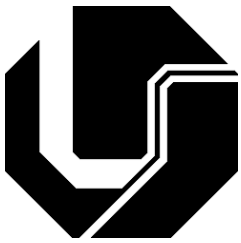
**UNIVERSIDADE FEDERAL DE UBERLÂNDIA**
**FACULDADE DE ENGENHARIA ELÉTRICA**
**PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA**



# Adequação de um Framework para suportar Realidade Virtual em Dispositivos Móveis.

**Mônica Rocha Ferreira de Oliveira**
**Orientanda**

**Alexandre Cardoso, Dr**
**Orientador**

**Edgard Afonso Lamounier Júnior, Dr**
**Co-orientador**

Uberlândia, MG, Fevereiro de 2011.

**UNIVERSIDADE FEDERAL DE UBERLÂNDIA**
**FACULDADE DE ENGENHARIA ELÉTRICA**
**PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA**

# Adequação de um Framework para suportar Realidade Virtual em Dispositivos Móveis.

Tese apresentada por Mônica Rocha Ferreira de Oliveira à Universidade Federal de Uberlândia para obtenção do título de Doutor em Ciências, avaliada em 25/02/2011 pela banca examinadora.

Área de Concentração

Processamento da Informação

Banca Examinadora:

Alexandre Cardoso, Dr (UFU) – Orientador

Edgard Afonso Lamounier Júnior, PhD (UFU) – Co-Orientador

Gilberto Carrijo, Dr. (UFU)

Robson Augusto Siscoutto, Dr. (UNOESTE)

Verônica Teichrieb, Dra. (UFPE)

Uberlândia, MG, Fevereiro de 2011.

# Adequação de um Framework para suportar Realidade Virtual em Dispositivos Móveis.

Tese apresentada por Mônica Rocha Ferreira de Oliveira à Universidade Federal de Uberlândia para obtenção do título de Doutor em Ciências.

| | |
|---|---|
| _____ | _____ |
| Alexandre Cardoso, Dr. | Edgard Afonso Lamounier Júnior, PhD. |
| Orientador/Coordenador | Co-Orientador |

Existem pessoas que estão sempre ao nosso lado...
Estejam onde estiverem...

Dedico este trabalho a meus avós, Adelaide e
Bartolomeu. Com amor..

# *AGRADECIMENTOS*

À Deus.

Aos meus familiares que sempre me apoiaram.

Ao meu esposo “Rogério”, pelo amor, carinho, compreensão, paciência e força...

Aos meus filhos, Matheus e Marianna, os grandes e verdadeiros tesouros da minha vida.

Aos prof. Dr. Alexandre Cardoso e prof. PhD. Edgard Lamounier Jr., pela amizade, incentivo e apoio irrestrito prestado para realização deste trabalho, por suas orientações, obrigada.

Ao pessoal do Laboratório de Computação Gráfica e do Grupo de Realidade Virtual (GRV-UFU) pela colaboração e disseminação do conhecimento.

As amigas Marli e Cinara pelas palavras de solidariedade nas horas mais oportunas.

Aos meus grandes amigos Luciano Ferreira Silva, Ezequiel Roberto Zorzal e Gérson Flávio Mendes de Lima, pela força e por sempre me incentivar nos meus objetivos. Aos amigos: Lilian, Maria Fernanda, Reane, Ana Paula, Webert, dentre tantos outros....

A FEIT/UEMG - Ituiutaba e a UNIUBE – Uberlândia, por permitirem o tempo necessário a este trabalho.

A CAPES pelo auxílio financeiro que possibilitou a realização deste trabalho.

M.R.F.O.

## Sumário

## *Índice de Figuras*

# Índice de tabelas

**Lista de Publicações**

SILVA, Luciano Ferreira ; Zorzal, R. Ezequiel ; Oliveira, Mônica ; CARDOSO, A. ; LAMOUNIER JUNIOR, Edgard A ; MENDES, Elise B ; TAKAHASHI, Eduardo ; Martins Silvia . Realidade Virtual e Ferramentas Cognitivas Usadas como Auxílio para o Ensino de Física. RENOTE. Revista Novas Tecnologias na Educação, v. 06, p. 01, 2008.

Zorzal, R. Ezequiel ; KIRNER, Cláudio ; CARDOSO, A. ; LAMOUNIER JÚNIOR, E. A. ; Oliveira, Mônica; SILVA, Luciano Ferreira . Ambientes Educacionais Colaborativos com Realidade Aumentada. RENOTE. Revista Novas Tecnologias na Educação, v. 06, p. 01, 2008.

Zorzal, R. Ezequiel ; SILVA, Luciano Ferreira ; Oliveira, Mônica; CARDOSO, A. . Aplicação de Jogos Educacionais com Realidade Aumentada.. RENOTE. Revista Novas Tecnologias na Educação, v. 06, p. 01, 2008

OLIVEIRA, M. R. F. ; C., Alexandre ; Lamounier, E.A. . Sistema de Realidade Virtual portátil de Apoio a Educação de crianças surdas - ERLENEN. In: Workshop de Aplicações de Realidade Virtual e aumentada, 2007, Itumbiara. Workshop de Aplicações de Realidade Virtual e aumentada, 2007. v. 1. p. 36-40.

Lista de abreviações:

AEECLSID - *Application Execution Environment Class ID*

API - *Application Programming Interface*

AR - Realidade Aumentada

ARM - *Advanced RISC Machine*

AWT - *Abstract Window Toolkit*

BDS - *BREW Distribution System*

BMP - *Windows Bitmap*

BREW - *Binary Runtime Environment for Wireless*

Brew MP PVS - BREW *Mobile Platform Porting Validation Suite*

CDMA - *Code Division Multiple Access*

CM - Computação Móvel

CDC - Configuração de Dispositivo Conectado

CLDC - Configuração de Dispositivo Conectado Limitado

CPU - *Central Processing Unit*

CSS - *Cascading Style Sheets*

DOM - *Document Object Model*

DSP - *Digital Signal Processor*

GPL - *General Public License*

GNU - acrônimo recursivo para "*GNU's Not Unix*"

GPS - *Global Positioning System*

GUI - *Graphical User Interfaces*

HTML - *HyperText Markup Language*

IA - Inteligência Artificial

ID - Identificação

IDE - *Integrated Development Environment*

IHC - *Interface* Humano Computador

J2ME - *Java Plataform Micro Edition*

J2SE - *Java Standard Edition*

JPG - *File Interchange Format* (a sigla é também conhecida por: JFIF ou JPEG)

JRS - *Java Specification Request*

KVM - Máquina Virtual *Kilo*

LCD - *Liquid Crystal Display*

LCDUI - *Limited Connected Device User Interface*

LWUIT - *Lightweight UI Toolkit*

M2G - *Mobile 2D Graphics*

M3G - *Mobile 3D Graphics API*

MID - *Mobile Information Device*

MIF - *Module Information File*

MMS - *Multimedia Messaging Service*

MIDP - *Mobile Information Device Profile*

MWT - *Micro Windows Toolkit*

OEM - *Original Equipament Manufacturer*

OpenGL ES - *Open Graphics Library for Embedded Systems*

PC – *Personal Computer*

PDA - *Personal Digital Assistant*

RAM - *Random Access Memory*

RMS - *Record Management System*

ROM – *Read Only Memory*

RV - Realidade Virtual

SDK - *Software Development Kit*

SO - Sistema Operacional

SPOT - *Smart Personal Objects Technology*

SVG - *Scalable Vectorial Graphics*

SWT - *Standard Widget Toolkit*

VM – *Virtual Machine*

W3C - *World Wide Web Consortium*

W-CDMA - *Wide-Band Code-Division Multiple Access*

WIPI - *Wireless Internet Platform for Interoperability*

WTK - *Sun Wireless Toolkit*

XML - *eXtensible Markup Language*

# *RESUMO*

Oliveira, Monica R. F. *RV_LWUIT: Um framework para manipulação de cenas 3D em dispositivos portáteis.*, Uberlândia, Faculdade de Engenharia Elétrica – UFU, 2011.
Palavras-Chave: Computação Móvel, Realidade Virtual, IHC, J2ME, *framework*, LWUIT, portabilidade.

Diante das inovações tecnológicas proporcionadas pelo novo paradigma da Computação Móvel, há, cada vez mais, um ambiente propício ao desenvolvimento de software para os mesmos. Grandes dificuldades são encontradas no desenvolvimento de interfaces amigáveis para dispositivos com telas pequenas (Abreu 2005). Pressman (Pressman 2006) considera o projeto de interfaces com o usuário um tema que tem se tornado cada vez mais importante à medida que aumenta o uso de computadores. Além disso, ele afirma que a maior parte das Interfaces Humano Computador (IHC) são realizadas por meio visual, e há uma tendência definida rumo à comunicação pictória (gráfica) no projeto de IHC. O autor também recomenda que o layout visual da interface deva ser baseado numa metáfora do mundo real (Pressman 1995). Este tema expande-se também para o uso de dispositivos móveis, à medida que cresce o desenvolvimento de software para os mesmos. Uma das maneiras de se conseguir esta metáfora do mundo real pode ser com o uso de técnicas de Realidade Virtual (RV). Esta tecnologia é uma avançada interface gráfica entre um usuário e um sistema computacional (Kirner 2004). RV também pode ser estendida para a Computação Móvel, promovendo maior usabilidade ao software desenvolvido para os mesmos. Este trabalho pesquisou as diferentes plataformas de desenvolvimento de software para dispositivos portáteis, investigando suas funcionalidades e adequação ao desenvolvimento de aplicações de Realidade Virtual. Durante este investigação constatou-se a necessidade de uma melhor padronização de arquiteturas que suportam RV em dispositivos móveis, principalmente sobre o requisito de portabilidade. Assim, um padrão de cenas 3D (JRS (*Java Specification Request)* 184 – M3G) de uma plataforma de desenvolvimento (J2ME) foi selecionado para avaliar as limitações da referida portabilidade. Para tanto, este trabalho propõe uma arquitetura que, por meio de um "menor denominador comum" para estas cenas, propicia o desenvolvimento de aplicações de Realidade Virtual para dispositivos portáteis, sem a necessidade de programação da interface de baixo nível da plataforma selecionada. Isto possibilita um melhor ambiente de suporte à portabilidade. Para avaliar a adequabilidade desta proposta, estas necessidades foram encapsuladas numa biblioteca e, posteriormente, agregada a um framework LWUIT, expandindo-o em RV_LWUIT, mantendo o mesmo padrão de portabilidade originalmente proposto pelo mesmo. Como resultados, consegue-se, em ambos os casos, uma redução significativa de código e conseqüente diminuição no tempo gasto para se colocar no mercado (*time to market*) aplicativos com RV para estes dispositivos.

# *ABSTRACT*

Oliveira, Monica R. F. *RV_LWUIT: A Framework for 3D Scenes manipulation on mobile devices.* Uberlândia, Faculty of Electrical Engineering – UFU,2011.
Key-words: Mobile Computing, Virtual Reality, IHC, J2ME, *framework*, LWUIT, Protability.

In face of the technological innovations provided by the new paradigm of Mobile Computing, there is, increasingly, an environment to develop software in this area. Major difficulties are detected in the development of friendly interfaces for devices with small screens (Abreu 2005). Pressman (Pressman 2006) considers the user interface design as a theme that has become very important with the increasing use of computers. Besides, he states that most of the Human Computer Interfaces (IHC) is performed by visual means, and there is a definitive tendency towards pictorial communication (graphic) design in HCI. The author also recommends that the visual interface layout must be based on a real world metaphor (Pressman 1995). This theme extends as well to the use of mobile devices, as its software development grows. One way to achieve this metaphor of the real world is by using Virtual Reality (VR) techniques. This technology is an advanced graphical interface between a user and a computer system (Kirner 2004). VR can also be extended to mobile computing, providing greater usability to the software developed for them. This work has researched different software development platforms for mobile devices, investigating their functionalities and its suitability for the development of Virtual Reality applications. During this investigation, the need for a better standardization of architectures that support VR in mobile device has been recognized, mainly when portability matters are considered. Thus, a standard 3D scene (JRS 184 - M3G) of a development platform (J2ME) has been selected to evaluate the limitations of the referred portability. To achieve such a goal, this work proposes an architecture that, through "lowest common denominator" for these scenes, allows the development of Virtual Reality applications for mobile devices without the need of low-level programming at selected platform. To evaluate the suitability of this proposal, these needs have been embedded into a library and, subsequently, attached to a framework, called LWUIT, expanding it in a new framework (RV_LWUIT), keeping the same portability pattern, originally proposed. As a result, it is possible, in both cases, a significant reduction of code and a consequent reduction in time spent to put on the market (time to market) with VR applications for these devices.

# 1. Introdução

Que a força do medo que tenho, não me impeça de ver o que anseio
Que a morte de tudo em que acredito, não me tape os ouvidos e a boca
Porque metade de mim é o que eu grito, mas a outra metade é silêncio.
O.M.*

## 1.1. Caracterização do tema da pesquisa

Diante do novo paradigma da Computação Móvel (CM) e de suas inovações tecnológicas, cada vez mais há um ambiente propício ao desenvolvimento de *software* para dispositivos portáteis nas mais diversas áreas, como educação, entretenimento, visualização da informação, dentre outras. Porém, projetar e desenvolver *interfaces* usáveis para dispositivos com telas pequenas é um desafio (Matos 2007).

Pressman (Pressman 2006) considera o projeto de *interfaces* com o usuário um tema que tem se tornado cada vez mais importante à medida que aumenta o uso de computadores. Este tema expande-se também para o uso de dispositivos móveis com alto índice de processamento e armazenamento, à medida que cresce o desenvolvimento de *software* para os mesmos. Assim, torna-se imperativa a necessidade de construir *interfaces* amigáveis (de fácil manipulação) a serem disponibilizadas para seus potenciais usuários. *Interfaces* estas que providenciam uma visão fortemente significativa do mundo real do usuário.

Adicionalmente, uma das maneiras de se conseguir esta metáfora do mundo real pode ser com o uso de técnicas de Realidade Virtual (RV). Esta é uma tecnologia de *interface* gráfica avançada entre um usuário e um sistema computacional (Garcia 2001). O objetivo dessa tecnologia é recriar ao máximo a sensação de realidade para um indivíduo, levando-o a adotar essa interação como uma de suas realidades temporais. Para isso, essa interação é realizada em tempo real, com o uso de técnicas e de equipamentos computacionais que ajudem na ampliação do sentimento de presença do usuário (Kirner 2004). Recentes estudos mostram que esta tecnologia pode ser estendida para a Computação Móvel, promovendo maior usabilidade ao *software* desenvolvido para os mesmos (Schaefer 2004; Ito 2008).

Assim, um cenário ideal seria aquele onde aplicativos baseados em Realidade Virtual pudessem ser executados em qualquer tipo de dispositivo móvel. Entretanto, é sabido que atualmente os aplicativos desenvolvidos para a CM atendem apenas a uma classe específica de dispositivos móveis. Este fato impacta negativamente na portabilidade de diversos aplicativos, inclusive aqueles baseados em Realidade Virtual.

É importante também destacar que esforços da comunidade científica têm sido feito no sentido de padronizar e melhorar a interface dos aplicativos para dispositivos móveis, por meio do desenvolvimento de *frameworks*[1] voltados para o suporte à portabilidade (Agrawal 2004; Sourceforge.net 2007; Sampaio 2010). Porém, tais *frameworks* não suportam diretamente (sem programação de ações e movimentações) aplicativos de Realidade Virtual (Sourceforge.net 2007; Araiz 2008; Microsystems 2009).

## 1.2. Atuais categorias de dispositivos

Dispositivos portáteis podem ser agrupados em três grandes categorias: os telefones básicos, os telefones de categoria avançada e os *smartphones* (ou telefones inteligentes) (Kari Pulli 2008). Mesmo dentro destas categorias, há diversas variações, e os limites não são rígidos. Os telefones básicos não são atrativos em termos de visualização de imagens, ficando geralmente restritos a menus de textos. Estes dispositivos possuem ambientes fechados, geralmente com sistemas operacionais

[1] *Framework* – infra-estrutura do esqueleto de implementação específica (fornece a estrutura e o comportamento genéricos para uma família de abstrações de software dentro de um contexto) Pressman, R. S., Ed. (2006). Engenharia de Software, McGrawHill.

proprietários e novas aplicações só podem ser desenvolvidas em associação estreita com os fabricantes dos equipamentos (Kari Pulli 2008). Também muito limitados em termos da sua capacidade de processamento, resolução e tamanho da tela. Esta categoria de telefones não possuem *hardware* ou *software* para apoio de gráficos e a CPU (*Central Processing Unit)*, limitada e simples, restringem o desempenho de aplicativos 3D, apesar de ser possível implementá-los (Kari Pulli 2008).

Os telefones de categoria avançada se apresentam mais atrativos para aplicações gráficas. Estes celulares representam grande parte do mercado em países desenvolvidos, e a maioria deles incorpora J2ME (*Java Micro Edition*) (Kari Pulli 2008). Estes oferecem programação suficiente para a maioria das necessidades das *interfaces* multimídia, incluindo gráficos 3D. Porém, abrangem uma grande variedade de diferenças de desempenho e funcionalidades dos dispositivos portáteis. Normalmente, possuem sistema operacional *Symbian* (Kari Pulli 2008).

A última categoria é o telefone inteligente. A conclusão do conceito de telefone inteligente é a que esses dispositivos evoluem para computadores móveis (Kari Pulli 2008). As principais características desta categoria incluem grandes *displays* de cores vívidas, poderosos processadores, abundância de memória, capacidade multimídia e conectividade de rede. Alguns dos mais recentes dispositivos incorporam também *hardware* dedicado para gráficos 3D. Possuem sistema operacional (SO), tais como *Symbian, Linux* e *Windows Mobile*, o que fornece apoio à instalação de aplicações de terceiros. Praticamente todos os telefones inteligentes possuem suporte para conteúdo em 3D (Kari Pulli 2008).

É importante destacar que em todas as três categorias de dispositivos, o suporte a Java[2] está presente.

## 1.3. Caracterização dos atuais *frameworks*

Os dispositivos móveis (mesmo os da categoria avançada) ainda são menos capazes do que computadores *desktop* (Kari Pulli 2008). Eles possuem velocidade mais baixa, telas menores em tamanho e em resolução, há menos memória para rodar e armazenar programas, e períodos curtos de autonomia de bateria. Mas as melhorias em

[2] Java – *TradeMark* (marca registrada).

computação e comunicação destes dispositivos vêm motivando o desenvolvimento de aplicativos para dispositivos portáteis que contemplem apresentação de imagens 3D (Filho 2005; Sourceforge.net 2007; Microsystems 2009). Entretanto, visualizar cenas 3D em dispositivos portáteis ainda é uma tarefa muito complicada devido à grande potência computacional necessária para alcançar um desempenho utilizável, restringindo seu uso a equipamentos da classe avançada.

Dentre os esforços para visualização de cenas 3D nos dispositivos da classe avançada, surgiram diversos *frameworks* com o objetivo de prover recursos para construção de aplicações para dispositivos móveis, alguns provendo melhorias no desenvolvimento de *interfaces* (Virkus 2005; EnoughSoftware 2007; Araiz 2008), e outros fornecendo recursos de RV (Filho 2005; Sourceforge.net 2007; Araiz 2008).

Entretanto, a natureza dos recursos oferecidos é dependente de programação. Isto é, todos os requisitos para suportar um ambiente de RV no dispositivo móvel, tais como: botões de controle, velocidade de translação, rotação e *zoom* (com seus respectivos parâmetros) etc, precisam ser previamente programados. Isto demanda um número maior de horas de programação por parte dos desenvolvedores de cada *framework* para lançar no mercado (*time to market*).

Ainda, estudos mostram que a inadequação entre as soluções implementadas nas *interfaces* de *software* para dispositivos portáteis e as características dos usuários finais causa problemas de usabilidade que merecem ser estudados mais profundamente (Santos, Freitas et al. 2009). As trocas frequentes que os usuários realizam de seus aparelhos por outros com SO diferentes e sem estudos centrados no usuário vêm a dificultar o uso desses dispositivos. Para aumentar o nível de usabilidade destes sistemas, diversos autores apontam para a necessidade de aplicação de padrões de projetos (Abreu 2005).

Um padrão de projeto é uma abstração que prescreve uma solução de projeto para um problema específico (Pressman 1995). No caso da Computação Móvel, além das dificuldades de desenvolvimento de padrões para IHC, há também dificuldades no que diz respeito a portabilidade de *software*. Estas *interfaces* estão cada vez mais sofisticadas e tem-se tornado tão comuns que uma grande variedade de padrões de projetos de *interfaces* com o usuário tem surgido nesta área.

A criação de padrões de projetos, sejam de usabilidade, sejam de programação, dentro deste novo paradigma da Computação Móvel é ainda um esforço da comunidade de pesquisadores (Caraciolo 2009). Isto se dá pela forte presença de restrições de

capacidade de processamento, de memória, de tamanho e resoluções de telas, autonomia de bateria, possibilidade de conexão com internet e as diversas plataformas existentes. Naturalmente, isto constitui um empecilho à criação de programas realmente portáveis entre estes dispositivos.

## 1.4. Contribuição da Tese

A abordagem aqui desenvolvida enquadra-se no âmbito de três grandes áreas: *Interface Humano Computador* (mais especificamente Realidade Virtual), Desenvolvimento de Aplicações para Dispositivos Portáteis e Padrões de Projetos, conforme apresentado na Figura 1.


**Figura 1 – Relacionamento entre as grandes áreas envolvidas na pesquisa.**

Ao final deste trabalho, pretende-se apresentar um conjunto de bibliotecas que possam ser usadas de forma padronizada no desenvolvimento de aplicativos de Realidade Virtual para dispositivos móveis.

Considerando os fatos supracitados, o objetivo desta tese é investigar e propor inovações em técnicas computacionais que suportem a portabilidade de aplicativos, baseados em técnicas de Realidade Virtual, em diferentes classes de dispositivos móveis. Para tanto, as seguintes metas são propostas:

**ETAPA 1.**Identificar as atuais categorias de dispositivos móveis e suas técnicas de programação e suporte à portabilidade, destacando suas vantagens e desvantagens;

**ETAPA 2.**Projetar e desenvolver uma plataforma (protótipo) que melhor suporte aplicativos de Realidade Virtual para as classes de dispositivos estudadas acima;

**ETAPA 3.**Investigar os diferentes *frameworks* que suportam portabilidade entre diferentes categorias de dispositivos, identificando suas limitações para aplicativos de Realidade Virtual;

**ETAPA 4.**Desenvolver bibliotecas de aplicativos de Realidade Virtual e avaliar a adequabilidade da mesma nos diferentes *frameworks* para dispositivos móveis.

## 1.5. Estrutura do trabalho

Para executar este trabalho , o mesmofoi assim dividido: no Capítulo 2 foi feito um estudo dos diversos ambientes de desenvolvimento de aplicações móveis (plataformas de desenvolvimento), buscando aquela em que há a maior possibilidade e/ou facilidade de testes das aplicações móveis desenvolvidas neste trabalho. Dentre as opções para o desenvolvimento de aplicativos para dispositivos móveis pesquisadas, inclui-se a plataforma BREW[3] da *QUALCOMM*, o padrão J2ME da *Sun Microsystems*, a adaptação da *Microsoft*, o *Windows CE* para Computação Móvel, além dos recentes *Android* e *iPhone*.

O Capítulo 3 traz um estudo das API's (*Application Programming Interface)* gráficas *OpenGL* ES e M3G para dispositivos portáteis inteligentes. As principais características desta categoria incluem grandes *displays* de cores vívidas, poderosos processadores, abundância de memória, capacidade multimídia e conectividade de rede e alguns incorporam também *hardware* dedicado para gráficos 3D. Praticamente todos os telefones inteligentes possuem Java, e ambas as API`s gráficas *OpenGL* ES e M3G estão disponíveis para conteúdo em 3D. Um estudo destas duas APIs, mostrando as suas características no que diz respeito ao desenvolvimento de ambientes em Realidade Virtual é realizado, levando a solução da segunda etapa proposta neste trabalho.

Já o Capítulo 4 faz um levantamento dos esforços no desenvolvimento de *framework*s de GUIs (*Graphical User Interfaces)*, demonstrando as características particulares de cada uma das implementações e selecionando uma para que seja então feita a modificação proposta por este trabalho.

[3] BREW – TradeMark (marca registrada).

O Capítulo 5 é a modelagem e a implementação das soluções propostas por este trabalho, mostrando detalhes do encapsulamento da chamada das cenas em RV e também a modificação do *framework* LWUIT, acrescentando as funcionalidades de RV sem perda das características de portabilidade fornecidas por este. Este capítulo também apresenta e discute os padrões de projeto utilizados nesta implementação.

Já o Capítulo 6 traz dois estudos de casos, um utilizando apenas a biblioteca desenvolvida e outro utilizando o *framework* LWUIT modificado por este trabalho (propondo um *software* de auxílio aos portadores de necessidades especiais (surdos) – ERLENEN).

O Capítulo 7 apresenta as conclusões e oferece sugestões de trabalhos futuros que podem ser desenvolvidos a partir deste trabalho. A Figura 2 traz a metodologia desta pesquisa para alcançar os objetivos propostos.

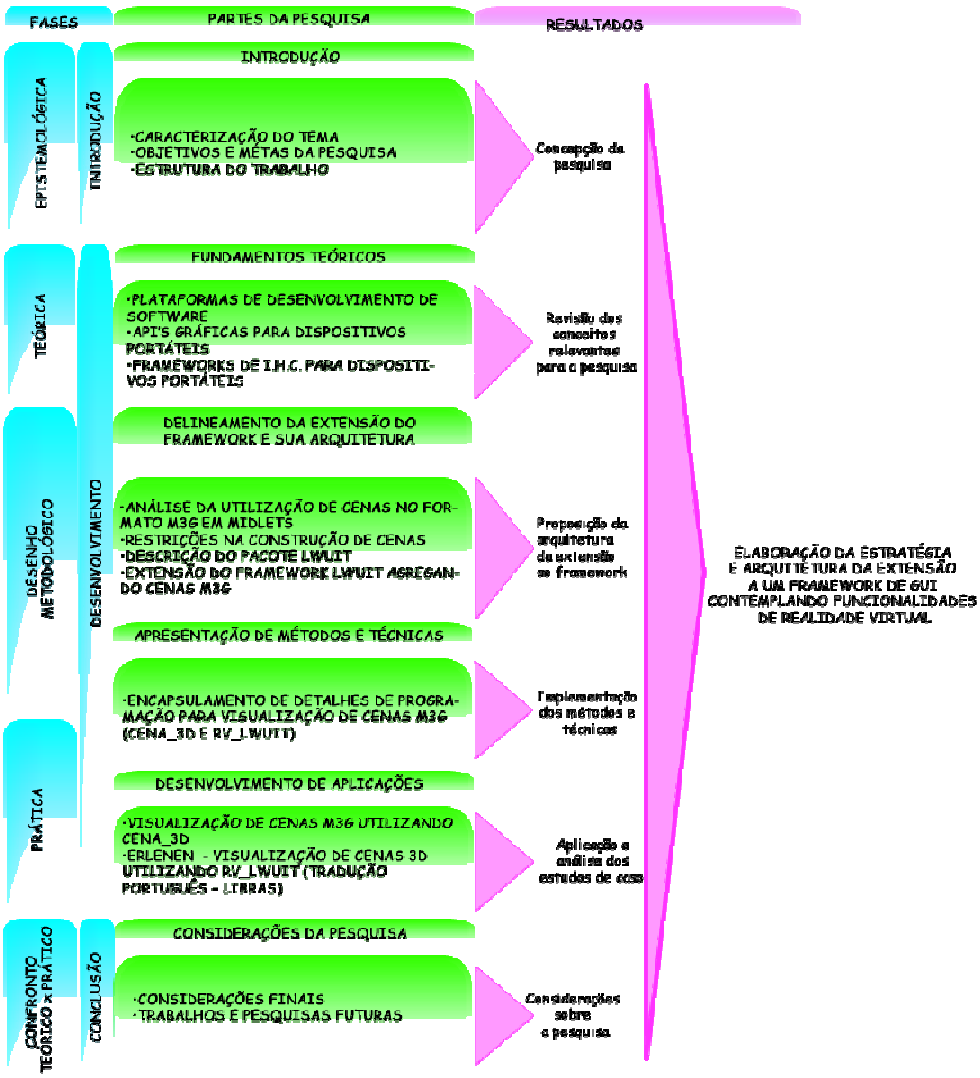



**Figura 2 – Representação das fases da pesquisa.**

# 2. Ambientes de programação para dispositivos móveis

Que a música que ouço ao longe seja linda ainda que tristeza
Que a mulher que eu amo seja pra sempre amada, mesmo que distante
Porque metade de mim é partida, mas a outra metade é saudade.
O.M.*

As opções para o desenvolvimento de aplicativos para dispositivos móveis, no lado do cliente usuário do equipamento, incluem a plataforma BREW da *QUALCOMM*, o padrão J2ME da *Sun Microsystems* e a adaptação da *Microsoft*, o *Windows CE* para Computação Móvel. Este capítulo discutirá as vantagens e desvantagens destas plataformas, diferenciando-as e definindo uma delas para o desenvolvimento deste trabalho.

## 2.1. BREW - *Binary Runtime Environment For Wireless - Qualcomm*

BREW é a sigla de *Binary Runtime Environment for Wireless* (ambiente binário de tempo de execução para aplicativos sem fio) (*Qualcomm* 2004). No nível básico, a plataforma BREW funciona como uma *interface*, ou camada de abstração, com relação ao sistema operacional do *chip* incorporado a um aparelho telefônico. É como um análogo à API Win32 para o *Microsoft Windows* no ambiente PC (*Personal Computer).* A plataforma BREW consiste em um conjunto de bibliotecas binárias compiladas, ligadas para execução nativa e otimizada de forma a permitir que os aplicativos

aproveitem os recursos e serviços de comunicação móvel. Ela controla o fluxo de eventos de e para os aplicativos, iniciando, interrompendo, suspendendo e retomando a execução em resposta a eventos apropriados. O ambiente de execução do BREW descobre aplicativos e quaisquer extensões associadas em tempo de execução (*Qualcomm* 2004).

Também provê acesso às funções de baixo nível subjacentes do dispositivo de comunicação móvel através de sua *interface* de aplicativos. Os desenvolvedores podem usar o BREW SDK (*Software Development Kit)* e a linguagem C/C++ e Java para escrever *software* para o dispositivo sem precisar ter qualquer conhecimento de sua complexidade. Como a *interface* de aplicativos não muda, portar aplicativos de um dispositivo BREW para outro se torna simples. Essas *interfaces* são coleções de funções relacionadas a um serviço específico.

As APIs são encapsuladas em classes de objetos com contagem de referências. Cada *interface* é associada a uma identificação (ID) de 32 bits exclusiva, chamada de AEECLSID (*Application Execution Environment Class ID*, ID de classe de ambiente de execução de aplicativos) (*Qualcomm* 2004).

Um desenvolvedor, visando aplicativos BREW, deve implementar todas as funções da *interface* chamada de *IApplet*. As classes do aplicativo são todas armazenadas em arquivos de módulo binários (.mod). O aplicativo também tem associados arquivos de recursos e um arquivo de informações do módulo (MIF, *Module Information File*). O MIF contém,além de outras informações, títulos e ícones a serem usados pelos *IApplets* e informações de registro e segurança/privilégios para serviços de nível de sistema acessados pelo aplicativo (E/S de arquivo, E/S de rede, assim por diante). Tais informações são usadas pelo BREW para descobrir aplicativos e extensões e garantir as regras necessárias em tempo de execução (*Qualcomm* 2004).

Para desenvolver, utilizando essa plataforma, que na verdade é um conjunto de APIs que funciona associada ao *Visual Studio*, é possível utilizar as linguagens de programação C/C++ e Java, embora a linguagem mais utilizada seja C. Ao contrário do J2ME, um desenvolvedor pode escrever código em C, que será compilado diretamente para um processador ARM (*Advanced RISC Machine,* desenvolvido pela ARCOM *Computers*, uma máquina RISC) em um ambiente altamente restrito.

O modelo de negócios do BREW contempla o desenvolvimento, a implantação, a descoberta, a aquisição, o *download* e o gerenciamento de aplicativos de forma transparente ao usuário. Esta é a função do BREW *Distribution System* (BDS).

Mediante solicitação de uma operadora de rede, aplicativos passam por um processo de testes que assegura que aplicativos perniciosos não sejam introduzidos na rede da operadora. O BDS permite que desenvolvedores submetam aplicativos testados e assinados digitalmente para um mercado virtual global de operadoras de redes. Todos os aplicativos BREW são assinados digitalmente por certificados da *VeriSign* para o desenvolvedor, da *QUALCOMM* (empresa que produz os *chipsets* dos celulares de tecnologia CDMA (*Code Division Multiple Access)* e W-CDMA (*Wide-Band Code-Division Multiple Access)* ) e da operadora de rede. O ambiente de execução procura por essas assinaturas antes de permitir que um aplicativo seja executado em seu ambiente (*Qualcomm* 2004). A Figura 3 mostra o modelo de negócio do BREW.

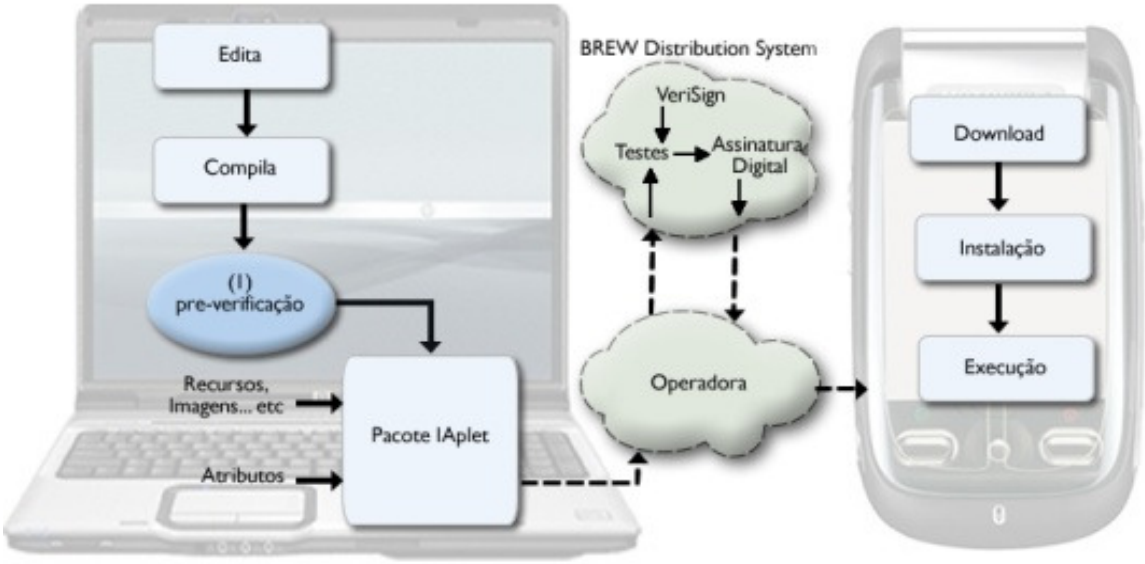



**Figura 3 - Modelo de negócio da Qualcomm (Qualcomm 2004).**

Desenvolver utilizando BREW dá ao programador excelentes oportunidades de negócios, uma vez que a empresa operadora de telefonia é responsável por avaliar e disponibilizar os *software* a todos os seus clientes. Não há outro meio de conseguir *software* a não ser via operadora. A comunidade de desenvolvedores é bastante fechada, e o modelo de negócio inviabiliza esta pesquisa, pois as operadoras exigem as certificações, e estas são demoradas. A instalação dos aplicativos é feita via *download* das operadoras, que não o fazem sem as certificações.

Atualmente, a empresa disponibiliza um SDK para *smartphones Android* que incorpora funções de Realidade Aumentada (AR), disponível para *download* em versão beta pública. Este SDK utiliza a tecnologia de visão computacional para alinhar os

gráficos com os objetos subjacentes e possui suporte para os objetivos da imagem, os marcadores de quadro, botões virtuais e objetos 3D simples (Qualcomm 2010).

Também recentemente, a empresa passou a fornecer um SDK com suporte para todo conjunto de API's C / C + +, *Adobe Flash*, a linguagem de marcação TrigML e Lua. A plataforma ainda oferece API`s que permitem que um aplicativo possa comunicar com o *Java Application Manager* do dispositivo, que gerencia o *Brew*, o Java e também *Flash* (Brew 2010). Observa-se que, apesar de abrir a plataforma para outras linguagens e também para instalação de aplicativos sem certificações, a empresa procurou manter seu canal de distribuição de *software*. Entretanto, para um aplicativo ser distribuído neste canal, permanece a necessidade de certificação *VeriSign* (Brew 2010).

Está presente também o aplicativo de estabilidade, que pode ser executado no simulador ou carregado e executado em qualquer dispositivo Brew MP. O aplicativo de Estabilidade faz parte do SDK do *Brew* MP e os *kits Brew* MP PVS (BREW *Mobile Platform Porting Validation Suite*) (Brew 2010) e visa fornecer ao programador independente informações sobre a estabilidade do aplicativo, a memória total em uso, intervalos de toques na tela (quando é usado este recurso), *timestamp* de início e fim da aplicação, dentre outras informações.

## 2.2. .NET *Micro Framework – Windows CE*

O *Microsoft* . NET *Micro Framework*, anteriormente conhecido como *Smart Personal Objects Technology* (SPOT), foi criado de baixo para cima para os pequenos dispositivos, ou seja, pensando primeiramente no *hardware* que o suportará, até o usuário final. A gama de funcionalidades nesses dispositivos, geralmente, é limitada por uma combinação de custos, memória, capacidade e poder de processamento. Foi utilizado em alguns dos menores dispositivos, incluindo relógios e dispositivos de navegação GPS (*Global Positioning System)* (*Microsoft* 2004).

Sendo orientado para estes tipos de dispositivos, torna-se possível para a plataforma .NET *Micro Framework* permitir o desenvolvimento e utilização sobre recursos significativamente reduzidos. Oferece também o gerenciamento de *interfaces* para o seu código, de modo a maximizar a duração da bateria dos seus dispositivos (*Microsoft* 2004).

A *Microsoft* desenvolveu o .NET *Micro Framework* com uma intenção em mente: construir aplicações comerciais. Estas aplicações devem exibir, coletar, processar e transmitir informações. Estes aplicativos devem oferecer aos utilizadores uma razão para transportar um dispositivo. Os dados com que estas trabalham, podem ser locais, afastados, ou uma combinação de ambos (Agrawal 2004). Ele simplifica o desenvolvimento da aplicação em dispositivos inteligentes, que inclui o *Pocket* PC, *Pocket* PC 2002, *Pocket* PC *Phone Edition* e outros dispositivos[4] rodando *Windows CE* .NET 4.1.

O .NET *Micro Framework* tem dois componentes principais: a linguagem comum (biblioteca de classes) e o *runtime* .NET *Micro Framework*. O *runtime* é a sua base e é responsável pela gestão de código em tempo de execução, e proporciona serviços essenciais, como a gerência de memória. A biblioteca de classe é um conjunto de classes reutilizáveis, úteis para desenvolver aplicações de forma rápida e fácil.

Portanto, basta que se utilize as técnicas de codificação para que as aplicações possam ser executadas em um *Pocket* PC ou em outras plataformas, como um telefone celular (Agrawal 2004).

O .NET *Micro Framework* oferece uma gama de ferramentas de alto nível e de apoio, incluindo os seguintes itens: linguagem globalizada, *thread*, coleta de lixo, gerenciamento de *drivers* para LCDs (*Liquid Crystal Display)*, gráficos primitivos, imagens tipo BMP e JPG (*File Interchange Format)*, entre outros.

As ferramentas do .NET *Micro Framework* estão perfeitamente integradas com o *Visual Studio*. Essa integração acontece em todo o processo de desenvolvimento, a partir da criação de projetos usando *templates* .NET *Micro Framework*, através de requerimento de desenvolvimento no computador usando o .NET *Framework* extensível ao *Micro* emulador, o *download* de aplicativos para o dispositivo executando. NET *Micro Framework* e, finalmente a depuração (em *Visual Studio*), o código que será executado diretamente no dispositivo (*Microsoft* 2004).

O .NET *Micro Framework* é uma ferramenta aplicada somente a dispositivos que utilizam como sistema operacional o *Windows CE* .NET 4.1, o que exclui a maioria de dispositivos celulares existentes no mercado. Além disso, é uma ferramenta proprietária, o que encarece os custos de qualquer pesquisa.

[4] Dentre os *smartphones* e PDA's somente os que rodam *Windows* CE. Dispositivos do tipo *Palm* rodam *Palm OS* e não são compatíveis com .NET *Micro Framework*.

## 2.3. *Android - Google*

Aplicativos do *Android* estão escritos na linguagem de programação Java ou C/C++. O *Android* inclui um conjunto de bibliotecas que fornece a maioria das funcionalidades disponíveis nas principais bibliotecas da linguagem de programação Java. Cada aplicação *Android* roda em seu próprio processo, com sua própria instância da máquina virtual *Dalvik*. Ela executa os arquivos em *Dalvik* executável (. DEX), formato que é otimizada para o consumo de memória mínimo. A VM (*Virtual Machine)* é baseada em registradores, e executa classes compiladas por um compilador de linguagem Java que foram *Transform*adas para o formato *dex*. Incluído pela "dx" ferramenta. A *Dalvik* VM invoca o *kernel* do Linux para a funcionalidade subjacente como encadeamento e de baixo nível de gerenciamento de memória (Android 2010).

Por padrão, cada aplicação é executada em seu próprio processo de Linux. *Android* começa o processo quando qualquer código do aplicativo precisa ser executado, e encerra o processo quando ele não é mais necessário e os recursos do sistema são exigidos por outras aplicações. Cada processo tem sua própria máquina virtual Java (VM), é desta forma que o código do aplicativo funciona de forma isolada do código de todas as outras aplicações. Ainda, para cada aplicação é atribuído um ID exclusivo de usuário Linux. As permissões são definidas para que os arquivos do aplicativo sejam visíveis apenas para a aplicação em si, apesar de existirem formas de exportá-los para outras aplicações também (Android 2010).

É possível organizar para duas aplicações para compartilharem o mesmo ID, neste caso, serão capazes de ver os arquivos uns dos outros. Para conservar os recursos do sistema, as aplicações com a mesma identificação também pode mandar executar no Linux mesmo processo, compartilhando a mesma VM.

## 2.4. *Symbian - J2ME*

O Sistema Operacional *Symbian* (*Symbian OS* ) é um sistema criado para rodar nos dispositivos móveis "multimídia" com suporte para câmeras fotográficas, MMS, *wireless*, *bluetooth*, entre outras funções. É predominantemente baseado em um ambiente gráfico bastante simples. Sua grande preocupação é evitar ao máximo o desperdício dos recursos do celular, como bateria e memória.

O *Symbian* é um consórcio de várias empresas (Nokia, Siemens, Samsung, Ericsson, Sony Ericsson e Panasonic), fundado em 1998 e que está em plena operação até os dias de hoje. Atualmente a Nokia, que adquiriu a quase totalidade de suas ações em dezembro de 2008. Empresas não-pertencentes ao consórcio podem licenciar o Sistema Operacional para utilização em seus produtos.

A grande maioria dos celulares modernos de hoje são operados pelo sistema operacional da Symbian. Ele é um sistema totalmente modular, e permite que cada empresa crie sua própria interface. Portanto este sistema não tem uma interface definida. Pode ser um simples sistema de textos em telas monocromáticas, ou um completo sistema operacional tão potente como o *PalmOS* ou *PocketPC* que já pode ser encontrado nos *SmartPhones* da Nokia, SonyEricsson, Foma, Siemens, Motorola, dentre outras.

Symbian OS é um sistema operacional muito versátil, permite o desenvolvimento de aplicativos em diversas linguagens como: *Symbian C/C++*, Java *Micro Edition* (J2ME), *FlashLite*, HTML5, *Perl*, *Python*, *Ruby*, Lua. Dentre estas, a mais comum é a J2ME.

A J2ME é destinada diretamente aos dispositivos com poder limitado (baixo poder de processamento, pouca memória, tela muito pequena, entre outros), dotando-os dos recursos inerentes da linguagem e da plataforma Java. Os recursos dentro da *Micro Edition* variam bastante de acordo com os diversos *hardware*s.

Desde o seu lançamento, em 1995, a Linguagem de Programação JAVA ampliou o seu alcance para além dos simples computadores pessoais, e sua inclusão mais significativa é a família *Micro Edition* (J2ME), que objetiva 'ferramentas de informação', dentre elas os aparelhos celulares (Muchow 2004).

São duas as configurações correntes da plataforma Java (J2ME), e as características típicas que as diferem são descritas a seguir (Teixeira 2005):

- Configuração de Dispositivo Conectado (CDC) - para dispositivos com conectividade de rede e largura de banda possivelmente persistente e alta. A máquina virtual Java, neste caso, tem a mesma especificação do J2SE (*Java Standard Edition*).
- Configuração de Dispositivo Conectado Limitado (CLDC) - para dispositivos de *interface* com o usuário restrita, baixa autonomia (normalmente alimentado por bateria) e conectividade de rede sem fio, com largura de banda baixa e acesso intermitente. Para a CLDC foi

desenvolvido uma implementação de referência de uma máquina virtual, chamada de *K Virtual Machine* (*Máquina Virtual Kilo* – KVM)

Segundo a especificação da J2ME, a KVM é designada para ser pequena em tamanho de código e com baixo requisito de memória, modular (podendo ser acrescida de partes à medida em que for necessário – para dispositivos embarcados) e customizável, enxuta e portável e tão completa e rápida quanto possível.

O conceito de Perfil visa tratar da ampla variedade de recursos dentro de uma mesma configuração. Ele fornece as bibliotecas para desenvolvimento de aplicativos para um tipo particular de dispositivo. O *Mobile Information Device Profile* (perfil de dispositivo de informação móvel - MIDP) define as API's (*interfaces* de programação de dispositivos), levando em consideração as limitações de tela e de memória dos dispositivos móveis (Muchow 2004).

A arquitetura J2ME é apresentada na Figura 4, e começa com um sistema operacional hospedeiro como base, seguido da máquina virtual Java para dispositivos CLDC, a KVM (*Kilo Virtual Machine*). Em seguida vêm às bibliotecas básicas CLDC e finalmente os perfis são a última camada, fornecendo as ferramentas particulares de uma família de dispositivos (Muchow 2004).


**Figura 4 - Arquitetura J2ME para dispositivos celulares (Muchow 2004).**

É importante observar os requisitos de *software* e de *hardware* mínimos dentro desta arquitetura. Com exceção da memória disponível, a CLDC não possui requisitos de *hardware* específicos. Os requisitos de *software* também são mínimos: o sistema operacional deve ser capaz de executar a KVM e gerenciar aplicativos, incluindo a seleção e ativação e a capacidade de remover aplicativos Java do dispositivo (devMedia 2007).

As diferenças básicas entre as máquinas virtuais JVM e KVM podem ser observadas na Tabela 1 – Diferenças básicas entre as máquinas virtuais Java (Muchow 2004).. Estas levam às diferenças fundamentais entre as linguagens J2SE e J2ME (Muchow 2004).

**Tabela 1 – Diferenças básicas entre as máquinas virtuais Java (Muchow 2004).**

| | JVM | KVM |
|---|---|---|
| **Matemática de ponto flutuante** | **Presente** | **Ausente** |
| **Suporte para chamada de métodos e API's nativas de outras linguagens de programação** | **Presente** | **Ausente** |
| **Carregador de classe** | **Ausente** | **Presente** |
| **Reflexão** | **Presente** | **Ausente** |
| **Threadgroup** | **Presente** | **Ausente** |
| **Finalização** | **Presente** | **Ausente** |
| **Referências fracas** | **Presente** | **Ausente** |

Visualmente, a arquitetura de um MIDP pode ser representada na Figura 5. O MID (*Mobile Information Device)*, dispositivo de informação móvel representa o *hardware* dos equipamentos portáteis. Logo acima dele, o seu sistema operacional. Após, vem o CLDC, que é onde está a Máquina Virtual K (KVM). As APIs MIDP rodam sobre a configuração CLDC, como pode-se visualizar na Figura 5. As classes OEM (*Original Equipament Manufacturer*) são classes não definidas pelo MIDP e que podem ser utilizadas para funcionalidades específicas de determinado aparelho. Quanto aos aplicativos, uma *MIDlet* é aquela aplicação que usa APIs definidas pelo MIDP e CLDC, e são portáveis entre vários aparelhos (um aplicativo MIDP é uma *MIDlet*). As aplicações nativas são as que estão implementadas diretamente no sistema e não são escritas em Java (Muchow 2004).

O gerenciador de aplicativo comunica-se com as *MIDlet*'s através de métodos dessa classe. Seu ciclo de vida é sempre considerado como estando em um dos três estágios: ativa – quando está em execução; pausa – quando é colocada em estado de pausa, e libera o máximo de recursos possível, e destruída, quando já liberou todos os recursos que adquiriu e foi destruída. Métodos específicos existem para atender a estes três estados: *startApp(), pauseApp() e destroyApp().* A Figura 6 apresenta os três estados do ciclo de vida das *MIDlet*'s.

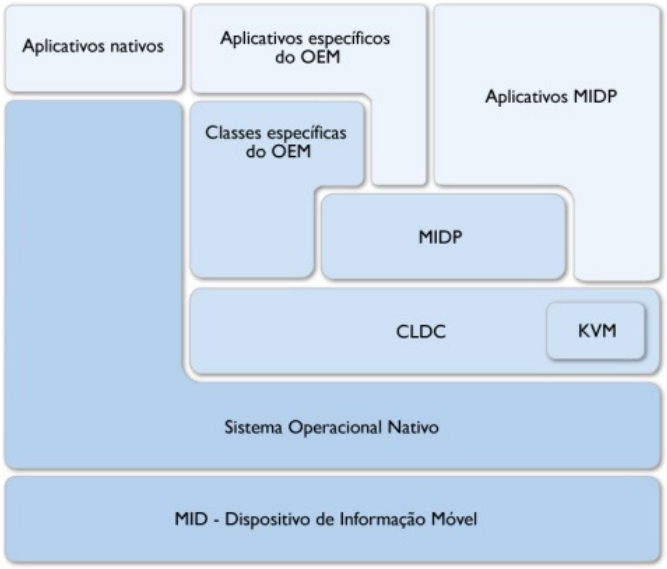



**Figura 5 – Arquitetura MIDP adaptada de (Muchow 2004).**

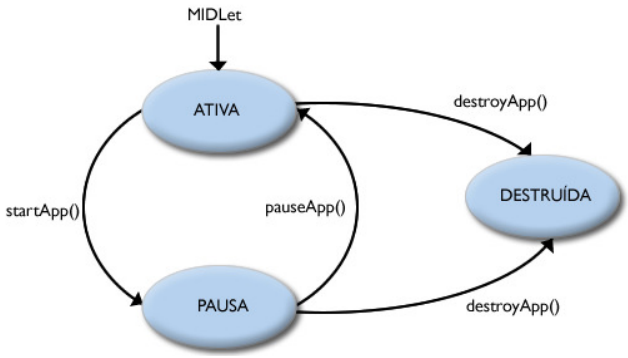



**Figura 6 – Ciclo de vida das MIDlet's (Muchow 2004).**

O reconhecimento de um evento em J2ME passa por três etapas: o reconhecimento pelo *hardware* de que algo ocorreu, a notificação do *software* do dispositivo (gerenciador de aplicativos) de que o evento ocorreu e o recebimento pela *MIDlet* de uma mensagem do gerenciador de aplicativos, onde as informações de como proceder a este evento estão descritas (Microsystems 2007).

O MIDP inclui duas subclasses de *Displayable*, *Screen* e *Canvas*. Os objetos da classe *Screen* são todos componentes de alto nível da *interface* com o usuário, enquanto o objeto *Canvas* é usado para elementos gráficos personalizados e tratamento de eventos de baixo nível. Define também um conjunto de constantes, ações de jogos, para facilitar o tratamento de eventos relacionados a jogos. O perfil inclui ainda um conjunto de métodos para interagir com dispositivos de ponteiros como um mouse ou uma tela de toque.

As classes *Canvas* e *Graphics* são as classes são mais adequadas para a criação de aplicações do tipo jogos. As APIs 2D e 3D devem usar a API MIDP de baixo nível como vinculativo para a exibição de suas imagens (Microsystems 2007).

Os principais sistemas operacionais presentes em dispositivos portáteis, assim como as *interfaces* de desenvolvimento utilizadas em cada um deles pode ser vista na Tabela 2 (Microsystems 2005).

Uma análise desta tabela mostra que a transparência de *hardware* e sistemas operacionais é conseguida utilizando-se as linguagens de programação C/C++ ou Java. Entretanto, J2ME como plataforma e linguagem de programação, continua a ser a mais utilizada e mais aceita (Baudisch 2007).

**Tabela 2 – Interfaces de desenvolvimento nas diversas plataformas portáteis (adaptada de Microsystems 2005).**

| | WINDOWS CE | QUALCOMM | SYMBIAN OS | ANDROID |
|---|---|---|---|---|
| IDE's | MS *Visual Studio*, Borland Developer Studio | *BREW* | GCC (GNU Compiler Collection), *Scene3D*, Eclipse, *Sun Wireless Toolkit*, etc. | Eclipse, NetBeans, etc. |
| Linguagens de programação | C/C++, C#, .NET, Java | C/C++, Java, Lua | C/C++, Java | C/C++, Java |

O modelo de negócios do J2ME, mostrado na Figura 7, contempla apenas o desenvolvimento, testes em simuladores, a implantação e testes nos equipamentos móveis, permitindo ao programador validar junto a usuários os seus aplicativos com maior facilidade.

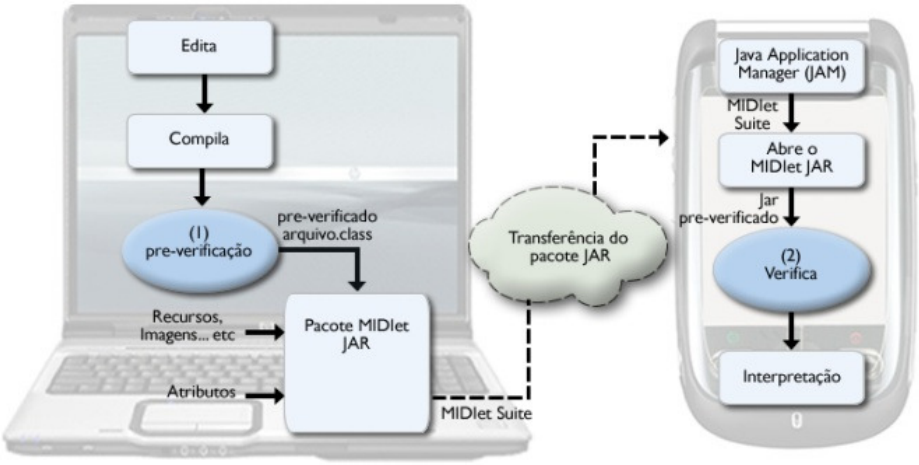



**Figura 7– Modelo de negócios J2ME.**

Após quase dez anos, a máquina virtual da Java ME, a Kilo *Virtual Machine* (KVM), encerrou o seu ciclo de vida. Este fim já era esperado, e ela está, atualmente, sendo substituída pela CLDC *HotSpot Virtual Machine*. Um projeto mais moderno e robusto, e que consegue atender melhor as atuais demandas por desempenho. Esta última é compatível com a primeira, ou seja, aplicativos construídos para funcionar em KVM, funcionam também na *HotSpot Virtual Machine*.

O que prolongou um pouco mais a vida da KVM foi o fato da *Sun* continuar utilizando-a em seus emuladores até os dias de hoje. Até a última versão do WTK (*Sun Wireless Toolkit*), a 2.5.2 (Oracle 2010), todos os emuladores ainda eram baseados na KVM. No entanto, com a chegada do Java ME *Platform* SDK 3 a *Sun* finalmente "aposentou" a KVM, trazendo de uma vez por todas, a CLDC *HotSpot Virtual Machine* também para os seus emuladores, inclusive para a configuração CDC. Com esta mudança, o comportamento das aplicações no emulador ficou mais parecido ao encontrado nos dispositivos reais.

Dentre as principais vantagens da CLDC *HotSpot Virtual Machine*, comparada à KVM, estão a compilação dinâmica das instruções de *bytecode* em instruções nativas, menor consumo e fragmentação de memória, maior economia da bateria, dentre outras. Em termos de números, a execução de uma instrução compilada dinamicamente, chega a ser cinquenta vezes mais rápida do que uma instrução interpretada.

## 2.5. Conclusões

Foram apresentadas as quatro principais plataformas de desenvolvimento para dispositivos móveis.

A plataforma BREW consiste em um conjunto de bibliotecas binárias compiladas, ligadas para execução nativa e otimizada de forma a permitir que os aplicativos aproveitem os recursos e serviços de comunicação móvel. Para desenvolver utilizando essa plataforma, é possível utilizar as linguagens de programação C/C++ e Java, embora a linguagem mais utilizada seja C.

O .NET *Micro Framework*, desenvolvido pela *Microsoft* com a intenção de construir aplicações que exibem, coletam, processam e transmitem informações. Estes aplicativos devem oferecer aos utilizadores uma razão para transportar um dispositivo. Os dados com que estes trabalham podem ser locais, remotos, ou uma combinação de

ambos (Agrawal 2004), simplificando o desenvolvimento da aplicação. Os dispositivos devem estar rodando *Windows CE* .NET. É necessário, para construir aplicativos que rodam sob esta plataforma, o *Visual Studio* .NET. Pode-se, então, construir aplicações usando Visual C# .NET, *Visual Basic* .NET, ou ambos.

Ainda, há o J2ME, que é destinado diretamente aos dispositivos com poder limitado, dotando-os dos recursos inerentes da linguagem e da plataforma Java. Os recursos dentro da *Micro Edition* variam bastante de acordo com os diversos *hardware*s. O J2ME trabalha com conceito de configurações (CLDC) e perfis (MIDP), definindo os recursos disponíveis em cada um deles, permitindo ao desenvolvedor escolher a configuração e o perfil da sua aplicação. É importante observar que os requisitos de *software* e de *hardware* dentro da arquitetura J2ME são mínimos. Com exceção da memória disponível, a CLDC não possui requisitos de *hardware* específicos. Os requisitos de *software* também são mínimos: o sistema operacional deve ser capaz de executar a KVM ou a *HotSpot Virtual Machine* e gerenciar aplicativos, incluindo a seleção e ativação e a capacidade de remover aplicativos Java do dispositivo (devMedia 2007).

Uma ampla quantidade de telefones celulares estão equipados com Java *Micro Edition*, tornando-se, de longe, a mais amplamente implantada plataforma no mundo. Praticamente todos as plataformas Java móveis fornecem a mesma solução para gráficos 3D: a API M3G. A plataforma Java dá segurança, estabilidade, portabilidade e é praticamente de graça, graças ao *design* da sua máquina virtual. Evidentemente, a segurança e a arquitetura robusta da máquina virtual de Java têm o seu preço: baixo desempenho nas aplicações e acesso limitado às capacidades da plataforma. O isolamento de aplicações a partir do *software* e do *hardware* subjacente bloqueia o acesso às bibliotecas e ao sistema nativo excluindo quaisquer otimizações de baixo nível. Outros dispositivos como *Palms* (que são dispositivos portáteis, mas não são telefones móveis), que possuem sistema operacional Palm OS, permitem a instalação da máquina virtual Java, assim como as plataformas BREW e *Windows CE* também permitem a instalação da JVM.

Aplicativos do *Android* estão escritos na linguagem de programação Java via bibliotecas desenvolvidas pela *Google* (Wikipédia 2010). Por ser bastante recente, o número de dispositivos e operadoras trabalhando com esta plataforma ainda é pequeno. Como o foco do trabalho é o desenvolvimento de um *framework* para incluir

características de RV de forma simples, realizando testes de portabilidade, a escolha da linguagem de programação Java atende às diversas plataformas incluindo esta.

O grande apelo comercial do *iPhone* faz com que o mesmo seja objeto de desejo de 6 entre 10 consumidores de *smartphones* (World/EUA 2010). Algumas soluções para esta plataforma incluem a programação original da *Apple* (*cocoa*), mas também alguns compiladores, como o *Jikes* que pode ser instalado direto no *Iphone* ou *iPodTouch*. Desta forma, é possível rodar Java no *iPhone* (Prado 2009). Isso significa que o Java também atende a esta plataforma, reforçando os motivos da escolha do J2ME como linguagem de programação.

A tabela 3 apresenta um comparativo entre as cinco plataformas existentes no mercado de dispositivos telefônicos móveis. Pode-se perceber que a única que fornece possibilidades de portabilidade, é *OpenSource* e possui facilidade de implantação e testes em mais de um modelo de aparelho é o J2ME, executando com sistema operacional *Symbian*. Além disso, esta plataforma representa ainda hoje a maior parte do mercado brasileiro (Santos, Freitas et al. 2009).

**Tabela 3 – Comparativo entre as plataformas de desenvolvimento para fim desta pesquisa.**

| PLATAFORMA | BREW (QUALCOMM) | .NET (WINDOWS CE) | J2ME (SYMBIAM) | ANDROID | IPHONE |
|---|---|---|---|---|---|
| JAVA/J2ME | ✓ | ✓ | ✓ | ✓ | ✓ |
| OPEN SOURCE | | | ✓ | ✓ | |
| FACILIDADE DE IMPLANTAÇÃO E TESTES | | ✓ | ✓ | ✓ | ✓ |
| PORTABILIDADE | | | ✓ | | |
| MAIOR QUANTIDADE DE EQUIPAMENTOS NO MERCADO | | | ✓ | | |

O modelo de negócio do BREW inviabilizou sua escolha no início desta pesquisa, pois exigia certificações para a implantação em dispositivos móveis, demandando um grande intervalo de tempo (atualmente isso não é mais necessário). Já o .NET *Micro Framework* é uma plataforma proprietária aplicada apenas a dispositivos com sistema operacional *Windows CE* 4.1, o que representa a minoria dos telefones móveis hoje no mercado. Por esta razão, será então descartada. Já o J2ME não apresenta nenhum dos dois problemas citados anteriormente, além de apresentar a facilidade de acesso a documentos e informações fornecidas pela *Sun*. Tanto o *Android* como o *iPhone* estão há muito pouco tempo no mercado, e não foram considerados no início deste trabalho como opções de pesquisa por ainda não estarem disponíveis. Desta

forma, deste ponto em diante, este trabalho está definido sobre a plataforma J2ME, configuração CLDC 1.1, perfil MIDP 2.0.

O J2ME apresenta também a possibilidade de visualizações 2D e 3D utilizando as padronizações JSR 226 – M2G e JSR 184 – M3G, respectivamente. Estas padronizações serão descritas no próximo capítulo.

# 3. API's gráficas para dispositivos móveis J2ME

Que essa minha vontade de ir embora se transforme na calma e na paz que eu mereço
Que essa tensão que me corrói por dentro seja um dia recompensada
Porque metade de mim é o que eu penso, mas a outra metade é um vulcão.
Que o medo da solidão se afaste, e que o convívio comigo mesmo se torne ao menos suportável.
O.M.*

Dispositivos portáteis podem ser agrupados em três grandes categorias: os telefones básicos, os telefones de categoria avançada e os *smartphones* (ou telefones inteligentes). A categoria avançada e os *smartphones* são interessantes para aplicações gráficas, e a maioria deles incorpora J2ME. O Java *Mobile* torna possível desenvolver aplicativos através de uma plataforma bastante uniforme de programação. Ele oferece programação suficiente para a maioria das necessidades das *interfaces* multimídia, incluindo gráficos 3D (*OpenGL* ES e M3G) e também abrange uma grande variedade de diferenças de desempenho e funcionalidades dos dispositivos portáteis.

Em todas as três categorias de dispositivos, o suporte a Java apresenta-se, juntamente com o formato de gráficos 3D M3G e *OpenGL* ES. Há ainda a opção de visualização de imagens 2D, e todas as possibilidades serão descritas a seguir, com ênfase ao formato M3G e ao *OpenGL* ES.

## 3.1. M2G - A API *Mobile 2D Graphics*

A variedade de resoluções de tela de dispositivos móveis cria um problema para o conteúdo 2D. Se os gráficos são carregados e distribuídos como bitmaps para diferentes dispositivos, são grandes as chances são de que a resolução do conteúdo da imagem seja diferente da resolução da tela do dispositivo de saída. *Bitmaps* gráficos exigem também quantidades significativas de memória para armazenamento e uma alta largura de banda para transmissão através de uma rede, e este problema só se agrava como a exibição imagens de melhor resolução. SVG (*Scalable Vectorial Graphics – Gráficos Vetoriais Escalonáveis*) pode representar uma solução para estes problemas. Se o conteúdo é representado como formas, tais como curvas e polígonos vez de pixels, às vezes pode ser codificado de forma mais compacta e também pode ser ajustado para ser exibido em diferentes resoluções sem perda de qualidade (Kari Pulli 2008).
A especificação JSR 226 (API Mobile 2D Graphics (M2G)) define um pacote opcional escalonável para renderização de gráficos vetoriais em 2D, incluindo arquivos de imagem W3C ( World Wide Web Consortium - formato SVG - Scalable Vectorial Graphics). A API é direcionada para a plataforma J2ME, com ênfase no MIDP. A API é dirigida à classe dispositivos que normalmente não possuem suporte de hardware para gráficos 2D ou aritmética de ponto flutuante e deve permitir utilização de recursos gráficos 2D nativo do dispositivo, quando aplicável (Process 2006). Esta API inclui a capacidade de carregar e exibir imagens vetoriais 2D armazenadas no formato W3C SVG e também a renderização destas imagens para diferentes resoluções de tela, além de sustentar sua execução.

A especificação JRS 287 define um pacote opcional para o reforço da renderização de gráficos vetoriais em 2D, sendo concebida como uma extensão da JSR 226 (sendo então plenamente compatível)

*Scalable Vector Graphics* (SVG) é uma linguagem XML (*eXtensible Markup Language*) baseada em quadros de gráficos vetoriais (W3C 2003). São imagens calculadas (como uma fórmula matemática) e exibidas. São também escaláveis e armazenadas como documentos XML, tendo duas grandes vantagens para as aplicações móveis: são compactas e escaláveis, suportando animação e *Transform*ações. A Figura 8 apresenta alguns exemplos de imagens SVG no emulador da *Sun*.

A API M2G está claramente centrada na reprodução e manipulação do conteúdo SVG e consiste em classes de alto nível para a criação e renderização vetoriais e classes

de baixo nível para manipular componentes de um vetor imagem como partes de um *Document Object Model* (DOM). Vetores de imagens são instâncias de *ScalableImage* obtidas a partir do método estático *createImage ()* dessa classe.

**Figura 8 – Exemplos de imagens SVG – M2G (Sun 2007).**

Para o caso de renderização animada SVG, a API prevê a classe *SVGAnimator.* Para aplicações MIDP, *SVGAnimator* cria e controla um objeto *Canvas* que manipula na tela atualizações automáticas em resposta a eventos e modificações programáticas para a imagem. Proporciona também um jogador como *interface* para controlar a reprodução da animação (Powers 2005).

As imagens em 2D são extremamente úteis a diversos tipos de aplicativos, como alguns jogos, construção de menus, formulários, determinação de rotas em mapas etc., são maioria nos aplicativos existentes no mercado e essenciais para a categoria de telefones simples.

## 3.2. M3G - A API Mobile 3D Graphics

A API *Mobile* 3D *Graphics* (JSR-184) é a primeira versão padronizado Java ME para desenvolvimento de gráficos 3D. Esta API fornece funcionalidades 3D em um pacote compacto que é apropriado para dispositivos CLDC / MIDP. A API foi projetada para resolver vários casos, incluindo a utilização em jogos, mapas de visualização, etc. Ela inclui suporte para uma única câmara (ou seja, um único ponto de vista na área visualização em 3D), com nevoeiro, luz, malhas e materiais. Uma das suas

características mais importantes é a capacidade de carregar modelos 3D e cenas compostas de ferramentas, como 3ds Max, Blender, Maya, entre outros (Process 2005). A especificação JRS-184 define que estas aplicações devem apoiar o formato de arquivo M3G. A Figura 9 apresenta uma amostra de aplicativos 3D no *Sun Wireless Toolkit*.

**Figura 9 – Aplicações 3D no *Sun Wireless Toolkit* (Sun 2008).**

Esta API é fundamentada no *OpenGL*, concebida com orientação a objetos e prevendo uma curva de aprendizado suave para programadores iniciantes e alta produtividade para programadores experientes em 3D.

Dois métodos são fornecidos para exibir o conteúdo de gráficos 3D. O modo imediato, que torna possível pedido diretos para criar e manipular elementos 3D (M3G pode ser pensado num nível mais alto, como um link para ferramentas para criação de conteúdo digital, tais como 3DsMax ou Maya (Autodesk), Softimage, Blender, etc.), e o modo retido, que permite carregar e exibir cenas 3D (como uma *interface* orientada a objeto *OpenGL* ES) .A utilização de uma combinação dos dois modos é normalmente mais adequada.

Esta API é dirigida a classe de dispositivos CLDC, que normalmente têm muito pouco poder de *Transform*ação e de memória e sem suporte de *hardware* para gráficos 3D ou ponto flutuante matemático, mas é também escalonável até dispositivos que apresentam visores coloridos, um DSP (*Digital Signal Processor*), uma unidade de ponto flutuante, ou mesmo *hardware* especializados para gráficos 3D, apoiando características que não são estritamente necessárias a plataformas muito limitadas.

O conteúdo interativo em dispositivos móveis inclui jogos, navegação, mensagens e *interfaces* customizadas para usuários, e todos estes podem ser tratadas por meio de um API iterativa 3D.

A Figura 10 mostra uma visão geral da arquitetura de *software* Java *Mobile* e o posicionamento de M3G nele. No topo do diagrama, temos as aplicações, chamadas de *MIDlet*s. Na camada inferior encontra-se o *OpenGL* ES (M3G é conceitualmente fundamentado em *OpenGL* ES). Na parte inferior, à direita, temos a Máquina Virtual Java (JVM), com suas bibliotecas. O núcleo bibliotecas está definido nas configurações dos dispositivos (CLDC e CDC).

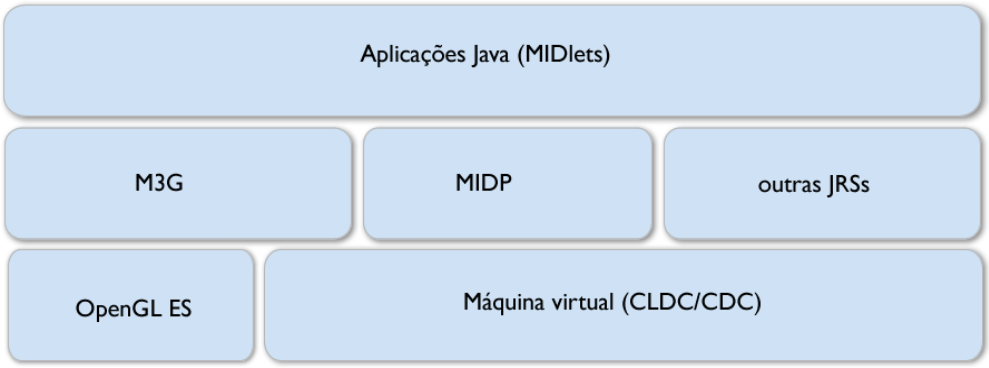



**Figura 10 – Arquitetura Java com M3G (Kari Pulli 2008).**

A Figura 11 mostra o diagrama de classes do M3G. Todas as classes estão definidas no pacote *javax.microedition.M3G*. A classe base da hierarquia é o *Object3D*. Existem apenas 4 classes que não derivam de *Object3D*: *Graphics*3D, responsável pela renderização; *Loader*, responsável pela importação de arte e arquivos de cena locais ou através da rede, *Transform*, que representa uma matriz genérica 4x4 de *Transform*ações e *RayIntersection*, usada para colocar objetos em cena.

Quase todas as características do *OpenGL* ES 1.0 estão disponíveis através M3G, embora algumas apareçam de forma simplificada. Na Figura 11, as classes em azul podem ser consideradas invólucros para conceitos existentes no *OpenGL*. Desta forma, todo o controle fica por conta do motor M3G, podendo ser armazenadas e processadas em código nativo. Para proporcionar aos desenvolvedores uma plataforma menos fragmentada, tudo o que é facultativo ou mal suportado no *OpenGL* ES pelo *hardware* não foi incluído. Para manter o gráfico da cena mais leve e simples, não há explicitamente apoio aos terrenos, sombras, portais, partículas, e outras funcionalidades avançadas de alto nível. Em uma cena pode-se fazer, no máximo, um nó pai, ou seja, não existe qualquer suporte para instâncias no nível de nós. No entanto, todos os dados

importantes, como por exemplo, texturas, vértices, índices, e animações, figuram como componentes do nó (Kari Pulli 2008). A Figura 12 apresenta um resumo dos pacotes de *interface* com os usuários disponíveis na programação J2ME.

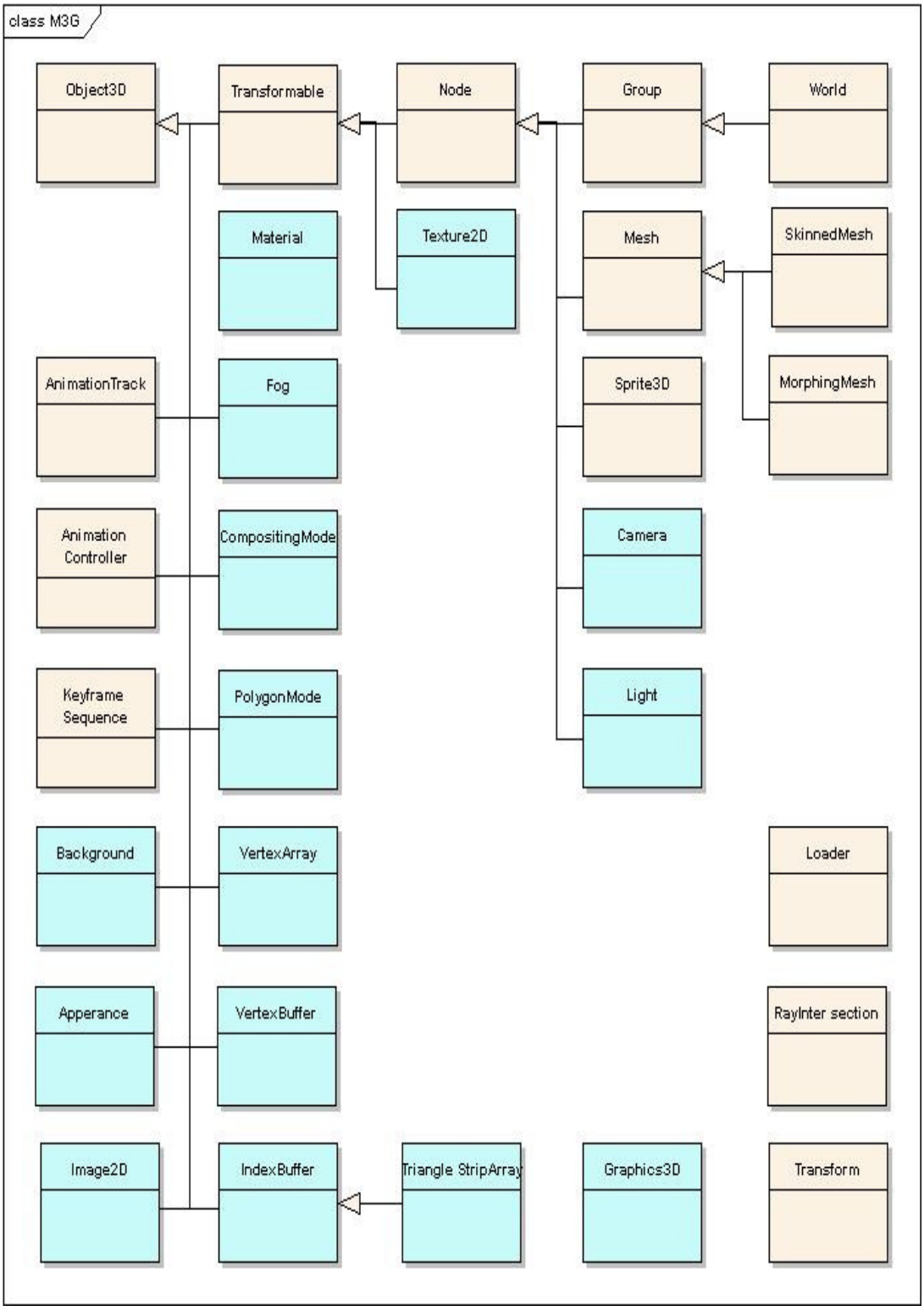



**Figura 11 – Hierarquia da classe M3G (Kari Pulli 2008).**

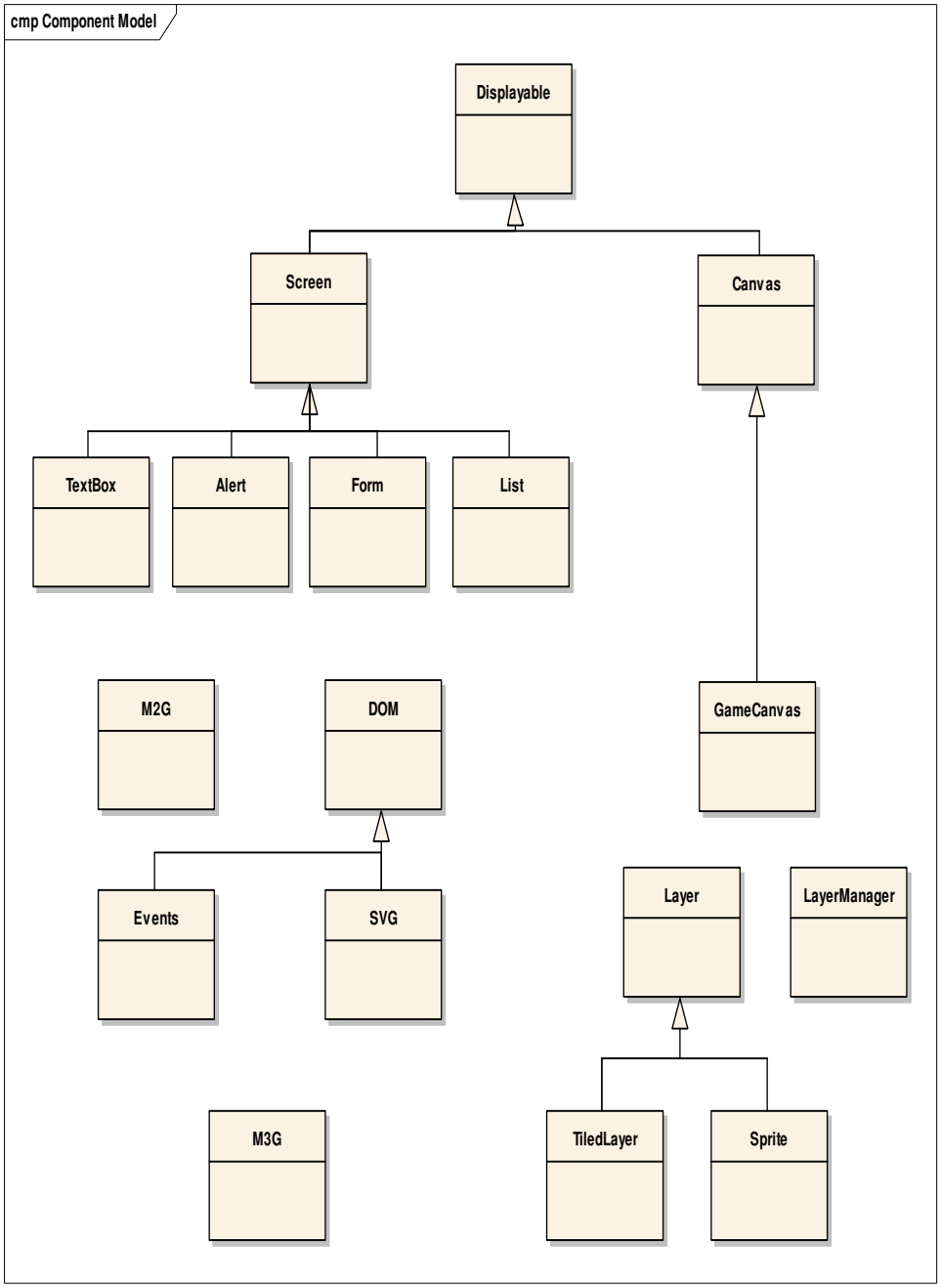



**Figura 12 - Funções de interface com usuário J2ME adaptada de (Kari Pulli 2008).**

## 3.3. OpenGL ES - *Open Graphics Library for Embedded Systems*

O *OpenGL* ES é uma API embutida de gráficos leve (ocupa pequena quantidade de memória RAM (*Random Access Memory*)) e de baixo nível (pertence a *interface* de baixo nível do J2ME), fornecendo a *interface* de programação entre *software* e *hardware*. Isto torna fácil e acessível trabalhar com gráficos 3D e jogos em todas as principais plataformas móveis que o tenha embutido. É baseado no *OpenGL* garantindo um cominho de migração para programadores.

As vantagens do uso desta plataforma são: *software* livre, baixo consumo de energia, requisitos mínimos de armazenamento, uniformidade de migração entre *OpenGL* e *OpenGL* ES, extensibilidade e evolução, documentação, entre outros.

A especificação *OpenGL* ES inclui a definição de vários perfis. Cada perfil é um subconjunto de uma versão da especificação *desktop OpenGL* mais algumas extensões específicas adicionais *OpenGL* ES. Atualmente, a especificação consiste de um total de duas definições de perfis: o perfil comum, e ao perfil de segurança críticas (Group 2007).

- O perfil comum é destinada a dispositivos tais como telefones celulares, PDAs (Personal Digital Assistant), consoles de jogos, etc. Aborda o mais amplo leque de mercado, incluindo o apoio às plataformas com diferentes capacidades.
- O perfil de segurança crítica é destinado ao consumo industrial e aplicações onde confiabilidade e certificabilidade são as principais necessidades.

A Figura 13 apresenta como os perfis se sobrepõem dentro da especificação *OpenGL*. O perfil comum é genérico, possui necessidade de memória mínima e função 3D completa, permitindo diversas aplicações gráficas. O perfil de segurança crítico, em sua área de intercessão com o perfil comum, algumas funcionalidade de jogos são removidas para diminuir ao mínimo absoluto a dimensão e complexidade do código, facilitando assegurando certificações (Group 2007).

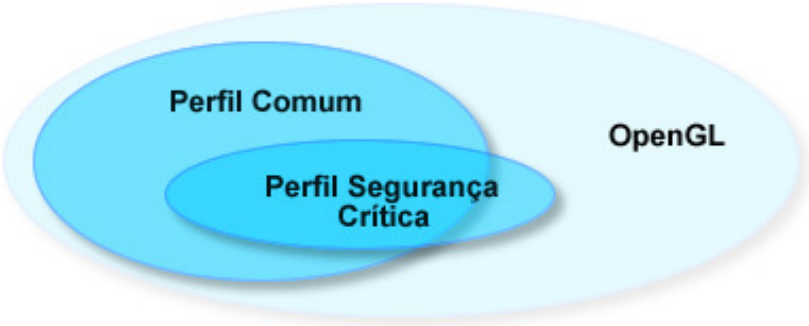



**Figura 13 – Diagrama dos perfis OpenGL ES (Group 2007).**

O *OpenGL* ES é, então, o subconjunto para celular do *OpenGL*, e é padronizado para APIs 3D. O conjunto Java para *OpenGL* ES API (JSR-239) foi concebido para ser um subconjunto do seu homólogo Java SE (JSR-231). A especificação *OpenGL* ES

também está dividida em perfis (como as especificações Java ME e SVG), a saber (Process 2007):

- *Comum Profile* (que fornece suporte para números em ponto flutuante) – CLDC 1.1
- *Comum Lite Profile* (não tem suporte para números em ponto flutuante) – CLDC 1.0

O conjunto Java para *OpenGL* ES implementa a API *OpenGL* ES perfil comum versão 1.0 e 1.1, e define dois novos pacotes:

- *javax.microedition.khronos.egl*
- *javax.microedition.khronos.opengles javax*

Contrariamente à especificação *Mobile* 3D *Graphics* 1.0, o Java para *OpenGL* ES API apoia operações de ponto flutuante, o que torna esta API exclusivamente para dispositivos CLDC 1.1. Além disso, esta API não é compatível com o formato de arquivo M3G definido na especificação *Mobile* 3D *Graphics* 1.0 (JSR 184), que foi definida para fornecer funcionalidade núcleo gráficos 3D. Enquanto JSR 184 (que é orientado a objeto) exige elevado nível de funcionalidade, *OpenGL* é uma biblioteca gráfica baixo nível que é adequada para gráficos 3D (Process 2007).

As ligações Java para *OpenGL* ES (JSR 239), visa fornecer Java *bindings* para as bibliotecas nativas do *OpenGL* ES, ainda está na sua fase inicial. Destina-se a fornecer a mesma funcionalidade JSR 184 – M3G (especificação que define o formato de cenas 3D em J2ME), mas serve para os desenvolvedores que já estão familiarizados com o *OpenGL* (Process 2007).

Os principais requisitos para a API são:

- Ativar o uso de gráficos 3D em uma variedade de aplicações;
- Não presumir a existência de altas capacidades de *hardware*;
- Não impor limites arbitrários sobre a dimensão ou complexidade dos conteúdos 3D;
- Suporte funções sofisticadas também em plataformas simples;
- Manter baixo o uso das memórias ROM (*Read Only Memory)* e RAM.

Para alcançar suficiente generalidade, sem sobrecarregar o desenvolvedor com o desenho individual de pixels, a API possui comandos de desenhos estáticos de objetos 3D e permite a construção e manutenção de um cenário com hierarquia, realizando

oclusão, animando os objetos na cena, e selecionando o ponto de vista, facilitando o desenvolvimento de aplicativos e diminuindo o tamanho da aplicação.

## 3.5. Conclusões

Atualmente, a maior parte dos dispositivos que integram processadores gráficos são telefones inteligentes, mas alguns telefones comuns com esta característica também podem ser encontrados. É razoável esperar que a aceleração gráfica tornar-se-á mais comum neste segmento também. Uma razão para isto é que a utilização de um processador gráfico dedicado é mais eficiente em termos de potência do que fazer os mesmos efeitos sobre uma CPU de propósito geral: a CPU pode exigir um relógio com velocidade até 20 vezes mais elevada do que um *chip* dedicado para alcançar o mesmo desempenho de renderização.

A JSR 226 (M2G - SVG) foi concluída em 2005, e pode ser encontrada em vários modelos de telefones de fabricantes tais como *Nokia* e *Sony Ericsson*. Já a JSR 287 é uma sucessora compatível a JSR 226. As melhorias desta API incluem os novos gráficos e funcionalidades multimídia, como por exemplo, opacidade, gradientes, caixas de texto, áudio e vídeo. Há um modo imediato que é compatível com *OpenVG* e projetado para alto desempenho. Esta API apoia tanto o modo retido de acesso (cena gráficos) como o modo imediato de acesso (o *OpenGL* ES subconjunto ou similar), e permite misturar e combinar os dois modos de uma forma unificada, tendo todos os métodos implementados, incluindo importadores para determinados tipos de dados fundamentais, incluindo malhas, texturas, gráficos e cena; os dados devem ser codificados em um formato binário compacto para armazenamento e transmissão, sendo exequível dentro de 150KB de um terminal móvel, fornecer coleta de lixo e interoperar adequadamente com outras APIs Java, especialmente MIDP (Qusay H. Mahmoud 2004).

M3G, ou JSR 184, pode ser pensado num nível mais alto, como um link para ferramentas para criação de conteúdo digital, tais como 3DsMax ou Maya (Autodesk), Softimage, Blender, etc., e, num nível mais baixo, como uma *interface* orientada a objeto *OpenGL* ES.

Esta API é dirigida a classe de dispositivos CLDC, que normalmente têm muito pouco poder de processamento e de memória e sem suporte de *hardware* para gráficos 3D ou ponto flutuante matemático, mas é também escalonável até dispositivos que

apresentam visores coloridos, um DSP (*Digital Signal Processor*), uma unidade de ponto flutuante, ou mesmo *hardware* especializados para gráficos 3D, apoiando características que não são estritamente necessárias a plataformas muito limitadas.

Sobre esta API serão pesquisados os *framework*s para visualização de imagens M3G citada no capítulo seguinte.

# 4. *Frameworks* de interfaces gráficas para dispositivos móveis

Que o espelho reflita em meu rosto um doce sorriso
Que eu me lembro ter dado na infância
Por que metade de mim é a lembrança do que fui
A outra metade eu não sei.
O.M.*

Nos últimos anos têm-se verificado melhorias em computação e comunicação em dispositivos portáteis. Apesar destas, os terminais móveis ainda são menos capazes do que computadores *desktop*. Eles possuem velocidade mais baixa, telas menores em tamanho e em resolução, menos memória para rodar e armazenar programas, e períodos curtos de autonomia de bateria. Para visualização de gráficos 3D, acrescidos aos problemas associados à tela do dispositivo, tem-se a necessidade de alguma potência computacional para alcançar um desempenho utilizável, tornando esta tarefa bastante complicada. Porém, com a introdução da cor nos *displays* e de processadores mais poderosos e até dedicados exclusivamente a esta tarefa, estes aparelhos tornaram-se capazes de renderizar gráficos 3D interativos.

Este capítulo apresenta trabalhos de visualização e interação com cenas 3D em dispositivos móveis, apresentando os principais *framework*s de *interfaces* gráficas para dispositivos portáteis.

## 4.1 *Frameworks* de Interfaces Gráficas de Usuário em J2ME

**a) APIME**

*APIME* é um *framework* que oferece diversas funcionalidades para programação em Java para dispositivos móveis (J2ME). O núcleo deste *framework* é a *interface* com o usuário, com componentes básicos para tornar os aplicativos mais atraentes, e com possibilidade de adaptar-se ao que cada desenvolvedor necessita (Araiz 2008).

É compatível com MIDP 1.0, embora exista uma versão para MIDP 2.0 e outro para a *Nokia*, para usar o recurso de tela cheia do MIDP 1.0 não oferece. A interação é com um ponteiro do mouse movido por teclado ou caneta. Também inclui classes para gerenciar arquivos e personalização (peles, internacionalização, teclados para diferentes línguas e celulares, etc.) (Araiz 2008).

Não é necessário modificar o código desenvolvido para usá-lo nas diferentes versões, e também baseia-se na estrutura do *Swing* (J2SE), o que promove uma rápida adaptação para os programadores (Araiz 2008).

Araiz desenvolveu os Skins, que são a base para personalizar o aspecto visual da aplicação. O teclado é configurável (para ser usada em celulares com diferentes códigos e linguagens) e a iteração é com um ponteiro do mouse. Seu *framework* permite a utilização e criação de fontes diferentes (nativo, imagem, ...). Possui também labels, campos de texto e áreas de textos, além de painéis de rolagem, Listas e *ComboBoxes* que aceitam imagens, barras de progresso e também permite descrições. O *APIME* é distribuído sob a licença GNU General Public License (GPL). Na Figura 14, pode-se visualizar algumas dos diversos objetos construídos com o *framework APIME*.


**Figura 14 – Diversos objetos construídos utilizando o APIME (Araiz 2008).**

A Figura 15 mostra algumas possibilidades de *interfaces* desenvolvidas com o *APIME*. Há recursos de cores, fontes e tabelas, porém nenhuma possibilidade de inserção de cenas 3D.

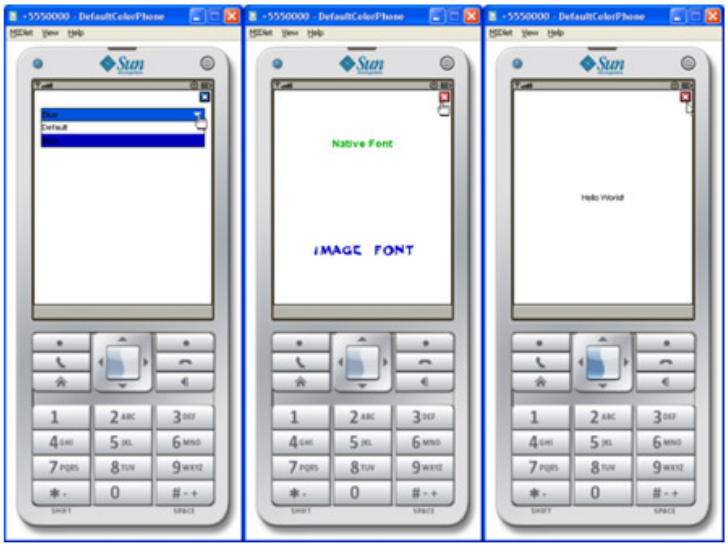



**Figura 15 - Interfaces desenvolvidas utilizando APIME (Araiz 2008)**

### *b)* J2ME *Polish*

J2ME Polish permite personalizar aplicativos J2ME sem alteração do o código fonte. O projeto, as animações e efeitos são especificados em arquivos *Cascading Style Sheets* – CSS[5] externos, muito semelhante ao padrão da web. Esse processo não apenas separa a concepção do desenvolvimento da lógica de negócios, mas também lhe permite criar diferentes personalizações muito facilmente. Pode-se criar diversas opções de *design*, utilizando fontes não-padrão, os efeitos destas fontes, animações como as transições de tela e opções de menu (Virkus 2005).

Os principais benefícios da utilização deste *framework* são (Virkus 2005):

- Processo de concepção semelhante ao projeto de páginas da web;
- *Design*ers e desenvolvedores podem trabalhar de forma independente;
- Possibilidade de personalizar a aplicação para clientes diferentes ou mesmo para usuários diferentes;

[5] *Cascading Style Sheets* – CSS - é uma linguagem de estilo utilizada para definir a apresentação de documentos escritos em uma linguagem de marcação, como HTML ou XML. Seu principal benefício é prover a separação entre o formato e o conteúdo de um documento.

- Uso de características de montagem para adaptar o projeto para diferentes resoluções de tela;
- É compatível com o padrão MIDP.

J2ME Polish permite desenvolver, projetar e construir aplicações em qualquer plataforma: Windows, Mac OS X, Linux ou qualquer outro sistema operacional onde o Java esteja habilitado. É baseado em Ant - o padrão Java para criação de aplicativos e pode-se usá-lo com todo o IDE (*Integrated Development Environment)*. Foram criados *plugins* que aceleram o desenvolvimento. J2ME *Polish* suporta o padrão global ME/J2ME Java e funciona em todos os telefones celulares habilitados para MIDP. Também funciona em *BlackBerry, Palm, DoJa* e dispositivos WIPI (*Wireless Internet Platform for Interoperability)* com *J2ME Polish*. *Windows Mobile, iPhone* e plataformas *Android* são suportados nativamente (Virkus 2005). A Figura 16 representa a portabilidade deste *framework*.

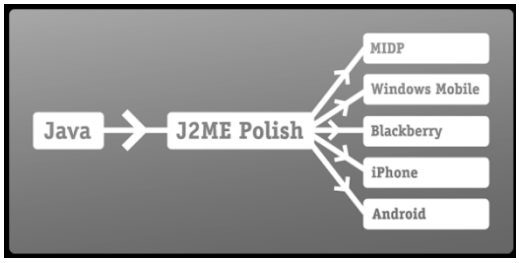



**Figura 16 - Portabilidade J2ME Polish (EnoughSoftware 2007).**

Armazenar dados em um telefone pode ser complicado, especialmente quando tem-se uma estrutura complexa, como imagens. O uso deste *framework* permite que, com apenas uma única chamada os dados sejam salvos ou lidos. Não há problema para a gestão persistência de dados (EnoughSoftware 2007). A Figura 17 representa a persistência de dados utilizando RMS (*Record Management System*) neste *framework*.

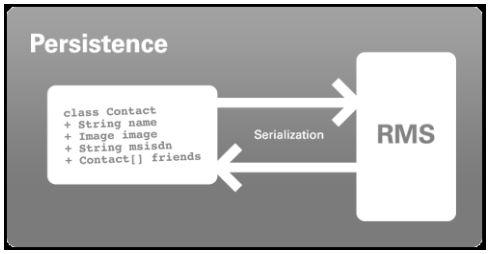



**Figura 17 - Persistência utilizando RMS no J2ME Polish (EnoughSoftware 2007).**

Para superar a barreira de fragmentação – inúmeras configurações de dispositivos – o J2ME *Polish* proporciona um banco de dados Open Source no dispositivo que contém várias informações, como a resolução da tela, *heapsize*, APIs

disponíveis entre outras informações, que são usadas para adaptar o aplicativo para plataformas seu destino.

J2ME *Polish* combina o poder do mundo Java móvel com a flexibilidade de *webdesigns* separando as configurações do projeto de código-fonte do aplicativo. Ao utilizar o web-padrão *Cascading Style Sheets* (CSS), personalização e *design* de aplicações torna-se muito fácil e flexível. Com o J2ME *Polish* qualquer *web-designer* pode projetar aplicações móveis, enquanto os programadores podem concentrar-se na lógica do negócio (EnoughSoftware 2007). A Figura 18 apresenta *interfaces* desenvolvidas utilizando este *framework*.

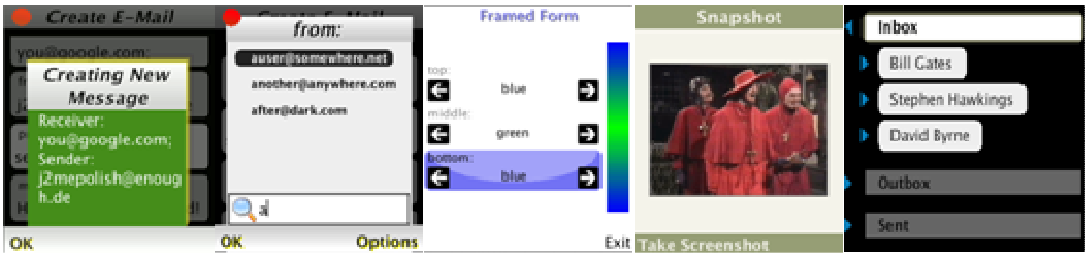



**Figura 18 - Objetos criados utilizando o J2ME Polish (Software 2007).**

Todas as transições implementadas no J2ME *Polish* são realizadas sobre figuras 2D, como por exemplo *cage Animation, diagonal Animation, domino Animation, fade Animation,* etc.

### *c) Micro Windows Toolkit*

*Micro Windows Toolkit* (MWT) fornece um *framework* de *interface* com o usuário projetado e otimizado para dispositivos pequenos. Ele foi inspirado no AWT (*Abstract Window Toolkit*), no *Swing* e no SWT (*Standard Widget Toolkit)*. Requer apenas as API's MIDP 1 e CLDC 1, por isso é totalmente portátil.

Contém uma API de alto nível, projetada para portabilidade de aplicações, que emprega um elevado nível de abstração, fornecendo muito pouco controle sobre aparência e comportamento e uma API de baixo nível, que fornece um bom controle sobre gráficos e eventos de entrada, mas há falta de componentes de *interface* do usuário.

Existe um pacote opcional que estende este *framework*, englobando as características MIDP 2 para este tipo de dispositivos, mas não dispensa o pacote básico (mwt.jar). Com esta extensão, consegue-se, por exemplo, alterar a cor e o tamanho de

imagens durante a execução, criar fontes dinamicamente, entre outros (Sourceforge.net 2007) A Figura 19 apresenta objetos criados com o pacote básico do MWT, onde pode-se ver a possibilidade de criação de menus e caixas de texto.

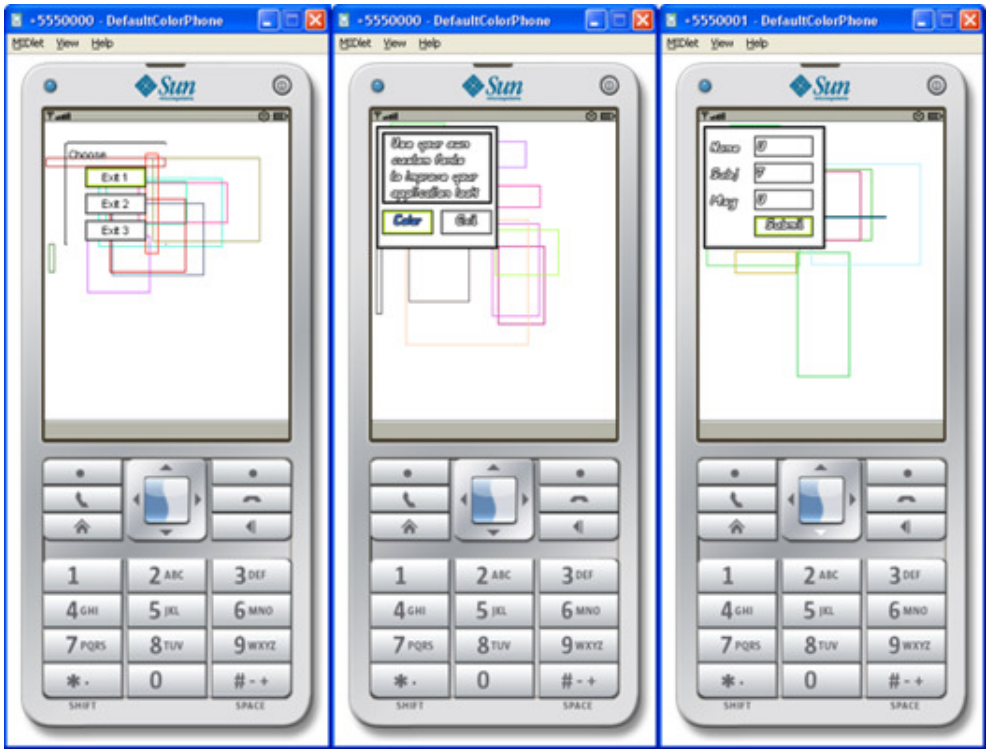



**Figura 19 - Objetos criados utilizando MWT (Sourceforge.net 2007).**

***d) VirTraM***

O VirTraM tem como objetivo principal prover uma arquitetura para a construção de aplicações voltadas ao treinamento de indivíduos utilizando dispositivos móveis com recursos de RV. As aplicações desenvolvidas a partir do VirTraM atendem a determinados requisitos em comum, como mobilidade (o usuário está em movimento e a aplicação deve poder ser usada em qualquer local e a qualquer momento, sempre que for preciso, sem haver dependência dos recursos de comunicação), interatividade (usuário escolher um caminho dentro do mundo virtual, podendo navegar livremente e manipular os objetos tridimensionais), portabilidade (podendo ser executada em diferentes plataformas, característica presente tanto na Computação Móvel como na RV, sendo compatível tanto com telefones celulares quanto com PDAs, suportando diferentes sistemas operacionais) e facilidade de uso (deve ser simples e fácil de usar). Utilizou-se para a implementação do VirTraM, a plataforma Java para dispositivos móveis (J2ME - Java 2 *Micro Edition*), principalmente pela sua grande portabilidade (suportado pela maioria dos fabricantes de dispositivos móveis). As classes do VirTraM

colaboram entre si e devem ser reutilizadas como um todo para a construção da aplicação. O VirTraM encapsula os detalhes de implementação do J2ME e permite ao programador concentrar-se nos aspectos da aplicação de treinamento (Filho 2005).

A Figura 20 apresenta a arquitetura do VirTraM, onde o Pacote 3D contém as classes essenciais para a construção dos ambientes tridimensionais (funções 3D, tais como: renderização, rotação, translação, mudança do ângulo de visão virtual e detecção de colisão) das aplicações. Um outro recurso importante desse pacote é o apontador, utilizado no ambiente para selecionar os objetos 3D.

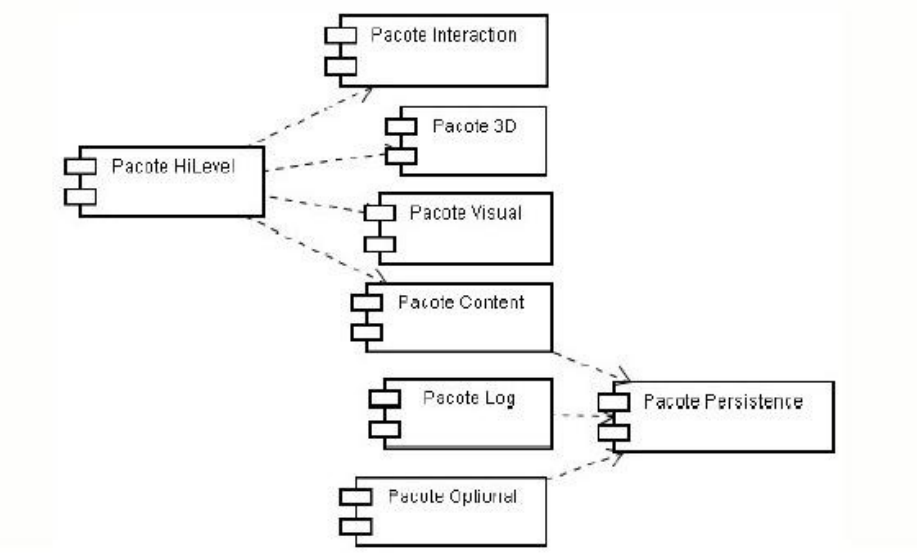



**Figura 20 - Arquitetura do VirTraM (Filho 2005).**

O Pacote de Interação é responsável por capturar as diferentes interações do usuário, disparar as funcionalidades solicitadas e retornar os resultados para a aplicação. As formas de interação previstas são o teclado ou a tela sensível ao toque do dispositivo móvel. Porém, o *framework* pode ser estendido para suportar outras formas de interação, como localização via GPS ou *Wi-Fi*[6]. O Pacote de Persistência fornece as classes responsáveis pelo acesso aos dados da aplicação. Entre as principais funcionalidades desse pacote pode-se destacar consulta, armazenamento e alteração das informações do treinamento. O Pacote Nativo fornece as classes responsáveis pelo acesso aos recursos específicos de determinados dispositivos, tais como: recursos de som e função para vibrar o dispositivo móvel. O Pacote opcional contém classes relacionadas a recursos complementares e não essenciais aos treinamentos. Estas classes tratam de características mais específicas dos dispositivos, podendo, assim, não estar

[6] Wi-Fi – TradeMark (marca registrada).

disponíveis em por determinados equipamentos. O Pacote *Log* tem como principal função fornecer ao desenvolvedor os recursos que possibilitem uma avaliação do desempenho do usuário no treinamento, através do registro de suas ações.

O Pacote Visual contém classes responsáveis por criar a *interface* com o usuário, exceto o ambiente 3D. Estas incluem as telas padrões para apresentação do conteúdo dos treinamentos baseados no VirTraM. O Pacote *Content* contém as classes responsáveis por obter, armazenar e disponibilizar o conteúdo do treinamento. Ele define a forma como o material descritivo do treinamento é associado à aplicação, mostrando como o desenvolvedor tem acesso ao conteúdo para poder apresentá-lo. O Pacote *HiLevel* é o principal pacote do VirTraM, responsável por integrar os pacotes anteriores e executar o treinamento. Ao integrar os recursos dos pacotes do VirTraM, o Pacote *HiLevel* fornece ao desenvolvedor as funcionalidades necessárias para a construção do treinamento, utilizando poucas linhas de código (Filho 2005). A Figura 21 apresenta o estudo de caso desenvolvido com o VirTraM, um museu virtual.


**Figura 21 - Imagens do Museu Virtual desenvolvido com o VirTraM (Filho 2005).**

Este trabalho, não apresenta testes de usabilidade e nem informa usuários (professores ou treinadores) do sistema que desenvolveram treinamento utilizando-o. A utilização dos pacotes apresentados necessita de programação em linguagem J2ME, o que é, por si só, um empecilho a usuários de outras áreas que não seja a computação. Existe então a clara necessidade de desenvolver um nível mais alto de *interface* entre o treinador e o sistema, para melhorar a usabilidade e aplicabilidade do mesmo.

**e) M3GE**

De acordo com Pamplona (2005), M3GE é um motor de jogos escrito em Java e baseado na API *Mobile* 3D *Graphics* (M3G). Dentre outras implementações, esta biblioteca suporta detecção de colisão, Inteligência Artificial (IA), controla a entrada e saída de informações, etc. Além disso, segue uma especificação definida pela *Java Community Process* (JCP), a *Java Specification Request* (JSR) 184. A M3GE foi projetada para ser utilizada em conjunto com a M3G. Ou seja, as duas bibliotecas interagem entre si. Isso proporciona ao desenvolvedor do jogo flexibilidade e velocidade quando for preciso. A Figura 22 ilustra a arquitetura do motor M3GE.


**Figura 22 - Arquitetura M3GE (Pamplona 2005).**

Ainda segundo Pamplona (2005), o projeto M3GE está dividido em dois grandes componentes: o responsável pela leitura dos arquivos (modelos 3D) no formato *Wavefront* e o motor de jogos em si (*core*). É sobre o *core* que as implementações dos desenvolvedores devem ficar, ou seja, o enredo e a lógica do jogo são implementados sobre ele.

O objetivo deste *framework* é o desenvolvimento de aplicações .NET que contemplem cenas 3D e navegação em primeira pessoa, com detecção de colisão. A Figura 23 apresenta uma cena 3D utilizando o M3GE.

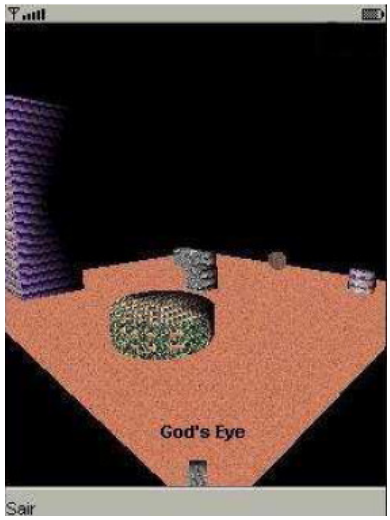



**Figura 23 - Imagem construída usando o M3GE(Pamplona 2005).**

### f) *Lightweight UI Toolkit* (LWUIT)

*Lightweight UI Toolkit* (LWUIT) é um *framework* que permite criar *interfaces* gráficas do usuário (GUI) bastante atraentes, para dispositivos móveis ou quaisquer outros dispositivos que suportem o perfil MIDP, do Java ME. Baseado no *Swing* do Java SE, possui compatibilidade com MIDP 2.0, CDC e Java SE (Lais Zanfoli 2009).

Com o LWUIT, diminui-se muito a necessidade de se desenhar telas em *Canvas* para se obter *interfaces* amigáveis. Ele não possui compatibilidade com a classe canvas (J2ME), mas oferece componentes visuais ricos, que podem ser tão atraentes quanto as *interfaces* desenhadas em LCDUI ou *Canvas* (Lais Zanfoli 2009).

O *framework* oferece melhorias sobre os componentes já existentes no Java ME, como *List, Form, Alert*, entre outros, oferecendo ainda suporte a *Touch Screen*, diversas fontes, animações, botões, transições de telas animadas, temas que podem ser incluídos pelos próprios usuários, *layout*s, utilização de abas (como no Java SE), integração 3D, caixas de diálogo, entre outros (Lais Zanfoli 2009).

A Figura 24 apresenta um comparativo entre *interfaces* construídas utilizando o J2ME padrão e o J2ME em conjunto com o *framework* LWUIT.

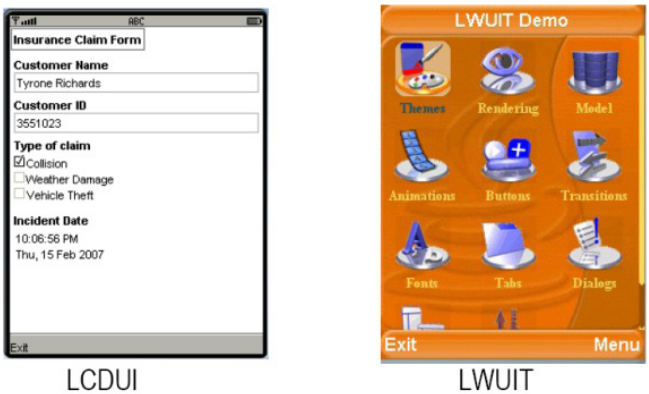



**Figura 24 - Interfaces construídas usando J2ME padrão e utilizando o LWUIT (Lais Zanfoli 2009).**

Para oferecer portabilidade, o LWUIT implementa sua própria camada no topo do sistema nativo *Canvas* e providencia uma abstração quanto aos diferentes dispositivos. Esta abstração é obtida através de classes chaves que escondem classes específicas, como *Graphics*, *Image* e *Font*.

A grande popularização deste *framework* entre os desenvolvedores Java ME fez com que a *Sun* transformasse este projeto numa espécie de padrão dentro da comunidade de desenvolvedores. A *Sun* incorporou o LWUIT como uma de suas

bibliotecas padrão, disponibilizando uma aplicação exemplo e integrando uma de suas ferramentas utilitárias, o *Resource Manager* (Sun 2008), dentro do Java ME *Plataform SDK 3.*

O LWUIT é um *framework* de componentes gráficos inspirado no *Swing* da plataforma *Java Standard Edition* (Java SE), especificamente modelado e construído para ambientes restritos em poder de processamento e memória, como o dos dispositivos móveis. A A Figura 25 representa a hierarquia de classes simplificada LWUIT (Microsystems 2009).

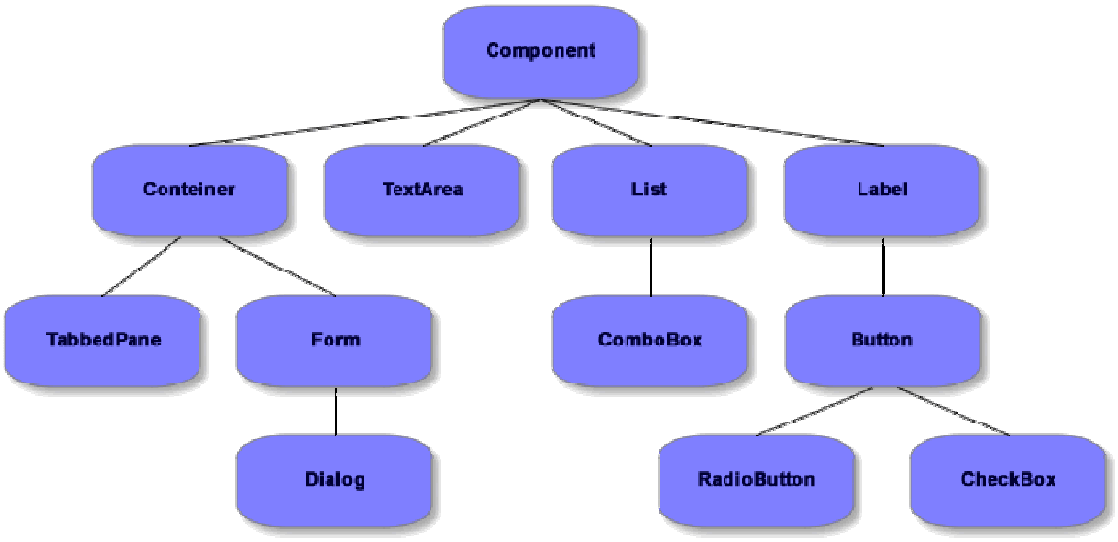



**Figura 25 - Hierarquia de classes simplificada LWUIT (Microsystems 2009).**

A biblioteca tem apenas 256 KB, e oferece aos desenvolvedores a partir de um custo baixo de espaço de armazenamento, uma alternativa aos componentes LCDUI oferecidos pela plataforma básica do Java ME. Esta biblioteca é *open-source* e todo código fonte está disponível com licença GPL v2 (Caraciolo 2009).

O *framework* LWUIT oferece a classe com.*Sun*.lwuit.M3G, que fornece suporte para JSR 184. Esta classe oferece uma *interface* interna (M3G.*Callback*) que deve ser implementada a fim de apresentar a cena 3D. Um método de pintura LWUIT M3G.*bind* (*Graphics*) deve ser invocado a fim de vincular a M3G, resultando em uma chamada de retorno contendo o objeto 3D adequado.

A integração do formato M3G à biblioteca LWUIT é construída em torno de um mecanismo de retorno que permite ao desenvolvedor para vincular um gráfico LWUIT para um objeto *Graphics*3D M3G. O apoio ao M3G é concebido para funcionar apenas em dispositivos que o oferecem. Se o dispositivo M3G não suporta a execução LWUIT evita a ativação código M3G. As regras para lidar com as limitações de dispositivos são

muito similares aos das regras de negociação com erros de dispositivo. Se é invocada no código uma API inexistente, gera-se uma exceção. Pode-se pegar essa exceção e ativar um sinalizador de desativação de funcionalidades relacionadas ao recurso. Um exemplo desta abordagem existe na classe de M3G LWUIT que é projetada para rodar em dispositivos que não suportam JSR 184 (Microsystems 2009).

Neste trabalho, estão presentes os recursos de transição 3D, como o apresentado na Figura 26 (efeito cubo). Qualquer outra visualização 3D precisa ser programada utilizando as classes do próprio *framework* LWUIT.

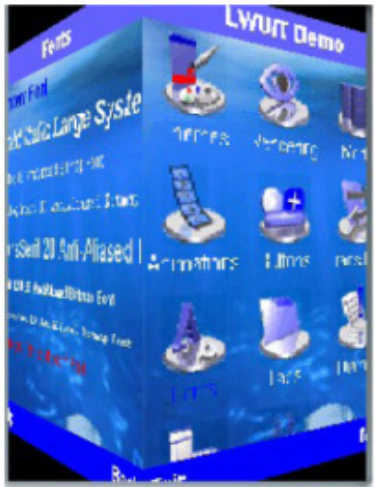



**Figura 26 – Efeito de transição 3D utilizando LWUIT (Microsystems 2009).**

As limitações relacionados com a aparência, com o número de cores, com a velocidade do dispositivo, com a resolução, entre outros, podem ser trabalhados em torno de uma multiplicidade de temas, e usa-se os métodos de exibição para verificar as capacidades gerais do dispositivo, em seguida, habilitar ou desabilitar algumas delas.

A solução adotada neste *framework* é permitir a seus usuários a configurabilidade, tanto quanto possível (para temas, animações, transições, etc), dando-lhes a opção de adequar a aplicação às necessidades de seus dispositivos. A Figura 27 mostra a variedade de *interfaces* que são conseguidas utilizando o LWUIT.

São notórias as características de portabilidade do LWUIT, sendo que o mesmo foi testado nos dispositivos no mercado atualmente. Os únicos pré-requisitos são MIDP 2.0 e CLDC 1.1. Por estas razões, este *framework* será o escolhido como objeto de estudo deste trabalho (Sampaio 2010).

**Figura 27 - Diferentes temas utilizando LWUIT (Microsystems 2009).**

## 4.2. Conclusões

Os terminais móveis ainda são menos capazes do que computadores *desktop*. Eles possuem velocidade mais baixa, telas menores em tamanho e em resolução, há menos memória para rodar e armazenar programas, e períodos curtos de autonomia de bateria. Mas as melhorias em computação e comunicação destes dispositivos vêm motivando o desenvolvimento de aplicativos para dispositivos portáteis que contemplem apresentação de imagens 3D. Entretanto, visualizar gráficos 3D em dispositivos portáteis ainda é uma tarefa muito complicada devido à grande potência computacional necessária para alcançar um desempenho utilizável.

Dentre os esforços para visualização de imagens 3D, o VirTraM tem como objetivo principal provêr uma arquitetura (*framework*) para a construção de aplicações voltadas ao treinamento de indivíduos utilizando dispositivos móveis com recursos de RV. As aplicações desenvolvidas a partir do VirTraM atendem a determinados requisitos em comum, como mobilidade, interatividade, portabilidade e facilidade de uso. Após diversas tentativas com o autor, o orientador e os avaliadores do VirTraM, não se conseguiu o código fonte do mesmo, o que inviabiliza a continuidade de sua pesquisa. Apesar de ser o *framework* que mais se ajusta ao objeto deste trabalho. Por esta razão, ele foi descartado.

Já o M3GE é um motor de jogos (biblioteca) escrito em Java e baseado na API *Mobile* 3D *Graphics* (M3G), que suporta detecção de colisão, Inteligência Artificial (IA), controla a entrada e saída de informações e segue a especificação da Java Specification Request (JSR) 184. A M3GE foi projetada para ser utilizada em conjunto

com a M3G interagindo entre si e proporcionando ao desenvolvedor do jogo flexibilidade e velocidade quando for preciso.

O *Micro Windows Toolkit* – MWT é um *framework* de UI projetado e otimizado para dispositivos pequenos inspirado no AWT, no *Swing* e no SWT. Requer apenas as APIs MIDP1 e CLDC1, e contém uma API de alto nível, projetada para portabilidade de aplicações, que emprega um elevado nível de abstração, fornecendo muito pouco controle sobre aparência e comportamento.

*APIME* é também um *framework* que oferece mais algumas funcionalidades para programação em J2ME. Seu núcleo é a *interface* do usuário, com componentes básicos para tornar os aplicativos mais atraentes, e com possibilidade de adaptar-se ao que cada desenvolvedor necessita. Embora interessante, também não há nenhuma referência a utilização de cenas M3G.

Já o J2ME *Polish* combina o poder do mundo Java móvel com a flexibilidade de *webdesigns* separando as configurações do projeto de código-fonte do aplicativo. Ao utilizar o *web*-padrão *Cascading Style Sheets* (CSS), a personalização e *design* de aplicações torna-se muito fácil e flexível. Porém nenhuma destas três propostas há referências a utilização de cenas 3D.

O LWUIT é um *framework* de componentes gráficos inspirado no *Swing* da plataforma Java SE, oferece a classe com.*Sun*.*lwuit*.M3G, que fornece suporte para JSR 184. Esta classe oferece uma *interface* interna (M3G.*Callback*) que deve ser implementada a fim de apresentar a cena 3D. A integração do formato M3G à biblioteca LWUIT é construída em torno de um mecanismo de retorno que permite ao desenvolvedor vincular um gráfico LWUIT para um objeto *Graphics*3D M3G. O apoio ao M3G é concebido para funcionar apenas em dispositivos que o oferecem.

O aproveitamento deste mecanismo de retorno e dos métodos da classe *com.Sun.lwuit*.M3G para a construção da adaptação para visualização de cenas M3G em alto nível, além da criação dos mecanismos de interação com estas cenas será objeto do próximo capítulo. A tabela 4 apresenta as características dos frameworks estudados neste capítulo, justificando a escolha do LWUIT.

A última característica citada na tabela 4, refere-se a possibilidade de utilização da extensão desenvolvida neste trabalho, uma vez que o framework, sendo o mais utilizado, aponta também para a maior utilização da extensão proposta.

**Tabela 4 - Comparativo entre os frameworks estudados.**

| | VIRTRAM | M3GE | MWT | APIME | J2ME POLISH | LWUIT |
|---|---|---|---|---|---|---|
| **Disponibilidade de código fonte** | | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Licença pública** | | ✓ | | ✓ | | ✓ |
| **Suporte a cenas 3D já previsto** | ✓ | ✓ | | | | ✓ |
| **Aceito pela comunidade de programadores** | | | | | | ✓ |

# 5. RV_LWUIT: uma proposta de modificação do framework LWUIT para manipulação de cenas M3G – JRS184 em dispositivos móveis.

Que não seja preciso mais do que uma simples alegria pra me fazer aquietar o espírito
E que o teu silêncio me fale cada vez mais... Porque metade de mim é abrigo...
Mas a outra metade é cansaço.
O.M.*

O *framework* LWUIT é inspirado no *Swing*. Um outro benefício de sua implantação é o pouco ou nenhum código específico para a diversa gama de dispositivos diferentes que ele suporta. Para garantir a portabilidade, o LWUIT foi construído com elementos comuns ao MIDP 2.0.

Aplicações LWUIT são executadas de forma consistente entre os diferentes dispositivos. Ele é personalizável e extensível e, se há uma característica ou componente em falta em um certo dispositivo, pode-se criá-los e ligá-lo no código. Este *framework* foi testado massivamente em dispositivos encontrados hoje no mercado (em sua forma original). Seus únicos requisitos são MIDP 2.0 e CLDC 1.1(Microsystems 2009).

Dentre as características inerentes ao LWUIT, chamam a atenção a flexibilidade em relação ao tamanho da tela dos dispositivos, o kit de ferramentas que permitem a criação de estilos (CSS), podendo estes serem alterados em tempo de execução, as fontes *bitmap* e presença de uma ferramenta que permite criar fontes no seu *desktop*, suporte para os eventos de toque (nenhuma codificação especial é necessária para uma

aplicação LWUIT para executar em um dispositivo habilitado para tocar), a integração com os gráficos 3D e SVG, e a possibilidade de animar a mudança de telas do dispositivo (efeitos de transição).

Uma atenção especial à característica de integração com gráficos 3D e a proposta de alteração e acréscimo a este *framework* será objetivo deste capítulo.

## 5.1. Análise da utilização de cenas no formato M3G em *Midlets*

### a) O formato M3G

A API *Mobile* 3D (M3G), definida na especificação Java JSR 184, nada mais é do que uma especificação que define uma API gráfica para *Mobile* 3D. Esta API provê funcionalidade 3D em um pacote compacto, mais apropriado para dispositivos CLDC/MIDP, ou seja, dispositivos que suportam a tecnologia Java ME, habilitando uma nova geração de aplicações 3D. A M3G já tem suporte disponível nos maiores fabricantes de dispositivo celular e operadores. Com ela é possível criar cenas 3D.

A API M3G pode ser dividida em dois modos: o "*immediate mode*" e o "*retained mode*" (Kari Pulli 2008).

A API "*immediate mode*" torna possível criar aplicações que manipulem diretamente a criação e uso de objetos 3D. O "*immediate mode*" é mais apropriado para aplicações que gerem gráficos 3D através de algoritmos, como visualizações científicas e gráficos estatísticos. É o modo compatível com o *OpenGL* ES, que é um subconjunto da conhecida biblioteca *Open GL* ES, e que influenciou a definição da API M3G de forma que fosse possível implementar eficientemente a M3G no topo do *OpenGL* ES.

Na modalidade "*Retained Mode"*, os comandos dos gráficos são emitidos diretamente e o motor executa-os imediatamente. Ao usar este método, o programador deve escrever o código que diz respeito especificamente ao motor que executar para cada um dos *frames* da animação. Uma câmera, e o jogo das luzes são associadas também com a cena, mas não são necessariamente parte dela. Na modalidade imediata é possível indicar objetos únicos, ou as cenas inteiras (ou mundos, com uma câmera, luzes, e fundo como partes da cena) (Kari Pulli 2008).

Na maioria das aplicações, o uso do "*Retained Mode*" é visto de modo mais frequente. Neste modo, um *designer* gráfico ou um artista que use uma ferramenta de modelagem 3D, poderá criar cenas e personagens para serem utilizados pela M3G. Estes arquivos devem ser salvos no formato especificado pela JSR 184. Em tempo de

execução, a API gráfica é utilizada para carregar e mostrar o conteúdo 3D existente no arquivo. As aplicações podem manipular partes das cenas carregadas para animar objetos e criar vários efeitos. A modalidade retida usa sempre o gráfico da cena, ligando todos os objetos geométricos no mundo 3D em uma estrutura de árvore, e especifica também a câmera, as luzes, e o fundo. A informação *Higher-level* sobre cada objeto - tal como sua estrutura, posição, e aparência geométrica - é retida a cada frame (Kari Pulli 2008).

De forma geral, pode-se imaginar o "*immediate mode*" como o nível mais baixo de acesso às funções 3D, e o "*Retained Mode*" como a forma mais abstrata e confortável para exibir os gráficos 3D.

M3G não deve ser confundido com o Java 3D, que foi projetado para PCs, o que implica em mais memória e poder de processamento do que em dispositivos móveis. M3G e Java 3D são duas APIs separadas e incompatíveis projetadas para finalidades diferentes (Kari Pulli 2008).

O padrão de M3G especifica também um formato para os dados do modelo 3D, incluindo dados da animação. Isto permite que os programadores criem o índice nos PCs que podem ser carregados por M3G em dispositivos móveis.

#### b) Restrições na construção de cenas

Na utilização do "*Retained Mode*", é necessária a construção de uma cena em uma ferramenta de modelagem e a posterior exportação para o formato M3G, atentando para os seguintes fatos (MobileFish 2008):

i. M3G impõe as seguintes restrições na atribuição de texturas para o modelo:

- As dimensões devem ser potências de dois (4, 8, 16, 32, 64, 128 ...).
- As dimensões de textura são limitados a um máximo de 256x256.
- Recomenda-se texturas pequenas, se possível, para reduzir o tamanho do arquivo M3G.
- Usar imagens paletizadas (256 cores).

ii. A ferramenta exportadora M3G tem algumas limitações:

- precisão *Integer*: Isso limita a precisão de malhas pequenas de apenas algumas unidades de tamanho. Para melhores resultados, usa-se

malhas relativamente grande. Escalar objetos também podem apresentar problemas de precisão.

- interpolação linear o: caminhos suavemente curvo definido por apenas algumas teclas não são suportados.
- Não há suporte para as sombras.
- Não se permite o uso de texturas processuais ou materiais.
- Adicionar um fundo para a cena. Para evitar que o fundo branco, é necessário definir o meio ambiente na cena, ou, alternativamente, uma superfície ou um camarote como pano de fundo.
- Adicionar sempre uma câmera na cena.
- Adicionar sempre uma fonte de luz na cena.
- Ao criar o modelo os vértices das coordenadas deverão estar no intervalo [-100, 100] e não [-1,1].
- O resultado é melhor ao exportar cenas simples ou modelos únicos.
- Não se deve compactar o arquivo M3G.
- Sempre manter a escala de unidade do *software* de modelagem em unidades de genéricos.
- A escala dos modelos será a mesma do *software* utilizado para modelagem.

Obedecidas estas restrições, a visualização da cena M3G pode ser feita diretamente num *software* visualizador, ou, alternativamente, através da construção de um programa J2ME. Neste trabalho, cenas que obedecem as restrições acima serão denominadas simples.

Mesmo parecendo simples, há a necessidade de programar diversos aspectos da cena, como câmera, teclas de ação, *world*, iluminação, dentre outras. Todos estes aspectos estão definidos em *Canvas*, e são chamados na *midlet* principal. Maiores detalhes desta implementação podem ser encontrados em http://www.mobilefish.com/.

A Figura 28 mostra as classes do pacote *javax.microedition*.M3G (Sun 2007).

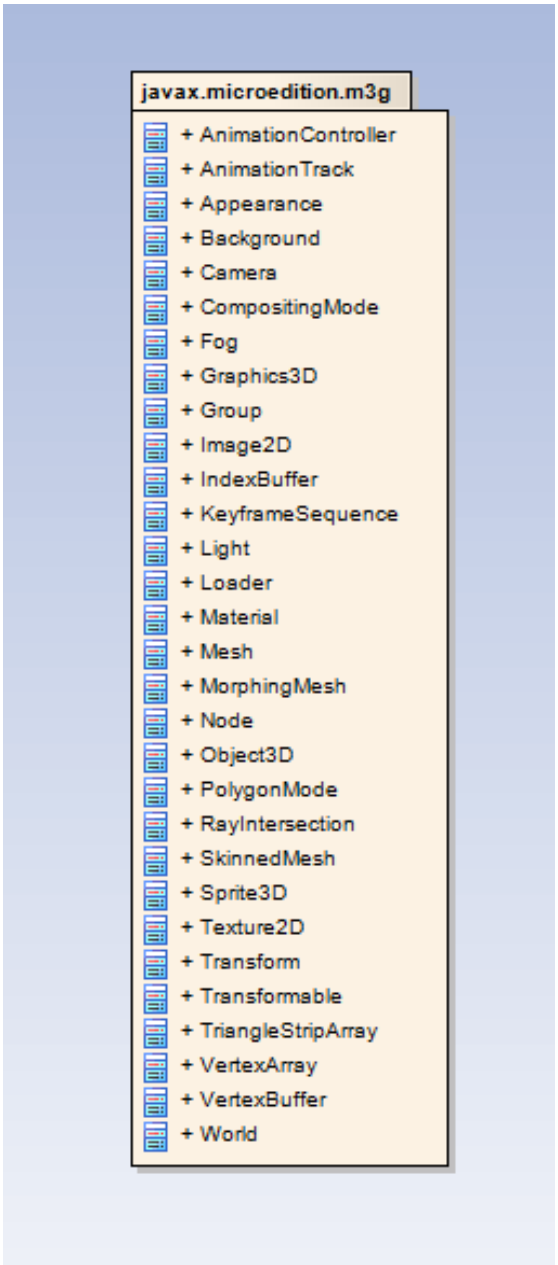



**Figura 28– Classes do pacote javax.microedition.M3G (Sun 2007).**

## 5.2. Encapsulamento de detalhes de programação para visualização de cenas M3G em *Midlets*

O objetivo desta parte do trabalho é separar todos estes detalhes de programação de baixo nível numa classe específica, livrando o *design*er gráfico de conhecer todos estes parâmetros.

Para avaliar esta possibilidade, foram escolhidos alguns parâmetros considerados gerais na maioria das aplicações de Realidade Virtual, como a posição e movimentação de uma câmera, efeitos de translação e rotação de objetos e também efeitos de

navegação em primeira pessoa pela cena. Todos estes parâmetros foram previamente programados e encapsulados na classe *Scene3D*, que após compilada, pode ser acrescida como recurso a qualquer *MIDlet*, tornando o uso destas ações bastante simples.

A Figura 29 apresenta a tela do *Scene3D*, onde a aplicação contém apenas a chamada da Imagem 3D e a importação da classe *Scene3D*. Observa-se que esta classe está compilada como recurso do projeto em questão (arquivo *Scene3D*.jar), e que a execução na tela do simulador obedece aos comandos de rotação, translação e zoom, definidos dentro de *Scene3D*.jar.

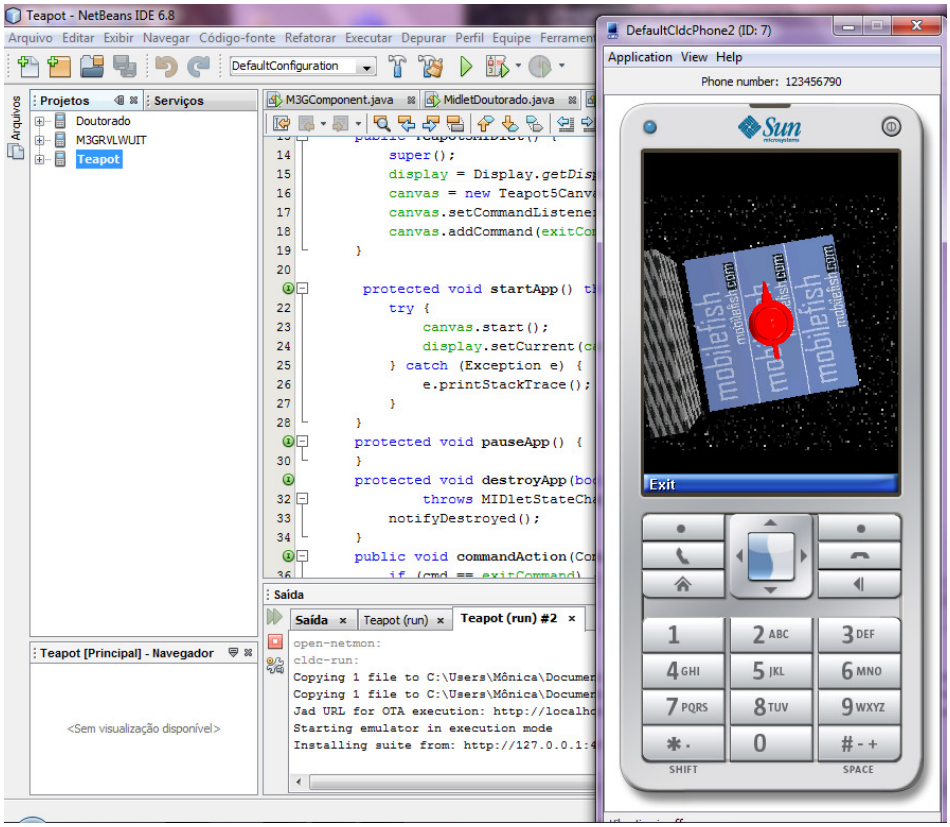



**Figura 29 – Visualização de cena no "retained mode" (MobileFish 2008), com detalhes da implementação do canvas.**

A classe *Scene3D* tem apenas 2 construtores, e sua chamada leva apenas o nome da cena M3G a ser carregada e, se necessário, as velocidades de rotação e translação da câmera. A Figura 30 mostra o diagrama de classes de *Scene3D* e seu relacionamento com as classes existentes no pacote M3G original.

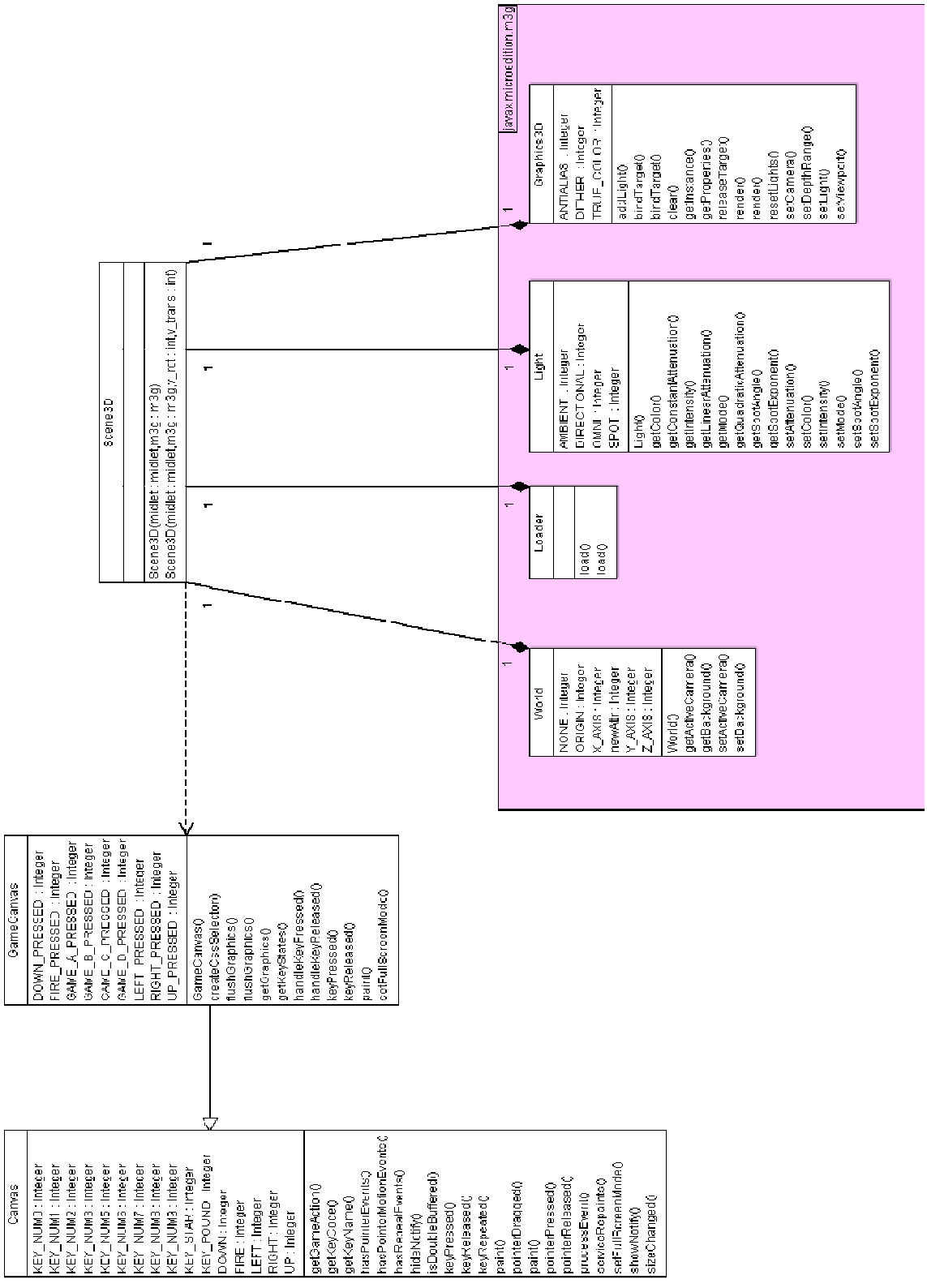



**Figura 30 - Diagrama de classes Scene3D e seus relacionamentos com as classes M3G, Canvas e GameCanvas.**

A Figura 31 apresenta detalhes da implementação da classe *Scene3D*, encapsulando os detalhes de *Canvas* e promovendo a chamada direta e objetiva de arquivos M3G.

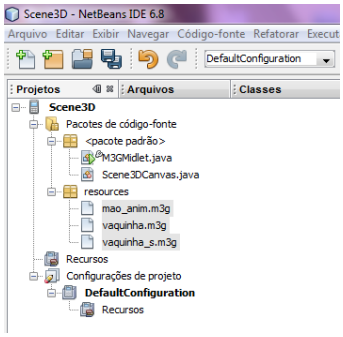



**Figura 31 - Detalhe da implementação da classe Scene3D (arquivos desenvolvidos).**

Seu uso se resume então a seguinte chamada:

Scene3D*Canvas* m3G*Canvas* = new Scene3D*Canvas*(this, "nome_do_arquivo.M3G");

Os parâmetros foram escolhidos analisando cenas 3D desenvolvidas no Laboratório de Computação Gráfica da UFU pelos alunos de graduação e pós-graduação, para disciplinas diversas e utilizando softwares também diversos. O plugin do Blender (desenvolvido em Pyton), possui a opção de gerar o arquivo M3G ou seu equivalente em Java. Através da opção de arquivo Java, foi possível verificar estes parâmetros e selecioná-los adequadamente. Uma vez selecionados estes parâmetros, a programação dos mesmos segue o padrão de programação do *Canvas* para cenas M3G. Na classe criada (*canvas* – renomeada para *Scene3D*), criou-se os construtores e em seguida, a mesma foi compilada, gerando o arquivo *Scene3D*.jar. Este pode então ser importado por qualquer aplicativo e usado apenas com a chamada dos construtores.

Diversos testes foram executados, com cenas diversas (escolhidas aleatoriamente – porém respeitando as restrições citadas – com o objetivo de testar a biblioteca), conforme pode ser visto na Figura 32 (utilização de *Scene3D* numa *MIDlet*, apontando para a chamada da cena M3G e para a importação da biblioteca Scene3D, no simulador do *NetBeans*) e Figura 33 (visualização de algumas das cenas escolhidas para teste, no simulador do *NetBeans*). Todos os testes foram feitos trocando-se apenas o nome da cena a ser apresentada na tela do simulador. Todos os recursos esperados de navegação na cena funcionaram perfeitamente bem, como mostrado na Figura 34 (visualização da navegação em uma cena, no simulador do *NetBeans*).

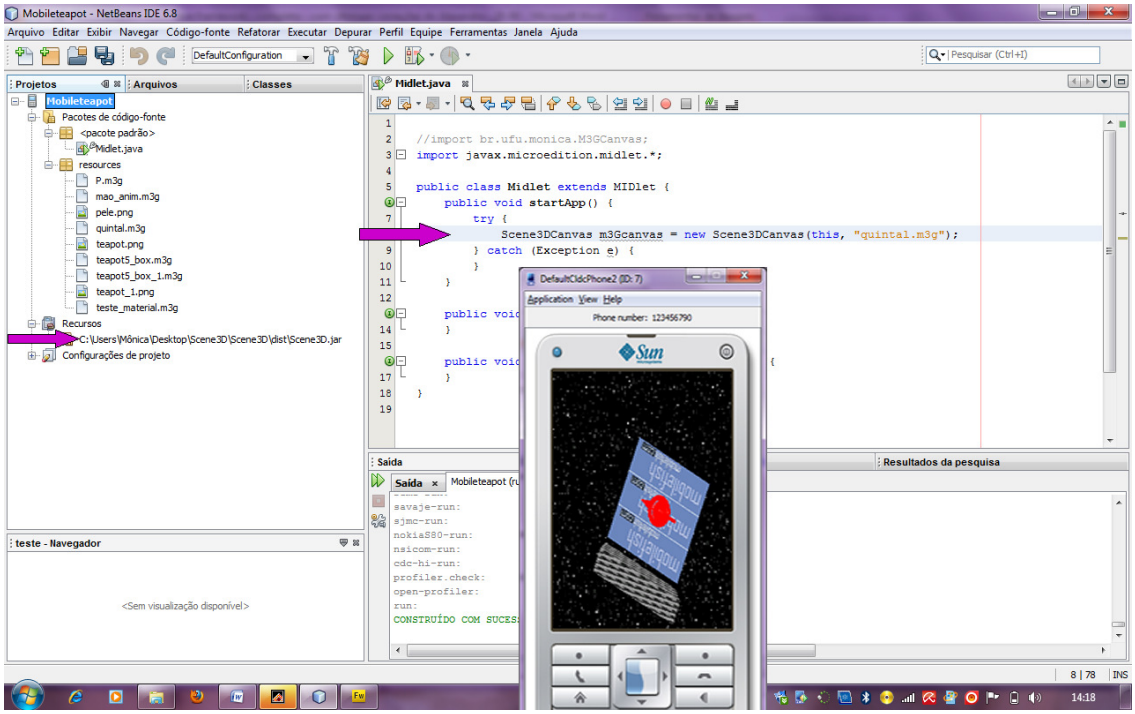



**Figura 32 - Utilização da classe Scene3D para chamada simples de cenas de RV.**

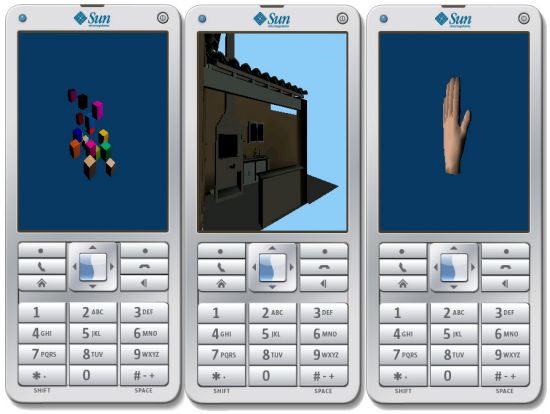


**Figura 33 - Visualização de cenas 3D utilizando a classe Scene3D.**

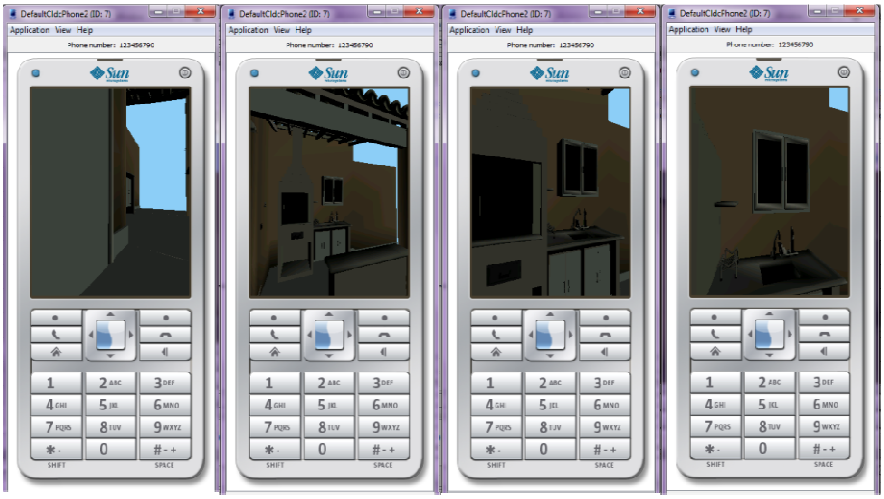


**Figura 34 – Exemplo de navegação na cena utilizando Scene3D.**

## 5.3. Descrição do pacote LWUIT

No que diz respeito à linguagem Java, a programação de *Interfaces* Gráficas com o Usuário, frequentemente designadas por GUIs (*Graphical User Interfaces*), teve seus primeiros passos com o *Abstract Window Toolkit* (AWT).

O AWT tem estado presente em todas as versões do Java. Os objetos AWT são construídos sobre objetos de código nativo, o que fornece um *look-and-feel* nativo. Costumam ser designados como "o menor denominador comum de todas as plataformas".

Os objetos *Swing* são escritos também em Java puro e apresentam o mesmo *look-and-feel* em todas as plataformas. No entanto, podem ser adaptados pelo programador para estilos particulares. Por este motivo, os objetos AWT caíram em desuso e em importância ao longo do tempo (Campos 2006).

De forma semelhante, tanto na terminologia como no *design*, os componentes LWUIT são colocados em um recipiente, e estes recipientes podem ser aninhados de maneira similar ao *Swing*/AWT. A Figura 35 mostra o diagrama de classes resumido de AWT e *Swing*.

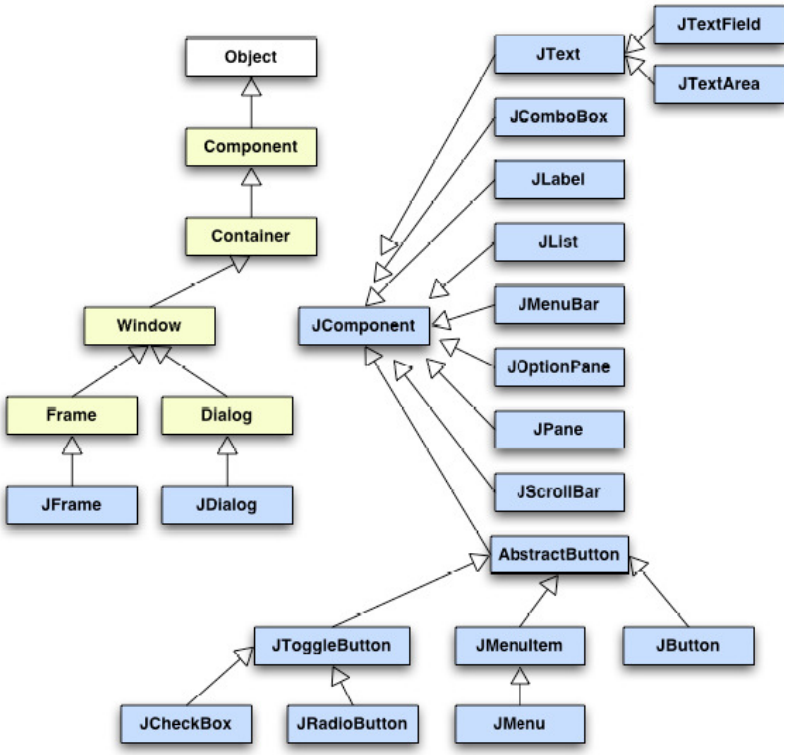



**Figura 35 - Diagrama resumido das classes AWT (amarelo) e Swing (azul) (Campos 2006).**

### *a)* ***LWUIT***

O pacote *com.sun.lwuit* é o principal pacote *widget*[7] que contém o componente (*container*), semelhante tanto na terminologia e *design* com o *Swing* e o AWT. Ao contrário do *Swing* / AWT, neste caso não se aplica um completo sistema de janelas, e as formas são colocadas usando uma abstração de exibição mais de acordo com a API MIDP.

Os componentes são colocados em um recipiente com os gerentes de *layout* que são usados para determinar o posicionamento de componentes (*com.sun.lwuit.layouts*), Estes recipientes podem ser aninhadas indefinidamente de uma maneira similar ao *Swing* / AWT. Todos os componentes são leves e elaborados pelo *UIManager* que permite a criação de temas usando estilos. *Também* permite a personalização da *interface* do usuário elaborada por derivação *look-and-feel* e substituindo métodos específicos para desenho / dimensionamento de componentes (Microsystems 2009).

O *container* é o recipiente padrão composto de um componente objeto. Permite colocar e organizar vários componentes usando uma arquitetura conectável de gerenciamento de *layout*. Recipientes podem ser aninhados uns dentro de outros para formar *interfaces* elaboradas.

Componentes adicionados a um recipiente são controlados em uma lista. A ordem da lista define a ordem do empilhamento dentro do *container*.

Um componente é um objeto com uma representação gráfica que pode ser exibido na tela e pode interagir com o usuário. Os botões, caixas de seleção e botões de rádio em uma *interface* gráfica típica são todos exemplos de um componente. Componente é a classe base do LWUIT.

O formulário (*Form*) é um componente de nível superior que serve como a raiz para a biblioteca da *interface*. Este *container* lida com o título e os menus e permite que

[7] Um *widget* pode ser definido em termos de um número de características comuns, como c*ores em background e foreground (*cada componente tem quatro atributos de cores: dois para cada *background e foreground*), f*onte text* (O texto pode ser renderizado usando estilos de fonte padrão como suportado pela plataforma, bem como fontes *bitmap*), T*ransparência background, imagem background*, m*argin e padding*.

o conteúdo seja colocado entre eles. Na Figura 36 pode-se visualizar as partes de um formulário.

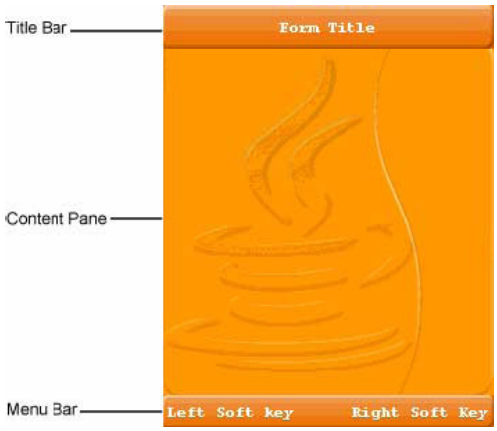



**Figura 36 - Componentes do formulário (Campos 2006).**

Porque o tamanho da tela é limitado, as listas são a base das UI *Widget* nos dispositivos. A lista apresenta ao usuário um grupo de itens exibidos em uma única coluna. O conjunto de elementos é processado usando uma *ListCellRenderer* e são extraídos usando o *ListModel.* O modelo da arquitetura *Swing* / *View* / *Controller* (MVC) torna possível uma lista representar muitos outros conceitos. O componente lista é relativamente simples. Ele chama o modelo, a fim de extrair o apresentado ou a informação selecionada, e chama o renderizador de célula para mostrá-lo para o usuário. A própria classe lista é totalmente dissociada de tudo, para que seja possível extrair o seu conteúdo de qualquer origem (por exemplo, a rede, armazenamento, etc.) e apresentar a informação, sob qualquer forma (por exemplo, caixas, ícones, entre outros) (Campos 2006).

A  apresenta o diagrama de pacotes do *framework* LWUIT, chamando a atenção para a separação entre a leitura de eventos (lwuit.events), necessária a navegação na cenas M3G, a implementação de *Canvas* (*lwuit.impl.midp*), necessária a programação de câmera, luz, etc., e a classe M3G e a *interface* M3G.*Callback*, presente em LWUIT.

É a *interface* M3G.*Callback* que permite a renderização de gráficos 3D. Esta *interface* é invocada como um resultado de uma chamada *renderM3G*. É da responsabilidade do programador utilizar este método adequadamente. O método descrito nesta *interface* é o *paintM3G*, que tem como parâmetros um gráfico M3G.

Quando este método é chamado, tem-se como resultado uma chamada *renderM3G* (classe M3G), que recebe um objeto pronto *Graphics*3D. É necessário que o programador certifique-se de que o retorno desta chamada esteja adequado a este método.

*RenderM3G* faz parte da classe M3G, que é a classe suporte para ligação a API gráfica 3D M3G (JSR 184). É ela que vai permitir integrar a *interface* 2D com efeitos especiais e transições 3D, mantendo os canais de renderização em sincronia. Essa classe é um *singleton* (em padrões de projetos, classe que garante a existência de apenas uma instância, mantendo um ponto global de acesso ao seu objeto) que inclui uma *interface* de *Callback* que abstrai a separação entre o *pipeline* 2D e 3D. Na Figura 37 podem ser visualizadas as classes do LWUIT.

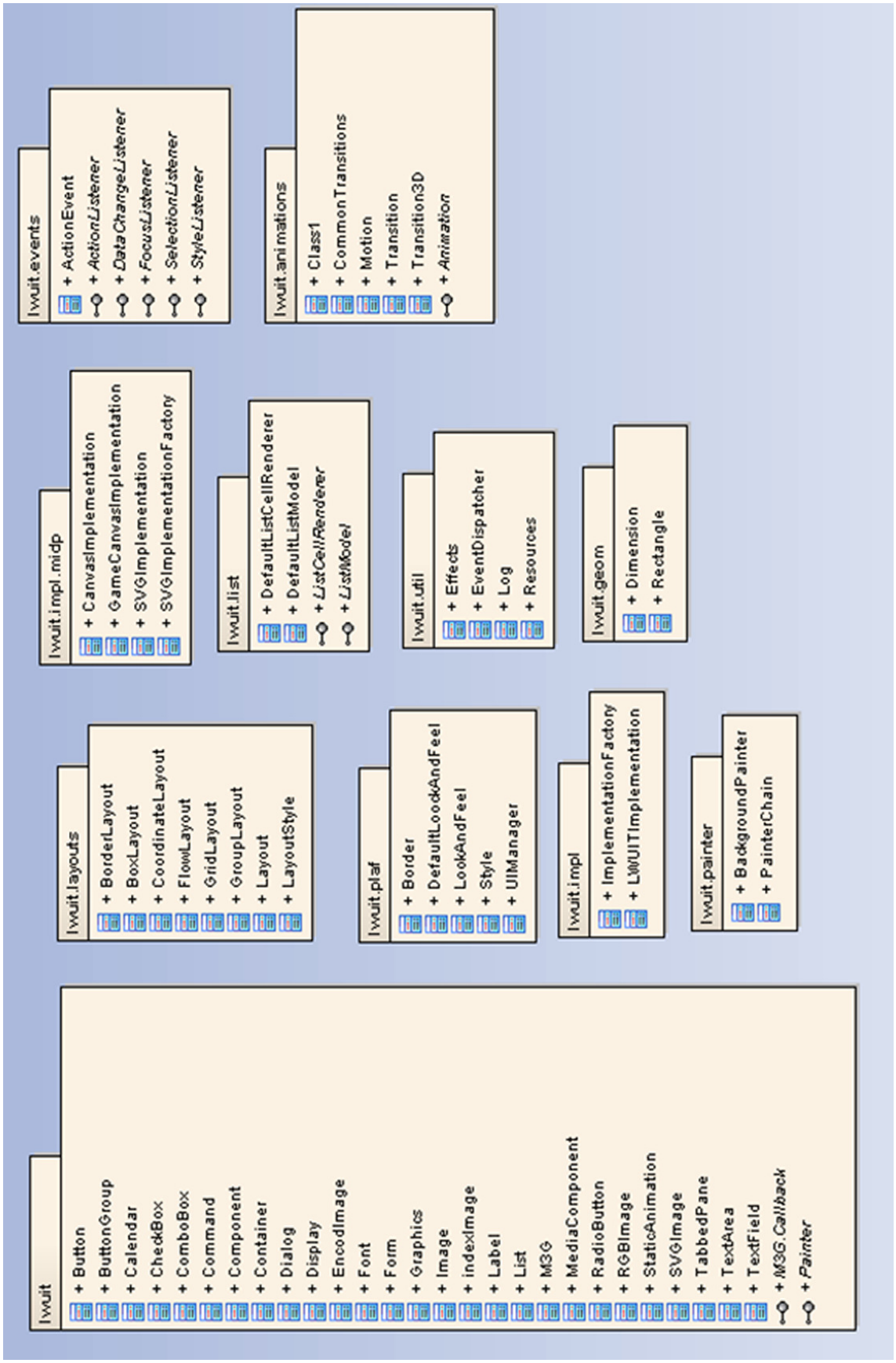



**Figura 37 - Diagrama de pacotes LWUIT (Campos 2006).**

Da forma como estão definidas as classes no *framework* LWUIT, existe a possibilidade de inserção de cenas M3G, porém de maneira similar ao que era possível utilizando *javax.microedition*.M3G. Ou seja, exige-se um esforço computacional na medida em que são necessárias as diversas configurações de câmera, navegação, etc. M3G está presente no LWUIT para integrar à *interface* 2D efeitos especiais e de transições 3D e também para permitir a programação das cenas em baixo nível (*Canvas*).

A renderização de imagens M3G utiliza o método *renderM3G*, tem como parâmetro um objeto *Graphics*, que é um resumo do contexto da plataforma de gráficos e permite alcançar a portabilidade entre dispositivos MIDP e dispositivos do CDC. Esta abstração simplifica e unifica as implementações de gráficos de várias plataformas dentro do *framework*. Uma instância gráfica nunca é criada pelo programador, e estará sempre acessado usando uma chamada de *painter* ou uma imagem mutável. Não há nenhuma maneira para criar esse objeto diretamente. O outro parâmetro do método *renderM3G*, é uma chamada M3G*Callback*.

A classe M3G pode ser vista na Figura 38, juntamente com todos os métodos nela implementados, dentro do *framework* LWUIT.

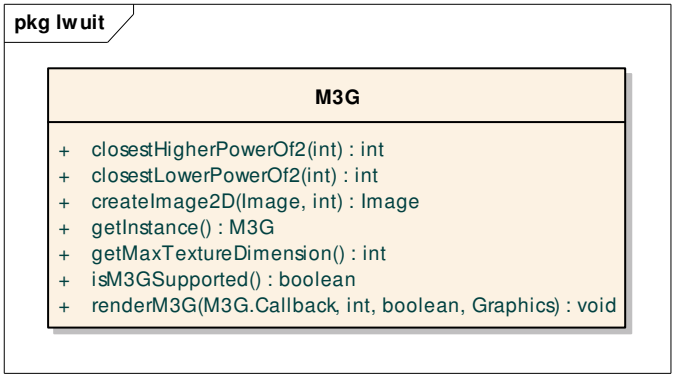



**Figura 38 - Métodos da classe M3G no LWUIT.**

Algumas aplicações podem optar por utilizar as capacidades 3D, a fim de proporcionar uma experiência de usuário mais atraentes por integrar 3D com efeitos especiais da *interface* do usuário 2D (por exemplo, um cubo como efeito de transição entre os formulários da aplicação). O caso de uso principal gira em torno de desenho de elementos 3D dentro do *framework* LWUIT ou o uso de *widget*s LWUIT. Na Figura 39 pode-se visualizar o diagrama de classes do *framework* LWUIT.

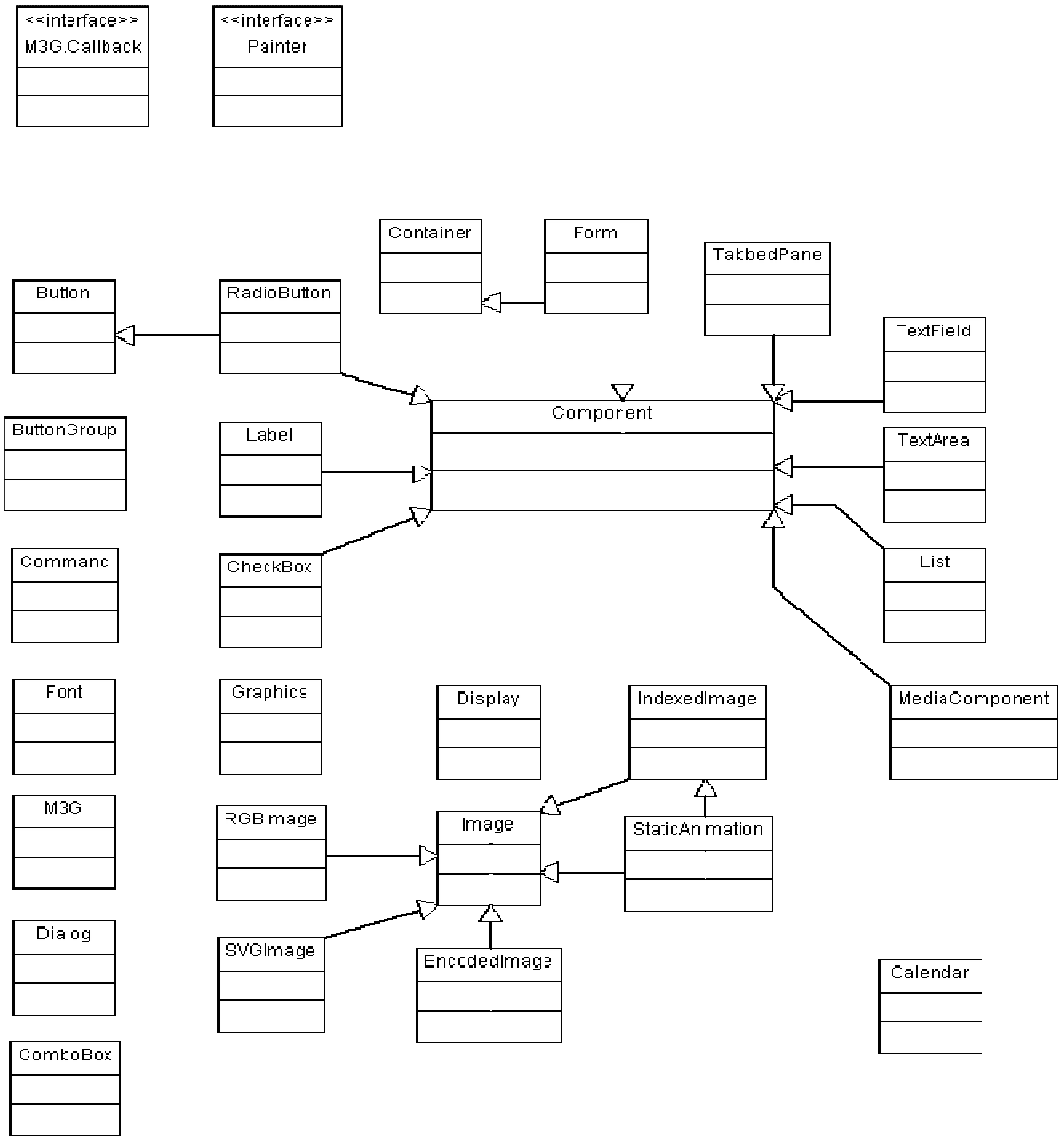



**Figura 39 - Diagrama de classes LWUIT.**

Normalmente M3G está diretamente vinculado ao objeto gráfico ou imagem durante o padrão renderização. O processo, entretanto, dentro do *framework* LWUIT, a chamada a estes tipos de objetos não funciona.

A integração M3G ao LWUIT é construída em torno de um mecanismo de retorno que permite ao desenvolvedor vincular um objeto gráfico LWUIT a um objeto *Graphics*3D M3G (sendo concebido para funcionar apenas em dispositivos que suportam – se o dispositivo não suporta a execução de M3G, o LWUIT evita a ativação desta parte do código).

***b) Padrões de projetos e LWUIT***

Padrões de projeto incentivam a reutilização de *design* através de estabelecidas concepções de soluções de sucesso para situações particulares, também conhecido como contextos. É uma valiosa ferramenta para o *design* orientado a objeto de uma série de importantes razões (Christiansson 2008):

- Padrões fornecem "... uma solução para um problema em um contexto."
- Padrões capturam a experiência de designers experientes de uma forma metódica e tornam-se ferramentas de projeto e de aprendizagem disponíveis para não-especialistas.
- Os padrões fornecem um vocabulário para discutir projeto orientado a objetos a um nível significativo de abstração.
- Padrões podem servir de catálogo ou glossário de expressões que ajudam na compreensão comum, bem como nas soluções complexas para problemas de design.

Entre 1991 e 1994, Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides, descreveram em seu livro *Design patterns: elements of reusable object-oriented software,* 23 padrões de projeto, cada um fornecendo uma solução a um problema comum de projeto de software na indústria. O livro agrupa os padrões de projeto em três categorias — padrões de projeto criacionais, padrões de projeto estruturais e padrões de projeto comportamentais (Deitel 2005).

- Padrões de projeto criacionais descrevem as técnicas para instanciar objetos (ou grupos de objetos). São eles: *Abstract Factory, Builder, Factory Method, Prototype, Singleton;*
- Padrões de projeto estruturais permitem que os projetistas organizem classes e objetos em estruturas maiores. São eles *Adapter, Bridge, Composite, Decorator, Façade, Flyweight, Proxy;*
- Padrões de projeto comportamentais atribuem responsabilidades a classes e objetos. Eles são: *Chain of Responsability, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, Visitor.*

Os padrões de projetos listados por Deitel (2005), associados aos componentes GUI do Java são:

a. Padrão de projeto *Factory Method*. O propósito exclusivo desse método é criar objetos permitindo que o sistema determine qual classe instanciar

em tempo de execução. Pode-se projetar um sistema que permita a um usuário especificar qual tipo de imagem criar em tempo de execução.

b. Padrão de projeto *Adapter* ajuda os objetos com interfaces incompatíveis a colaborar entre si.
c. Padrão de projeto *Bridge* ajuda os projetistas a aprimorar a independência de plataformas nos seus sistemas.
d. Padrão de projeto *Composite* fornece uma maneira para que projetistas organizem e manipulem objetos.
e. Padrão de projeto *Chain of Responsibility* determina em tempo de execução o objeto que tratará uma mensagem particular.
f. Padrão de projeto *Command* permite que um projetista encapsule um comando de modo que ele possa ser utilizado entre vários objetos. Define um comando, ou instrução, a ser executado.
g. Padrão de projeto *Observer* promove o acoplamento fraco entre um objeto-assunto e objetos observadores — um objeto-assunto notifica os objetos observadores quando o assunto altera o estado.
h. Padrão de projeto *Strategy* utiliza uma superclasse abstrata que os métodos que descrevem os comportamentos dos estados dos objetos *strategy*. O objeto *strategy* encapsula um algoritmo em vez de informações sobre o estado. Permite que vários objetos contenham algoritmos distintos.
i. Padrão de projeto *Template Method* também lida com algoritmos. Ele requer que todos os objetos compartilhem um único algoritmo definido por uma superclasse.

A análise da documentação fornecida por (Microsystems 2009), mostra que todos estes padrões citados por Deitel (Deitel 2005) foram também utilizados na construção do *framework* LWUIT.

Um exemplo da utilização do Padrão de Projetos neste *framework* é o uso do Padrão *Composite*, que permite o assentamento e a organização dos múltiplos componentes, gerenciando o *layout*. Os recipientes podem ser aninhados um dentro do outro, de forma elaborada, permitindo o projeto da *interface* do usuário de forma complexa. Este padrão de Projetos é utilizado na Classe *Container*, que extende *Component*. Esta, por sua vez, é a classe base para todas as classes que utilizam o

padrão *Composite*, de uma forma similar ao AWT (Microsystems 2009). Este Padrão de Projetos também é utilizado nas classes *Label, List, e TextArea.*

## 5.4. Modificação do *framework* LWUIT agregando cenas M3G e utilizando Scene3D.

A segunda parte deste trabalho trata de integrar as cenas M3G, em específico, algo similar a classe *Scene3D* desenvolvida anteriormente neste trabalho, ao *framework* LWUIT, de modo que ela realmente obedeça a todos os efeitos de *interface* definidos dentro do *framework*, ou seja, dentro desta mesma classe, separar os mesmos detalhes de programação da hipótese anterior, numa classe específica, livrando o *design*er gráfico de conhecer todos estes parâmetros.

Para avaliar esta possibilidade, foram escolhidos os mesmos parâmetros considerados gerais na maioria das aplicações de Realidade Virtual, como a posição e movimentação de uma câmera, efeitos de translação e rotação de objetos e também efeitos de navegação em primeira pessoa pela cena.

Para resolver esta hipótese, e atender a certeza de que o *framework* continuará a acessar a classe M3G através da *interface* M3G*Callback*, optou-se por implementar uma nova classe chamada RV_LWUIT (canto superior direito da figura 40 ), integrada a arquitetura LWUIT. Desta forma, mantém-se a classe M3G, utilizada normalmente para os efeitos propostos pelo *framework* e a classe desenvolvida neste trabalho, RV_LWUIT, integrada ao conjunto de classes existentes, será então utilizada para chamada de cenas 3D, através de um contêiner no LWUIT. A Figura 40 mostra o diagrama de classes LWUIT com a alteração proposta nesta tese. Observa-se que RV_LWUIT será colocada num contêiner LWUIT.

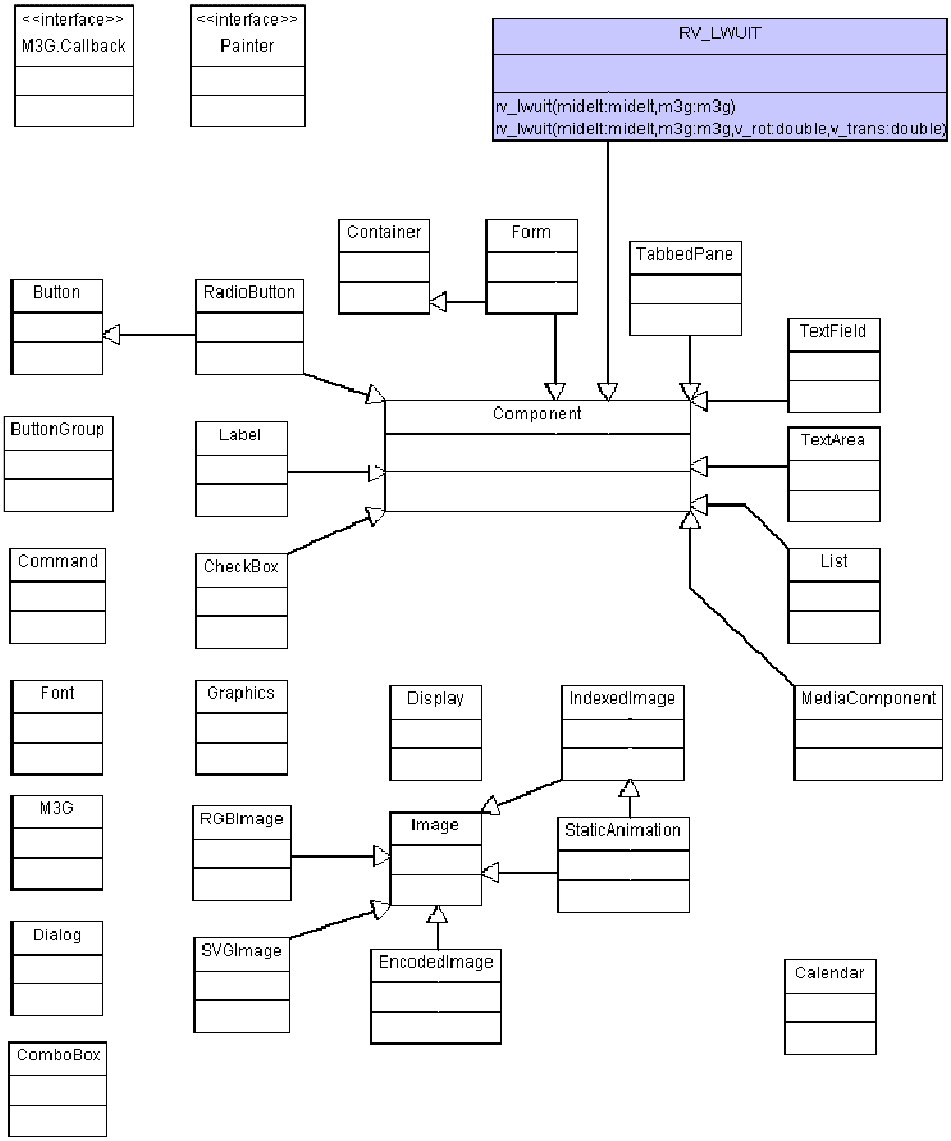



**Figura 40 - Diagrama de classes modificado do LWUIT.**

Uma visão da arquitetura proposta pode ser vista na Figura 41, onde o LWUIT está posicionado sobre o M3G e MIDP (fazendo uso de J2ME e Open GL ES). O acréscimo proposto busca na camada inferior (M3G) as necessidades de navegação, e na camada LWUIT na montagem de interfaces. Algumas classes da camada M3G foram necessárias para prover interatividade. Foi necessário o desenvolvimento de um *container* para integração total ao LWUIT, pois todos os componentes LWUIT são acondicionados em *containers* para apresentação na tela do dispositivo. Este *container* foi desenvolvido neste trabalho atendendo aos padrões de projetos utilizados no *framework.*

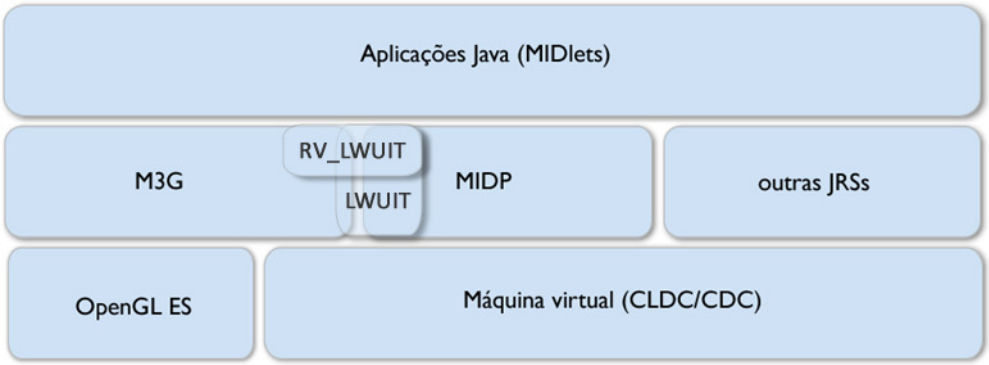



**Figura 41 - Arquitetura Java com RV_LWUIT**

Utilizou-se o Padrão de Projetos *Command*, mantendo assim as características originais do *framework* LWUIT.

A integração dos recursos de RV precisa ser feita com as classes que provêem estas ações dentro do LWUIT, uma vez que as classes *Canvas* e game*Canvas*, utilizadas no encapsulamento naterior (*Scene3D*), são incompatíveis com o mesmo. A integração considera que as cenas M3G respeitarão todas as restrições comentadas no item 5.1 (a) a respeito das cenas M3G, e a renderização destas obedecerá a todos os efeitos implementados no LWUIT.

A classe desenvolvida, RV_LWUIT é então a responsável por construir o *Canvas* e as ações de game*Canvas* para as cenas que se deseje apresentar. Este *Canvas* contém todas as especificações necessárias para se construir a cena 3D, e o diagrama de caso de uso do LWUIT, mostrado na Figura 42, permitindo a chamada simples de uma cena de RV, lendo um arquivo M3G e apresentando-o na tela do dispositivo portátil com os recursos de navegação de RV considerados gerais na maioria das aplicações, como a posição e movimentação de uma câmera, efeitos de translação e rotação de objetos e também efeitos de navegação em primeira pessoa pela cena. As Tabela  e 7 trazem a documentação dos casos de uso e as Figuras 42 e 43 o diagrama de casos de uso e o diagrama de atividades do mesmo, respectivamente.

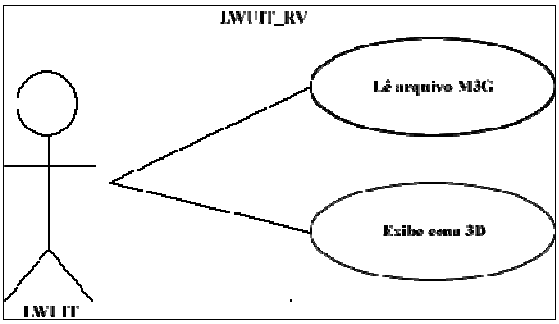



**Figura 422 - Diagrama e documentação de casos de uso para LWUIT_RV**

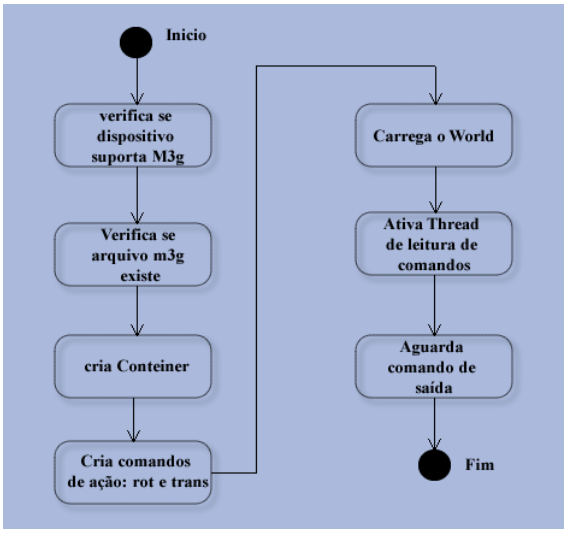



**Figura 43 – Diagrama de atividades de criação da cena.**

**Tabela 5 – Documentação do caso de uso LÊ ARQUIVO M3G.**

| **Nome do caso de uso** | **Lê arquivo M3G** |
|---|---|
| **Caso de uso geral** | |
| **Ator principal** | LWUIT |
| **Atores secundários** | - |
| **Resumo** | Este caso de uso descreve as etapas percorridas pelo LWUIT para encontrar e carregar um arquivo M3G |
| **Pré-condições** | - |
| **Pós-condições** | Se o arquivo M3G com o nome especificado existir, ele estará carregado e pronto para ser exibido na tela do dispositivo portátil. |
| **Prioridade:** | X Essencial __ Importante __ Desejável |
| **Ações do ator** | **Ações do sistema** |
| 1 – Informa nome do arquivo M3G (exigido), velocidade de rotação (opcional) e velocidade de translação (opcional). | |
| | 2 – Verifica se o dispositivo suporta o formato M3G, se não suporta, emite mensagem e termina. |
| | 3 – Se suporta, verifica se o arquivo M3G existe. Se não existe, emite mensagem de erro e termina. |
| | 4 – Se existe, cria o contêiner utilizando um formulário do LWUIT.*Component*. |
| | 5 – Chama a *interface* M3G*Callback*, utilizando LWUIT.M3G. |
| | 6 – Verifica se há animação de animação no dispositivo, utilizando LWUIT.*Component*. |
| | 7 – Define o tamanho de apresentação do arquivo de acordo com o *display* do dispositivo portátil, utilizando LWUIT.*Component*. |
| | 8 – Define as ações de navegação utilizando Game*Canvas*. |
| | 9 – Associa as ações de navegação às teclas do dispositivo, utilizando Game*Canvas*. |
| | 10 – Carrega todas as informações na memória do dispositivo. |
| **Restrições** | - |

**Tabela 6 - Documentação do caso de uso EXIBE CENA 3D.**

| **Nome do caso de uso** | **Exibe cena 3D** |
|---|---|
| **Caso de uso geral** | |
| **Ator principal** | LWUIT |
| **Atores secundários** | - |
| **Resumo** | Este caso de uso descreve as etapas percorridas para apresentar a cena 3D na tela do dispositivo portátil, com recursos de navegação. |
| **Pré-condições** | Informações sobre o arquivo e sobre a navegabilidade na cena devem estar na memória do dispositivo. |
| **Pós-condições** | A cena estará na tela do dispositivo, com possibilidade de navegação na mesma. |
| **Prioridade:** | X Essencial __ Importante __ Desejável |
| **Ações do ator** | **Ações do sistema** |
| 1 – Solicita a apresentação dos dados do arquivo M3G na tela do dispositivo. | |
| | 2 – Carrega o Word, utilizando M3G.*World*. |
| | 3 – Carrega objeto 3d, utilizando M3G.*Loader*. |
| | 4 – Carrega informações adicionais como câmera, iluminação, etc., utilizando M3G.*Light* e M3G.Graphics3D. |
| | 5 – Ativa *thread* de leitura das teclas, aguardando comandos de navegação pela cena. |
| | 6 – Aguarda comando de saída. |
| **Restrições** | - |

Novamente os métodos de RV_LWUIT são apenas os construtores, onde define-se a *MIDlet* onde a cena irá ser apresentada e passa-se o nome do arquivo M3G. Como opção, pode-se também informar as velocidades de rotação e de translação na cena. São utilizados os recursos do *framework* LWUIT original, como *isM3Gsupported( ) e Threads* do LWUIT.

As classes *Canvas* e *GameCanvas* são usadas para construção das possibilidades de rotação, translação, zoom e navegação nas cenas 3D, o que significaria "imersão" num mundo virtual no dispositivo portátil. Elas fazem parte da *interface* de baixo nível presente no J2ME. Como prova de conceito, foram implementadas a posição e movimentação de uma câmera, efeitos de translação e rotação de objetos e também efeitos de navegação em primeira pessoa pela cena. Na chamada do método construtor há, a possibilidade de inserir velocidades apenas de rotação e translação.

É ainda importante salientar a diferença entre as Figuras 30 e 44, sendo que na primeira, não há nenhuma referência ao *framework* LWUIT, e na segunda, há a utilização do *Container* e da *Interface M3GCallback*, do *framework* para realizar a integração da classe RV_LWUIT ao LWUIT.

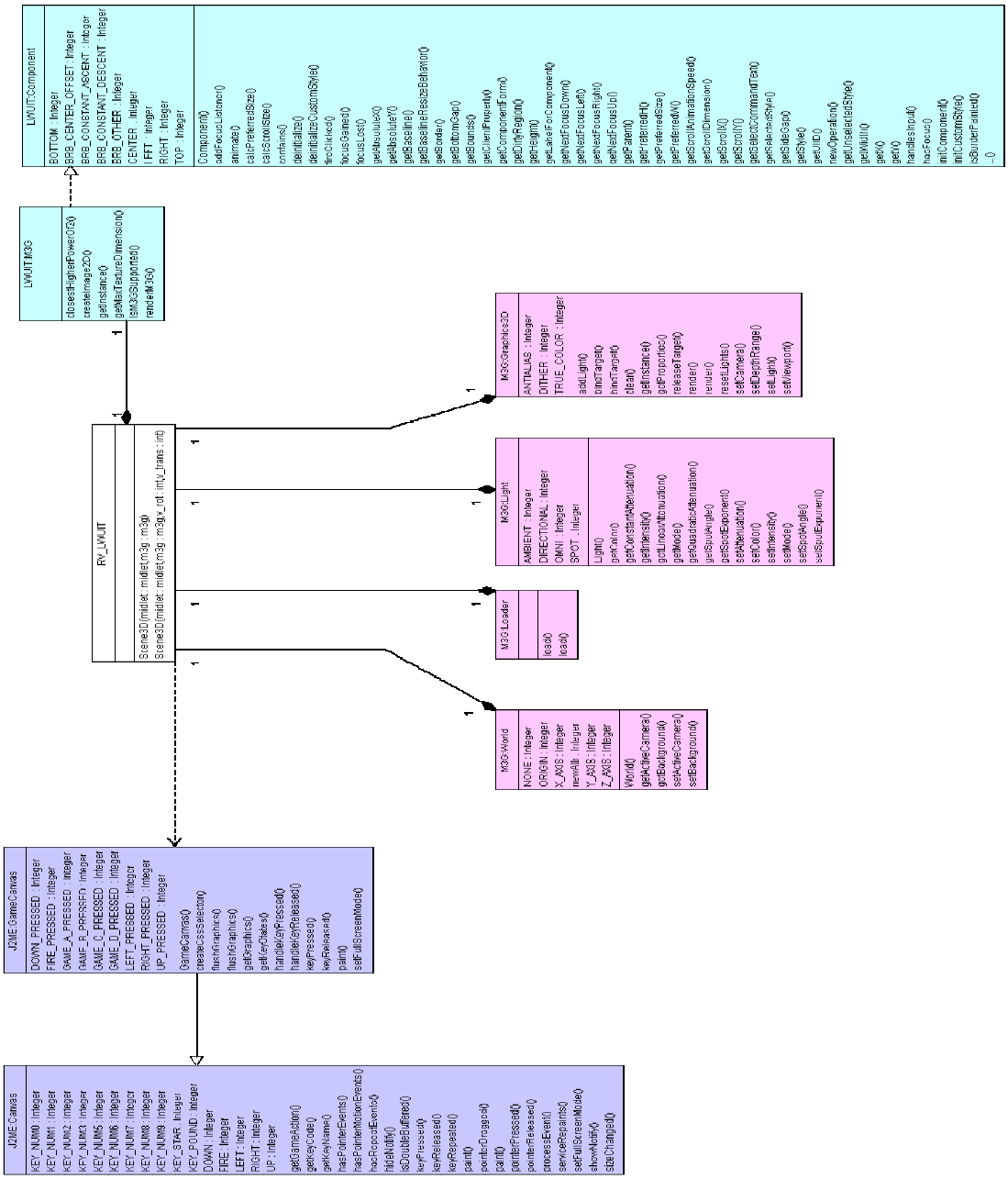



**Figura 44– Proposta de modificação no framework LWUIT para suportar RV.**

As classes *Word, Loader, Light e Graphics3D*, do pacote M3G – JRS 184, são utilizadas para a construção do mundo virtual, atendendo as restrições comentadas no item 5.1 (b).

Com a alteração proposta na Figura 44, consegue-se que as cenas M3G passem a configurar como um objeto dentro de um contêiner, ou seja, como qualquer componente de um formulário comum (*textbox, list, etc*.). Também consegue-se que não sejam necessários os detalhes de implementação do *canvas*, reduzindo o trabalho de programação, conforme proposto. A comunicação entre os módulos envolvidos na construção da classe RV_LWUIT pode ser visto na Figura 45.

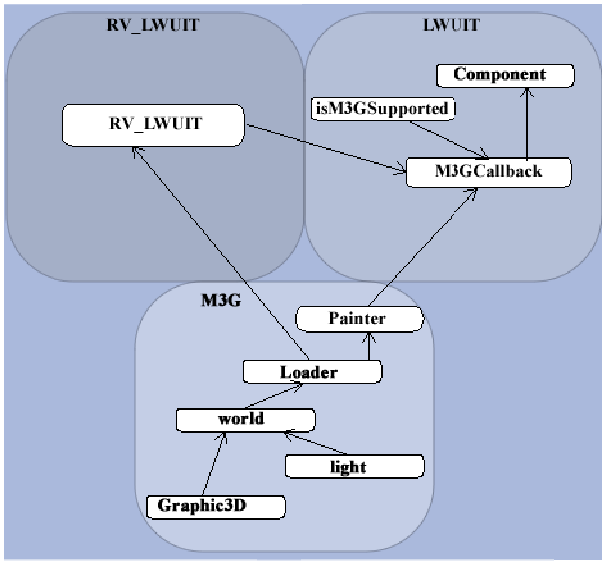



**Figura 45 - Comunicação entre os módulos.**

MidletDoutorado.java | M3GComponent.java

```
1
2  import com.sun.lwuit.Form;
3  import com.sun.lwuit.Graphics;
4  import com.sun.lwuit.M3G;
5  import javax.microedition.m3g.Graphics3D;
6  import javax.microedition.midlet.MIDlet;
7
8  public class M3GComponent extends Form {
9
10     private M3G.Callback callback;
11
12     public M3GComponent(MIDlet midlet, String m3gName) {
13         this.callback = new M3GCanvas(midlet,m3gName,0.5,0.5);
14        }
15
16      public void paint(Graphics g) {
17         M3G.getInstance().renderM3G(g, true, Graphics3D.DITHER | Graphics3D.TRUE_COLOR, this.callback);
18     }
19 }
20
```

**Figura 46 – Criação do contêiner M3GComponent.**

Na Figura 46 é apresentada a criação do contêiner para a cena M3G, *M3GComponent*. São levados como parâmetros para este contêiner apenas a *midlet* onde a cena será apresentada e o nome do arquivo M3G que contém a cena. Este contêiner utiliza a *interface M3GCallback* do *framework* LWUIT e também o *M3GCanvas*, desenvolvido em *Scene3D*.

Uma consideração importante é a verificação da portabilidade, que é um dos pilares do *framework* LWUIT (os testes foram realizados em simuladores e emuladores diversos). Conforme pode ser visto na Figura 47, não houve perda de portabilidade, pois o mesmo continua funcionando em diversos emuladores e simuladores de dispositivos portáteis. É também importante ressaltar que, nesta mesma figura, foram utilizados dois emuladores e um simulador (*Nokia* S60), e, nos simuladores, houve a perda do movimento implementado ao objeto "mão.M3G". Já no simulador da *Nokia*, este movimento não foi perdido. Pode-se perceber claramente a integração total da classe RV_LWUIT ao *framework* LWUIT, pois as cenas agora aparecem como face do cubo (efeito de transição) e também permanecem com a manipulação implementada.

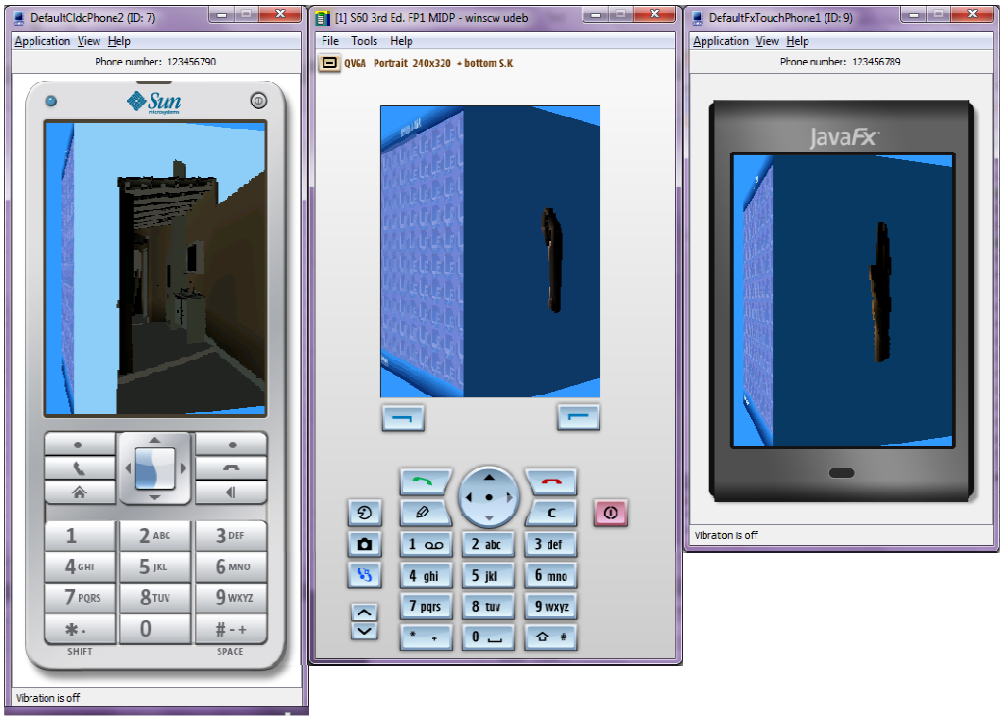



**Figura 47 - Teste de portabilidade do LWUIT com RV_LWUIT.**

## 5.5. Conclusões

Neste capítulo, provou-se que é possível separar os detalhes de programação envolvidos na construção do *canvas* numa classe específica, livrando os *designers* gráficos de conhecer todos estes itens de programação.

Foram escolhidos alguns parâmetros, considerados pela pesquisadora como gerais, na maioria das aplicações de Realidade Virtual, como a posição e movimentação de uma câmera, efeitos de translação e rotação de objetos e também efeitos de navegação em primeira pessoa pela cena.

Estes foram previamente programados e encapsulados na classe *Scene3D* e também na classe RV_LWUIT, que após compiladas, podem ser acrescidas como recurso a qualquer *midlet*, tornando o uso das cenas 3D bastante simples.

A classe *Scene3D* não necessita de nenhum detalhe a mais, a não ser a importação da mesma durante a escrita do programa. Já a classe RV_LWUIT necessita também de um *contêiner*, *M3GComponent*, para integrá-la ao *framework*. Com este *contêiner*, permanecem funcionando os recursos de navegação da câmera, os efeitos de rotação e translação de objetos e também de navegação em primeira pessoa, além de continuarem funcionando os efeitos de transição previstos no mesmo. Sem ele, apenas os recursos de navegação implementados funcionam.

Em ambos os casos, testes com várias cenas 3D foram realizados, e todos os recursos implementados permaneceram funcionando. No caso do LWUIT, testes de portabilidade foram também realizados, e não houve danos a mesma.

Também em ambos os casos, não foram levados em consideração os recursos para cenas animadas, ficando esta como sugestão para trabalhos futuros.

Foi mantida a utilização de padrões de projeto de software, de forma congruente ao desenvolvido originalmente no *framework* LWUIT.

Os testes foram realizados sobre a plataforma J2ME, configuração CLDC 1.1, perfil MIDP 2.0, utilizando os simuladores disponíveis no pacote NetBeans, acrescido de emuladores NOKIA S60, NOKIA E71, Motorola A5 e A6 e ainda o Siemens E80 disponível no SMTK CORE PACK V2. Todos os testes foram realizados utilizando um computador *Sony* com processador Intel Core 2 Duo, 200GHz, Sistema Operacional *Windows 7* (64 bits), 4 GB de RAM. Ainda, foi utilizado o *Wireless Toolkit* 2.5.2.

# 6. Estudos de casos

Que a arte nos aponte uma resposta
Mesmo que ela não saiba
E que ninguém a tente complicar
Porque é preciso simplicidade pra fazê-la florescer
Porque metade de mim é platéia
E a outra metade é canção.
O.M.*

Este capítulo apresentará duas aplicações utilizando as bibliotecas desenvolvidas. A primeira é uma aplicação utiliza a extensão RV_LWUIT no desenvolvimento de um software de tradução português Libras, onde os sinais Libras são visualizados em 3D, e permite-se a interação com o mesmo, possibilitando a visualização do sinal por diversos ângulos. A segunda é um software de visualização de cenas 3D em dispositivos portáteis, onde o usuário desenvolverá suas próprias cenas, as carregará em seu dispositivo portátil e poderá, utilizando o mesmo, visualizá-la sem que seja necessária nenhuma programação. Este software utiliza a biblioteca Scene3D.

## 6.1. Desenvolvimento de um software para visualização de cenas variadas em dispositivos portáteis

O software ERLENEN ( ERLENEN = aprender, em alemão) foi desenvolvido neste trabalho com o intuito de servir como ferramenta de auxílio a portadores de

deficiência auditiva a aprenderem novos conceitos e sinais da LIBRAS. Este nome foi escolhido porque os deficientes auditivos, com algum grau de alfabetização, ao entrarem em contato com uma nova palavra em língua portuguesa, tendem a "achar" que conhecem aquele conceito por "ser parecido" com algo que já "ouviram" ou viram. Esta é a mesma sensação que temos ao olhar a palavra ERLENEN (parece com aprender em inglês, será que não houve erro de digitação???). Estar diante de uma palavra desconhecida, cuja aparência nos é familiar: este foi o intuito da escolha do nome do software.

Ao aprender um conceito novo, numa língua que não é a sua, é imprescindível que o usuário visualize o conjunto de letras LIBRAS que compõem este novo conceito e, caso exista, o sinal que representa esta palavra em LIBRAS, de vários ângulos, podendo avaliar as diversas posições de dedos e mãos em relação ao corpo. Também é desejável que o usuário possa modificar a velocidade de apresentação do sinal, bem como das letras que formam a palavra em português, aprendendo assim a escrevê-la corretamente na língua portuguesa.

Portanto, o desenvolvimento deste software atende a necessidade de visualização de cenas variadas em dispositivos portáteis, sendo adequado como estudo de caso tratado deste trabalho.

A Figura 48 mostra o diagrama de caso de uso do sistema proposto como estudo de caso, e as tabelas 7, 8, 9 e 10, a documentação dos casos de uso do diagrama da Figura 48.

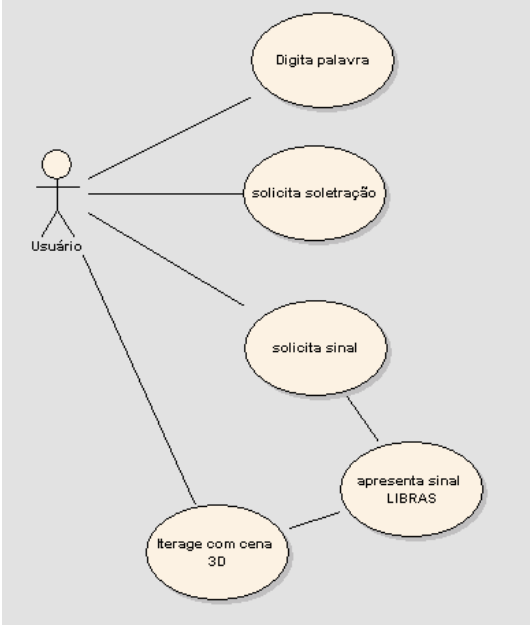



**Figura 48 - Diagrama de caso de uso do sistema ERLENEN.**

**Tabela 7 – Documentação do caso de uso DIGITA PALAVRA**

| **Nome do caso de uso** | **DIGITA PALAVRA** |
|---|---|
| **Caso de uso geral** | |
| **Ator principal** | **USUÁRIO** |
| **Atores secundários** | **-** |
| **Resumo** | **Este caso de uso descreve as etapas percorridas pelo usuário para pesquisar uma palavra no sistema.** |
| **Pré-condições** | |
| **Pós-condições** | **Se a palavra existir no sistema, o texto informativo do seu significado estará na tela do dispositivo.** |
| **Prioridade:** | **X Essencial __ Importante __ Desejável** |
| **Ações do ator** | **Ações do sistema** |
| **1 – Informa a palavra** | |
| | **2 – Procura o arquivo txt correspondente a palavra digitada.** |
| | **3 – Exibe o arquivo na tela.** |
| | **4 – Aguarda comando de saída.** |
| **Restrições** | **-** |

**Tabela 8 – Documentação do caso de uso SOLICITA SOLETRAÇÃO.**

| **Nome do caso de uso** | **SOLICITA SOLETRAÇÃO** |
|---|---|
| **Caso de uso geral** | |
| **Ator principal** | **USUÁRIO** |
| **Atores secundários** | **-** |
| **Resumo** | **Este caso de uso descreve as etapas percorridas pelo usuário para solicitar a soletração da palavra em LIBRAS.** |
| **Pré-condições** | **Já ter digitado uma palavra válida** |
| **Pós-condições** | |
| **Prioridade:** | **X Essencial __ Importante __ Desejável** |
| **Ações do ator** | **Ações do sistema** |
| **1 – Informa solicitação de soletração** | |
| | **2 – Enquanto houver letra na palavra, faça**<br>**i. Procure imagem correspondente à letra;**<br>**ii. Exibe sinal na tela;**<br>**iii. Aguarde 30 milisegundos;**<br>**iv. Passe para próxima letra.** |
| | **3 – Aguarda comando de saída.** |
| **Restrições** | **-** |

**Tabela 9 - Documentação do caso de uso SOLICITA SINAL.**

| Nome do caso de uso | SOLICITA SINAL |
|---|---|
| Caso de uso geral | |
| Ator principal | USUÁRIO |
| Atores secundários | - |
| Resumo | Este caso de uso descreve as etapas percorridas pelo usuário para solicitar a visualização do sinal correspondente a palavra em LIBRAS. |
| Pré-condições | Já ter digitado uma palavra válida |
| Pós-condições | |
| Prioridade: | __Essencial X_ Importante __ Desejável |
| Ações do ator | Ações do sistema |
| 1 – Informa solicitação de sinal | |
| | 2 – Se houver sinal para a palavra, apresenta-o na tela. Senão, envia mensagem de erro |
| | 3 – Solicita: Iterage com cena 3D. |
| Restrições | Precisa existir o sinal da palavra. |

**Tabela 10 – Documentação do caso de uso INTERAGE COM A CENA 3D.**

| Nome do caso de uso | ITERAGE COM A CENA 3D |
|---|---|
| Caso de uso geral | |
| Ator principal | USUÁRIO |
| Atores secundários | ERLENEN |
| Resumo | Este caso de uso descreve as etapas percorridas pelo usuário e pelo sistema ERLENEN para solicitar a visualização do sinal correspondente a palavra em LIBRAS. |
| Pré-condições | Sinal 3D da palavra deve estar na tela do dispositivo |
| Pós-condições | |
| Prioridade: | __Essencial X_ Importante __ Desejável |
| Ações do ator principal | Ações do sistema |
| 1 – Informa solicitação de iteração com o sinal 3D | |
| 2 – Informa comando de iteração. | |
| | 3 – Enquanto recebe comando, execute-o |
| | 4 – Aguarda comando de saída. |
| Restrições | |

A construção deste software, sem a utilização de nenhuma ferramenta ou observação sobre as cenas a serem carregadas é bastante complicada. Para cada cena, é necessária a construção de uma classe canvas, com as particularidades de cada uma. A construção de uma interface atraente e amigável, a partir dos recursos únicos do J2ME também resulta em muitas linhas de código. A Figura 49 mostra a interface desenvolvida nesta parte do estudo de caso.


**Figura 49 - ERLENEN construído com J2ME puro.**

A execução deste aplicativo é restrita ao simulador do NetBeans. Nenhuma característica de portabilidade foi considerada, apenas seguiu-se as instruções de programação dos tutoriais presentes em Muchow (2004).

Nas tentativas de execução em diferentes simuladores, perdeu-se características como tamanho das figuras (ficaram distorcidas em diferentes tamanhos e resoluções de telas), movimento das cenas, interação com as mesmas ou mesmo sua visualização.

Uma solução para um destes problemas é a utilização da biblioteca Scene3D, que diminui consideravelmente o tamanho do código, uma vez que a chamada das cenas é feita com apenas uma linha de código. Porém, deseja-se a portabilidade do aplicativo, além de uma interface mais atraente. Este é o propósito do *framework* LWUIT.

## 6.2. Utilização da biblioteca Scene3D juntamente com LWUIT

A utilização da primeira classe desenvolvida (Scene3D) em conjunto com o *framework* LWUIT é conseguida sem nenhuma alteração de programação, uma vez que

há o suporte ao M3G neste *framework*. O que acontece é que a integração não é total, uma vez que as cenas 3D são objetos instanciados fora dos *conteiners* desenvolvidos no *framework*. Isto faz com que os efeitos de transição das configurações de *layout* não funcionem adequadamente quando a cena está sendo chamada e/ou utilizada.

Neste caso, há ainda a entrada da cena na tela do dispositivo portátil que não atende as configurações de animação 3D que tanto apelo gráfico traz ao pacote LWUIT. O efeito de transição é perdido ao se chamar uma cena M3G com a utilização da classe *Scene3D*. A Figura 50 mostra o efeito de transição de cubo, presente no *framework* LWUIT, e mostra também que a chamada de uma cena M3G, utilizando a classe *Scene3D*, não obedece a este mesmo efeito, pois não está dentro do contêiner padrão do LWUIT.

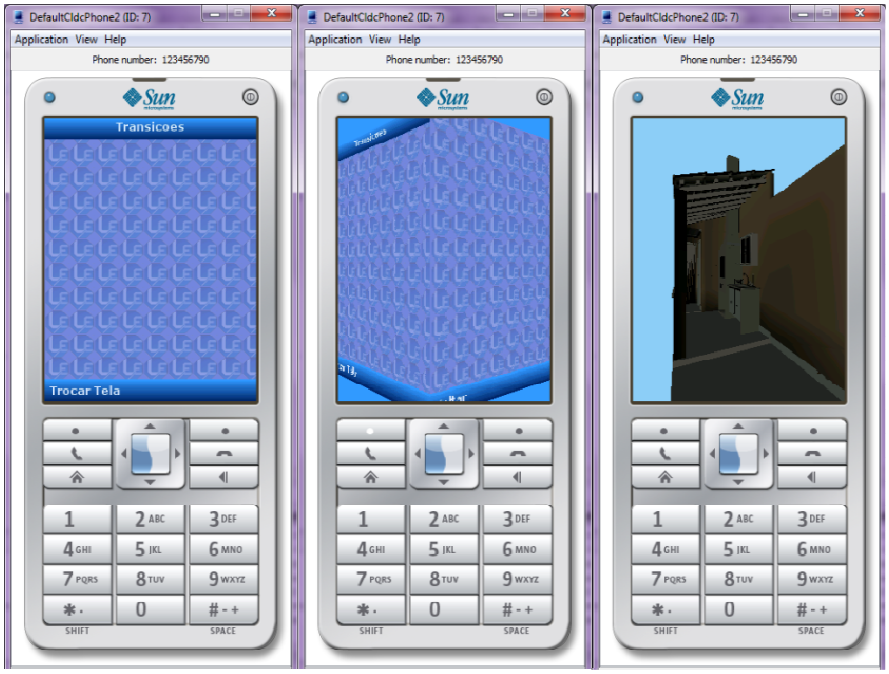



**Figura 50 - Efeito de transição "cubo" do framework LWUIT perdido ao chamar uma cena M3G.**

Outra opção existente é utilizar a classe *Scene3D* dentro de um contêiner (no caso um formulário), e obter, assim, a integração ao efeito de transição. Porém perde-se as implementações de navegação na imagem, uma vez que algumas das classes utilizadas em *Scene3D* foram modificadas na criação do *framework* LWUIT.

## 6.3. Desenvolvimento do software ERLENEN utilizando o framework LWUIT

A solução para o inconveniente apontado no item anterior é a utilização da segunda biblioteca desenvolvida e compilada no *framework* RV_LWUIT.

Desta forma, o *menu* inicial consta de uma tela de apresentação do *software* (Figura 51a, 51b), uma tela para digitar a palavra a ser aprendida (Figura 51c), um ambiente provido pelo próprio *framework* LWUIT para digitação de texto (Figura 51d) onde há opção de troca de teclado e OK, para indicar término da digitação (Figura 51e).

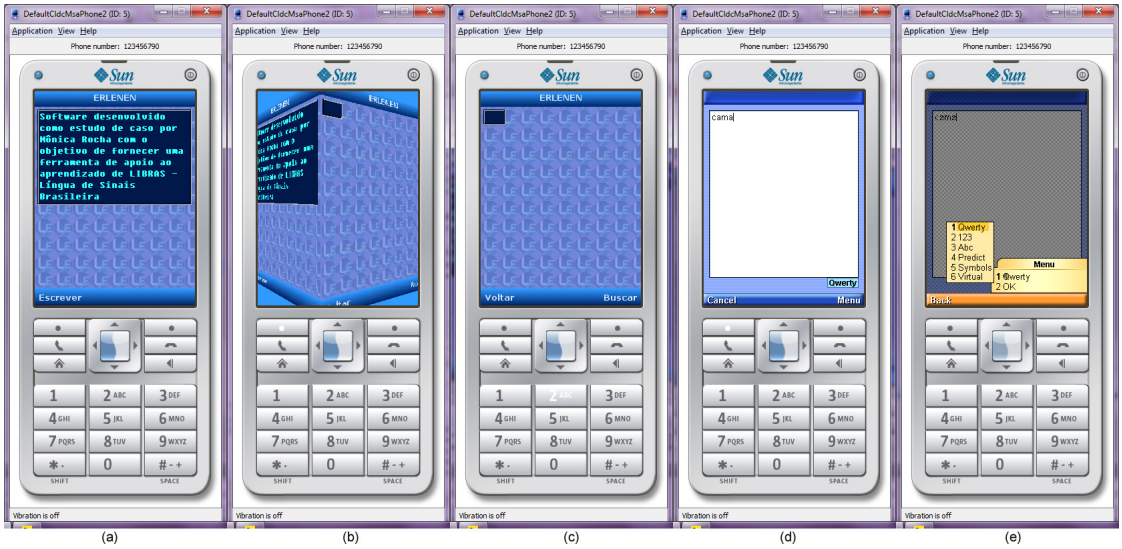



**Figura 51 – Telas iniciais do aplicativo ERLENEN.**

Em seguida, são apresentados, o conceito em português (texto), em 2D os sinais das letras que formam a palavra digitada (soletração em sinais LIBRAS), e, em 3D, o sinal LIBRAS para a palavra (se houver) em questão. Todas estas etapas são visualizadas na Figura 51. Na visualização do sinal, há a possibilidade de navegabilidade pela cena, uma vez que pode ser necessário modificar o ângulo para perceber algum detalhe do sinal.

Na Figura 52 pode-se ver a entrada na tela do dispositivo móvel das letras que compõem a palavra utilizada como exemplo. E na Figura 53, a navegação no sinal LIBRAS que representa a palavra exemplo. Para navegação, utiliza-se os botões direita/esquerda acima/abaixo do dispositivo móvel ou, eventualmente, o *joystick* ou a caneta em telas *ToutchScreen.*

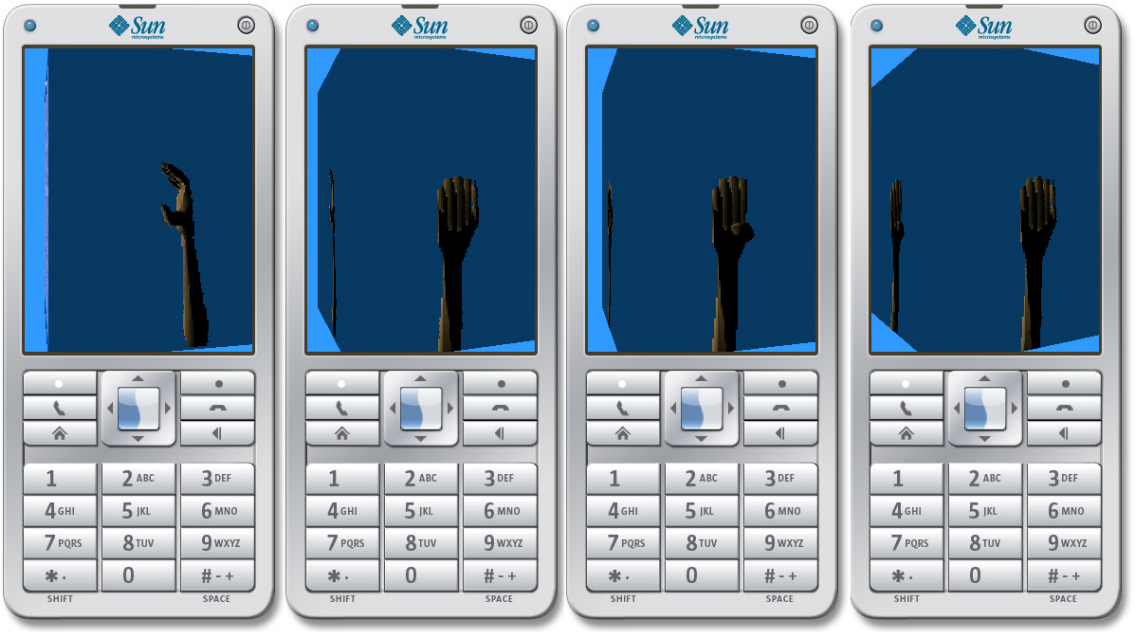


**Figura 52 - Telas de soletração no aplicativo ERLENEN para um exemplo.**

**Figura 53 - Telas de navegação no aplicativo ERLENEN para um exemplo.**

## 6.4. Criação de um programa para visualização de cenas M3G utilizando Scene3D

Para testar a funcionalidade da classe *Scene3D* desenvolvida na primeira parte deste trabalho, desenvolveu-se um programa, M3GViewer, com o objetivo de visualizar cenas 3D em dispositivos portáteis. O programa considera que as cenas serão desenvolvidas utilizando um *software* de modelagem 3D presente no mercado, como 3DsMax, Maya, etc., e estas cenas serão colocadas no dispositivo portátil através de um mecanismo qualquer de inserção de dados: cartão de memória, cabo USB, etc. Uma vez executado o M3GViewer, este buscará nos dispositivos de armazenamento presentes, os arquivos com extensão .M3G, permitindo a seleção de um deles, renderizando-o na tela do mesmo.

Neste desenvolvimento foram utilizados outros recursos da linguagem de programação J2ME, como a JRS154 (arquivos) e o *framework Floggy* que fornece persistência de dados a aplicativos *mobile*. A execução permite a escolha entre abrir um arquivo ou visualizar uma demonstração de imagem M3G (no caso, foi utilizada a fornecida pela *Sun*). A escolha do comando Abrir leva a tela de escolha de diretórios do dispositivo portátil, e, caso escolha-se uma imagem M3G, novas telas de permissões serão abertas (recurso exigido pelo Java). Aceitando-se todas elas, a cena será então renderizada. A Figura 53 mostra telas do aplicativo, com as diversas exigências do J2ME. Quando o *software* é inicado, existe a possibilidade de visualizar uma cena demonstrativa clicando em DEMO (cena *pongoroo.m3g, disponível no Wireless Toolkit*) ou abrir um outro arquivo M3G ( ABRIR). Se a opção for a primeira, a cena é apresentada na tela e os recursos de navegação disponibilizados para o usuário. Caso a opção seja a segunda, o programa irá buscar no diretório raiz cenas M3G. É possível navegar nos diretórios do dispositivo, permitindo localizar o arquivo M3G desejado. Até este ponto, pode-se visualizar na Figura 53. O simulador não contém diretórios para ser selecionado, mas no dispositivo móvel, navega-se perfeitamente nos diretórios e é possível selecionar a cena. Selecionando-a, ela é renderizada na tela e os recursos de navegação disponibilizado para o usuário.


**Figura 53 - Aplicativo M3GViewer desenvolvido como estudo de caso.**

Não fossem todas as permissões exigidas pelo Java, este aplicativo se assemelharia ao *plugin* Cortona, utilizado para explorar cenas 3D *desktop* dentro de navegadores web.

## 6.5. Conclusões

Desenvolveu-se um programa, M3GViewer, com o objetivo de visualizar cenas 3D em dispositivos portáteis. O programa considera que as cenas serão desenvolvidas utilizando softwares de modelagem 3D presentes no mercado, como 3DsMax, Maya, etc., e estas cenas serão colocadas no dispositivo portátil através de um mecanismo qualquer de inserção de dados: cartão de memória, cabo USB, etc. Uma vez executado o M3GViewer, este buscará nos dispositivos de armazenamento presentes, os arquivos com extensão .M3G, permitindo a seleção de um deles, renderizando-o na tela do mesmo. Foram utilizados outros recursos da linguagem de programação J2ME, como a JRS154 (arquivos) e o *framework* Floggy que fornece persistência de dados a aplicativos *mobile*. A execução permite a escolha entre abrir um arquivo ou visualizar uma demonstração de cena M3G (no caso, foi utilizada a fornecida pela Sun e diversas outras criadas no Laboratório de Computação Gráfica da UFU).

Não fossem as restrições de abertura de arquivos impostas pela linguagem J2ME, ter-se-ia um aplicativo similar ao Cortona: clicando sobre o nome do arquivo que contém a cena 3D ela seria aberta. As restrições do Java impõem perguntas sobre a

abertura do arquivo, necessitando aceitá-las para que o mesmo ocorra. Contornar estas restrições é sugestão para trabalhos futuros.

O desenvolvimento de aplicativos para usuários com necessidades especiais, em especial deficientes auditivos, também tem sido foco de diversas pesquisas, incluindo inclusive dispositivos móveis. Existem diversos projetos que exemplificam isto, como o projeto Rybená (Rybená 2005), o projeto Falibras (Coradine 2005), e o projeto SYNFACE (Sheard 2003; Synface 2007), entre outros.

Quando se pretendem apresentar e ensinar conceitos que são inerentemente espaciais e visuais, como no caso da Língua Brasileira de Sinais, atender a este requisito torna-se essencial. Os requisitos de mobilidade, portabilidade e facilidade de uso não podem ser esquecidos, e, para atender a todas estas necessidades, o reaproveitamento de recursos de programação é indispensável. Neste contexto, a utilização de frameworks como o LWUIT é uma importantíssima ferramenta de programação, provendo recursos de interface aos aplicativos e dando suporte a utilização de cenas 3D, no formato M3G.

O aplicativo ERLENEN atende aos requisitos de mobilidade, portabilidade e facilidade de uso, todos estes atendidos perfeitamente com a utilização do *framework* LWUIT. Entretanto o mesmo atende também ao requisito de visualização e navegabilidade nos sinais LIBRAS, que são inerentemente espaciais (3D), pois utiliza o *framework* LWUIT com a alteração proposta no trabalho: RV_LWUIT.

Todos os testes de emulação e simulação atenderam perfeitamente ao requisito de portabilidade, tanto no M3GViewer quanto no ERLENEN.

Visualizar imagens 3D em dispositivos móveis não foi, até hoje, uma tarefa trivial. Porém, com as alterações propostas neste trabalho, torna-se bastante simples inserir cenas 3D em aplicações para dispositivos portáteis.

# 7. Conclusões e trabalhos futuros

E que a minha loucura seja perdoada... Porque metade de mim é amor.
E a outra metade... Também.
*Oswaldo Montenegro

Este capítulo apresenta as conclusões deste trabalho baseado na pesquisa realizada. Além disso, um conjunto de sugestões de pesquisa é apontado como objetos de estudo para o futuro.

## 7.1 Conclusões

Como mencionado anteriormente, este trabalho foi motivado devido à necessidade de possibilitar o uso de aplicativos de Realidade Virtual para dispositivos móveis de forma direta (apenas com a chamada da cena no programa). Devido à identificação de uma falta de padronização nos sistemas de suporte à dispositivos móveis, definiu-se pelo desenvolvimento de um conjunto de bibliotecas que suportassem esta padronização (J2ME). Dentro deste, o *framework* LWUIT foi escolhido como referência para a implementação destas bibliotecas por motivos já justificados previamente. Finalmente, após um conjunto de testes sobre as bibliotecas desenvolvidas e seu acoplamento ao *framework* mencionado, esta pesquisa conduziu para as seguintes conclusões:

a. Cada vez mais o Java vem tornando um padrão de programação. O J2ME é, atualmente, aceito em todos os tipos de dispositivos portáteis.
b. De uma forma geral, a usabilidade de interfaces de programas para dispositivos portáteis vem sendo alvo de pesquisas;
c. A Realidade Virtual fornece uma metáfora do mundo real, usada em projeto de interfaces para estes dispositivos;
d. A linguagem de programação J2ME fornece uma padronização (M3G) para construção de cenas 3D, e esta já está integrada aos softwares gráficos como o 3DsMax, Maya, entre outros, na forma de um *plugin* de exportação dos arquivos;
e. Visualizar estas cenas em dispositivos móveis não é trivial, exige programação da interface de baixo nível (canvas) do J2ME;
f. É possível encapsular os detalhes de implementação de ações e movimentos para cenas 3D numa biblioteca (Scene3D), no formato M3G (padrão para o J2ME), de forma a atender uma grande gama de cenas, desde que respeitadas às restrições citadas no capítulo 5;
g. A biblioteca desenvolvida não pode ser utilizado juntamente com o *framework* LWUIT, porque este possui classes próprias para M3G (que foram redefinidas na construção do *framework*). Portanto, foi necessário modificar a biblioteca *Scene3D* para adequar-se as necessidades do *framework* LWUIT.
h. Foi possível adequar a biblioteca ao LWUIT, recompilando-o para RV_LWUIT. Desta forma, foi possível integrar de modo direto, cenas 3D ao *framework*, melhorando-o e colocando mais uma possibilidade de transformá-lo numa metáfora do mundo real.
i. Manteve-se o padrão de projeto de *software* original do *framework* LWUIT, utilizando o padrão de projetos *Command,* que permitiu o encapsulamento da funcionalidade desejada em um objeto reutilizável.
j. É possível utilizar ambas as bibliotecas (Scene3D e RV_LWUIT) para a construção de quaisquer aplicativos, incluindo jogos, atentando para que os recursos de navegação implementados são apenas a navegação à direita /esquerda e acima/abaixo. Neste caso, o motor do jogo necessitaria de ser desenvolvido, observando a integração ao *framework* LWUIT.

## 7.2 Trabalhos Futuros

Este trabalho iniciou uma pesquisa na intersecção das áreas de Programação para Dispositivos Portáteis, Padrão de Projetos e Realidade Virtual. Porém, várias sugestões de trabalhos futuros podem ser percebidas ao longo do texto e citas a seguir:

a. Desenvolver recursos no framework RV_LWUIT para cenas animadas, estudando e encapsulando as características de "menor demoninador comum" presente nas animações;

b. Elaborar um estudo de forma a "driblar" as requisições de confirmações de arquivos presente na plataforma Java (J2ME), tornando o software M3GViewer um programa apto a se tornar a interface padrão para abertura de cenas M3G em dispositivos portáteis;

c. Testes de usabilidade e de aplicabilidade com usuários surdo-mudos são sugestões para trabalhos futuros.

d. Utilizar os recursos dos dispositivos portáteis de reconhecimento de voz, em especial os telefones celulares, para integrar a tradução de voz contínua de português para LIBRAS, estendendo as funcionalidades do software ERLENEN;

e. Realizar um estudo visando o acoplamento ou a extensão das bibliotecas visando aplicativos com Realidade Aumentada.

f. Elaborar um estudo da aplicabilidade do framework RV_LWUIT aplicado a TV digital, uma vez que a plataforma LWUIT já está integrada à mesma.

## 7.3 Considerações Finais

Este trabalho foi desenvolvido sobre a plataforma J2ME, configuração CLDC 1.1, perfil MIDP 2.0. Foram utilizados os simuladores disponíveis no pacote NetBeans, acrescido de emuladores NOKIA S60, NOKIA E71, Motorola A5 e A6, Siemens E80

disponível no SMTK CORE PACK V2. Todos os testes foram realizados utilizando um computador Sony com processador Intel Core 2 Duo, 200GHz, Sistema Operacional Windows 7 (64 bits), 4 GB de RAM. Foi ainda utilizado o *Wireless Toolkit* 2.5.2.

Somente são contempladas neste trabalho cenas consideradas simples (aquelas que atendem as restrições do item 5.1 (b)). Cenas complexas não obedecem a especificação JSR 184, portanto, não podem ser visualizadas utilizando classes definidas nesta especificação e, obviamente, também não o podem com as classes desenvolvidas neste trabalho.

Mesmo restringindo ao "menor denominador comum" a necessidades das classes envolvidas, portabilidade em dispositivos móveis ainda não é uma realidade. A autora acredita que as padronizações e necessidades de fabricantes e usuários ainda levará a isso.

Sem considerar a curva de aprendizado do programador, tanto da especificação JSR 184 como das classes do framework LWUIT, este trabalho contribui com uma redução de aproximadamente 200 linhas de código quando da utilização de Scene3D e de aproximadamente 400 linhas de código quando da utilização do framework LWUIT.

As contribuições deste trabalho são:

iii. Encapsulamento das características de RV para o padrão M3G (JRS 184); e

iv. Extensão do framework LWUIT com as características de RV mantendo o padrão de portabilidade do mesmo;

# Referências bibliográficas


Abreu, L. M. d. (2005). Usabilidade de Telefones Celulares com base em Critérios Ergonômicos. Programa de Pós- Graduação em Design do Departamento de Artes & Design da PUC-Rio. Rio de Janeiro, Pontífica Universidade Católica. **mestre**.

Agrawal, A. K. (2004). "Building Applications with .NET Compact Framework " Retrieved 17 de março, 2008, from http://www.c-sharpcorner.com/UploadFile/amit_agrl/BuildingApplicationswith.NETCompactFramework12022005022332AM/BuildingApplicationswith.NETCompactFramework.aspx.

Android, D. (2010). "Application Fundamentals." Retrieved 08/02-2010, 2010, from http://developer.android.com/guide/topics/fundamentals.html.

Araiz, V. M. C. (2008). "Java4ever." Retrieved 25/08, 2009, from http://www.java4ever.com/index.php?section=j2me&project=apime&menu=main&lang=_en.

Baudisch, A. R. (2007). "Escolha da plataforma Mobile: Uma análise mais abrangente.

." Retrieved 04/2008, from http://desenvolvedormovel.auriummobile.com/index.php/2007/11/01/escolha-da-plataforma-mobile-uma-analise-mais-abrangente/.

Brew. (2010). "Get Authenticated." Retrieved 10/12, 2010, from https://developer.brewmp.com/go-to-market/get-authenticated.

Brew. (2010). "Languages." Retrieved 10/12, 2010, from https://developer.brewmp.com/resources/family/languages.

Brew. (2010). "Testing with the Stability Application Legacy." Retrieved 10/12, 2010, from https://developer.brewmp.com/resources/how-to/testing-stability-application-legacy.

Campos, P. F. N., Nuno J. (2006). Interação Homem-Máquina – Guia dos Laboratórios. Madeira, DEPARTAMENTO DE MATEMÁTICA E ENGENHARIAS - UNIVERSIDADE DA MADEIRA.

Caraciolo, M. (2009). "LWUIT: Framewortk Gráfico para Java ME." Retrieved 12/11, 2009, from http://www.mobideia.com/2008/05/lwuit-framework-grfico-para-javame.html.

Christiansson, B. F., M.; Hagen, I. ; Hansson, K.; Jonasson, J. ; Jonasson, M.; Lott, F. ; Olsson, S. ; Rosevall, T. (2008). GoF Design Patterns - with examples using Java and UML2.

Coradine, L. C. (2005). "Interpretação de Pequenas Frases com Análise Léxica no Sistema Falibras: Tradutor do Português para a LIBRAS S." Retrieved 26/05/2008, 2008, from http://www.niee.ufrgs.br/cbcomp/cbcomp2004/html/pdf/Forum/t170100284_3_1.pdf.

Deitel, H. M. D., P. J., Ed. (2005). Java - Como Programar, Pearson Prentice Hall.

devMedia. (2007). "Interface Gráfica para aplicações J2ME." Interface Gráfica para aplicações J2ME, 5, from http://www.devmedia.com.br/articles/viewcomp.asp?comp=2775.

EnoughSoftware. (2007). "J2ME Polish." Retrieved 28/08, 2009, from http://www.j2mepolish.org/cms/topsection/home.html.

Filho, E. M. d. B. (2005). VirTraM: Um Framework para o Desenvolvimento de Treinamentos Utilizando Realidade Virtual em Dispositivos Móveis. Ciência da Computação. Fortaleza, Universidade Federal do Ceará. **Mestre**.

Garcia, F. L. S. (2001). "METODOLOGIA PARA CRIAÇÃO DE AMBIENTES VIRTUAIS TRIDIMENSIONAIS." IV International Conference on Graphics Engineering for Arts and Design **1**.

Ito, G. C. (2008) "UMA ARQUITETURA PARA GERAÇÃO DE INTERFACES ADAPTATIVAS PARA DISPOSITIVOS MÓVEIS." INPE.

Júnior, E. M. (2009). Java ME Platform SDK 3 - O sucessor do Wireless Toolkit "Velho de Guerra". Java Magazine. Brasil, Dev Magazine. **69**.

Kari Pulli, T. A., Ville Miettinen, Jani Vaarala (2008). Mobile 3D Graphics with OpenGL and M3G.

Kirner, C. T., R. (2004). Realidade Virtual - Conceitos e Tendências.

Lais Zanfoli, R. C. F., Ricardo Ogliari, Robison Cris Brito (2009). "LWUIT - Introdução." Retrieved 25/11, 2009, from http://www.javamovel.com/2009/08/lwuit-introducao.html.

Matos, P. P., M. B. Carmo, et al. (2007). Visualização de Informação Georeferenciada em Dispositivos Móveis. Faculdade de Ciências da Universidade de Lisboa.

Microsystems, D. O.-W. O. S. S.-S. (2005). "Java Micro Edition." from http://images.dimas4u.multiply.com/attachment/0/RRF8tAoKCqsAAAYJw4E1/sc1-low-res.pdf?nmid=11616750.

Microsystems, S. (2007). "The Java ME GUI APIs at a Glance." Retrieved 14 de Fevereiro, 2008, from http://developers.sun.com/mobility/midp/articles/guiapis/.

Microsystems, S. (2009). Developer's Guide - Lightweight UI Toolkit. S. Microsystems.

MobileFish. (2008). "Export model to m3g file." Retrieved 19/04, 2009, from http://www.mobilefish.com/tutorials/3dsmax/3dsmax_quickguide_export_184.html.

Muchow, J. W. (2004). Core J2ME Tecnologia & MIDP. São Paulo, Pearson Education do Brasil.

Powers, M. (2005). "Getting Started with Mobile 2D Graphics for J2ME." Retrieved 03 de fevereiro, 2008, from http://developers.sun.com/mobility/midp/articles/s2dvg/index.html.

Prado, M. (2009, 08/04/2009). "Jikes: Java e iPhone de mãos dadas." Retrieved 18/12/2010, 2010, from http://imasters.com.br/artigo/12280/java/jikes_java_e_iphone_de_maos_dadas/.

Pressman, R. S., Ed. (1995). Engenharia de Software. São Paulo, Pearson Makron Books.

Pressman, R. S., Ed. (2006). Engenharia de Software, McGrawHill.

Process, J. C. (2006). "JSR 226: Scalable 2D Vector Graphics API for J2METM." Retrieved 02 de fevereiro, 2008, from http://www.jcp.org/en/jsr/detail?id=226.

Qualcomm. (2010). "Augmented Reality " Retrieved 10/12, 2010, from http://www.qualcomm.com/products_services/augmented_reality.html.

Rybená. (2005). "Projeto Rybená." Retrieved 26/05, 2008, from http://www.rybena.org.br/rybena/default/index.jsp.

Sampaio, R. (2010). "Java ME: Construindo Ótimas Interfaces com LWUIT." Retrieved 03/02, 2010, from http://javafree.uol.com.br/noticia/4359/Java-ME-Construindo-otimas-Interfaces-com-LWUIT.html.

Santos, R., S. Freitas, et al. (2009). "Usabilidade de telefones celulares - um estudo com consumidores das classes C e D." Retrieved 05/11/2010, 2010, from http://www.slideshare.net/robsonsantos/usabilidade-de-telefones-celulares.

Schaefer, C. (2004) "PROTÓTIPO DE APLICATIVO PARA TRANSMISSÃO DE DADOS A PARTIR DE DISPOSITIVOS MÓVEIS APLICADO A UMA EMPRESA DE TRANSPORTES."

Sheard, M. T., N (2003). "The SYNFACE project: development and evaluation of a talking face telephone." The HCI 2003 - Designing for Society Conference in Bath, England.

Sourceforge.net. (2007). "MicroWindowToolkit - The open source framework for developing user interfaces in J2ME " Retrieved 25/05, 2009, from http://j2me-mwt.sourceforge.net/index.html.

Sun. (2008). "Sun Java Wireless Toolkit for CLDC." from http://java.sun.com/products/sjwtoolkit/download.html.

Synface. (2007). "Synface." from http://www.speech.kth.se/synface/.

Teixeira, J. C. e. C. (2005). "J2ME." Retrieved 22 de novembro, 2007, from http://www.portaljava.com/home/modules.php?name=Content&pa=showpage&pid=24.

Virkus, R. (2005). Pro J2ME Polish: Open Source Wireless Java Tools Suite. Apress. **1**.

W3C. (2003). "Scalable Vector Graphics (SVG) 1.1 Specification." Retrieved 6 de março, 2008, from http://www.w3.org/TR/SVG/.

Wikipédia. (2010). "Android." Retrieved 10/12, 2010, from http://pt.wikipedia.org/wiki/Android.

World/EUA, N. (2010) "iPhone é o preferido entre compradores de smartphone nos EUA, diz pesquisa." MacWorld.