UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

CARLOS KUHN

Desenvolvendo Aplicativos Musicais em PDAs: Fundamentos e Experimentação Prática

Trabalho de Graduação.

Prof. Dr. Marcelo Soares Pimenta
Orientador

Me. Luciano Vargas Flores
Co-orientador

Porto Alegre, junho de 2009.

AGRADECIMENTOS

Gostaria de agradecer aos meus orientadores, Marcelo Pimenta e Luciano Flores, pelo apoio, confiança, inspiração e excelência no trabalho.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

BPM	Batidas Por Minuto, *Beats Per Minute*
GUI	*Graphical User Interface*
HCI	*Human-Computer Interaction*
IDE	*Integrated Development Environment*
IHC	Interação Humano-Computador
PDA	*Personal Digital Assistant*
XML	*Extensible Markup Language*

LISTA DE FIGURAS

RESUMO

Este trabalho tem por objetivo investigar as possibilidades de desenvolvimento de aplicações musicais em dispositivos móveis, em particular PDAs. Para isto, é apresentada uma revisão de conceitos fundamentais tanto de computação musical quanto de desenvolvimento de aplicações em dispositivos móveis. Em seguida, um framework concebido especificamente para facilitar este desenvolvimento é proposto, e os principais aspectos de sua definição e implementação são apresentados.

Palavras-Chave: Computação Musical, Framework, Dispositivos Móveis, .NET.

Developing Musical Applications on PDA: Foundations and Practical Experimentation

ABSTRACT

This work aims at investigating the possibilities for the development of musical applications on mobile devices, particularly PDAs. In order to achieve that, a review of the fundamental concepts related to mobile device's musical applications development is presented. Then, a framework conceived specifically to help this development is proposed, and the main aspects of its definition and implementation are presented.

Keywords: Computer Music, Framework, Mobile Devices, .NET.

1 INTRODUÇÃO

1.1 Motivação e Delimitação do Escopo

A área da computação musical já vem pesquisando o uso dos computadores na música há várias décadas. Como resultado já existe uma vasta gama de soluções conhecidas para apoiar tarefas musicais, principalmente para as plataformas *desktop*. Por outro lado, os paradigmas de computação *desktop* e móvel possuem cada um suas peculiaridades e muitas diferenças entre si. Portanto é desejável, ao desenvolver aplicativos musicais para dispositivos móveis, que não se faça simplesmente uma transposição dos aplicativos *desktop* para a plataforma móvel, mas sim que se leve em conta características específicas dessa nova plataforma e que se desenvolva aplicativos apropriados a este novo suporte.

Com a evolução tecnológica dos dispositivos móveis, como telefones celulares e PDAs, surge a possibilidade de desenvolvimento de aplicações musicais para essa plataforma.

Computadores de mão (ou PDAs – *Personal Digital Assistants*) se diferenciam do paradigma *desktop* ou *laptop*, pois seu objetivo é cumprir (ou às vezes complementar) apenas algumas funções ou aplicações dos computadores de mesa, e atendê-las sendo mais portáteis (cabendo no bolso e na palma da mão), móveis e de mais rápido acesso, o que tem muita relação com o tipo de aplicações que executam – agendas, anotações, calcular, tocar músicas, etc. A idéia é levar um "pedaço" do seu computador de mesa consigo, principalmente para aquelas funções necessárias quando se está "em trânsito". Junta-se a isso, funcionalidades de comunicação e de acesso rápido a informações em redes (Internet, LAN, PAN – troca de informação entre dispositivos portáteis em redes *ad hoc*, etc.).

Para atender às funções específicas a que se destinam, e à necessidade de serem portáteis, os PDAs possuem recursos próprios de hardware e software, diferentes dos computadores de mesa, principalmente em sua interface com o usuário: telas menores, de cristal líquido, geralmente sensíveis a toques (forma principal de interação); ausência de teclado alfanumérico e de mouse; poucos botões, com funções específicas, e em geral um conjunto de quatro botões direcionais; e alguns modelos podem incluir microfone, alto-falante e câmera fotográfica digital. Obviamente essa interface diferenciada implica em uma forma de interação usuário-sistema também diferente.

O aluno autor deste trabalho possui experiência em desenvolvimento para PDAs, especificamente em .NET Compact Framework, para Pocket PCs com Windows Mobile. Também tem interesses musicais e em aperfeiçoar seus conhecimentos de programação para PDAs nos aspectos de processamento de multimídia e da interação

com o usuário. O trabalho foi realizado junto ao grupo de pesquisa em computação musical (www.inf.ufrgs.br/lcm) e poderá ser integrado com outros trabalhos deste grupo nos temas de música em dispositivos móveis e de composição musical colaborativa em rede.

Este trabalho de graduação iniciou com o objetivo de se desenvolver um aplicativo musical rítmico para PDAs, incluindo o estudo de como resolver este problema numa plataforma com menos recursos que a *desktop*. Seriam identificadas quais são essas restrições e propostas alternativas para contorná-las na programação desse sistema para manipulação de som e música.

Durante a investigação das possibilidades, foi identificada como solução para facilitar o desenvolvimento de aplicativos musicais em dispositivos móveis a criação de um framework que já lidasse com as restrições dos dispositivos móveis e implementasse métodos para tais tipos de aplicativos. Assim, o programador ao usar o framework pode se concentrar no funcionamento geral da aplicação e na criação e experimentação de novas interfaces com o usuário.

Como o objetivo do trabalho passou a ser o desenvolvimento de um framework de aplicação, para fins de delimitação de escopo, devido às restrições de prazo para sua realização, este trabalho não vai tratar sobre:

- As alternativas de design de interfaces musicais em PDAs, e as implicações do uso do *touchscreen* ou outras modalidades de interação nessas interfaces;
- As implicações dessas interfaces fornecerem maior ou menor possibilidade de configuração para o usuário final;
- As possibilidades em termos de aplicações musicais genuinamente móveis, que aproveitem as características de mobilidade e conectividade dos PDAs.

No estudo dos aspectos supracitados, obviamente, poderá ser empregado o framework proposto. Mas salientamos que o presente trabalho tratará apenas da *definição do framework* e de um *estudo de caso* por meio de uma implementação desse framework em Windows Mobile e de experimentos na criação de diferentes interfaces para controle musical rítmico em PDAs usando essa implementação do framework.

1.2 Objetivos

Existem atualmente no mercado dezenas ou até centenas de modelos de PDAs de diferentes tipos, funcionalidades e fabricantes e a cada ano surgem mais e mais, mesmo dentro de uma família de plataforma como .NET, Android, etc. Esse é o principal problema encontrado nesta área: é necessário desenvolver sempre um novo aplicativo, basicamente partindo do zero, para projetar e implementar programas com funcionalidades relativas a atividades interativas musicais. A partir desta necessidade este trabalho pretende agilizar e facilitar a construção de tais sistemas com a criação de um framework.

Os quatro benefícios principais de nosso framework são:

a) Não gastar tempo desnecessário modelando e implementando elementos já existentes;

b) Convergir todos os esforços para modelar e implementar o que é específico de cada novo projeto;

c) Utilizando o framework em diferentes projetos, propicia-se que os desenvolvedores partam da mesma base já conhecida e amplamente testada; e

d) Por possuir uma arquitetura modular, permite que se use componentes de diferentes desenvolvedores ao mesmo tempo em uma aplicação.

Este trabalho tem por objetivo investigar as possibilidades de desenvolvimento de aplicações musicais em dispositivos móveis, em particular PDAs. O resultado principal do presente trabalho é a definição de um framework para aplicativos musicais em dispositivos móveis que possibilite uma rápida implementação, facilitando assim a experimentação de novas interfaces para tais aplicativos.

1.3 Estrutura do Trabalho

Após esta introdução, o capítulo 2 apresenta uma revisão de conceitos fundamentais tanto de computação musical quanto de desenvolvimento de aplicações em dispositivos móveis e também de frameworks. Além disto, resume características dos principais trabalhos relacionados de forma a permitir compreender a motivação, as decisões tomadas e o desenvolvimento deste trabalho. O capítulo 3 descreve o framework concebido especificamente para facilitar o desenvolvimento de aplicações musicais em dispositivos móveis, e os principais aspectos de sua definição e implementação são apresentados. No capítulo 4 a implementação é detalhada com as características de uma implementação possível, criada em .NET para Windows Mobile 5, a plataforma básica existente em uma série de PDAs (Pocket-PCs), em particular do PDA disponível para testes do presente trabalho. Ainda nesse capítulo exercitamos o framework ao criar novas interfaces-base usando a implementação em Windows Mobile. Finalmente, o capítulo 5 apresenta as conclusões.

2 DESENVOLVENDO APLICATIVOS MUSICAIS EM DISPOSITIVOS MÓVEIS: CONCEITOS, PRÁTICAS E TRABALHOS RELACIONADOS

Através dos séculos, os músicos sempre tiveram interesse nos mais recentes avanços tecnológicos de suas épocas, e os experimentaram na produção de material musical. Por conseguinte, desde os meados do século 20 a tecnologia de computação também veio sendo cada vez mais explorada por pesquisadores em música e compositores, para a produção e análise musicais. Seguindo nessa linha, mais recentemente o uso de tecnologias de comunicação e computação em rede na computação musical despontou como um foco de pesquisas natural (a área de *Networked Music* – ver BARBOSA, 2003; ORGANISED SOUND, 2005; WEINBERG, 2002).

A combinação da tecnologia móvel com a música é mais uma promessa de desenvolvimentos instigantes em uma área rapidamente emergente. Dispositivos como telefones celulares, *walkmans* e tocadores de MP3 já fornecem música independente de localização e de situação para seus usuários, modificando a forma como estes experienciam a paisagem urbana. Incorporando novos recursos como conectividade *ad hoc*, acesso à Internet e sensibilidade ao contexto, a tecnologia musical nos dispositivos móveis oferece inúmeras novas oportunidades sócio-culturais, comerciais e artísticas para a criação, apreciação e compartilhamento de música (MOBILEMUSICWORKSHOP.ORG, 2009).

No campo da computação musical este assunto tem recentemente recebido a atenção da comunidade, motivando a pesquisa e desenvolvimento de ambientes para a criação, experimentação e performance musicais utilizando dispositivos móveis, tanto de forma cooperativa quanto individual (BASSOLI et al., 2003 e 2004; HÅKANSSON et al., 2005; JACOBSSON et al., 2005; SCHIEMER e HAVRYLIV, 2005; TANAKA, 2004; YAMAUCHI e IWATAKE, 2005). Nestas experiências, as formas de representação e de interação com informações musicais, o aproveitamento das vantagens da mobilidade e do acesso imediato a informação, as possibilidades em termos de conectividade e as formas de sincronização de tarefas cooperativas daí derivadas, bem como as alternativas para contornar as limitações dos dispositivos portáteis, são algumas das questões mais importantes. Foros típicos onde este tema emergente tem sido discutido são o International Workshop on Mobile Music Technology, a International Conference on New Interfaces for Musical Expression – NIME, e a International Conference on Ubiquitous Computing – UbiComp.

A mobilidade sempre esteve presente na música. Boa parte dos instrumentos musicais acústicos (*lo-tech*) são portáteis, e músicos ambulantes, como seresteiros, bardos, trovadores podem mesmo ser considerados os precursores da "música móvel"

(*Mobile Music*, como esta recente área já está sendo chamada – BEHRENDT, 2005; GAYE et al., 2006). Mesmo hoje, com a disponibilidade de recursos *hi-tech*, continua existindo a preocupação em criar soluções tecnológicas que atendam à mobilidade intrínseca a um músico de palco, por exemplo. E ainda há o lado criativo, com necessidades ainda não atendidas: a inspiração pode chegar a um compositor em momentos inesperados... E se ele não tiver um instrumento à mão? E se ele não tiver meios de anotar aquela idéia? E se, por outro lado, ele tivesse a possibilidade de contatar imediatamente um colega, mesmo que localizado remotamente, e discutir aquela idéia, chegando a um acordo estético e assim refinando-a?

Mais recentemente, o fenômeno da convergência tecnológica tem dotado, por exemplo, os telefones celulares de um crescente poder de processamento para outras tarefas (inclusive musicais), e estes e os PDAs têm incorporado conectividade bidirecional, proporcionada por diferentes sistemas como Wi-Fi, UMTS ou Bluetooth. Essa conectividade expande radicalmente as possibilidades para a música móvel, especialmente se combinada àquela primeira característica crucial, a da crescente capacidade de processamento. Músicos, artistas e pesquisadores têm explorado computadores interconectados, como a Internet, já há algum tempo. Agora eles encaram novos desafios num momento em que a tecnologia conectada e digital encontra a mobilidade. Os dispositivos são utilizados "em trânsito" e portanto devem atender a novos requisitos funcionais, de interação (novas interfaces) e de cooperação.

2.1 Aplicativos Musicais em Dispositivos Móveis

Para desenvolvermos aplicações musicais em dispositivos móveis, precisamos compreender sua relação com computação móvel. A computação móvel implica na capacidade de se utilizar uma tecnologia computacional livre de conexões físicas, em contextos/ambientes remotos e móveis (não estáticos, em movimento), em oposição à computação *desktop* ou *stationary*. Mesmo o uso deste termo evoluiu, e hoje em geral implica que a computação ocorra de forma conectada (sem fios) a redes de dados, sem que haja prejuízo da tarefa devido a deslocamentos físicos do usuário. Um "computador móvel", ou dispositivo móvel, é qualquer dispositivo portátil com capacidade de computação, construído para poder mudar de localização (ser movido) enquanto mantém sua funcionalidade (FIUCZYNSKI, 1994).

O embrião dos aplicativos musicais móveis começou a surgir com a popularização dos telefones celulares, quando as melodias de toque dos telefones começaram a ser configuráveis. Logo foi utilizado o padrão MIDI de codificação musical para os arquivos dessas melodias (que precisavam ser compactos), e sua execução no telefone era sintetizada no processador DSP. O resultado foi o surgimento de um mercado bastante popular de toques de celular, os *ringtones*, que hoje podem ser buscados via Internet, comprados e carregados no telefone (BRANDON, 2006). Hoje também já se tornaram populares os telefones celulares com capacidade de processamento e execução de arquivos no formato MP3, que agora também podem ser definidos no aparelho como o toque de chamada. No entanto, como bem apontado por Flores (2008), isto é música *para* dispositivos móveis e o seu respectivo mercado comercial.

Este trabalho faz parte de uma investigação exploratória (a tese de doutorado de Flores (2009)), visando expandir esta idéia para significar também a "música *em* dispositivos móveis", ou "música *com* dispositivos móveis", ou seja, o uso dos

dispositivos móveis como ferramenta musical. Um dispositivo móvel é então usado como plataforma para o desenvolvimento de aplicativos musicais.

2.2 Trabalhos Relacionados

Nesta seção vamos resumir os principais trabalhos relacionados ao desenvolvimento de aplicativos musicais em dispositivos móveis, grande parte deles publicados em um encontro denominado International Workshop on Mobile Music Technology (MOBILEMUSICWORKSHOP.ORG, 2009). Apesar disso, ainda existe pouco trabalho atual envolvendo as áreas de música e computação móvel integradamente, e principalmente ainda poucos trabalhos com enfoque computacional. Por exemplo, Dunlop e Brewster (2002) e Fagrell (et al., 1999) consideram apenas a interação humano-computador e o trabalho cooperativo em dispositivos móveis, respectivamente. Os trabalhos de Schrott e Glückler (2004), Bassoli (et al., 2003; 2004) e D'Arcangelo (2005) enfatizam o aspecto social da tecnologia móvel, enquanto que Behrendt (2005) enfoca somente aspectos artísticos/estéticos derivados dessa tecnologia.

Os poucos trabalhos relacionados que exploram as áreas em questão de forma mais integrada, incluindo avanços computacionais, em geral o fazem em contextos de atividades musicais não-usuais, portanto tendendo igualmente para uma experimentação que resulta mais em propostas artísticas do que em soluções práticas para necessidades reais dos músicos em contextos musicais mais comuns.

Andante (UEDA e KON, 2003; UEDA e KON, 2004), por exemplo, é uma infraestrutura baseada em agentes móveis musicais para a criação de sistemas distribuídos de composição e performance musical. Os agentes geram som nas máquinas em que se hospedam, tendo sido previamente programados com a técnica generativa de som/música que o "compositor" da obra preferir. Estes algoritmos generativos podem incluir respostas a determinadas intervenções do músico humano ou à interação com outros agentes, as quais, como se espera, são percebidas pelos agentes através de seus sensores. As reações podem ser na forma de mudanças no som gerado, ou também da migração do agente para outro nodo da rede. Se o agente migrar, ele interrompe a execução musical na máquina de origem e a prossegue na máquina destino. Com este conjunto de características tem-se a base para experimentos instigantes do ponto de vista musical (que não são discutidos aqui por fugirem ao escopo deste texto). Entretanto, como já foi dito, não se resolve nenhum problema específico das atividades musicais usuais.

O trabalho de Furlanete e Manzolli (2005) propõe uma plataforma para jogos sonoros distribuídos, explorando novas formas de interação musical com computadores. A música se baseia no paradigma composicional da auto-organização, sendo executada de forma distribuída em uma rede, através de agentes. Nesse modelo o compositor determina os processos dos quais emergirá a música, e pode interferir nesta "música emergente". O trabalho sugere a formalização dos processos de interação musical em um modelo semiótico, buscando resolver o difícil problema da interferência e do controle em sistemas que são suficientemente complexos para permitir a emergência de comportamentos auto-organizados. Novamente aqui a aplicabilidade limita-se à música erudita contemporânea, embora a solução computacional proposta talvez possa ser generalizável e útil a outros domínios, não-musicais.

No ambiente Push!Music (JACOBSSON et al., 2005), cada música, nos PDAs, é um agente móvel. Ao ser estabelecida uma rede *ad hoc*, estes agentes verificam as *playlists*

dos outros aparelhos. Se uma música-agente concluir que o dono do outro aparelho iria "gostar dela", esta migra para o outro aparelho. Aqui a aplicabilidade já é mais trivial, mas no entanto tem resultados mais sociais e culturais (pelo intercâmbio automático de músicas, os usuários se socializam e ampliam seu universo musical) do que musicais propriamente ditos (já que os usuários não interferem musicalmente no processo).

Yamauchi e Iwatake (2005) propõem um sistema para colaboração gráfica e musical em redes *ad hoc* formadas com PDAs. Semelhante ao Push!Music, mas as músicas são trocadas intencionalmente entre os usuários conectados. Ainda, permite que a interação resulte em uma "música coletiva" distribuída, através do intercâmbio não de músicas, mas de sons (*samples*), que são executados em grupo nos PDAs interconectados. Mesmo assim, a interferência dos participantes na "composição coletiva" é mínima, e o sistema também carece de mecanismos mais sofisticados de suporte a este trabalho cooperativo.

O conceito de Pocket Gamelan (gamelão portátil), de Schiemer e Havryliv (2005), é o de um conjunto de instrumentos musicais móveis. Seu projeto busca desenvolver um protótipo de software para uma rede de instrumentos portáteis. Um dos diferenciais deste trabalho é que explora a tecnologia Bluetooth como canal de conectividade entre controladores, sensores e instrumentos. O objetivo maior, portanto, é a mobilidade do músico no palco. Entretanto, esse trabalho, assim como o de Yamauchi e Iwatake (2005), trata de uma solução específica para música em rede, e não do suporte para desenvolver aplicativos para fazer música com um PDA, que o nosso trabalho procura enfocar.

Outro trabalho é o sistema de Tanaka (2004), para criação musical colaborativa em redes *wireless* móveis. Assemelha-se ao trabalho de Yamauchi e Iwatake, mas proporciona amplo suporte à cooperação musical. Entretanto, esse trabalho necessita de dispositivos adicionais além do dispositivo móvel para funcionar. Destaca-se o uso de acelerômetros (sensores de movimento) acoplados aos dispositivos, como um experimento com novas modalidades de interação (figura 2.1). Os músicos podem fazer movimentos com o aparelho, como se o próprio fosse um objeto sonoro (como um chocalho), e este movimento interfere na música coletiva em execução.

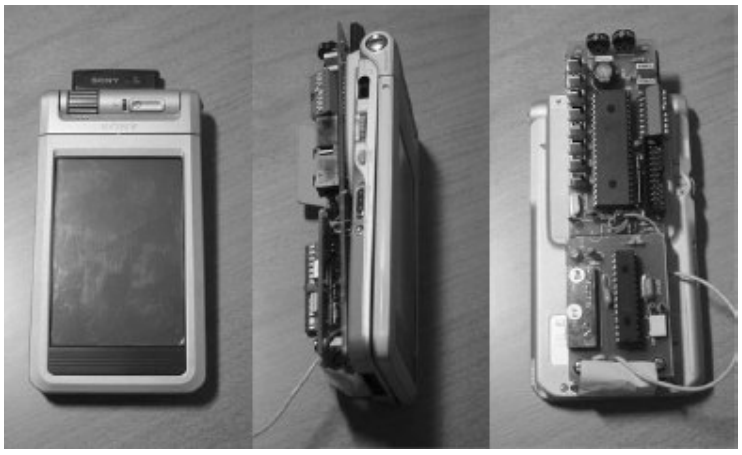

Figura 2.1: Um PDA como terminal móvel para música, com subsistema de sensores (TANAKA, 2004).

Por fim, mais um exemplo de trabalho recente relacionado é um sistema comercial denominado miniMIXA (SSEYO, 2006), baseado nas APIs Tao intEnt Sound System (ou Tao iSS – TAO GROUP, 2006) para dispositivos portáteis ou embarcados. Em termos de aplicação, este é o trabalho mais próximo dos nossos objetivos. Trata-se de um aplicativo para composição musical em PDAs ou *smartphones*, com muitas das características de sistemas de composição já existentes para plataformas *desktop* (figura 2.2), mas adaptadas às limitações dos dispositivos móveis.

Figura 2.2: Tela de execução de padrões sonoros do SSEYO miniMIXA.

Todos os trabalhos citados nesta seção são trabalhos relacionados ao desta monografia por serem exemplos do desenvolvimento de aplicativos musicais em dispositivos móveis. O principal diferencial entre eles e o nosso trabalho, é que todos esses trabalhos são soluções específicas para cada aplicação, e nenhum deles analisa ou propõe alguma solução mais generalizável a outras aplicações musicais móveis, ou aplicável ao desenvolvimento de aplicativos musicais móveis em geral. Além disto, alguns dos trabalhos que são mais generalizáveis destinam-se a contextos de atividades musicais não-usuais (música erudita contemporânea e experimental, ou acadêmica), e sua experimentação demonstra o enfoque em propostas artísticas mais do que em soluções práticas para necessidades reais dos músicos em contextos musicais mais comuns, que é a proposta do nosso trabalho.

Nosso trabalho não propõe uma solução específica a um caso: propomos um framework que será aplicável ao desenvolvimento de aplicativos musicais móveis em geral, para contextos de atividades musicais mais usuais. O único trabalho mais generalizável aparece no último caso citado, o miniMIXA: é uma solução específica, mas a API Tao iSS que utiliza é uma solução genérica para o desenvolvimento de aplicativos. No entanto, essa API não está mais disponível, e tratava-se de uma solução comercial, proprietária. O trabalho aqui proposto é livre: nosso framework está todo definido e explicado nesta monografia, e qualquer desenvolvedor poderá implementá-lo em outras linguagens se optar por não usar nossa implementação em .NET.

2.3 Síntese Deste Capítulo

Dispositivos móveis têm características próprias que podem ser úteis para atividades musicais (apoio à mobilidade dos músicos, acesso rápido e distribuído a informação

musical e ao processamento de música, conectividade para atividades musicais). A intenção é definir e implementar o suporte computacional móvel adequado a elas. Neste trabalho, o suporte computacional será desenvolvido em PDAs.

Obviamente, os dispositivos móveis que estamos enfocando – PDAs – não foram criados, em princípio, visando o uso musical. Portanto, uma adaptação terá que ser feita, e nisto consiste boa parte do nosso trabalho. Analisando os recursos dos dispositivos móveis e o estado-da-arte descrito acima, concluímos que as principais questões a serem consideradas incluem as formas de representação e de interação com informações musicais (considerando, inclusive, os possíveis usos por não-músicos); o adequado aproveitamento do poder crescente de processamento dos dispositivos móveis; e, sem dúvida, as alternativas para contornar as *limitações* dos dispositivos portáteis, em termos de interface com o usuário, poder de processamento *versus* economia de energia, e capacidade de memória e de armazenamento.

3 UM FRAMEWORK PARA DESENVOLVIMENTO DE APLICAÇÕES MUSICAIS EM DISPOSITIVOS MÓVEIS

O framework proposto é um framework de aplicação e seu objetivo principal é dar suporte no desenvolvimento de aplicativos interativos musicais em dispositivos móveis. Neste capítulo, definiremos o framework independentemente de linguagem ou plataforma, sendo que o pré-requisito da linguagem é que ela seja orientada a objetos, possibilitando a flexibilidade e abstração de classes para construir novos modelos, facilidade de estabelecer ligações entre as classes, encapsulamento da implementação de métodos, reusabilidade de código, a extensibilidade através da inserção de novas classes e a robustez no tratamento de exceções.

Os principais requisitos do framework são:

- Permitir a criação de aplicativos que possam ser totalmente configuráveis pelo usuário, podendo ele escolher e configurar os componentes que serão utilizados.
- Funcionar como um sequenciador musical, isto é, permitirá ao usuário criar sequências de músicas passo a passo, além de permitir que se use como um instrumento virtual.
- Os componentes devem ser extensíveis para que o programador possa criar novas interfaces.
- Fornecer a base do aplicativo no modelo MVC (Model-View-Controller), sendo os componentes a parte View.
- A estrutura de dados necessária para composição e execução de música é pré-definida, fazendo com que componentes criados por diferentes programadores possam ser utilizados de forma conjunta.
- O framework deve ser flexível o bastante para que permita criar diferentes interfaces para criação musical, sempre visando facilitar o desenvolvimento de novos componentes.
- O código implementado deve ser eficiente para execução em dispositivos móveis, que possuem um poder de processamento limitado.

As seções seguintes deste capítulo vão apresentar as definições necessárias para o entendimento da proposição, a arquitetura, os elementos (classes e seus métodos) e as principais características do framework.

3.1 Terminologia Utilizada

Para facilitar a compreensão do texto, definiremos alguns termos quanto ao contexto utilizado neste trabalho:

Usuário: Quem vai utilizar os aplicativos baseados no framework. Como o framework funciona como um sequenciador, os potenciais usuários de aplicativos criados com base nele serão os mesmos que usariam um sequenciador para apoio na criação e execução de músicas, isto é, músicos ou aspirantes a músicos.

Programador: Irá desenvolver novos controles a partir do framework, criando novos aplicativos baseados nele.

Controle: Refere-se a um componente de interface gráfica.

Aplicativo: Um aplicativo que utiliza o framework como base, isto é, utiliza controles criados usando o framework.

3.2 Framework

Framework é uma coleção de classes cooperativas que implementam os mecanismos que são essenciais para um domínio de aplicação em particular. Sendo assim, permite a criação de programas reusáveis e define uma estrutura para o domínio em questão (HORSTMANN, 2007).

A partir de um framework é possível definir a estrutura de classes e os esquemas de colaboração, estabelecendo o fluxo de controle da aplicação e ficando com sua estrutura pré-definida. Também permite o reuso através de especialização (criação de subclasses) e instanciação de métodos e classes específicas. Devido a essas características, obviamente será definida utilizando orientação a objetos.

Um **framework de aplicação** é um conjunto de classes que implementa serviços comuns a um tipo de aplicação.

3.3 Criando um Framework Extensível Utilizando Builder com Reflexão

Para que seja possível a criação do framework extensível, será utilizado o Design Pattern Builder (GAMMA, 2000) (figura 3.1), sendo este instanciado através de reflexão. O Builder tem como objetivo separar a construção de um objeto complexo de sua representação para que o mesmo processo de construção possa criar diferentes representações.

É definida uma classe base que deve ser derivada para criação da classe a ser instanciada. O objeto é instanciado através do Design Pattern Factory Method (GAMMA, 2000).

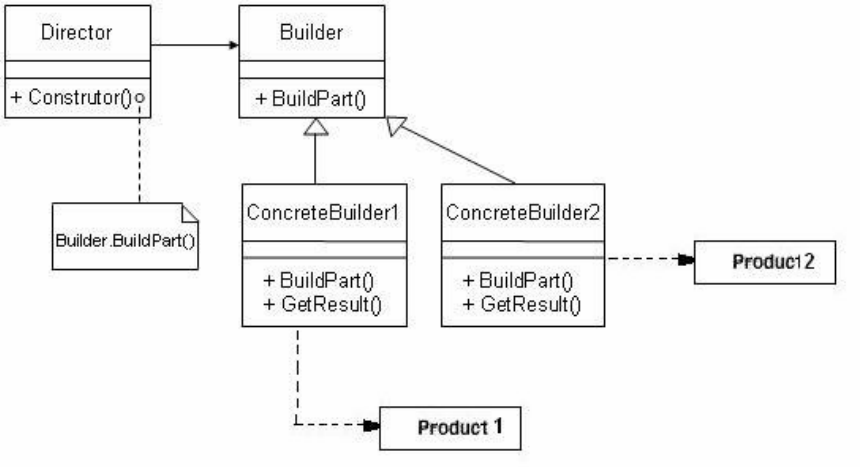

Figura 3.1: O *design pattern* Builder.

São componentes do Builder:

Builder:

- Especifica a interface para criar partes de um produto. (Define os métodos a implementar/sobrescrever.)

Concrete Builder:

- Constrói e monta as partes do produto ao implementar a interface definida no *Builder*.
- Define e mantém a representação que criou.
- É quem implementa os métodos, sendo assim, quem executa as partes do objeto *Builder*.

Director:

- É quem constrói o objeto usando a interface do *Builder*.

Product:

- Instância do objeto complexo em construção. O *Concrete Builder* constrói a representação interna e define o processo pelo qual é montado.
- O *Concrete Builder* é quem define que tipo de produto foi criado.

No caso no nosso framework, será pré-definido nas classes base toda estrutura de dados e uma biblioteca de métodos que auxiliarão no desenvolvimento do aplicativo, sendo deixado para desenvolver apenas os aspectos referentes à interface com o usuário.

A reflexão (presente na plataforma .NET e linguagens como PHP, Java e Ruby) permite que se verifique que classes foram implementadas em um arquivo executável e instanciá-las de acordo.

Assim, podemos conferir o tipo de classe (e classe base) implementado e instanciar a classe de acordo com a sua classe base, conforme os passos:

1. Seleção do arquivo executável (por exemplo, uma .dll no .NET, um .jar no Java).
2. Verifica entre as classes implementadas qual tem como base uma classe pertencente ao framework.

3. Instancia e constrói a classe de acordo com a classe base.

Dessa forma, o usuário poderá escolher o arquivo a carregar e o framework irá verificar se este possui uma das classes passíveis de uso, fazendo a instanciação conforme a classe.

Com isso, ganhamos muito mais flexibilidade do que usando a técnica normalmente mostrada na literatura sobre Factory Method (GAMMA, 2000) (DOFACTORY, 2009), pois assim podemos implementar novas classes sem precisar alterar o framework.

3.4 Componentes do Framework

A seguir descrevemos as classes que compõem o framework.

3.4.1 baseGUIController

A classe baseGUIController será o ponto central do aplicativo a criar. Ele será o único componente que será fornecido com sua interface pré-definida, pois a partir dele se poderá instanciar novos componentes, configurar o aplicativo, definir o andamento da execução da música (batidas por minuto) e controlar a execução.

Em relação ao Design Pattern Builder, descrito no item 3.3, ele será o *director* (figura 3.1), pois através do método addControl, serão instanciados ConcreteBuilders.

A figura 3.2 abaixo mostra a interface do baseGUIController.

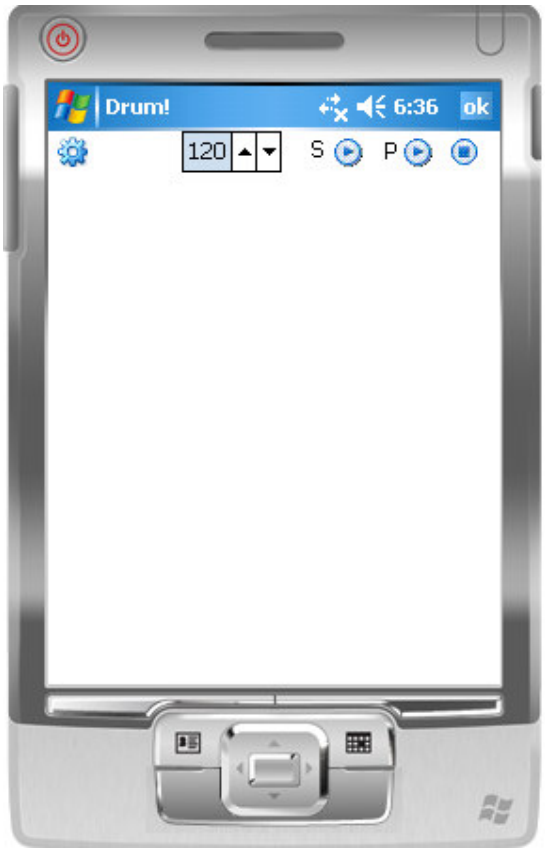

Figura 3.2: Interface do baseGUIController.

Os componentes da interface do baseGUIController são (da esquerda para a direita):

- **Configuração:** Abre o menu para configuração do aplicativo. Possui as opções:
  - Posição: Altera a posição dos controles carregados.
  - Wav: Altera o arquivo de som de um controle.
  - Salva Configuração: Salva a configuração do framework (controles carregados, suas posições e arquivos de som).
  - Carrega Configuração: Carrega uma configuração previamente salva.
  - Salva Song: Salva os dados do Song.
  - Carrega Song: Carrega dados de um Song salvo.
  - Ajuda: Exibe ajuda do sistema.
- **Batidas por minuto:** Controla o andamento (velocidade de execução) do Song e Pattern.
- **Executa Song:** Executa o Song atual.
- **Executa Pattern:** Executa o Pattern atual.
- **Para execução:** Para a execução de song ou pattern.

A classe possui membros descritos na figura 3.3 abaixo.

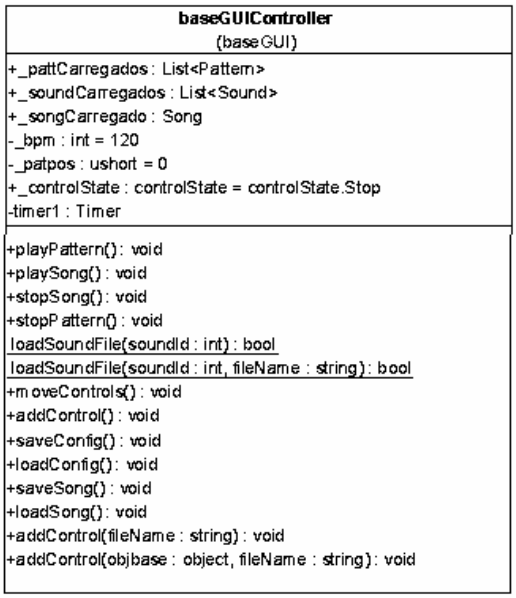

Figura 3.3: A classe baseGUIController.

Campos:

_pattCarregados: Lista de controles de classe Pattern carregados.

_soundCarregados: Lista de controles de classe Sound carregados.

_songCarregado: O controle de classe Song carregado.

_bpm: Indica o andamento do Pattern e Song durante a execução. A unidade de tempo é em batidas por minuto (bpm).

_patPos: Posição atual no Pattern, usado quando está sendo executado, à cada intervalo de tempo calculado a partir de _bpm, ele é acrescido de uma unidade.

Métodos:

moveControls: Permite ao usuário mover os controles já inseridos.

playPattern: Inicia a execução dos Patterns carregados.

playSong: Inicia a execução do Song.

stopPattern: Para execução dos Patterns.

stopSong: Para execução do Song.

calculaIntervaloBPM:A partir do _bpm, calcula o intervalo em milissegundos entre partes do Pattern.

loadSoundFile: Carrega um arquivo de som na memória para posterior execução. Se não for fornecido o nome do arquivo, este será pedido através de uma dialog box.

saveConfig: Salva a configuração atual: controles carregados, a posição dos controles e os arquivos de som carregados em um arquivo no formato XML.

loadConfig: Carrega uma configuração previamente salva.

saveSong: Salva os Patterns e dados do Song atual em um arquivo no formato XML.

loadSong: Carrega os dados de um Song previamente salvo.

As figuras 3.4 e 3.5 abaixo ilustram o diagrama de classes do framework.

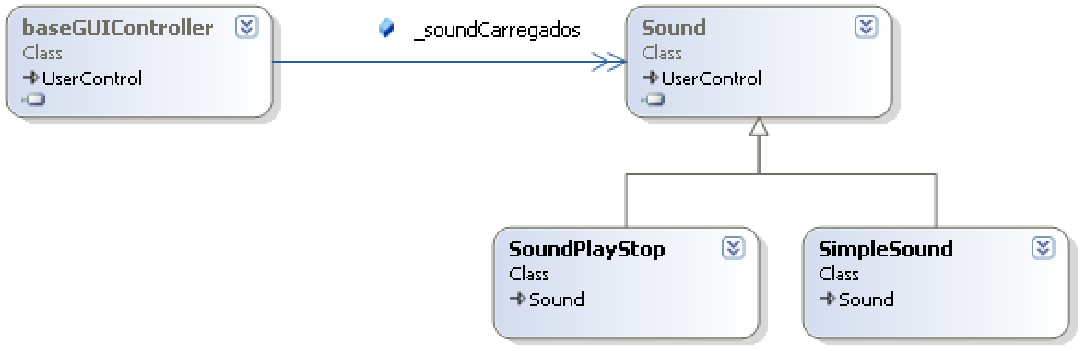

Figura 3.4: Diagrama de classes do framework, referente a Sound.

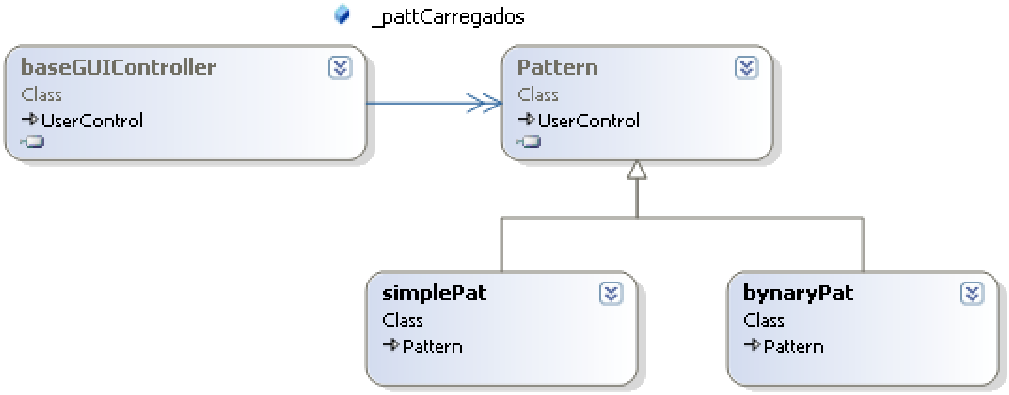

Figura 3.5: Diagrama de classes do framework, referente a Pattern.

A partir do método addControl da classe baseGUIController, pode-se instanciar classes derivadas de Sound, Pattern e Song dinamicamente. Usando o exemplo da figura 3.5, quando o usuário seleciona os fontes de simplePat ou bynaryPat estes são instanciados e controlados pelo baseGUIController. As classes Sound, Pattern e Song são o equivalente à classe Builder no Design Pattern Builder (figura 3.1).

3.4.2 Sound

A classe Sound (figura 3.6) é uma classe abstrata que definirá a base para interação direta do usuário, funcionando como um instrumento de percussão, por exemplo. A partir dela o programador poderá definir eventos que irão disparar sons, assim como a aparência do controle Sound.

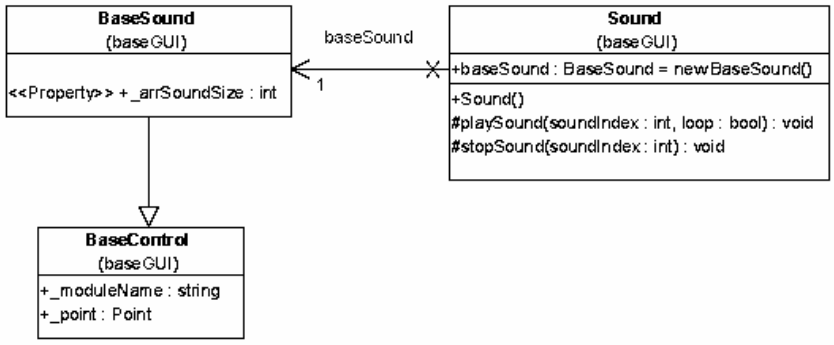

Figura 3.6: A classe Sound.

Classe base:

Por ser uma classe base para um componente gráfico, tem como classe base a classe do controle gráfico mais básica da linguagem onde está sendo implementada. Como exemplo, em .NET, a classe base seria windows.forms.UserControl (MSDN, 2009-a), se for em Java Swing, a classe base seria java.swing.JComponent (WIKIPEDIA, 2009).

Para facilitar a serialização da configuração (a partir do método SaveConfig() da classe BaseGUIController), os campos e propriedades estão na classe BaseSound, facilitando o salvamento dos dados dos Sound carregados.

Campos:

_moduleName: Ajustado automaticamente ao carregar controle, é usado para carregar configuração (método LoadConfig() da classe BaseGUIController).

_point: Também é ajustado automaticamente ao carregar ou mover controle e é salvo/carregado.

Propriedades:

_arrSoundSize: indica quantos sons o controle usará.

Métodos:

playSound(int soundId, bool loop): Executa o som referenciado pelo índice. Loop indica se o som irá ser executado novamente ao chegar ao fim.

stopSound(int soundId): Interrompe a execução do som.

Ao ser tocado pela primeira vez, o framework verifica se o som ainda não foi definido e nesse caso irá pedir ao usuário para selecionar o som a executar.

Desta forma, para definir um Sound para ser utilizado no framework, deve-se:

1. Definir a parte gráfica propriamente dita: imagem ou cor de fundo e tamanho.
2. Definir os eventos que disparam os sons, por exemplo dispara um som conforme a área onde o usuário clicar.

A figura 3.7 abaixo exemplifica um Sound.

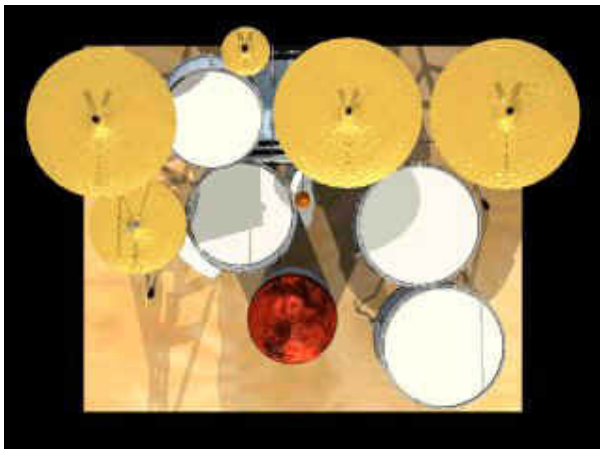

Figura 3.7: Exemplo de implementação de um Sound (Copyright 2001 By Mike James, http://www.nextcraft.com/).

Na figura 3.7, um som diferente é disparado conforme o toque em uma das partes da bateria ilustrada.

3.4.3 Pattern

A classe Pattern (figura 3.8) permite definir um padrão rítmico de acordo com divisões de tempo determinadas. Por motivo de simplicidade, a precisão do Pattern será fixa em um quarto de batida (cada divisão, ou célula, do Pattern representa um quarto de batida). Esta precisão é bastante utilizada em sequenciadores eletrônicos.

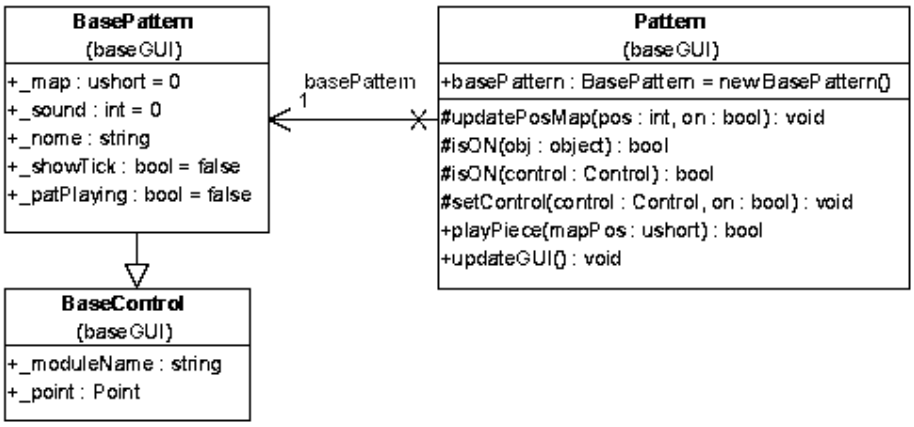

Figura 3.8: A classe Pattern.

O Pattern inteiro, com 16 divisões, representa portanto 4 batidas ou tempos. O andamento será regulado pelo membro _bpm ou o controle BPM (batidas por minuto) na classe baseGUIController.

Pode-se carregar vários Patterns, que serão executados simultaneamente. Assim, pode-se por exemplo criar uma bateria tendo um pattern associado à cada peça dela (por exemplo: bumbo, caixa, chipô).

Assim como no Sound, para facilitar a serialização da configuração (a partir do método SaveConfig() da classe BaseGUIController), os campos e propriedades estão na classe BasePattern, facilitando o salvamento dos dados dos Pattern carregados.

Campos:

_moduleName: Ajustado automaticamente ao carregar controle, é usado para carregar configuração (método LoadConfig() da classe BaseGUIController).

_point: Também é ajustado automaticamente ao carregar ou mover controle e é salvo/carregado.

_sound: O som a tocar.

_map: Mapa de bits que representa o padrão rítmico. Cada bit representa um pedaço do tempo a tocar. Por exemplo '10' é uma execução seguido de uma pausa.

_nome: Nome do controle, usado quando o controle é carregado.

_showTick: Booleano, se verdadeiro, o controle irá marcar a passagem do tempo (normalmente um dos Pattern irá ter _showTick verdadeiro e os outros falso).

Métodos:

updatePosMap(int pos,bool on): Ajusta o campo _map no bit indicado pelo parâmetro 'pos' conforme o parâmetro 'on': se verdadeiro seta o bit para 1, se falso, seta para 0. Este método deve ser chamado para atualizar o _map de

acordo com a interface gráfica (por exemplo, quando o usuário clicar e alterar o padrão rítmico.

isON(Control control): verifica se o controle está ligado indicando que o bit do _map deve estar ligado também. Deve ser reescrito caso o controle não use cor de fundo para indicar se está ligado (por exemplo, se for uma imagem).

setControl(Control control, bool on): Ajusta o controle gráfico conforme o valor passado no parâmetro 'on'. Também deve ser reescrito como o método isON.

playPiece(ushort mapPos): Método para verificar se deve tocar na posição dada. Pode ser sobrescrita para executar outras ações (como atualizar parte gráfica ou trocar o som a tocar). Nesse caso, colocar na última linha: return base.playPiece. O parâmetro mapPos indica a posição a tocar (0 a primeira, 1 a segunda...).

updateGUI(): Chamado do baseGUIController para atualizar a parte gráfica do pattern porque alterou dados (por exemplo, carregou novo Song) refletindo no controle Pattern o valor do _map. Pode utilizar o método setControl e playPiece para esta tarefa. Este método deve ser sobre escrito conforme a interface criada.

Desta forma, para definir um Pattern para ser utilizado no framework, deve-se:

1. Definir a parte gráfica propriamente dita: Representar o padrão rítmico graficamente (por exemplo, um controle para cada bit do _map (figura 3.9)).
2. Definir eventos para atualização do _map e dos controles que o representam.
3. Implementar o método updateGUI, que será chamado ao executar um Song ou ao carregar um novo Song.

Figura 3.9: Interface de um Pattern implementado.

Na figura 3.9 o padrão rítmico é representado por 16 quadros correspondentes aos bits do _map. Os quadros escuros indicam que o som deve ser executado na porção de tempo referente ao quadro.

3.4.4 Song

O Song é um sequenciamento de Patterns (figura 3.10). Ao se definir quais Patterns usar e em que ordem, pode-se criar uma música completa.

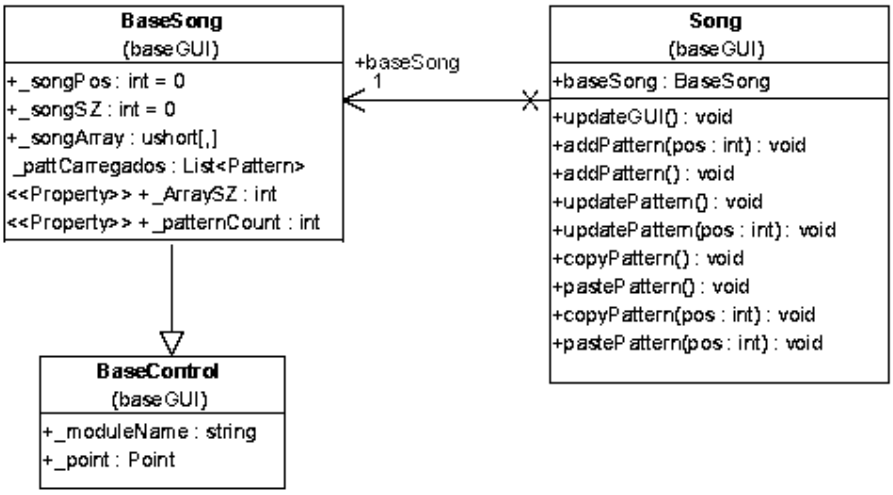

Figura 3.10: A classe Song.

Assim como no Sound e Pattern, para facilitar a serialização da configuração (a partir do método SaveConfig() da classe BaseGUIController), os campos e propriedades estão na classe BaseSong, facilitando o salvamento dos dados do Sound.

Campos:

_songPos: Posição que está sendo manipulada no song. Funciona como um cursor de posicionamento.

_songSZ: O tamanho da música, ou seja o número de sequencias criadas.

_ArraySZ: Tamanho do array do Song. Indica o máximo de Patterns que poderão ser sequenciados.

_pattCarregados: Patterns carregados, que tocarão simultaneamente. É uma referência à lista de mesmo nome em BaseGUIController.

_songArray: O array que contém os Patterns sequenciados. A primeira dimensão do array indica posição e a segunda indica o mapa do pattern.

Métodos:

addPattern(int pos): Adiciona os mapas de _pattCarregados no Song, na posição indicada no parâmetro 'pos'.

addPattern(): Idem ao anterior, porém na posição de _songPos.

copyPattern(int pos): Copia para o clipboard do sistema os mapas do Song da posição indicada no parâmeto 'pos'.

copyPattern(): Idem ao anterior, porém copia da posição de _songPos.

pastePattern(int pos): Copia do clipboard para o Song os mapas previamente copiados, na posição indicada pelo parâmetro pos.

pastePattern(): Idem ao anterior, porém copia na posição de _songPos.

updateGUI(): Atualizar a parte gráfica do Song e dos Pattern porque alterou dados (por exemplo, carregou novo Song) refletindo na parte gráfica os valores

dos campos. Este método deve ser sobreescrito de acordo com a interface criada.

Desta forma, para definir um Song para ser utilizado no framework, deve-se:

1. Definir a parte gráfica propriamente dita: Incluir visualização para a posição atual do Song, e criar interface para os métodos desejados.
2. Implementar o método UpdateGUI, para que possa ser chamado ao executar um Song.

3.4.5 Configuration

A classe Configuration é uma classe estática que permite a configuração de cores e pastas a ser utilizadas no framework.

Os campos para definição de cor são para dar uma unidade na interface, e são utilizadas pelos métodos isOn e setControl da classe Pattern. Porém, não é obrigatório o uso delas, e pode-se sobreescrever estes métodos.

Possui os seguintes campos, todos estáticos:

_colorHighLightSel: Cor que definirá a passagem do tempo no Pattern (quando o tempo estiver em uma posição, a cor deve ser esta).

_colorSelected: Cor indicando que o som do pattern vai ser tocado.

_colorUnSelected: Cor indicando que o som do pattern não vai ser tocado.

_gridSize: O tamanho do grid usado quando vão ser movidos os controles.

_modulePath: Pasta inicial para a seleção dos módulos a carregar. Valor inicial é 'modules'.

_wavPath: Pasta inicial para seleção dos arquivos de som.

_configurationPath: Pasta inicial dos arquivos de configuração.

_songPath: Pasta inicial dos arquivos de Song, onde poderão ser salvos ou carregadas músicas compostas pelo framework.

4 EXEMPLO DE IMPLEMENTAÇÃO

Neste capítulo vamos mostrar a implementação do framework, descrevendo o ambiente e linguagem escolhidos, a implementação de classes auxiliares que são específicas da plataforma escolhida e criaremos interfaces a partir da derivação das classes base do framework (Sound, Pattern e Song), a fim de exercitar as possibilidades do framework.

4.1 Dispositivo e Linguagem Escolhidos

Devido à experiência do autor em desenvolvimento de sistemas em C#, linguagem do .NET, no sistema operacional Windows Mobile, e ao fato do Instituto de Informática da UFRGS possuir um dispositivo com esse sistema operacional, um Ipaq 2490b com Windows Mobile 5.0, este foi o escolhido para a implementação do framework. Seu formato é semelhante ao da figura 3.2. São características do modelo:

- Processador: Intel PXA270.
- Clock: 520 MHz.
- Tela: Resolução de 320x200 pixel, com *touchscreen* que capta apenas um toque por vez.
- Memória: 192MB, sendo 128 de ROM e 64 de RAM
- Slot cartões Security Digital: Possibilidade de expansão de memória de até 2GB.
- Slot cartões Compact Flash, que permite também expansão de memória até 2GB ou inserção de expansões como câmera fotográfica, GPS, ou mini impressoras.
- Microfone: Possui um microfone cuja finalidade é a gravação de lembretes e notas. Grava em formato WAV podendo assim ser diretamente usado no framework implementado.
- Alto Falante: Possui um altofalante embutido, porém com uma resposta pouco linear (praticamente corta os graves).
- Saída de Fone de Ouvido: Podendo ser ligados também a caixas de som externas, permite uma melhor audição da música sendo criada.
- Bluetooth: Permite conexão com alto falantes externos sem fio, e também controlar outros dispositivos, como outros PDA.
- Rede Wireless padrão 802.11b.

4.2 Considerações Sobre o Windows Mobile 5.0 e o .NET Compact Framework

Esta seção irá descrever as características específicas da plataforma escolhida para desenvolvimento.

4.2.1 Performance

O Ipaq 2490b, usado na implementação mostrou possuir um poder de processamento muito bom, permitindo em testes utilizar até 6 Patterns simultâneos sem problemas de performance.

4.2.2 Reprodução de arquivos de áudio

A versão 2.0 do .NET Compact Framework não possui API para reprodução de áudio em canais simultâneos. Por isso, foi necessário implementar em C uma biblioteca para execução de arquivos (maiores detalhes em 4.3.3). Isso é possível somente com arquivos WAV, por não ser necessário processamento ou decodificação do arquivo. A versão 3.5, lançada recentemente, já possui implementado em .NET as funções necessárias para execução de áudio, porém não foi lançada a tempo de ser utilizada no presente trabalho.

4.2.3 Precisão do Touch Screen

É importante ressaltar que a precisão do Touch Screen é muito maior do que quando se usa mouse. Pode-se tocar diferentes partes da tela com muito mais rapidez.

Usando-se o *stylus* (uma espécie de caneta com a ponta de plástico usada para tocar a tela), podemos criar áreas de 16 por 16 pixel, como na figura 3.9 e acessá-las sem erro e rapidamente.

4.3 Implementação do Framework

Relataremos a seguir detalhes da implementação do framework que não foram descritas no capítulo 3 por serem específicas da plataforma.

4.3.1 Pré-Requisitos

Para o desenvolvimento em .NET para o Windows Mobile, é necessário o Visual Studio 2008 professional. Outras versões como a Express Edition (que é gratuita) e a Standard não incluem o Compact Framework.

4.3.2 Classe Auxiliar: FileSelect

O Windows Mobile 5 não possui uma forma adequada para seleção de arquivos, pois a classe do .NET para isso (System.Windows.Forms.OpenFileDialog) lista os arquivos conforme o tipo e não possui uma organização por pastas como os usuários são acostumados a utilizar. Além disso, nem todos os arquivos são listados, por exemplo os que estão em cartões Security Digital. Por isso, foi necessária a criação da classe FileSelect para seleção de módulos e arquivos de som a carregar.

Figura 4.1: Interface da classe FileSelect.

A figura 4.1 Ilustra uma instância da classe FileSelect para que seja selecionado o som a reproduzir. Com essa classe é possível navegar através de diretórios (que aparecem entre colchetes). Como no exemplo acima, '[..]' é a pasta pai de '\storage card\wav', no caso, '\storage card'. Após selecionado o arquivo, deve-se clicar em 'OK' ou no botão 'Enter' (botão central abaixo da tela) para confirmar a operação. Essa classe é usada no método addControl da classe baseGUIController.

4.3.3 Implementando a execução dos arquivos de som

A parte de execução de arquivos de som foi implementada em linguagem C++, baseando-se na biblioteca 'waveOut' (MSDN, 2009-d) que permite a execução de múltiplos arquivos no formato Wave.

A conexão com a parte .NET foi feita através de métodos que permitem carregar arquivo, descarregar arquivo, executar e parar execução. Externamente, é usado um número identificador para cada arquivo de som. Assim, ao carregar um arquivo de som, é retornado um número que identifica unicamente o arquivo, sendo possível executá-lo referindo-se apenas ao índice. Esse modo de representação permite que se acesse os arquivos mais rapidamente do que pelo nome do arquivo, pois é acessado diretamente pelo índice.

4.3.4 Serialização

O .NET Compact Framework possui métodos para serialização dos objetos em arquivos no formato XML. Algumas transformações foram necessárias pois não é permitida a serialização de objetos complexos diretamente, sendo necessário uma decomposição do objeto para que possa ser salvo, por exemplo, o mapa de *waves* carregados é salvo em um objeto HashTable (MSDN, 2009-e), que é uma lista de pares chave e valor que permite pesquisa de campos chave. O .NET não permite a serialização de HashTable, então tivemos que converter este objeto em dois *arrays* na serialização e ao de-serializar remontamos o HashTable.

4.4 Experimentando Interfaces

Nesta seção iremos avaliar o framework através da criação de diferentes interfaces, podendo assim verificar o potencial e adequação do framework para criação de aplicativos musicais. Todas experimentações serão disponibilizadas como templates (projetos pré-definidos, descritos no apêndice), que poderão servir como base para criação de novos controles.

4.4.1 Implementando Sound

Descreveremos aqui como criar um controle Sound, conforme descrito no final do item 3.4.2, seguindo os passos para criação da classe.

4.4.1.1 SimpleSound

Começaremos com o Sound mais simples possível: Um *pad* que ao ser tocado executa um arquivo Wave.

Definição da parte gráfica:

- A parte gráfica pode ser definida por um Panel (MSDN, 2009-b), que é um retângulo com uma cor de fundo. O Visual Studio permite a inserção de controles arrastando o controle desejado para controle sendo criado.

Definição do evento que dispara o som:

- O Evento de tocar a tela é chamado de MouseDown. A implementação em .NET é:

```
this.panel1.MouseDown += new MouseEventHandler(panel1_MouseDown);
```

- Ao acrescentar a linha acima no construtor da classe, definimos que panel1_MouseDown será chamado a cada vez que o Panel1 for tocado.

No método Panel1_MouseDown, chamamos o playSound que é membro da classe Sound. Os parâmetros são índice do som a tocar (0 é o primeiro) e se vai ser tocado em *loop* ou não. O código da implementação é:

```
void panel1_MouseDown(object sender, MouseEventArgs e)
{
    playSound(0, false);
}
```

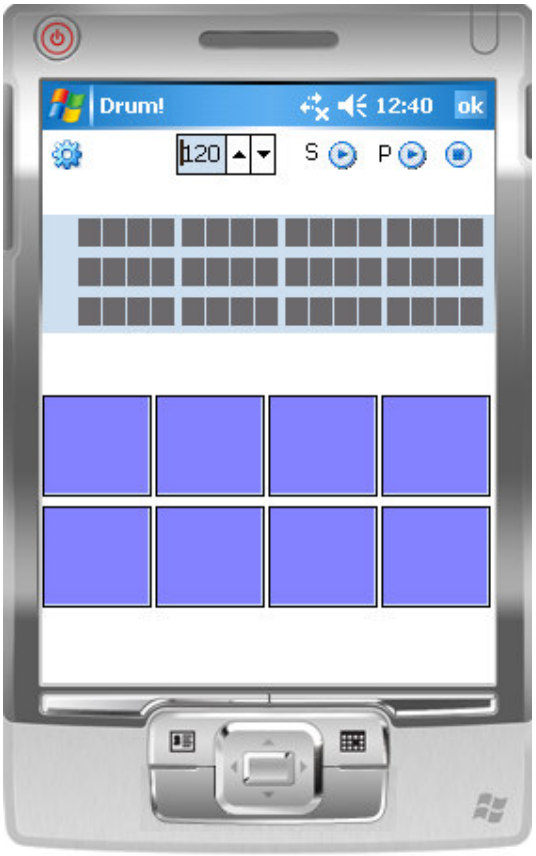

Figura 4.2: De cima para baixo: baseGUIController, três Patterns e oito SimpleSounds.

4.4.1.2 SimpleDrum

Este Sound possui mais arquivos de som para representar uma bateria completa.

Definição da parte gráfica:

- Cria-se um PictureBox (MSDN, 2009-c) com a imagem como o da figura 3.7. PictureBox é um controle usado para exibição de imagens.

Definição do evento que dispara o som:

- Antes é necessário redefinir a quantidade de sons que o Sound toca. Basta incluir uma atribuição para o membro da classe BaseSound _arrSoundSize.

- Chama-se o método playSound com um som diferente para cada área tocada.

Figura 4.3: SimpleDrum.

4.4.1.3 RecoReco

A idéia é representar um reco-reco virtual (MELOTECA, 2009): ao se deslizar o *stylus* sobre o controle, são disparados sons. Para isso, criou-se uma distância arbitrária a partir da qual o som é disparado.

A representação gráfica está definida na figura 4.4.

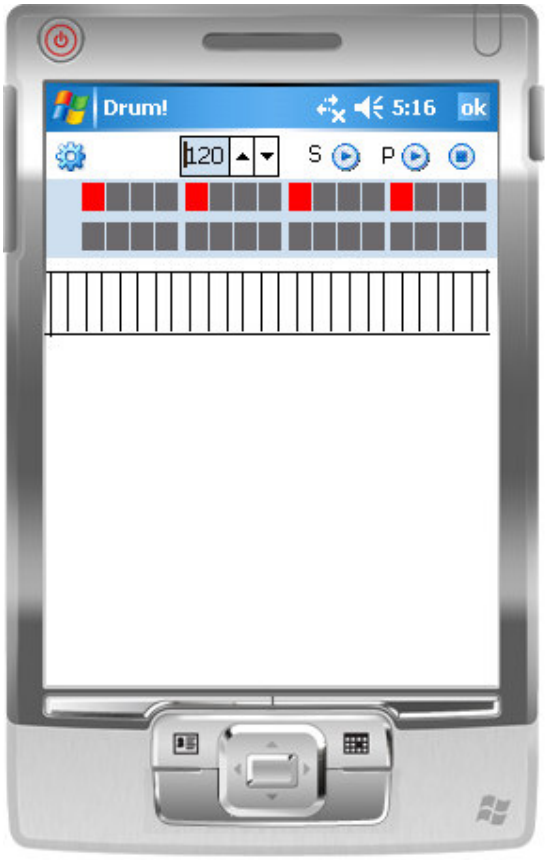

Figura 4.4: De cima para baixo: Um baseGUIController, dois Patterns e um RecoReco.

Utilizamos dois eventos: o já citado MouseDown e o MouseMove, que detecta o deslocamento sobre a tela.

Deve-se incluir no construtor do controle:

```
pictureBox1.MouseDown += new MouseEventHandler(pb1_MouseDown);
pictureBox1.MouseMove += new MouseEventHandler(pb1_MouseMove);
```

E adicionar os campos e métodos abaixo:

```
//Indica a posição anterior ao evento MouseMove:
int _X = 0;

//Indica o passo, a cada 10 unidades de movimento, é disparado o playSound
int _step = 10;

/// <summary>
/// Evento disparado quando usuário toca a tela
/// </summary>
/// <param name="sender">quem disparou o evento</param>
/// <param name="e">indica o ponto onde a tela foi tocada</param>

void pb1_MouseDown(object sender, MouseEventArgs e)
{
        _X = e.X;
}
```

```
/// <summary>
/// Evento disparado após usuário tocar a tela, quando ele
/// movimenta o cursor pela tela
/// </summary>
/// <param name="sender">quem disparou o evento</param>
/// <param name="e">indica o ponto onde a tela foi tocada</param>
void pb1_MouseMove(object sender, MouseEventArgs e)
{
        while (Math.Abs(_X - e.X) > _step)
        {
                if ((_X - e.X) > 0)
                        _X -= _step;
                else
                        _X += _step;

                this.playSound(0, false);

        }
}
```

No código acima, é disparado o playSound a cada deslocamento no eixo X, para qualquer direção, que exceda _step. Outra opção de implementação seria disparar diferentes sons de acordo com o eixo ou direção deslocados.

4.4.2 Implementando Pattern

Descreveremos aqui como criar um controle Pattern. Conforme descrito no final do item 3.4.3, seguiremos os passos para criação da classe.

4.4.2.1 SimplePattern

Esta implementação é a mais básica para representar o Pattern graficamente. A figura 3.9 mostra a parte gráfica do controle. Cada uma das 16 posições de um Pattern é representado por um Panel (MSDN, 2009-b) de 16x16 pixel. Ao se tocar em um Panel, a posição selecionada é ligada ou desligada, como um *toggle*, e isso é visualizado pelo usuário através da mudança de cor no panel selecionado. Aqui serão utilizados métodos e membros da classe Pattern.

Após criados os Panel no controle, falta associar os eventos aos Panel. Para isso adiciona-se no construtor da classe:

```
_arrPanel = new Panel[] { panel2, panel3, panel4, panel5,panel6, panel7, panel8,
panel9, panel10, panel11, panel12, panel13,panel14, panel15, panel16, panel17};

for(int i=_arrPanel.GetLowerBound(0); i<= _arrPanel.GetUpperBound(0); i++)
{

   Panel p = _arrPanel[i];

   p.Tag = i;

    p.MouseDown += new MouseEventHandler(panel_MouseDown);

}
```

Com isso, a cada vez que se tocar em um Panel, o método panel_mouseDown será chamado. O membro Tag, presente em todos controles, serve para se incluir dados adicionais. No caso, utilizamos para guardar o índice que será usado no método panel_mouseDown para ajustarmos o padrão rítmico.

A implementação do panel_mouseDown é:

```
void panel_MouseDown(object sender, MouseEventArgs e)
{
    Panel p = (Panel)sender;
    //inverte o valor: se true, seta pra false...
    setControl(p, !isON(p));
    updatePosMap((int)p.Tag, isON(p));
}
```

No Visual Studio 2008, os métodos para eventos são gerados automaticamente. Assim, após digitar "p.MouseDown +=", o esqueleto do método "panel_MouseDown" é gerado pela IDE.

SetControl irá ajustar a cor do controle conforme seu estado (ligado ou desligado).

UpdatePosMap irá atualizar o membro _map que é usado ao executar ou salvar um Pattern. Mais detalhes sobre estes métodos são descritos na seção 3.4.

Finalmente, é necessário implementar o método updateGUI, que é chamando quando um song está sendo executado, para fazer uma atualização da parte gráfica de acordo com o membro _map:

```
public override void updateGUI()
{
 for (int i = _arrPanel.GetLowerBound(0); i <= _arrPanel.GetUpperBound(0); i++)
 {
        setControl(_arrPanel[i],playPiece((ushort)i));
 }

}
```

O _arrPanel é um array que contém os Panels que representam o Pattern. Assim, basta chamar o método setControl como acima para que seja feita a atualização.

4.4.2.2 SelectPattern

A ideia desta implementação é agilizar a criação de padrões rítmicos através de uma interface que facilite a partir de menu para seleção de parte do Pattern e botões para cópia de uma parte para outra do Pattern.

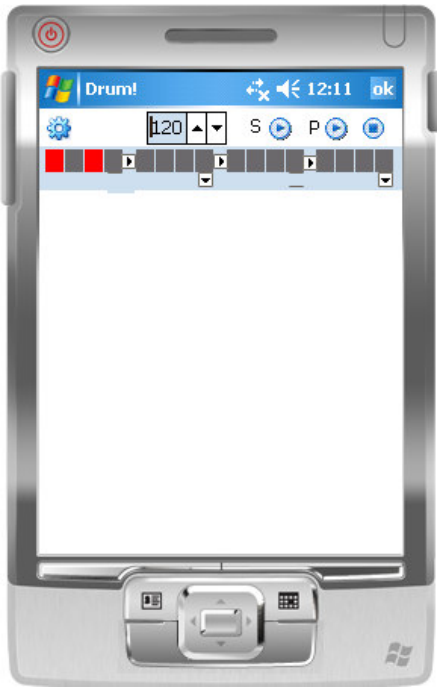

Figura 4.5: selectPattern, antes da cópia.

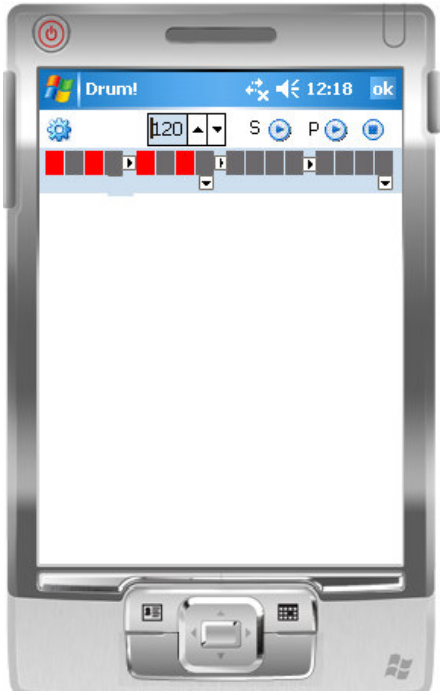

Figura 4.6: selectPattern, depois da cópia.

Na figura 4.6, após usuário clicar na primeira seta para direita, o padrão das quatro primeiras posições é copiado para as próximas quatro.

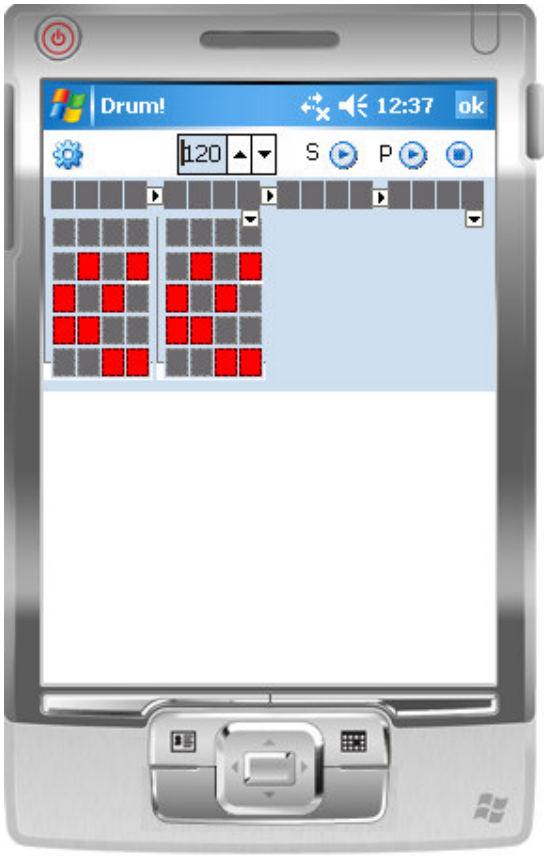

Figura 4.7: selectPattern.

Ao clicar na primeira seta para baixo, é exibido um menu para seleção do padrão rítmico.

No exemplo acima (figura 4.7), exercitamos o trabalho com grupos de quatro posições para a cópia. diminui-se o número de toques pela metade em média, pois podemos ajustar quatro posições com apenas um toque na tela.

Para o menu de padrões, utilizamos grupo de oito posições. No caso do menu, o ganho é maior ao ajustarmos oito posições com dois toques (um para mostrar o menu e outro para selecionar o padrão).

4.4.3 Implementando Song

Descreveremos aqui como criar um controle Song. Conforme descrito no final do item 3.4.4, seguiremos os passos para criação da classe.

4.4.3.1 SongSample

O SongSample exemplifica a implementação de uma classe para criação de Song, com navegação pelo Song a partir de um comboBox (MSDN, 2009-d), e implementação de métodos addPattern, copyPattern e pastePattern da classe Song. É importante lembrar que um Controle Song utiliza controles Pattern através do membro _pattCarregados. A figuras 4.8 e 4.9 abaixo ilustram o controle criado em detalhe e junto com controles Pattern.

Figura 4.8: Um controle SongSample.

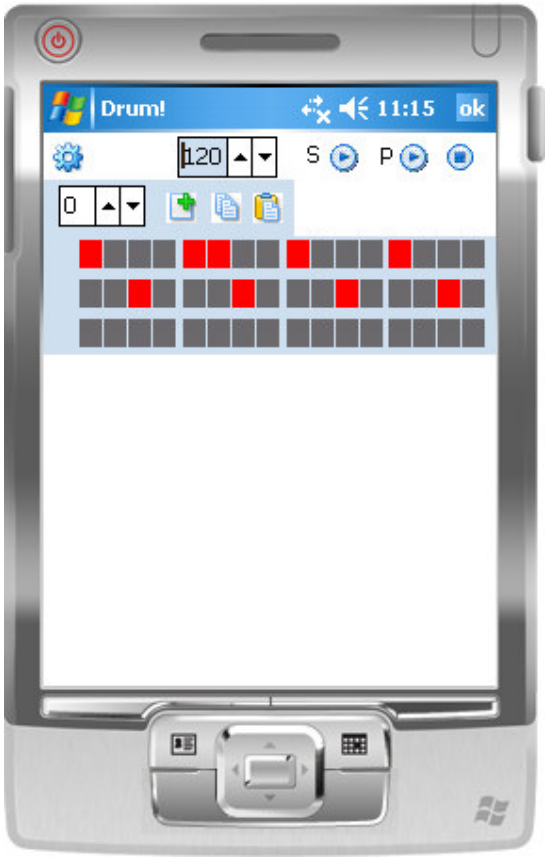

Figura 4.9: De cima para baixo: um baseGUIController, um Song, e três Pattern.

Os componentes da figura 4.8, da esquerda para direita, são:

Navegação: Um combo box é utilizado para navegação entre partes do Song. A posição é indicada pelo número no controle, assim, ao clicar nas setas do controle, podemos avançar ou retroceder no Song.

- Para implementar a navegação, basta acrescentar o seguinte código no evento "valuechanged", que é disparado quando o usuário altera o valor do combo box.

```
_songPos = (int)this.numericUpDown1.Value;
updatePattern();
```

- Assim, basta alterar a posição no Song pelo membro _songPos e chamar o membro updatePattern(), que irá atualizar visualmente os Patterns carregados.

Adiciona: ao ser clicado, copia os dados dos Pattern ao Song na posição indicada pelo controle Navegação. Para isso, basta chamar o método addPattern().

Copia: ao ser clicado, copia os dados dos Pattern para o clipboard interno do Song. Para isso, basta chamar o método copyPattern().

Cola: ao ser clicado, copia os dados do clipboard interno do Song para os Pattern carregados. Para isso, basta chamar o método copyPattern().

4.5 Síntese Deste Capítulo: Resultados da Experimentação

A partir da implementação do framework e a criação de controles Sound, Pattern e Song foi possível constatar que o framework agiliza e facilita a criação de aplicativos musicais. Assim, a experimentação de novas interfaces fica com uma mesma estrutura de dados, permitindo que controles criados por diferentes programadores sejam utilizados em conjunto. O dispositivo mostrou-se capaz de executar os aplicativos com uma boa performance, executando os arquivos de som sem atraso.

5 CONCLUSÕES

Este trabalho abordou o desenvolvimento de aplicativos musicais em dispositivos móveis,um campo de pesquisa emergente que integra computação musical e ubíqua, apresentando novos desafios e possibilidades para composição e execução de peças musicais.

Em particular, foi proposto um framework para desenvolvimento de aplicações futuras e suas principais características e elementos foram apresentados.

Exemplos de aplicação musical foram construídos com o uso do framework e apresentados para ilustrar sua utilidade. Ao criar ferramentas musicais em objetos de consumo do dia a dia como PDAs, leigos em música tem a chance de participar de uma comunidade crescente de praticantes de música. Ao incluir a criação de música como uma extensão de atividades do dia a dia reduz o custo cognitivo de anos de treinamento com interfaces não intuitivas e o conhecimento altamente especializado da prática comum da sintaxe musical. Ao utilizar o suporte proporcionado pelo framework e descrito no texto, nós acreditamos ter pavimentado o caminho para um gama de possibilidades no que tange o uso de dispositivos móveis como interfaces musicais.

Após descrever o cenário de aplicativos musicais em dispositivos móveis no capítulo 2, no capítulo 3 foi proposto um framework um para aplicativos musicais em dispositivos móveis que possibilite uma rápida implementação, facilitando assim a experimentação de novas interfaces para tais aplicativos.

Então no capítulo 4 escolhemos o dispositivo e a linguagem para implementar o framework e descrevemos a implementação e definição de novas interfaces. Pudemos observar que é possível criar novos controles muito rapidamente, com poucas linhas de código. Um Sound básico (item 4.4.1.1), é implementado com duas linhas de código digitadas. Pôde-se observar que utilizando o framework é possível focar apenas na interface, já que a estrutura de dados e métodos de controle já estão implementados, gerando um ganho muito grande no desenvolvimento de novos aplicativos.

A possibilidade de se carregar novos controles em tempo de execução permite implementar aplicativos flexíveis, que permitiriam ao usuário criar novas combinações de interface, adequando-as à sua preferência e habilidade. O framework também gera uma unificação dos controles ao definir uma estrutura de dados única, podendo-se utilizar controles criados por diferentes desenvolvedores.

Focamos o trabalho e experimentação em instrumentos de percussão, porém , caso incluíssemos processamento dos arquivos de som, como modulação (para alterar o tom do som) ou efeitos, como eco e reverberação, seria fácil criar instrumentos musicais mais versáteis. Porém, foi necessário reduzir o escopo do trabalho, para que pudéssemos implementá-lo de forma adequada.

Foi observado que o temporizador do .NET não possui uma precisão absoluta. Na maioria das vezes o processamento se dá no tempo correto, porém algumas vezes ele sofre algum atraso.

Assim, para trabalhos futuros, seria importante:

- Re-escrever a parte que executa o som em .NET versão 3.5, para facilitar a instalação do framework, já que a implementação em C++ precisa ser recompilada para versões diferentes do Windows Mobile.
- Adicionar processamento dos arquivos de som, como modulação e outros efeitos e também permitir maior flexibilidade no gerenciamento dos arquivos, permitindo configurar o modo de execução (se o som será interrompido ao ser mandado executar de novo ou será executado em paralelo, por exemplo).
- Incluir métodos para:

  Exportar Song para arquivo de áudio.

  Gravar execução em arquivo de áudio.

  Exportar para arquivo MIDI.
- Revisar o temporizador e reimplementá-lo para que ele seja mais adequado à aplicativos musicais.
- Estudar a possibilidade de criação de hardware para extensão do dispositivo através do slot Compact Flash. Poderíamos assim acrescentar entradas de áudio para poder conectar instrumentos como guitarras.
- A experimentação de interface se limitou ao *touchscreen* do PDA, porém podemos prever novos tipos de interação, através do microfone, onde o usuário pode cantar uma melodia e o framework traduzir em música (transformando determinados fonemas em sons musicais correspondentes por exemplo) ou utilização de outros dispositivos que poderiam ser conectados via internet ou Bluetooth.

REFERÊNCIAS

BARBOSA, A. Displaced Soundscapes: A Survey of Network Systems for Music and Sonic Art Creation. Leonardo Music Journal, Cambridge, Massachusetts, v.13, p. 53-60, 2003.

BASSOLI, A. et al. TunA: A Mobile Music Experience to Foster Local Interactions. In: INTERNATIONAL CONFERENCE ON UBIQUITOUS COMPUTING, UbiComp'03, 5., 2003, Seattle. Adjunct Proceedings… [S.l.:s.n.], 2003. (Poster.)

BASSOLI, A. et al. TunA: Local Music Sharing with Handheld Wi-Fi Devices. In: WIRELESS WORLD CONFERENCE, 5., 2004, Surrey, UK. Proceedings… [S.l.:s.n.], 2004.

BEHRENDT, F. From Calling a Cloud to Finding the Missing Track: Artistic Approaches to Mobile Music. In: INTERNATIONAL WORKSHOP ON MOBILE MUSIC TECHNOLOGY, 2., May 2005, Vancouver. Proceedings… [S.l.:s.n.].

BRANDON, A. You Can Take It With You. In: MIX, Nov. 2006. Emeryville, EUA: Prism Business Media. p. 70-2.

D'ARCANGELO, G. The New Cosmopolites: Activating the Role of Mobile Music Listeners. Second International Workshop on Mobile Music Technology, May 2005, Vancouver: [s.n.].

DOFACTORY. Pattern Builder,
disponível em <http://www.dofactory.com/Patterns/PatternBuilder.aspx>

DUNLOP, M.; BREWSTER, S. The challenge of mobile devices for human-computer interaction. Personal and Ubiquitous Computing 6(4): 235-6. [S.l.:s.n.], 2002.

DEITEL, H. M.; DEITEL, P. J.; NIETO, T. R.; STEINBUHLER, K. Wireless Internet & Mobile Business: How To Program. Upper Saddle River, NJ: Prentice Hall, 2002.

GAMMA, E. et alli. Padrões de Projeto: Soluções reutilizáveis de Software Orientado a Objetos, Bookman, 2000.

FAGRELL, H.; KRISTOFFERSEN, S.; LJUNGBERG, F. Exploring support for knowledge management in mobile work. In: Proceedings of the Sixth European Conference on Computer-Supported Cooperative Work, Copenhagen, Denmark. Dordrecht: Kluwer Academic Publishers, 1999. p. 259-75.

FIUCZYNSKI, M. E. Mobile Computing at the University of Washington. 1994. Disponível em: <http://www.cs.washington.edu/research/mobicomp/brian.html>. Acesso em: jul. 2007.

FLORES, L. V. Revisão Bibliográfica sobre Recursos de Dispositivos Móveis Visando o Suporte a Atividades Musicais. 2006. Trabalho Individual (Doutorado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

FLORES, L.V., PIMENTA, M.S., KELLER, D. Rumo à Música Ubíqua: Interação via Dispositivos Móveis Cotidianos na Composição Cooperativa e na Eco-Composição. In: 2ND WORKSHOP ON PERVASIVE AND UBIQUITOUS COMPUTING, 2008. UFMS, Campo Grande, Brazil.

FURLANETE, F.; MANZOLLI, J. Interações musicais em rede. In: X BRAZILIAN SYMPOSIUM ON COMPUTER MUSIC, 2005, Belo Horizonte. Proceedings... Belo Horizonte: UFMG / PUC Minas, 2005. p.299-302.

GAYE, L. et al. Sonic City: The Urban Environment as a Musical Interface. In: INTERNATIONAL CONFERENCE ON NEW INTERFACES FOR MUSICAL EXPRESSION, NIME03, 3., May 2003, Montréal. Proceedings… Montréal: [s.n.], 2003. p. 109-15.

GAYE, L. et al. Mobile Music Technology: Report on an Emerging Community. In: INTERNATIONAL CONFERENCE ON NEW INTERFACES FOR MUSICAL EXPRESSION, NIME06, 6., June 2006, Paris. Proceedings… Paris: IRCAM, 2006. p. 22-5.

GAYE, L.; HOLMQUIST, L. E. In Duet with Everyday Urban Settings: A User Study of Sonic City. In: INTERNATIONAL CONFERENCE ON NEW INTERFACES FOR MUSICAL EXPRESSION, NIME04, 4., June 2004, Hamamatsu, Japan. Proceedings… Hamamatsu, Japan: Shizuoka University of Art and Culture, 2004. p. 161-4.

HÅKANSSON, M. et al. Designing a Mobile Music Sharing System Based on Emergent Properties. In: INTERNATIONAL CONFERENCE ON ACTIVE MEDIA TECHNOLOGY, AMT 2005, 3., 2005, Takamatsu, Japan. Proceedings… [S.l.:s.n.], 2005.

HENTSCHKE, L. Um Estudo Longitudinal Aplicando a Teoria Espiral de Desenvolvimento Musical de Swanwick com Crianças Brasileiras da Faixa Etária de 6 a 10 Anos de Idade: Pólo Porto Alegre – 1994. Música: Pesquisa e Conhecimento, Porto Alegre: CPG em Música – Mestrado e Doutorado / UFRGS – NEA, n.2, p. 9-34, 1996.

HORSTMAN, C; Padrões e Projeto Orientados a Objeto, Bookman, 2007.

INFOMUS LAB. NIME 2008. Disponível em: <http://nime2008.casapaganini.org/> e em: <http://www.nime.org/2008>. Acesso em: jul. 2007.

INTERNATIONAL SYMPOSIUM ON HANDHELD AND UBIQUITOUS COMPUTING, HUC'99, 1., September 1999, Karlsruhe, Germany. Proceedings... Berlin: Springer Verlag, 1999. (Lecture Notes in Computer Science, v.1707.)

JACOBSSON, M. et al. Push!Music: Intelligent Music Sharing on Mobile Devices. In: INTERNATIONAL CONFERENCE ON UBIQUITOUS COMPUTING, UbiComp 2005, 7., 2005, Tokyo, Japan. Adjunct Proceedings… [S.l.:s.n.], 2005. (Demonstration.)

LEE, E. Past NIME conferences. Disponível em: <http://www.nime.org/pastnimes.html>. Acesso em: jul. 2007-d.

MAISONNAVE, F. Z. Estudo Comparativo entre Plataformas de Desenvolvimento e Execução de Aplicações para Dispositivos Móveis. 2004. 75 f. Projeto de Diplomação (Bacharelado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

MELOTECA, Reco-Reco, disponível em <http://www.meloteca.com/dicionario-instrumentos.htm#r> acesso em jan-2009.

MOBILEMUSICWORKSHOP.ORG, 5th International Mobile Music Workshop – CFP, disponível em <http://www.uni-ak.ac.at/personal/kirisits/mmw2008/papers.html> acesso em jun-2009.

MSDN, .NET Framework Developer Center, disponível em <http://msdn.microsoft.com/en-us/library/system.windows.forms.usercontrol.aspx> acesso em jun-2009-a.

MSDN, .NET Framework Developer Center, disponível em <http://msdn.microsoft.com/en-us/library/system.windows.forms.panel.aspx> acesso em jun-2009-b.

MSDN, .NET Framework Developer Center, disponível em <http://msdn.microsoft.com/en-us/library/system.windows.forms.picturebox.aspx> acesso em jun-2009-c.

MSDN, Multimedia Functions, disponível em <http://msdn.microsoft.com/en-us/library/ms712636(VS.85).aspx> acesso em jun-2009-d

MSDN, HashTable, disponível em <http://msdn.microsoft.com/en-us/library/system.collections.hashtable.aspx> acesso em jun-2009-e

ORGANISED SOUND, Cambridge, UK, v.10, n.3 [edição sobre Networked Music], December 2005.

POPE, S. T. The State of the Art in Sound and Music Computing. In: WEEKLY COMPUTER SCIENCE COLLOQUIUM, Feb. 1996, UCSB, Berkeley. (Presentation.)

POPE, S. T. Editor's Note: A Taxonomy of Computer Music. Computer Music Journal, Cambridge, Massachusetts, v.18, n.1, Spring 1994.

SCHIEMER, G.; HAVRYLIV, M. Pocket Gamelan: A Blueprint for Performance using Wireless Devices. In: INTERNATIONAL COMPUTER MUSIC CONFERENCE, ICMC 2005, Barcelona, Spain. Proceedings… Barcelona: ICMA, 2005. p. 600-3.

SCHROTT, G.; GLÜCKLER, J. What makes mobile computer supported cooperative work mobile? Towards a better understanding of cooperative mobile interactions. International Journal on Human-Computer Studies 60: 737-52. [S.l.:] Elsevier, 2004.

SILVA, C. A. da. Padrões de Interface para Dispositivos Móveis. 2006. 53 f. Projeto de Diplomação (Bacharelado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

SSEYO. SSEYO miniMIXA V2. Disponível em http://www.sseyo.com/products/miniMIXA/index.html. Acesso em maio de 2006.

TANAKA, A. Mobile Music Making. In: INTERNATIONAL CONFERENCE ON NEW INTERFACES FOR MUSICAL EXPRESSION, NIME04, 4., June 2004, Hamamatsu, Japan. Proceedings… Hamamatsu, Japan: Shizuoka University of Art and Culture, 2004. p. 154-6.

TAO GROUP. intEnt Audio. Disponível em <http://tao-group.com/main.php?pageid=358674.php&temptype=t1&pagename= intent%20Audio>. Acesso em maio de 2006.

UEDA, L. K.; KON, F. Andante: Composition and Performance with Mobile Musical Agents. Proceedings of the International Computer Music Conference 2004, November 2004, Miami. [S.l.:s.n.].

UEDA, L. K.; KON, F. Andante: A Mobile Musical Agents Infrastructure. Proceedings of the IX Brazilian Symposium on Computer Music, August 2003, Campinas. [S.l.:s.n.] p.87-94.

UBICOMP. Previous Events. Disponível em: <http://www.ubicomp.org/previous_conferences.html>. Acesso em: jul. 2007.

WEINBERG, G. The Aesthetics, History, and Future Challenges of Interconnected Music Networks. In: INTERNATIONAL COMPUTER MUSIC CONFERENCE, 2002, Göteborg, Sweden. Proceedings… [S.l.]: ICMA, 2002.

WEISER, M. Ubiquitous Computing. 1996. Disponível em: <http://www.ubiq.com/hypertext/weiser/UbiHome.html>. Acesso em: jul. 2007.

WIKIPEDIA. Handheld Device. Disponível em: <http://en.wikipedia.org/wiki/Handheld_device>. Acesso em: jul. 2007-a.

WIKIPEDIA. Mobile Computing. Disponível em: <http://en.wikipedia.org/wiki/Mobile_computing>. Acesso em: jul. 2007-b.

WIKIPEDIA. Mobile Music. Disponível em: <http://en.wikipedia.org/wiki/Mobile_music>. Acesso em: jul. 2007-c.

WIKIPEDIA. Java Swing. Disponível em: <http://en.wikipedia.org/wiki/Swing_(Java)>. Acesso em jul. 2009.

YAMAUCHI, T.; IWATAKE, T. Mobile User-Interface For Music. In: INTERNATIONAL WORKSHOP ON MOBILE MUSIC TECHNOLOGY, 2., May 2005, Vancouver. Proceedings… [S.l.:s.n.].

APÊNDICE: TUTORIAL/MANUAL DE USO

O Visual Studio permite que se instalem templates, que são projetos pré-definidos. Assim, incluímos para uso alguns controles controles descritos no item 4.4. São eles: SimpleSoundSample, RecoReco, SongSample, PatternSample. Além dos controles descritos, incluímos: baseGUIApplication, um aplicativo que servirá de base para inclusão de controles, contendo o controle baseGUIController.

A-Pré-requisitos:

É necessário que o Visual Studio seja versão 2008 Professional, com o .NET Compact Framework 3.5 instalado.

B-Instalando os templates no Visual Studio:

Para instalar os templates, basta descompactar o arquivo baseGUI-Templates.zip que está no CD anexo na pasta 'meus documentos\Visual Studio 2008', que é a pasta onde o Visual Studio salva os projetos e configurações.

C-Utilizando os templates

Após a instalação, deve-se:

1-Inicializar o Visual Studio

2-Abrir novo projeto em File/new/Project

3-Aparecerá a tela da figura C1.

4- No campo “Project types” (acima a esquerda) selecione Visual C#/Smart Device

5-Escolha o template à direita abaixo de “my templates”

6-Ajuste dos demais campos:

'name':Nome do projeto. Este nome será utilizado para formar o namespace do projeto.

'Location': A pasta onde o projeto será criado(uma pasta com o nome do projeto será criada dentro desta selecionada.

'Solution Name': Solution(Solução) é um conjunto de projetos.

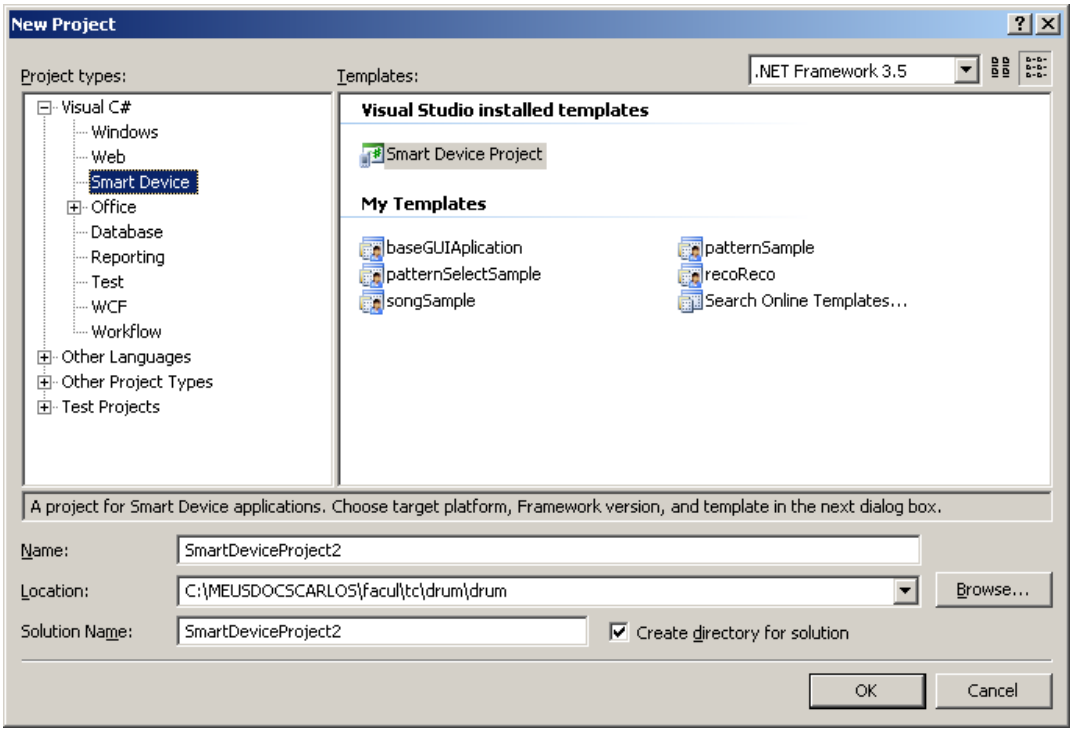

Figura C1: Criando novo projeto.