Profile* (MIDP), Depiné (2002) afirma que o perfil é voltado especificamente aos dispositivos portáteis. Este perfil especifica a interface com o usuário, entrada e persistência de dados, manipulação de eventos, modelo de armazenamento orientado a registro, rede, mensagens e aspectos relacionados ao ambiente de programação para esses dispositivos.

Conforme Muchow (2004), um dispositivo que deseja implementar o MIDP deve seguir os seguintes requisitos. Requisitos de *hardware*:

- A tela deve suportar pelo menos 96 x 54 *pixels*.
- Deve haver pelo menos um tipo de entrada de dados para o usuário interagir: teclado para uma mão (teclado do telefone), teclado para duas mãos (típico teclado QWERTY) ou tela sensível ao toque.
- 128 *kilobytes* de memória não volátil para rodar os componentes do *Mobile Information Device* (MID).
- Pelo menos 8 *kilobytes* de memória não volátil para que as aplicações armazenem dados persistentes, como configurações da aplicação e dados.
- 32 *kilobytes* de memória volátil para rodar Java.
- Conectividade a rede *Wireless*.

Os requisitos de *software* são:

• O sistema operacional rodando no dispositivo deve prover escalonamento mínimo, tratamento de exceções e processamento de interrupções. Deve haver também capacidades suficientes para rodar uma JVM.

• O *software* deve dar suporte a escrita de gráficos *bitmapped* na tela.

• Usando qualquer um dos três tipos de entrada de dados descritos anteriormente, o *software* deve aceitar a entrada e passar a informação para a JVM.

• Para dar suporte a dados persistentes, deve haver capacidade de ler e escrever da memória não volátil e para ela. Não há requisitos definidos para o sistema de arquivos.

A figura 2.3 mostra a arquitetura do dispositivo móvel de informação (MID). Onde temos a parte do *hardware* (*Mobile Information Device*), logo acima, rodando sobre o *hardware*, temos o sistema operacional (*Native Operating System*). Do lado direito, acima, temos as aplicações nativas do dispositivo (*Native Applications*). Um exemplo destes aplicativos seria os programas de configuração que vêm instalados nos dispositivos. O CLDC é instalado no sistema operacional e é a base para o MIDP. Existem as classes específicas providas pelo fabricante do equipamento (OEM-*Specific Classes*) e as aplicações específicas do fabricante. Estas podem acessar APIs do MIDP e/ou as classes específicas do fabricante.

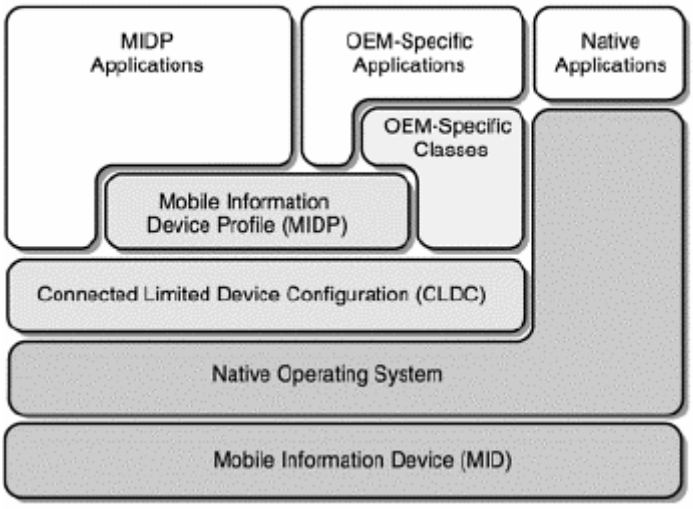

Figura 2.3 Arquitetura do MID (Fonte: (Muchow, 2004))

Além das configurações e perfis o J2ME possui APIs opcionais. Segundo Almeida (2004), as APIs opcionais são funcionalidades adicionais específicas que não serão encontradas em todos os dispositivos de uma determinada configuração ou perfil, mas

importantes o suficiente para serem padronizadas. Exemplos de APIs opcionais são o WMA (*Wireless Messaging* API), que adiciona controle de mídia aos programas J2ME nos dispositivos que o permitem, e o MMAPI (*Mobile Media* API), que permite manipular funcionalidades multimídia dos dispositivos. Esta API é explicada com mais detalhes na seção 2.5.

Juntos, configurações, perfis e APIs opcionais formam uma pilha que define o que está disponível para o desenvolvedor em um dado dispositivo ou classe de dispositivos, fornecendo o escopo de recursos que podem ser acessados pelos aplicativos escritos para eles.

Para Almeida (2004), um aplicativo MIDP é chamado de midlet, por herdar de uma classe denominada MIDlet (javax.microedition.midlet.MIDlet). Com essa herança, a classe recebe a capacidade de se comunicar com o Gerenciador de Aplicativos que os dispositivos MIDP possuem e de interagir com a interface do aparelho.

Um aplicativo MIDP é um conjunto de classes que implementam seu funcionamento, reunidas em um pacote chamado MIDlet Suíte. O aplicativo é desenvolvido em um computador *desktop*, compilado, pré-verificado, empacotado (JAR) e então instalado em um dispositivo para execução ou em um emulador. O arquivo JAR que contém o aplicativo é acompanhado por um descritor da aplicação, um arquivo JAD (*Java Application Descriptor*), que é processado pelo *Application Manager* (gerenciador de dispositivos) do dispositivo, fornecendo informações sobre os requisitos de funcionamento da MIDlet Suíte e permitindo que os usuários decidam pela sua instalação.

2.5 Mobile Media API

Segundo Mahmoud (2005), a *Mobile Media* API (MMAPI) é um pacote opcional dentro do J2ME, que provê uma API padrão para renderização e captura de mídias baseadas em tempo, como trilhas de áudio e vídeo clipes. Desenvolvida dentro da Java *Community process* como JSR 135, MMAPI foi desenvolvida para ser flexível e independente de plataformas; ela não faz nenhuma suposição sobre os formatos de mídia, protocolos, ou características suportadas por vários dispositivos.

A MMAPI possui quatro componentes principais: *Player*, *Manager*, *DataSource* e *Control*. A figura 2.4 mostra a interação entre esses quatro componentes.

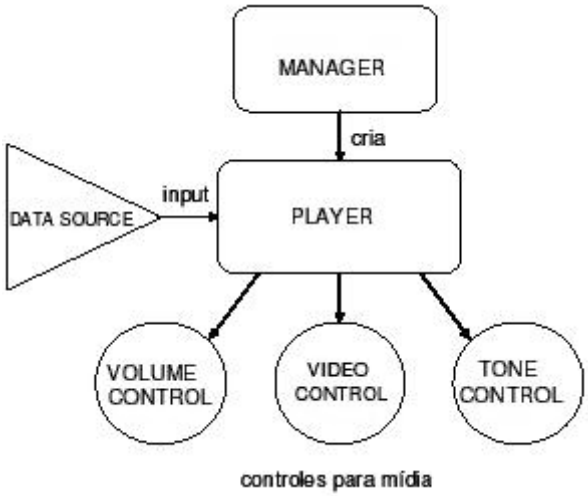

Figura 2.4 Interação entre os componentes do MMAPI (Fonte: (Clemente, 2006))

Conforme Clemente (2006), o processamento de mídias pode ser dividido em duas partes principais: a aquisição e entrega das mídias e no processamento das mídias para a apresentação. Correspondendo a estas partes, existem as duas classes principais da MMAPI que são a *DataSource* e o *Player*, respectivamente.

Sendo assim, a classe *DataSource* é responsável por capturar os dados da mídia, seja através de um sistema de arquivos, de um protocolo padrão, ou de algum mecanismo proprietário. Desta forma, toda a complexidade de acesso à mídia fica encapsulada nesta classe. Através de métodos específicos, o *DataSource* permite que o *Player* possa pedir o acesso a mídia.

O *DataSource* é uma classe abstrata, ou seja, precisa ser estendida e ter certos métodos obrigatoriamente implementados. Isto possibilita ao desenvolvedor criar seus próprios mecanismos de acesso à mídia. Contudo, na prática, o desenvolvimento de *DataSources*, por ser complexo, é realizado apenas pelos fabricantes de celular.

Com a mídia acessível, cabe ao *Player* processá-la, decodificá-la e colocá-la em uma saída. Esta classe ainda atua sobre as classes de controle de mídia, como vídeo e volume, e ainda disponibiliza métodos para o controle direto da mídia, como *play* e *stop*.

Os métodos de controle do *Player* o fazem navegar entre seus estados. Na Figura 2.5 tem-se esta representação. Ainda segundo Clemente (2006), no estado *Unrealized* o objeto *Player* foi instanciado na memória, mas ainda não foi alocado nenhum recurso. No *Realized*, o *Player* fez a requisição do recurso de mídia e já possui a mídia acessível, através de um protocolo ou do sistema de arquivo. No estado *Prefetched*, ele fez requisição de outros recursos, como controle de vídeo e áudio do dispositivo, e a mídia já está na memória, pronta para ser consumida. Quando o *Player* já começou a tocar a mídia ele está

no estado *Started* e quando ele liberou os recursos e não pode ser mais tocado, está no estado *Closed*.

A classe *Manager* é responsável por criar um *Player* através de um *DataSource*. A classe *Manager* é uma *factory* estática, ou seja, uma classe que cria outras classes e é única durante a execução. Para simplificar o desenvolvimento, a classe *Manager* também possibilita criar *Players* através de URLs, caminho local, ou *InputStreams*, ao invés de passar um *DataSource*. Por fim, as classes de controle são classes que podem ser requeridas pelo *Player* para executar funções.

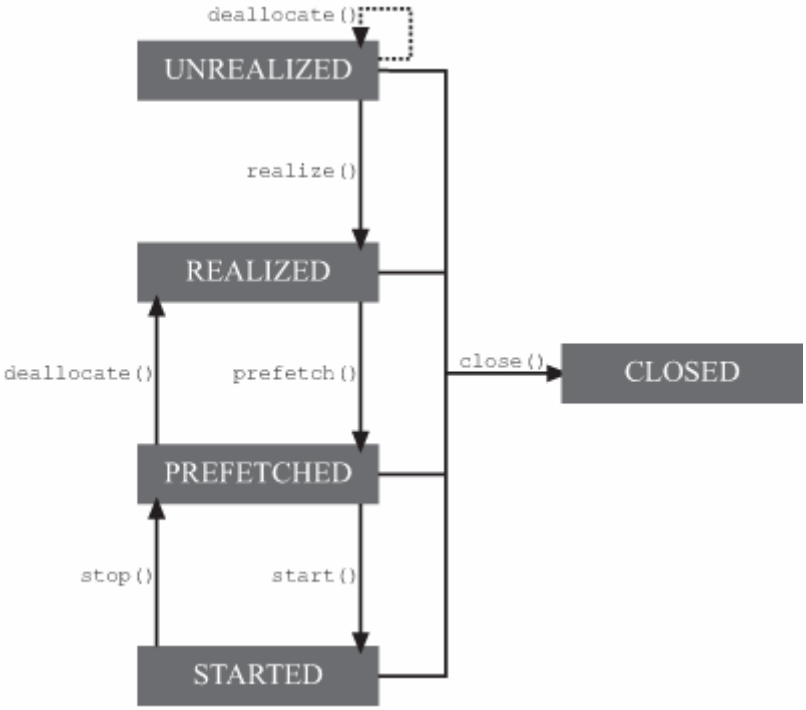

Figura 2.5 Estados de um *Player* (Fonte: (Cardoso, 2006))

3. METODOLOGIA

3.1 Tipo de Pesquisa

Segundo Jung (2004) e observando o método científico tem-se que este projeto é de natureza aplicada, com objetivos de caráter exploratório, utilizando procedimentos experimentais.

Este projeto é de natureza aplicada pois usa os conhecimentos básicos de desenvolvimento de aplicações móveis e conhecimentos básicos de treinamento auditivo para músicos para a geração de um novo produto que seria o sistema móvel de treinamento auditivo para músicos.

Os objetivos são de caráter exploratório pois pretende-se obter um novo produto de *software*. Que seria caracterizado como uma inovação tecnológica.

Os procedimentos utilizados são experimentais pois foi construído um protótipo de *software*.

3.2 Procedimentos

Para a concretização deste trabalho, primeiramente foi feito um levantamento dos *softwares* já existentes para treinamento auditivo, e verificação de suas diferenças e semelhanças. Deste modo foi possível conhecer o estado da arte.

Concorrentemente, foi realizado um estudo da tecnologia Java usada para desenvolvimento de aplicações móveis, J2ME. Esta tecnologia foi usada para a implementação do *software*.

Após esta fase inicial de aquisição de conhecimento, deu-se inicio a fase de desenvolvimento. Foi feita uma modelagem do sistema, utilizando a UML, antes de começar a implementação.

Para implementação, foi utilizado um Pentium III, de 1 GHz, com 256 mega bytes de memória RAM, com sistema operacional Windows XP. A ferramenta utilizada para o auxílio ao desenvolvimento foi o Netbeans 5.5 equipado com o pacote Netbeans Mobility 5.5 para dar suporte ao desenvolvimento utilizando o J2ME. Foi utilizado o Java 2 SDK, (*Standard Edition*, versão 1.4.2). A ferramenta que foi utilizada para a realização de testes

preliminares foi o J2ME Wireless Toolkit 2.2. Após a verificação de que o *software* estava funcionando neste ambiente, foram realizados testes nos emuladores Motorola Java ME SDK 6.3, Sony Ericsson SDK 2.2.4 e Nokia Series 40 SDK 3rd Edition.

4. RESULTADOS E DISCUSSÃO

4.1 Considerações Iniciais

Como resultado da pesquisa realizada foi desenvolvido o *Mobile Ear Trainer:* Sistema móvel de treinamento auditivo para músicos. Nas seções a seguir serão mostradas as características do *software* produzido, a modelagem UML (*Unified Modeling Language*) do sistema. A seguir, vem a descrição da implementação, onde são mostradas as decisões tomadas, as classes e como é o fluxo do sistema. Por fim, são descritos os testes realizados.

4.2 Características

Para que fosse possível reproduzir sons MIDI (*Musical Instrument Digital Interface*) foi necessário utilizar a API do J2ME chamada MMAPI (*Mobile Media API*). Foi utilizada a versão 1.1 da MMAPI. Pelo fato desta ser uma API opcional do J2ME, é necessário que o dispositivo tenha implementado o método *createPlayer(DataSource source).* Caso contrário, não é possível criar um *player* para tocar as notas MIDI.

Além disso, para que um aparelho celular rode o sistema corretamente deve possuir suporte a configuração CLDC 1.1 e ao perfil MIDP 2.0. O MIDlet tem o tamanho 56105 *bytes*. Portanto o aparelho deve possuir este espaço livre em memória para que a aplicação seja instalada.

4.3 Modelagem UML

A figura 4.1 apresenta o diagrama de casos de uso do sistema. O usuário poderá treinar intervalos, escalas, tríades e tétrades. Sendo que podem ser treinados os intervalos harmônicos, melódicos ascendentes ou melódicos descendentes. As escalas ascendentes. As tríades e tétrades tocadas simultaneamente ou em arpejos ascendentes e descendentes.

O diagrama de classes é apresentado na figura 4.2 da seção 4.4.2.

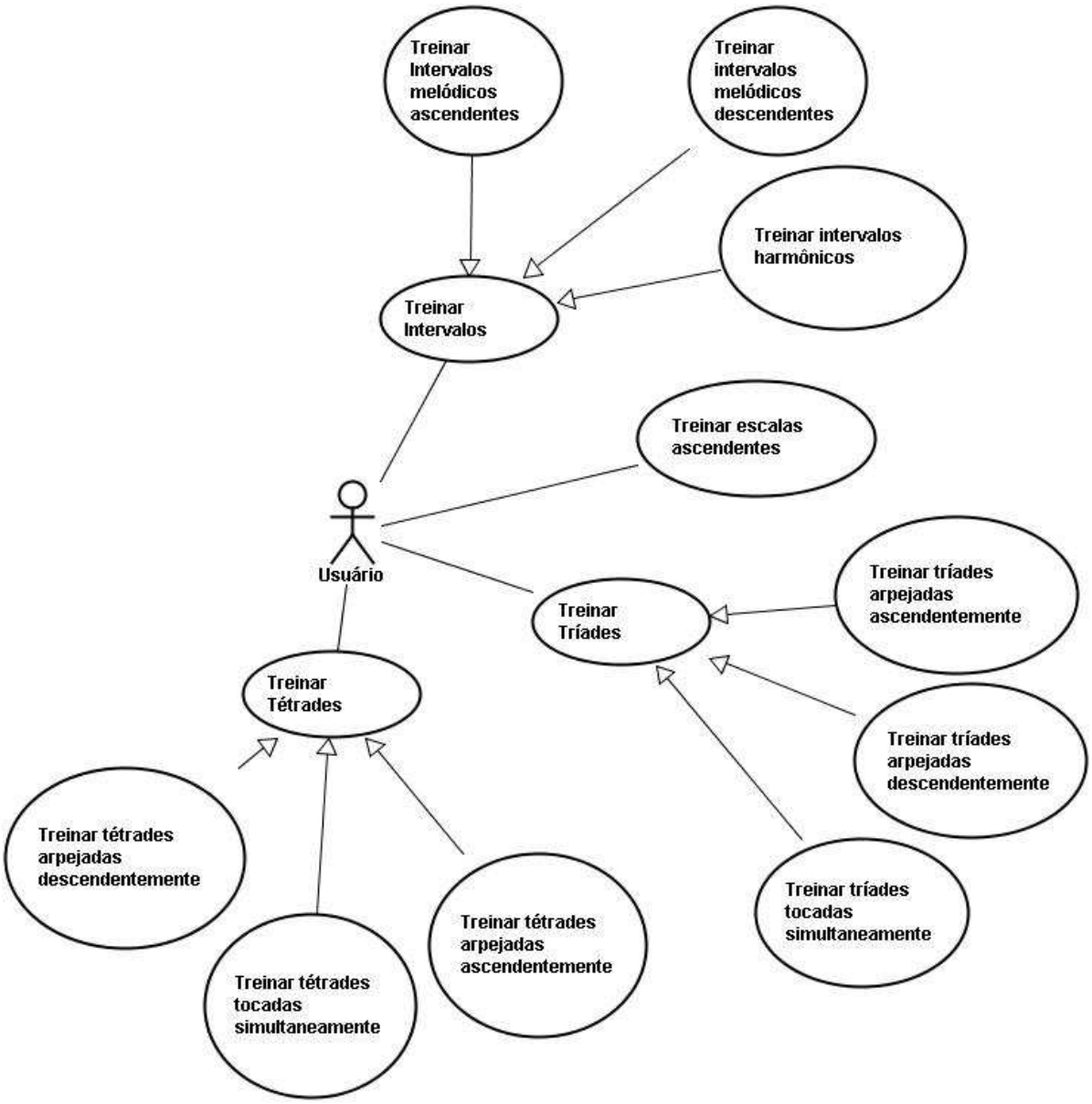

Figura 4.1 Diagrama de casos de uso do sistema

4.4 Descrição da Implementação

4.4.1 Decisões Tomadas

O sistema deve gerar notas MIDI iniciais de modo aleatório, para que a partir desta se forme os intervalos, escalas e acordes. Foi escolhido então qual o intervalo que seriam geradas as notas. A nota mais grave é a nota MIDI 40 que corresponde ao E3 no piano e a mais aguda é a nota 76 que corresponde ao E5.

O piano foi escolhido para ser simulado via MIDI, por ser o mais usado em treinamentos. O fato de usar este instrumento não interfere no resultado final do treinamento. Podendo ser usado seja qual for o instrumento tocado pelo músico.

Para que fosse possível reproduzir sons polifônicos, para executar os intervalos harmônicos e os acordes tocados simultaneamente, foi utilizada a classe *MidiControl* do MMAPI. Já para tocar escalas, intervalos melódicos e acordes arpejados, a princípio estava sendo adicionado a um *player* o controle *ToneControl*, podendo assim ser programado a seqüência de tons a ser tocada. Mas a qualidade de reprodução do som que estava sendo alcançada não era satisfatória pois o som estava sendo reproduzido no formato *tone sequence*. Então, para aproveitar a qualidade do formato MIDI, foi utilizado o *MidiControl* também. Mas para que fosse possível reproduzir uma nota após a outra foi necessário parar a execução da *thread* por determinado tempo através do método *sleep* da classe *Thread*. O tempo em que a *thread* fica parada é o tempo especificado no andamento. Como o andamento é especificado em batidas por minuto (bpm) e o método recebe como parâmetro a quantidade de milisegundos que a *thread* deve parar, foi necessário realizar a conversão de bpm para milisegundos.

4.4.2 Classes do Sistema

Segundo Kamra et al (2004), na arquitetura 3-camadas têm-se a interface, a lógica da aplicação e os dados mantidos em camadas diferentes. Como o *software* não necessita de armazenamento de dados, ele foi desenvolvido utilizando a arquitetura 2-camadas. Possuindo apenas a camada de interface e a camada da lógica da aplicação.

A figura 4.2 apresenta o diagrama de classes do sistema. A responsável pela camada de interface é a classe *MainEar.java*. É nela onde é implementada toda a parte de telas e tratamento das opções escolhidas pelo usuário. A partir dela é que são chamados os métodos responsáveis pela execução dos intervalos, acordes e escalas. O fluxo de telas é descrito com detalhes na seção 4.4.3

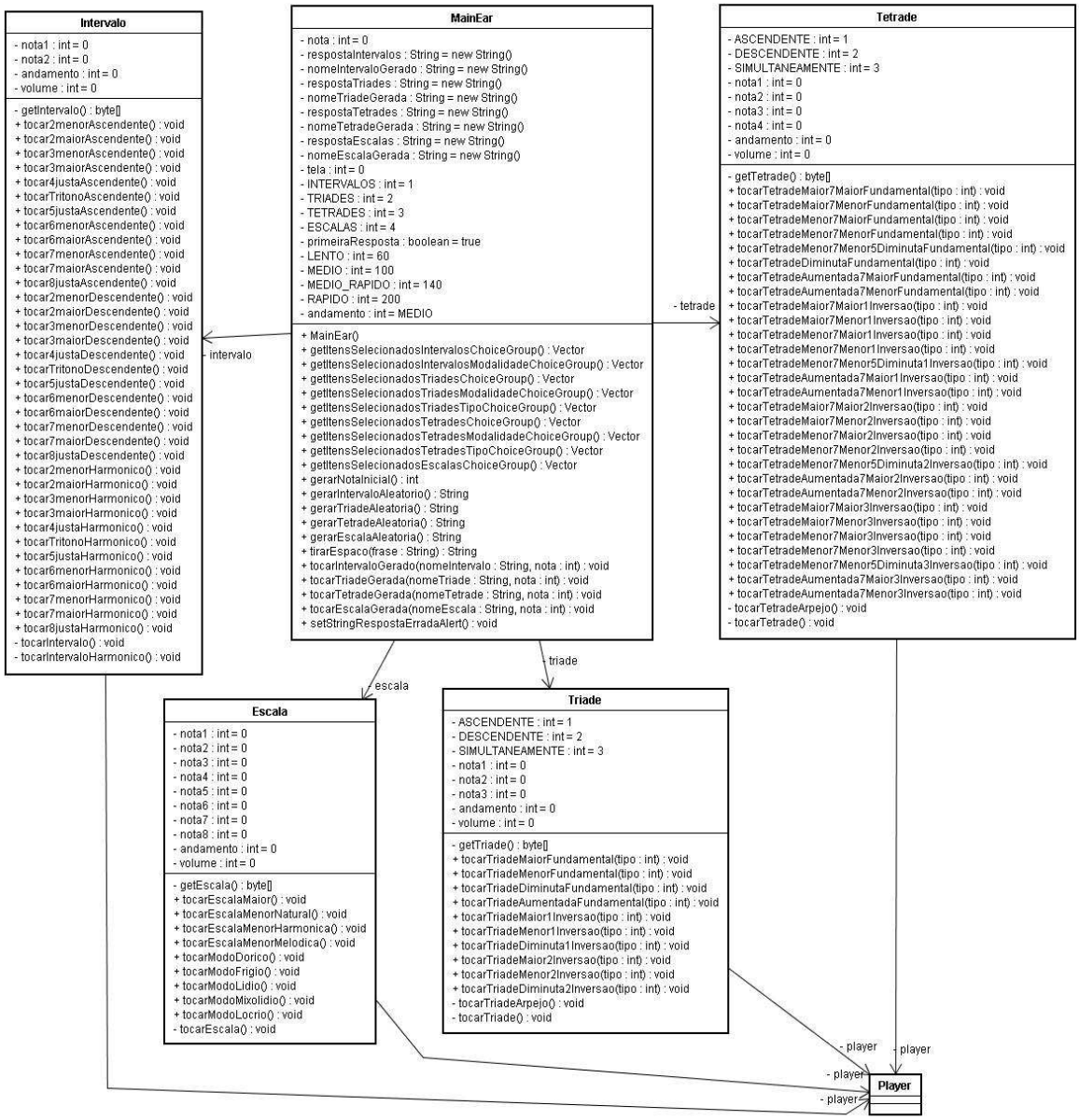

Figura 4.2 Diagrama de classes

As classes *Escala.java*, *Intervalo.java*, *Triade.java* e *Tetrade.java* são responsáveis pela lógica da aplicação. Todas estas classes possuem os atributos andamento e volume com que serão tocadas. Além disso, possuem um *player* responsável pela reprodução do áudio. O *player* é uma instância da classe *Player* que é responsável pela reprodução dos sons.

A classe *Escala.java* possui como atributos sete notas que formam as escalas. Possui métodos para definir cada uma das sete notas de acordo com a escala e um método para que a escala seja tocada.

A classe *Intervalo.java* possui duas notas como atributos e métodos para definir todos os intervalos melódico ascendente, melódico descendente e harmônico. Além dos métodos responsáveis por tocar os intervalos melódicos e harmônicos.

As classes *Triade.java* e *Tetrade.java* possuem, respectivamente, como atributos três e quatro notas e métodos para definir as tríades e tétrades. Possui também os métodos para tocar os acordes de forma simultânea ou arpejada.

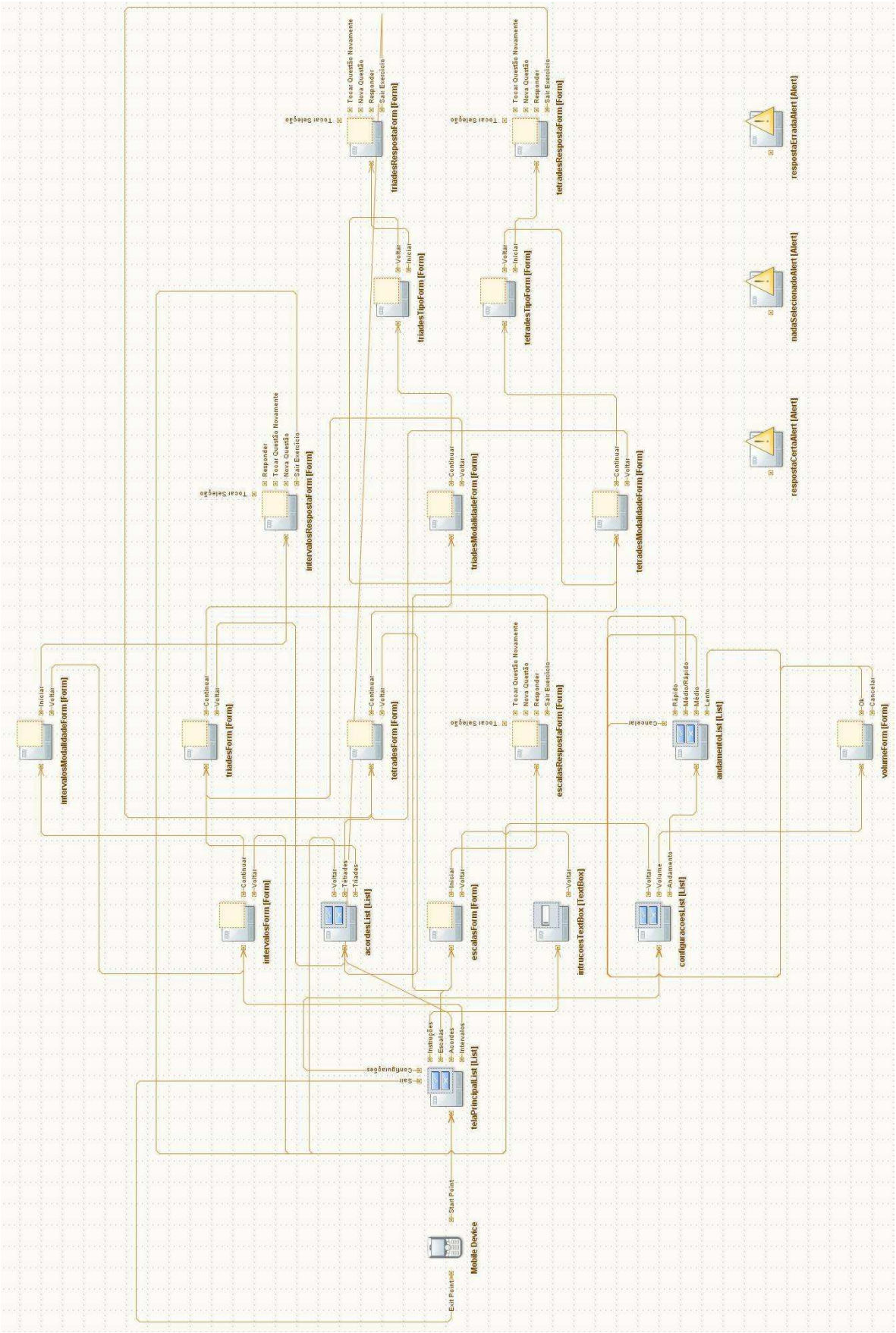

Figura 4.3 Fluxo do sistema

4.4.3 Fluxo do Sistema

O objetivo desta seção é explicar o funcionamento do sistema e apresentar a sua interface. A figura 4.3 mostra como é o fluxo do sistema. Assim que o usuário iniciar a aplicação aparecerá a tela *telaPrincipalList* (Figura 4.4). Então ele poderá usar o menu e escolher entre as opções (*treinar intervalos, acordes ou escalas*) ou então escolher a opção *alterar as configurações*. Há ainda um item com instruções sobre como utilizar o sistema e uma opção para sair.

Figura 4.4 Tela *telaPrincipalList*

Escolhendo a opção intervalos, o usuário vai para a tela *intervalosForm*. Nesta, ele poderá escolher quais intervalos deseja treinar. A figura 4.5 mostra esta tela com alguns intervalos selecionados. Após escolher os intervalos desejados e em seguida a opção de continuar, a tela *intervalosModalidadeForm* (Figura 4.6) vai aparecer. Nesta tela, o usuário pode escolher se os intervalos vão ser tocados harmonicamente ou melodicamente. Na figura 4.6 esta tela é mostrada com todas as opções escolhidas.

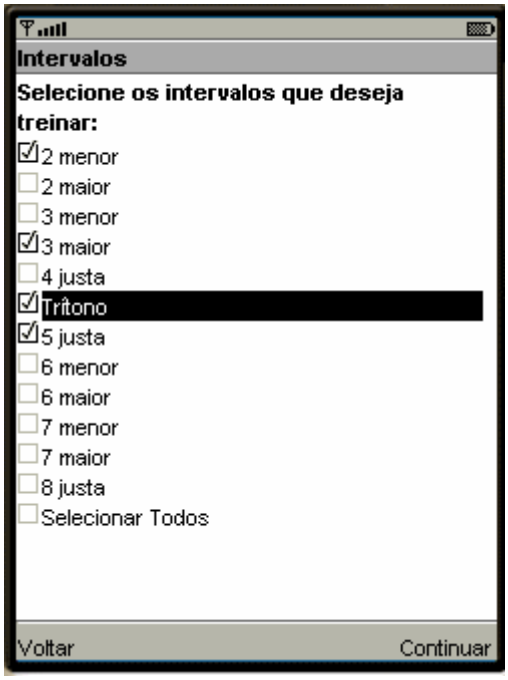

Figura 4.5 Tela *intervalosForm*

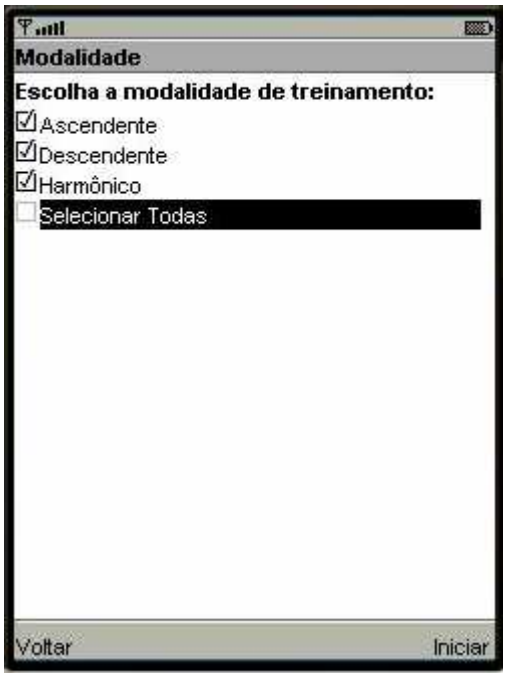

Figura 4.6 Tela *intervalosModalidadeForm*

Se o usuário escolheu, no menu principal, treinar acordes, ele vai para a tela *acordesList* (Figura 4.7). Nesta, poderá escolher entre treinar tríades ou tétrades. Então, irá para a tela *triadesForm* (Figura 4.8) ou *tetradesForm* (Figura 4.9) dependendo da opção escolhida. Nestas telas, respectivamente, ele escolherá quais tríades e tétrades deseja treinar. Depois irá, respectivamente, para as telas *triadesModalidadeForm* (Figura 4.10) e *tetradesModalidadeForm* (Figura 4.11). Nestas telas pode-se escolher quais inversões dos

acordes deseja treinar. Prosseguindo, vai respectivamente para as telas *triadesTipoForm* e *tetradesTipoForm*, onde será feita a escolha de como os acordes serão tocados, simultaneamente ou arpejados. A figura 4.12 mostra a tela *triadesTipoForm*. A *tetradesTipoForm* possui estrutura semelhante.

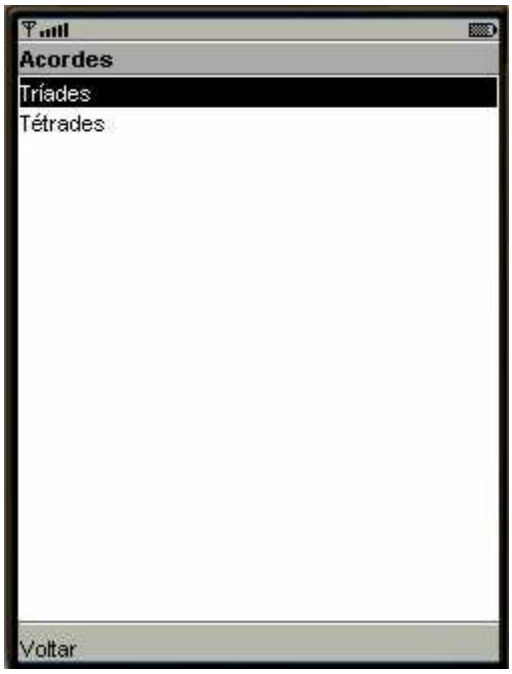

Figura 4.7 Tela *acordesList*

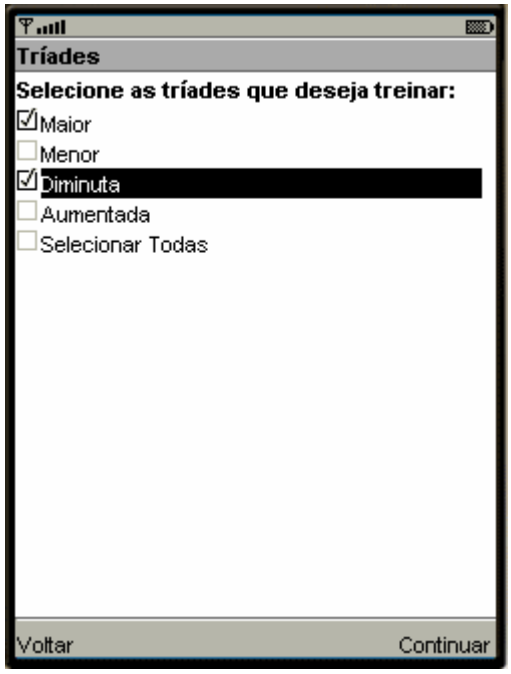

Figura 4.8 Tela *triadesForm*

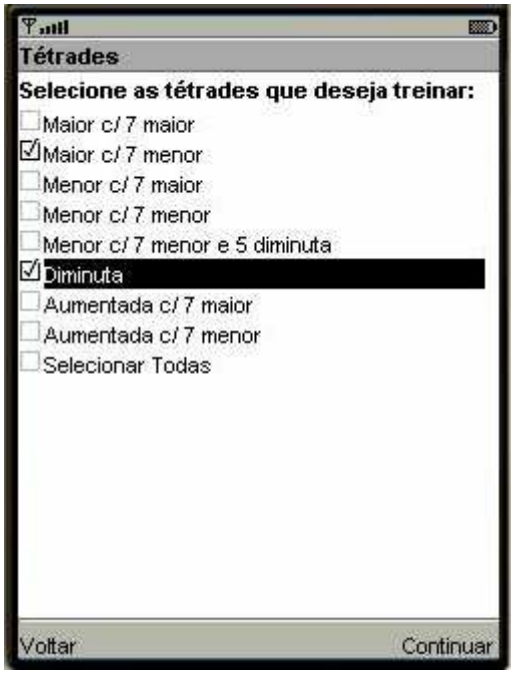

Figura 4.9 Tela *tetradesForm*

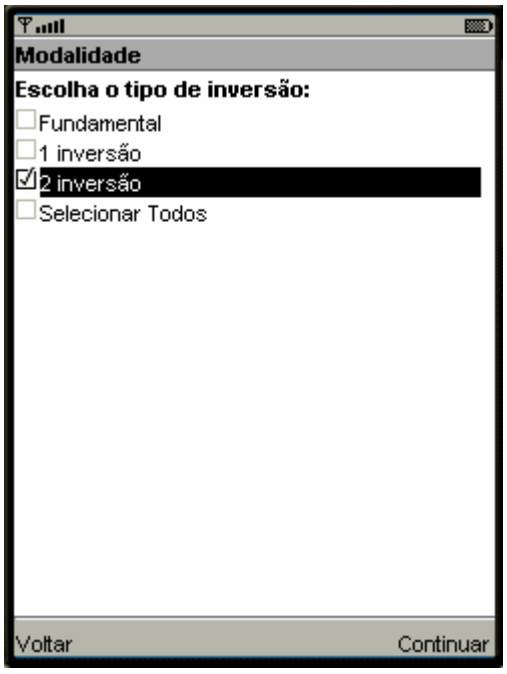

Figura 4.10 Tela *triadesModalidadeForm*

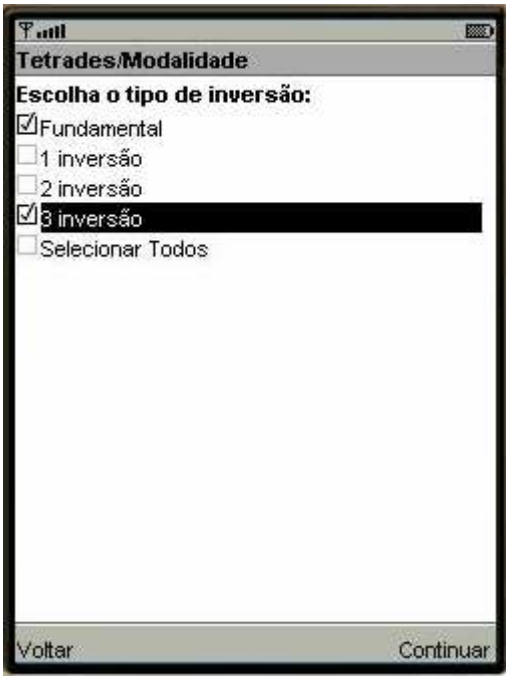

Figura 4.11 Tela *tetradesModalidadeForm*

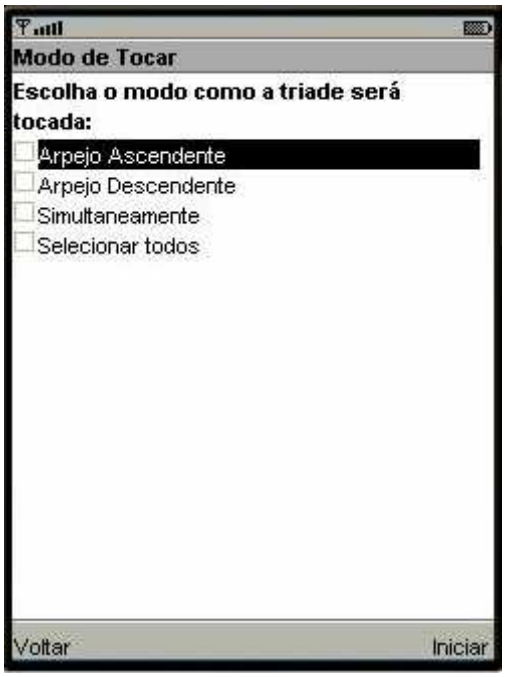

Figura 4.12 Tela *triadesTipoForm*

Escolhendo a opção escalas no menu principal, o usuário irá para a tela *escalasForm*. Nesta, escolherá quais escalas deseja treinar. A figura 4.13 mostra esta tela com algumas opções selecionadas.

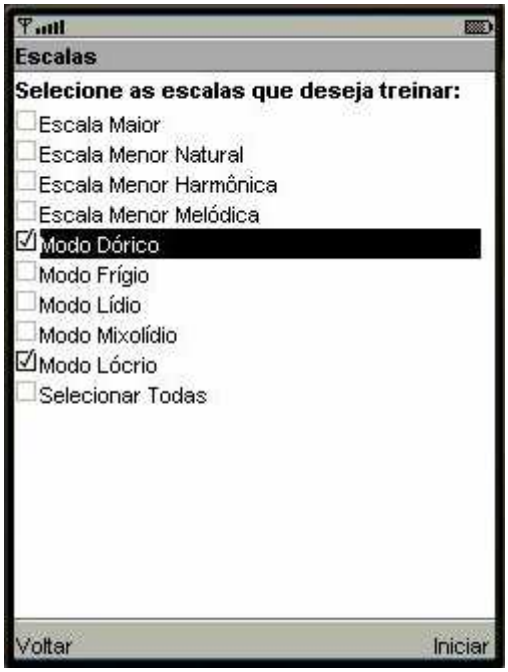

Figura 4.13 Tela *EscalasForm*

Depois de todas opções escolhidas, dá-se início ao exercício. As telas *intervalosRespostaForm*, *triadesRespostaForm*, *tetradesRespostaForm* e *escalasRespostaForm* possuem a mesma estrutura. Nelas, aparecerão as opções de resposta de acordo com as opções escolhidas anteriormente. Então, será tocado o intervalo, acorde ou escala, e o usuário tentará identificá-lo selecionando uma opção. Indo ao menu nesta tela, terá as opções: responder, nova questão, tocar questão novamente e tocar seleção. A figura 4.14 mostra como exemplo a tela *triadesRespostaForm*, quando o botão menu foi pressionado.

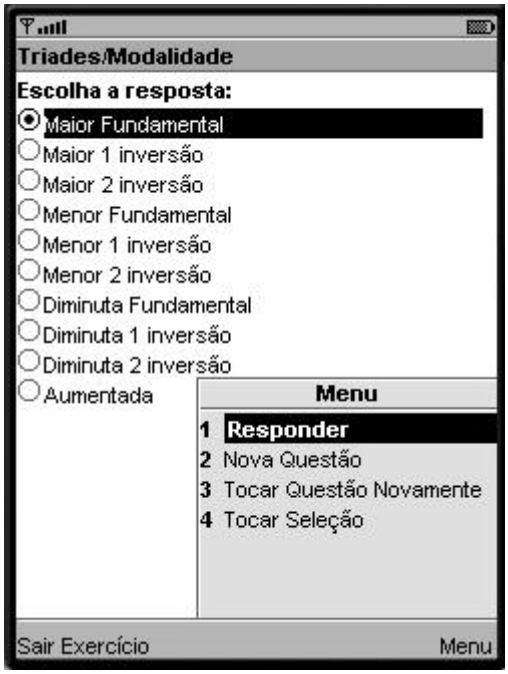

Figura 4.14 Tela *triadesRespostaForm*

Ao ir em responder, será verificado se a opção escolhida é a resposta certa. Se o usuário acertou a questão, aparecerá a tela *respostaCertaAlert* (Figura 4.15) que lhe informa isto. Caso contrário, aparecerá a tela *respostaErradaAlert* informando que o usuário errou a questão e diz qual era a resposta certa para que ele possa comparar a opção escolhida e a opção certa se desejar. O usuário só poderá tentar responder uma vez cada questão. A figura 4.16 mostra a tela *respostaErradaAlert* informando ao usuário, que estava treinando tríades, qual era a resposta certa.

Figura 4.15 Tela *respostaCertaAlert*

Figura 4.16 Tela *respostaErradaAlert*

Quando escolher a opção nova questão no menu, será gerada aleatoriamente uma nova uma questão. Pra escutar novamente a questão basta selecionar opção tocar questão novamente. E por fim, ao escolher a opção tocar seleção, escutará a opção de intervalo, acorde ou escala que está selecionada.

Ao entrar em configurações no menu principal, o usuário poderá configurar o volume que será reproduzido o áudio e configurar o andamento, ou seja, velocidade com

que os sons serão tocados. As opções de andamentos são lento, médio, médio/rápido e rápido. Escolhendo o andamento lento as questões serão tocadas na velocidade de 60 bpm, escolhendo médio, tocadas a 100 bpm, o médio/rápido equivale a 140 bpm e o rápido a 200 bpm. A figura 4.17 mostra a tela *andamentoList* onde é feita a configuração do andamento.

Figura 4.17 Tela *andamentoList*

Nas telas *intervalosForm*, *intervalosModalidadeForm*, *triadesForm*, *triadesModalidadeForm*, *triadesTipoForm*, *tetradesForm*, *tetradesModalidadeForm*, *tetradesTipoForm* e *escalasForm* o usuário deve escolher pelo menos uma opção, caso não o fizer, aparecerá a tela de erro *nadaSelecionadoAlert* (Figura 4.18). Esta tela fica visível por 1500 milisegundos. Passado este tempo, o programa retorna a tela em que o usuário estava.

Figura 4.18 Tela *nadaSelecionadoAlert*

4.5 Testes

Durante o desenvolvimento foi utilizado o emulador J2ME Wireless Toolkit 2.2 para testes e correções de erros. Quando o *software* já estava funcionando perfeitamente neste ambiente, foram realizados testes com emuladores fornecidos por alguns fabricantes de celulares, como Sony Ericsson, Motorola e Nokia.

Para testar nos modelos da Sony Ericsson foi utilizado o emulador Sony Ericsson SDK 2.2.4. Segundo o manual do *software*, este é uma versão modificada do J2ME Wireless Toolkit 2.2. Deste modo, o aplicativo funcionou corretamente em todos os modelos MIDP 2.0.

O emulador da Motorola usado foi o Motorola Java ME SDK 6.3. A aplicação foi simulada em vários modelos do fabricante que apresentam MIDP 2.0 e CLDC 1.1. Durante os testes, foi constatado que em alguns modelos a aplicação não conseguia acessar o dispositivo MIDI. Deste modo, o sistema não consegue criar o *player*, gerando uma exceção. Mas em certos modelos a aplicação funcionou do modo esperado. Entre os dispositivos testados estes funcionaram: V1100, MOTORAZR V3xx, MOTORAZR maxx V6, V980, V975, RAZR V3x, E770, E1070, E1000, E1000R, C980, C975.

No emulador Series 40 SDK 3rd Edition da Nokia o *software* apresentou o mesmo problema encontrado em alguns modelos do emulador da Motorola, o sistema não conseguiu criar o *player*.

Segundo Goyal (2005), o perfil MIDP 2.0 contém um subconjunto da MMAPI 1.1. O subconjunto apenas suporta tons e áudio com dois controles para cada, *ToneControl* e *VolumeControl.* Além disso, *datasources* não são suportados. Deste modo, a classe *Manager* no MIDP 2.0 é simplificada e não provê o método *createPlayer(DataSource source).* Este método foi utilizado na aplicação para a reprodução de sons MIDI, portanto as falhas encontradas ao reproduzir o som podem ser explicadas devido a este fato.

5.CONSIDERAÇÕES FINAIS

5.1 Conclusão

Graças à evolução dos aparelhos celulares hoje é possível realizar funções antes só possíveis em computadores. Principalmente ao que se refere ao uso de dados multimídia. No caso deste trabalho, foi feito o uso dos recursos de áudio disponíveis para criar uma aplicação de utilidade não apenas para músicos como também pessoas interessadas em compreender melhor as músicas que ouvem.

Sendo assim, este trabalho apresentou o *Mobile Ear Trainer* que é um sistema móvel de treinamento auditivo para músicos. Como tal possui as funcionalidades básicas de treinamento de intervalos, tríades, tétrades e escalas.

O sistema foi implementado com sucesso, apresentando todos os requisitos. Ele foi testado nos emuladores fornecidos pela Sun, Sony Ericsson, Motorola e Nokia. Nos emuladores da Sun, Sony Ericsson e no emulador da Motorola nos modelos V1100, MOTORAZR V3xx, MOTORAZR maxx V6, V980, V975, RAZR V3x, E770, E1070, E1000, E1000R, C980, C975, o sistema funcionou normalmente.

Porém em alguns modelos da Motorola e no emulador da Nokia, o sistema não conseguiu reproduzir os sons, devido ao fato do método *createPlayer(DataSource source)* da MMAPI não estar implementado nesses modelos. Infelizmente não foi possível realizar testes com aparelhos reais para verificar se o sistema funciona normalmente como nos emuladores.

Portanto, apesar do J2ME ser uma linguagem portável e os celulares atuais darem suporte a linguagem Java, quando se usa APIs opcionais como a MMAPI ainda não se encontra uma uniformidade nas implementações, ficando a caráter do fabricante implementar ou não métodos específicos previstos na especificação do MMAPI.

5.2 Trabalhos Futuros

Como trabalhos futuros, pode-se adicionar novos tipos de treinamento ao *Mobile Ear Trainer*. Dentre estas novas funcionalidades poderia ser acrescentado o treinamento da percepção rítmica, onde seriam tocados algumas figuras rítmicas e o usuário deveria escrevê-las. Outra funcionalidade interessante de ser acrescentada seria o treinamento da

identificação progressões de acordes, onde seriam tocadas seqüências de acordes e o usuário as identificaria. Seria bom que se pudesse especificar novas progressões também.

Além disso, pode-se criar um modo tutor onde o sistema irá propondo em qual ordem os exercícios devem ser treinados e decidindo se o usuário deve prosseguir para a próxima fase ou não.

Também seria possível criar uma base de dados de onde os usuários poderiam baixar novos exercícios e também poderiam enviar exercícios criados por eles, tornando-os disponíveis para que outros usuários possam baixar e treinar.

6. BIBLIOGRAFIA CITADA

ALMEIDA, L. B. de. **Introdução à J2ME e Programação MIDP**. *Mundo Java*, Curitiba, n. 5, 2004.

AMORIM, A. R. de; **Desenvolvimento de aplicações móveis com J2ME.** In:Seminário de Informática- Jornada Integrada de Trabalhos de Conclusão de Curso em Computação,Universidade Luterana do Brasil, 2005.

ARAÚJO, C. R.; **Uma Proposta de Ferramenta de Apoio à Educação Musical via Web Usando Java e XML.** Dissertação de Mestrado, Universidade Estadual de Campinas, 2002.

CARDOSO, J; **Java para Telemóveis** , 2006. Disponível em: <http://livromidp.jorgecardoso.org/html/chap11.html>. Acesso em: 23/02/2007.

CLEMENTE, R. G.; **Uma solução de streaming de vídeo para celulares: conceitos, protocolos e aplicativo**. Trabalho de Conclusão de Curso (Graduação em Engenharia Eletrônica e de Computação) - Universidade Federal do Rio de Janeiro, 2006.

CORBERA, R. G.; **Tutorial de programação J2ME**. ,2003. Disponível em: <http://geocities.yahoo.com.br/brasilwireless/tutorial_j2me/j2me_01/j2me_01.html>. Acesso em: 30/09/2006.

COURDEC, P.; KERMARREC, A. M**. ;Improving Level of Service for Mobile User Using Contex-Awareness.** Proceedings of 18th Symposium on Reliable Distributed System. Lausanne, Switzerland, 1999.

DEPINÉ, F. M.; **Protótipo de software para dispositivos móveis utilizando Java ME para cálculo de regularidade em rally**. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Universidade Regional de Blumenau, 2002.

FICHEMAN, I. K.; LIPAS, R. A.; KRUGER, S. E.; LOPES, R. D. ; **Editor Musical: uma Aplicação para a Aprendizagem de Música apoiada por Meios Eletrônicos Interativos.** XIV Simpósio Brasileiro de Informática na Educação – NCE – IM/UFRJ: Rio de Janeiro, 2003.

FORMAN, G. H. ; ZAHORJAN, J.; **The challenges of mobile computing**, IEEE Computer, 27, pp. 38–47, 1994

FRITSCH, E. F.; VICCARI, R. M.; KRÜGER, S. E.; **Avaliação Pedagógica do Software STR**. Revista Brasileira de Informática na Educação. Florianópolis: v.8, p.21 - 33, 2001.

GOYAL, V; **J2ME Tutorial: Multimedia and MIDP 2.0**., 2005. Disponível em: <http://today.java.net/pub/a/today/2005/09/27/j2me4.html>. Acesso em: 23/02/2007.

GUEST, I. **Arranjo: Método Prático.** Rio de Janeiro/RJ: Lumiar Editora, 1996.

HENTSCHKE, L.; **Relações da prática com a teoria na Educação Musical**. In: II Encontro Anual da ABEM. Porto Alegre, 1993.

JUNG, C. F. **Metodologia para pesquisa & desenvolvimento: aplicada a novas tecnologias, produtos e processos.** Rio de Janeiro/RJ: Axcel Books do Brasil Editora, 2004.

KAMRA, A.; MISRA, V.; NAHUM, E. **Controlling the Performance of 3-Tiered Web sites: Modeling, Design and Implementation**, In: ACM SIGMETRICS Performance Evaluation Review, Proceedings of the joint international conference on Measurement and modeling of computer systems. v.32, p. 414 415. ACM Press, 2004

KRÜGER, E. S.; **Desenvolvimento, testagem e proposta de um roteiro para avaliação de software para educação Musical**. Dissertação (Mestrado em Música) - Universidade Federal do Rio Grande do sul, 2000.

KUHNEN, A.; **Protótipo de uma Aplicação LBS Utilizando GPS Conectado em Celular para Consultar Dados Georeferenciados.** Trabalho de Conclusão de Curso (Ciência da Computação), Universidade Regional de Blumenau, 2003.

LACERDA, O. **Compêndio de teoria elementar da música.** São Paulo/SP: Ricordi Brasileira S.A., 1961.

LEAL, M. **Começando com Java Wireless.** Java Magazine, Rio de Janeiro, nº 18, p. 16-19, 2004.

LEE, V.; SCHNEIDER, H.; SCHELL, R. **Aplicações Móveis: arquitetura, projeto e desenvolvimento.** São Paulo/SP: Pearson Education do Brasil, 2005.

MAHMOUD, Q. H.; **Comparing J2ME multimedia options**, 2005. Disponível em: <http://developers.sun.com/techtopics/mobility/apis/articles/mmapioptions/>. Acesso em: 23/02/2007.

MILETTO, E. M.; COSTALONGA, L. L.; FLORES, L. V.; FRITSCH, E. F.; PIMENTA, M. S.; VICARI, R. M. **Educação Musical Auxiliada por Computador: Algumas Considerações e Experiências.** RENOTE - Revista Novas Tecnologias na Educação, Porto Alegre, RS, v. 2, 08 mar. 2004.

MIRANDA, C. **Multimídia e Jogos no Java Micro Edition.** Java Magazine, Curitiba, nº 1, 2002.

MUCHOW, J. W. **Core J2ME – Tecnologia & MIDP.** São Paulo/SP: Pearson Makron Books, 2004.

PALUDO, L.; **Um Estudo Sobre as Tecnologias Java de Desenvolvimento de Aplicações Móveis.** Trabalho de Conclusão de Curso (Ciência da Computação), Universidade Federal de Santa Catarina, 2003.

PEREIRA, D. A.; **Jogos Ubíquos com Bluetooth.** Trabalho de Conclusão de Curso (Ciência da Computação), Universidade Federal de Pernambuco, 2006.

POZZOLI, H. **Guia teórico-prático de ditado musical.** Milão/Itália: G. Ricordi & C., 1977.

SADIE, S. **Dicionário grove de música.** Rio de Janeiro/RJ: Jorge Zahar Editor, 1994.

SOARES, M. **A Tecnologia Java como Diferencial no Mundo dos Celulares.** Mundo Java, Curitiba, nº 5, p. 53-54, maio. 2004.

SUN. **Java 2 plataform,** micro edition. 2006. Disponível em: <http://java.sun.com/j2me>. Acesso em: 01/10/2006

TOPLEY, K.; **J2ME in a Nutshell** .Estados Unidos: O'Reilly & Associates, 2002.