PAULO ALEXANDRE LOBATO

2DMG — UMA ARQUITETURA DE CLASSES DE MIDDLEWARE PARA A PORTABILIDADE DE APLICATIVOS GRÁFICOS 2D EM DISPOSITIVOS MÓVEIS

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

VIÇOSA
MINAS GERAIS - BRASIL
2011

Ficha catalográfica preparada pela Seção de Catalogação e Classificação da Biblioteca Central da UFV

T
L796d
2011

Lobato, Paulo Alexandre, 1980-
2DMG – uma arquitetura de classes de middleware para a portabilidade de aplicativos gráficos 2D em dispositivos móveis / Paulo Alexandre Lobato. – Viçosa, MG, 2011.
xiv, 58f. : il. (algumas col.) ; 29cm.

Inclui apêndice.
Orientador: Mauro Nacif Rocha.
Dissertação (mestrado) - Universidade Federal de Viçosa.
Referências bibliográficas: f. 55-58.

1. Computação. 2. Computação móvel. 3. Computação gráfica. 4. Middleware. I. Universidade Federal de Viçosa. II. Título.

CDD 22. ed. 004

PAULO ALEXANDRE LOBATO

2DMG – UMA ARQUITETURA DE CLASSES DE MIDDLEWARE PARA A PORTABILIDADE DE APLICATIVOS GRÁFICOS 2D EM DISPOSITIVOS MÓVEIS

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

APROVADA: 28 de julho de 2011.

José Luis Braga
(Co-Orientador)

Zenilton Kleber G. do P. Júnior

Mauro Nacif Rocha
(Orientador)

Dedico esta dissertação aos meus pais
Paulo Célio e Glória

À minha querida esposa
Fernanda

Ao meu filho
Lucas

E à minha irmã
Polyanna

AGRADECIMENTOS

Em primeiro lugar agradeço a Deus, que me deu força e coragem para realizar este trabalho. Sou grato por sempre colocar pessoas certas em meu caminho, que me auxiliaram não só nesta dissertação como em toda a vida. “Sabemos que Deus age em todas as coisas para o bem daqueles que o amam, dos que foram chamados de acordo com o seu propósito.” (Romanos 8.28, Bíblia, versão NVI).

Aos meus pais, Paulo Célio e Glória, pelo apoio incondicional e presença constante, seja por telefone, Internet ou nas indas e vindas entre Guaçuí-ES e Viçosa-MG. Agradeço por sempre acreditarem em mim; sem vocês nada disso seria possível. Obrigado por toda a educação que recebi e pela oportunidade que me deram de realizar meus sonhos.

À minha irmã e meu cunhado, Poly e Vini, que mesmo de longe me apoiam e estão sempre prontos a ajudar. Obrigado irmã por você existir na minha vida e por todo o carinho que tem por mim. Sou desafiado por você a ser uma pessoa melhor e a correr atrás dos meus sonhos.

Agradeço à minha querida e amada esposa Fernanda, ao meu sogro e sogra, Alberto e Marlúcia, e à minha cunhada Tatiana, por sempre se preocuparem comigo e acreditarem em mim. Em especial, à Fernanda por fazer parte deste sonho e pelos momentos que dividimos. Meu amor, muito obrigado por realizar um outro sonho ao final do mestrado: o de ser pai. Lucas, meu filho querido, papai agradece por você não chorar muito e nos deixar dormir durante a noite.

Agradeço ao Professor Mauro Nacif Rocha pelo companheirismo, auxílio em detalhes que extrapolavam sua obrigação e por acreditar em mim. Você, além de orientador, é também um amigo com quem pude compartilhar minha vida pessoal e sempre soube separar os assuntos. Agradeço também aos meus co-orientadores, Professores José Luis Braga e Marcus Vinícius Alvim Andrade, pelos conselhos e opiniões sobre o trabalho. Em especial ao José Luis, ou simplesmente Zé, por ser amigo e conselheiro desde a graduação.

Aos colegas do mestrado e graduação, que me incentivaram, ajudaram e contribuíram, direta ou indiretamente, para a realização deste trabalho. De forma especial aos amigos Bruno Vaz de Mello Rodrigues, Elizio Rezende e Luiz Gustavo Palla Montanha, companheiros de trabalhos e de estudos durante a graduação. Formamos o grupo mais famoso do DPI, "Os Manés".

Meu agradecimento aos alunos de graduação, Marcelo Ferreira Bottoni e Yugo Sakamoto, que tiveram papéis importantes no desenvolvimento do projeto. A todos do corpo docente do Departamento de Informática da Universidade Federal de Viçosa pela formação e conhecimento recebidos. Ensinamentos não só para o crescimento profissional, como para a vida toda. Muito obrigado UFV!

Por fim, mas não menos importante, o meu agradecimento à Igreja Presbiteriana de Viçosa (IPV) por ser um porto seguro. Local onde fiz grandes amizades e onde pude moldar o meu caráter durante a graduação e o mestrado. Agradeço também à Editora Ultimato e a tantos outros companheiros de caminhada.

BIOGRAFIA

PAULO ALEXANDRE LOBATO, filho de Paulo Célio Lobato e Maria da Glória Fonseca Lobato, brasileiro nascido em 09 de abril de 1980 na cidade de Guaçuí, no estado do Espírito Santo.

No ano de 2001 ingressou no curso de graduação em Engenharia Elétrica na Universidade Federal de Viçosa (UFV). Em 2004 foi aprovado na prova de transferência para o curso de Ciência da Computação, concluindo-o no ano de 2007.

Trabalhou como Analista de Sistemas na Fundação Arthur Bernardes (Funarbe), da UFV, durante 2007.

Em 2008 foi aprovado na seleção do programa de pós-graduação do Departamento de Informática (DPI), onde, em agosto de 2008, iniciou o mestrado em Ciência da Computação, defendendo sua dissertação em julho de 2011.

Trabalhou como Analista de Sistemas Web na Editora Ultimato de agosto de 2008 a dezembro de 2010, e como professor na União de Ensino Superior de Viçosa (Univiçosa), em 2009 e 2010.

Aprovado no concurso público federal em 2010, atualmente é Analista de Tecnologia da Informação na Universidade Federal do Espírito Santo (UFES), campi de Alegre-ES, desde janeiro de 2011.

SUMÁRIO

LISTA DE FIGURAS.....vii

LISTA DE TABELAS.....ix

LISTA DE QUADROS.....x

LISTA DE ABREVIATURAS E SIGLAS.....xii

RESUMO.....xiii

ABSTRACT.....xiv

1 INTRODUÇÃO.....1

2 PORTABILIDADE EM DISPOSITIVOS MÓVEIS.....6

3 2D MOBILE GRAPHICS (2DMG).....12

4 IMPLEMENTAÇÕES E DISCUSSÕES.....25

5 EXPERIMENTAÇÃO NO IOS.....40

6 CONCLUSÕES.....49

APÊNDICE A.....52

REFERÊNCIAS BIBLIOGRÁFICAS.....55

LISTA DE FIGURAS

Figura 2.1. Tela do PVANet sem adaptação (esquerda) e adaptado (direita) para uso em um dispositivo móvel.....7

Figura 2.2. Arquitetura de software móvel em alto nível.....9

Figura 3.1. Diagrama em alto nível mostrando a arquitetura do 2DMG dentro do contexto de desenvolvimento de software para dispositivos móveis.....13

Figura 3.2. Possível arquitetura em camadas de um sistema móvel, destacando o middleware 2DMG.....14

Figura 3.3. Divisão em camadas comparando a implementação em uma ou duas camadas.....15

Figura 3.4. Diagrama de classes que compõem esta primeira versão do 2DMG, destacando as duas classes principais.....16

Figura 3.5. Classe Fig com seus atributos e métodos.....17

Figura 3.6. Classe Line com seus atributos e métodos.....18

Figura 3.7. Classe Rectangle com seus atributos e métodos.....18

Figura 3.8. Classe Elipse com seus atributos e métodos.....19

Figura 3.9. Classe Circle com seus atributos e métodos.....19

Figura 3.10. Classe Polygon com seus atributos e métodos.....20

Figura 3.11. Classe Point em detalhe.....20

Figura 3.12. Classe Text com seus atributos e métodos.....20

Figura 3.13. Classe Composite com seus atributos e métodos.....21

Figura 3.14. Classe Color com seus atributos e métodos.....22

Figura 3.15. Classe Window com seus atributos e métodos.....23

Figura 3.16. Classe FigFactory com seus atributos e métodos.....24

Figura 4.1. Execução do código-teste nos emuladores da iOS SDK (esquerda) e da Symbian SDK (direita).....37

Figura 4.2. Representação gráfica da Tabela 4.1.....39

Figura 5.1. Aspecto da interface do AutoPerfil.....41

Figura 5.2. Aspecto da interface do software Desmet.....42

Figura 5.3. Simulador iOS SDK com a tela inicial do Desmet para o iPhone....47

Figura 5.4. Simulador iOS SDK mostrando a tela do perfil HEB com os dados de seus atributos.....48

LISTA DE TABELAS

Tabela 1.1. Participação dos sistemas operacionais no mercado, medido por acesso à Internet no mês de maio de 2011.....2

Tabela 4.1. Número de linhas de código utilizado para desenhar uma figura com cada linguagem nativa e com o 2DMG.....38

LISTA DE QUADROS

Quadro 4.1. Implementação do método draw() da classe Circle em C++ (Windows Mobile).....26

Quadro 4.2. Implementação do método draw() da classe Rectangle em C++ (Windows Mobile).....26

Quadro 4.3. Implementação do método draw() da classe Polygon em C++ (Windows Mobile).....27

Quadro 4.4. Implementação do método draw() da classe Circle em C++ (Symbian).....27

Quadro 4.5. Implementação do método draw() da classe Rectangle em C++ (Symbian).....28

Quadro 4.6. Implementação do método draw() da classe Polygon em C++ (Symbian).....28

Quadro 4.7. Implementação do método draw() da classe Circle em Java ME.. 29

Quadro 4.8. Implementação do método draw() da classe Rectangle em Java ME. 30

Quadro 4.9. Implementação do método draw() da classe Polygon em Java ME. 30

Quadro 4.10. Implementação do método draw() da classe Circle em Obj-C (iOS).....31

Quadro 4.11. Implementação do método draw() da classe Rectangle em Obj-C (iOS).....32

Quadro 4.12. Implementação do método draw() da classe Polygon em Obj-C (iOS).....32

Quadro 4.13. Trecho do código teste em C++ (Symbian).....34

Quadro 4.14. Trecho do código teste em Obj-C (iOS).....35

Quadro 7.1. Script que descreve a imagem do perfil apresentado na Figura 5.4. 52

Quadro 7.2. Arquivo de dados usado na Figura 5.4.....53

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CAD	Computer-Aided Design
GPS	Global Positioning System
IDE	Integrated Development Environment
IHC	Interface Homem-Computador
J2ME	Java Micro Edition
JSR	Java Specification Request
JVM	Java Virtual Machine
MIDP	Mobile Information Device Profile
OS	Operating System
PDA	Personal Digital Assistants
RGB	Red Green Blue
RTF	Rich Text Format
RUP	Rational Unified Process
SDK	Software Development Kit
SOA	Service-Oriented Architecture
SVG	Scalable Vector Graphics
UMPC	Ultra Mobile PC

RESUMO

LOBATO, Paulo Alexandre, M.Sc., Universidade Federal de Viçosa, julho de 2011. **2DMG – Uma arquitetura de classes de middleware para a portabilidade de aplicativos gráficos 2D em dispositivos móveis**. Orientador: Mauro Nacif Rocha. Co-Orientadores: José Luis Braga e Marcus Vinícius Alvim Andrade.

Desde o início do uso de recursos computacionais, pesquisadores, engenheiros e empresas têm se beneficiado de ferramentas científicas cada vez mais sofisticadas em relação à computação. Softwares de CAD (*computer-aided design*) e de cálculos estatísticos são uma pequena amostra desse vasto campo de aplicações. Os avanços recentes na área da Computação Móvel permitem ampliar consideravelmente a aplicabilidade das ferramentas computacionais, na medida em que estas podem ser carregadas em dispositivos menores e com acesso à rede sem fio. Embora o uso de dispositivos móveis já esteja bem consolidado para o acesso à Internet, a sistemas corporativos, à organização de dados pessoais e entretenimento, sua utilização ainda é pequena em aplicações técnicas e científicas. Isto se deve à falta de recursos de programação adequados, especialmente no tocante ao uso de métodos numéricos e computação gráfica, muito comuns nesse tipo de aplicação. Neste trabalho, apresenta-se uma arquitetura de classes como camadas de adaptação de software (*middleware*) que auxilia o desenvolvimento e a portabilidade de aplicações em diferentes plataformas, como iOS (iPhone/iPad/iPodTouch), Android, Symbian e Windows Mobile.

ABSTRACT

LOBATO, Paulo Alexandre, M.Sc., Universidade Federal de Viçosa, July, 2011. **2DMG – A middleware class architecture for 2D graphics application portability on mobile devices**. Adviser: Mauro Nacif Rocha. Co-Advisers: José Luis Braga and Marcus Vinícius Alvim Andrade.

Since the beginning of use of computing resources, researchers, engineers and companies have benefited from scientific tools increasingly sophisticated in relation to computing. Softwares CAD (computer-aided design) and statistical calculations are a small sample of this wide range of applications. Recent advances in the area of mobile computing allows us to extend considerably the applicability of computational tools, as these can be loaded onto smaller devices with access to a wireless network. Although the use of mobile devices is already well established for Internet access, business systems, organization of personal data and entertainment, their use is still small in technical and scientific applications. This may be due to a lack of appropriate programming resources, especially regarding the use of numerical methods and computer graphics, very common in this type of application. In this work, we present an architecture classe a software as adaptation layers (middleware) that supports the development and application portability across different platforms, such as iOS (iPhone/iPad/iPodTouch), Android, Symbian and Windows Mobile.

1 INTRODUÇÃO

De acordo com Loureiro e colaboradores (2003), “Computação móvel é um paradigma computacional que tem como objetivo prover ao usuário acesso permanente a uma rede fixa ou móvel independente de sua posição física. É a capacidade de acessar informações em qualquer lugar e a qualquer momento. Este paradigma também recebe o nome de computação ubíqua ou computação nômade”. Ainda segundo esses mesmos autores, o paradigma da Computação Móvel “...está mudando a forma como trabalhamos, comunicamos, divertimos, estudamos e fazemos outras atividades quando estamos em movimento ou não desejamos ficar (presos) a uma infraestrutura fixa de comunicação de dados... Computação móvel está se tornando uma área madura e parece destinada a se tornar o paradigma computacional dominante no futuro” (LOUREIRO et al., 2003).

De fato, vemos hoje que a evolução dos dispositivos portáteis como PDAs (Personal Digital Assistants), também chamados *handhelds*, UMPCs (Ultra Mobile PCs), smartphones e tablets, em conjunto com a evolução das redes de comunicação sem fio, têm possibilitado o uso de recursos computacionais e acesso a dados de uma maneira surpreendente. Além disso, esses dispositivos móveis estão incorporando muitas outras facilidades, funcionalidades e interfaces, como GPS (Global Positioning System), tocadores de áudio, câmeras fotográficas digitais, jogos eletrônicos cada vez mais sofisticados e placas de comunicação sem fio multiprotocolos.

Com todos esses recursos associados à evolução computacional dos dispositivos portáteis, em relação à capacidade de processamento e memória, foi mais do que necessário a criação de sistemas operacionais (SOs) específicos, como Symbian OS, iOS (anteriormente chamado de iPhone OS), Windows Mobile e Android. O mercado desses dispositivos está crescendo muito e atingindo diversas áreas como negócios, indústria, escolas, hospitais etc. Podemos ver na tabela abaixo (Tabela 1.1) a popularidade desses sistemas operacionais no acesso à Internet.

Destacamos o sistema operacional da Apple, que aparece no topo da lista dos sistemas para dispositivos móveis.

Tabela 1.1. Participação dos sistemas operacionais no mercado, medido por acesso à Internet no mês de maio de 2011.

Sistema Operacional	Market Share maio/2011
Windows	88,69%
Mac	5,32%
iOS (iPhone, iPad e iPod Touch)	2,38%
Java ME	1,15%
Linux	0,91%
Android	0,76%
Symbian	0,31%
BlackBerry	0,16%
Playstation	0,02%
Windows Mobile	0,02%

Fonte: Sítio http://marketshare.hitslink.com, acessado em junho de 2011

A grande quantidade de SOs e linguagens de programação diferentes existentes hoje para o desenvolvimento de aplicações em dispositivos móveis dificulta consideravelmente a portabilidade do software entre dispositivos diferentes. Uma das formas de facilitar essa portabilidade e, portanto, reduzir o custo de desenvolvimento do software é através do uso de camadas intermediárias de programação (*middleware*).

1.1 Motivação

Juntamente com a evolução dos dispositivos móveis, algumas ferramentas relevantes têm sido colocadas à disposição dos desenvolvedores, algumas destas focando principalmente na produção de software de entretenimento (CALLOW et al., 2007). Especialmente relevante é o uso da biblioteca OpenGL (*Open Graphics Library*) em dispositivos móveis e sistemas embarcados (PULLI et al., 2007). Tal biblioteca, chamada OpenGL ES (OpenGL for *Embedded Systems*), de rotinas de programação de baixo nível, é comumente usada em aplicações contendo gráficos e animações em 3D (CHITTARO, 2006). Embora ela permita também a construção de aplicações contendo recursos gráficos em 2D, obtendo uma maior portabilidade entre diferentes plataformas, seu uso não é trivial e demanda um esforço considerável por parte do programador.

Sendo assim, é comum a construção de camadas de adaptação (*middleware*) com o objetivo de abstrair os detalhes de suas sub-rotinas. O uso desse conjunto

(biblioteca gráfica padrão mais camada de adaptação) permite separar melhor a camada de apresentação gráfica dentro do software, de modo que o mesmo possa ser mais facilmente adaptado para diferentes sistemas como Windows Mobile, iOS ou Android.

De modo semelhante ao OpenGL, a linguagem Java ME (Java Platform, Micro Edition) oferece acesso a recursos gráficos dos dispositivos e possui também uma boa portabilidade entre os diversos sistemas operacionais. Nesta linguagem destaca-se a Mobile 3D Graphics API, conhecida como M3G, uma API (*Application Programming Interface*) que estende as capacidades do Java ME em relação à computação gráfica 3D. Ela foi especificada no âmbito do *Java Community Process* como JSR 184, em meados de 2005, e sua versão atual é a 1.1. Mas está em desenvolvimento o M3G 2.0 sob a JSR 297 (PULLI et al., 2008).

Apesar de existirem arquiteturas de *middleware* para a implementação de gráficos 3D em dispositivos móveis, como M3G, elas são voltadas especialmente para o desenvolvimento de jogos e animações gráficas (MOSMONDOR et al., 2006; CALLOW et al., 2007). Embora essas arquiteturas possam ser usadas para facilitar também a portabilidade de aplicações que usam gráficos 2D, a complexidade das mesmas torna o trabalho inicial de projeto e implementação muito dispendioso. Várias dessas ferramentas, como o OpenGL ES, nem possuem rotinas 2D simples, por exemplo, para o desenho de retângulos ou círculos, já que a renderização de gráficos em jogos é composta basicamente pela composição de pontos, linhas retas, triângulos e texturas.

Quanto à implementação de gráficos 2D em dispositivos móveis, ainda não existem arquiteturas específicas. A elaboração de uma arquitetura com uma interface de classes única para a implementação de gráficos 2D irá, portanto, facilitar sobremaneira a portabilidade de aplicações ou partes de aplicações que usam esses recursos, em especial os científicos, de engenharia ou que utilizam recursos estatísticos.

1.2 Objetivos

O objetivo geral do projeto é construir uma arquitetura de classes como camada de adaptação (*middleware*) que auxilie o desenvolvimento de aplicações que utilizam gráficos 2D (aplicações científicas, visualização de mapas e dados estatísticos,

projetos de engenharia, entre outros) em dispositivos móveis, em particular PDAs e smartphones genéricos.

Os objetivos específicos deste trabalho são:

- Desenvolver uma arquitetura de classes como camada de adaptação (*middleware*) que auxilie o desenvolvimento de aplicações que utilizam gráficos 2D.
- Efetuar a implementação do *middleware* para as plataformas de desenvolvimento iOS (iPhone/iPad/iPod Touch), Java ME, Windows Mobile, Symbian e Android.
- Registrar o projeto e hospedá-lo em repositórios como GitHub (http://github.com/), SourceForge (http://sourceforge.net) ou Google Code (http://code.google.com), para que dessa forma ele seja criticado e melhorado constantemente por outros desenvolvedores.
- Como estudo de caso, utilizar o 2DMG para efetuar a adaptação de partes da ferramenta computacional Desmet, desenvolvida no Laboratório de Computação Aplicada à Engenharia Estrutural (LCEE) do Departamento de Engenharia Civil da UFV (DEC-UFV) (VERÍSSIMO & PAES, 2000), permitindo a sua execução no sistema Apple iOS (iPhone/iPad/iPod).

1.3 Organização deste documento

Os resultados alcançados neste trabalho estão fortemente embasados em dados e informações publicadas na literatura. Para tanto, foi realizada uma extensa pesquisa bibliográfica buscando trabalhos científicos que estivessem relacionados ao contexto do problema e da solução proposta neste projeto. O Capítulo 2 apresenta o problema da portabilidade gráfica em dispositivos móveis, descreve algumas alternativas existentes e discute suas dificuldades em atender a hipótese desta dissertação. São apresentados também trabalhos correlatos a este, que abordam a portabilidade de aplicativos em dispositivos móveis.

O Capítulo 3 descreve a solução proposta e como foi obtida. Apresenta os padrões de projetos utilizados e as classes que compõem o *middleware*. Além disso, aborda suas respectivas implementações e como utilizá-las.

No Capítulo 4 são apresentados os resultados da avaliação do *middleware* como solução para o problema descrito no Capítulo 2. Compara-se as suas

respectivas implementações nas diferentes plataformas e aborda a diferença entre o uso da linguagem nativa e a arquitetura de classes presente no 2DMG.

O Capítulo 5 relata a experimentação da solução proposta na adaptação de partes do aplicativo Desmet, ferramenta computacional desktop desenvolvida no LCEE do Departamento de Engenharia Civil da UFV, permitindo sua execução no sistema operacional iOS, presente nos dispositivos iPhone, iPad e iPod Touch da Apple.

Finalmente, o Capítulo 6 expõe as conclusões obtidas e as principais contribuições disponibilizadas por este projeto, destacando a arquitetura de classes como camada de adaptação e suas respectivas implementações. São disponibilizadas também sugestões para trabalhos futuros, ressaltando a possibilidade de novas classes que auxiliem ainda mais na portabilidade de gráficos 2D em dispositivos móveis e a implementação em outras plataformas de desenvolvimento, a partir da arquitetura proposta.

Ao final do documento são apresentadas as referências utilizadas para o embasamento teórico desta dissertação.

2 PORTABILIDADE EM DISPOSITIVOS MÓVEIS

2.1 Projetando *software* para dispositivos móveis

Um erro bastante comum no projeto de *software* para dispositivos móveis é a tentativa de apenas simplificar e "encolher" as telas da interface do software de modo que caibam nas pequenas telas desses dispositivos (JONES & MARSDEN, 2006). Para tornar tais aplicativos de fato fáceis e agradáveis de serem usados, é necessário re-projetar suas interfaces, dadas as grandes diferenças existentes não só nas limitações físicas do dispositivo, como também no contexto em que os mesmos são utilizados (SAFFER, 2007).

Embora no projeto de "aplicações web", seja possível realizar boas adaptações seguindo práticas recomendadas de implementação de interfaces (NOGUEIRA et al., 2008), em casos mais complexos, o emprego de técnicas mais avançadas de Inteligência Artificial (IA) permite que outras informações relevantes sejam usadas, tais como o perfil do usuário, a localização em que o mesmo se encontra e a forma como ele costuma usar o dispositivo (MENDES et al., 2008). O contexto nesses casos pode ser, portanto, bastante dinâmico, variando no tempo e no espaço, e sua resolução e percepção são importantes não apenas no campo da computação ubíqua, mas também em áreas como colaboração, assistentes pessoais e em computação no modelo *peer-to-peer*. Em muitas situações, a detecção desse contexto permite que os dispositivos móveis sejam capazes de inferir as necessidades de informação e serviços de um usuário e prestar o serviço de forma mais eficiente.

Dentro do Departamento de Informática da UFV (DPI-UFV), têm ocorrido experiências de sucesso na pesquisa de soluções envolvendo Computação Móvel, em especial na área de Bancos de Dados Geográficos (MARTINS, LISBOA FILHO, 2005; MONTEIRO, 2007). Experiências mais recentes envolvem a adaptação do ambiente de Ensino à Distância (EAD ou *e-learning*) da UFV (chamado PVANet) para dispositivos móveis, dentro de um projeto de dissertação de mestrado (SILVA et al., 2008; NOGUEIRA et al., 2008). A Figura 2.1 representa uma amostra desses resultados.

Figura 2.1. Tela do PVANet sem adaptação (esquerda) e adaptado (direita) para uso em um dispositivo móvel.

Fonte: (SILVA, 2008)

2.2 Portabilidade gráfica em dispositivos móveis

Desenvolver *software* para dispositivos móveis é um desafio. Além das necessidades de adaptação em relação ao tamanho da tela e limitação de recursos em termos de memória, processador, conexão intermitente etc, um outro grande problema é a portabilidade entre inúmeros dispositivos móveis e sistemas operacionais existentes no mercado.

Uma boa alternativa a este último é a criação de camadas de adaptação (*middlewares*) com o objetivo de abstrair as nuances de cada plataforma de programação. Em relação aos recursos gráficos, o uso do conjunto (biblioteca gráfica padrão mais camada de adaptação) permite separar melhor a camada de apresentação gráfica dentro do software, de modo que o mesmo possa ser mais facilmente adaptado para diferentes sistemas.

Infelizmente muitos dispositivos móveis, como telefones celulares, tradicionalmente têm seu sistema fechado, no sentido de que, após a compra, nenhum aplicativo possa ser instalado. Essa prática também pode ser vista em sistemas para consoles de jogos, onde os fabricantes tentam controlar quem pode desenvolver jogos para seu sistema, além do hardware ser todo personalizado. Com esta situação, há uma limitação na demanda para APIs no padrão aberto (PULLI et al., 2005).

Mesmo assim, existem algumas APIs gráficas que são padrões para muitas pesquisas e aplicações móveis. Para gráficos 3D, destacam-se a OpenGL ES e M3G. Kari Pulli e alguns parceiros de trabalho já publicaram muitas informações sobre o uso dessas ferramentas (CAPIN et al., 2008). Uma das publicações que Pulli faz sozinho apresenta um panorama sobre as principais APIs existentes (PULLI, 2006), este serviu de base para a descrição das seguintes ferramentas:

2.2.1 OpenGL ES

Sem dúvida a OpenGL é uma das bibliotecas gráficas mais populares para o desenvolvimento de aplicativos gráficos, jogos e ambientes 3D. Desde a sua criação, há quase 20 anos, ela tem sido ampliada e vem incorporando funcionalidades quando um novo hardware é lançado. Não é por acaso que está resistindo ao teste do tempo.

No entanto, a OpenGL é muito grande para dispositivos móveis. Sendo assim, o Khronos Group realizou as modificações necessárias para que a API possa atender as restrições dos dispositivos móveis. Assim surgiu a OpenGL ES, uma versão da famosa biblioteca gráfica agora para sistemas embarcados (PULLI et al., 2005).

A primeira versão (OpenGL ES 1.0) foi lançada em 2003, e um ano depois saiu o pacote da versão 1.1 juntamente com o projeto da OpenGL 2.0, esta última lançada somente no final de 2006 (PULLI et al., 2008). Hoje é amplamente adotada pela indústria de dispositivos móveis, como os aparelhos equipados com o Symbian OS. A versão 1.1 é usada como biblioteca de gráficos 3D do iPhone e até mesmo pelo console de jogos Playstation 3 da Sony.

Embora ela permita também a construção de aplicações contendo recursos gráficos em 2D, seu enfoque é para gráficos 3D, componente indispensável para jogos e animações, que por sua vez são objetos de consumo da indústria de entretenimento. Além disso, o uso da API de baixo nível não é trivial e demanda um esforço considerável por parte do programador que só necessita de recursos 2D para auxiliar na portabilidade da sua aplicação.

2.2.2 Mobile 3D Graphics API (M3G)

É sabido que a OpenGL ES é voltada para programas escritos em C/C++. Mesmo que existam muitos dispositivos com sistemas que permitem a execução de

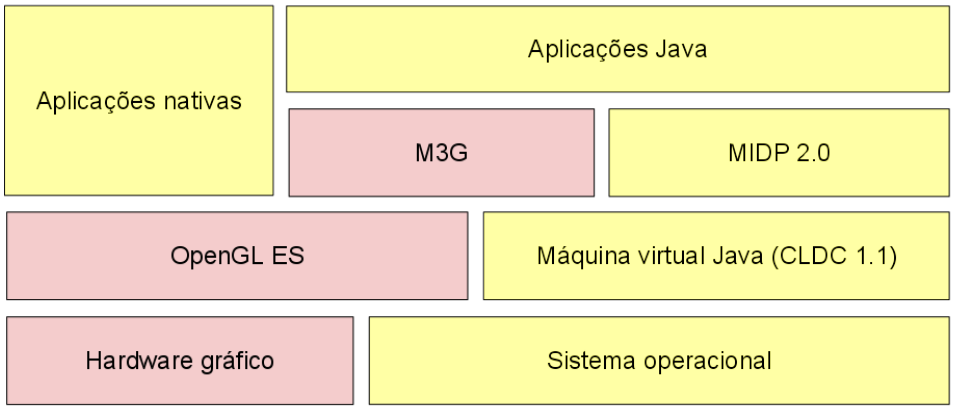
programas nessas linguagens, o número é bem menor quando se comparado aos aparelhos que suportam Java ME (Java Micro Edition).

Diante desse quadro, se viu necessário a criação de uma API de gráficos 3D para a plataforma Java Micro Edition. Diferentemente do que foi feito com a OpenGL ES, que é uma versão enxuta da OpenGL, a M3G (JSR 184) foi construída seguindo os princípios de Java 3D, mas como uma API totalmente nova (PULLI, 2006).

Mobile 3D Graphics API é uma interface de alto nível, eficiente e compacta, oferecida aos programadores Java que precisam gerir cenas 3D. Apesar de ser baseada no *pipeline* de renderização da OpenGL ES, ela é projetada com base na orientação a objetos. Provendo assim uma pequena curva de aprendizado aos desenvolvedores de aplicações móveis com recursos gráficos 3D (PULLI et al., 2008).

Dessa forma, observa-se que a diferença básica entre as APIs está no fato da OpenGL ES ser de baixo nível, enquanto a M3G é de alto nível. Isso que dizer que esta última é mais amigável ao programador, porém aquela é mais indicada para aplicações que necessitam de um desempenho melhor. A Figura 2.2 exemplifica uma arquitetura em alto nível para aplicações móveis. A M3G provê gráficos 3D para *midlets* Java, enquanto OpenGL ES serve tanto a aplicações nativas quanto a M3G (PULLI et al., 2005).

Figura 2.2. Arquitetura de software móvel em alto nível.



Fonte: (Adaptada de PULLI et al., 2005).

Recursos gráficos 2D foi um dos temas mais discutidos pelo grupo responsável pela padronização da M3G. No final, o grupo tentou chegar a um meio termo entre a compatibilidade com padrões OpenGL ES e também permitir otimizações em

motores de softwares que não estariam usando aceleração de hardware. Infelizmente, isso significou algumas limitações desagradáveis no conjunto de recursos 2D. Na verdade, alguns elementos, como desenho de retângulos ou círculos, não têm significado em gráficos 3D, já que a renderização de gráficos em jogos é composta basicamente pela composição de pontos, linhas retas, triângulos e texturas (PULLI et al., 2008).

2.2.3 OpenVG

Até o momento as bibliotecas apresentadas focavam recursos gráficos 3D, principalmente jogos e animações. Durante a criação da OpenGL ES 1.0 houve muita discussão se deveriam expandir a API para ter suas funcionalidades 2D mais completas e que poderiam ser usadas para implementar um SVG player (Scalable Vector Graphics), mas no final manteve-se uma API quase que puramente 3D (PULLI, 2006).

No entanto, ainda havia a necessidade de se criar uma API aberta mais dedicada a gráficos bidimensionais. O mesmo Khronos Group, em agosto de 2005, lançou a versão 1.0 da OpenVG, uma API de baixo nível para gráficos 2D. Sua estrutura é bastante semelhante ao OpenGL ES, para que desenvolvedores de aplicativos móveis que já conhecem esta API tenham sua curva de aprendizado suavizada (KHRONOS, 2005).

A OpenVG fornece uma interface de aceleração de hardware, portanto de baixo nível, para bibliotecas de gráficos vetoriais, tais como SVG e Flash. As aplicações-alvo para o qual a ela foi desenvolvida incluem visualizadores SVG, aplicações de mapeamento, e-books, jogos, interfaces de usuário escalável etc (PULLI, 2006; KHRONOS, 2005).

Naturalmente o que move essas iniciativas é a indústria de entretenimento e percebe-se que mesmo tratando de gráficos bidimensionais, o foco desta API é a animação 2D. Por causa desse foco e por se tratar de uma interface baixo nível, o programador que desejar usar recursos 2D estáticos em sua aplicação técnica terá um maior esforço para usar a OpenVG.

2.2.4 Outros trabalhos correlatos

Computação Móvel é uma área muito nova na Ciência da Computação. A pesquisa na área é mais avançada dentro das empresas privadas fabricantes de dispositivos móveis e derivados. No meio acadêmico experimentamos um crescimento principalmente por investimentos dessas empresas em projetos dentro de universidades e institutos de pesquisa.

No Brasil existem alguns polos bem desenvolvidos por esse tipo de investimento privado, mas ainda não é fácil encontrar muitos trabalhos acadêmicos, visto que boa parte dos cursos superiores na área nem possuem esta disciplina em sua grade curricular.

Ao se limitar a área de desenvolvimento e recursos gráficos dentro da Computação Móvel, o leque de trabalhos correlatos é ainda mais fechado, porém foi possível encontrar uma dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação no Instituto de Informática da Universidade Federal do Rio Grande do Sul (UFRGS), em 2006, sob orientação do Professor Dr. Luigi Carro.

O título da pesquisa é "Conjunto de Classes para Aplicações Gráficas 2D em Sistemas Embarcados Baseados no Femtojava", tendo como autor Leandro Augusto de Oliveira, que escreve em seu resumo: "Este trabalho descreve o estudo de técnicas em software para reduzir o consumo de energia e aumentar o desempenho de sistemas embarcados com recursos gráficos, baseados no microprocessador Femtojava. Como subproduto deste trabalho, foi desenvolvida uma biblioteca gráfica para a plataforma Femtojava. Dois estudos de caso foram desenvolvidos para analisar a biblioteca desenvolvida, caracterizando o seu consumo de energia e desempenho." (OLIVEIRA, 2006).

Apesar de focar nos recursos gráficos 2D de uma aplicação móvel, o projeto trata de um nível mais baixo de desenvolvimento e cria uma biblioteca para o Femtojava, um microcontrolador que executa *bytecodes* Java diretamente, sem uma máquina virtual.

A grande semelhança entre os trabalhos está na intenção de criar uma ferramenta que auxilie o programador no desenvolvimento de aplicações que usam gráficos 2D. Contudo um é restrito à sistemas embarcados baseados no Femtojava e o outro é focado na portabilidade entre sistemas operacionais e dispositivos móveis.

3 2D MOBILE GRAPHICS (2DMG)

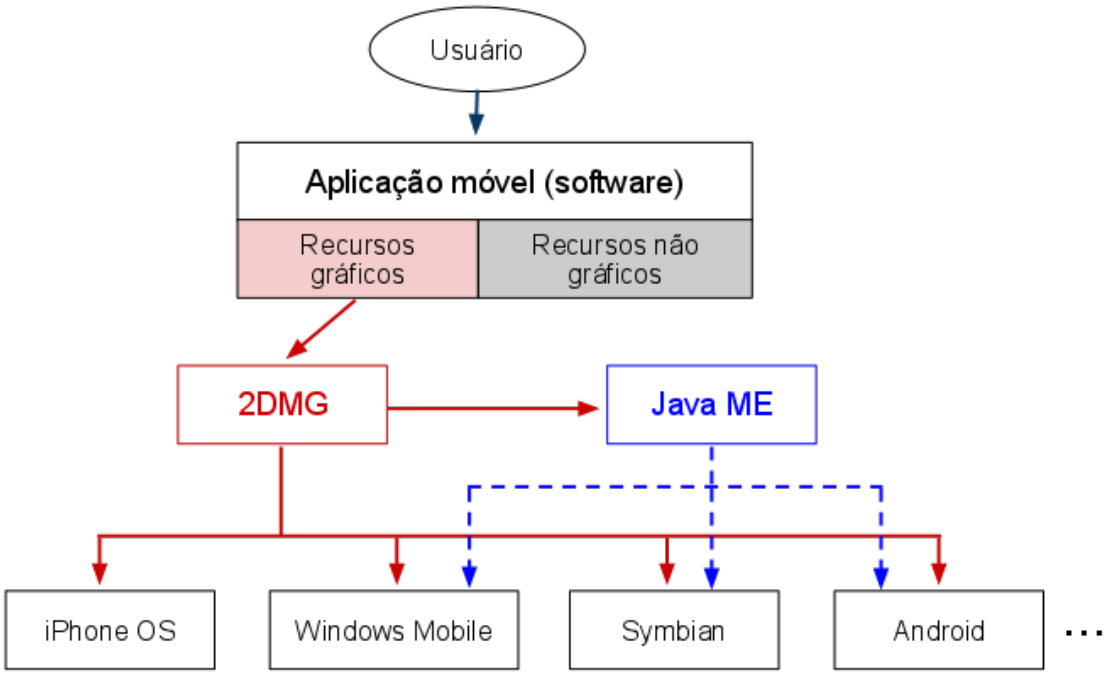
A grande quantidade de sistemas operacionais e linguagens de programação diferentes que existem hoje para o desenvolvimento de aplicações em dispositivos móveis dificulta, consideravelmente, a portabilidade do software entre sistemas e dispositivos diferentes. Uma das formas de facilitar essa portabilidade e, portanto, reduzir o custo de desenvolvimento do *software*, se da através do uso de camadas intermediárias de programação (*middlewares*).

Este trabalho foca na solução de parte desse problema, que é a portabilidade nos recursos gráficos 2D de uma aplicação móvel. Depois de avaliar algumas alternativas existentes, relatadas no Capítulo 2, observa-se que, em sua maioria, elas se dedicam à implementação de gráficos 3D em dispositivos móveis, voltadas especialmente para o desenvolvimento de jogos e animações gráficas (MOSMONDOR et al., 2006; CALLOW et al., 2007).

2D Mobile Graphics (2DMG) é uma arquitetura de classes como camada de adaptação (*middleware*) que auxilia o desenvolvimento de aplicações que utilizam gráficos 2D (aplicações científicas, visualização de mapas e dados estatísticos, projetos de engenharia, entre outros) em dispositivos móveis, em particular PDAs, smartphones genéricos e tablets.

Diferentemente da API M3G, que é uma especificação sob o Java ME, o 2DMG especifica um conjunto de classes abstratas que foram implementadas em linguagem nativa aos SOs iOS (iPhone/iPad/iPod Touch), Symbian, Windows Mobile e Android, além da implementação em Java ME. Sendo assim, o programador pode usufruir de uma interface única de desenvolvimento de gráficos 2D em diferentes dispositivos atendidos por esses SOs. Um diagrama em alto nível (Figura 3.1) exemplifica onde o *middleware* 2DMG está inserido no contexto de desenvolvimento de software para dispositivos móveis.

Figura 3.1. Diagrama em alto nível mostrando a arquitetura do 2DMG dentro do contexto de desenvolvimento de software para dispositivos móveis.



Fonte: Elaborada pelo autor.

3.1 A arquitetura

Segundo o glossário disponível no sítio do Software Engineering Institute (SEI)[1], a arquitetura de software é a organização estrutural de um sistema, compreendida pelos seus elementos, relacionamentos entre si e com o meio externo.

De acordo com Garlan (2000), "Uma arquitetura de software envolve a descrição de elementos arquiteturais dos quais os sistemas serão construídos, interações entre esses elementos, padrões que guiam suas composições e restrições sobre estes padrões".

Normalmente a arquitetura de um software é organizada em visões. No Rational Unified Process (RUP), parte-se de um conjunto típico de visões, denominado "esquema de visão 4+1", composto pela visão de casos de uso, visão lógica, visão de implementação, visão de processos e visão de implantação (KRUCHTEN, 1995).

Embora tais visões possam especificar todo o projeto de um sistema, a arquitetura está focada em alguns aspectos específicos, como a estrutura do modelo

[1]http://www.sei.cmu.edu, acessada em maio de 2011.

(divisão em camadas), alguns elementos essenciais (casos de uso críticos, classes principais etc), os cenários mostrando os principais fluxos de controle e os serviços.

Este trabalho propõe um conjunto de classes como camada de adaptação (*middleware*), que não é propriamente um software e sim uma parte dele. Sendo assim, a arquitetura não é especificada aqui como sugerem as boas práticas da engenharia de software. No entanto, usaremos alguns artefatos e elementos que julgamos necessário para o bom entendimento da estrutura do modelo, seus elementos e sua interação entre si e com outros externos.

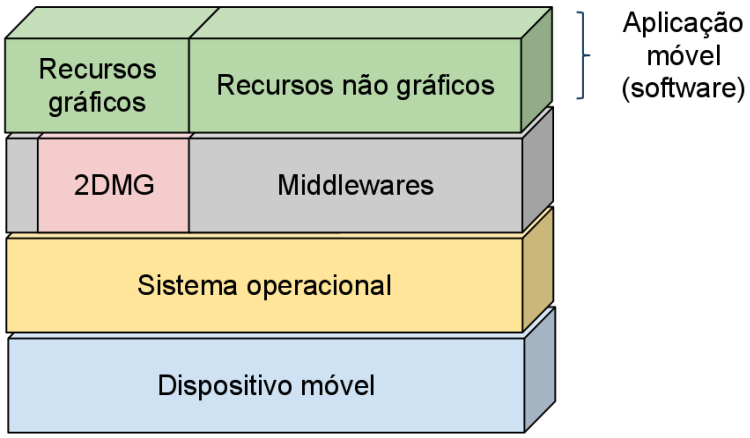
3.2 Divisão em camadas

A divisão em camadas auxilia no entendimento da estrutura do modelo. Ela permite um particionamento lógico do sistema e subsistemas, além de mostrar as regras de como estabelecer relacionamentos entre camadas.

Num exemplo de arquitetura em quatro camadas, podemos ter uma camada mais inferior como sendo o sistema operacional, depois uma camada de adaptação (interface com banco de dados, construtores de interface com o usuário, componentes responsáveis pela parte gráfica etc), outra camada com partes específicas do negócio usadas em vários aplicativos, e, por último, no nível superior, a camada de aplicativos com serviços específicos.

A Figura 3.2 exemplifica uma arquitetura em camadas de um sistema móvel e onde o objeto deste trabalho está inserido. É possível perceber que ele é parte de uma arquitetura, dificultando assim a especificação de uma arquitetura completa para ele.

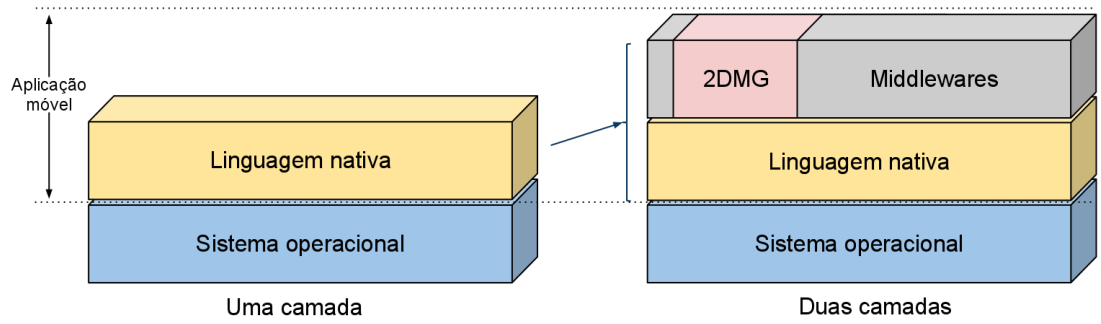
Figura 3.2. Possível arquitetura em camadas de um sistema móvel, destacando o *middleware* 2DMG.



Fonte: Elaborada pelo autor.

A visão da implementação pode ser detalhada (Figura 3.3) destacando a diferença entre o desenvolvimento da aplicação móvel (software) em uma camada ou em duas camadas, este imprescindível para o uso do objeto deste trabalho. A diferença entre a implementação em uma camada ou duas é discutida e exemplificada no Capítulo 4, onde compara-se a codificação em linguagem nativa e usando o 2DMG.

Figura 3.3. Divisão em camadas comparando a implementação em uma ou duas camadas.



Fonte: Elaborada pelo autor.

3.3 Classes como camada de adaptação

A construção de camadas de adaptação (*middleware*) com o objetivo de abstrair os detalhes de suas sub-rotinas é comum no desenvolvimento de software. O uso desse conjunto (biblioteca gráfica padrão mais camada de adaptação) permite separar melhor a camada de apresentação gráfica dentro da aplicação, de modo que o mesmo possa ser mais facilmente adaptado para diferentes sistemas, como Windows Mobile, Symbian, iOS ou Android.

3.3.1 Diagrama de classes

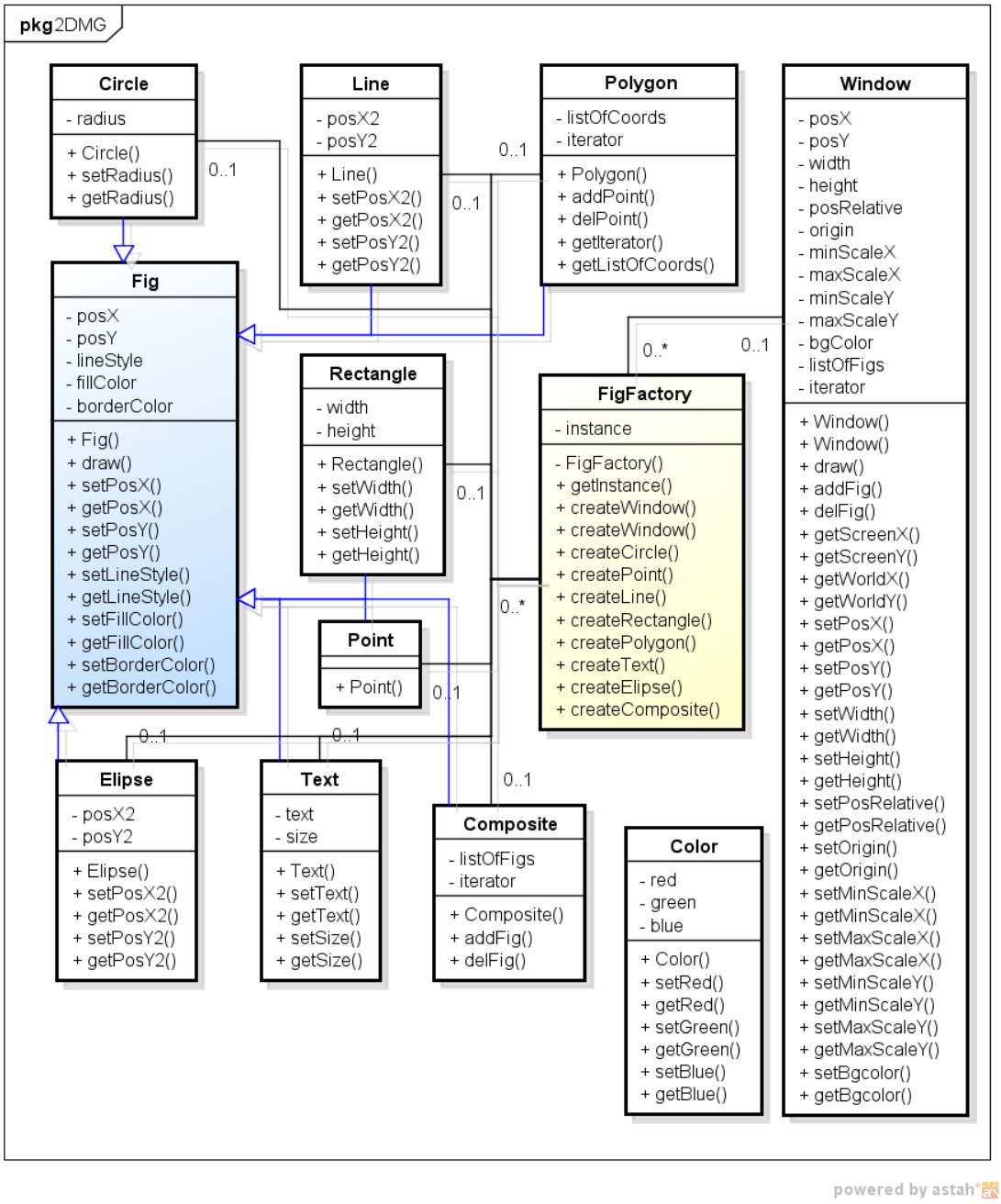
Depois de avaliar algumas alternativas existentes, principalmente em relação a recursos gráficos 3D, e levantar os elementos básicos necessários ao desenvolvimento de aplicações que utilizam recursos gráficos 2D, como gráficos estatísticos ou figuras em duas dimensões, foi possível especificar o conjunto de classes e suas relações.

Destaca-se duas classes principais: *Classe Fig* (generalização das classes que representam os elementos gráficos específicos) e a *Classe FigFactory* (padrão de

projeto usado para abstrair a criação dos elementos). Será relatado neste capítulo a função desse padrão de projeto e o porquê da sua escolha.

O diagrama de classes dessa primeira versão do *middleware* pode ser visto na Figura 3.4. Foram omitidos os parâmetros dos métodos e algumas outras informações no intuito de melhorar a visualização geral das classes e seus relacionamentos. Ainda nas seções seguintes do capítulo atual, as classes serão detalhadas separadamente.

Figura 3.4. Diagrama de classes que compõem esta primeira versão do 2DMG, destacando as duas classes principais.

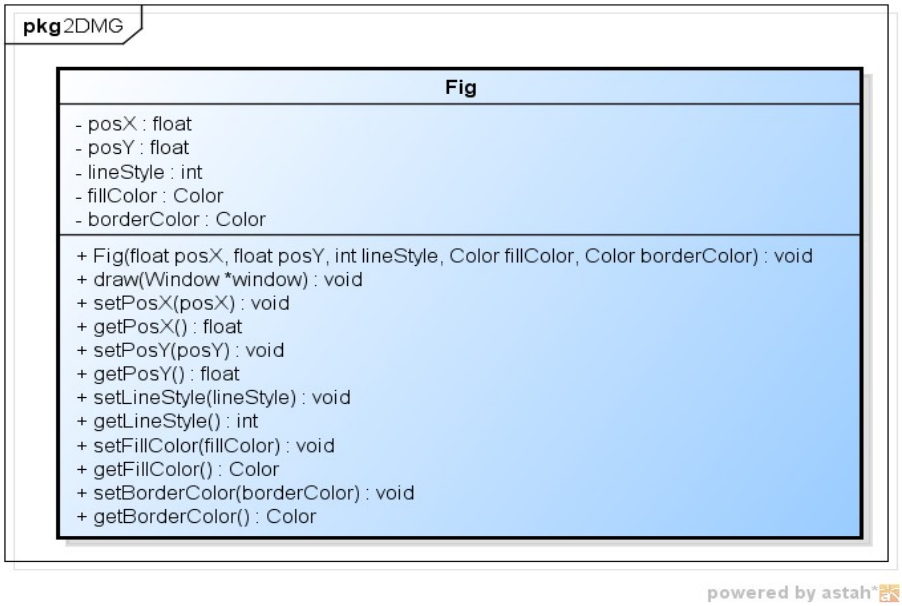


Fonte: Elaborada pelo autor.

3.3.2 Classe *Fig*

É a generalização das classes dos elementos gráficos 2D que compõem o *middleware*. Classe abstrata de figura que representa os atributos e métodos comuns a esses elementos, sendo que todas as figuras são subclasses dela (Figura 3.5).

Figura 3.5. Classe Fig com seus atributos e métodos.



Fonte: Elaborada pelo autor.

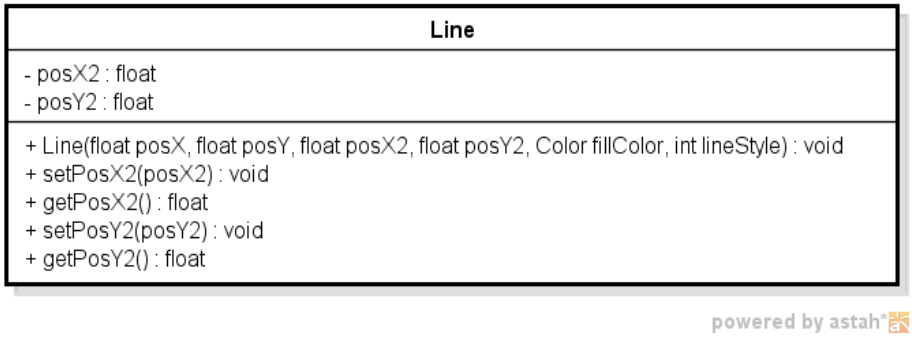
Os atributos desta classe, comuns aos elementos gráficos, representam sua posição (coordenada x – *posX*, coordenada y – *posY*), estilo da borda (*lineStyle*), cor do preenchimento da figura (*fillColor*) e cor da borda (*borderColor*).

Além dos métodos *getters* e *setters* e o construtor (Fig), a classe possui o método que desenha (*draw()*) a figura em uma janela (parâmetro *window*). Este método é comuns a todos os elementos gráficos representados por especializações da classe Fig.

3.3.3 Classe *Line*

Esta classe representa o elemento gráfico linha. Além dos atributos da classe Fig, ela possui atributos específicos, que por sua vez representam a posição final da linha (coordenada x – *posX2*, coordenada y – *posY2*) (Figura 3.6).

Figura 3.6. Classe Line com seus atributos e métodos.

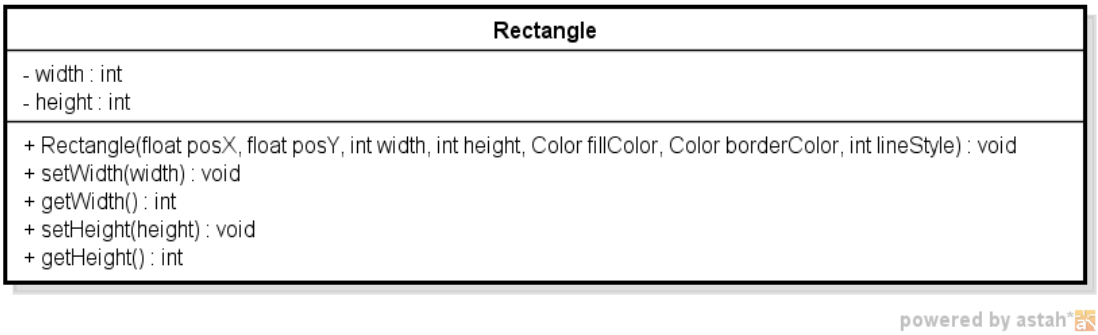


Fonte: Elaborada pelo autor.

3.3.4 Classe *Rectangle*

Classe que representa o elemento gráfico retângulo. Além dos atributos da classe Fig, ela possui seus atributos específicos: largura (*width*) e altura (*height*) (Figura 3.7).

Figura 3.7. Classe Rectangle com seus atributos e métodos.

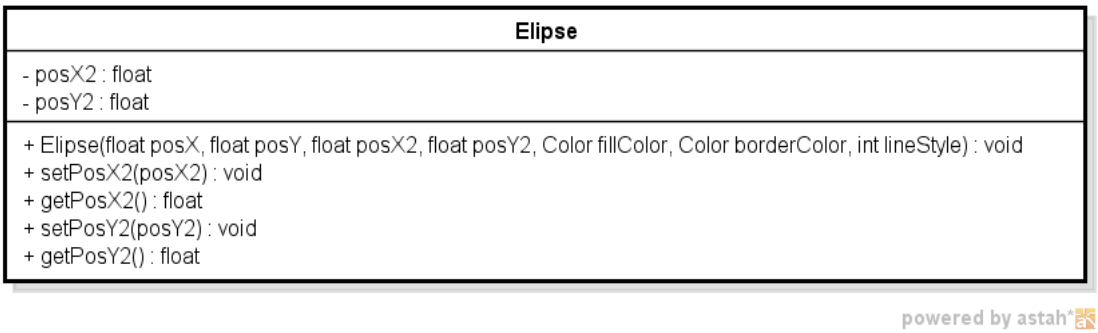


Fonte: Elaborada pelo autor.

3.3.5 Classe *Elipse*

Uma elipse tem dois focos, que no caso do círculo são sobrepostos. Sendo assim, esta classe necessita de mais dois atributos específicos para representarem as coordenadas do segundo foco (*posX2*, *posY2*), já que o primeiro foco é herdado da classe Fig através dos atributos (*posX*, *posY*) (Figura 3.8).

Figura 3.8. Classe Elipse com seus atributos e métodos.

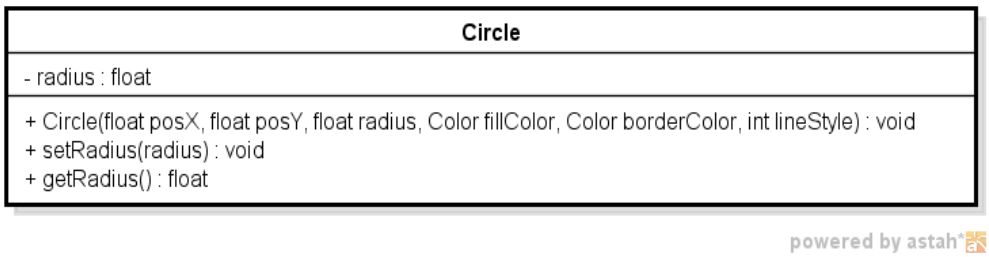


Fonte: Elaborada pelo autor.

3.3.6 Classe *Circle*

Esta classe define o elemento gráfico círculo. As coordenadas do centro são herdadas da classe Fig (*posX*, *posY*), portanto foi necessário especificar apenas um atributo no intuito de representar o raio (*radius*) (Figura 3.9).

Figura 3.9. Classe Circle com seus atributos e métodos.



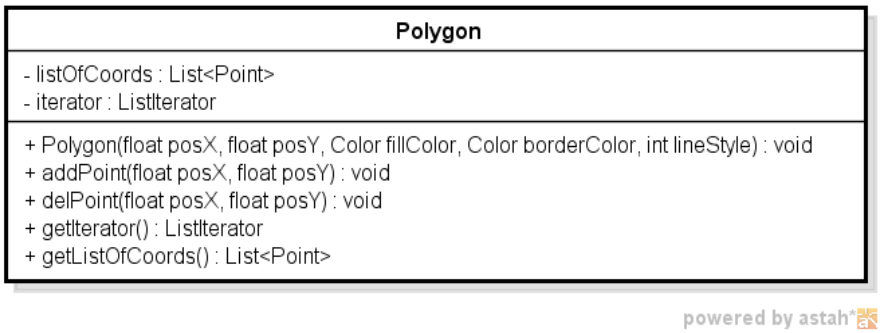
Fonte: Elaborada pelo autor.

3.3.7 Classe *Polygon*

A classe Polygon (Figura 3.10) representa o elemento gráfico polígono, este definido por uma lista de pontos (*listOfCoords*) que se conectam por linhas. O outro atributo (*iterator*) é usado como iterador para percorrer a lista de coordenadas.

Além dos dois atributos citados, esta classe possui dois métodos específicos. Um para adicionar pontos ao polígono (*addPoint()*) e outro para removê-los (*delPoint()*).

Figura 3.10. Classe Polygon com seus atributos e métodos.

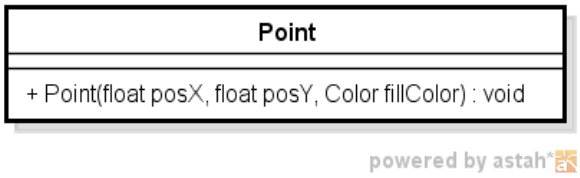


Fonte: Elaborada pelo autor.

3.3.8 Classe *Point*

Esta classe define o elemento gráfico ponto (Figura 3.11). O mesmo é representado pelas coordenadas (*posX*, *posY*) herdados da classe Fig, portanto não possui atributos e métodos específicos além do construtor da classe.

Figura 3.11. Classe Point em detalhe.

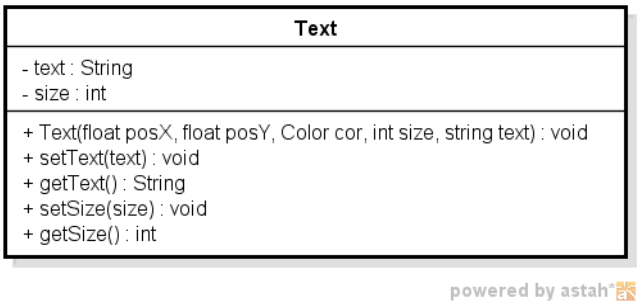


Fonte: Elaborada pelo autor.

3.3.9 Classe *Text*

Diferentemente das classes descritas até o momento, esta define o elemento gráfico texto e não uma figura geométrica. Ela possui dois atributos específicos que representam o texto (*text*) e o tamanho dele (*size*) (Figura 3.12).

Figura 3.12. Classe Text com seus atributos e métodos.



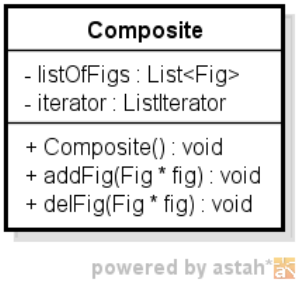
Fonte: Elaborada pelo autor.

3.3.10 Classe *Composite*

Esta classe também não define uma figura geométrica, e sim uma lista de elementos gráficos (figuras) representada pelo atributo (*listOfFigs*). Assim como na classe Polygon, o outro atributo (*iterator*) é usado como iterador para percorrer a lista, porém de figuras e não de pontos (Figura 3.13).

Também se fez necessário a definição de dois métodos específicos para adicionar (*addFig()*) e remover (*delFig()*) figuras à lista.

Figura 3.13. Classe Composite com seus atributos e métodos.



Fonte: Elaborada pelo autor.

3.3.11 Classe *Color*

Como um dos objetivos deste trabalho é definir uma interface única para o desenvolvedor que deseja utilizar recursos gráficos 2D em sua aplicação, auxiliando a portabilidade em diferentes sistemas operacionais, foi necessário especificar esta classe auxiliar para representar a cor usada no preenchimento (*fillColor*) e na borda (*borderColor*), atributos presentes na classe generalizada Fig.

Assim sendo, tal tipo de dado fica independente da linguagem nativa usada na implementação do software móvel. Para isso foram utilizados três parâmetros (*red*, *green* e *blue*), ver Figura 3.14, para representar a cor no sistema RGB, abreviação de Red (vermelho), Green (verde) e Blue (azul). Este padrão é usado para reproduzir cores em monitores de TV e computadores, assim como em telas de dispositivos móveis.

Figura 3.14. Classe Color com seus atributos e métodos.

Fonte: Elaborada pelo autor.

3.3.12 Classe *Window*

Classe que representa o elemento gráfico janela, onde as figuras geométricas e outros elementos são desenhados (Figura 3.15). Uma aplicação móvel pode conter várias janelas com elementos gráficos 2D diferentes, escalas diversas, posições diferentes em relação à tela do dispositivo etc. Para isso a classe possui os seguintes atributos e métodos:

Atributos

- *posX* e *posY* – coordenadas X e Y da posição da janela em relação à tela;
- *width* e *heigth* – largura e altura da janela;
- *posRelative* – posição relativa da janela em relação à tela;
- *origin* – posição da origem, coordenadas (0,0), na janela;
- *minScaleX* e *maxScaleX* – limite inferior e superior para coordenadas no eixo-x;
- *minScaleY* e *maxScaleY* – limite inferior e superior para coordenadas no eixo-y;
- *bgColor* – cor de fundo da janela;
- *listOfFigs* – lista de figuras que são desenhadas na janela;
- *iterator* – usado para percorrer a lista de figuras;

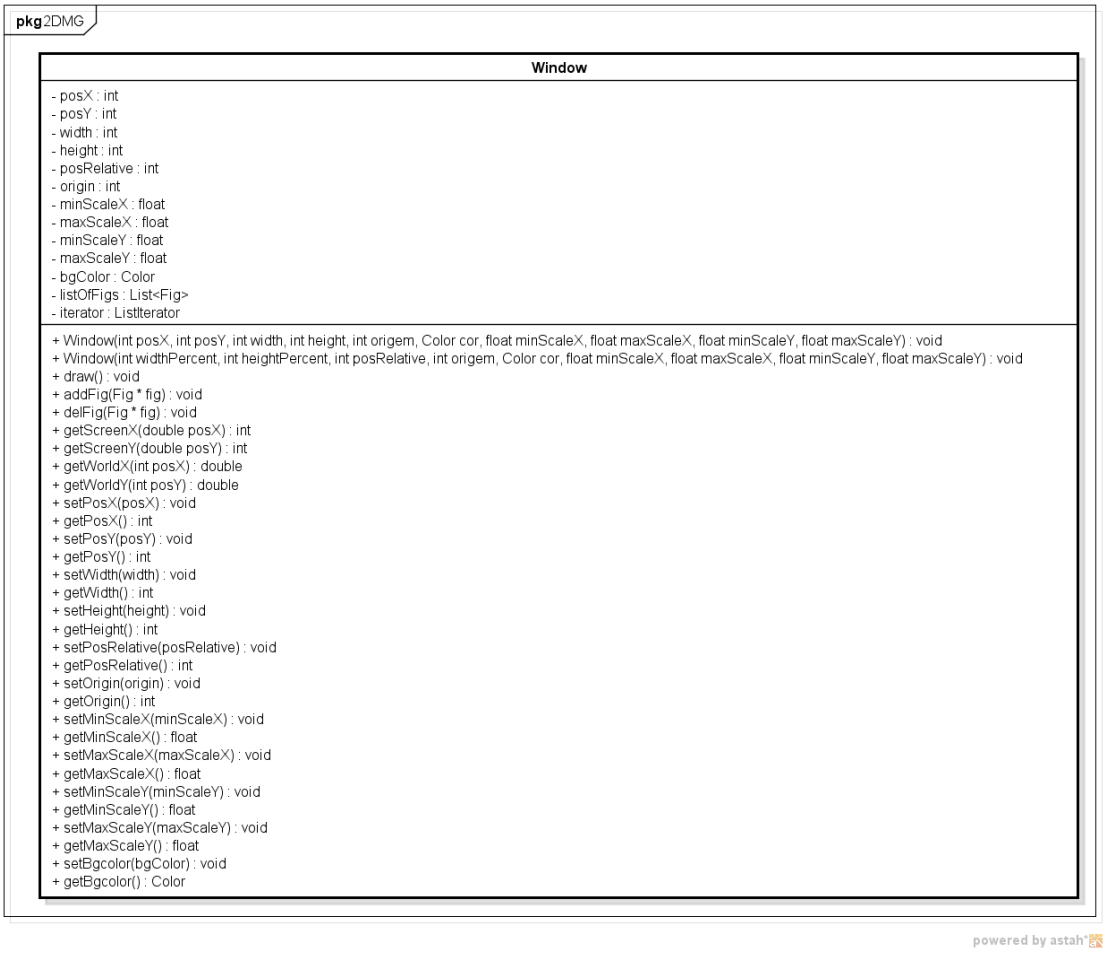
Os objetos da classe window podem ser criados de duas formas e por isso ela possui dois construtores. O primeiro recebe como parâmetro as coordenadas (posX, posY) da posição da janela em relação à tela, sua largura (*width*) e altura (*heigth*), a

origem da janela (canto superior esquerdo ou inferior esquerdo), a cor de fundo, a escala mínima e máxima no eixo-x e a escala mínima e máxima no eixo-y.

O segundo se diferencia do outro porque recebe a largura e a altura como porcentagens em relação à tela, bem como a posição relativa (*posRelative*), definida pelas constantes (TOP_LEFT, TOP_RIGHT, BOTTOM_LEFT, BOTTOM_RIGHT, CENTER, CENTER_LEFT, CENTER_RIGHT, CENTER_TOP, CENTER_BOTTOM, CUSTOM).

Além dos métodos *getters* e *setters* dos seus atributos, a classe Window possui os métodos para adicionar (*addFig()*) e remover (*delFig()*) figuras à janela e outros quatro que são responsáveis pela conversão das coordenadas X e Y na escala definida para a janela: *getScreenX() e getScreenY()* convertem coordenadas "reais" para as da tela e *getWorldX() e getWorldY()* fazem o contrário. Por último, o método que desenha a janela (*draw()*), que basicamente varre a lista de figuras (*listOfFigs*) chamando o método de desenho de cada uma.

Figura 3.15. Classe Window com seus atributos e métodos.



Fonte: Elaborada pelo autor.

3.4 Padrão *Factory (FigFactory)*

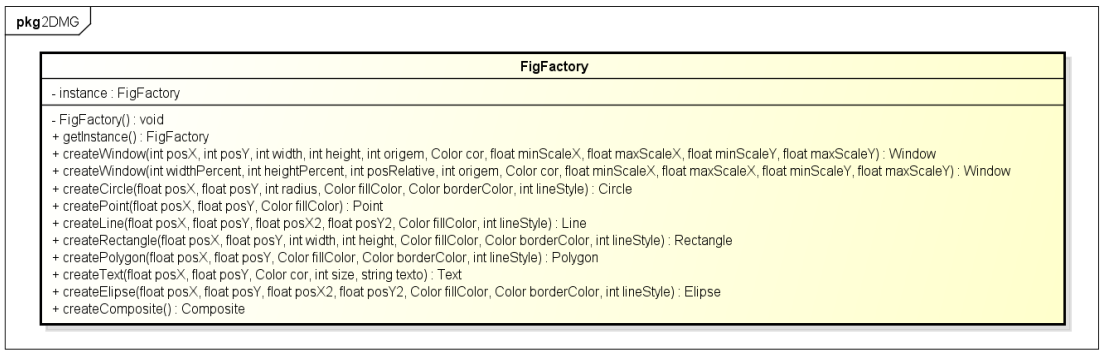
Outra classe importante no *middleware*, juntamente com a Fig, é a classe FigFactory. Ela é implementada tendo como base o padrão de projeto *Abstract Factory*, em que os objetos são criados segundo uma interface que abstrai a especificação de suas classes concretas (Figura 3.16).

O *Factory* é classificado como padrão de criação dentro das famílias de padrões "GoF" (*Gang of Four*), em português "Gangue dos Quatro". Em resumo, o objetivo desse padrão é prover uma interface para criar uma família de objetos relacionados ou dependentes sem especificar suas classes concretas (GAMMA, 1995).

Assim sendo, o uso do padrão Fábrica neste trabalho se justifica porque as figuras se relacionam e o desenvolvedor passa a ter uma interface única para a criação dos objetos, mesmo com classes concretas diferentes para cada sistema operacional.

No Capítulo 4 as implementações são comparadas e é possível observar que até nomes de atributos e métodos foram alterados na implementação das classes, pois estavam conflitando com nomes da linguagem nativa. Com o uso do padrão *Abstract Factory* tal problema foi minimizado.

Figura 3.16. Classe FigFactory com seus atributos e métodos.



Fonte: Elaborada pelo autor.

A seguir relatamos algumas comparações e discussões sobre as implementações do modelo proposto neste capítulo nas respectivas linguagens nativas dos sistemas operacionais móveis. Também demonstramos resultados que comprovam a eficacia do modelo no intuito de auxiliar o programador na codificação de programas que necessitam de recursos gráficos 2D.

4 IMPLEMENTAÇÕES E DISCUSSÕES

O 2DMG foi projetado para ser uma interface em alto nível que auxilie a portabilidade dos recursos gráficos 2D de uma aplicação móvel. Para que ele seja realmente útil para o programador foi preciso implementar o modelo descrito no Capítulo 3 em algumas plataformas e sistemas operacionais.

Boa parte do tempo do projeto foi gasto nesta fase, mas na dissertação foram suprimidos os códigos completos das implementações pois são muito extensos. Eles serão publicados no repositório de projetos *open source* Google Code[2] e estarão disponíveis após a publicação da dissertação.

Contudo, para exemplificar as comparações relatadas neste capítulo em relação às linhas de código, fez-se necessário mostrar parte da implementação das classes **Circle**, **Rectangle** e **Polygon** nas linguagens nativas C++ (Windows Mobile), C++ (Symbian), Java (J2ME) e Objective-C ou Obj-C (iOS). Como essas linguagens são orientadas a objetos, o que mais difere nas implementações, além dos detalhes da sintaxe, é o método **draw()**, responsável pelo desenho do respectivo elemento gráfico da classe. Então eliminamos o cabeçalho da classe, os construtores, os destrutores e os métodos *getters* e *setters*, pois não seriam relevantes para a avaliação.

4.1 Implementação do 2DMG para o Windows Mobile

Para o desenvolvimento das classes foi usada a IDE (Integrated Development Environment, ou em português "Ambiente Integrado de Desenvolvimento") Visual Studio[3] com a SDK do Windows Mobile. Um editor de texto comum poderia ter sido usado na codificação, mas como a SDK trabalha em conjunto com essa IDE, o uso da mesma foi imprescindível, principalmente por fornecer um simulador para verificar a execução das rotinas de cada classe.

[2] http://code.google.com/p/2dmg/, acessada em julho de 2011.

[3] http://www.microsoft.com/visualstudio/en-us/products/2010-editions/windows-phone-developer-tools, acessada em abril de 2011.

O código-fonte da implementação do método draw() das classes Circle, Rectangle e Polygon pode ser visto nos respectivos Quadros 4.1, 4.2 e 4.3. Suprimimos todos os comentários e linhas em branco para deixar o código mais enxuto para a contagem do número de linhas.

Quadro 4.1. Implementação do método draw() da classe Circle em C++ (Windows Mobile).

```
1.    void Circle::draw (Window *window){
2.          HDC hdc = GetWindowDC(window->GetHWnd());
3.          SetBkMode(hdc, TRANSPARENT);
4.          HBRUSH NewBrush = CreateSolidBrush(RGB(GetColor().GetRed(),
   GetColor().GetGreen(), GetColor().GetBlue()));
5.          SelectObject(hdc, NewBrush);
6.          HPEN hPen =
   CreatePen(GetLineStyle(),1,RGB(GetBorderColor().GetRed(),
   GetBorderColor().GetGreen(), GetBorderColor().GetBlue()));
7.          SelectObject(hdc, hPen);
8.          if (window->GetOrigin() == TOP_LEFT)
9.                Ellipse (hdc, window->GetScreenX(posX-radius), window-
   >GetScreenY(posY-radius), window->GetScreenX(posX+radius), window-
   >GetScreenY(posY+radius));
10.         else if (window->GetOrigin() == BOTTOM_LEFT)
11.               Ellipse (hdc, window->GetScreenX(posX-radius), (window-
   >GetHeight())-window->GetScreenY((posY-radius)), window-
   >GetScreenX(posX+radius), (window->GetHeight())-window-
   >GetScreenY((posY+radius)));
12.   }
```

Fonte: Elaborado pelo autor.

Quadro 4.2. Implementação do método draw() da classe Rectangle em C++ (Windows Mobile).

```
1.    void Rectangle_::draw (Window *window){
2.          HDC hdc = GetWindowDC(window->GetHWnd());
3.          SetBkMode(hdc, TRANSPARENT);
4.          HBRUSH NewBrush = CreateSolidBrush(RGB(GetColor().GetRed(),
   GetColor().GetGreen(), GetColor().GetBlue()));
5.          SelectObject(hdc, NewBrush);
6.          HPEN hPen =
   CreatePen(GetLineStyle(),1,RGB(GetBorderColor().GetRed(),
   GetBorderColor().GetGreen(), GetBorderColor().GetBlue()));
7.          SelectObject(hdc, hPen);
8.          RECT rect;
9.          if (window->GetOrigin() == TOP_LEFT){
10.               rect.left = window->GetScreenX(posX);
11.               rect.right = window->GetScreenX(posX+width);
12.               rect.top = window->GetScreenY(posY);
13.               rect.bottom = window->GetScreenY(posY+height);
14.         }
15.         else if (window->GetOrigin() == BOTTOM_LEFT){
16.               rect.left = window->GetScreenX(posX);
17.               rect.right = window->GetScreenX(posX+width);
18.               rect.top = (window->GetHeight())-window-
   >GetScreenY((posY));
19.               rect.bottom = (window->GetHeight())-window-
   >GetScreenY((posY+height));
20.         }
21.         Rectangle (hdc, rect.left, rect.top, rect.right, rect.bottom);
22.   }
```

Fonte: Elaborado pelo autor.

Quadro 4.3. Implementação do método draw() da classe Polygon em C++ (Windows Mobile).

```
1.    void Polygon_::draw (Window *window){
2.          HDC hdc = GetWindowDC(window->GetHWnd());
3.          SetBkMode(hdc, TRANSPARENT);
4.          HBRUSH NewBrush = CreateSolidBrush(RGB(GetColor().GetRed(),
    GetColor().GetGreen(), GetColor().GetBlue()));
5.          SelectObject(hdc, NewBrush);
6.          HPEN hPen =
    CreatePen(GetLineStyle(),1,RGB(GetBorderColor().GetRed(),
    GetBorderColor().GetGreen(), GetBorderColor().GetBlue()));
7.          SelectObject(hdc, hPen);
8.          POINT *points = new POINT [listOfCoords.size() / 2];
9.          iterator = listOfCoords.begin ();
10.         int aux = 0;
11.         if (window->GetOrigin() == TOP_LEFT)
12.               while (iterator != listOfCoords.end()){
13.                     points[aux].x = window->GetScreenX(*iterator);
14.                     iterator++;
15.                     points[aux].y = window->GetScreenY(*iterator);
16.                     iterator++;
17.                     aux++;
18.               }
19.         else if (window->GetOrigin() == BOTTOM_LEFT)
20.               while (iterator != listOfCoords.end()){
21.                     points[aux].x = window->GetScreenX(*iterator);
22.                     iterator++;
23.                     points[aux].y = (window->GetHeight())-window-
    >GetScreenY((*iterator));
24.                     iterator++;
25.                     aux++;
26.               }
27.         Polygon(hdc, points, listOfCoords.size() / 2);
28.         delete points;
29.   }
```

Fonte: Elaborado pelo autor.

4.2 Implementação do 2DMG para o Symbian OS

Na codificação das classes foi usada a IDE Carbide.c++[4]. Ela é construída pela Nokia em cima da conhecida Eclipse IDE e trabalha em conjunto com a Symbian SDK provendo também um simulador para verificar a execução das rotinas de cada classe.

O código-fonte da implementação do método draw() das classes Circle, Rectangle e Polygon pode ser visto nos respectivos Quadros 4.4, 4.5 e 4.6.

Quadro 4.4. Implementação do método draw() da classe Circle em C++ (Symbian).

```
1.    void Circle::draw(const Window_ *aWindow){
2.          CWindowGc& gc = aWindow->SystemGc();
3.          if (aWindow->GetOrigin() == TOP_LEFT)
4.                gc.SetOrigin(TPoint (0, 0));
5.          else if (aWindow->GetOrigin() == BOTTOM_LEFT)
```

[4] http://www.developer.nokia.com/Develop/Other_Technologies/Symbian_C++/, acessada em junho de 2011.

```
6.                gc.SetOrigin(TPoint (0, aWindow->GetHeight()));
7.          gc.SetPenColor(TRgb (iBorderColor->GetRed(), iBorderColor-
   >GetGreen(), iBorderColor->GetBlue()));
8.          gc.SetBrushColor(TRgb (iFillColor->GetRed(), iFillColor-
   >GetGreen(), iFillColor->GetBlue()));
9.          gc.SetBrushStyle(CGraphicsContext::ESolidBrush);
10.         if (iLineStyle == LINE_SOLID)
11.               gc.SetPenStyle(CGraphicsContext::ESolidPen);
12.         else if (iLineStyle == LINE_DASHED)
13.               gc.SetPenStyle(CGraphicsContext::EDashedPen);
14.         else if (iLineStyle == LINE_NULL)
15.               gc.SetPenStyle(CGraphicsContext::ENullPen);
16.         if (aWindow->GetOrigin() == TOP_LEFT)
17.               gc.DrawEllipse(TRect (aWindow->GetScreenX(iPosX-
   iRadius), aWindow->GetScreenY(iPosY-iRadius), aWindow-
   >GetScreenX(iPosX+iRadius), aWindow->GetScreenY(iPosY+iRadius)));
18.         if (aWindow->GetOrigin() == BOTTOM_LEFT)
19.               gc.DrawEllipse(TRect (aWindow->GetScreenX(iPosX-
   iRadius), -aWindow->GetScreenY(iPosY+iRadius), aWindow-
   >GetScreenX(iPosX+iRadius), -aWindow->GetScreenY(iPosY-iRadius)));
20.   }
```

Fonte: Elaborado pelo autor.

Quadro 4.5. Implementação do método draw() da classe Rectangle em C++ (Symbian).

```
1.    void Rectangle::draw(const Window_ *aWindow){
2.          CWindowGc& gc = aWindow->SystemGc();
3.          if (aWindow->GetOrigin() == TOP_LEFT)
4.                gc.SetOrigin(TPoint (0, 0));
5.          else if (aWindow->GetOrigin() == BOTTOM_LEFT)
6.                gc.SetOrigin(TPoint (0, aWindow->GetHeight()));
7.          gc.SetPenColor(TRgb (iBorderColor->GetRed(), iBorderColor-
   >GetGreen(), iBorderColor->GetBlue()));
8.          gc.SetBrushColor(TRgb (iFillColor->GetRed(), iFillColor-
   >GetGreen(), iFillColor->GetBlue()));
9.          gc.SetBrushStyle(CGraphicsContext::ESolidBrush);
10.         if (iLineStyle == LINE_SOLID)
11.               gc.SetPenStyle(CGraphicsContext::ESolidPen);
12.         else if (iLineStyle == LINE_DASHED)
13.               gc.SetPenStyle(CGraphicsContext::EDashedPen);
14.         else if (iLineStyle == LINE_NULL)
15.               gc.SetPenStyle(CGraphicsContext::ENullPen);
16.         TRect rect;
17.         if (aWindow->GetOrigin() == TOP_LEFT)
18.               rect = TRect (aWindow->GetScreenX(iPosX), aWindow-
   >GetScreenY(iPosY), aWindow->GetScreenX(iPosX+iWidth), aWindow-
   >GetScreenY(iPosY+iHeight));
19.         else if (aWindow->GetOrigin() == BOTTOM_LEFT)
20.               rect = TRect (aWindow->GetScreenX(iPosX), -aWindow-
   >GetScreenY(iPosY+iHeight), aWindow->GetScreenX(iPosX+iWidth),
   -aWindow->GetScreenY(iPosY));
21.         gc.DrawRect(rect);
22.   }
```

Fonte: Elaborado pelo autor.

Quadro 4.6. Implementação do método draw() da classe Polygon em C++ (Symbian).

```
1.    void Polygon::draw(const Window_ *aWindow){
2.          CWindowGc& gc = aWindow->SystemGc();
3.          if (aWindow->GetOrigin() == TOP_LEFT)
4.                gc.SetOrigin(TPoint (0, 0));
5.          else if (aWindow->GetOrigin() == BOTTOM_LEFT)
6.                gc.SetOrigin(TPoint (0, aWindow->GetHeight()));
7.          gc.SetPenColor(TRgb (iBorderColor->GetRed(), iBorderColor-
   >GetGreen(), iBorderColor->GetBlue()));
```

```
8.          gc.SetBrushColor(TRgb (iFillColor->GetRed(), iFillColor-
   >GetGreen(), iFillColor->GetBlue()));
9.          gc.SetBrushStyle(CGraphicsContext::ESolidBrush);
10.         TInt aux = 0;
11.         while (aux < iNumOfPoints){
12.               if (aWindow->GetOrigin() == TOP_LEFT)
13.                     iPoints[aux].SetXY(aWindow-
   >GetScreenX(iListOfPoints2->operator [](aux*2)), aWindow-
   >GetScreenY(iListOfPoints2->operator [](aux*2 + 1)));
14.               else if (aWindow->GetOrigin() == BOTTOM_LEFT)
15.                     iPoints[aux].SetXY(aWindow-
   >GetScreenX(iListOfPoints2->operator [](aux*2)), -aWindow-
   >GetScreenY(iListOfPoints2->operator [](aux*2 + 1)));
16.               aux++;
17.         }
18.         gc.DrawPolygon(iPoints, iNumOfPoints,
   CGraphicsContext::EWinding);
19.   }
```

Fonte: Elaborado pelo autor.

4.3 Implementação do 2DMG em Java ME

Para o desenvolvimento das classes em Java ME foi usada a NetBeans IDE[5]. Ela vem equipada com a mais recente Java ME SDK, que fornece a emulação de dispositivos e um conjunto de utilitários para desenvolvimento rápido de aplicações J2ME.

O código-fonte da implementação do método draw() das classes Circle, Rectangle e Polygon podem ser vistos nos respectivos Quadros 4.7, 4.8 e 4.9.

Quadro 4.7. Implementação do método draw() da classe Circle em Java ME.

```
1.    public void draw(Graphics g, Window window) {
2.        int WIDTH, HEIGHT;
3.        WIDTH = window.getScreenX(getPosX() + getRadius()) -
   window.getScreenX(getPosX());
4.        HEIGHT = window.getScreenY(getPosY() + getRadius()) -
   window.getScreenY(getPosY());
5.        g.setColor(getFillColor().getRed(), getFillColor().getGreen(),
   getFillColor().getBlue());
6.        if (window.getOrigin() == Config2DMG.TOP_LEFT) {
7.            g.fillArc(window.getPosX() + window.getScreenX(getPosX()),
   window.getPosY() + window.getScreenY(getPosY()), WIDTH, HEIGHT, 0,
   360);
8.        } else if (window.getOrigin() == Config2DMG.BOTTOM_LEFT) {
9.            g.fillArc(window.getPosX() + window.getScreenX(getPosX()),
   window.getPosY() + window.getHeight() - HEIGHT -
   window.getScreenY(getPosY()), WIDTH, HEIGHT, 0, 360);
10.       }
11.       if (getLineStyle() != Config2DMG.LINE_NULL) {
12.           g.setColor(getBorderColor().getRed(),
   getBorderColor().getGreen(), getBorderColor().getBlue());
13.           if (getLineStyle() == Config2DMG.LINE_SOLID) {
14.               g.setStrokeStyle(Graphics.SOLID);
15.           } else if (getLineStyle() == Config2DMG.LINE_DASH) {
16.               g.setStrokeStyle(Graphics.DOTTED);
17.           }
18.           if (window.getOrigin() == Config2DMG.TOP_LEFT) {
```

[5] http://www.netbeans.org/, acessada em maio de 2011.

```
19.              g.drawArc(window.getPosX() +
   window.getScreenX(getPosX()), window.getPosY() +
   window.getScreenY(getPosY()), WIDTH, HEIGHT, 0, 360);
20.          } else if (window.getOrigin() == Config2DMG.BOTTOM_LEFT) {
21.              g.drawArc(window.getPosX() +
   window.getScreenX(getPosX()), window.getPosY() + window.getHeight() -
   HEIGHT - window.getScreenY(getPosY()), WIDTH, HEIGHT, 0, 360);
22.          }
23.      }
24.  }
```

Fonte: Elaborado pelo autor.

Quadro 4.8. Implementação do método draw() da classe Rectangle em Java ME.

```
1.    public void draw(Graphics g, Window window) {
2.        int WIDTH, HEIGHT;
3.        WIDTH = window.getScreenX(getPosX() + mWidth) -
   window.getScreenX(getPosX());
4.        HEIGHT = window.getScreenY(getPosY() + mHeight) -
   window.getScreenY(getPosY());
5.        g.setColor(getFillColor().getRed(), getFillColor().getGreen(),
   getFillColor().getBlue());
6.        if (window.getOrigin() == Config2DMG.TOP_LEFT) {
7.            g.fillRect(window.getPosX() + window.getScreenX(getPosX()),
   window.getPosY() + window.getScreenY(getPosY()), WIDTH, HEIGHT);
8.        } else if (window.getOrigin() == Config2DMG.BOTTOM_LEFT) {
9.            g.fillRect(window.getPosX() + window.getScreenX(getPosX()),
   window.getPosY() + window.getHeight() - HEIGHT -
   window.getScreenY(getPosY()), WIDTH, HEIGHT);
10.       }
11.       if (getLineStyle() != Config2DMG.LINE_NULL) {
12.           g.setColor(getBorderColor().getRed(),
   getBorderColor().getGreen(), getBorderColor().getBlue());
13.           if (getLineStyle() == Config2DMG.LINE_SOLID) {
14.               g.setStrokeStyle(Graphics.SOLID);
15.           } else if (getLineStyle() == Config2DMG.LINE_DASH) {
16.               g.setStrokeStyle(Graphics.DOTTED);
17.           }
18.           if (window.getOrigin() == Config2DMG.TOP_LEFT) {
19.               g.drawRect(window.getPosX() +
   window.getScreenX(getPosX()), window.getPosY() +
   window.getScreenY(getPosY()), WIDTH, HEIGHT);
20.           } else if (window.getOrigin() == Config2DMG.BOTTOM_LEFT) {
21.               g.drawRect(window.getPosX() +
   window.getScreenX(getPosX()), window.getPosY() + window.getHeight() -
   HEIGHT - window.getScreenY(getPosY()), WIDTH, HEIGHT);
22.           }
23.       }
24.   }
```

Fonte: Elaborado pelo autor.

Quadro 4.9. Implementação do método draw() da classe Polygon em Java ME.

```
1.    public void draw(Graphics g, Window window) {
2.        g.setColor(getFillColor().getRed(), getFillColor().getGreen(),
   getFillColor().getBlue());
3.        g.setColor(getBorderColor().getRed(),
   getBorderColor().getGreen(), getBorderColor().getBlue());
4.        if (window.getOrigin() == Config2DMG.TOP_LEFT) {
5.            for (int i = 0; i < mNumElementos - 2;) {
6.                g.drawLine(window.getPosX() +
   window.getScreenX(mListaDeCoordenadas[i++]),
7.                        window.getPosY() +
   window.getScreenY(mListaDeCoordenadas[i++]),
8.                        window.getPosX() +
```

```
    window.getScreenX(mListaDeCoordenadas[i]),
9.                        window.getPosY() +
    window.getScreenY(mListaDeCoordenadas[i + 1]));
10.           }
11.           g.drawLine(window.getPosX() +
    window.getScreenX(mListaDeCoordenadas[mNumElementos - 2]),
12.                   window.getPosY() +
    window.getScreenY(mListaDeCoordenadas[mNumElementos - 1]),
13.                   window.getPosX() +
    window.getScreenX(mListaDeCoordenadas[0]),
14.                   window.getPosY() +
    window.getScreenY(mListaDeCoordenadas[1]));
15.       } else if (window.getOrigin() == Config2DMG.BOTTOM_LEFT) {
16.           for (int i = 0; i < mNumElementos - 2;) {
17.               g.drawLine(window.getPosX() +
    window.getScreenX(mListaDeCoordenadas[i++]),
18.                       window.getPosY() + window.getHeight() -
    window.getScreenY(mListaDeCoordenadas[i++]),
19.                       window.getPosX() +
    window.getScreenX(mListaDeCoordenadas[i]),
20.                       window.getPosY() + window.getHeight() -
    window.getScreenY(mListaDeCoordenadas[i + 1]));
21.           }
22.           g.drawLine(window.getPosX() +
    window.getScreenX(mListaDeCoordenadas[mNumElementos - 2]),
23.                   window.getPosY() + window.getHeight() -
    window.getScreenY(mListaDeCoordenadas[mNumElementos - 1]),
24.                   window.getPosX() +
    window.getScreenX(mListaDeCoordenadas[0]),
25.                   window.getPosY() + window.getHeight() -
    window.getScreenY(mListaDeCoordenadas[1]));
26.       }
27.   }
```

Fonte: Elaborado pelo autor.

4.4 Implementação do 2DMG para o iOS

Para o desenvolvimento das classes na linguagem Objective-C foi usada a IDE Xcode[6], um conjunto de ferramentas para construção de aplicações Mac OS X e iOS. Ela trabalha em conjunto com a iOS SDK provendo um emulador para o iPhone usado para verificar a execução das rotinas de cada classe.

O código-fonte da implementação do método draw() das classes Circle, Rectangle e Polygon podem ser vistos nos respectivos Quadros 4.10, 4.11 e 4.12.

Quadro 4.10. Implementação do método draw() da classe Circle em Obj-C (iOS).

```
1.    -(void) draw: (Window*)window {
2.          CGContextRef context = UIGraphicsGetCurrentContext();
3.          if(lineStyle == LINE_DASHED) {
4.                const CGFloat dashPattern[2] = {3,2};
5.                CGContextSetLineDash(context, 0, dashPattern, 2);
6.          }
7.          CGContextSetRGBStrokeColor(context, [borderColor
    getRed]/255.0, [borderColor getGreen]/255.0, [borderColor
    getBlue]/255.0, 1.0);
8.          CGContextSetRGBFillColor(context, [fillColor getRed]/255.0,
    [fillColor getGreen]/255.0, [fillColor getBlue]/255.0, 1.0);
```

[6] http://developer.apple.com/xcode/, acessada em junho de 2011.

```
9.          if([window getOrigin] == TOP_LEFT) {
10.               CGContextAddEllipseInRect(context, CGRectMake([window
    getScreenX:posX-radius], [window getScreenY:posY-radius], [window
    getScreenX:posX+radius]-[window getScreenX:posX-radius], [window
    getScreenY:posY+radius]-[window getScreenY:posY-radius]));
11.         }else if([window getOrigin] == BOTTOM_LEFT) {
12.               CGContextAddEllipseInRect(context, CGRectMake([window
    getScreenX:posX-radius], [window getHeight]-[window getScreenY:posY-
    radius], [window getScreenX:posX+radius]-[window getScreenX:posX-
    radius], [window getScreenY:posY-radius]-[window
    getScreenY:posY+radius]));
13.         }
14.         CGContextDrawPath(context, kCGPathFillStroke);
15.         CGContextSetLineDash(context, 0, NULL, 0);
16.   }
```

Fonte: Elaborado pelo autor.

Quadro 4.11. Implementação do método draw() da classe Rectangle em Obj-C (iOS).

```
1.    -(void) draw: (Window*)window {
2.          CGContextRef context = UIGraphicsGetCurrentContext();
3.          if(lineStyle == LINE_DASHED) {
4.                CGFloat dashPattern[2] = {3,2};
5.                CGContextSetLineDash(context, 0, dashPattern, 2);
6.          }
7.          CGContextSetRGBStrokeColor(context, [borderColor
    getRed]/255.0, [borderColor getGreen]/255.0, [borderColor
    getBlue]/255.0, 1.0);
8.          CGContextSetRGBFillColor(context, [fillColor getRed]/255.0,
    [fillColor getGreen]/255.0, [fillColor getBlue]/255.0, 1.0);
9.          if([window getOrigin] == TOP_LEFT) {
10.               CGContextAddRect(context, CGRectMake([window
    getScreenX:posX], [window getScreenY:posY], [window
    getScreenX:posX+width]-[window getScreenX:posX], [window
    getScreenY:posY+height]-[window getScreenY:posY]));
11.         } else if([window getOrigin] == BOTTOM_LEFT) {
12.               CGContextAddRect(context, CGRectMake([window
    getScreenX:posX], [window getHeight]-[window getScreenY:posY], [window
    getScreenX:posX+width]-[window getScreenX:posX], [window
    getScreenY:posY]-[window getScreenY:posY+height]));
13.         }
14.         CGContextDrawPath(context, kCGPathFillStroke);
15.         CGContextSetLineDash(context, 0, NULL, 0);
16.   }
```

Fonte: Elaborado pelo autor.

Quadro 4.12. Implementação do método draw() da classe Polygon em Obj-C (iOS).

```
1.    -(void) draw: (Window*)window {
2.          CGContextRef context = UIGraphicsGetCurrentContext();
3.          if(lineStyle == LINE_DASHED) {
4.                CGFloat dashPattern[2] = {3,2};
5.                CGContextSetLineDash(context, 0, dashPattern, 2);
6.          }
7.          CGContextSetRGBStrokeColor(context, [borderColor
    getRed]/255.0, [borderColor getGreen]/255.0, [borderColor
    getBlue]/255.0, 1.0);
8.          CGContextSetRGBFillColor(context, [fillColor getRed]/255.0,
    [fillColor getGreen]/255.0, [fillColor getBlue]/255.0, 1.0);
9.         NSEnumerator* enumerator = [listOfCoordinates
    objectEnumerator];
10.         NSNumber* pointX = [enumerator nextObject];
11.         NSNumber* pointY = [enumerator nextObject];
12.         if([window getOrigin] == TOP_LEFT) {
13.               CGContextMoveToPoint(context, [window getScreenX:
```

```
    [pointX floatValue]], [window getScreenY:[pointY floatValue]]);
14.                 while(pointX = [enumerator nextObject]){
15.                         pointY = [enumerator nextObject];
16.                         CGContextAddLineToPoint(context, [window
    getScreenX:[pointX floatValue]], [window getScreenY:[pointY
    floatValue]]);
17.                 }
18.           } else if([window getOrigin] == BOTTOM_LEFT) {
19.                 CGContextMoveToPoint(context, [window getScreenX:
    [pointX floatValue]], [window getHeight]-[window getScreenY:[pointY
    floatValue]]);
20.                 while(pointX = [enumerator nextObject]){
21.                         pointY = [enumerator nextObject];
22.                         CGContextAddLineToPoint(context, [window
    getScreenX:[pointX floatValue]], [window getHeight]-[window getScreenY:
    [pointY floatValue]]);
23.                 }
24.           }
25.           CGContextClosePath(context);
26.           CGContextDrawPath(context, kCGPathFillStroke);
27.           CGContextSetLineDash(context, 0, NULL, 0);
28.           [pointX release];
29.           [pointY release];
30.    }
```

Fonte: Elaborado pelo autor.

4.5 Interface padrão para o programador

Um dos objetivos do 2DMG é prover uma interface padrão para o programador, independente da plataforma que ele esteja usando em sua aplicação móvel. Para isso, ele deve usar a implementação do *middleware* correspondente ao seu ambiente de desenvolvimento.

Para testar as classes concretas desenvolvidas neste projeto, criou-se um código-teste para cada implementação do 2DMG. Observou-se que mesmo usando linguagens e SOs diferentes, os códigos ficaram praticamente iguais, exceto por alguns detalhes próprios de cada linguagem. No intuito de evitar muitos quadros extensos nesta dissertação, opta-se por mostrar as construções em C++ (Symbian) e Objective-C (iOS), já que as outras duas são uma também em C++ (Windows Mobile) e a outra mais popular (J2ME).

Os trechos dos códigos-teste presentes nos Quadros 4.13 e 4.14 possuem a seguinte estrutura: cria-se o objeto fábrica (FigFactory), que é usado para instanciar os elementos gráficos (figuras) e a janela; essas figuras são adicionadas às listas de figuras de duas janelas com o método addFig(); finalmente, chama-se o método draw() das janelas que, por sua vez, percorre a lista de figuras invocando o método draw() de cada uma.

4.5.1 Teste C++ (Symbian)

Quadro 4.13. Trecho do código teste em C++ (Symbian).

```
1.    // O objeto fabrica
2.    FigFactory *fabrica = FigFactory::GetInstanceL();
3.
4.    // Criando algumas cores
5.    Color *cor_1 = Color::NewL(0, 0, 0);                // Preto
6.    Color *cor_2 = Color::NewL(255, 255, 255);          // Branco
7.    Color *cor_3 = Color::NewL(255, 0, 0);              // Vermelho
8.    Color *cor_4 = Color::NewL(0, 255, 0);              // Verde
9.    Color *cor_5 = Color::NewL(0, 0, 255);              // Azul
10.   Color *cor_6 = Color::NewL(180, 180, 180);          // Cinza claro
11.   Color *cor_7 = Color::NewL(255, 175, 175);          // Vermelhoclaro
12.   Color *cor_8 = Color::NewL(175, 255, 175);          // Verde claro
13.   Color *cor_9 = Color::NewL(175, 175, 255);          // Azul claro
14.   Color *cor_10 = Color::NewL(255, 126, 0);           // Laranja
15.
16.   // Criando algumas figuras usando a FigFactory
17.   Circle * circulo = fabrica->CreateCircleL(-1.0, 0.0, 1.0, cor_2,
    cor_3, LINE_SOLID);
18.   Line * linha1 = fabrica->CreateLineL(-0.5, 0.0, 0.5, 1.1, cor_5,
    LINE_SOLID);
19.   Line * linha2 = fabrica->CreateLineL (0.5, 1.1, 1.0, 2.2, cor_5,
    LINE_DASHED);
20.   Rectangle * retangulo = fabrica->CreateRectangleL(-0.666, 1.43,
    1.23, 0.873, cor_7, cor_3, LINE_SOLID);
21.
22.   Polygon * poligono = fabrica->CreatePolygonL(-0.8, 1.0, cor_8,
    cor_4, LINE_SOLID);
23.   poligono->AddPoint (-0.8, 1.0);
24.   poligono->AddPoint (-0.2, 1.0);
25.   poligono->AddPoint (0.0, 1.3);
26.   poligono->AddPoint (-0.2, 1.6);
27.   poligono->AddPoint (-0.35, 1.15);
28.
29.   Elipse * elipse = fabrica->CreateElipseL(0.5, 0.75, 1.9, 1.15,
    cor_9, cor_5, LINE_DASHED);
30.   Rectangle * caixa1 = fabrica->CreateRectangleL(-1.5, -3.15, 3, 2,
    cor_2, cor_1, LINE_SOLID);
31.   Rectangle * caixa2 = fabrica->CreateRectangleL(-1.3, -2.95, 2.6,
    1.6, cor_6, cor_1, LINE_DASHED);
32.   Text * texto1 = fabrica->CreateTextL (-1.05, -2.90, cor_3,
    FONT_SMALL, "2DMG");
33.   Text * texto2 = fabrica->CreateTextL (-1.05, -2.55, cor_1,
    FONT_MEDIUM, "C++");
34.   Text * texto3 = fabrica->CreateTextL (-1.05, -2.05, cor_5,
    FONT_LARGE, "Symbian");
35.
36.   // Criando uma composição de figuras usando a FigFactory
37.   Composite * CaixaDeTextos = fabrica->CreateCompositeL();
38.
39.   // Criando duas Janelas usando a FigFactory
40.   iJanela1 = fabrica->CreateWindowL(49, 100, TOP_LEFT, TOP_LEFT,
    cor_6, -2.0, 2.0, -4.0, 4.0);
41.   iJanela2 = fabrica->CreateWindowL(49, 100, TOP_RIGHT, BOTTOM_LEFT,
    cor_6, -1.0, 1.0, -2.0, 2.0);
42.
43.   // Inserindo as figuras no composite
44.   CaixaDeTextos->AddFig(caixa1);
45.   CaixaDeTextos->AddFig(caixa2);
46.   CaixaDeTextos->AddFig(texto1);
47.   CaixaDeTextos->AddFig(texto2);
48.   CaixaDeTextos->AddFig(texto3);
49.
```

```
50.   // Inserindo as figuras nas janelas
51.   iJanela1->AddFig (circulo);
52.   iJanela1->AddFig (linha1);
53.   iJanela1->AddFig (linha2);
54.   iJanela1->AddFig (retangulo);
55.   iJanela1->AddFig (poligono);
56.   iJanela1->AddFig (elipse);
57.   iJanela1->AddFig (CaixaDeTextos);
58.
59.   iJanela2->AddFig (circulo);
60.   iJanela2->AddFig (linha1);
61.   iJanela2->AddFig (linha2);
62.   iJanela2->AddFig (retangulo);
63.   iJanela2->AddFig (poligono);
64.   iJanela2->AddFig (elipse);
65.   iJanela2->AddFig (CaixaDeTextos);
66.
67.   // Desenhando as janelas
68.   iJanela1.Draw();
69.   iJanela2.Draw();
```

Fonte: Elaborado pelo autor.

4.5.2 Teste Objective-C (iOS)

Quadro 4.14. Trecho do código teste em Obj-C (iOS).

```
1.    // O objeto fabrica
2.    FigFactory *fabrica = [[FigFactory alloc] initFigFactory];
3.
4.    // Criando algumas cores
5.    Color *cor_1 = [[Color alloc] initColor: 0: 0: 0];
6.    Color *cor_2 = [[Color alloc] initColor: 255: 255: 255];
7.    Color *cor_3 = [[Color alloc] initColor: 255: 0: 0];
8.    Color *cor_4 = [[Color alloc] initColor: 0: 255: 0];
9.    Color *cor_5 = [[Color alloc] initColor: 0: 0: 255];
10.   Color *cor_6 = [[Color alloc] initColor: 100: 100: 100];
11.   Color *cor_7 = [[Color alloc] initColor: 255: 175: 175];
12.   Color *cor_8 = [[Color alloc] initColor: 175: 255: 175];
13.   Color *cor_9 = [[Color alloc] initColor: 175: 175: 255];
14.   Color *cor_10 = [[Color alloc] initColor: 255: 126: 0];
15.
16.   // Criando algumas figuras usando a FigFactory
17.   Circle * circulo = [fabrica createCircle: -1.0: 0.0: 1.0: cor_2:
   cor_3: LINE_SOLID];
18.   Line * linha1 = [fabrica createLine: -0.5: 0.0: 0.5: 1.1: cor_5:
   LINE_SOLID];
19.   Line * linha2 = [fabrica createLine: 0.5: 1.1: 1.0: 2.2: cor_5:
   LINE_DASHED];
20.   Rectangle * retangulo = [fabrica createRectangle: -0.666: 1.43:
   1.23: 0.873: cor_7: cor_3: LINE_SOLID];
21.
22.   Polygon * poligono = [fabrica createPolygon: -0.8: 1.0: cor_8:
   cor_4: LINE_SOLID];
23.   [poligono addPoint: -0.8: 1.0];
24.   [poligono addPoint: -0.2: 1.0];
25.   [poligono addPoint: 0.0: 1.3];
26.   [poligono addPoint: -0.2: 1.6];
27.   [poligono addPoint: -0.35: 1.15];
28.
29.   Ellipse * elipse = [fabrica createEllipse: 0.5: 0.75: 1.9: 1.15:
   cor_9: cor_5: LINE_DASHED];
30.   Rectangle * caixa1 = [fabrica createRectangle: -1.5: -3.15: 3: 2:
   cor_2: cor_1: LINE_SOLID];
31.   Rectangle * caixa2 = [fabrica createRectangle: -1.3: -2.95: 2.6:
   1.6: cor_6: cor_1: LINE_DASHED];
```

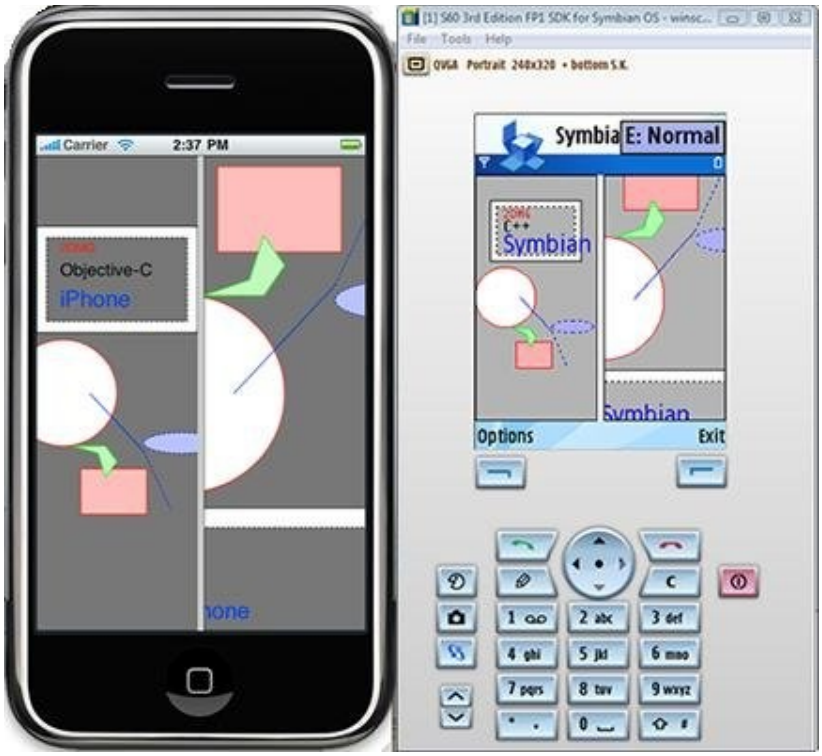
```
32.   Text * texto1 = [fabrica createText: -1.05: -2.90: cor_3:
   FONT_SMALL: @"2DMG"];
33.   Text * texto2 = [fabrica createText: -1.05: -2.55: cor_1:
   FONT_MEDIUM: @"Objective-C"];
34.   Text * texto3 = [fabrica createText: -1.05: -2.05: cor_5:
   FONT_LARGE: @"iPhone"];
35.
36.   // Criando uma composição de figuras usando a FigFactory
37.   Composite * CaixaDeTextos = [fabrica createComposite];
38.
39.   // Criando duas Janelas usando a FigFactory
40.   Window* iJanela1 = [fabrica createWindow: 49: 100: TOP_LEFT:
   TOP_LEFT: cor_6: -1.5: 1.5: -4.5: 4.5];
41.   Window* iJanela2 = [fabrica createWindow: 49: 100: TOP_RIGHT:
   BOTTOM_LEFT: cor_6: -0.8: 0.8: -2.4: 2.4];
42.
43.   // Inserindo as figuras no composite
44.   [CaixaDeTextos addFig: caixa1];
45.   [CaixaDeTextos addFig: caixa2];
46.   [CaixaDeTextos addFig: texto1];
47.   [CaixaDeTextos addFig: texto2];
48.   [CaixaDeTextos addFig: texto3];
49.
50.   // Inserindo as figuras nas janelas
51.   [iJanela1 addFig: circulo];
52.   [iJanela1 addFig: linha1];
53.   [iJanela1 addFig: linha2];
54.   [iJanela1 addFig: retangulo];
55.   [iJanela1 addFig: poligono];
56.   [iJanela1 addFig: elipse];
57.   [iJanela1 addFig: CaixaDeTextos];
58.
59.   [iJanela2 addFig: circulo];
60.   [iJanela2 addFig: linha1];
61.   [iJanela2 addFig: linha2];
62.   [iJanela2 addFig: retangulo];
63.   [iJanela2 addFig: poligono];
64.   [iJanela2 addFig: elipse];
65.   [iJanela2 addFig: CaixaDeTextos];
66.
67.   // Desenhando as janelas
68.   [self.view addSubview:iJanela1];
69.   [self.view addSubview:iJanela2];
70.   [self.view setNeedsDisplay];
```

Fonte: Elaborado pelo autor.

4.5.3 Resultado da execução

O resultado da execução dos códigos-teste nos emuladores das respectivas SDKs, iOS e Symbian pode ser visto na Figura 4.1. Ela mostra em cada dispositivo as duas janelas e os elementos gráficos que fazem parte desta primeira versão do 2DMG, como círculo, linha, retângulo, polígono, elipse, texto e a composição de figuras (composite).

Figura 4.1. Execução do código-teste nos emuladores da iOS SDK (esquerda) e da Symbian SDK (direita).



Fonte: Elaborada pelo autor.

4.6 Uma camada *versus* duas camadas

Além de proporcionar uma interface padronizada para criação de elementos gráficos 2D, pode-se avaliar o auxílio do *middleware* ao desenvolvedor utilizando outros parâmetros.

Na Engenharia de Software existem algumas medidas comumente usadas na avaliação de programas e seu desenvolvimento. Em (PRESSMAN, 2000) essas métricas são classificadas como:

- Diretas: são obtidas através de aspectos de custo e esforço. Exemplos: linhas de código, velocidade de execução e tamanho de memória etc.
- Indiretas: traduzem uma ideia subjetiva e são obtidas indiretamente por meio da agregação de métricas diretas. Exemplos: as características de qualidade, como funcionalidade, eficiência, manutenibilidade, confiabilidade, entre outras.

Para comparar a abordagem em uma camada (desenvolvimento na linguagem nativa) e em duas camadas (utilizando o 2DMG como camada adaptativa), foi usada

a métrica direta linhas de código para traduzir o esforço do programador. As outras métricas diretas, velocidade de execução e tamanho de memória, não foram utilizadas pois os dados obtidos não foram relevantes, já que a aplicação móvel é pequena e usa poucos recursos computacionais.

Os Quadros 4.1 a 4.12, que mostram a construção do método de desenho dos elementos gráficos Circle, Rectangle e Polygon em suas respectivas linguagens nativas, foram utilizados na contagem das linhas de código para se desenhar uma figura na tela ou *display* do dispositivo. A primeira linha e a última não foram consideradas na medida, pois fazem parte da definição do método.

Na contagem de linhas gastas para se desenhar uma figura na tela utilizando o *middleware*, considerou-se as seguintes rotinas do código, presente no Quadro 4.13: a criação da figura pela fábrica (linha 17), a adição do elemento à janela (linha 51) e a chamada do método draw (linha 68). Para o elemento Polygon ainda foram contadas as rotinas para se adicionar dois pontos (linhas 23 e 24), o mínimo para se ter uma figura do tipo.

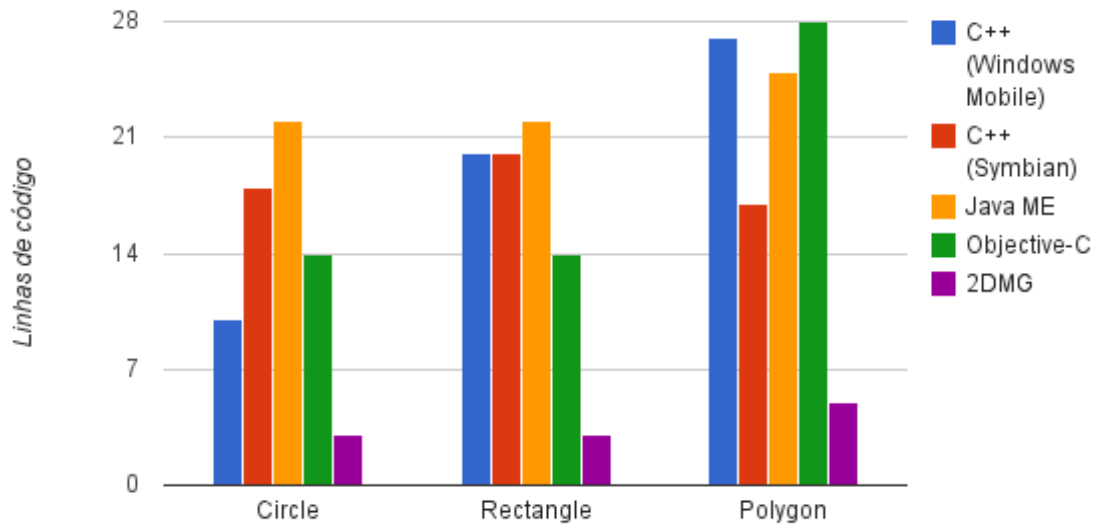
O resultado dessas medidas e a comparação entre o desenvolvimento utilizando a linguagem nativa e o 2DMG podem ser vistas na Tabela 4.1. Considerando que em uma aplicação real dificilmente teríamos apenas uma figura (círculo, retângulo ou polígono), a diferença entre a implementação em uma camada e em duas camadas ficaria ainda maior. A representação gráfica dessa comparação pode ser vista na Figura 4.2.

Tabela 4.1. Número de linhas de código utilizado para desenhar uma figura com cada linguagem nativa e com o 2DMG.

	Circle	Rectangle	Polygon
C++ (Windows Mobile)	10	20	27
C++ (Symbian)	18	20	17
Java ME	22	22	25
Objective-C	14	14	28
2DMG	3	3	5

Fonte: Elaborada pelo autor.

Figura 4.2. Representação gráfica da Tabela 4.1.



Fonte: Elaborada pelo autor.

O próximo capítulo relatará um estudo de caso que usa o 2DMG para implementar parte de um aplicativo que faz uso de recursos gráficos 2D, permitindo sua execução no iOS. Esse aplicativo, chamado de **Desmet**, é uma ferramenta computacional aplicada à área de engenharia estrutural do Departamento de Engenharia Civil da UFV e amplamente usada no ensino de graduação.

5 EXPERIMENTAÇÃO NO IOS

No intuito de validar a solução proposta no Capítulo 3 desta dissertação, resolvemos selecionar um software acadêmico que faz uso de recursos gráficos 2D e realizar sua adaptação para ser executado em um dispositivo móvel, fazendo uso do 2DMG para facilitar a sua implementação.

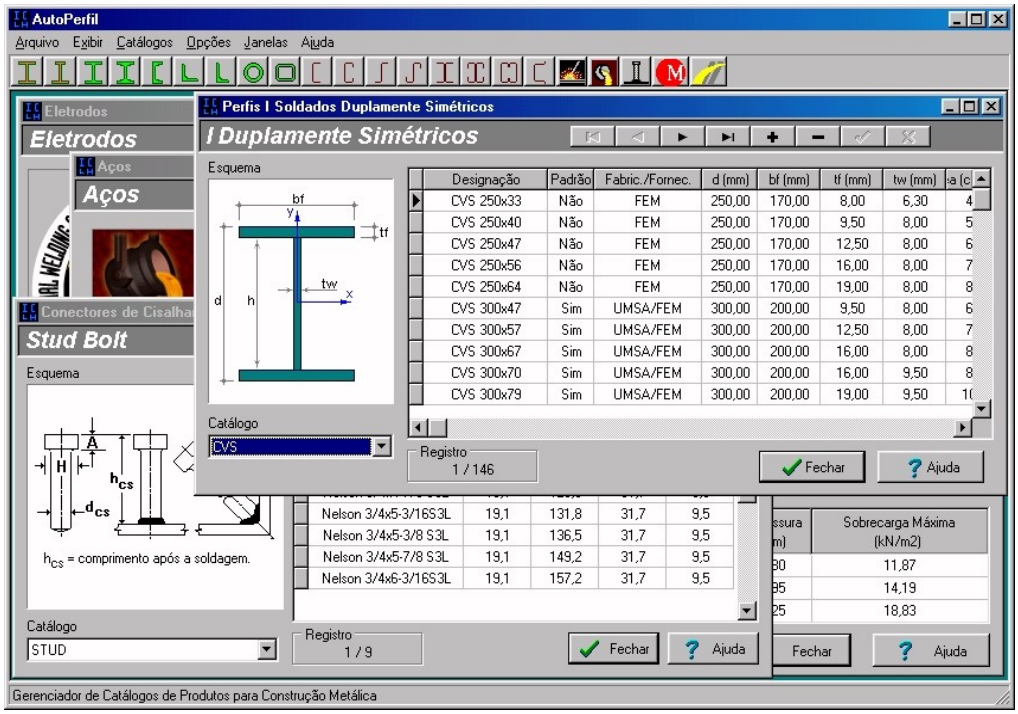
Como o sistema operacional **iOS** (iPhone) é o mais diferente em termos da linguagem nativa e plataforma de desenvolvimento, ele foi selecionado como SO base para esta experimentação.

5.1 A ferramenta computacional Desmet

O programa **Desmet**, usado como estudo de caso, foi escolhido dentro de uma parceria com o Laboratório de Computação Aplicada à Engenharia Estrutural (LCEE) do Departamento de Engenharia Civil da UFV (DEC-UFV), e que serve hoje de apoio a projetos técnicos, ensino e pesquisa na área de estruturas metálicas.

O LCEE, diante da possibilidade de produzir aplicativos na área de construção metálica, desenvolveu primeiramente um software capaz de gerenciar uma base de dados a ser utilizada por outros programas de dimensionamento. Assim, foi desenvolvido um gerenciador de catálogos de produtos para construção metálica denominado AutoPerfil (Figura 5.1).

Figura 5.1. Aspecto da interface do AutoPerfil.



Fonte: Página do software na Web.

O AutoPerfil não apenas armazena as dimensões das seções dos perfis metálicos, como também calcula uma série de propriedades mecânicas necessárias à verificação do dimensionamento de elementos estruturais.

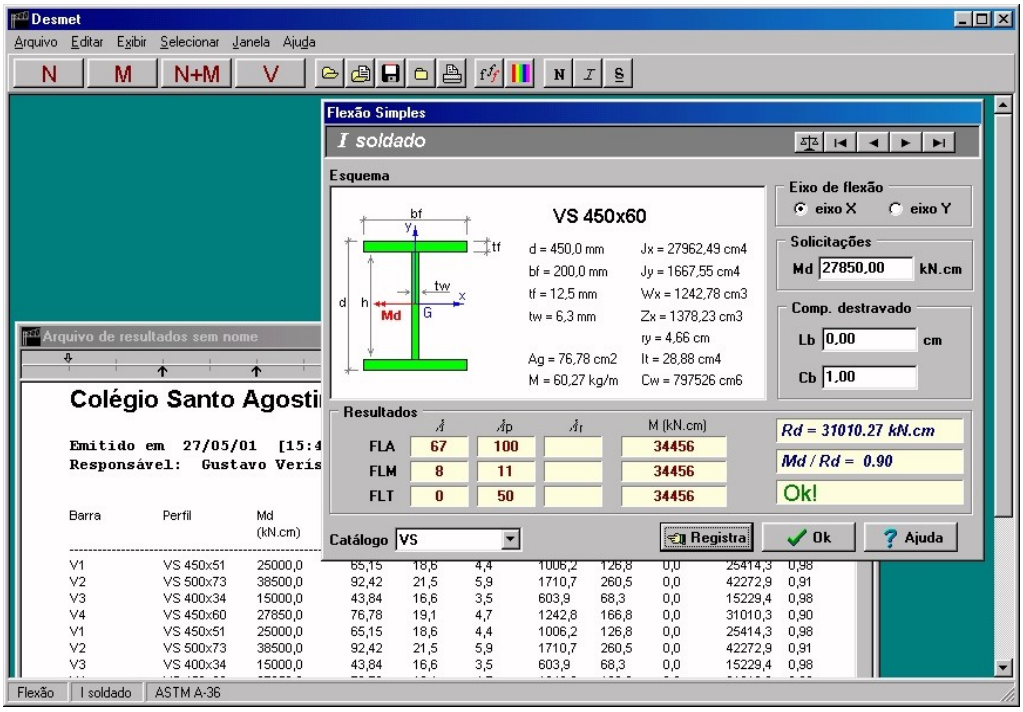
A potencialidade de uma aplicação como o AutoPerfil ficou evidenciada logo após o seu desenvolvimento, quando a AÇOMINAS o utilizou para determinar as propriedades mecânicas de sua nova linha de perfis, quando da conclusão do seu laminador, e a CELSA (Cia Espanhola de Laminação), uma siderúrgica de Barcelona (Espanha), contratou o desenvolvimento de uma versão personalizada chamada PERFIL-CELSA (VERÍSSIMO et al., 2002).

O Desmet (Software para Dimensionamento de Elementos Estruturais Metálicos) é um dos softwares desenvolvido no LCEE que utiliza a base de dados do AutoPerfil. Ele foi desenvolvido com base no Método dos Estados Limites, segundo a NBR 8800:1986. Esta norma fixa as condições exigíveis para projetos de estruturas de aço de edifícios destinados à habitação, ao uso comercial e industrial e a edifícios públicos (VERÍSSIMO & PAES, 2000). Um aspecto de sua interface pode ser vista na Figura 5.2. Vale destacar alguns dos recursos mais importantes dessa ferramenta, a saber:

- Processo rápido e automático de dimensionamento de elementos estruturais.

- Exibição dos resultados referentes às verificações de resistência e estabilidade associadas a cada estado limite aplicável.
- Pesquisa no catálogo corrente um perfil com menor peso que atende às condições prescritas.
- Geração de relatório com as respostas do dimensionamento (memorial de cálculo).
- Ajuda online sensível ao contexto.

Figura 5.2. Aspecto da interface do software Desmet.



Fonte: VERÍSSIMO & PAES, 2000.

5.2 Implementação do Desmet para o iOS

5.2.1 Necessidades de adaptação

Para que o uso de ferramentas computacionais como essas e outras similares, de diversas outras áreas das ciências, possa ser feito de modo mais ubíquo e com maior segurança, torna-se necessário disponibilizá-las em dispositivos portáteis menores. Para que isso seja feito, é necessário realizar uma série de adaptações às suas interfaces (formas de interação entre a pessoa e o computador) e ao uso de recursos computacionais, que são em geral muito limitados nesses dispositivos.

A ferramenta Desmet descrita anteriormente é uma boa candidata para tais adaptações tanto por sua alta aplicação técnica, como por ser um programa considerado pequeno em termos de linhas de código. Ainda assim, sua adaptação para dispositivos móveis é um grande desafio, envolvendo diversas áreas da Ciência da Computação, especialmente no tocante ao uso dos recursos de computação gráfica e dos métodos numéricos empregados, além da interface homem-computador (IHC), como mencionado acima.

5.2.2 Requisitos funcionais e não-funcionais

A ferramenta foi desenvolvida para o iPhone atendendo os seguintes requisitos **funcionais**:

- R.F.1 – O aplicativo deve exibir uma tabela com os dados das dimensões do perfil.
- R.F.2 – O aplicativo deve exibir uma imagem do perfil.

E os seguintes requisitos **não-funcionais**:

- R.N.F.3 – As imagens devem ser desenhadas utilizando os recursos do 2DMG.
- R.N.F.4 – As imagens devem ser dinâmicas, ou seja, devem sofrer modificações de acordo com os dados que estão sendo visualizados pelo usuário na tabela.

5.2.3 Desenvolvimento

Após identificar as funções do Desmet que se aplicam ao uso em dispositivos móveis, foi desenvolvida uma versão reduzida do mesmo utilizando as classes do 2DMG para o desenho do perfil e exibição dos dados de suas dimensões. Relata-se nesta sessão alguns aspectos importantes da implementação.

Para melhor representar os perfis no Desmet Mobile, foi preciso criar mais duas classes concretas para representarem os elementos gráficos seta reta unidirecional e seta reta bidirecional, denominadas **Arrow1** e **Arrow2** respectivamente. Ambas não estão definidas na arquitetura original do 2DMG, mas foram facilmente criadas seguindo o padrão do modelo.

Arrow1 implementa as funções de seta e foi definida como herança da classe Line (presente no modelo padrão), passando como parâmetros a posição X e Y do

ponto inicial, a posição X e Y do ponto final, a cor e o estilo da linha (sólida ou pontilhada). Por padrão, a cabeça da seta fica localizada no ponto final e é calculada no construtor da classe.

Arrow2 implementa a seta com duas cabeças, passando como parâmetros a posição X e Y do ponto inicial, a posição X e Y do ponto final, a cor e o estilo da linha (sólida ou pontilhada). Obviamente, as cabeças são desenhadas nos pontos inicial e final da seta. A implementação é semelhante a classe Arrow1, acrescentando-se apenas o cálculo da cabeça no ponto inicial da seta.

No lugar de figuras estáticas utilizadas na versão desktop, foi desenvolvido um **interpretador** que permite que o desenho dos perfis seja feito através de um *script* e de forma dinâmica, utilizando os dados das dimensões especificadas na tabela para cada perfil. Dessa forma, ele é desenhado de forma proporcional às suas medições reais.

Fez-se necessário definir algumas regras e comandos para descrever um desenho a ser interpretado, a saber:

1. Uma figura deve ser descrita com um comando seguido por seus parâmetros.
2. O comando e os parâmetros devem ser separados por espaços, por exemplo: LINE 15 15 15 5 0 0 0 0 (comando LINE, que representa uma linha, e seus parâmetros).
3. Os valores dos atributos do perfil podem ser referenciados colocando o nome do atributo entre “<” e “>” (parênteses angulados), por exemplo:
   LINE 15 <h> 0 <h> 0 0 0 0.
4. Cálculos simples como adição, subtração, multiplicação e divisão podem ser feitos entre o símbolo “#” (cerquilha), por exemplo: #25/7+2#.
5. As operações de cálculo devem permitir a utilização das referências de atributos do perfil, por exemplo: LINE 15 #<h>+15# 0 #<h>+15# 0 0 0 0 (dois parâmetros são informados referenciando um atributo do perfil (h) mais o valor 15).
6. Os comandos disponíveis são:

- ARROW1 (seta unidirecional): posição inicial X e Y, posição final X e Y (onde ficará a cabeça da seta), a cor no padrão RGB e o estilo de linha (sólida ou tracejada).
  `ARROW1 PosX PosY PosX2 PosY2 Red Green Blue LineStyle`

- ARROW2 (seta bidirecional, com a cabeça em ambas as pontas): posição inicial X e Y, posição final X e Y, a cor no padrão RGB e o estilo de linha (sólida ou tracejada).
  `ARROW2 PosX PosY PosX2 PosY2 Red Green Blue LineStyle`
- CIRCLE (círculo): posição X e Y do centro do círculo, raio do círculo, cor no padrão RGB e o estilo de linha.
  `CIRCLE PosX PosY Radius Red Green Blue LineStyle`
- LINE (linha): posição inicial X e Y, posição final X e Y, cor no padrão RGB e o estilo de linha.
  `LINE PosX PosY PosX2 PosY2 Red Green Blue LineStyle`
- POINT (ponto): posição X e Y, cor no padrão RGB e o estilo de linha.
  `POINT PosX PosY Red Green Blue LineStyle`
- RECT (retângulo): posição inicial X e Y, largura, altura, cor de preenchimento do retângulo no padrão RGB, cor do contorno no padrão RGB e o estilo de linha.
  `RECT PosX PosY Width Height Red Green Blue Red Green Blue LineStyle`
- TEXT (texto): posição X e Y, cor do texto no padrão RGB, tamanho do texto e o texto a ser escrito.
  `TEXT PosX PosY Red Green Blue Size Text`
- DRAW (polígono desenhável): cor de preenchimento do polígono no padrão RGB, cor do contorno no padrão RGB, estilo de linha (sólida ou tracejada), posição inicial X e Y, e as direções para onde deve ser desenhada a linha de contorno. As direções foram definidas usando as primitivas U (para cima), D (para baixo), L (para a esquerda), R (para a direita), seguidos do tamanho da linha. Também foi definida uma primitiva A para direcionar a linha de contorno para um determinado ângulo. Este deve ser seguido do valor do ângulo para onde a linha será direcionada em graus e o tamanho da linha.
  `DRAW Red Green Blue Red Green Blue LineStyle PosX PosY (...)`
- WINDOW (janela onde as figuras são colocadas): posição inicial X e Y, largura, altura, origem da janela (superior ou inferior esquerdo), cor de preenchimento do retângulo no padrão RGB, valor mínimo e máximo da escala em X e o valor mínimo e máximo da escala em Y.

```
WINDOW PosX PosY Width Height Origin Red Green Blue
MinScaleX MaxScaleX MinScaleY MaxScaleY
```

Como padrão da arquitetura 2DMG, as figuras devem ser adicionadas a uma janela (WINDOW) para serem desenhadas; ou seja, a janela deve ser o primeiro elemento a ser criado, para posteriormente inserir as demais figuras nela.

Todas essas instruções, que descrevem uma imagem a ser desenhada pelo interpretador, são escritas em arquivos no formato ".rtf" (Rich Text Format), seguindo as especificações dos comandos citados anteriormente. Ainda neste capítulo será apresentada uma tela do programa em execução com a figura de um perfil e o *script* usado para desenhá-la.

Um **outro interpretador** foi criado para ser usado no programa. Ele é responsável por ler os dados dos atributos de um perfil em um arquivo no formato ".plist" (*Property List*), usado nos sistemas Mac OS X para armazenar objetos de forma serializada. Esse arquivo deve seguir algumas especificações, a saber:

1. A primeira linha deve conter o número de designações do perfil (ND) que possui no arquivo.
2. A segunda linha deve conter o número de atributos (NA) que o perfil possui.
3. As próximas NA linhas devem conter as legendas, com a unidade indicada entre parênteses, para cada atributo do perfil.
4. As próximas ND*(NA+1) linhas devem conter o nome de cada perfil com a designação, seguido dos respectivos valores dos atributos na ordem estabelecida pela legenda dos atributos.

Um exemplo do arquivo de dados pode ser visto no Apêndice A desta dissertação (Quadro 7.2). Ele foi usado na Figura 5.4 da próxima sessão.

Além desses aspectos relevantes sobre a implementação, é importante destacar o uso do *middleware* no desenvolvimento. Resumidamente, o programador fez um interpretador para ler o arquivo com as instruções da figura a ser desenhada, que por sua vez, ao encontrar um comando, usa as classes do 2DMG, implementadas para o iOS, para criar os elementos gráficos que formam a figura e a janela que a contém. Agilizando, e muito, a codificação da versão móvel do Desmet.

5.2.4 Resultados

A versão do *middleware* para o iOS manteve as especificações inicialmente levantadas para a arquitetura de classes 2DMG, exceto pela sintaxe que é

característica da linguagem Objective-C. Sendo assim, o alvo de servir como uma interface única para o programador foi alcançado, pois ele não precisou fazer alterações nas classes do modelo implementadas para o iOS ao desenvolver o estudo de caso.

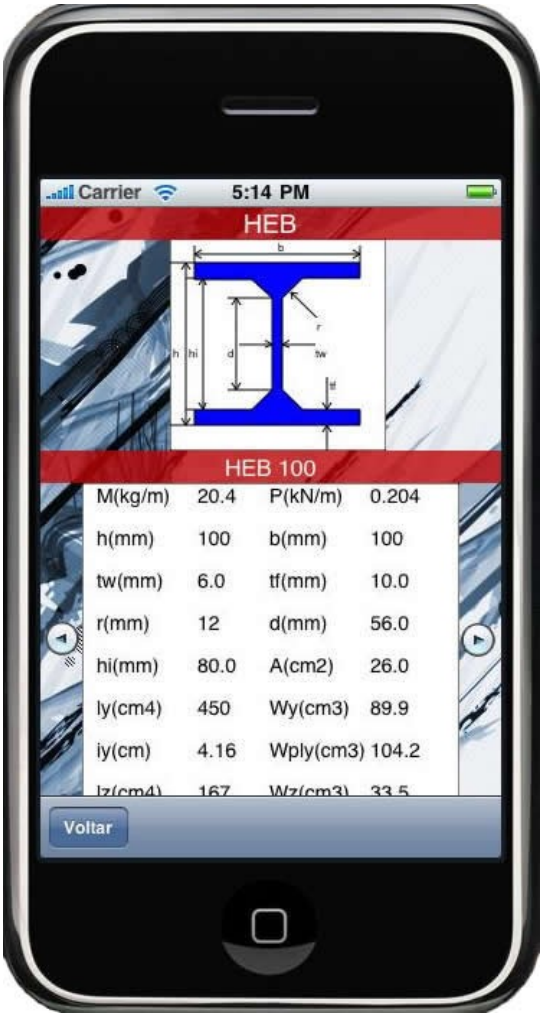
A Figura 5.3 exemplifica a tela inicial do Desmet Mobile sendo executado no simulador da iOS SDK. Ela contém a lista de perfis disponíveis nesta primeira versão (HEB, HEA, HEM, IPE, IPN, UPN e Ang).

Figura 5.3. Simulador iOS SDK com a tela inicial do Desmet para o iPhone.

Fonte: Elaborada pelo autor.

Ao escolher um perfil, o usuário é direcionado a uma outra tela com o desenho do perfil (criado com elementos gráficos do 2DMG) e os atributos de dimensão do mesmo (Figura 5.4). Os botões laterais alteram a designação do perfil; ou seja, aumentam ou diminuem as dimensões do mesmo. Ao mesmo tempo, o software atualiza o desenho do perfil dinamicamente de acordo com as alterações.

Figura 5.4. Simulador iOS SDK mostrando a tela do perfil HEB com os dados de seus atributos.



Fonte: Elaborada pelo autor.

A imagem em duas dimensões do perfil, que se encontra na parte superior da Figura 5.4, é toda desenhada utilizando-se as classes do 2DMG. É importante destacar que na versão desktop da aplicação essas figuras são imagens estáticas, portanto não se alteram dinamicamente em função de seus atributos.

A utilização do interpretador para o *script* com as rotinas de criação da imagem simplificou a tarefa de atualização da figura do perfil em tempo real. Ao modificar o arquivo com as rotinas, o interpretador é chamado novamente para redesenhar a imagem. O arquivo descritor de imagem utilizado na Figura 5.4 pode ser observado no Apêndice A (Quadro 7.1).

6 CONCLUSÕES

De fato, a grande quantidade de SOs e linguagens de programação existentes hoje para o desenvolvimento de aplicações em dispositivos móveis dificulta consideravelmente a portabilidade de um *software*. Apesar de haver bibliotecas como a OpenGL ES, a API M3G e outras alternativas que auxiliam a portabilidade entre SOs, elas são mais voltadas para jogos e animações, tornando-as mais complexas. Sendo assim, o uso das mesmas torna o trabalho do programador muito dispendioso, principalmente quando a aplicação em desenvolvimento necessita somente de recursos gráficos 2D.

Antes de elaborar uma solução para o problema acima mencionado, que é o objetivo principal desse trabalho, foi preciso investigar outras ferramentas usadas como camadas de adaptação. De tal forma fica o registro desse panorama, exposto no Capítulo 2, em relação ao desenvolvimento de aplicações móveis, principalmente quanto a recursos gráficos. Lembrando que a área está em constante evolução tecnológica e a cada ano novos recursos são incorporados aos dispositivos móveis, proporcionando novas aplicabilidades.

Diante do quadro supracitado, o 2DMG é apresentado como uma solução à portabilidade nesses casos. Com uma arquitetura de classes e interface padrão para o desenvolvedor, o *middleware*, em sua primeira versão, foi implementado nas linguagens nativas C++ (Symbian), C++ (Windows Mobile), Objective-C (iOS) e Java ME, atendendo assim a diversos dispositivos móveis.

No estudo de caso escolhido, foi possível adaptar partes de uma ferramenta computacional (usada no ensino de graduação do curso de Engenharia Civil da UFV) para ser executada nos aparelhos da Apple (iPhone, iPad e iPod Touch). Tal experimento foi imprescindível para o teste e validação da solução apresentada por este trabalho, além de contribuir para a modernização de uma ferramenta de ensino desta instituição.

As classes padronizadas da arquitetura 2DMG permitem ao desenvolvedor programar aplicativos que utilizam objetos gráficos bidimensionais de forma portável e produtiva, como observado no estudo de caso desta dissertação. Ao usar as classes do *middleware* no desenvolvimento de um aplicativo, o mesmo pode ser implementado em outras plataformas sem grandes esforços para traduzir o código.

Outra contribuição deste projeto foi o envolvimento de dois estudantes de graduação com tecnologias recentes e plataformas de desenvolvimento móvel. Ambos estavam em fase de término do curso de Ciência da Computação na UFV, sendo que um deles tinha bolsa de iniciação científica (PIBIC/CNPq).

É importante destacar que atualmente a procura por profissionais nessa área é muito grande, como aponta a notícia "Os 4 profissionais de TI mais demandados por grandes empresas no País", publicada no sítio do programa Olhar Digital[7] em 01 de junho de 2011, que diz: "Os cargos mais procurados são arquiteto SOA, **arquiteto de mobilidade**, gerente de segurança da informação e gerente de projetos".

Finalmente, ressalta-se que a principal contribuição deste trabalho é a especificação e implementação do *middleware* 2DMG e sua disponibilização como projeto *open source.* O mesmo já foi registrado no repositório Google Code[8] e estará disponível após a publicação da dissertação, facilitando assim a manutenção do projeto e a contribuição da comunidade em trabalhos futuros.

6.1 Trabalhos futuros

As classes construídas nesta primeira versão do *middleware* não abrangem todos os tipos de elementos gráficos 2D, mas se ateve aos essenciais. Um exemplo disso é a especificação das classes *Arrow1 e Arrow2* durante o processo de desenvolvimento do caso de uso para o iOS. Entretanto, a definição do modelo padrão auxilia a criação de outras classes especializadas da classe abstrata *Fig*. Dessa forma, entende-se que o 2DMG pode ser ampliado em trabalhos futuros.

Apesar do Java ME ser uma plataforma disponível em boa parte dos sistemas operacionais móveis e implementada neste trabalho, é possível atender mais SOs disponíveis no mercado, como Blackberry e Android, fazendo a implementação usando recursos nativos da linguagem dos sistemas. Este último era objetivo deste

[7] http://olhardigital.uol.com.br/, acessada em junho de 2011.

[8] http://code.google.com/p/2dmg/, acessada em julho de 2011.

projeto, mas infelizmente devido a limitação de tempo não foi possível fazer a implementação para o sistema Android.

Outra sugestão para trabalhos futuros é a elaboração de dois manuais. Um, que descreveria os passos detalhados para a implementação das classes concretas em outras plataformas. E outro, para servir de manual do usuário, desenvolvedor de aplicações móveis que utiliza recursos gráficos 2D, detalhando formas de uso que não foram abordadas nos capítulos anteriores.

O escopo do trabalho está definido na implementação de uma arquitetura de classes e interface comum para o desenvolvedor no que tange a recursos gráficos 2D. Em relação a gráficos 3D, já existem alternativas amplamente usadas. Mas com o avanço em termos de processamento e memória dos dispositivos móveis é possível criar *middlewares* para diversos recursos de um software móvel, como métodos numéricos e estatísticos, que são frequentemente usados em ferramentas científicas.

APÊNDICE A

O arquivo para descrever a figura de um perfil é escrito no formato ".rtf" (*Rich Text Format*), exemplificado a seguir (Quadro 7.1).

Quadro 7.1. *Script* que descreve a imagem do perfil apresentado na Figura 5.4.

```
1:    WINDOW 90 2 150 150 1 255 255 255 0 #<b>+30# 0 #<h>+30#
2:    DRAW 0 0 255 0 0 0 0 15 15 R #<b># D #<tf># L #<b>/2-<tw>/2-(<hi>-
      <d>)/2# A #<r>+5# 225 D #<d># A #<r>+5# 315 R #<b>/2-<tw>/2-(<hi>-
      <d>)/2# D #<tf># L #<b># U #<tf># R #<b>/2-<tw>/2-(<hi>-<d>)/2# A
      #(<hi>-<d>)*0.7# 45 U #<d># A #<r>+5# 135 L #<b>/2-<tw>/2-(<hi>-
      <d>)/2#
3:    LINE 15 15 15 5 0 0 0 0
4:    LINE #15+<b># 15 #15+<b># 5 0 0 0 0
5:    ARROW2 15 10 #<b>+15# 10 0 0 0 0
6:    TEXT #(<b>+30)/2# 0 0 0 0 8 "b"
7:    LINE 15 15 0 15 0 0 0 0
8:    LINE 15 #<h>+15# 0 #<h>+15># 0 0 0 0
9:    ARROW2 10 15 10 #<h>+15# 0 0 0 0
10:   TEXT 2 #(<h>+30)/2# 0 0 0 8 "h"
11:   LINE 15 #<tf>+15# 10 #<tf>+15# 0 0 0 0
12:   LINE 15 #<hi>+<tf>+15# 10 #<hi>+<tf>+15# 0 0 0 0
13:   ARROW2 20 #<tf>+15# 20 #<hi>+<tf>+15# 0 0 0 0
14:   TEXT 12 #(<h>+30)/2# 0 0 0 8 "hi"
15:   LINE #15+<b>/2-<tw>/2# #(<hi>-<d>)/2+<tf>+15# #<b>/2-15# #(<hi>-
      <d>)/2+<tf>+15# 0 0 0 0
16:   LINE #15+<b>/2-<tw>/2# #<hi>+<tf>+15-(<hi>-<d>)/2# #<b>/2-15#
      #<hi>+<tf>+15-(<hi>-<d>)/2# 0 0 0 0
17:   ARROW2 #<b>/2-10# #15+(<h>-<d>)/2# #<b>/2-10# #15+<h>-(<h>-<d>)/2# 0
      0 0 0
18:   TEXT #<b>/2-15# #(<h>+30)/2# 0 0 0 8 "d"
19:   ARROW1 #<b>/2-8# #15+<h>/2# #15+<b>/2-<tw>/2# #15+<h>/2# 0 0 0 0
20:   ARROW1 #<b>/2+38# #15+<h>/2# #15+<b>/2+<tw>/2# #15+<h>/2# 0 0 0 0
21:   TEXT #<b>/2+38# #15+<h>/2# 0 0 0 8 "tw"
22:   ARROW1 #15+4*<h>/5# #<h>-15# #15+4*<h>/5# #15+<h>-<tf># 0 0 0 0
23:   ARROW1 #15+4*<h>/5# #<h>+30# #15+4*<h>/5# #15+<h># 0 0 0 0
24:   TEXT #17+4*<h>/5# #<h>-15# 0 0 0 8 "tf"
25:   ARROW1 #15+<b>/2+(<hi>-<d>)# #15+<tf>+(<hi>-<d>)# #17+<b>/2+(<hi>-
      <d>)/4# #17+<tf>+(<hi>-<d>)/4# 0 0 0 0
26:   TEXT #15+<b>/2+(<hi>-<d>)# #15+<tf>+(<hi>-<d>)# 0 0 0 8 "r"
```

Fonte: Elaborado pelo autor.

Para o arquivo de dados de um perfil foi adotado o formato ".plist" (*Property List*), usado nos sistemas Mac OS X para armazenar objetos de forma serializada, como no exemplo apresentado a seguir (Quadro 7.2).

Quadro 7.2. Arquivo de dados usado na Figura 5.4.

```
1.    <?xml version="1.0" encoding="UTF-8"?>
2.    <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
    "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
3.    <plist version="1.0">
4.    <array>
5.          <integer>2</integer>
6.          <integer>25</integer>
7.          <string>M(kg/m)</string>
8.          <string>P(kN/m)</string>
9.          <string>h(mm)</string>
10.         <string>b(mm)</string>
11.         <string>tw(mm)</string>
12.         <string>tf(mm)</string>
13.         <string>r(mm)</string>
14.         <string>d(mm)</string>
15.         <string>hi(mm)</string>
16.         <string>A(cm2)</string>
17.         <string>ly(cm4)</string>
18.         <string>Wy(cm3)</string>
19.         <string>iy(cm)</string>
20.         <string>Wply(cm3)</string>
21.         <string>lz(cm4)</string>
22.         <string>Wz(cm3)</string>
23.         <string>iz(cm3)</string>
24.         <string>Wplz(cm3)</string>
25.         <string>lt(cm4)</string>
26.         <string>lw(cm6)</string>
27.         <string>AL(m2/m)</string>
28.         <string>AG(m2/t)</string>
29.         <string>Avz(cm2)</string>
30.         <string>Sy(cm3)</string>
31.         <string>sy(cm)</string>
32.         <string>HEB 100</string>
33.         <string>20.4</string>
34.         <string>0.204</string>
35.         <string>100</string>
36.         <string>100</string>
37.         <string>6.0</string>
38.         <string>10.0</string>
39.         <string>12</string>
40.         <string>56.0</string>
41.         <string>80.0</string>
42.         <string>26.0</string>
43.         <string>450</string>
44.         <string>89.9</string>
45.         <string>4.16</string>
46.         <string>104.2</string>
47.         <string>167</string>
48.         <string>33.5</string>
49.         <string>2.53</string>
50.         <string>51.4</string>
51.         <string>9.3</string>
52.         <string>3387</string>
53.         <string>0.567</string>
54.         <string>27.76</string>
55.         <string>9.04</string>
56.         <string>52.1</string>
57.         <string>8.6</string>
58.         <string>HEB 120</string>
59.         <string>26.7</string>
60.         <string>0.267</string>
61.         <string>120</string>
62.         <string>120</string>
63.         <string>6.5</string>
64.         <string>11.0</string>
```

```
65.          <string>12</string>
66.          <string>74.0</string>
67.          <string>98.0</string>
68.          <string>34.0</string>
69.          <string>864</string>
70.          <string>144.1</string>
71.          <string>5.04</string>
72.          <string>165.2</string>
73.          <string>318</string>
74.          <string>52.9</string>
75.          <string>3.06</string>
76.          <string>81.0</string>
77.          <string>13.9</string>
78.          <string>9431</string>
79.          <string>0.686</string>
80.          <string>25.71</string>
81.          <string>10.96</string>
82.          <string>82.6</string>
83.          <string>10.5</string>
84.    </array>
85.    </plist>
```

Fonte: Elaborado pelo autor.

REFERÊNCIAS BIBLIOGRÁFICAS

ABNT, NBR 8800 - Projeto e Execução de Edifícios em Estrutura de Aço. **Associação Brasileira de Normas Técnicas**. Rio de Janeiro, 1986.

CALLOW, M., BEARDOW, P., BRITTAIN, D. Big Games, Small Screens, **ACM Queue**, v. 5 No. 7, p. 40-50, Nov./Dec. 2007.

CAPIN, T., PULLI, K., MÖLLER, T. The State of the Art in Mobile Graphics Research, **IEEE Computer Graphics and Applications**, p. 74-84, July/August 2008.

CHITTARO, L. Visualizing Information on Mobile Devices, **IEEE Computer**, v. 39, n. 3, p. 40-45, 2006.

GAMMA, E., HELM, R., JOHNSON, R., VLISSIDES, J. **Design Patterns:** Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

GARLAN, D. "Software Architecture: a Roadmap". **International Conference on Software Engineering: Future of SE Track**, p. 91-101, Limerick, Ireland, 2000.

JONES, M., MARSDEN, G. Mobile Interaction Design, **John Wiley & Sons**, 2006.

KHRONOS. OpenVG 1.0 Specification, Khronos OpenVG API Registry, 2005. Disponível em http://www.khronos.org/registry/vg/, acessada em maio de 2009.

KRUCHTEN, P. "The 4+1 View Model of Architecture", **IEEE Software**. 12(6), Nov. 1995.

LOUREIRO, A. A. F., SADOK, D. F. H., MATEUS, G. R., NOGUEIRA, J. M. S., KELNER, J. Comunicação Sem Fio e Computação Móvel: Tecnologias, Desafios e Oportunidades. **XXII Jornada de Atualização em Informática**. Campinas, SP: Sociedade Brasileira de Computação, v. 2, p. 195-244, 2003.

MARTINS, W. S., LISBOA FILHO, J., Cupuaçu - Sistema de Cadastro de BCI-Boletim de Cadastro Imobiliário para Dispositivos Móveis, 2005.

MENDES, T. S., ROCHA, M. N., SILVA, N. R. M. O., A User Profile and Location-Based Services Matching Model Architecture, **IADIS International Conference - Information Systems**, 2008.

MONTEIRO, B. R., Aplicações de Sistemas de Informação Geográfica Móveis: um estudo voltado para iniciativas de governo eletrônico na administração pública municipal. **Dissertação (mestrado) - Universidade Federal de Viçosa**, CAPES. Orientador: Jugurta Lisboa Filho, 2007.

MOSMONDOR, M., KOMERICKI, H., PANDZIC, I. S. 3D Visualization on Mobile Devices, **Telecommunication Systems**, v. 32, p. 181–191, 2006.

NOGUEIRA, H. M., LADEIRA, P. H., ROCHA, M. N., The Challenge of Displaying Web Mobile Pages, **IADIS International Conference - Information Systems,** 2008.

OLIVEIRA, L. A., Conjunto de Classes para Aplicações Gráficas 2D em Sistemas Embarcados Baseados no Femtojava. **Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul**. Programa de Pós-Graduação em Computação. Orientador: Luigi Carro. Porto Alegre, BR - RS, 2006.

PRESSMAN, R. S. **Software Engineering: a practitioner's approach**. 5Th Edition, McGraw-Hill, 2000.

PULLI, K., AARNIO, T., ROIMELA, K., VAARALA, J. Designing Graphics Programming Interfaces for Mobile Devices, **IEEE Computer Society**, p. 66-75 Nov./Dec. 2005.

PULLI, K. **"New APIs for Mobile Graphics"** Proc. SPIE Electronic Imaging: Multimedia on Mobile Devices II, SPIE, 2006.

PULLI, K., VAARALA, J., MIETTINEN, V. Refining OpenGL for Embedded Systems, **ACM Queue**, v. 5 n. 7, p. 42-43, Nov./Dec. 2007.

PULLI, K., AARNIO, T., VILLE, M., ROIMELA, K., VAARALA, J. **Mobile 3D Graphics with OpenGL ES and M3G**. Morgan Kaufmann, 2008.

SAFFER, D., Designing for Interaction – Creating Smart Applications and Clever Devices, **AIGA / New Riders**, 2007.

SEI. Software Engineering Institute. **CMMI Web Site**. Disponível em http://www.sei.cmu.edu/cmmi/, acessada em 20 de fevereiro de 2009.

SILVA, N. R. M. O., ROCHA, M. N., MENDES, T. S., LADEIRA, P. H., NOGUEIRA, H. M., The Use of Mobile Devices in Educational Environments - A Case Study, **IADIS International Conference - Mobile Learning,** 2008.

VERÍSSIMO, G. S., PAES, J. L. R. Software e Ensino de Estruturas Metálicas: um Problema ou uma Realidade? **III Seminário Internacional "O Uso de Estruturas Metálicas na Construção Civil"**, Belo Horizonte, MG, Sociedade Mineira de Engenheiros, 2000.

VERÍSSIMO, G. S., RIBEIRO, J. C. L., PAES, J. L. R., Software para Cálculo de Vigas Mistas. **Revista Construção Metálica**, São Paulo, v. 56, p. 32 - 39, 11 nov. 2002.