**UNIVERSIDADE REGIONAL DE BLUMENAU**

**CENTRO DE CIÊNCIAS EXATAS E NATURAIS**

**CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO**

# BIBLIOTECA DE INTERFACE GRÁFICA PARA CELULARES

**MARCOS GORLL**

**BLUMENAU**
**2007**

**2007/2-26**

**MARCOS GORLL**

# BIBLIOTECA DE INTERFACE GRÁFICA PARA CELULARES

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

Prof. Paulo César Rodacki Gomes, Doutor - Orientador

**BLUMENAU**
**2007**

**2007/2-26**

# BIBLIOTECA DE INTERFACE GRÁFICA PARA CELULARES

Por

**MARCOS GORLL**

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

| | |
|---|---|
| Presidente: | Prof. Paulo Césasr Rodacki Gomes, Doutor – Orientador, FURB |
| Membro: | Prof. Francisco Adell Péricas, Mestre – FURB |
| Membro: | Prof. Dalton Solano dos Reis, Mestre – FURB |

Blumenau, 07 de dezembro de 2007

Dedico este trabalho a todos os amigos, especialmente aqueles que me ajudaram diretamente na realização deste, ao orientador que soube me ajudar, a empresa que permitiu tempo extra para dedicação ao presente trabalho.

## AGRADECIMENTOS

À minha família sempre esteve presente.

Aos meus amigos, pelos empurrões e cobranças.

Ao meu orientador, Paulo César Rodacki Gomes, por ter me ajudado em todas as necessidades e novas idéias.

Uma pessoa somente fala o que pensa, quando não pensa no que fala.

Autor desconhecido

# RESUMO

Este trabalho apresenta a especificação e implementação de uma biblioteca gráfica para celulares, baseada da especificação *Java* 2 *Micro Edition* (J2ME) . A API contempla a criação de interfaces baseadas em componentes gráficos, como botões e caixas de textos. Possui, ainda, sistema de controle de eventos e gerenciadores de *layout.* Este trabalho demonstra a necessidade cada vez maior de mobilidade dentro dos sistemas de informação, e assim, a necessidade de novas ferramentas para auxílio no desenvolvimento de softwares na área móvel.

Palavras-chave: Interface. API gráfica. Biblioteca gráfica. Dispositivos móveis. J2ME.

# ABSTRACT

This work presents the specification and implementation of a library for mobile graphics, based specification of the Java 2 Micro Edition (J2ME). The API includes the creation of interfaces based on graphical components, such as buttons and textfields. It is, still, system of control of events and managers layout. This work demonstrates the growing need for mobility within the information systems, and thus the need for new tools to aid in the development of software in the mobile area..

Key-words: Interface. GraphicsAPI. Graphics library. Móbile device. J2ME.

# LISTA DE ILUSTRAÇÕES

# LISTA DE SIGLAS

API – *Application Programming Interface*

AWT – *Abstract Window Toolkit*

CDC – *Connected Device Configuration*

CLDC – *Connected Limited Device Configuration*

HTML – *Hyper Text Markup Language*

HTTP – *HyperText Transport Protocol*

HTTPS – *HyperText Transport Protocol Secure*

J2ME – *Java* 2 *Micro Edition*

J2SE – *Java* 2 *Standart Edition*

JTGL – *Java Tiny Gfx Library*

JVM – *Java Virtual Machine*

M3G – *Mobile 3D Grapics*

MIDP – *Mobile Information Device Profile*

PDA – *Personal Digital Assistant*

UML – *Unified Modeling Language*

XML – eXtensible Markup Language

# SUMÁRIO

# 1 INTRODUÇÃO

A evolução da indústria tecnológica vem acelerando, trazendo consigo novas possibilidades para a produção do software. Uma destas possibilidades, talvez a mais importante, é a mobilidade, ou seja, o uso dos sistemas de informação em equipamentos portáteis e móveis, como *Personal Digital Assistant* (PDA) e celular.

Como todo recurso inexplorado, este também pode trazer maior complexidade na construção dos sistemas. Para resolver este problema, nada melhor do que utilizar os novos recursos e possibilidades, mantendo a mesma forma de desenvolvimento, ou seja, não se deixando perder parte do conhecimento adquirido pelos desenvolvedores de software ao passar do tempo.

Um, entre muitos problemas encontrados ao se criar softwares para dispositivos móveis, fica a cargo da interface gráfica, interface esta que fará a comunicação entre o usuário final e o sistema.

O desenvolvimento de *software* para dispositivos móveis tem dificuldades relacionadas a quantidade de recursos, tanto na forma de *hardware*, pouca memória disponível e baixo poder de processamento, como na parte de *software,* poucas APIs e poucos *frameworks* de desenvolvimento disponíveis. Como tentativa de resolver parte da dificuldade no desenvolvimento de aplicativos para dispositivos móveis, relacionada à diferença entre a programação para *desktop* e programação para os dispositivos móveis, a empresa Sun Microsystems criou uma especificação de máquina virtual para dispositivos móveis.

O desenvolvimento para dispositivos móveis é muito diferente do desenvolvimento de sistemas para *desktop* (HUOPANIEMI, 2003, p. 107). Estas diferenças estão relacionadas principalmente a forma de desenvolvimento dos sistemas embarcados, devido às diferenças entre as APIs disponíveis para *desktop* e para dispositivos móveis. Estas diferenças se encontram em todas as partes do desenvolvimento, mas tornam-se mais perceptíveis na criação da interface gráfica com o usuário, onde o uso dos componentes e a implementação dos eventos diferem da forma de desenvolvimento para *desktop*. A idéia, portanto, é criar uma biblioteca que tenha foco somente na interface gráfica, e que tenha funcionamento o mais próximo possível da *Application Programming Interface* (API) gráfica Java *Abstract Window Toolkit* (AWT) para computadores hoje.

Alguns dispositivos móveis, como computadores de mão, já possuem exemplos de APIs com características semelhantes. Sendo assim, optou-se por desenvolver uma biblioteca

somente para celulares, com o intuito de criar uma plataforma padrão para desenvolvimento que atenda a todos os dispositivos que suportem os requisitos da plataforma.

## 1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver uma biblioteca de interface gráfica para celulares, tendo o funcionamento dos eventos e componentes o mais próximo possível ao funcionamento da API Java AWT.

Os objetivos específicos do trabalho são:

a) definir e implementar um conjunto de classes para a criação dos componentes de tela os quais são `panel`, `radiobutton`, `checkbox`, `button`, `textbox`, `combobox`, `label` e `menu`;
b) definir e implementar um controle de eventos de interface;
c) definir e implementar gerenciadores de disposição de componentes, ou seja, gerenciadores de *layout*;
d) criar uma aplicação piloto para teste dos componentes, eventos e gerenciador de *layout*;
e) integrar a API com o *mobile 3D graphics* (M3G), API gráfica 3D para celulares.

## 1.2 ESTRUTURA DO TRABALHO

A estrutura deste trabalho está apresentada em capítulos, divididos em cinco principais temas. O primeiro capítulo apresenta uma visão sobre dispositivos móveis e tecnologias disponíveis. O segundo capítulo aborda de forma mais profunda a tecnologia utilizada no trabalho, J2ME e suas particularidades. A especificação, implementação, desenvolvimento e metodologia são apresentados no terceiro capítulo. E, por final, são apresentadas a conclusões e discussões sobre o trabalho no quarto e último capítulo.

# 2 DISPOSITIVOS MÓVEIS

"Mobilidade é o termo utilizado para identificar dispositivos que podem ser operados a distância ou sem fio. Dispositivos que podem ser desde um simples BIP, até os mais modernos Pocket PC's" (MOSIMANN NETTO, 2005).

Como exemplo destes dispositivos tem-se desde Pocket PC's até celulares, passando pelos *smartphones*, dispositivos estes que estão cada vez mais recebendo atenção por parte das empresas desenvolvedoras de softwares. Segundo Huopaniemi (2003), o desenvolvimento para dispositivos móveis é muito diferente do desenvolvimento de sistemas para *desktop*. Estas diferenças estão tanto ao *hardware*, onde os recursos disponíveis são menores, como a parte de *software* onde as APIs são diferentes das APIs para *desktops*, o que sugere alguma dificuldade para os desenvolvedores de software com alvo nos dispositivos móveis. Estas dificuldades estão ligadas a todas as etapas do processo de desenvolvimento, onde o desenvolvedor deve aprender a utilizar as APIs existentes, e assim, não poderá utilizar o conhecimento adquirido com o desenvolvimento de sistemas para *desktop*. Estas diferenças são apresentadas em grau maior no desenvolvimento da interface gráfica para o usuário do sistema, onde a criação de componentes e tratamento de eventos difere das APIs convencionais para desenvolvimento para *desktop.*

Como estes dispositivos possuem um menor poder de processamento e menor quantidade de memória disponível, e por conseqüência, menos recursos, a empresa Sun MicrosSystems, desenvolvedora da linguagem de programação JAVA, criou uma especificação da linguagem JAVA para dispositivos móveis, chamada J2ME, ou *Java* 2 *Micro Edition*. Esta especificação contempla desde a *Java Virtual Machine* (JVM) até as classes que estão disponibilizadas na API para uso dos desenvolvedores.

Mesmo dentro da especificação J2ME foi necessário criar subdivisões para atender a grande gama de dispositivos diferentes. Estas subdivisões são chamadas configurações. Segundo Topley (2002, p. 10, tradução nossa), "uma configuração é uma especificação que define o ambiente de software para uma parte de dispositivos definidos por algumas características, que normalmente são memória disponível, poder de processamento e formas de conexão". As configurações disponíveis são:

a) *Connected Device Configuration* (CDC);

b) *Connected Limited Device Configuration* (CLDC).

A configuração CDC é a especificação para dispositivos que possuem maior poder de

processamento, mais memória disponível e mais recursos, já a configuração CLDC é a especificação para os dispositivos mais pobres em recursos, processamento e memória. Como exemplos temos Pocket PC's e celulares para as configurações CDC e CLDC respectivamente.

Dentro da especificação da configuração CLDC existem recursos para desenvolvimento de um sistema completo, porém, recursos da parte de interface gráfica tornam o desenvolvimento custoso tanto em tempo quanto em necessidade de aprendizado. Visando minimizar este problema, a idéia é criar uma API de interface gráfica que permita ao desenvolvedor utilizar o conhecimento adquirido do desenvolvimento para *desktop* e que possua recursos interessantes não disponibilizados pela especificação J2ME.

Além da divisão por parte das configurações, dentro da especificação J2ME, existem pacotes chamados *profiles*.

> *Profile* é uma extensão de uma configuração. Ele provê as bibliotecas para o desenvolvedor escrever aplicações para um tipo particular de dispositivo. Por exemplo, *Mobile Information Device Profile* (MIDP) define APIs para uso de componentes de interface, tratamento de eventos, persistência de dados, comunicação e redes, levando em consideração as limitações dos dispositivos móveis. (MUCHOW, 2001, p. 19).

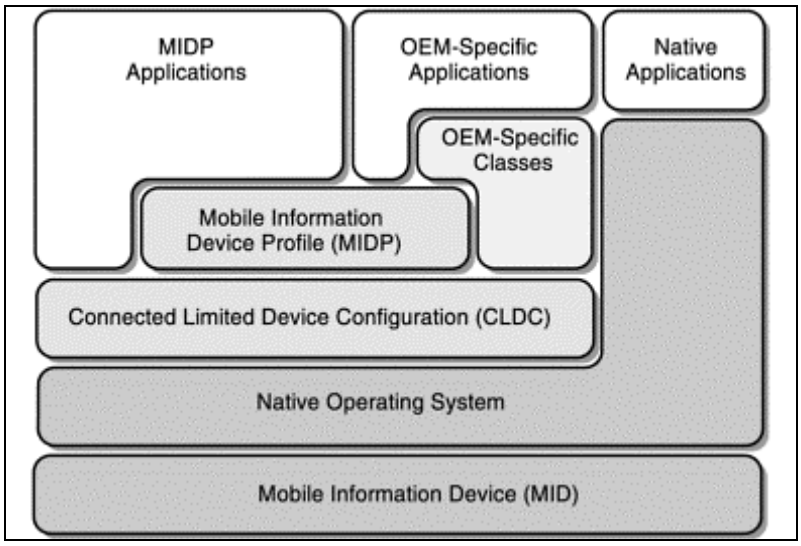
## 2.1 CONFIGURAÇÃO CLDC

Conforme Muchow (2001, p. 18, tradução nossa), “uma configuração define uma plataforma para uma gama de dispositivos. Uma configuração está amarrada à máquina virtual. Na verdade, uma configuração define os recursos que a linguagem JAVA e as bibliotecas do núcleo que estarão disponíveis”. Hemphill e White (2002, p. 25) explicam que o maior objetivo da configuração CLDC é funcionar sobre os dispositivos com menor poder de processamento e menor quantidade de memória disponível. Assim, esta configuração está destinada para aplicativos do porte de celulares, por exemplo.

Esta configuração é responsável por disponibilizar a maioria dos recursos existentes para os desenvolvedores de software.

A figura 1 mostra a divisão da plataforma J2ME, onde se podem ver as diferentes camadas que constituem a arquitetura J2ME. A camada base é identificada pelo *hardware* do dispositivo, ou seja, o celular ou PDA. A camada logo acima do dispositivo é o sistema operacional disponível no dispositivo, após a camada do sistema operacional existe a camada

da arquitetura J2ME que faz a ponte entre a linguagem JAVA para dispositivos móveis e o sistema operacional. As camadas acima da configuração CLDC podem ser consideradas extensões da linguagem JAVA para dispositivos móveis, nestas camadas se encontram o *profile* MIDP, as APIs e *frameworks* disponibilizados pelos fabricantes dos dispositivos.




Fonte: Muchow (2001, p. 19).

Figura 1 – Divisões da arquitetura J2ME, com relação aos principais componentes

## *2.2* PROFILE MIDP

Hemphill e White (2002, p. 23) argumentam que um *profile* adiciona funcionalidades a uma configuração disponível, como as configurações CDC ou CLDC. Comentam ainda que estas funcionalidades sejam mais específicas do que as funcionalidades provenientes da própria configuração, sendo assim, mais restrita entre os hardwares disponíveis.

Segundo Hemphill e White (2002, p. 32), MIDP foi o primeiro *profile* liberado pela Sun Microsystems, e tem como dispositivos alvo os celulares.

O *profile* MIDP traz consigo os recursos de interface gráfica para com o usuário, mas o seu principal foco está em prover conectividade. Este conjunto de classes disponibiliza recursos para conexão *HyperText Transfer Protocol* (HTTP), *HyperText Transfer Protocol Secure* (HTTPS) e conexões seriais através de *sockets*, além de prover recursos para a utilização de certificados digitais.

## *2.3* MOBILIDE 3D GRAPHICS

*Mobile* 3D *Graphics* (M3G) é um *framework* para dispositivos de pequeno porte, ou seja, baixo consumo de memória e exige pouco poder de processamento, mas, é flexível o suficiente para poder ser suportado por um grande número de dispositivos (SUN MICROSYSTEMS, 2006b).

O M3G tem como objetivo proporcionar os recursos necessários para o desenvolvimento de gráficos 3D interativos, tais como mensagens animadas, extensão de componentes de interface, proteções de tela, entre outros.

Utilizando-se estes recursos, o desenvolvimento de componentes gráficos torna-se mais fácil, além de mais padronizado, uma vez que esta especificação é adotada na produção dos celulares.

Os principais recursos disponibilizados pelo *framework* estão relacionados à utilização de texturas e imagens, à criação de malhas de polígonos para dar suporte a criação de motores gráficos, jogos, mapas e mensagens animadas. Os requisitos para o suporte ao M3G é o suporte a especificação J2ME CLDC e suporte ao *profile* MIDP.

## 2.4 AWT

Tendo em vista a idéia de padronizar o desenvolvimento da parte gráfica, além de fazer com que o desenvolvimento desta parte fique o mais próxima possível do desenvolvimento da interface gráfica para *desktop* na linguagem JAVA, é necessário espelhar o funcionamento da API em alguma já existente. Para tanto, foi escolhida a AWT, por ser a API gráfica base da empresa Sun.

Conforme Sun Microsystems (2006a, tradução nossa), “AWT é uma API para desenvolvimento de interfaces gráficas”. Assim sendo, esta API tem o foco em desenvolvimento de interfaces com o usuário. Para cumprir este objetivo, a API possui uma gama de componentes gráficos, como botões, caixas de texto, entre outros. Segundo Sun Microsystems (2006a, tradução nossa), “os principais recursos da biblioteca são: grande conjunto de componentes de interface, sistema robusto de eventos, suporte a gráficos e imagens, controladores de *layout* e classes auxiliares para transferência de dados, utilizadas

nos recursos de copiar e colar dados".

## 2.5 JAVADOC

Conforme Sun Microsystems (2007a, tradução nossa), "JavaDoc é uma ferramenta para gerar documentação no formato HTML baseado nos comentários e documentações no código fonte".

Assim, os comentários em atributos de classes, métodos, construtores e cabeçalho da classe são processados e transformados em um documento HTML, facilitando a disponibilização de toda a documentação, além de fornecer uma ferramenta prática para a geração de documentos, não existindo a necessidade de dar manutenção no documento final, apenas manter a documentação no próprio código fonte.

## 2.6 TRABALHOS CORRELATOS

Algumas bibliotecas desempenham papel semelhante ao proposto neste trabalho, cada qual com as suas particularidades. Dentre elas foram selecionadas: *Java Tiny Gfx Library* (JTGL) e Thinlet.

### 2.6.1 JTGL

JTGL é uma biblioteca gráfica de código aberto. Tem por objetivo tornar possível a portabilidade entre os mais diferenciados ambientes, como por exemplo, celulares que utilizam o ambiente MIDP ou computadores de mão, que utilizam da configuração CDC para suportar o JTGL (MRMX, 2006).

A JTGL possui uma gama de recursos que abstrai as particularidades de cada ambiente, tornando assim, o uso para o desenvolvedor igual, mesmo na implementação para celulares ou para PDAs. Assim sendo, a própria biblioteca garante a portabilidade entre os diversos dispositivos suportados, não cabendo ao desenvolvedor a responsabilidade de manter

o código funcional entre as diferentes arquiteturas, como celulares e PDAs.

É relevante afirmar que não existem modificações substancias entre o código implementado para um ambiente, quando portado para outro também suportado pela API. Esta biblioteca foi especialmente desenhada para ter baixo consumo de memória e ser altamente portável, ou seja, rodar em vários dispositivos sem existir alteração no código (MRMX, 2006).

As funcionalidades gerais são: recursos para desenvolvimento de jogos e recursos para desenvolvimento de interface gráfica com o usuário. Pode-se, ainda, dividir a biblioteca nos seguintes quesitos:

a) gráficos: primitivas, cores e imagens, ou seja, recursos mais básicos como classes para trabalho com cores, leitura de imagens, etc.;
b) *framework* de extensão leve: dispositivos de mídia, como câmeras de celulares, entre outros;
c) *framework* genérico para entrada de dados: *mouse* e teclado;
d) implementações específicas para cada ambiente: AWT, MIDP, etc. trazendo assim a portabilidade, que é o objetivo da API;
e) recursos para desenvolvimento de jogos, como um `canvas` com suporte a câmeras e movimentação de objetos.

### 2.6.2 THINLET

Thinlet é um conjunto de recursos, ou seja, uma biblioteca que tem por objetivo disponibilizar funcionalidades para a construção de interface gráfica com o usuário. É portável para qualquer ambiente que suporte a API gráfica AWT, ou seja, é funcional em ambientes CDC e *Java* 2 *Standart Edition* (J2SE) (BAJZAT, 2006).

Seu funcionamento é baseado na escrita de arquivos *eXtensible Markup Language* (XML), que são a matriz da composição dos componentes de interface. Pode-se dividir o seu funcionamento da seguinteforma :

a) escrita do arquivo XML referente a interface, arquivo este que é a implementação da tela em si. No caso de Thinlet, apenas a execução dos eventos fica implementado em classes;
b) motor central da biblioteca faz a leitura do arquivo;
c) motor traduz os dados vindos da leitura em componentes de interface Thinlet.

Neste ponto, o motor cria os componentes de tela;

d) componentes são apresentados utilizando a API AWT.

# 3 DESENVOLVIMENTO DA API

O desenvolvimento deste trabalho consiste em uma API para desenvolvimento de interfaces gráficas com o usuário para a plataforma J2ME, visando os aparelhos de celulare. O objetivo é suprir a necessidade pelo recurso gráfico em ambientes móveis, agregando as funcionalidades de APIs existentes para *desktop*, bem como, permitir ao desenvolvedor utilizar do seu conhecimento prévio no desenvolvimento de interfaces gráficas para *desktop.*

Para agregar mais valor a API bem como deixá-la mais genérica, e assim, disponível para um maior número de dispositivos, nenhum *framework* de auxílio à criação dos componentes foi utilizada, ou seja, todo o trabalho de criar e desenhar os componentes ficou a cargo da própria API implementada.

Como instrumento de modelagem do sistema para descrever o projeto, optou-se por trabalhar com a *Unified Modeling Language* (UML).

## 3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

A seguir estão dispostos os Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF) pertinentes a API desenvolvida.

Os requisitos são:

a) possuir os componentes básicos de interface gráfica, ou seja, `panel`, `radiobutton`, `checkbox`, `button`, `textbox`, `combobox`, `label` e `menu` (RF);
b) possuir sistema de eventos que comporte os seguintes eventos: `focuslost`, `focusgained`, `keydown` e `keyup` (RF);
c) ter suporte a gerenciamento de *layout* tabular (RF);
d) ser desenvolvido em JAVA (RNF);
e) ter baixo consumo de memória, não ultrapassando 500 kilobytes (RNF);
f) possuir integração com a API M3G para a criação dos componentes (RNF);
g) possuir documentação em forma de JavaDoc (RNF);
h) possuir forma de desenvolvimento parecida com a API AWT (RNF).

## 3.2 ESPECIFICAÇÃO

Na especificação técnica da API é apresentado, na forma de diagramas UML, o diagrama de casos de uso, diagrama de seqüência e diagrama de classes. Algumas especificações, contanto, serão feitas em forma textual. A ferramenta utilizada para a diagramação foi *Enterprise Architect.*

### 3.2.1 Diagrama de Caso de Uso

Os casos de uso são usados para representar a interação entre o usuário com o sistema, bem como as funcionalidades do sistema.

Um caso de uso é a descrição do comportamento do sistema do ponto de vista do usuário. Para os desenvolvedores, os casos de uso são uma ferramenta muito útil, pois eles podem ser considerados uma técnica do tipo tentativa e erro para obter os requisitos do sistema a partir da visão do cliente (MACORATTI, 2006).

Pode-se compreender um caso de uso como a descrição de uma coleção de cenários de sucesso ou falha que descrevem um determinado processo do sistema com a finalidade de atender um objetivo do usuário (XEXEO, 2007, p. 228).

Como este trabalho é uma API para desenvolvimento de interfaces gráficas, o usuário deve ser considerado como o desenvolvedor dos sistemas. Apresentam-se na figura 2, da perspectiva do usuário, as funcionalidades oferecidas. O diagrama abrange apenas as áreas mais importantes do trabalho.

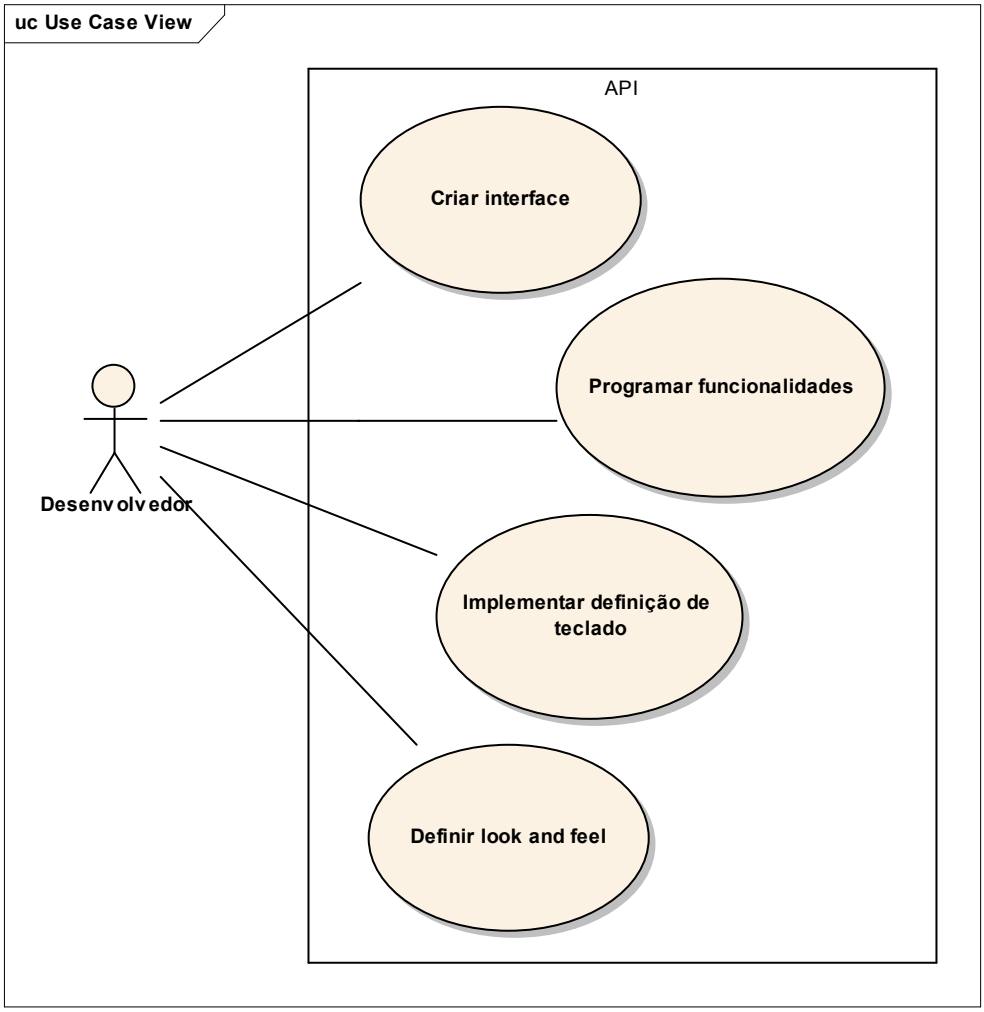



Figura 2 – Diagrama do caso de uso da API

#### 3.2.1.1 Caso de uso Criar interface

Neste caso de uso o usuário-desenvolvedor desenvolve a sua interface com o usuário final, ou seja, o desenvolvedor adiciona os componentes, criando a sua tela. A API carrega todos os componentes, lê os atributos de cada componente e desenha os componentes conforme os parâmetros passados pelo desenvolvedor.

Os valores dos atributos dos componentes correspondem a visibilidade, se o componente está ou não habilitado, tamanho e disposição de cada componente na tela. A API, então, aplica o *layout* escolhido pelo desenvolvedor sobre cada componente.

### 3.2.1.2 Caso de uso Programar funcionalidades

No caso de uso programar funcionalidades, o usuário-desenvolvedor implementa todos os eventos de interface relevantes a sua necessidade, ou seja, o desenvolvedor aplica a sua lógica de negócio na interface criada. O desenvolvedor, então, programa os seus eventos nos componentes gráficos.

A API, por usa vez, é responsável por receber os eventos disparados pelo usuário final, tratá-los e efetuar as chamadas necessárias para que cada evento alcance a implementação criada pelo usuário desenvolvedor.

A figura 3 representa na forma de um diagrama de seqüências, o funcionamento da execução de um evento, desde o disparo pelo usuário final, até o tratamento por parte do desenvolvedor.

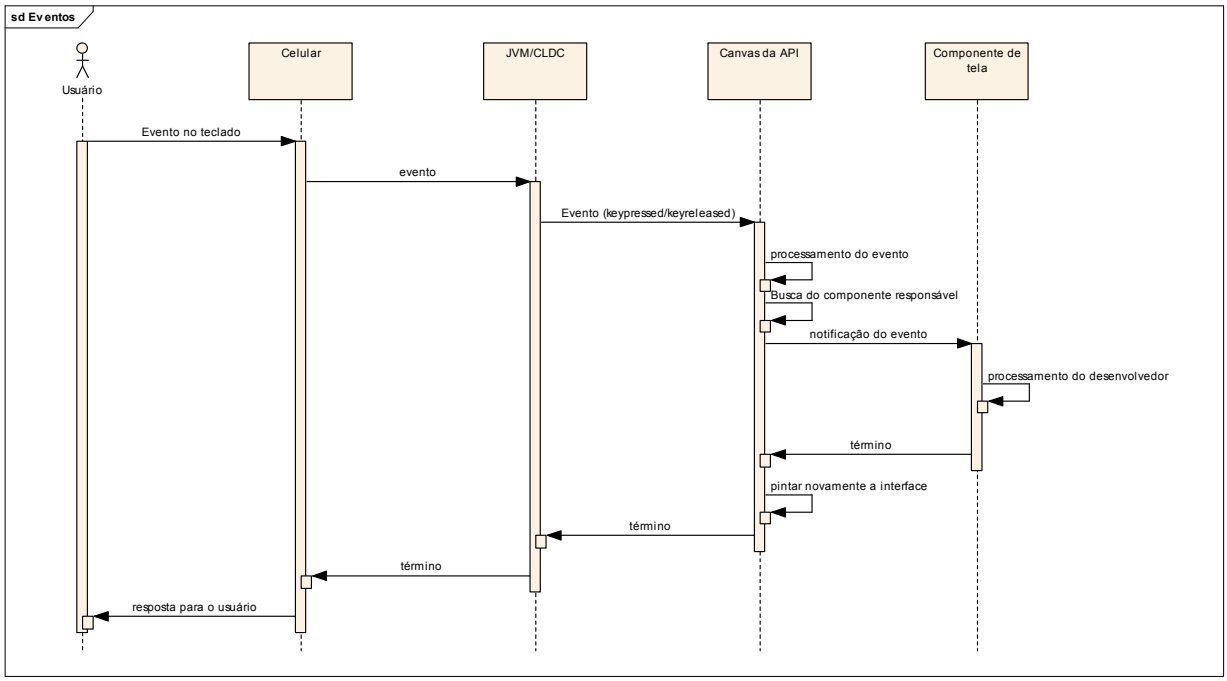



Figura 3 – Diagrama de seqüência do funcionamento de eventos da API

Seguindo os passos descritos na figura 3, o usuário dispara algum evento através de interação com o teclado do dispositivo, passado para o celular como dispositivo, no próximo passo este evento alcança a JVM do dispositivo, o evento é passado para a API, neste ponto o evento é processado para, então, descobrir qual o componente responsável pelo evento, ao final deste passo o evento é passado para o componente do qual se originou o evento e então será executada toda a implementação feita pelo usuário-desenvolvedor.

### 3.2.1.3 Caso de uso Implementar definição de teclado

Cada celular possui diferenças e particularidades em seu teclado, e como não poderia ser diferente, não há um padrão ou normalização para os valores das teclas recebidas através dos eventos. Assim optou-se por criar uma forma do desenvolvedor criar e desenvolver a sua própria especificação de teclado, conseguindo assim a flexibilidade de funcionar em qualquer dispositivo móvel do tipo celular.

A figura 4 representa, através de um diagrama de classes, a classe que deve ser estendida e os métodos que devem ser reimplementados para o desenvolvimento de uma definição de teclado.

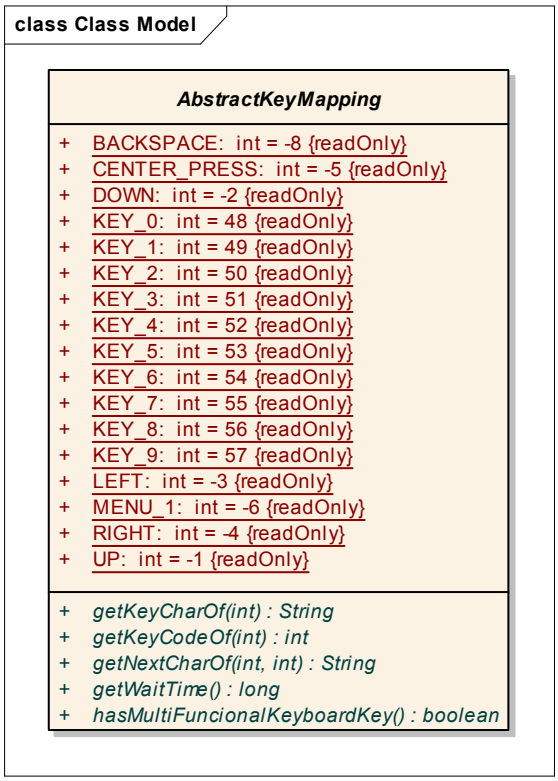



Figura 4 – Classe que especifica um mapa de teclado

Assim, se faz necessária a implementação dos seguintes métodos:

a) `getKeyCharOf(int key)`: este método tem por objetivo normalizar os códigos das teclas vindos de diferentes dispositivos. O retorno do método corresponde ao valor da tecla;

b) `getKeyCodeOf(int key)`: este método tem por objetivo normalizar o próprio código da tecla. O retorno do método corresponde as constantes pré-definidas na classe pai;

c) `getNextCharOf(int key, int pos)`: este método é necessário uma vez que em

dispositivos como celulares a mesma tecla possui mais de um valor, assim o retorno do método será o próximo valor da tecla em relação à posição passada como parâmetro;

d) `getWaitTime()`: este método retorna o tempo em milisegundos que a API deverá esperar antes de mudar o cursor para a próxima posição. Isso se faz necessário em teclados que possuem mais de uma função, quando é preciso executar várias vezes a ação até chegar ao valor desejado;

e) `hasMultiFuncionalKeyboard()`: este método indica para a API se o teclado possui mais de uma função por tecla.

#### 3.2.1.4 Caso de uso Definição de *look and feel*

A idéia de *look and feel* é proporcionar um meio do usuário-desenvolvedor mudar as cores dos componentes sem precisar atribuir as cores para cada componente em separado. Existe uma implementação padrão provida pela API, portanto, não existe obrigatoriedade da implementação por parte do usuário-desenvolvedor

Para cumprir este objetivo foi criada uma especificação que atenda os requisitos de mudanças de cores de todos os componentes, a figura 5 mostra, através de um diagrama de classes, a *interface* que pode ser implementada para a criação de um *look and feel* customizado.


Figura 5 – Interface que especifica um classe de *look and feel*

É necessária a implementação dos seguintes métodos:

a) `getButtonBackgroundColor()`: deve retornar a cor de fundo padrão dos botões;

b) `getButtonColor()`: deve retornar a cor do texto padrão dos botões;

c) `getButtonPressedColor()`: deve retornar a cor do botão enquanto está pressionado;

d) `getCheckBoxBackgroundColor()`: deve retornar a cor de fundo padrão para os componentes do tipo `checkbox`;

e) `getCheckBoxColor()`: deve retornar a cor do texto padrão para os componentes do tipo `checkbox`;

f) `getComboBoxBackgroundColor()`: deve retornar a cor de fundo padrão para os componentes do tipo `combobox`;

g) `getComboBoxColor()`: deve retornar a cor do texto padrão para os componentes do tipo `combobox`;

h) `getDisabledBackgroundColor()`: deve retornar a cor de fundo padrão para os componentes desabilitados;

i) `getDisabledBorderColor()`: deve retornar a cor padrão para a borda de componentes desabilitados;

j) `getDisabledTextColor()`: deve retornar a cor padrão para o texto de componentes desabilitados;
k) `getLableColor()`: deve retornar a cor padrão o texto de componentes do tipo `label`;
l) `getPanelBackgroundColor()`: deve retornar a cor de fundo padrão para os componentes do tipo `panel`;
m) `getPanelColor()`: deve retornar a cor do texto padrão para os componentes do tipo `panel`;
n) `getRadioGroupBackgroundColor()`: deve retornar a cor de fundo padrão para os componentes do tipo `radiogroup`;
o) `getRadioColor()`: deve retornar a cor do texto padrão para os componentes do tipo `radiogroup`;
p) `getSelectionColor()`: deve retornar a cor padrão para a borda de componentes que estão selecionados;
q) `getTableBackgroundColor()`: deve retornar a cor de fundo padrão para os componentes do tipo `table`;
r) `getTableBoxColor()`: deve retornar a cor de fundo padrão para os componentes do tipo `table`;
s) `getTextBackgroundColor()`: deve retornar a cor de fundo padrão para os componentes do tipo `text`;
t) `getTextColor()`: deve retornar a cor de fundo padrão para os componentes do tipo `text`;
u) `getWindowBackgroundColor()`: deve retornar a cor de fundo padrão para os componentes do tipo `window`;
v) `getWindowBoxColor()`: deve retornar a cor de fundo padrão para os componentes do tipo `window`.

#### 3.2.1.5 Diagrama de classes

O diagrama de classes é utilizado para demonstrar, em sua totalidade, os recursos disponíveis. A figura 6 mostra o diagrama de classes da API, neste diagrama estão apenas os pacotes disponíveis, cada pacote é tratado em separado nas sessões seguintes.

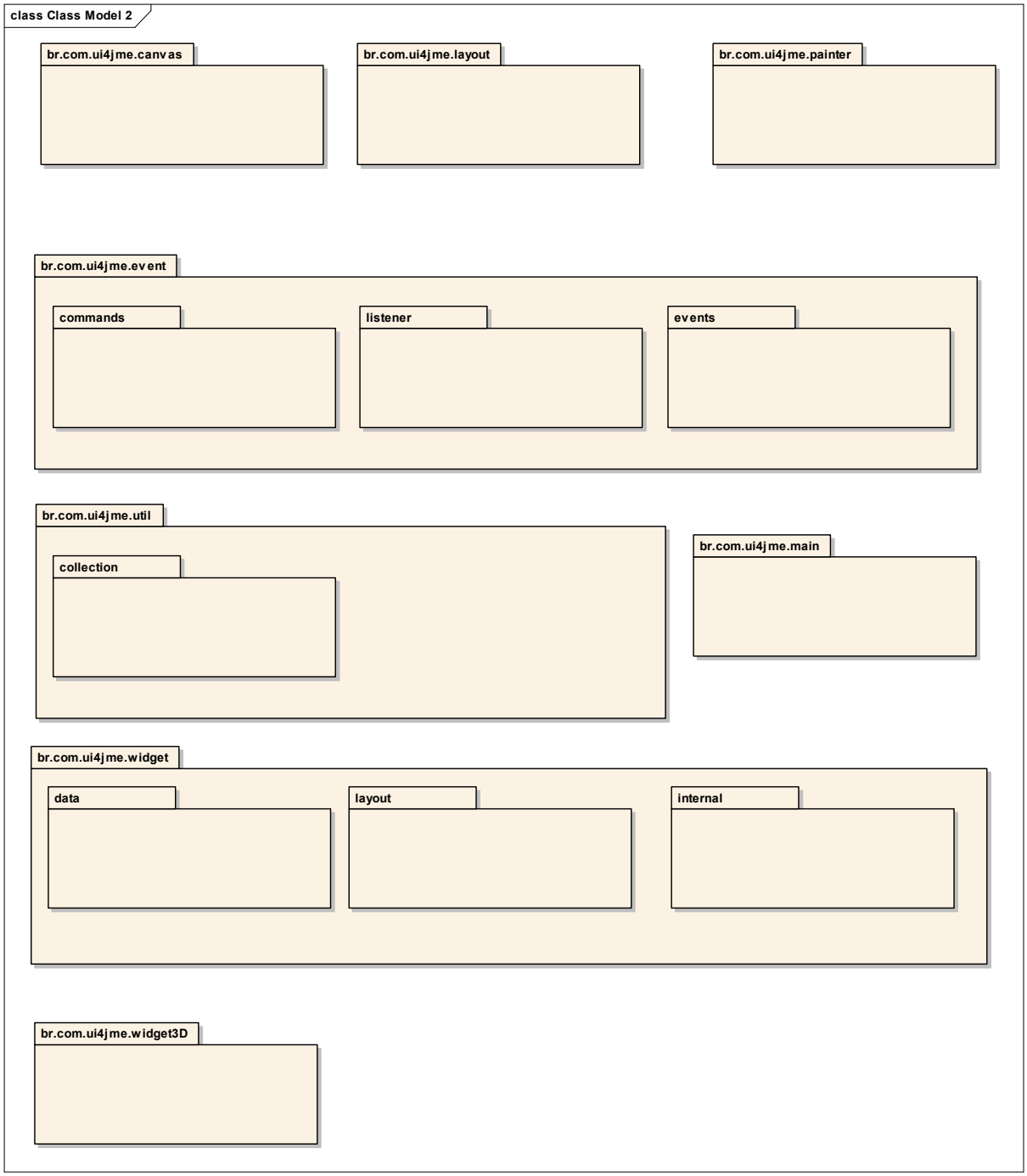



Figura 6 – Diagrama de classes da API

Os pacotes disponíveis são:

a) `br.com.ui4jme.canvas`: onde estão localizadas as classes relacionadas ao *canvas* utilizado pela API;

b) `br.com.ui4jme.layout`: onde estão localizadas as classes criadas para os gerenciadores de *layout*;

c) `br.com.ui4jme.painter`: onde estão localizadas as classes criadas para o pintor de componente e sua interface de abstração;

d) `br.com.ui4jme.event`: onde estão localizadas as classes para tratamento de

eventos, os seus pacotes internos `br.com.ui4jme.event.listener`, `br.com.ui4jme.event.events` e `br.com.ui4jme.event.command` possuem classes para os ouvidores de eventos, as classes dos eventos disponíveis, e comandos para eventos, respectivamente;

e) `br.com.ui4jme.util`: onde estão localizadas as classes utilitárias para a API, como listas encadeadas e mapas de valores. Este pacote, por sua vez, possui um pacote interno, chamado `br.com.ui4jme.util.collection`, para tratamento de coleções de dados;

f) `br.com.ui4jme.main`: onde estão as classes que o desenvolvedor utilizará para iniciar a execução do seu sistema;

g) `br.com.ui4jme.widget`: onde estão localizados as classes dos componentes de tela disponibilizados pela API. Este pacote possui os pacotes internos `br.com.ui4jme.widget.listener`, `br.com.ui4jme.widget.internal` e `br.com.ui4jme.widget.layout` para classes utilitárias relacionadas aos componentes;

h) `br.com.ui4jme.widget3D`: onde estão localizados a classe que especifica a integração da API com o *framework* M3G.

#### 3.2.1.6 O pacote `br.com.ui4jme.canvas`

A figura 7 mostra o diagrama de classes apenas deste pacote.

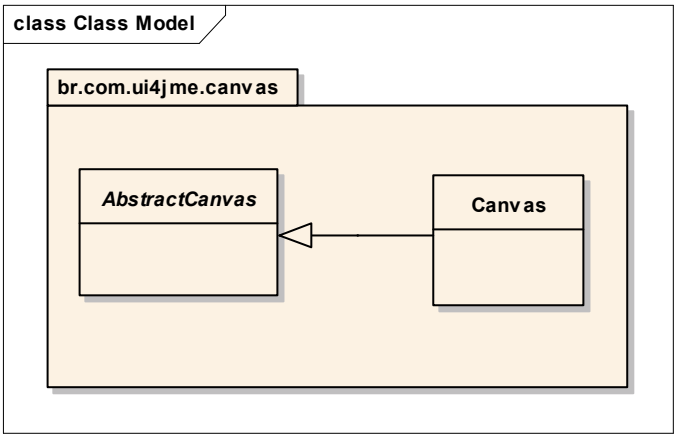



Figura 7 – Diagrama de classes do pacote br.com.ui4jme.canvas

Pode-ser ver as duas classes que compõem o pacote, a classe `AbstractCanvas`, que representa a abstração da área de desenho, para assim permitir que o desenvolvedor crie a sua

própria implementação de `canvas` se necessitar e a classe `Canvas`, que é a implementação padrão para a API.

### 3.2.1.7 O pacote `br.com.ui4jme.layout`

A figura 8 mostra o diagrama de classes deste pacote.

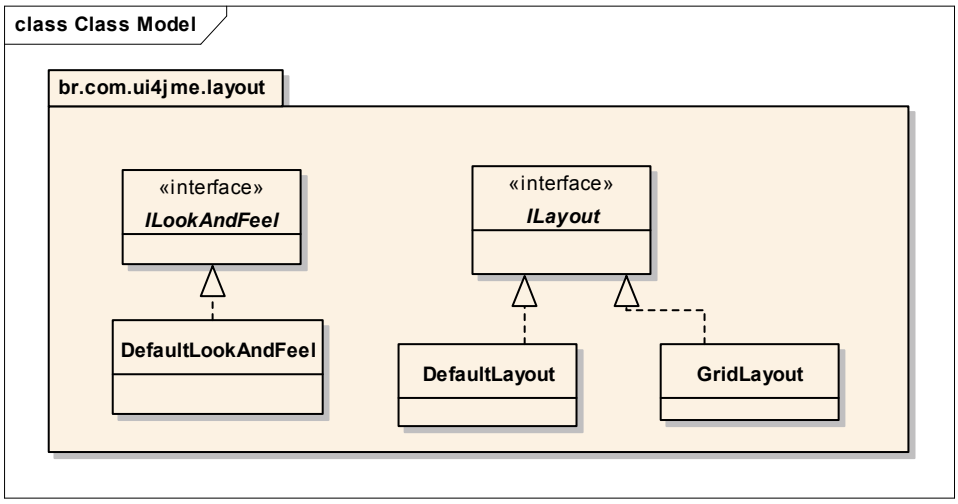



Figura 8 – Diagrama de classes do pacote br.com.ui4jme.layout

Como mostrado na figura 8, este pacote tem por responsabilidade manter todas as classes que se referem aos gerenciadores de *layouts*, e sistema de *look and feel*.

### 3.2.1.8 O pacote `br.com.ui4jme.painter`

A figura 9 mostra o diagrama de classes deste pacote.

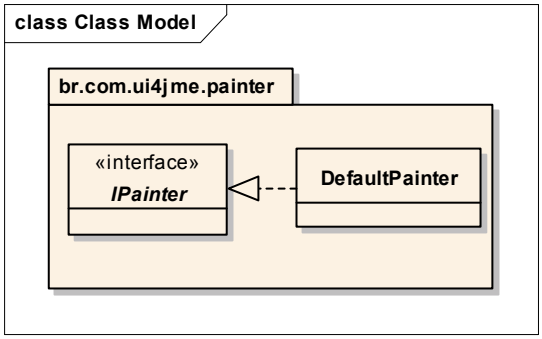



Figura 9 – Diagrama de classes do pacote br.com.ui4jme.painter

Como mostrado na figura 9, este pacote mantém as classes relacionadas aos pintores, classes que possuem o objetivo de desenhar os componentes na tela. O componente possui a

interface `IPainter`, para criar uma abstração, criando assim a possibilidade do desenvolvedor criar um pintor próprio, ale da classe pintor padrão utilizada pela API, chamada `DefaultPainter`.

### 3.2.1.9 O pacote `br.com.ui4jme.event`

A figura 10 mostra todas as classes e pacotes internos que compõem o pacote `br.com.ui4jme.event`. Este pacote mantém todas as classes e interfaces necessárias para a execução dos eventos, como eventos de foco e teclado. O pacote mantém, ainda, os mapas de teclado e suas abstrações, além dos gerentes e fábrica de eventos.

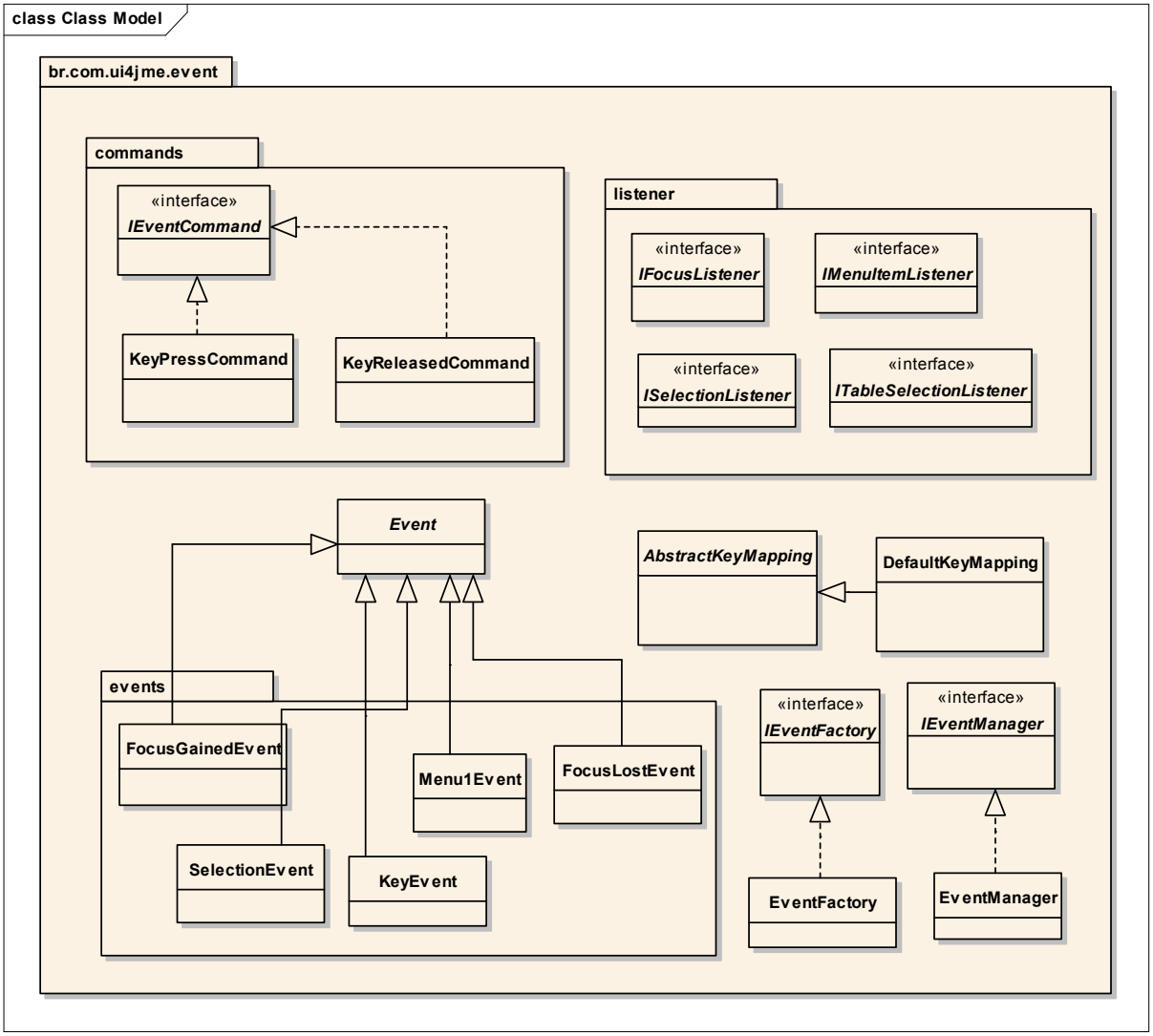



Figura 10 – Diagrama de classes do pacote br.com.ui4jme.event

#### 3.2.1.10 O pacote `br.com.ui4jme.main`

A figura 11 mostra o diagrama de classes deste pacote.

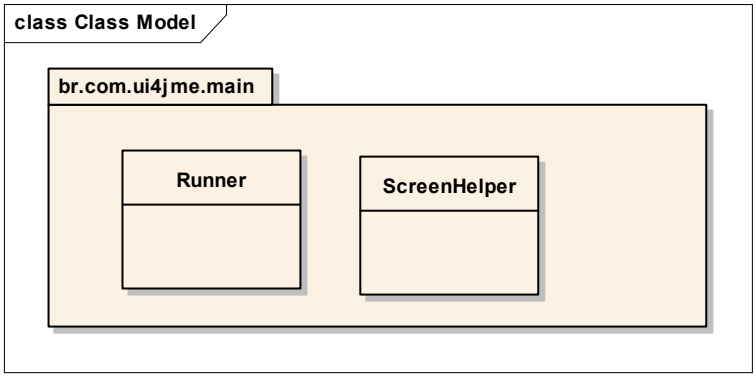



Figura 11 – Diagrama de classes do pacote br.com.ui4jme.main

A figura 11 mostra as classes que compõem o pacote `br.com.ui4jme.main`. Este pacote mantém as classes que correspondem ao núcleo da API, classes que são necessárias para manter dados da tela do dispositivo, no caso da classe `ScreenHelper`, e a classe para iniciar a execução do aplicativo, no caso da classe `Runner`.

#### 3.2.1.11 O pacote `br.com.ui4jme.util`

A figura 12 mostra o diagrama de classes deste pacote.

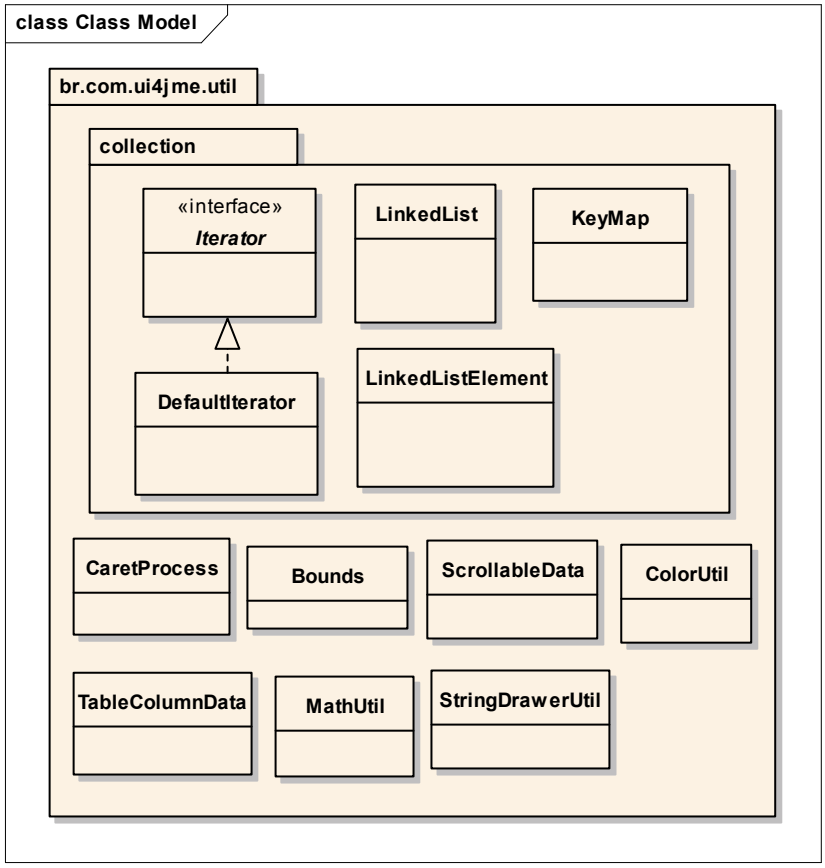



Figura 12 – Diagrama de classes do pacote br.com.ui4jme.util

A figura 12 mostra as classes e pacotes internos que compõem o pacote `br.com.ui4jme.util`. Este pacote tem por objetivo manter todas as classes utilitárias a API, que são classes relativas ao tratamento de dados, tratamento de cálculos matemáticos, utilitários para trabalhar com cores, cursores e trabalhar com desenho de textos na tela.

#### 3.2.1.12 O pacote `br.com.ui4jme.widget`

A figura 13 mostra o diagrama de classes desse pacote.

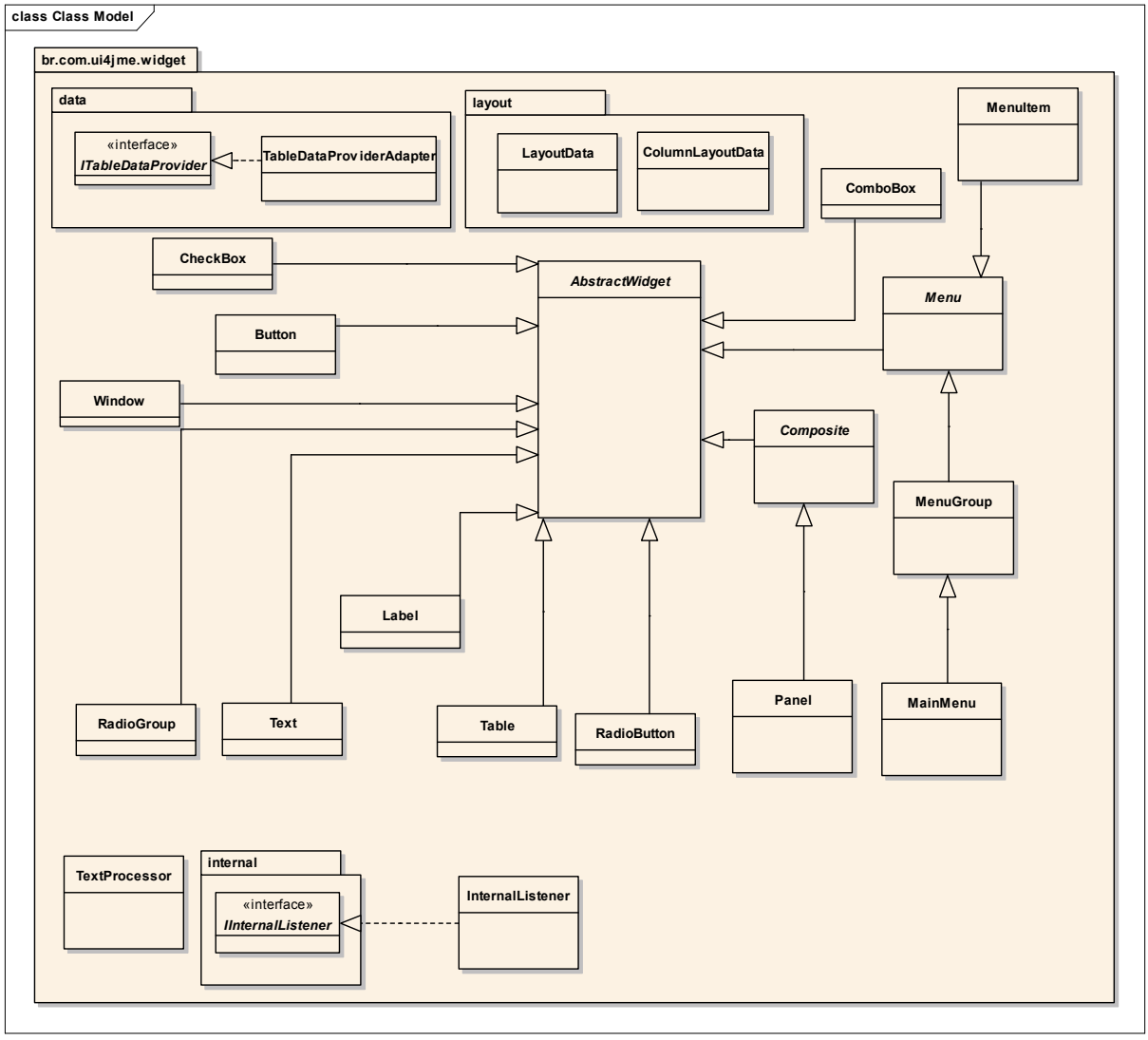



Figura 13 – Diagrama de classes do pacote br.com.ui4jme.widget

A figura 13 mostra as classes e pacotes internos que compõem o pacote `br.com.ui4jme.widget`. Este pacote possuía responsabilidade de manter todos os componentes de interface implementados pela API, bem como o provedor de dados para o componente do tipo `table`. Além dos componentes e provedor de dados, o pacote mantém classes para a configuração do *layout* tabular.

### 3.2.1.13 O pacote `br.com.ui4jme.widget3D`

A figura 14 mostra o diagrama de classes deste pacote.

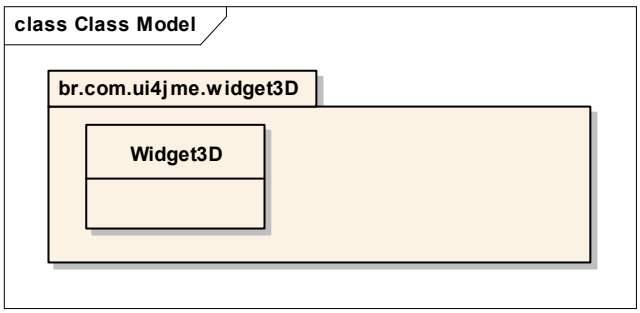



Figura 14 – Diagrama de classes do pacote br.com.ui4jme.widget3D

A figura 14 mostra a classe que compõem o pacote `br.com.ui4jme.widget3D`. Este tem como objetivo manter as classes para integração entre a API e o *framework* de desenvolvimento 3D para dispositivos móveis, M3G.

### 3.2.1.14 Os componentes de interface

Os componentes implementados pela API são o `label`, `text`, `gadriogroup`, `radiobutton`, `checkbox`, `combobox`, `menu` e `table` A figura 15 mostra o diagrama de classes somente dos componentes.

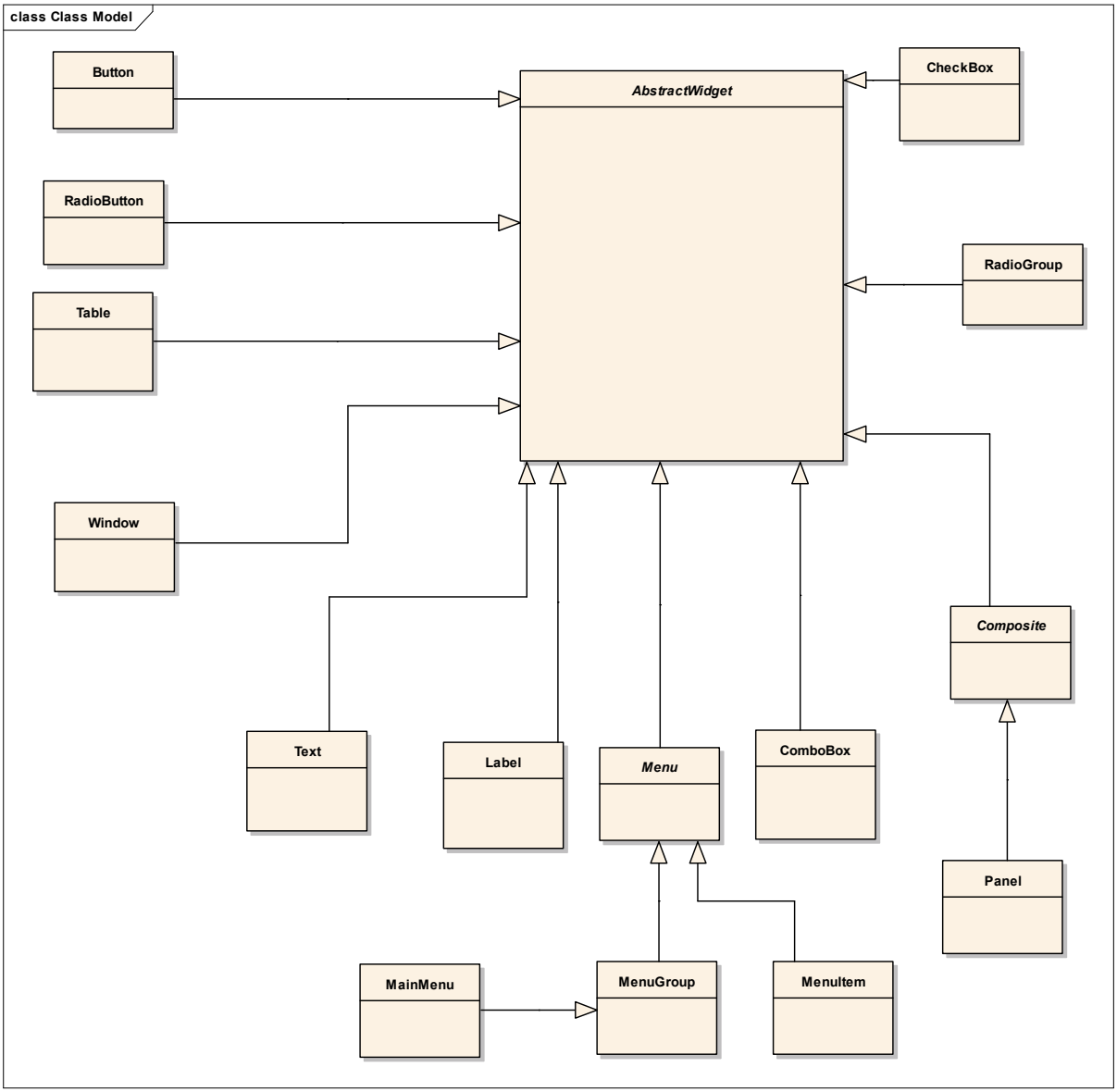



Figura 15 – Diagrama de classes dos componetes gráficos da API

Na figura 15 pode-se ver o diagrama de classes somente das classes que constituem os componentes disponibilizados pela API do presente trabalho para o desenvolvedor utilizar na criação de suas interfaces. A figura 15 ainda mostra a classe abstrata AbstractWidget, classe base para todos os outros componentes, esta classe possui controles referentes a cor de fundo e de texto do componente, visibilidade, identificador único e um controle respectivo ao componente estar habilitado ou não.

Ainda na figura 15 pode-se ver os outros componentes, como `o Button, RadioGroup, RadioButton, Table, Window, Text, Label, Composite, Panel, Menu, MenuGroup, MenuItem e CheckBox`. Estes são todos os componentes disponilizados por padrão, pela API do presente trabalho.

### 3.2.1.15 Especificação do gerenciador de *layout*

Um gerenciador de *layout* tem por objetivo tratar a disposição dos componentes na tela fazendo com que o desenvolvedor tenha o menor trabalho possível. Para tanto, foi criada uma especificação para a criação de gerenciadores de *layoout*.

A figura 16 mostra o diagrama de classes da interface que especifica um gerenciador de *layout*.

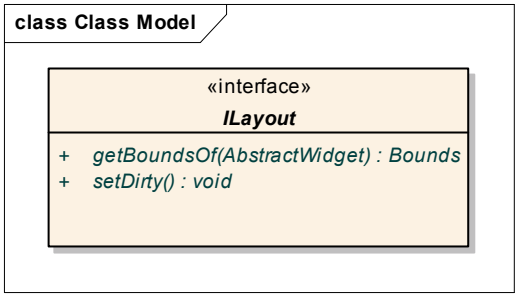



Figura 16 – Diagrama de classes da interface dos gerenciadores de *layout*

Os métodos a serem implementados, seguindo a especificação são:

i) `getBoundsOf(AbstractWidget widget)`: este método deve retornar a largura, altura e posições nos eixos x e y;

j) `setDirty()`: este método deve ser chamado para avisar a API que os valores para os componentes devem ser recalculados.

### 3.2.1.16 Integração com M3G

Para o funcionamento da API em ambientes 3D em celulares, é necessária a integração com o *framework* M3G. A integração da API não será completa, uma vez que existe apenas uma especificação de componentes que auxiliam no desenvolvimento de telas em ambientes 3D, através do M3G, para a integração completa é necessário criar recursos para o gerenciamento da disposição dos componentes e a manutenção de eventos em ambiente 3D. Estes recursos não estão implementados pela API, e ficam a cargo do desenvolvedor a implementação de tais recursos.

A figura 17 mostra em um diagrama de classes, a classe abstrata que deve ser usada para criação de componentes de tela em M3G.

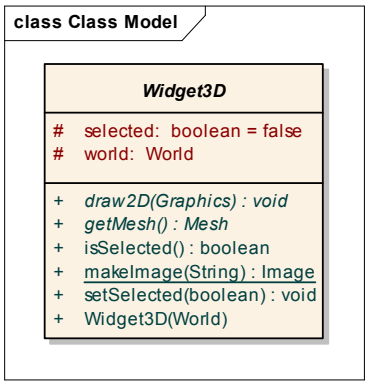



Figura 17 – Diagrama de classes da classe que especifica um componente 3D

A figura 17 mostra a classe que especifica a integração entre a API do presente trabalho e o *framework* gráfico M3G.

Os métodos da classe são:

a) `draw2D(Graphics g)`: este método é responsável por desenhar tudo o que está em 2D do componente;

b) `getMesh()`: este método é responsável por retornar a textura que deverá ser utilizada para o componente;

c) `isSelected()`: este método é responsável por saber se o componente está ou não selecionado;

d) `makeImage(String path)`: este método é responsável por criar uma imagem a partir de um caminho, qual deve apontar para um arquivo do tipo imagem;

e) `setSelected(boolean selected)`: este método é responsável por avisar o componente que este está selecionado;

f) `Widge3D(World w)`: é o construtor do objeto, deve ser passado como parâmetro o mundo 3D, objeto este proveniente do *framework* M3G.

## 3.3 IMPLEMENTAÇÃO

Nesta sessão serão apresentados tópicos pertinentes a técnicas, ferramentas e operacionalidade da API desenvolvida neste trabalho.

### 3.3.1 Técnicas e ferramentas utilizadas

Neste tópico serão apresentadas as principais ferramentas e técnicas utilizadas no desenvolvimento deste trabalho. Abordando a ferramenta de desenvolvimento utilizada para implementação, o emulador e exemplificação das técnicas utilizadas, além do código implementado.

#### 3.3.1.1 Eclipse

Segundo Eclipse (2007, tradução nossa), "O Eclipse é um ambiente de desenvolvimento integrado (IDE), mantido pela Eclipse Foundation, que tem por objetivo ser uma plataforma aberta focada na criação de *frameworks* integrados e extensíveis".

A IDE Eclipse foi escolhida devido a grande quantidade de produtos integrados, como emuladores, mas principalmente devido a gama de *plugins* existentes para uso e extensão da própria IDE.

A figura 18 mostra a IDE Eclipse.

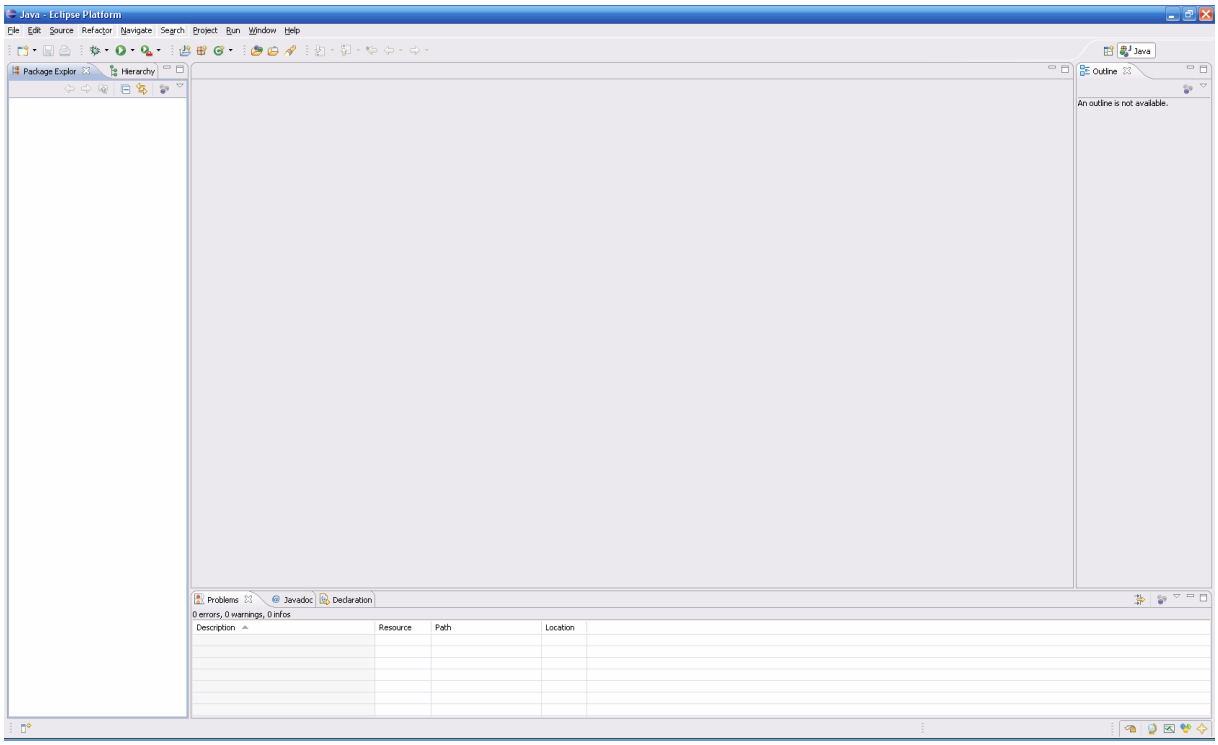



Figura 18 – IDE do Eclipse

### 3.3.1.2 EcliseME

Segundo EclipseME (2007, tradução nossa), “EclipseME é um *framework* para desenvolvimento J2ME, tendo como objetivo permitir o desenvolvedor se focar no desenvolvimento da sua solução em vez de se preocupar com particularidades do desenvolvimento embarcado”.

Esse *plugin* foi escolhido devido a forte integração com a IDE Eclipse, bem como integração com o ferramental de desenvolvimento embarcado disponibilizado pela Sun Microsystems, empresa qual criou a especificação do J2ME.

A figura 19 mostra o ambiente de configuração desta extensão da IDE Eclipse.

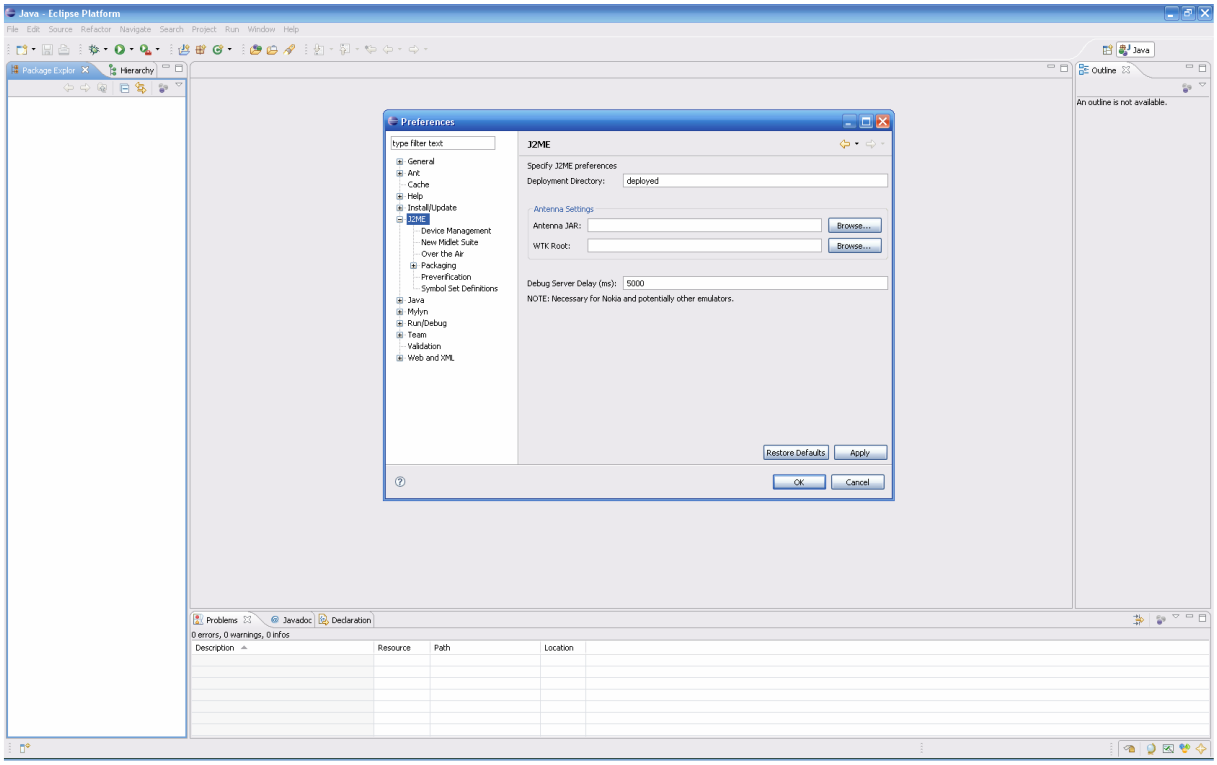



Figura 19 – Tela de configuração do plugin EclipseME

### 3.3.1.3 Sun Java *Wireless Toolkit*

Segundo Sun Microsystems (2007b, tradução nossa) “Sun Java Wireless Toolkit é um ambiente integrado para desenvolvimento de aplicativos móveis, focado da configuração CLDC dentro do J2ME, que inclui emuladores, ferramentas de otimização e *tunning*, exemplos e documentação para tornar rápido e fácil o desenvolvimento embarcado”.

O emulador disponibilizado pelo toolkit é apresentado na figura 20.


Figura 20 – O emulador para desenvolvimento Java para celulares

### 3.3.2 Operacionalidade da implementação

Para demonstrar a operacionalidade da API, é apresentado o desenvolvimento de uma interface gráfica com o usuário, utilizando-se de todos os recursos disponibilizados pelo trabalho.

Cada linha de código é explicada separadamente, e, toda linha de código que alterar o

estado visual da interface terá uma figura demonstrativa da alteração. Para melhor entendimento, é apresentado separadamente o uso de cada gerenciador de *layout* implementado pela API.

#### 3.3.2.1 Umexemplo simples

Nesta seção é demonstrado os requisitos fundamentais para o conhecimento da API. É apresentado um exemplo básico, mostrando como iniciar corretamente a API e como incluir os primeiros componentes.

O quadro 1 mostra uma classe do tipo `MIDlet`, classe que é base para o desenvolvimento de aplicativos J2ME baseados na configuração CLDC.

```
package br.com.ui4jme.sample;
import javax.microedition.midlet.MIDlet;
public class Example extends MIDlet {

   public Example() {
   }
   protected void destroyApp(boolean unconditional) throws MIDletStateChangeException {
   }
   protected void pauseApp() {
   }
   protected void startApp() throws MIDletStateChangeException {
   }
}
```

Quadro 1 – Um exemplo de classe para dispositivos móveis

O quadro 1 demonstra o contrato da classe a ser implementada, trazendo os seguintes métodos:

a) `destroyApp(boolean unconditional)`: este método deve ser invocado para terminar a execução da aplicação. Nele deve ser implementado a liberação de recursos;
b) `pauseApp()`: este método é automaticamente invocado quando a aplicação entra em estado de pausa;
c) `pauseApp()`: este método é automaticamente invocado quando a aplicação entra em estado de execução.

O quadro 2 mostra o início do desenvolvimento da interface para o usuário, utilizando

a API do presente trabalho.

```
package br.com.ui4jme.sample;

import javax.microedition.midlet.MIDlet;

public class Example extends MIDlet {

   public Example() {

   }

   protected void destroyApp(boolean unconditional) throws MIDletStateChangeException {

   }

   protected void pauseApp() {

   }

   protected void startApp() throws MIDletStateChangeException {
      IPainter painter = new DefaultPainter(); //Passo 1
      ILayout layout = new DefaultLayout();    //Passo 2
      ILookAndFeel lookAndFeel = new DefaultLookAndFeel(); //Passo 3

      ScreenHelper.configure(painter, layout, lookAndFeel); //Passo 4
   }
}
```

Quadro 2 – Início do desenvolvimento utilizando a API

No passo 1 é criado o objeto responsável por desenhar os componentes realmente, a classe `DefaultPainter` é disponibilizada pela API, não sendo assim necessária a implementação por parte do desenvolvedor.

No passo 2 é criado o objeto de *layout* que será utilizado para gerenciar a disposição dos componentes na tela. A classe `DefaultLayout` é disponibilizada pela API, este gerenciador deixa a cargo do desenvolvedor a disposição dos componentes, ou seja, o desenvolvedor terá que se preocupar com as posições dos componentes dos eixos X e Y, bem como a altura e largura de cada componente.

No passo 3 é criado o objeto responsável pelo gerenciamento do *look and feel* que será utilizado pela API. A classe `DefaultLookAndFeel` é disponibilizada pela API.

O passo 4 é o mais importante do código demonstrado, neste passo todas diretivas criadas são passadas para a API, informando assim, que estas serão as diretivas utilizadas durante o processamento da interface desenvolvida.

O quadro 3 mostra a continuação do desenvolvimento.

```
package br.com.ui4jme.sample;

import javax.microedition.midlet.MIDlet;

public class Example extends MIDlet {


   public Example() {

   }

   protected void destroyApp(boolean unconditional) throws MIDletStateChangeException {

   }

   protected void pauseApp() {
   }

   protected void startApp() throws MIDletStateChangeException {
      IPainter painter = new DefaultPainter(); //Passo 1
      ILayout layout = new DefaultLayout();    //Passo 2
      ILookAndFeel lookAndFeel = new DefaultLookAndFeel(); //Passo 3

      ScreenHelper.configure(painter, layout, lookAndFeel); //Passo 4

      AbstractKeyMapping keyMapping = new DefaultKeyMapping(); //Passo 5
      IEventFactory eventFactory = new EventFactory(); //Passo 6
      IEventManager eventManager = new EventManager(keyMapping, eventFactory);//Passo 7

      Runner r = new Runner(eventManager); //Passo 8

   }
}
```

Quadro 3 – Criando os mapas de teclado e eventos para a API

No passo 5 é criado o objeto que tem por objetivo trabalhar e tratar as diferenças entre os teclados dos diferentes dispositivos de celulares existentes no mercado. Esta classe poderá ter que ser implementada pelo desenvolvedor caso o dispositivo alvo do sistema que se utiliza da presente API não estiver dentro dos padrões implementados pelos emuladores do *Sun Wireless Toolkit*.

No passo 6 é criado o objeto que tem por objetivo fabricar os eventos necessários ao funcionamento da API, estes eventos fabricados são na verdade uma normalização dos eventos recebidos pelo dispositivo. No passo 7 é criado o gerenciador de eventos, que se utilizará do objeto responsável por normalizar o teclado e o objeto responsável por fabricar os eventos. No passo 8 é criado o objeto principal da API, este objeto dará inicio a execução da API, e assim, os componentes serão apresentados na tela.

No quadro 4 é mostrado a adição dos primeiros componentes.

```
package br.com.ui4jme.sample;

import javax.microedition.midlet.MIDlet;

public class Example extends MIDlet {


   public Example() {

   }

   protected void destroyApp(boolean unconditional) throws MIDletStateChangeException {

   }

   protected void pauseApp() {

   }

   protected void startApp() throws MIDletStateChangeException {
      IPainter painter = new DefaultPainter(); //Passo 1
      ILayout layout = new DefaultLayout();    //Passo 2
      ILookAndFeel lookAndFeel = new DefaultLookAndFeel(); //Passo 3

      ScreenHelper.configure(painter, layout, lookAndFeel); //Passo 4

      AbstractKeyMapping keyMapping = new DefaultKeyMapping(); //Passo 5
      IEventFactory eventFactory = new EventFactory(); //Passo 6
      IEventManager eventManager = new EventManager(keyMapping, eventFactory);//Passo 7

      Runner r = new Runner(eventManager); //Passo 8

      Window window = new Window("Hello World"); //Passo 9
      Panel panel = new Panel(); //Passo 10
      window.setMainComponent(panel); //Passo 11
   }
}
```

Quadro 4 – Criando os primeiros componentes de tela

No passo 9 é criado o objeto de `Window`, este objeto representa a tela inteira, tendo a idéia de uma janela. O parâmetro passo no construtor é o título que a janela apresentará. No passo 10 é criado um objeto do tipo `Panel`, este objeto pode receber componentes filhos. No passo 11 é atribuído à janela, o objeto de `Window` o seu componente principal de trabalho. A janela aceita somente objetos do tipo `Panel` como componentes principais. No quadro 5 é demonstrado a adição de componentes e as últimas linhas de código para então rodar a aplicação.

```
package br.com.ui4jme.sample;

import javax.microedition.midlet.MIDlet;

public class Example extends MIDlet {


   public Example() {

   }

   protected void destroyApp(boolean unconditional) throws MIDletStateChangeException {

   }

   protected void pauseApp() {

   }

   protected void startApp() throws MIDletStateChangeException {
      IPainter painter = new DefaultPainter(); //Passo 1
      ILayout layout = new DefaultLayout();    //Passo 2
      ILookAndFeel lookAndFeel = new DefaultLookAndFeel(); //Passo 3

      ScreenHelper.configure(painter, layout, lookAndFeel); //Passo 4

      AbstractKeyMapping keyMapping = new DefaultKeyMapping(); //Passo 5
      IEventFactory eventFactory = new EventFactory(); //Passo 6
      IEventManager eventManager = new EventManager(keyMapping, eventFactory);//Passo 7

      Runner r = new Runner(eventManager);

      Window window = new Window("Hello World"); //Passo 9
      Panel panel = new Panel(); //Passo 10
      window.setMainComponent(panel); //Passo 11

      Label label = new Label("Label"); //Passo 12
      label.setBounds(new Bounds(10, 32, 50, 20)); //Passo 13
      panel.addChild(label); //Passo 14

      Display display = Display.getDisplay(this); //Passo 15
      r.run(display, window); //Passo 16
   }
}
```

Quadro 5 – Adicionando os primeiros componentes à tela

No passo 12 é criado um objeto do tipo `Label`, este objeto representa um texto estático que será apresentado na tela. No passo 13 é configurado o posicionamento e tamanho do componente do tipo `Label`. No passo 14 o componente é adicionado definitivamente à tela, no passo 15 é recuperado a instância de `display` atual, onde existem métodos auxiliares provenientes do *profile* MIDP, e no passo 16 a API é executada para desenhar os componentes. Na figura 21 tem-se o resultado da execução no emulador.


Figura 21 – Execução da API com os primeiros componentes

Na figura 21 pode-se observar o componente `Label` adicionado, o título da janela que foi passado como parâmetro, bem como, as configurações de *look and feel* padrões em funcionamento. Neste exemplo a cor do texto do `Label` não foi configurada, mas ela se apresenta na cor preta assim como as cores de fundo estão em tons de cinza e preto, sem serem configuras, provenientes, assim, do gerenciador de *look and feel* previamente configurado.

### 3.3.2.2 Utilizando os eventos

Nesta sessão é demonstrada o uso de eventos junto a API. Os eventos são os de execução em botões, ganho e perda de foco. No quadro 6 é demonstrado como adicionar um botão à tela, bem como configurar um evento para o botão adicionado.

```
package br.com.ui4jme.sample;

import javax.microedition.midlet.MIDlet;

public class Example extends MIDlet {


   public Example() {

   }

   protected void destroyApp(boolean unconditional) throws MIDletStateChangeException {

   }

   protected void pauseApp() {
   }

   protected void startApp() throws MIDletStateChangeException {
      IPainter painter = new DefaultPainter(); //Passo 1
      ILayout layout = new DefaultLayout();    //Passo 2
      ILookAndFeel lookAndFeel = new DefaultLookAndFeel(); //Passo 3

      ScreenHelper.configure(painter, layout, lookAndFeel); //Passo 4

      AbstractKeyMapping keyMapping = new DefaultKeyMapping(); //Passo 5
      IEventFactory eventFactory = new EventFactory(); //Passo 6
      IEventManager eventManager = new EventManager(keyMapping, eventFactory);//Passo 7

      Runner r = new Runner(eventManager);

      Window window = new Window("Hello World"); //Passo 9
      Panel panel = new Panel(); //Passo 10
      window.setMainComponent(panel); //Passo 11

      final Label label = new Label("Label"); //Passo 12
      label.setBounds(new Bounds(10, 32, 50, 20)); //Passo 13
      panel.addChild(label); //Passo 14

      Button button = new Button("Button"); //Passo 17
      button.setBounds(new Bounds(70, 32, 50, 20)); //Passo 18
      panel.addChild(button); //Passo 19

      Button.setSelectionLisntener(new ISelectionListener() { //Passo 20
         public void keydown(Event e) { //Passo 21
            System.out.println("passando no keydown"); //Passo 22
         }
         public void keyup(Event e) { //Passo 23
            label.setText("novo texto"); //Passo 24
            System.out.println("passando no up"); //Passo 25
         }
      });


      Display display = Display.getDisplay(this); //Passo 15
      r.run(display, window); //Passo 16

   }
}
```

Quadro 6 – Implementação de eventos na API

No passo 17 é criado um objeto do tipo `Button`, este objeto representa um botão que será apresentado na tela. No passo 18 é configurado o posicionamento e tamanho do componente do tipo `Button`. No passo 19 o componente é adicionado definitivamente à tela, e nos passos 15 e 16 a API é executada para desenhar os componentes. Na figura 22 tem-se o resultado da execução no emulador. Os passos 20 a 25 mostram a configuração de um *listener*

para os eventos de seleção relacionados ao botão. O contrato deste ouvidor é:

a) `keydown(Event e)` método que será chamado ao botão ser pressionado;

b) `keyup(Event e)` método que será chamado ao botão ser liberado.


Figura 22 – Execução do código contendo eventos

A figura mostra os componentes adicionados à tela corretamente desenhados. A figura 23 mostra a tela, após a execução do evento sobre o botão e a saída no console, respectivamente.

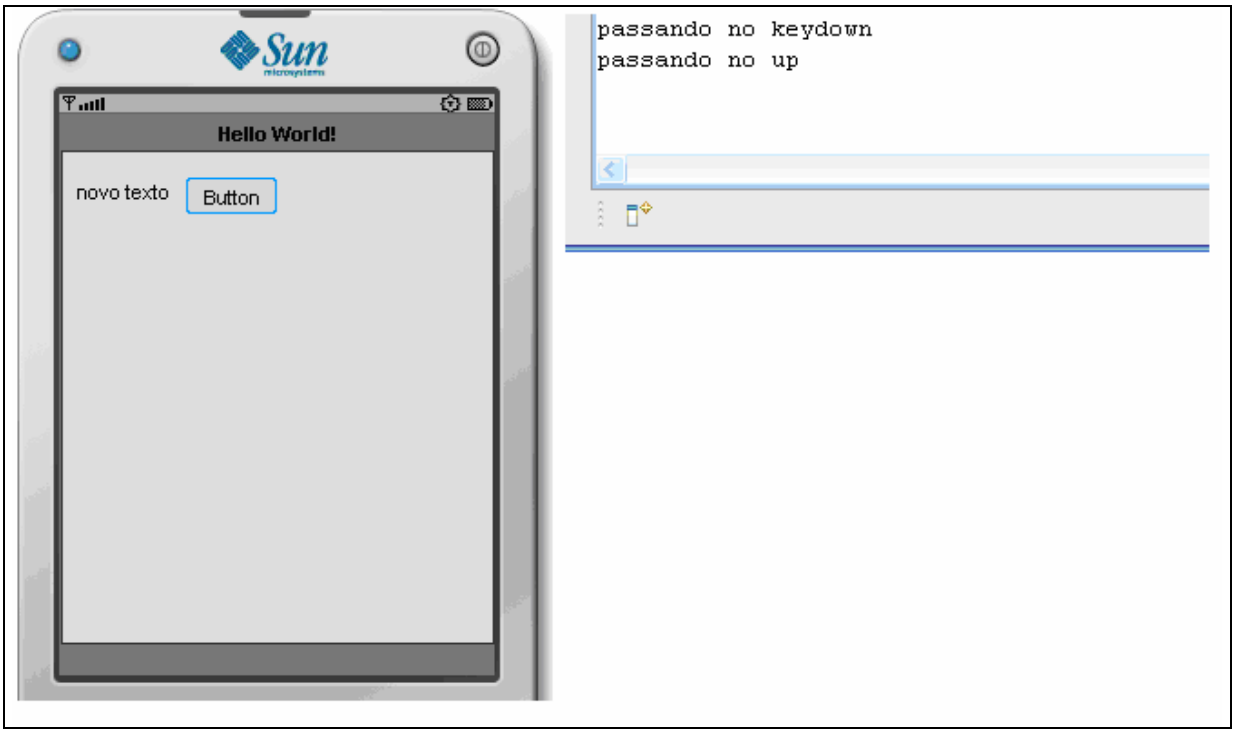



Figura 23 – Execução após o evento

Na figura pode-se perceber que o texto do componente `Label` foi alterado, conforme código no *listener*do botão.

### 3.3.2.3 Utilizando o *layout*tabular

O gerenciador de *layout* tabular tem por objetivo retirar da responsabilidade do desenvolvedor a parte de tamanho e disposição dos campos na tela. O seu funcionamento consiste em criar linhas e colunas onde os componentes serão inseridos. Como forma do desenvolvedor configurar as linhas e colunas, é criado o componente `LayoutData` onde podem ser atribuídas diretivas quanto aos pesos das linhas e colunas, além da quantidade de colunas. Os pesos das linhas e colunas são os percentuais da largura ou altura que cada linha ou coluna receberá, em referência ao todo disponível, ou seja, a largura ou altura de uma linha ou coluna será o peso atribuído multiplicado pelo valor em *pixels* de um peso. A medida de um peso é a largura ou altura total disponível em *pixels* dividida pela soma dos pesos.

Um gerenciador de *layout* se faz necessário não somente pela facilidade de desenvolvimento em não precisar atribuir os tamanhos e posicionamento para cada componente, mas sim para um ajuste automático da disposição da tela no caso da execução do mesmo aplicativo em dispositivos com visores de tamanhos diferentes. Neste ponto, o

gerenciador de *layout* trataria a disposição dos componentes de forma coerente em relação à área visível disponível. No quadro 7 existe um exemplo da facilidade em se utilizar do gerenciador de *layout* ao invés de configurar a posição e tamanho de cada componente em separado.

Utilizando-se do mesmo exemplo de aplicação visto até este ponto, agora adiciona-se o uso do *layout* tabular. O quadro 7 mostra um exemplo de utilização de*layout*tabular .

```
package br.com.ui4jme.sample;

import javax.microedition.midlet.MIDlet;

public class Example extends MIDlet {


   public Example() {

   }

   protected void destroyApp(boolean unconditional) throws MIDletStateChangeException {

   }

   protected void pauseApp() {
   }

   protected void startApp() throws MIDletStateChangeException {
      IPainter painter = new DefaultPainter();
      ILayout layout = new GridLayout();    //Passo 1
      ILookAndFeel lookAndFeel = new DefaultLookAndFeel();

      ScreenHelper.configure(painter, layout, lookAndFeel);

      AbstractKeyMapping keyMapping = new DefaultKeyMapping();
      IEventFactory eventFactory = new EventFactory();
      IEventManager eventManager = new EventManager(keyMapping, eventFactory);

      Runner r = new Runner(eventManager);

      Window window = new Window("Hello World");
      Panel panel = new Panel();

      LayoutData layoutData = panel.getLayoutData(); // Passo 2
      layoutData.setColumnsWeight(new int[] {1, 1}); // Passo 3

      window.setMainComponent(panel);

      final Label label = new Label("Label");
      panel.addChild(label);

      Button button = new Button("Button");
      panel.addChild(button);

      Button.setSelectionLisntener(new ISelectionListener() {
         public void keydown(Event e) {
            System.out.println("passando no keydown"); //Passo 22
         }
         public void keyup(Event e) { //Passo 23
            label.setText("novo texto"); //Passo 24
            System.out.println("passando no up"); //Passo 25
         }
      });
      Display display = Display.getDisplay(this); //Passo 15
      r.run(display, window); //Passo 16

   }
}
```

Quadro 7 – Código com exemplo de utilização de *layouyt* tabular

No passo 1 tem-se a criação de um novo tipo de *layout*, o `GridLayout`, que representa um *layout* tabular. No passo 2 é apresentado o uso do configurador de *layouts,* chamado `LayoutData`. Este componente agrega os atributos relacionados aos pesos de linhas e colunas configurados para cada componente. No passo 3 pode-ser ver a configuração de pesos, sendo 2 colunas de pesos idênticos. Na figura 25, do lado esquerdo, tem-se o resultado da aplicação. A saída apresenta os dois componentes exatamente com a mesma largura, qual foi atribuída através de pesos para as colunas, no caso pesos 1 e 1 para as colunas.

Para tornar o botão maior que o texto, por exemplo, deve-se alterar o valor do peso da coluna do botão para um valor superior ao valor do peso da coluna em que o texto se encontra, como apresentado no quadro 8. Na figura 24, do lado direito, é apresentado o botão o código que faz com que o botão tenha o dobro de largura em relação ao tamanho do texto.

```
protected void startApp() {
     ...
     LayoutData layoutData = panel.getLayoutData(); // Passo 2
     layoutData.setColumnsWeight(new int[] {1, 2}); // Passo 3
     ...
  }
```

Quadro 8 – Código com exemplo de utilização de *layouyt* tabular configurado

No passo 3 é atribuído o peso 1 para a coluna 1, onde se encontra o texto, e o peso 2 para a coluna 2, onde se encontra o botão

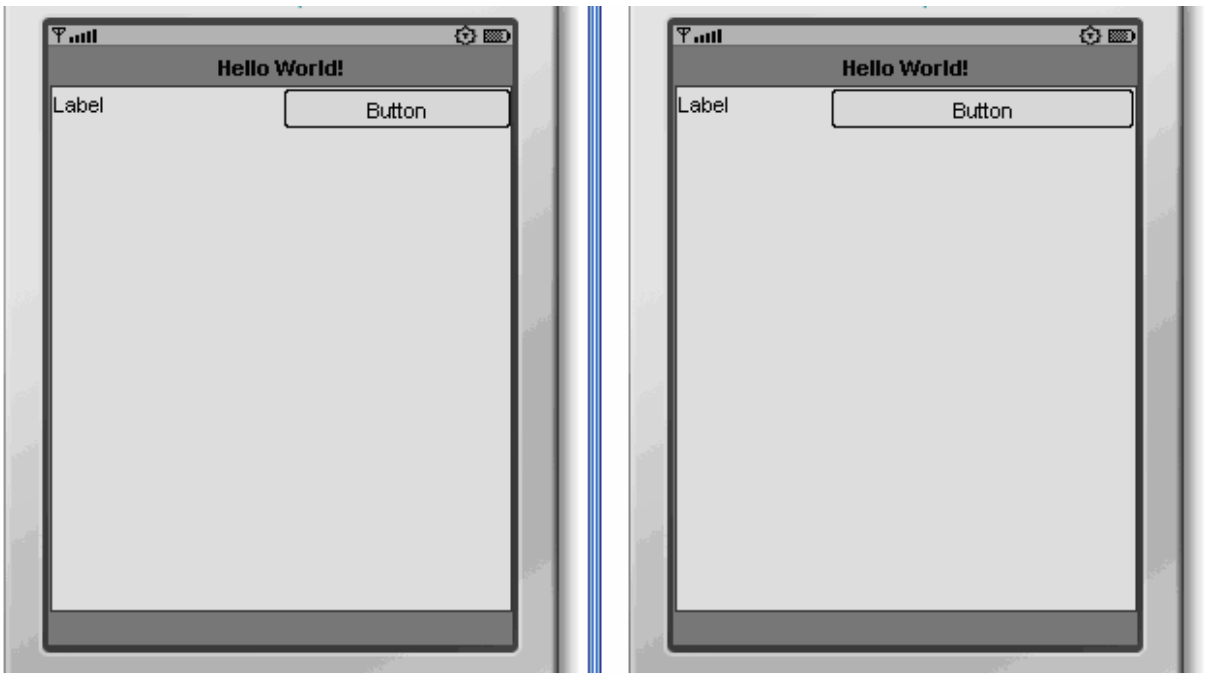



Figura 24 – Execução do aplicativo utilizando *layout* tabular

#### 3.3.2.4 Semelhança entre AWT e o trabalho

A semelhança entre o desenvolvimento de telas utilizando-se da API AWT desenvolvida pela empresa Sun Microsystems e o presente trabalho tem por objetivo tornar mais fácil a adaptação dos desenvolvedores em relação ao uso da API.

No quadro 9 tem-se o código de uma aplicação desenvolvida em AWT.

```
...
this.setSize(450, 250);
Panel panel = new Panel();
Button button = new Button("Botao");
button.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent ae) {
      System.out.println("Clique no botao");
   }
});
panel.add(button);
...
```

Quadro 9 – Código exemplo em AWT

A execução do código do quadro 9 é apresentado na figura 25.

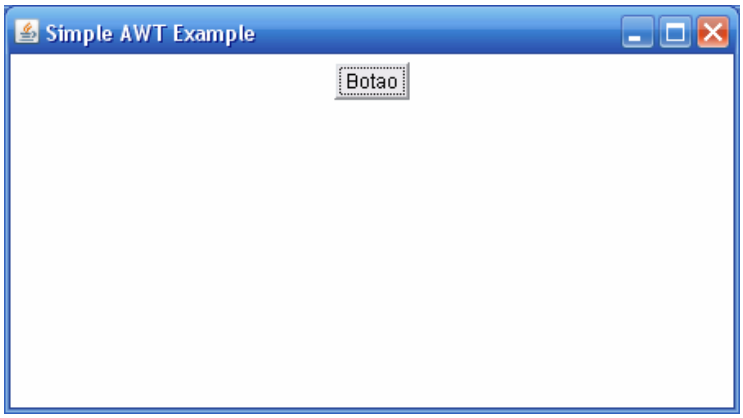



Figura 25 – Execução do código da API AWT

No quadro 10 é mostrado o código implementado com a API do presente trabalho, mostrando a semelhança com a API AWT.

```
...
Panel panel = new Panel();
Button button = new Button("Botao");
button.setSelectionListener(new ISelectionListener() {
   public void keydown(Event e) {
      System.out.println("Clique no botao");
   }
   Public void keyup(Event e) {
      System.out.println("Clique no botao");
   }
});
panel.addChild(button);
...
```

Quadro 10 – Código exemplo da API do presente trabalho

A execução do código do quadro 10 é mostrada na figura 26.


Figura 26 – Execução do código da API do presente trabalho para comparação com a API AWT

Comparando-se os códigos dos quadros 9 e 10, pode-se ter a idéia da semelhança entre a API do presente trabalho e a API AWT, onde a forma de adicionar componentes é a mesma, apenas adicionando os componentes aos seus respectivos pais, e a forma de programar eventos é a mesma, utilizando-se de *listeners*.

#### 3.3.2.5 Um exemplo com todos os componentes

Os componentes implementados pela API são o `label`, `text`, `gadriogroup`, `radiobutton`, `checkbox`, `combobox`, `menu` e `table`.

A figura 27 mostra uma tela contendo todos os componentes, exceto a `table`, que será demonstrada na figura 28.

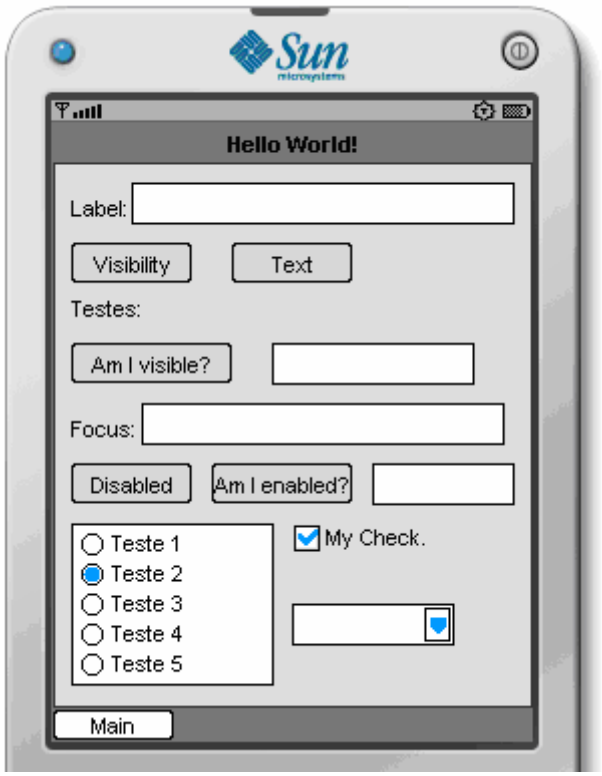



Figura 27 – Exemplo de aplicação com vários componentes

No exemplo da figura 27 pode-se ver a disposição dos componentes em um aplicativo real, que visa demonstrar os componentes disponíveis para utilização do desenvolvedor. Na primeira linha tem-se exemplos dos componentes `label` e `text`, passando estes componentes, existem exemplos de componentes `button`, até chegar aos componentes `radioGroup`, `checkBox` e `comboBox`.

A figura 28 mostra um exemplo com a utilização do componente de listagem de dados, a `table`. Neste exemplo a idéia é mostrar o componente `table`, que tem por objetivo permitir a exposição de uma quantidade maior de dados de forma organizada. Este componente foi construído com a idéia de paginação dos dados, ou seja, se a quantidade de dados a ser exibida for maior do que a quantidade de dados que pode ser mostrado dentro do tamanho do componente, serão criadas mais páginas automaticamente.

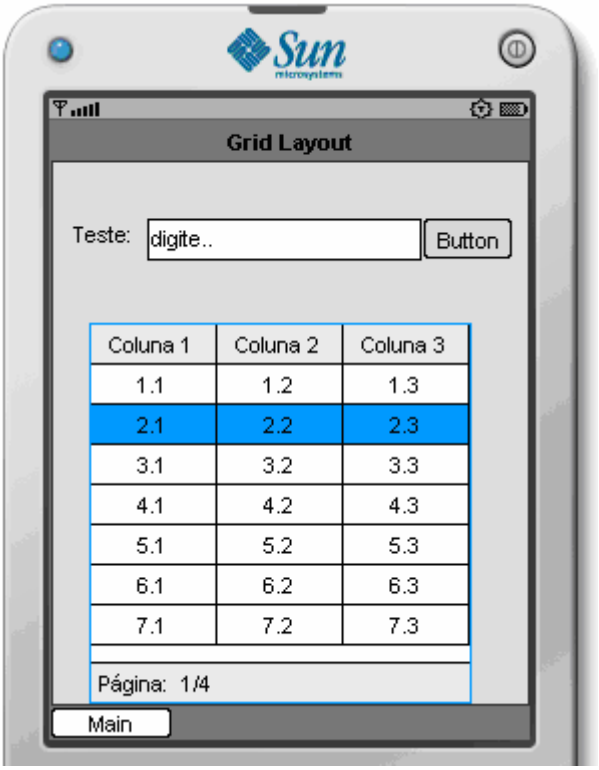



Figura 28 – Exemplo de aplicação com vários componentes, e uma table

### 3.3.3 RESULTADOS E DISCUSSÃO

A realização dos testes foi feita em três etapas: a primeira etapa está relacionada ao desenvolvimento dos componentes de tela, a segunda etapa para os testes de consolidação dos gerenciadores de *layout* e a terceira e última etapa tem como objetivo validar a integração da API com a tecnologia M3G.

Todas as etapas foram desenvolvidas e validadas utilizando-se o emulador disponibilizado para a criação de *softwares* embarcados sobre a tecnologia J2ME. A idéia de não utilizar celulares reais para o desenvolvimento se deu pelo tempo necessário para recriar toda a camada pertinente ao trabalho com os diferentes teclados de celular, ou seja, mapear cada tecla e seus respectivos valores.

Na primeira etapa foram desenvolvidos os componentes propostos `label`, `text`, `gadriogroup`, `radiobutton`, `checkbox`, `combobox` e `menu`, houve ainda a adição de um novo componente, a `table`. Este novo componente foi criado devido à necessidade de expor, de forma fácil, uma maior quantidade de dados. Nesta etapa houve o cuidado de criar uma estrutura de componentes e eventos que facilitassem o desenvolvimento das telas bem como facilitasse o uso por parte do usuário final.

Na segunda etapa fez-se um estudo sobre a melhor forma de criar um gerenciador de *layout* que atendesse de forma satisfatória a disposição dos componentes em tamanhos de telas menores, como os celulares.

Na terceira etapa foi criada uma especificação de componente à ser seguida para a integração com a tecnologia M3G. Nenhuma classe concreta foi construída, uma vez que, a forma de utilização da M3G está intimamente ligada à forma de desenvolvimento do jogo proposto, assim, a melhor forma de integração é criar apenas uma especificação a ser seguida pelo desenvolvedor.

Em comparação com os trabalhos correlatos, JTGL não pode ser testada e comparada ao presente trabalho pois o *site* da API esteve indisponível por grande parte do tempo de desenvolvimento do trabalho. A comparação com o Thinlet pôde ser feita, e é constatado que Thinlet possui todos os recursos do presente trabalho, além de um maior número de componentes e maior número de recursos, porém, não existe a integração com M3G, uma vez que esta API esta disponível apenas na configuração CLDC da J2ME. Também não é possível utilizá-lo em celulares, apenas em dispositivos móveis de maior porte, como *Pocket PCs*, logo que a API utiliza recursos não presentes em configurações menos poderosas, como o CLDC. Assim, a integração com M3G e o funcionamento da configuração CLDC são as mais importantes contribuições do presente trabalho.

Como forma de documentação da API, foi escolhido o uso do JavaDoc, ferramenta que gera documentação HTML baseada na documentação escrita nos códigos fonte que compõe o aplicativo. A figura 29 mostra um exemplo do JavaDoc gerado.

Assim, a API se torna bastante atrativa para o desenvolvimento de aplicações para celulares, pois é altamente customizável e extensível, além de atender as principais necessidades do ambiente móvel de desenvolvimento de aplicações.

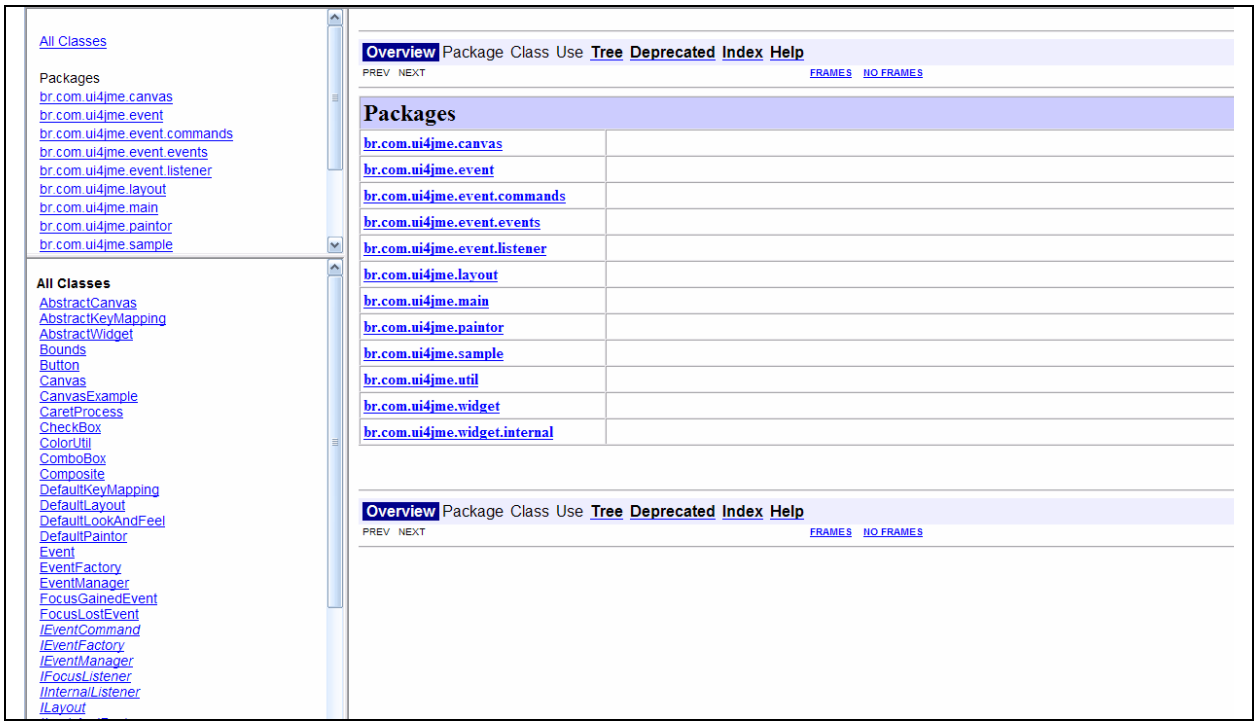



Figura 29 – Exemplo de JavaDoc gerado

# 4 CONCLUSÕES

Dispositivos móveis se tornaram uma grande oportunidade para o desenvolvimento de aplicações, mas como não poderia ser diferente, a criação de software para estes dispositivos possui a necessidade de conhecimentos específicos pertinentes ao próprio ambiente móvel, bem como conhecimentos sobre as limitações de hardware, limitações de recursos oferecidos, enfim, limitações dos dispositivos móveis.

Estas limitações, que são grandes em *Pocket PCs* e PDAs, tornam-se ainda mais visíveis e problemáticas no caso dos celulares. Uma destas limitações encontra-se no desenvolvimento de interfaces para com o usuário, ou seja, o desenvolvimento das telas dos sistemas. Visando suprir esta limitação, foi desenvolvida a API de interface gráfica para celulares.

O presente trabalho cumpriu todos os requisitos propostos, e foram adicionados outros durante o desenvolvimento do trabalho, como a idéia de criar um sistema de *look and feel* para facilitar a customização de cores sem a necessidade de alterar todo o legado.

As vantagens no uso desta API se encontram na facilidade de aderência por parte de desenvolvedores JAVA com algum conhecimento das APIs gráficas utilizadas para *desktop*, além de recursos como gerenciadores de *layout*, integração com M3G, porém, o ponto mais forte do trabalho está na portabilidade entre dispositivos, uma vez que foram criadas especificações para as partes onde não existe um padrão entre os aparelhos de celular. O único grande trabalho a ser feito pelo desenvolvedor da aplicação está relacionado ao mapeamento dos teclados dos aparelhos de celular em que a aplicação deverá funcionar.

A principal limitação da API está relacionada ao trabalho com imagens, apenas a especificação de integração com M3G possuía funcionalidade de uso de imagens, todos os componentes implementados não possuem tal recurso, esta limitação existe devido a falta de tempo para término das implementações.

## 4.1 EXTENSÕES

Sugere-se, para futuros trabalhos, a criação de funcionalidades para o uso de imagens nos componentes criados, a criação de componentes mais complexos como `progressbar` e

`treeview`, mas principalmente a construção de um ambiente para o desenvolvimento das telas para o sistema.

Este ambiente deve facilitar a construção das telas, e seu funcionamento poderia ser feito com a utilização de *drag and drop* dos componentes, tornando-se, assim, a construção das telas totalmente visual, e o desenvolvedor não teria mais a preocupação com a disposição dos componentes na tela através do código. O ambiente poderia ser tanto uma aplicação separada, como também poderia ser um *plugin* para a IDE de desenvolvimento Eclipse, uma extensão do Eclipse integraria ainda mais o desenvolvimento embarcado, facilitando ainda mais o desenvolvimento das aplicações.

# REFERÊNCIAS BIBLIOGRÁFICAS


BAJZAT, Robert: **Thilet.** [S.l.], [2006]. Disponível em: <http://thinlet.sourceforge.net/home.html>. Acesso em: 17 set. 2006.

Eclipse. **Eclipse:** an open development plataform**.** [S.l.], [2007]. Disponível em: <http://www.eclipse.org/>. Acesso em: 11 set. 2007.

EclipseME. **EclipseME:** an eclipse plugin**.** [S.l.], [2007], Disponível em: <http://www.eclipse.org/>. Acesso em: 11 set. 2007.

HEMPHILL, Davic; White, James. **Java 2 Micro Edition:** Java is small things. New York, NY: Manning Publications, 2002.

HUOPANIEMI, Jyri et al. **Programming wireless devices with the Java 2 Plataform, Micro Edition.** 2nd.ed. Boston, MA: Addison Wesley, 2003.

MACORATTI, José Carlos. **UML**: principais diagramas da linguagem**.** [S.l.], [2006]. Disponível em: < http://www.macoratti.net/net_uml3.htm> Acesso em: 11 set. 2007.

MOSIMANN NETTO, Max. **Microsoft .NET Compact Framework:** conheça a plataforma para dispositivos móveis criada pela Microsoft. [São Paulo], 2005. Disponível em: <http://www.linhadecodigo.com.br/artigos.asp?id_ac=646>. Acesso em: 13 set. 2007.

MRMX: **Java Tiny Gfx Library Project.** [S.l.], [2006?]. Disponível em: <http://www.jtgl.org/>. Acesso em: 17 set. 2006.

MUCHOW, John W. **Core J2ME:** technology & MIDP. Indianapolis, IN: Prentice Hall PTR, 2001.

SUN Microsystems. **Abstract Window Toolkit (AWT).** [S.l.], [2006a], Disponível em: <http://java.sun.com/j2se/1.5./docs/guide/awt/index.html>. Acesso em: 5 nov. 2006.

SUN Microsystems. **JSR 184**: Mobile 3D Graphics API form J2ME**.** [S.l.], [2006b]. Disponível em: <http://jcp.org/en/jsr/detail?id=184>. Acesso em: 17 set. 2006.

SUN Microsystems. **JavaDoc**: JavaDoc tool home page**.** [S.l.], [2007a]. Disponível em: <http://java.sun.com/j2se/javadoc/>. Acesso em: 11 set. 2007.

SUN Microsystems. **JavaME**: Sun Java wireless toolkit for CLDC**.** [S.l.], [2007b]. Disponível em: <http://java.sun.com/products/sjwtoolkit/>. Acesso em: 11 set. 2007.

TOPLEY, Kin. **J2ME in a nutshell:** a desktop quiick reference. Sebastopol, CA: O'Reilly, 2002.

XEXEO, Geraldo**. Modelagem de sistemas de informação**: da análise de requisitos ao modelo de interface. Rio de Janeiro: ebook, 2007. 311 p. Disponível em: <http://ge.cos.ufrj.br/tikiwiki/tiki-download_file.php?fileId=1>. Acesso em: 11 set. 2007.