Thiago Henrique de Paula Figueiredo

MultiMAD:
Uma ferramenta multimodelo de desenvolvimento de aplicações para dispositivos móveis

Dissertação apresentada ao Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Belo Horizonte

29 de abril de 2005

Agradecimentos

Antes de tudo, agradeço a Deus por ter me dado todas as oportunidades, força e inspiração necessárias para poder ter chegado aqui e, prinicpalmente, por ter colocado em vinha vida todas as pessoas e coisas às quais vou agradecer agora.

Agradeço a meus pais, Vagner e Neli, ao meu irmão Thales e à minha família inteira.

Agradeço ao meu orientador e amigo Loureiro.

Agradeço à minha namorada, Cristilene, esta benção divina que Deus me deu.

Agradeço muito ao Departamento de Ciência da Computação por ter me dado uma formação tão boa e completa na graduação e por ter me dado a chance de fazer um mestrado. Além disso, através do Synergia, o DCC me deu a primeira oportunidade de trabalhar como um bacharel em ciência da computação. Agradeço também à professora Wong, ao professor Bigonha e à Túlia.

Agradeço muito também ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) por ter me proporcionado a bolsa que recebi durante quase todo o meu mestrado.

Agradeço a todas as pessoas da ETEG Internet Ltda, especialmente aos seus sócios Rafael, Rodrigo e Walter.

Agradeço também a toda a minha turma da graduação, principalmente meus grandes amigos Waldir, Charles, Fagner, Flávio, Cláudio, André Amado, Tassni e Ana Paula, com quem tanto convivi e aprendi. Do mestrado, agradeço também à Fátima, ao Max, Vinícius, Daniel, Renan, Rabelo, Lula, Rainer, Martin, Alla, ...

Agradeço a todos os meus amigos e amigas, especialmente ao Matheus e toda sua família, Edilayne, Aninha Marcelina e André. E um grande abraço para meus amigos dos Bootneiros, do time de basquete do ICEX e do Winchester United.

Resumo

Esta dissertação apresenta o MultiMAD (*Multimodel Mobile Application Development Tool* Ferramenta Multimodelo de Desenvolvimento de Aplicações para Dispositivos Móveis), uma ferramenta de desenvolvimento rápido especializada em dispositivos móveis. Ela provê uma interface gráfica na qual uma aplicação é descrita a partir de blocos de construção que podem ou não ser específicos para um determinada plataforma de aplicações móveis. Isto permite que aplicações possam ser desenvolvidas para diversas plataformas utilizando todos os seus recursos, característica não encontrada na quase totalidade das outras ferramentas similares. Quando esta descrição está pronta, o usuário a submete para um gerador de aplicações que irá gerar parcialmente, de forma automática, código-fonte e outros arquivos que eventualmente façam parte da implementação da aplicação. Atualmente, o MultiMAD gera código-fonte que implementa a interface gráfica de usuário e o relacionamento entre seus elementos, possibilitando que o desenvolvedor possa focar seus esforços na lógica da aplicação. O MultiMAD também é uma ferramenta de prototipagem rápida, já que ele também gera um protótipo de implementação da lógica e, por conseqüência, gera um protótipo de aplicação compilável e testável sem que o usuário tenha que escrever uma única linha de código ou tenha que sair da ferramenta. Este protótipo de implementação é o ponto de partida da implementação definitiva da lógica da aplicação. Foram implementados geradores para as plataformas Wireless Application Protocol (WAP) 1.1 e 2.0 e Java 2 Micro Edition (J2ME). Também é proposto o MobileVC, uma arquitetura especialmente criada para aplicações para dispositivos móveis.

Abstract

We present MultiMAD (*Multimodel Mobile Application Development Tool*), a rapid application development (RAD) tool for mobile devices. It provides a graphical interface at which an application can be described using construction blocks that can be specific to some mobile application platform or not. This allows MultiMAD to be used to develop applications to many platforms using all their features. This MultiMAD feature is not found in almost all similar tools. When this description is finished, the user submits it to an application generator that generates automatically part of the source code and other files that implement the application. Currently, MultiMAD generates source code that implements the aplication's graphical user interface and its elements relationship, thus allowing the developer to focus on the application logic. MultiMAD is also a rapid application prototyping tool, as it also generates a prototype implementation of the application logic and thus generates a compilable and testable application prototype without the user writing a single line of code or leaving the tool. This prototype implementation is the starting point of the definitive implementation of the application logic. Generators were implemented for the Wireless Application Protocol (WAP) 1.1 and 2.0 and Java 2 Micro Edition (J2ME) platforms. We also present MobileVC, one architecture designed specifically for applications for mobile devices.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Os últimos anos testemunharam uma grande evolução e popularização da computação móvel, incluindo-se a telefonia celular. Os dispositivos móveis estão cada vez mais inseridos no dia-a-dia das pessoas, fazendo com que elas possam acessar, armazenar e processar dados, se comunicar e se informar em qualquer lugar. Estes dispositivos podem ser divididos em duas grandes classes não disjuntas[1]: celulares e PDAs (*Personal Digital Assistants*)[2]. Ambos já provêem acesso à Internet e a possibilidade de se executar aplicações tanto *on-line* (que utilizam alguma rede) quanto *off-line* (sem nenhum tipo de conexão de rede).

O número de usuários de dispositivos móveis vem aumentando a cada dia. No mundo inteiro, apenas em 2004, foram vendidos 9,2 milhões de PDAs. As vendas anuais de PDAs foram superiores a 10 milhões de unidades desde 1999 [22]. Estes números não incluem dispositivos que também são telefones celulares. De todos os PDAs vendidos em 2004, 44% podem se conectar a redes de telefonia celular e/ou WLANs (*Wireless Local Area Network*, rede local sem fio) [65]. A plataforma Java 2 Micro Edition (J2ME) já é suportada por mais de 100 operadoras e 400 diferentes modelos de dispositivos móveis [63]. O número de usuários de aparelhos com esta tecnologia é de aproximadamente 300 milhões no início de 2005 [62].

De acordo com [46], as tecnologias mais utilizadas para acessar a Internet

[1] Neste trabalho são considerados apenas os dispositivos móveis de uso pessoal.

[2] Os aparelhos de telefonia celular que provêm funcionalidades de PDA são chamados de *smartphones*.

através de dispositivos móveis no ano de 2002 foram o WAP (*Wireless Application Protocol*) [37, 38], padrão aberto gerenciado pelo WAP Forum; Java 2 Micro Edition (J2ME) [57], tecnologia criada e mantida pela Sun Microsystems; e BREW [43], criada e mantida pela Qualcomm.

Tamanha popularidade dos dispositivos móveis produz uma demanda de desenvolvimento de aplicações específicas para eles. Dispositivos móveis, porém, têm características diferentes daquelas encontradas em computadores pessoais, por exemplo [17]. A CPU tem um menor poder de processamento. A memória tem menor capacidade. A fonte de energia, em geral baterias ou pilhas, faz com que o uso destes dispositivos seja restrito. A tela é muito menor. Os mecanismos de entrada de dados são diferentes. A conexão de rede tem uma largura de banda reduzida, latência e taxa de erros maiores quando comparados com redes cabeadas. Além disso, as desconexões podem ser freqüentes. Este conjunto de características faz com que o desenvolvimento de aplicações para dispositivos móveis não seja feito da mesma forma e nem com as mesmas ferramentas utilizadas em aplicações para computadores pessoais ou *mainframes*. A conseqüência deste fato é a necessidade da construção de novas ferramentas de desenvolvimento de aplicações específicas para este tipo de dispositivo.

1.1 Objetivos

Os objetivos deste trabalho são especificar e implementar uma ferramenta, MultiMAD (*Multimodel Mobile Application Development Tool*, Ferramenta de Desenvolvimento de Aplicações para Dispositivos Móveis), que, com o mínimo possível de esforço do desenvolvedor de aplicações para dispositivos móveis, gere a maior parte possível da aplicação, diminuindo-se assim a complexidade, os custos e o tempo de desenvolvimento. A meta é fazer com que o desenvolvedor possa focar na especificação da lógica da aplicação, deixando para a ferramenta a tarefa de, por exemplo, gerar a interface de usuário e implementar o módulo de comunicação com um eventual servidor da aplicação. Ela deve permitir a implementação tanto de aplicações que utilizem comunicação via rede quanto de aplicações que utilizem apenas dados locais.

A ferramenta deve fornecer um modelo no qual aplicações possam ser especificadas independentemente do dispositivo móvel no qual ela eventualmente venha a ser executada. A implementação de aspectos específicos a algum dispositivo ou plataforma serão implementadas em geradores e modelos de aplicação específicos ou pelo próprio desenvolvedor, depois que o código-fonte inicial da aplicação tenha sido gerado pela ferramenta.

Outro objetivo deste trabalho é a proposição de uma arquitetura específica para aplicações para dispositivos móveis.

1.2 Contribuições

As principais contribuições deste trabalho são:

- A elaboração e a construção de uma ferramenta, MultiMAD, que tem como objetivo facilitar o desenvolvimento de aplicações para dispositivos móveis. Elas crescem a cada dia em número de usuários e em importância tecnológica e econômica.
- A apresentação do conceito de múltiplos modelos de aplicações. Todas as outras ferramentas de nosso conhecimento não possuem esta característica. O conceito de modelos de aplicações será discutido na seção 5.1.3.
- A proposição de uma arquitetura de aplicações para dispositivos móveis: MobileVC.

1.3 Estrutura deste documento

Este documento está dividido em 10 capítulos.

O capítulo 1 introduz o problema tratado e discute os objetivos e as contribuições neste trabalho. O capítulo 2 descreve as atuais plataformas de aplicações móveis. O capítulo 3 descreve os trabalhos relacionados ao MultiMAD. O capítulo 4 apresenta a especificação de requisitos proposta para a ferramenta desenvolvida neste trabalho. O capítulo 5 descreve a ferramenta partindo de seus

conceitos fundamentais, passando pela sua arquitetura interna e interface gráfica e chegando a sua forma de uso. O capítulo 6 apresenta os modelos de aplicações implementados neste trabalho: modelo genérico e modelo específico para a plataforma Java 2 Micro Edition. O capítulo 7 discorre sobre a arquitetura MobileVC de aplicações para dispositivos móveis. O capítulo 8 descreve os geradores de aplicações desenvolvidos neste trabalho: gerador para a plataforma WAP 1.1 e 2.0 e gerador para a plataforma Java 2 Micro Edition. O capítulo 9 mostra um exemplo de desenvolvimento de aplicação no MultiMAD: uma aplicação de controle de gastos pessoais. Finalmente, o capítulo 10 apresenta as conclusões e os trabalhos futuros. O apêndice A apresenta um glossário e uma lista de acrônimos.

Capítulo 2

Plataformas de desenvolvimento para dispositivos móveis

Neste capítulo é apresentada uma lista não exaustiva de plataformas de desenvolvimento de aplicações para dispostivos móveis. Neste, considera-se que uma plataforma é um conjunto de especificações, protocolos, componentes de *software* e/ou sistema operacional sobre o qual aplicações são construídas. Cada uma das plataformas é descrita brevemente. Elas serão divididas em duas categorias: linguagens de marcação e plataformas de programação.

2.1 Linguagens de marcação

Plataformas baseadas em linguagens de marcação fazem pouco ou nenhum processamento no dispositivo, utilizando um navegador (*browser*) para exibir para o usuário documentos requisitados a um servidor. É nele em que todo o processamento e armazenamento de dados das aplicações acontece.

Nesta seção serão descritas as principais plataformas baseadas em linguagens de marcação utilizadas em aplicações para dispositivos móveis: HTML, XHTML Basic, WAP, cHTML e VoiceXML.

2.1.1 HTML

HTML (*HyperText Markup Language*, Linguagem de Marcação de Hipertexto) [68] é um formato de padrão aberto baseado em SGML [23], sendo a língua franca da WWW (*World Wide Web*).

Por outro lado, HTML é uma linguagem pouco utilizada no mundo dos dispositivos móveis. Além de não ter sido projetada para este tipo de cenário, páginas HTML são geralmente projetadas para serem visualizadas em telas de resolução maior que as encontradas tipicamente em dispositivos móveis. Este fato faz com que o usuário visualize apenas uma pequena fração da página, tornando-a de difícil leitura e forçando o usuário a rolar a tela muitas vezes ao tentar visualizar a página inteira. A tecnologia Small Screen Rendering^{TM} [40], da Opera Software ASA, se apresenta como uma solução para este problema, reformatando páginas para que a página tenha a largura da tela do dispositivo, eliminando o rolamento horizontal.

Outra solução é a construção de páginas HTML específicas para o ambiente móvel. Um exemplo disso é a arquitetura WCA (*Web Clipping Application*, Aplicação de Aparamento de Páginas Web) em dispositivos que usam a plataforma PalmOS [41]. Esta arquitetura é formada de aplicações no lado do cliente, servidores *proxy* que traduzem conteúdo entre HTML e o formato usado na WCA e servidores de conteúdo. A aplicação é construída em HTML, convertida para o formato WCA e instalada no dispositivo. O conteúdo deve ser gerado pelo provedor de conteúdo em um subconjunto do padrão HTML 3.2 [44]. As aplicações são acessadas no dispositivo através de um visualizador específico. Esta solução tem a desvantagem da necessidade de conversão de conteúdo, já que o formato das páginas não é o utilizado na Web.

2.1.2 XHTML Basic

XHTML Basic [69] é um subconjunto de XHTML 1.0 [70] concebido como um formato de documento que pudesse ser compartilhado entre diferentes tipos de dispositivos que acessam a Internet tais como computadores pessoais, PDAs, telefones celulares e aparelhos de televisão. Ele foi definido pelo Consórcio W3C [67] como o padrão de formato de conteúdo para dispositivos móveis.

2.1.3 WAP

O WAP (*Wireless Application Protocol*, Protocolo de Aplicações sem Fio) é um protocolo desenvolvido pela Open Mobile Alliance (antigo WAP Forum) [35] para prover acesso à Internet através de dispositivos móveis.

Em sua versão 1.0, o navegador WAP [37] exibe documentos escritos em WML 1.1 (*Wireless Markup Language*, Linguagem de Marcação sem Fio), formato baseado em XML. A comunicação entre o navegador WAP e o servidor Web onde está a página requisitada é feita através de um *gateway* que realiza a conversão de WML para o formato binário definido pela arquitetura WAP.

Já a versão 2.0 de WAP [38], além de retirar a necessidade de um *gateway*, usa o XHTML Mobile Profile como o formato de documento a ser utilizado [39]. Ele é baseado em XHTML Basic.

2.1.4 cHTML

A linguagem cHTML (*compact HyperText Markup Language*, Linguagem Compacta de Marcação de Hipertexto) é um subconjunto de HTML escolhido para ser usado em dispositivos móveis tais como telefones celulares e PDAs [25]. Esta linguagem foi criada pela empresa japonesa Access Co. para ser usada no i-mode, serviço de Internet para telefones celulares da operadora japonesa NTT DoCoMo. Em dezembro de 2004 este serviço possuía mais de 43 milhões de usuários, praticamente todos no Japão [34].

2.1.5 VoiceXML

O VoiceXML (*Voice Extensible Markup Language*, linguagem de marcação extensível de voz) é uma linguagem de marcação baseada em XML [72]. Ele permite a criação de formas de interação entre navegador e usuário baseadas em áudio, incluindo voz sintetizada, áudio digitalizado, reconhecimento de palavras faladas e gravação de entradas faladas. O maior objetivo do VoiceXML é trazer as vantagens do desenvolvimento baseado na Web e de disponibilização de conteúdo para aplicações interativas de voz.

2.2 Plataformas de programação

Nas plataformas de programação, ao contrário de plataformas baseadas em linguagens de marcação, o processamento de dados é realizado no próprio dispositivo. Aplicações podem ou não se comunicar com outros dispositivos ou servidores através de uma rede.

Nesta seção serão descritas as principais plataformas de programação utilizadas em aplicações para dispositivos móveis: Java 2 Micro Edition, PalmOS, Symbian OS, Windows CE, Waba e BREW.

2.2.1 Java 2 Micro Edition

A plataforma Java 2 Micro Edition [57], da Sun Microsystems, é uma adaptação da plataforma Java para dispositivos tais como telefones celulares, PDAs e dispositivos embutidos. A figura 2.1 mostra sua arquitetura. Ela é divida em configurações e perfis, permitindo que a plataforma possa ser implementada de acordo com os requisitos e limitações de uma grande variedade de dispositivos.

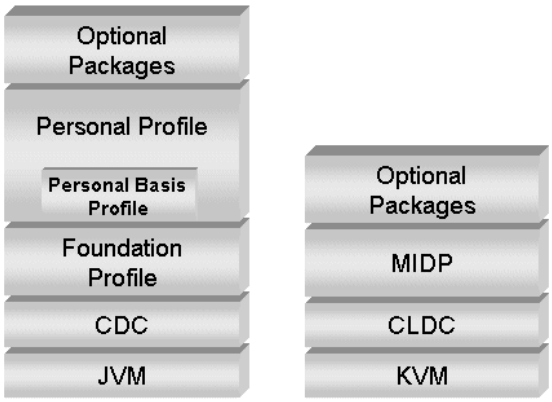

Figura 2.1: Arquitetura da plataforma Java 2 Micro Edition

Configurações são compostas de uma máquina virtual Java e um conjunto mínimo de bibliotecas de classes. Elas provêm um funcionalidade básica para um determinado conjunto de dispositivos que têm em comum algumas características tais como conectividade a redes e limitações de memória. Atualmente há duas configurações J2ME: a CLDC (*Connected Limited Device Configuration*, Configuração

de Dispositivo Limitado e Conectado) [50], direcionada a dispositivos com maiores limitações; e a CDC (*Connected Device Configuration*, Configuração de Dispositivo Conectado) [49], direcionada a dispositivos com maior memória, processadores mais rápidos e maior largura e confiabilidade de rede.

Perfis são conjuntos de APIs (*Application Program Interface*, Interface de Programação de Aplicação) que definem o modelo de ciclo de vida de aplicações, a interface de usuário e o acesso a propriedades específicas de dispositivos. Atualmente, existem três perfis: MIDP (*Mobile Information Device Profile*, Perfil de Dispositivo de Informação Móvel) [59], construído para ser usado sobre o CLDC; FP (*Foundation Profile*, Perfil Base), contruído para ser o perfil mais simples para ser usado sobre o CDC; e o PF (*Personal Profile*, Perfil Pessoal), construído para ser o perfil mais completo para ser usado sobre o CDC. Este perfil inclui todo o AWT (*Abstract Window Toolkit* [48], Caixa de Ferramentas de Janelas Abstratas) de Java 2 Standard Edition [54].

A plataforma Java 2 Micro Edition é suportada por mais de 400 diferentes modelos de aparelhos celulares [63] e em PDAs utilizando o sistema operacional PalmOS.

2.2.2 PalmOS

A plataforma PalmOS [41], da PalmSource, Inc., é específica para PDAs. A figura 2.2 mostra sua arquitetura. Aplicações nativas são escritas na linguagem C com APIs específicas para o sistema operacional PalmOS. Uma característica muito interessante é a tecnologia HotSync™, que permite fazer a sincronização e cópia de todos os dados do dispositivo para um computador pessoal apenas apertando o botão HotSync™do dispositivo. Aplicações Palm também podem trocar dados com aplicações em um computador pessoal através de *conduits* (canos). Eles são *plugins*, escritos pelo desenvolvedor da aplicação, que também são executados quando o botão HotSync™é apertado.

Atualmente, existem duas APIs diferentes para PalmOS: a 68K e a PalmOS Garnet. A primeira, utilizada em PDAs que usam processadores da família Motorola 68000, podem ser executadas em qualquer dispositivo que use o sistema operacional PalmOS. Por outro lado, a API PalmOS Garnet (suportada pelo PalmOS

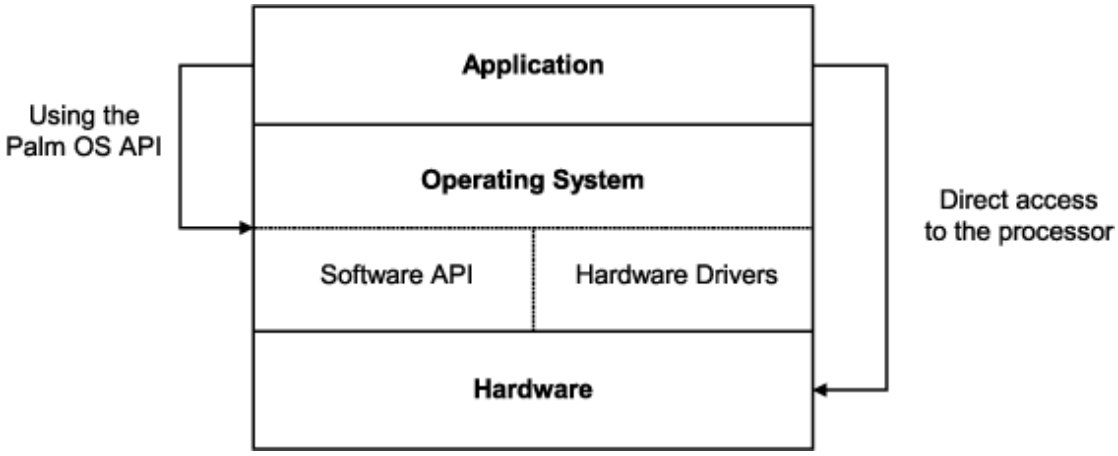

Figura 2.2: Arquitetura da plataforma PalmOS

versão 5 e superiores) é executada sobre processadores da família ARM [6] de forma nativa. Isto permite que a execução de funções da API seja feita de forma direta no processador, não passando pelo sistema operacional e permitindo que ela execute tão rápido quanto o processador permite.

2.2.3 Symbian OS

O Symbian OS é um sistema operacional produzido pela Symbian Ltd. especialmente para telefones celulares [64]. A figura 2.3 mostra sua arquitetura.

Suas aplicações nativas são escritas em C++ utilizando APIs específicas. A conectividade com computadores pessoais é suportada através de SyncML (*(Synchronization Markup Language)*, Linguagem de Marcação de Sincronização) [36], um padrão aberto de sincronização de informação independente de plataforma.

Sua arquitetura é formada por uma camada de abstração de *kernel* e *hardware*, uma de serviços básicos (incluindo sistema de arquivos), uma de serviços de sistema operacional (tais como serviços gráficos e de conectividade com computadores pessoais), uma de serviços de aplicação (tais como mensagens, agenda, navegação Web e sincronização de dados) e uma de interface gráfica de usuário.

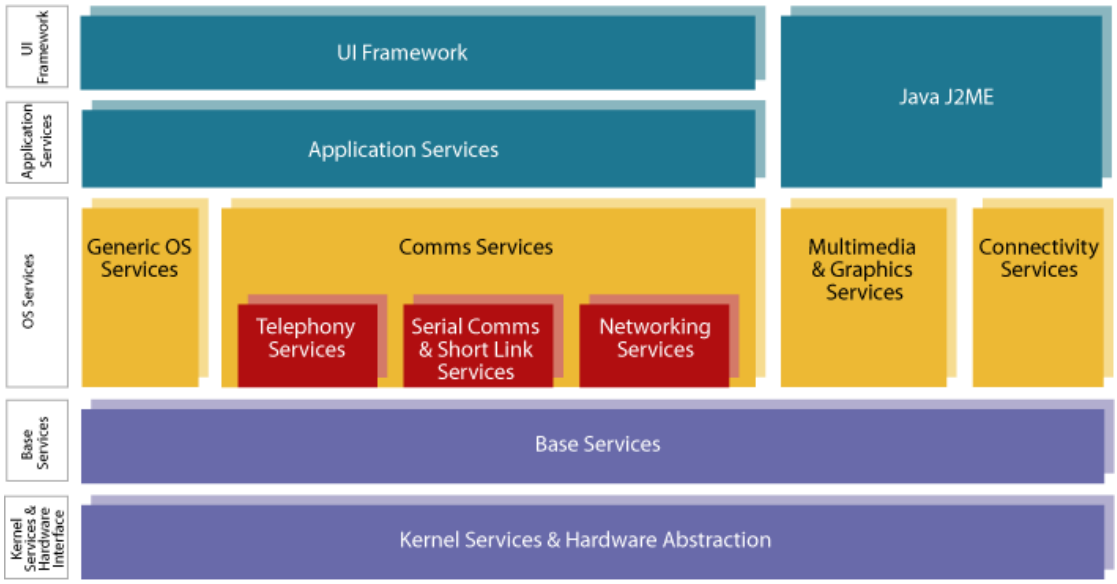

Figura 2.3: Arquitetura da plataforma Symbian OS

2.2.4 Windows CE

O Windows CE é um sistema operacional produzido pela Microsoft Corporation especialmente para dispositivos móveis ou embutidos [30]. A figura 2.4 mostra sua arquitetura.

Suas aplicações nativas são escritas em C++ utilizando APIs específicas. A conectividade com computadores pessoais é suportada através da tecnologia ActiveSync [27].

Sua carecterística mais interessante é a possibilidade de escolha de quais componentes do sistema operacional WindowsCE serão realmente colocadas no dispositivo. Isto é feito através da ferramenta Platform Builder, parte do Microsoft .NET Compact Framework [28], ferramenta de desenvolvimento do Windows CE.

2.2.5 Waba

Waba [66], da Wabasoft Inc., é uma plataforma de programação para dispositivos móveis. Apesar da linguagem utilizada ser um subconjunto de Java, Waba não é uma implementação desta linguagem, tendo uma máquina virtual, formato de código binário e um conjunto de classes fundamentais próprias. Ela foi dese-

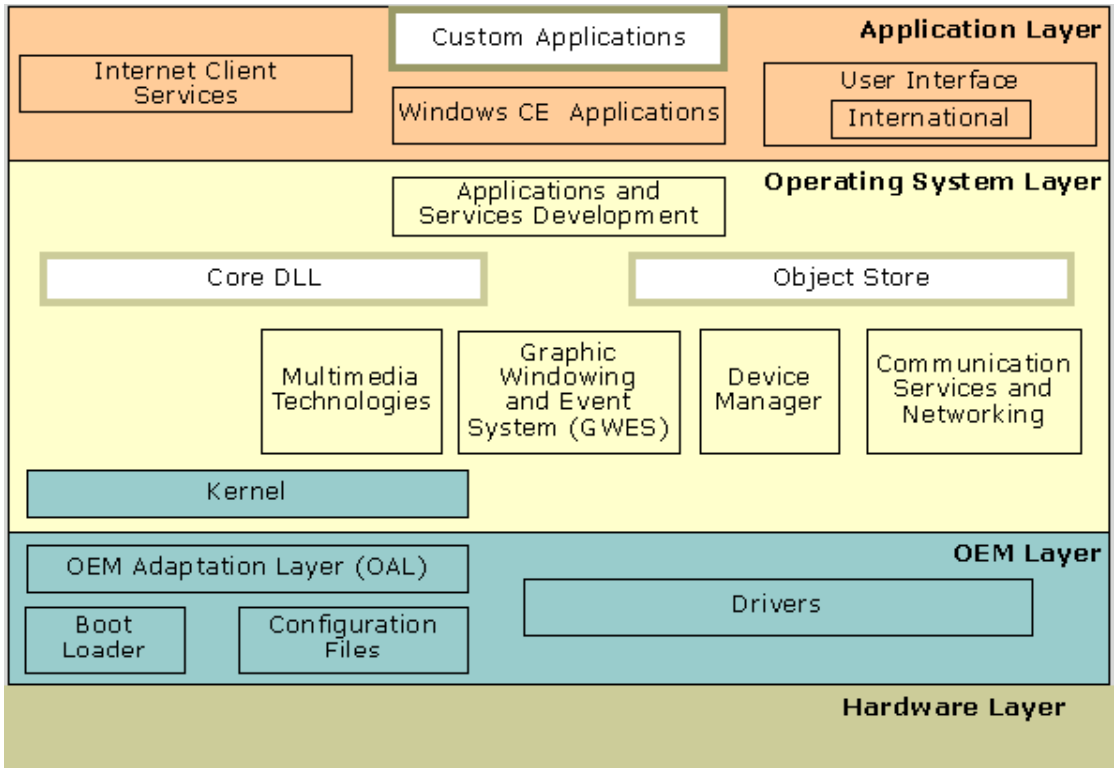

Figura 2.4: Arquitetura da plataforma Windows CE

nhada de forma que o desenvolvimento de aplicações pudesse ser feito em qualquer ferramenta utilizada para Java.

A linguagem, a máquina virtual e o formato de código objeto Waba foram criados de forma otimizada para dispositivos móveis. Há máquinas virtuais cujo tamanho é menor que 64 kilobytes e que executam programas utilizando menos que 10 kilobytes de memória.

Tal como Java, programas Waba, depois de compilados, podem ser executados sem nenhuma alteração em qualquer plataforma que tenha uma máquina virtual Waba. A *garbage collection* e a checagem de limites de vetores também são herdados de Java.

Estão disponíveis *kits* de desenvolvimento Waba para Microsoft Windows [31], Linux, Solaris [56], AmigaOS [1] MacOS [5] e OS/2 [21]. Há implementações da máquina virtual para as plataformas WindowsCE [30], PalmOS [41], e iPaq [19].

2.2.6 BREW

BREW (*Binary Runtime for Wireless Environment*, Código Binário de Tempo de Execução para Ambientes sem Fio) é uma plataforma de aplicações para telefones celulares criada pela Qualcomm [43]. Esta plataforma é uma camada de *software* que é executada sobre o *hardware* do dispositivo. Atualmente, apenas modelos de telefones celulares que possuem o *chipset* Qualcomm MSM são suportados.

As aplicações BREW nativas podem ser escritas nas linguagens C e C++. Uma biblioteca provê o acesso aos recursos do aparelho tais como a tela, alocação de memória e memória persistente. Uma máquina virtual Java está sendo desenvolvida para rodar sobre BREW.

Ao contrário de Java 2 Micro Edition, não é possível construir aplicações BREW utilizando apenas ferramentas gratuitas. É necessário o uso do Microsoft Visual C++ [29] para desenvolver estas aplicações.

Capítulo 3

Trabalhos relacionados

Neste capítulo é apresentada uma lista não exaustiva de ferramentas para desenvolvimento de aplicações para dispostivos móveis. Cada uma delas é descrita brevemente. Uma comparação entre estas ferramentas e o MultiMAD é apresentada na seção 4.2.

3.1 Mobile Aplication Builder (MAB)

O MAB (*Mobile Application Builder*, Construtor de Aplicações Móveis) [32] é uma ferramenta projetada para o desenvolvimento rápido de aplicações para dispositivos móveis. A ferramenta é baseada em uma interface de manipulação, onde o usuário define o fluxo da aplicação através de recursos de arrastar-e-soltar e folhas de propriedades. São utilizadas três unidades de interação com o usuário final, utilizadas como blocos de construção de aplicações: lista, texto e formulário. Uma vez que a especificação da aplicação esteja terminada, ela é submetida para um ou mais geradores de aplicações. Estes geradores são componentes de *software* com interface padronizada que tomam a especificação de uma aplicação e geram código-fonte que a implemente. MAB provê dois geradores, sendo um para WAP 1.1 e outro para a plataforma J2ME. O código-fonte gerado é Java em ambos os geradores.

As vantagens do MAB são a especificação da aplicação de forma gráfica e,

principalmente, a possibilidade de desenvolvimento de aplicações para diversas plataformas. Suas principais desvantagens são o reduzido número de blocos de construção de aplicações e a não possibilidade de uso de recursos específicos de plataformas. Outra desvantagem é a forma utilizada pela interface gráfica da ferramenta para a passagem de parâmetros para os blocos de construção. Ela é sempre feita através de *strings* contendo pares da forma `nome=valor` separados pelo caractere '`;`'. Em alguns contextos, deve-se passar uma lista de listas de parâmetros, cada uma separada da outra pelo caractere '`|`'.

3.2 BeanWatcher

O BeanWatcher [14] é uma ferramenta de desenvolvimento de aplicações de monitoramento que utiliza um modelo de componentes que padroniza os elementos monitorados. As aplicações podem ser geradas em Java 2 Standard Edition (J2SE) [54], Java 2 Micro Edition (J2ME) [57] e C++. Os componentes utilizados pelo BeanWatcher são baseados no PECOS (*PErvasive COmponent System*, Sistema de Componentes Pervasivos), um modelo destinado à aplicações em sistemas embutidos [33].

A ferramenta é composta pelos módulos repositório, apresentação e processamento. O módulo repositório contém os componentes implementados nas linguagens de programação suportadas pela ferramenta. O módulo de apresentação apresenta uma interface gráfica na qual o usuário pode criar novos componentes e reimplementá-los em outras linguagens e uma área de trabalho onde a aplicação é montada. O módulo de processamento é responsável por criar e atualizar os componentes e montar e gerar as aplicações feitas através da ferramenta.

3.3 NetBeans

O NetBeans [55] é um ambiente de desenvolvimento integrado (*Integrated Development Environment*, IDE) para a linguagem de programação Java.

Um de seus módulos, o Mobility Pack [51], provê várias facilidades para o desenvolvimento de aplicações para a plataforma Java 2 Micro Edition. As principais

estão na lista a seguir.

- Editor gráfico de aplicações J2ME, possibilitando a criação da interface gráfica de usuário e o fluxo da aplicação usando arrastar-e-colar (*drag-and-drop*). O código gerado não pode ser editado manualmente no NetBeans [52].
- Assistente (*wizard*) para a criação aplicações cliente/servidor (cliente J2ME, servidor Java 2 Enterprise Edition [53]), gerando código de ambos os lados.
- Integração com o Java 2 Micro Edition Wireless Toolkit 2.2 [58] e outros emuladores de dispositivos J2ME.
- Possibilidade de se escrever blocos de código específicos para determinados dispositivos.
- Internacionalização de aplicações.
- Otimização e obfuscação[1] de código-objeto.
- Simulação de *download* de aplicações via OTA (*Over-The-Air provisioning*, Entrega Através do Ar) [60].
- Geração de *script* Ant [2] para compilação, empacotamento e execução de aplicações.

3.4 AVIDRapidTools

AVIDRapidTools (ART) [8], da AVIDWireless, é um *framework* Java criado com o objetivo de ajudar desenvolvedores a construir aplicações para dispositivos móveis rapidamente. Este *framework* foi construído de forma a poder trabalhar com o conjunto atual de IDEs, reduzindo-se assim as curvas de aprendizado associadas a se trabalhar com novos ambientes ou pacotes.

As aplicações são escritas de forma independente de dispositivo. Todo o trabalho de adaptação de conteúdo ao dispositivo é feito de forma transparente pela

[1] Obfuscação é o nome dado à alteração de código-objeto de forma a dificultar ou impossibilitar a realização de engenharia reversa sobre ele.

ferramenta. Os blocos de construção são basicamente os mesmos da ferramenta MAB, adicionando-se hiperligações (*links*), imagens, tabelas e a possibilidade do uso de texto formatado. A comunicação entre o cliente e o servidor pode ser feita através de HTML [68], WML [37, 38] ou outras linguagens de marcação. A documentação da ferramenta afirma que ela suporta mais de 170 diferentes dispositivos móveis entre telefones celulares, PDAs, *pagers* e *laptops*.

Não é possível escrever aplicações *off-line* utilizando-se este *framework*, pois ele é baseado na arquitetura cliente-servidor, executa exclusivamente no servidor e apenas se comunica com o cliente através de linguagens de marcação.

3.5 Simplicity

Simplicity for Mobile Devices [11] e Simplicity for PalmOS Platform [13], ambas da Data Representations, são ferramentas de desenvolvimento rápido de aplicações J2ME. A primeira é específica para telefones celulares e a segunda para PDAs que utilizam a plataforma PalmOS, sendo esta a única diferença considerável entre elas.

Suas principais características são o simulador integrado de dispositivo e as tecnologias Execution-On-The-Fly, Visual Composer e Sourcerer. O simulador integrado, junto com o Execution-On-The-Fly, permite que qualquer alteração feita na aplicação seja imediatamente refletida na sua simulação. O Visual Composer provê ao desenvolvedor a facilidade de se criar a interface de usuário da aplicação através de cliques do *mouse*, existindo a possibilidade da personalização de cada componente. Cada alteração em um componente aparece instantaneamente no simulador. Já o Code Sourcerer escreve automaticamente mais de 50000 trechos de código Java através de perguntas ao usuário em linguagem natural. A ferramenta também possui um editor de código-fonte Java e um depurador embutido.

O Simplicity for Mobile Servers [12] é um produto adicional, muito semelhante aos já citados nesta seção, que possui todas as suas características mais a possibilidade de lidar com aspectos de um eventual servidor da aplicação tais como acesso a bancos de dados pelo dispositivo móvel e transmissão de dados entre servidor e cliente.

3.6 Speedware MobileDev

Speedware MobileDev [47] é um ambiente de desenvolvimento para a plataforma WAP. Ele provê assistentes (*wizards*) para criar rapidamente baralhos (*decks*) e cartões (*cards*) WML [37, 38] ou HDML [26] como objetos num mapa de aplicação. Enquanto os cartões vão sendo geradas, ao lado de seu código WML ou HDML correspondente, o mapa é atualizado com os novos objetos, mostrando as suas relações na forma de diagramas de fluxo (*flow diagrams*).

O MobileDev também provê um explorador de aplicações, um explorador de bancos de dados, um editor de documentos e um simulador de navegador WAP integrado. Outros navegadores podem ser incorporados.

Esta ferramenta também gera o código Perl ou ASP necessário para criar aplicações WAP dinâmicas. Ela também gera o código necessário para se conectar os baralhos, além de permitir que aplicações WAP acessem bancos de dados dinamicamente.

3.7 Seasam Time

O Seasam Time [45], da Seasam Time Oy, é uma ferramenta que combina duas aplicações diferentes: o Seasam Time, um ambiente de desenvolvimento integrado; e o Seasam Application Management Server (SAMS), uma plataforma para aplicações Seasam Time.

A IDE provê várias funcionalidades. O editor de documentos permite a edição de páginas individuais e blocos virtuais de construção de aplicações tais como lista de seleção, resultados de consultas a bancos de dados e hiperligações (*links*). Não é necessário escrever código-fonte, já que ele é todo gerado automaticamente. O mapa de aplicações provê uma visão geral da aplicação, das relações e tipos de páginas. O SQL Builder Editor permite a criação de aplicações dinâmicas que acessam bancos de dados. O emulador permite testar a aplicação durante qualquer momento do processo de desenvolvimento. Aplicações desenvolvidas no Seasam Time podem ser estendidas pela criação de componentes personalizados ou de terceiros.

O SAMS permite que a mesma aplicação seja executada em diferentes dispo-

sitivos móveis. Num primeiro momento, ele detecta qual é o tipo do dispositivo. Tendo esta informação, o SAMS determina qual linguagem de marcação será utilizada (WML [37, 38], HTML [68], XHTML [70] e outras). Desta forma, a mesma aplicação pode ser utilizada em diferentes plataformas de forma automática.

Capítulo 4

Especificação de requisitos

Este capítulo apresenta uma lista não exaustiva de requisitos desejáveis em uma ferramenta de desenvolvimento de aplicações para dispositivos móveis. Além disso, ele mostra, na tabela 4.1, a situação do MultiMAD e de cada uma das ferramentas relacionadas descritas no capítulo 3 em relação a eles. A seção 4.2 apresenta uma discussão sobre as ferramentas em função dos requisitos.

4.1 Lista de requisitos

A lista de requisitos, mostrada abaixo, está dividida em duas partes. A primeira contém os requisitos propostos originalmente por Minelli [32]. Eles são numerados de 1 a 9. A segunda lista contém requisitos propostos por este trabalho, a maioria deles inspirada em recursos oferecidos pelas ferramentas descritas no capítulo 3. Eles são numerados de 10 a 18.

Os elementos da lista de requisitos não estão em nenhuma ordem específica. O termo *projeto* representa uma aplicação enquanto ela está sendo desenvolvida na ferramenta.

1. **Possibilidade de desenvolvimento de aplicações em mais de uma plataforma**
   Com a diversidade de plataformas de desenvolvimento de aplicações para

dispositivos móveis, ficar restrito a apenas uma delas seria uma grande limitação da ferramenta e do mercado das aplicações nela desenvolvidas. A opção de fazer implementações de uma mesma aplicação utilizando diferentes ferramentas faz com que o desenvolvedor tenha que aprender mais de uma ferramenta e implementar mais de uma vez a mesma aplicação.

2. **Disponibilidade de um modelo (conjunto de blocos de construção de aplicações) independente de dispositivo no qual as aplicações sejam descritas**
   A geração de código-fonte e de eventuais arquivos auxiliares de uma aplicação para um dado dispositivo ou plataforma deve ser realizada por geradores de aplicações específicos. A entrada para estes geradores seria a descrição da aplicação no modelo disponibilizado.

3. **Modelagem de geradores de aplicações como componentes de *software* com uma interface padronizada**
   O usuário da ferramenta deve poder adicionar geradores prontos à ferramenta ou escrever e adicionar os seus próprios sem ter que alterar a ferramenta. Como conseqüência, os geradores funcionariam de forma semelhante a *plugins*. Eles podem interagir com o usuário caso isto seja interessante ou necessário.

4. **Facilidade de inserção de código-fonte externo**
   Como esta ferramenta não tem o objetivo de gerar a aplicação inteira, faz-se necessária a inserção do código-fonte que implemente a parte restante. Os geradores de aplicações devem gerar código-fonte de modo que a inserção da parte escrita pelo desenvolvedor seja fácil e que, idealmente, o código-fonte gerado não precise ser alterado pelo desenvolvedor. Desta forma, caso a aplicação seja gerada novamente, não há perda de esforço, já que toda alteração no código-fonte gerado é perdida.

5. **Recursos de armazenamento permanente de dados**
   Muitas aplicações demandam que informações estejam disponíveis entre uma execução e outra. Podemos citar como exemplos agendas de compromissos e de telefones. Desta maneira, seria muito interessante que a ferramenta

provesse alguma forma de armazenamento e acesso a dados persistentes. Nem todos os dispositivos móveis possuem memória permanente para dados de aplicações, porém este problema pode ser resolvido com o armazenamento de informações em um servidor Web, por exemplo.

6. **Visualização de projetos**
   Uma visualização gráfica da aplicação sendo especificada é uma característica que ajuda o usuário a compreendê-la melhor. Um exemplo seria um grafo mostrando as telas da aplicação como vértices e os possíveis fluxos de execução dentre elas como arestas.

7. **Recursos de verificação de aplicações**
   Deve-se garantir a correção da aplicação antes que ela comece a ser utilizada pelos seus usuários finais. Uma forma de se atingir este objetivo é a utilização de depuradores (*debuggers*) durante o desenvolvimento da aplicação. Eles permitem ao desenvolvedor executar a aplicação passo a passo, podendo-se verificar o valor de variáveis para determinar se a execução está correta. Outra forma é a utilização de verificação formal da aplicação. Esta técnica pode ser baseada, por exemplo, na modelagem da aplicação em uma máquina de estados finitos. A partir deste modelo pode-se especificar estados válidos ou inválidos que serão verificados pela ferramenta de verificação formal tal como o NuSMV [9].

8. **Facilidade de uso**
   Uma ferramenta de desenvolvimento de aplicações tende a ser menos utilizada se ela tem uma curva de aprendizado acentuada ou é de difícil utilização. O uso de técnicas de engenharia de usabilidade na interface gráfica de usuário agregaria valor à ferramenta. Além disso, a ferramenta deve ser de fácil instalação e desinstalação.

9. **Geração automática de código de comunicação entre cliente e servidor para aplicações que não sejam baseadas em linguagens de marcação**
   Aplicações baseadas em linguagens de marcação tem a comunicação de dados entre cliente e servidor bastante limitada e já implementada por protocolos

de transporte (WTP [37] ou HTTP [16], por exemplo). Por outro lado, estes protocolos não bastam para outros tipos de aplicações. Se faz então necessária a implementação de código que realize a comunicação de dados com um servidor ou outro dispositivo que faça parte da mesma aplicação distribuída. A geração automática deste código facilitaria muito o trabalho do desenvolvedor.

10. **Possibilidade de criação de aplicações tanto baseadas em linguagens de marcação quanto baseadas em plataformas de programação**
    Aplicações baseadas em linguagens de marcação são tipicamente executadas através da rede utilizando-se um navegador (*browser*) que apenas exibe para o usuário documentos. Tipicamente, não há processamento de dados no cliente, apenas no servidor onde está a aplicação. Por outro lado, aplicações baseadas em plataformas de programação executam processamento no dispositivo, podendo eventualmente trocar dados com um servidor ou outro dispositivo.

11. **Possibilidade de estender o modelo de aplicações provido pela ferramenta**
    Este requisito abre a possibilidade de se criar modelos estendidos para plataformas específicas, tais como PalmOS [41], WAP [37, 38] ou Java 2 Micro Edition [57], sem que a ferramenta tenha que ser alterada. Os modelos deveriam também ser modelados como componentes de *software* com uma interface padronizada. A ferramenta deve prover o modelo base, enquanto o usuário poderá adicionar modelos prontos ou escrever e adicionar o seu próprio. A interface de usuário também pode ser determinada, em todo ou em parte, pelo modelo utilizado no momento.

12. **Portabilidade da ferramenta**
    A ferramenta deve ser escrita de forma a ser portada, com o menor esforço possível, para outras plataformas além daquela na qual ela foi desenvolvida. Ferramentas que executam em apenas uma plataforma têm um universo de usuários limitado.

13. **Utilização de uma linguagem ou plataforma de desenvolvimento**

que tenha uma grande penetração entre os desenvolvedores de aplicações para dispositivos móveis
Este requisito é muito importante pois, na maioria dos casos, a implementação de novos modelos ou geradores de aplicações para a ferramenta deve ser feita na sua linguagem ou plataforma de desenvolvimento.

14. **Internacionalização da ferramenta**
A ferramenta deve ser escrita de forma que a sua tradução para outras línguas seja feita com o menor esforço possível. Este requisito também se aplica a modelos e geradores de aplicações.

15. **Recursos de compilação, empacotamento e execução de aplicações geradas**.
Depois que o código-fonte e eventuais arquivos auxiliares de uma aplicação são gerados, ainda é necessário que ela seja compilada e empacotada de acordo com o dispositivo ou plataforma na qual ela irá ser executada. Para que aplicação possa ser testada, é interessante que ela seja executada em emuladores ou simuladores antes de ser executada nos dispositivos em si. Desta forma, é importante prover facilidades para compilação, empacotamento e execução da aplicação sendo modelada.

16. **Recursos de prototipagem**
É muito interessante que uma ferramenta que gere parte da implementação de uma aplicação também possa gerar uma implementação temporária do resto da implementação. Desta forma, junto com recursos de compilação, empacotamento e execução de aplicações em emuladores e simuladores, a ferramenta pode gerar um protótipo da aplicação e executá-lo. Isto permite ao usuário, mesmo leigo na plataforma onde a aplicação está sendo gerada, ter uma primeira visão do aplicação mesmo sem ter escrito nenhuma linha de código.

17. **Emulação de projetos**
A incorporação de um emulador de aplicações na ferramenta acelera o processo de desenvolvimento por permitir que o usuário consiga visualizar o resultado de alterações no projeto sem sair da ferramenta.

18. **Possibilidade de edição de aplicações de forma gráfica**
    A possibilidade de edição de aplicações de forma gráfica, utilizando-se, por exemplo, arrastar e soltar (*drag and drop*), é uma característica que facilitaria muito o trabalho do desenvolvedor, já que ele estaria editando a aplicação ao mesmo tempo em que a visualiza.

Requisito	MAB	Bean-Watcher	Net-Beans	AVID-Rapid-Tools	Simplicity	Mobile-Dev	Seasam Time	MultiMAD
1	√	√	×	√	×	√	√	√
2	√	√	○	√	○	√	√	√
3	√	√	○	○	×	×	√	√
5	×	×	×	×	√	√	√	×
6	√	√	√	○	√	√	√	√
7	×	×	√	○	√	×	×	×
9	×	√	√	○	√	○	○	×
10	√	×	×	×	×	×	×	√
11	×	√	×	×	×	×	√	√
12	×	√	√	√	√	×	×	√
13	×	○	○	○	○	○	√	√
14	×	×	√	○	×	×	×	√
15	×	×	√	○	√	√	√	√
16	×	×	√	○	√	√	√	√
17	×	×	×	○	√	√	√	×
18	×	√	√	○	√	×	√	×

Legenda: √ - satisfaz × - não satisfaz ○ - não se aplica

Tabela 4.1: Comparação das ferramentas relacionadas e o MultiMAD em função dos requisitos

Observações:

- Como os requisitos 4 e 8 se referem a um conceito subjetivo, neste caso a facilidade, eles não foram incluídos nesta comparação.
- Requisito 12: As ferramentas MAB, MobileDev e Seasam Time estão disponíveis apenas para o sistema operacional Microsoft Windows. Por outro lado, as outras ferramentas são escritas em Java, sendo portáveis para praticamente todos os sistemas operacionais atuais (aqueles que têm uma implementação da máquina virtual Java).
- Requisito 13: Foi considerado que este requisito não se aplica a ferramentas que não permitem a adição de modelos de aplicação e geradores (todas as ferramentas aqui citadas, exceto MAB e MultiMAD). Além disso, considera-se que MAB não satisfaz este requisito por ter sido escrita em Delphi, linguagem que não é utilizada para o desenvolvimento de aplicações para dispositivos móveis.

4.2 Discussão das ferramentas em função dos requisitos

Dentre os 18 requisitos listados na tabela 4.1, o MultiMAD e o Seasam Time foram as ferramentas que mais atendem requisitos, 11, seguido por Simplicity (9), BeanWatcher e NetBeans (8), MobileDev (7), MAB (5) e AVIDRapidTools (3). A lista de requisitos apresentada neste capítulo é extensa, fazendo que mesmo uma ferramenta que satisfaça parte deles não deixe de ser interessante.

O BeanWatcher destoa das outras ferramentas discutidas neste trabalho por ter sido desenhada para o desenvolvimento de um determinado tipo de aplicação, as de monitoração, enquanto as outras não são especializadas em nenhum tipo.

O AVIDRapidTools, sendo um *framework*, possui poucos recursos em relação às outras ferramentas, o que já era esperado.

Sendo inspirado inicialmente no MAB, o MultiMAD também têm os conceitos de blocos de construção de aplicações genéricos (modelo genérico de aplicações)

e o de geradores. O grande diferencial do MultiMAD é a possibilidade de adição de blocos que sejam específicos para uma determinada plataforma. Além disso, o MAB não possui recursos de compilação, empacotamento e execução de aplicações e executa apenas em um sistema operacional.

O MobileDev possui recursos interessantes, porém serve apenas para desenvolvimento de aplicações baseadas em linguagens de marcação.

O NetBeans, em conjunto com o Mobility Pack, é uma ferramenta de desenvolvimento com uma grande variedade de recursos. Por outro lado, ele pode ser utilizado apenas para o desenvolvimento de aplicações para a plataforma Java 2 Micro Edition.

O Seasam Time é uma ferramenta muito interessante, tendo uma filosofia diferente das outras: o usuário não escreve código-fonte, montando sua aplicação a partir de componentes prontos. Suas desvantagens têm como origem exatamente esta característica. Caso o usuário utilize apenas componentes prontos, as possibilidades de especificação da lógica da aplicação são muito limitadas, sendo praticamente apenas possível fazer operações sobre um banco de dados. Caso contrário, deve-se criar componentes específicos, o que não pode ser feito dentro da ferramenta. Além disso, não é possível desenvolver aplicações baseadas em plataformas de programação.

O MultiMAD, que será descrito nesta dissertação, é uma ferramenta que satisfaz muitos requisitos e pode ser estendida para satisfazer outros. Além disso, junto com o BeanWatcher, é a única ferramenta com recursos que permitem que novos blocos de construção sejam adicionados e que pode ser usada para o desenvolvimento de aplicações em várias plataformas. Isto permite ao MultiMAD ser facilmente estendido para poder suportar o desenvolvimento de aplicações para outras plataformas, inclusive aquelas que ainda serão criadas, não limitando as possibilidades de seus usuários.

Capítulo 5

MultiMAD

Este capítulo descreve o MultiMAD, partindo dos seus conceitos fundamentais (seção 5.1), passando pela arquitetura interna (seção 5.2), interface gráfica (seção 5.3) e forma de uso (seção 5.4).

5.1 Conceitos fundamentais

Esta seção discute os quatro conceitos fundamentais do MultiMAD: elemento, projeto, modelo de aplicações e gerador de aplicações. Detalhes sobre a implementação destes conceitos podem ser encontrados na seção 5.2

5.1.1 Elemento

Elementos de um projeto são as telas que formam a interface gráfica da aplicação. Cada tela no MultiMAD é indivisível, tem um determinado tipo, um nome (utilizado no código gerado) e um título (mostrado para o usuário).

Elementos relacionam-se entre si através do conceito de próximo elemento: o elemento A é o próximo elemento de B se, após o usuário terminar sua interação com B, o próximo elemento a ser mostrado é A. O próximo elemento pode ser estático, definido pelo usuário durante a especificação da aplicação; ou dinâmico, definido pelo código implementado pelo usuário. Um mesmo elemento pode ter

mais de um próximo elemento. Menus, por exemplo, possuem vários próximos elementos, tipicamente um para cada um de seus itens.

Existem também elementos especiais. Eles representam comandos para a plataforma na qual a aplicação está executando, tais como sair da aplicação ou sincronizar dados com um computador. Desta forma, eles apenas podem ser o próximo elemento de elementos normais, não podendo ser editados.

5.1.2 Projeto

Projetos são descrições de aplicações. Eles contêm um nome, uma descrição, um conjunto de elementos, uma referência para o primeiro elemento que será apresentado ao usuário quando a aplicação é iniciada (elemento inicial), o nome do diretório no qual os arquivos gerados serão colocados, uma referência para o modelo de aplicação utilizado, a configuração deste modelo (caso exista), o gerador utilizado (caso algum esteja sendo utilizado) e a sua configuração (caso exista).

5.1.3 Modelo de aplicações

Os modelos de aplicações do MultiMAD são um conjunto de blocos de construção que uma determinada plataforma pode oferecer para o desenvolvimento de aplicações. Este conjunto pode tanto ser específico para alguma plataforma de aplicações para dispositivos móveis quanto genérico. A vantagem do primeiro tipo é a possibilidade de melhor utilização das funcionalidades do dispositivo. Por outro lado, projetos que seguem modelos específicos não são gerados para outras plataformas sem adaptações.

Atualmente, cada modelo de aplicações define um conjunto de tipos de elementos, um conjunto de tipos de campos de formulário e um conjunto de elementos especiais. Um modelo só deve oferecer um determinado tipo de campo se a plataforma à qual ela se refere não permitir que o usuário entre valores inválidos (por exemplo, o valor `abc` para um campo de tipo número).

A possibilidade de se ter diferentes modelos de aplicações e geradores faz com que esta ferramenta seja, ao mesmo tempo, genérica e específica em relação a plataformas de desenvolvimento de aplicações para dispositivos móveis. Genérica por-

que o MultiMAD provê um modelo genérico, descrito na seção 6.1, cujos tipos de elementos foram definidos com o objetivo de serem suportados pela maioria (idealmente, pela totalidade) das plataformas de aplicações para dispositivos móveis. Específica porque a ferramenta permite que modelos de aplicações específicos para determinadas plataformas sejam adicionados sem que ela seja alterada.

Modelos de aplicações podem ser configuráveis ou não. Estas configurações são sempre específicas para um determinado projeto e são informadas ao gerador no momento oportuno. O modelo genérico provido pela ferramenta não é configurável. Além de configurações específicas por projeto, modelos podem ter configurações gerais, chamadas de propriedades. Um exemplo de propriedade é, num gerador de aplicações para a plataforma Java 2 Micro Edition [57], o diretório onde está instalado o J2ME Wireless Toolkit [58].

5.1.4 Gerador de aplicações

Geradores de aplicações, chamados aqui simplesmente de geradores, são componentes de *software* que recebem um projeto e geram o código-fonte e eventuais arquivos auxiliares que implementam parcialmente uma aplicação para uma determinada plataforma de dispositivos móveis.

Tal como modelos de aplicações, geradores podem ser configuráveis e ter propriedades.

5.2 Arquitetura

A arquitetura interna do MultiMAD foi desenvolvida com dois objetivos principais: extensibilidade e flexibilidade. A extensibilidade é alcançada pela modelagem de modelos de aplicações e geradores como *plugins*, podendo ser facilmente adicionados sem alteração, recompilação ou resinstalação da ferramenta. A arquitetura é flexível pois impõe poucas e pequenas restrições sobre como modelos de aplicações e geradores podem ser modelados e implementados.

A ferramenta pode ser dividida em quatro partes principais. A primeira é o núcleo do MultiMAD, descrito na seção 5.2.1. Este núcleo faz a integração das ou-

tras partes recebendo comandos, delegando ações e armazenando informações sobre o seu estado. A segunda provê um ou mais modelos de descrição de aplicações (chamados simplesmente de modelos de aplicações). Neste documento, esta descrição será chamada de projeto. Esta parte é descrita na seção 5.2.2. O armazenamento do projeto em memória secundária é também tarefa desta parte. A terceira provê geradores de aplicações. Eles recebem como entrada um projeto e têm como saída o código-fonte que o implemente parcial ou totalmente. Esta parte é descrita na seção 5.2.3. De acordo com o contexto, outros arquivos podem ser gerados. A quarta parte é a interface gráfica de usuário, descrita na seção 5.3. A figura 5.1 mostra as relações entre as principais classes do MultiMAD.

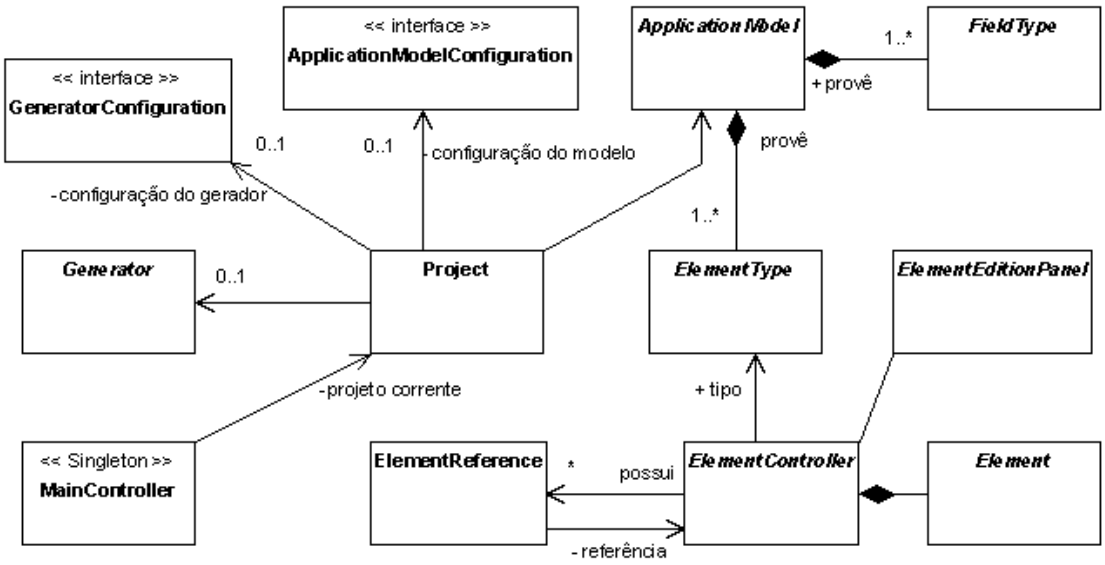

Figura 5.1: Diagrama das classes principais do MultiMAD

5.2.1 Núcleo do MultiMAD

O núcleo de processamento do MultiMAD é a classe `thiagohp.multimad.core.MainController`. Esta classe, um Singleton (classe que possui apenas uma instância) [18], é responsável por:

- Inicializar a ferramenta;

- Manter o estado da ferramenta;
- Processar de todos os comandos requisitados através de menus e combinações de teclado;
- Implementar todas as operações sobre projetos; abrir, fechar, salvar, salvar como, adicionar elemento, remover elemento; alterar modelo de aplicações sendo seguido, alterar gerador a ser usado;
- Localizar e carregar modelos de aplicações e geradores; e
- Notificar objetos interessados quando algum certos tipos de eventos ocorrem. Os tipos são:
  - Alteração de projeto (adição, remoção e alteração de elemento, troca e armazenamento de projeto corrente).
  - Alteração de modelo de aplicações sendo seguido pelo projeto corrente.
  - Alteração de gerador sendo utilizado pelo projeto corrente
  - Alteração da língua sendo utilizada.

As classes que implementam conceitos do MultiMAD serão descritas a seguir Todas elas fazem parte do pacote `thiagohp.multimad.core`, a não ser que que haja indicação em contrário.

`Element`

Objetos da classe abstrata `Element` representam elementos de aplicação. Ela apenas define o conjunto mínimo de campos e métodos que todos devem ter. Instâncias desta classe devem apenas armazenar informações, deixando todas as tarefas de validação e consistência para instâncias de `ElementController`.

`Element` tem dois métodos abstratos: `abstract ElementType getType()`, que retorna o tipo deste elemento; e `abstract boolean needsExternalCode()`, que diz se este elemento precisa de implementação a ser feita pelo usuário do MultiMAD. Seus outros métodos retornam ou alteram os campos `name` (nome do elemento), `title` (título) e `description` (descrição).

`ElementController`

Instâncias da classe abstrata `ElementController` têm a função de agir como *proxies* para instâncias de `Element`. A relação entre estas duas classes é 1:1. Cada `ElementController` tem também as tarefas de validar e manter a consistência do `Element` correspondente.

Esta classe declara dois campos privados: `element` e `editionPanel`. O primeiro é o `Element` correspodente. O segundo é uma instância da classe abstrata `ElementEditionPanel`, descrita na seção 5.2.1. É através dela que o usuário da ferramenta edita o elemento.

Seguindo o modelo MVC (`Model-View-Controller`, Modelo-Visão-Controlador), ninguém deve acessar de forma direta objetos que armazenam dados de uma entidade. Isto deve ser feito através de um objeto controlador. Desta forma, nenhuma classe deve acessar diretamente um `Element`, devendo utilizar o `ElementController` correspondente. Para realizar esta tarefa, `ElementController` tem métodos que delegam suas chamadas para os métodos correspondentes de `Element`.

A classe `ElementController` define três métodos que devem ser implementados pelas suas subclasses: `boolean isReady()`, que retorna verdadeiro se o elemento correspondente está pronto para ter seu código gerado; `ElementReference[] getElementReferences()`, que retorna uma lista com todas as referências para outros elementos que o elemento controlado tem; e `String getElementReferenceLabel(int n)`, que retorna uma *string* que descreve a `n`-ésima referência que o elemento controlado tem.

`ElementReference`

Para que o MultiMAD possa ter controle total sobre as referências (relacionamentos) entre os elementos, elas não devem ser feitas de forma direta, e sim através de instâncias da classe `ElementReference`. Isto facilita a leitura de projetos de arquivos e permite que, quando o usuário requisita a remoção de um elemento, o núcleo do MultiMAD possa determinar se há elementos que fazem referência a ele e alertar o usuário deste fato. Instâncias desta classe podem ser referências nulas (referência a nenhum elemento).

Nome	Descrição
`ElementController getReference()`	Retorna o `ElementController` referenciado ou `null` caso esta referência seja nula.
`void setReference( ElementController reference)`	Altera o elemento referenciado por este objeto. Se reference for `null`, este objeto passa a ser uma referência nula.
`org.jdom.Element getReferenceAsXML()`	Retorna uma representação em XML deste objeto.
`static void createElementReferenceFromXML( org.jdom.Element element, ElementReference reference)`	Faz com que `reference` tenha o valor descrito em XML por `element`. Este método faz a operação inversa de `getReferenceAsXML()`.

Tabela 5.1: Alguns métodos da classe `ElementReference`

Os principais métodos da classe `ElementReference` estão descritos na tabela 5.1.

`ElementType`

A classe abstrata `ElementType` representa tipos de elementos de aplicação no MultiMAD. Cada instância de `Element` e de `ElementController` está associada a algum `ElementType`.

Instâncias desta classe têm as tarefas de criar novos elementos de aplicação, gerar uma descrição XML de elementos e fazer a operação inversa. Dentro do MultiMAD, esta descrição é implementada na forma de um objeto da classe `org.jdom.Element`, parte integrante do *parser* JDOM [20]. Opcionalmente, `ElementType`s podem especificar um ícone para o tipo representado.

Os principais métodos desta classe estão descritos na tabela 5.2.

`FieldType`

A classe abstrata `FieldType` representa tipos de campos de formulários. Seus métodos abstratos estão descritos na tabela 5.3.

Nome	Descrição
`abstract String getName()`	Retorna o nome do tipo de elementos.
`abstract String getDescription()`	Retorna uma descrição do tipo de elementos.
`abstract ElementController createElementControllerFromXML( org.jdom.Element element)`	Retorna um `ElementController` que controla um `Element` criado a partir da descrição XML contida no parâmetro `element` do método. Operação inversa de `getElementAsXML`.
`abstract org.jdom.Element getElementAsXML( ElementController elementController)`	Retorna uma descrição XML do elemento de aplicação controlado por `elementController`. Operação inversa de `createElementControllerFromXML`.
`abstract javax.swing.Icon getIcon()`	Retorna um ícone que será usado para representar graficamente o tipo de elemento. Pode retornar `null` se não há um ícone para este tipo de elemento.
`abstract ElementController newElementController( String elementName, String elementDescription)`	Cria um novo `ElementController` cujo `Element` controlado terá o nome contido em `elementName` e a descrição contida em `elementDescription`.

Tabela 5.2: Métodos da classe `ElementType`

Nome	Descrição
`abstract String getName()`	Retorna o nome do tipo de campo de formulário.
`abstract Class getValueClass()`	Retorna a classe que representa os valores deste campo. Este método é utilizado para se decidir qual componente de interface gráfica será utilizado para se mostrar e editar o valor inicial de campos deste tipo.
`Object valueOf( String initialValue)`	Converte o valor inicial contido na `String initialValue` e a converte para a classe que representa os valores deste campo.
`abstract String getInitialValueToApplicationInfo( Object initialValue)`	Converte o valor inicial contido em `initialValue` para uma `String` que vai ser usada para representá-lo no código gerado.
`abstract String getInitialValueToApplicationInfo( Object initialValue)`	Converte o valor inicial contido em `initialValue` para uma `String` que vai ser usada para representá-lo no código gerado.

Tabela 5.3: Métodos abstratos da classe `FieldType`

`ElementEditionPanel`

A classe abstrata `ElementEditionPanel` (pacote `thiagohp.multimad.gui`) define o conjunto mínimo de campos e métodos de todos os objetos de interface gráfica utilizados para se editar elementos de aplicação. Instâncias desta classe têm uma relação 1:1 com instâncias de `ElementController`.

Os métodos abstratos de `ElementEditonPanel` estão descritos na tabela 5.4.

Nome	**Descrição**
`abstract void doSaveData()`	Atualiza o elemento editado com os valores definidos pelo usuário neste `ElementEditionPanel`.
`abstract void doValidateData()`	Valida os valores dados para o elemento pelo usuário. Deve levantar a exceção `IllegalArgumentException` caso algum valor não seja válido.
`abstract void doUpdateElementDataView()`	Atualiza este `ElementEditionPanel` com os valores do elemento sendo editado.
`abstract void setElementController( ElementController elementController)`	Especifica o `ElementController` que controla o elemento de aplicação que será editado.
`abstract ElementController getElementController( ElementController elementController)`	Retorna o `ElementController` que controla o elemento de aplicação sendo editado.

Tabela 5.4: Métodos abstratos da classe `ElementEditionPanel`

5.2.2 Modelos de aplicações

Os modelos de aplicações do MultiMAD são instâncias da classe abstrata `ApplicationModel` (pacote `thiagohp.multimad.core`). A tabela 5.5 mostra os métodos desta classe.

Método	**Descrição**
`abstract String getName()`	Retorna o nome deste modelo.
`abstract String getDescription()`	Retorna a descrição deste modelo.
`abstract ElementType[] getElementTypes()`	Retorna os tipos de elementos que este modelo provê.
`abstract FieldType[] getFieldTypes()`	Retorna os tipos de campos de formulário que este modelo provê.
`abstract boolean isCompatibleWith( ElementType elementType)`	Diz se este modelo pode lidar com elementos to tipo `elementType`.
`abstract Version getVersion()`	Retorna a versão do modelo.
`final ElementController createElementControllerFromXML( org.jdom.Element element)`	Retorna uma instância de `ElementController` criada de acordo com a descrição XML contida no parâmetro `element`.
`abstract boolean isConfigurable()`	Diz se este modelo é configurável ou não.
`abstract ApplicationModelConfiguration newApplicatioinModelConfiguration()`	Caso este modelo seja configurável, este método cria e retorna uma nova instância da classe que representa as configurações deste modelo.

Tabela 5.5: Classe `ApplicationModel`

Classes que representam modelos de aplicações devem seguir o padrão de projeto Singleton [18], permitindo que exista apenas uma instância da classe, já que modelos não possuem estado. O método que retorna a única instância deve ser estático, ter nome `getInstance` e não ter parâmetros. É este o método que o MultiMAD usará para obter o objeto que representa o modelo.

Modelos de aplicações são modelados como *plugins* com o objetivo de não ser necessário alterar o MultiMAD para que um novo modelo de aplicações pudesse ser adicionado à ferramenta. Eles devem ser empacotados em arquivos JAR (*Java ARchive*), forma padronizada de se criar bibliotecas e empacotar programas Java [61]. Informações sobre o conteúdo de um JAR podem ser armazenadas em um arquivo de nome `MANIFEST.MF`.

Para que o MultiMAD reconheça um modelo de aplicações, a classe que o implementa deve ser um Singleton, tal como descrito acima. Além disso, ele deve ser empacotado em um JAR e seu `MANIFEST.MF` deve conter um campo `ApplicationModels` cujo valor é o nome completo da classe que implementa o modelo de aplicações. Caso haja mais de um modelo de aplicações no mesmo JAR, seus nomes devem ser separados por vírgula no campo `ApplicationModels`.

Este JAR deve ser armazenado dentro do subdiretório `lib` do diretório onde a ferramenta é executada. Desta forma, a ferramenta consegue localizar e carregar modelos de aplicações de forma automática quando é executada.

5.2.3 Geradores de aplicações

Os geradores de aplicações do MultiMAD têm a tarefa de gerar a maior parte possível do código-fonte e eventuais arquivos auxiliares que façam parte da implementação da aplicação. São implementados como subclasses da classe abstrata `Generator`. Seus métodos estão listados na tabela 5.6.

Classes que representam geradores devem seguir o padrão de projeto Singleton, permitindo que exista apenas uma instância da classe, já que geradores não possuem estado. O método que retorna a única instância deve ser estático, ter nome `getInstance` e não ter parâmetros. É este o método que o MultiMAD usará para obter o objeto que representa o gerador.

Tal como modelos de aplicações, geradores foram modelados como *plugins*. Eles também devem ser empacotados em arquivos JAR. Para que o MultiMAD reconheça um gerador, ele deve ser empacotado em um JAR e seu `MANIFEST.MF` deve conter um campo `Generators` cujo valor é o nome completo da classe que implementa o gerador. Caso haja mais de um gerador no mesmo JAR, seus nomes devem ser separados por vírgula.

Nome	Descrição
`abstract String getName()`	Retorna o nome do gerador.
`abstract String getDescription()`	Retorna uma descrição do gerador.
`abstract ApplicationModel getReferenceApplicationModel()`	Retorna uma instância de `ApplicationModel` que representa o modelo de aplicações utilizado como referência para este gerador.
`abstract void generate() throws MultimadException`	Gera o código-fonte e outros arquivos que formam a implementação do projeto. Sinaliza erros que porventura aconteçam levantando uma exceção do tipo `MultimadException`.
`boolean isCompatibleWith( ElementType elementType)`	Diz se este gerador suporta o tipo de elementos `elementType`.
`abstract boolean isConfigurable()`	Diz se este gerador é configurável ou não.
`abstract GeneratorConfiguration newGeneratorConfiguration()`	Caso este gerador seja configurável, este método cria e retorna uma nova instância da classe que representa as configurações deste gerador.

Tabela 5.6: Classe `Generator`

Este JAR deve ser armazenado dentro do subdiretório `lib` do diretório onde a ferramenta é executada. Desta forma, a ferramenta consegue localizar e carregar modelos de aplicações de forma automática quando é executada.

5.3 Interface gráfica

A interface gráfica do MultiMAD possui, em sua janela principal, quatro partes principais: menus (descritos na seção 5.3.1), barra de ferramentas (seção 5.3.2), área de trabalho (seção 5.3.3) e painel de elementos (seção 5.3.4). A figura 5.2 mostra a janela principal e todas as suas partes. Cada uma delas será discutida nas seções seguintes. Além da janela principal, quando algum *script* Ant [2] está sendo executado através do MultiMAD, é mostrada uma janela, independente da principal, que mostra a saída da execução em tempo real e mostra um alerta ao usuário quando a execução termina. Um exemplo desta janela é a figura 9.16, na página 104. A seção 5.4 sugere uma forma de uso do MultiMAD.

5.3.1 Menus

Os menus do MultiMAD ficam no topo da janela principal. Cada um deles será descrito a seguir.

Arquivo

O menu Arquivo provê comandos relacionados à abertura, fechamento, leitura e escrita de projetos e fechamento do MultiMAD:

- `Novo projeto`: cria um novo projeto. Caso algum projeto esteja aberto no momento, o procedimento de fechamento de projetos é executado antes. Atalho de teclado: `Control-N`.
- `Fechar`: fecha o projeto. Caso ele tenha alguma modificação ainda não salva, o MultiMAD oferece ao usuário três opções: salvar as modificações e sair, não salvar as modificações e sair, cancelar o fechamento do projeto. Só disponível quando há um projeto aberto. Atalho de teclado: `Control-F4`.

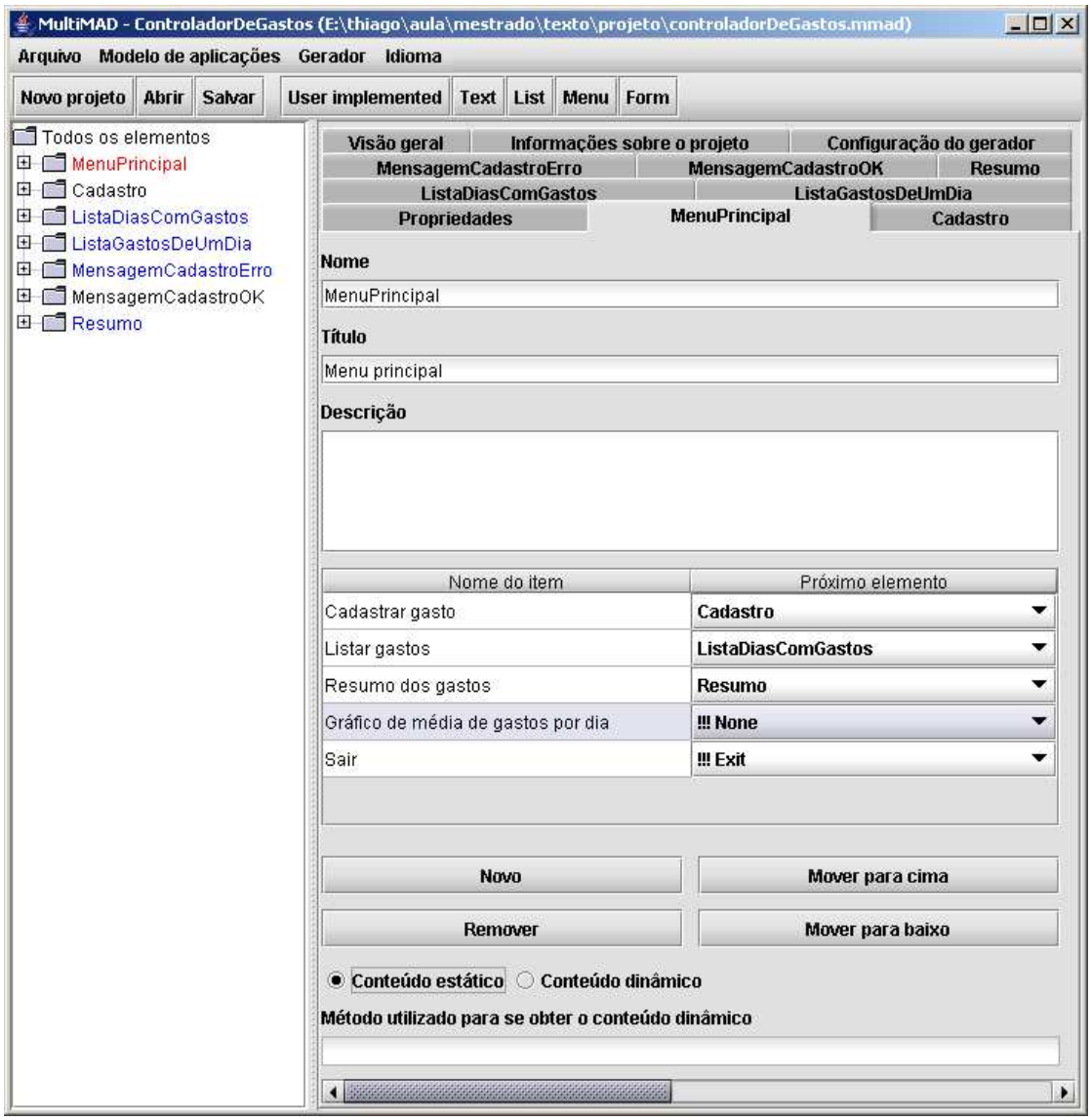

Figura 5.2: Janela principal do MultiMAD

- `Abrir`: carrega um projeto de um arquivo (extensão `mmad`). Caso haja um projeto aberto, é realizado o procedimento de fechamento antes. Atalho de teclado: `Control-O`.

- `Salvar`: escreve o projeto corrente no arquivo correspondente. Caso o projeto não tenha sido salvo antes, é realizado o procedimento de salvar como. Só disponível quando há um projeto aberto. Atalho de teclado: `Control-S`.

- `Salvar como`: escreve o projeto corrente em um arquivo a ser escolhido pelo usuário. Caso o projeto não tenha sido salvo antes, é realizado o procedimento de salvar como. Só disponível quando há um projeto aberto. Atalho de teclado: `Control-A`.

- `Sair`: fecha o MultiMAD. Caso algum projeto esteja aberto, é realizado primeiro o procedimento de fechamento de projeto.

Modelo de aplicações

O menu `Modelo de aplicações` é usado para escolher o modelo de aplicações a ser utilizado pelo projeto. Para cada modelo disponível há uma opção no menu e o modelo utilizado pelo projeto é marcado como selecionado. Para se trocar o modelo, basta escolher uma das outras opções. Ao se trocar o modelo, suas configurações, caso existam, são removidas do projeto.

Gerador

O menu `Gerador` é usado para escolher o gerador a ser utilizado pelo projeto e para requisitar a sua geração. Além de uma opção `Nenhum`, para cada gerador disponível há uma opção no menu e o gerador utilizado pelo projeto é marcado como selecionado. Para trocar o gerador, basta escolher uma das outras opções. Ao se trocar o gerador, suas configurações, caso existam, são removidas do projeto.

A geração do projeto pode ser feita através deste menu ou do atalho de teclado `Control-G`.

Idioma

O menu `Idioma` apresenta uma opção para cada uma dos idiomas nos quais o MultiMAD pode ser utilizado. O idioma corrente estará marcado. Caso o usuário escolha alguma outra opção, a ferramenta automaticamente trocará de idioma.

O MultiMAD, ao ser carregado, verifica qual é o idioma corrente do sistema operacional. Caso ele seja um dos disponíveis, ele será utilizado. Caso contrário, o MultiMAD executará inicialmente utilizando a língua inglesa.

Atualmente, estão disponíveis para a ferramenta os idiomas português brasileiro e inglês americano.

5.3.2 Barra de ferramentas

A barra de ferramentas, localizada logo abaixo dos menus, é dividida em duas partes. A primeira, estática, apresenta botões `Novo projeto`, `Abrir` e `Salvar`, cada um deles executando a mesma função das opções homônimas no menu `Arquivo`. A segunda parte provê um botão para cada um dos tipos de elementos providos pelo modelo de aplicações seguido pelo projeto. Quando um destes botões é clicado, aparecerá uma janela de diálogo onde o usuário entra o nome de um novo elemento daquele tipo e ele é adicionado ao projeto.

5.3.3 Área de trabalho

A área de trabalho ocupa a parte direita da janela principal do MultiMAD, logo abaixo da barra de ferramentas. Ela possui abas nas quais o usuário especifica sua aplicação:

- `Visão geral`: visualização em forma de grafo dos elementos e dos relacionamentos entre eles. As setas que representam os relacionamentos não podem ser editadas. Elementos que não têm um próximo elemento definido não possuem nenhuma seta saindo deles no grafo. Desta forma, é possível que algum elemento não tenha nenhuma seta apontando para ele.

  A posição de cada elemento pode ser alterada através de arrastar e soltar (*drag and drop*) e é armazenada no projeto. A figura 9.13, na página 101, mostra um exemplo da aba de visão geral.

- `Informações sobre o projeto`: edição do nome, da descrição, do diretório onde o projeto será gerado e do elemento inicial da aplicação. Esta aba traz também informações sobre o modelo de aplicações e o gerador utilizados. A figura 9.3, na página 93, mostra um exemplo da aba de informações sobre o projeto.

- Painéis de edição de elemento: abas, uma por elemento do projeto, contendo painéis onde são editados os elementos. Cada tipo de elemento tem seu tipo de painel próprio. Exemplos destes painéis podem ser encontrados na seção 9.1.
- `Configuração do modelo de aplicações`: caso o modelo seguido pelo projeto seja configurável, haverá uma aba para a edição de suas configurações.
- `Configuração do gerador`: caso o gerador utilizado pelo projeto seja configurável, haverá uma aba para a edição de suas configurações.
- `Propriedades`: Caso algum modelo de aplicações ou gerador possua propriedades, haverá uma aba para a sua edição. A figura 9.15, na página 104, mostra um exemplo de propriedades.

5.3.4 Painel de elementos

O painel de elementos, localizado no lado esquerdo do MultiMAD, apresenta na forma de uma árvore todos os elementos do projeto, seus relacionamentos e algumas informações adicionais.

Elementos que não estão prontos para serem gerados são apresentados com a cor vermelha. Os que estão prontos para serem gerados e não precisarão de código implementado pelo usuário são apresentados na cor preta e os que precisarão são apresentados na cor azul.

Cada elemento é apresentado como um galho ou uma folha de uma árvore. Seus galhos mostram para quais outros elementos do projeto o usuário pode ir a partir daquele elemento. Como estes relacionamentos podem ser estáticos ou dinâmicos, apenas os estáticos são mostrados, já que relacionamentos dinâmicos são implementados pelo código-fonte escrito pelo usuário. Um elemento do tipo menu apresenta uma folha para cada um de suas opções, caso ele seja estático, ou nenhuma folha, caso seja dinâmico. Os outros elementos do modelo genérico mostram seu próximo elemento como uma folha. Caso este próximo elemento seja estático, seu nome é mostrado. Caso contrário, o nome `NULL` é mostrado.

A remoção de elementos do projeto é feita clicando-se com o botão direito do *mouse* em sua representação no painel de elementos e escolhendo-se a opção

`Remover`. O MultiMAD verifica se existe alguma referência a ele. Caso exista, a ferramenta pede uma confirmação para o usuário e remove as referências ao elemento.

5.4 Forma de uso

Não há apenas uma forma correta de uso do MultiMAD no desenvolvimento de aplicações. A ferramenta é flexível no sentido de não obrigar o usuário a seguir nenhum determinado processo de desenvolvimento de aplicações. A seguir, são feitas sugestões de forma de uso e de especificação de aplicações.

5.4.1 Forma de uso sugerida

1. Fazer uma especificação textual da aplicação.
2. A partir da especificação anterior, fazer uma especificação das telas da aplicação levando-se em conta os tipos de elementos oferecidos pelo modelo de aplicações a ser seguido pelo projeto.
3. A partir da especificação anterior, especificar a aplicação no MultiMAD. Este passo será descrito em detalhes na seção 5.4.2.
4. Requisitar a geração do código através do menu `Gerador` ou da combinação de teclado `Control-G`.
5. Implementar a lógica da aplicação e a interface gráfica de elementos do tipo `UserImplemented`, caso haja algum.

5.4.2 Forma sugerida de especificação de aplicações

1. Fora da ferramenta, criar um diretório para conter o projeto caso ele não esteja criado ainda.
2. Criar um novo projeto através dos menus ou da barra de ferramentas.
3. Na aba `Informações sobre o projeto`, dar um nome para o projeto.

4. Salvar o projeto no diretório criado para conter o projeto.

5. No menu `Modelo de aplicações`, selecionar o modelo de aplicações a ser seguido pela aplicação.

6. Caso o modelo de aplicações selecionado tenha configurações, preenchê-las na aba `Configurações do modelo de aplicações` da área de trabalho.

7. Criar o elemento inicial seguindo o procedimento de criação de elementos descrito abaixo.

8. Na aba `Informações sobre o projeto`, preencher o campo elemento inicial.

9. Executar o procedimento de criação de elementos para cada elemento da aplicação que ainda não tenha sido criado.

10. Na aba `Visão geral` da área de trabalho, reorganizar o grafo, caso seja necessário, e verificar se os relacionamentos entre elementos e os possíveis fluxos de execução da ferramenta estão condizentes com a especificação da aplicação. Caso não estejam, corrigir os elementos cujas referências estão incorretas.

11. No menu `Gerador`, selecionar um gerador para o projeto.

12. Caso o gerador selecionado tenha configurações, preenchê-las na aba `Configurações do gerador` da área de trabalho.

Procedimento de criação de um elemento cujo nome é *novo*:

1. Na barra de ferramentas, clicar no botão correspondente ao tipo do elemento *novo* para criá-lo.

2. Preencher a caixa de texto com o nome do elemento *novo*. Nomes de elementos devem começar com uma letra e podem ser formados de letras, dígitos, hífen (-) ou sublinhado (_). Além disso, nenhuma letra pode ser acentuada. Não pode haver dois elemento com o mesmo nome.

3. O MultiMAD cria uma aba com o nome dado ao elemento. Especificar o elemento *novo* preenchendo os campos do painel de edição contido nesta aba.

4. Para cada elemento ref referenciado pelo elemento *novo*:

   - Se ref ainda não foi criado, criá-lo seguindo o procedimento de criação de elementos.

5. Setar as referências de *novo* que ainda não foram setadas porque o elemento referido ainda não havia sido criado.

Um exemplo passo a passo da especificação e implementação de uma aplicação utilizando o MultiMAD é apresentado no capítulo 9.

Capítulo 6

Modelos de aplicações

Este capítulo apresenta os modelos de aplicações implementados neste trabalho: modelo genérico (seção 6.1) e o modelo específico para Java 2 Micro Edition (seção 6.2). O conceito de modelo de aplicações está descrito na seção 5.1.3.

6.1 Modelo genérico

O modelo genérico, implementado pela classe `thiagohp.multimad.applicationmodel.GenericApplicationModel`, provê tipos de elementos e de campos de formulários escolhidos com o objetivo de serem suportados pela maior parte possível das plataformas de aplicações para dispositivos móveis. Este modelo não possui configurações nem propriedades.

O modelo genérico de aplicações define três tipos de campos de formulários:

- `String`: uma *string* (seqüência de caracteres).
- `Password`: o mesmo que o tipo `String`, porém os caracteres entrados pelo usuário da aplicação não são mostrados. Usado para a entrada de senhas.
- `Boolean`: um valor booleano (verdadeiro ou falso).

Os tipos de elementos providos pelo modelo genérico estão descritos nas seções seguintes. Todo elemento, independente de seu tipo, tem um nome, utilizado

para identificação dentro do código gerado; e um título, mostrado ao usuário da aplicação gerada. Todos, exceto `Menu`, também têm uma referência opcional para o seu próximo elemento.

6.1.1 `Text`

Elementos do tipo `Text` mostram um texto não editável para o usuário final. O texto pode ser tanto estático, sendo definido em tempo de especificação da aplicação; ou dinâmico, definido em tempo de execução através de uma chamada a uma função que deverá ser implementada pelo usuário da ferramenta.

Elementos deste tipo não estão prontos para serem gerados caso eles tenham um texto vazio, no caso de conteúdo estático; ou sem nome do método que retorne o conteúdo, no caso de conteúdo dinâmico.

6.1.2 `Menu`

Elementos do tipo `Menu` mostram uma lista de opções para o usuário final. Cada uma destas opções leva a um outro elemento da aplicação. O conteúdo do menu pode ser tanto estático, sendo definido em tempo de especificação da aplicação; ou dinâmico, definido em tempo de execução através de uma chamada a uma função que deverá ser implementada pelo usuário da ferramenta.

Cada ítem de um menu contém um rótulo (texto mostrado ao usuário final) e o nome do elemento que será mostrado ao usuário da aplicação caso este ítem seja escolhido. Um ítem é válido se ele tem uma referência definida (diferente de `NULL`) e se seu rótulo tem pelo menos um caractere.

Caso um menu seja estático, ele está pronto para ser gerado se tiver pelo menos um item e se todos os itens forem válidos. Caso contrário, ele está pronto se houver sido informado o nome do método que retornará o seu conteúdo.

6.1.3 `List`

Elementos do tipo `List` mostram uma lista de opções para o usuário final. Cada uma destas opções é composta por um rótulo, mostrado ao usuário, e um

valor associado a ele. Um ítem é válido se seu rótulo e seu valor tiverem pelo menos um caractere cada.

O conteúdo da lista pode ser tanto estático, sendo definido em tempo de especificação da aplicação; ou dinâmico, definido em tempo de execução através de uma chamada a uma função que deverá ser implementada pelo usuário da ferramenta.

Caso uma lista seja estática, ela está pronta para ser gerada se ela tiver pelo menos um ítem e se todos os itens forem válidos. Caso contrário, ela está pronta se houver sido informado o nome do método que retornará o seu conteúdo.

6.1.4 `Form`

Elementos do tipo `Form` mostram um formulário para o usuário final. Um formulário é composto por uma lista de campos. Cada campo é formado por:

- um nome. Ele é utilizado para identificação do campo tanto no código gerado quanto no código implementado pelo usuário. Por isso, nomes de campos válidos devem começar com uma letra e conter apenas letras sem acento, dígitos, hífen (-) e sublinhado (_).
- um rótulo. Ele é mostrado ao usuário da aplicação ao lado ou acima do elemento de interface gráfica utilizado para a entrada do valor do campo.
- um tipo. Os tipos de campos que um formulário pode ter são definidos pelo modelo de aplicações sendo usado.
- um valor inicial (opcional). A interface gráfica não permite que o valor inicial não seja válido para o tipo do campo.

Um campo é válido caso o seu nome seja válido e seu rótulo tiver pelo menos um caractere.

Elementos deste tipo estão prontos para serem gerados se tiverem pelo menos um campo e se todos os campos forem válidos.

6.1.5 `UserImplemented`

Elementos do tipo `UserImplemented` mostram um elemento que, ao contrário dos outros tipos, não é implementado por código gerado pelo MultiMAD e sim pelo usuário da ferramenta. Isto permite que ele tenha a liberdade de implementar seus elementos da aplicação da forma que melhor lhe convier, não o prendendo apenas aos providos pelo modelo de aplicação utilizado.

Estes campos estão sempre prontos para geração, já que o painel de edição de elementos não permite ao usuário fazer nada sem que o nome e o título do elemento editado estejam preenchidos e válidos.

6.2 Modelo de aplicações para a plataforma Java 2 Micro Edition (MIDP 1.0)

Este modelo é uma extensão do modelo genérico e provê tipos de campos específicos da plataforma Java 2 Micro Edition [57] no seu perfil MIDP 1.0 [59]. A classe `thiagohp.multimad.applicationmodel.midp.MIDP1ApplicationModel` implementa este modelo.

Este modelo provê tipos adicionais de campos:

- `Integer`: um campo de texto que só aceita números inteiros, tanto positivos quanto negativos[1].
- `Gauge`: um campo que permite a entrada de valores inteiros positivos de pequena magnitude (recomendável para valores menores que 20) de uma forma gráfica. Se não for dado um valor inicial a um campo deste tipo, seu valor inicial será 0. Um exemplo deste tipo de campo é mostrado na figura 6.1.
- `Date`: campo que permite a entrada de uma data (dia, mês e ano) de forma gráfica. Num formulário, campos deste tipo apenas mostram o valor atual.

[1] Parece haver um *bug* na implementação deste tipo de campo no Java 2 Micro Edition Wireless Toolkit versão 2.2 que não permite a entrada de números negativos, o que não acontece na versão 1.0.4

Quando o campo é selecionado, passa-se para uma tela na qual o valor pode ser alterado. No MultiMAD, o valor inicial de campos deste tipo deve ser entrado no formato `YYYY/MM/DD`, sendo `YYYY` o ano expresso em quatro dígitos, `MM` o mês e `DD` o dia do mês. Por exemplo, o dia 29 de abril de 2005 deve ser entrado como 2005/04/29. Caso nenhum valor inicial seja dado, ele será a data corrente no momento em que o formulário é mostrado ao usuário. A figura 6.2 mostra como estes campos deste tipo aparecem para o usuário no dispositivo móvel e a figura 6.3 mostra como a edição é feita no dispositivo móvel.

- `Time`: campo que permite a entrada de uma hora do dia (hora e minutos) de forma gráfica. Num formulário, campos deste tipo apenas mostram o valor atual. Quando o campo é selecionado, passa-se para uma tela na qual o valor pode ser alterado. No MultiMAD, o valor inicial de campos desse tipo deve ser entrado no formato `hh:mm`, sendo `hh` a hora e `mm` os minutos. Por exemplo, a hora 14 horas e 15 minutos deve ser entrada como 14:15. Caso nenhum valor inicial seja dado, ele será a hora e minutos no momento em que o formulário é mostrado ao usuário. A figura 6.2 mostra como estes campos deste tipo aparecem para o usuário no dispositivo móvel e a figura 6.4 mostra como a edição é feita no dispositivo móvel.

- `DateTime`: campo que é uma combinação dos tipos `Date` e `Time`. Permite a entrada de uma data e hora (dia, mês e ano, hora e minutos) de forma gráfica. Num formulário, campos deste tipo apenas mostram o valor atual. Quando o campo é selecionado, passa-se para uma tela na qual o valor pode ser alterado. No MultiMAD, o valor inicial de campos deste tipo deve ser entrado no formato `YYYY/MM/DD hh:mm`, sendo `YYYY` o ano expresso em quatro dígitos, `MM` o mês, `DD` o dia do mês, `hh` a hora e `mm` os minutos. Por exemplo, o dia 29 de abril de 2005, às 14 horas e 15 minutos, deve ser entrado como 2005/04/29 14:15. Caso nenhum valor inicial seja dado, ele será a data e hora correntes no momento em que o formulário é mostrado ao usuário. A figura 6.2 mostra como estes campos deste tipo aparecem para o usuário no dispositivo móvel. A edição de campos deste tipo no dispositivo móvel é feita de forma separada: a data tal como se fosse um campo `Date` e a hora

e os minutos tal como se fosse um campo `Time`.

Figura 6.1: Exemplo de campo do tipo `Gauge`

Figura 6.2: Exemplos de campos dos tipos `Date`, `Time` e `DateTime`

Figura 6.3: Exemplo de edição de campo do tipo `Date`

Figura 6.4: Exemplo de edição de campo do tipo `Time`

Capítulo 7

Arquitetura MobileVC

A arquitetura MobileVC é uma adaptação do conhecido padrão de projeto (*design pattern*) MVC (*Model-View-Controller*, Modelo-Visão-Controlador), que define a separação entre o modelo (forma de armazenamento de dados), a visão (interface de usuário) e o controlador (lógica da aplicação) [18], para aplicações para dispositivos móveis. O MVC foi concebido para aplicações de grande porte. O objetivo do MobileVC é definir uma estrutura comum para implementações em uma mesma linguagem de programação de uma determinada aplicação para diferentes plataformas de aplicações para dispositivos móveis. Como diferentes plataformas têm formas muito diferentes de armazenamento de dados, o MobileVC, por enquanto, não tenta lidar com este aspecto.

Esta arquitetura promove a separação total entre a implementação da interface gráfica e da lógica da aplicação, prática consagrada pela Engenharia de Software. Assim, o MobileVC permite usar componentes prontos para a interface gráfica e facilita a realização de testes sobre a implementação da lógica da aplicação.

7.1 Visão geral

A arquitetura MobileVC define que aplicações são estruturadas a partir destas interfaces:

- `Controller`: implementada pela classe que implementa a lógica da aplicação.

Descrita na seção 7.2.

- `ViewRenderer`: implementada pela classe que implementa a interface gráfica de elementos que não são do tipo `UserImplemented`. Descrita na seção 7.3.
- `UserImplementedViewRenderer`: implementada pela classe que implementa a interface gráfica de elementos do tipo `UserImplemented`. Descrita na seção 7.4. Só é necessária a implementação de um `UserImplementedViewRenderer` caso a aplicação tenha um ou mais elementos do tipo `UserImplemented`.
- `ApplicationInfo`: encapsula informações sobre a aplicação. Usada apenas por implementações de `ViewRenderer`.

A figura 7.1 mostra o diagrama de seqüência que modela a relação entre estas interfaces.

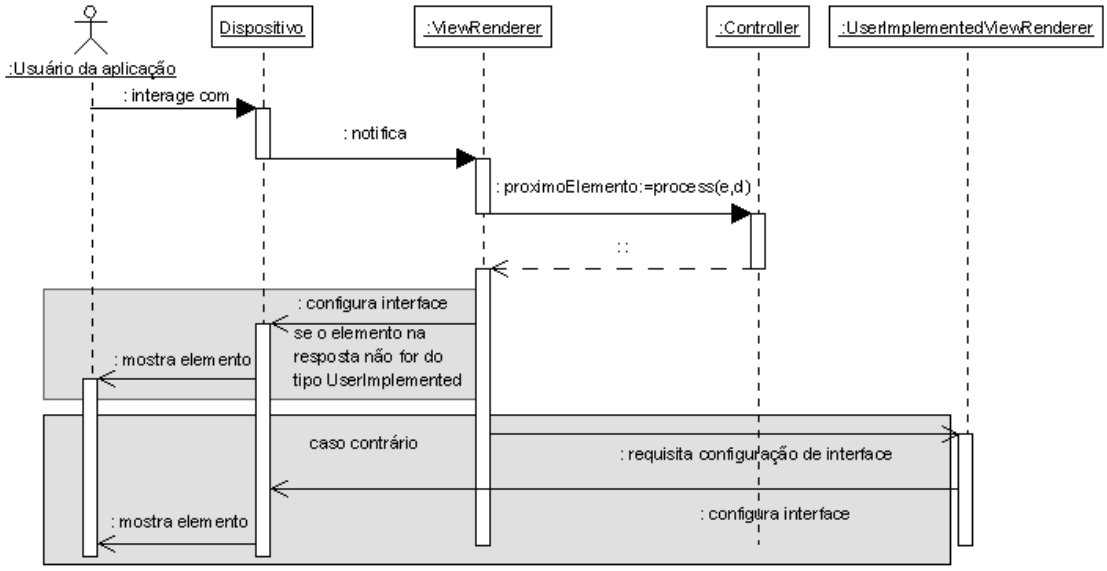

Figura 7.1: Diagrama de seqüência do núcleo do MobileVC

Uma característica importante desta arquitetura é que as implementações de `Controller` e de `UserImplementedViewRenderer` são, quase sempre, passivas. Em outras palavras, elas tipicamente apenas respondem a requisições feitas pelo `ViewRenderer` à medida em que o usuário da aplicação gerada interage com ela.

Outra característica importante é que cada instância de `Controller`, `ViewRenderer` e `UserImplementedViewRenderer` é usada para uma única execução da aplicação por um determinado usuário.

Geradores de aplicações que seguem esta arquitetura tipicamente provêem ou geram uma implementação de `ViewRenderer`, de `ApplicationInfo` e de código que faz a integração destes com a implementação de `Controller` e de `UserImplementedViewRenderer` (caso seja necessário) escritas pelo usuário do MultiMAD.

Todas as classes e interfaces citadas neste capítulo estão implementadas em Java [7], porém podem ser adaptadas para qualquer linguagem de programação orientada por objetos. As classes estão localizadas no pacote `thiagohp.multimad.mobilevc.core`, a não ser que haja indicação em contrário.

7.2 Interface `Controller`

A classe que implementa a lógica da aplicação deve implementar a interface `Controller`.

Os métodos de `Controller` estão descritos na tabela 7.1.

7.3 Interface `ViewRenderer`

A interface `ViewRenderer` é implementada por classes que encapsulam a implementação da interface gráfica de elementos que não sejam do tipo `UserImplemented`. Seus métodos estão descritos na tabela 7.2.

Tipicamente, geradores de aplicações que seguem a arquitetura MobileVC fornecerão ou gerarão uma implemenetação pronta de `ViewRenderer`, permitindo ao usuário do MultiMAD ter que escrever código de interface gráfica apenas de elementos do tipo `UserImplemented`.

Nome	Descrição
`void setup()`	Método chamado quando o usuário entra na aplicação.
`String process( ElementInfo elementInfo, Object data)`	Método que processa uma ação do usuário e retorna o nome do elemento que deve ser mostrado em seguida. Instâncias da interface `ElementInfo` descrevem elementos na aplicação gerada.
`Object getParameter( String elementName)`	Retorna o parâmetro a ser usado quando o elemento de nome `elementName` for mostrado para o usuário.
`String getMainElementName()`	Retorna o nome do elemento a ser mostrado ao usuário quando ele inicia sua execução da aplicação.
`String getSpecialElementNextElementName( String specialElementName)`	Retorna o nome do elemento a ser mostrado ao usuário quando sua ação foi escolher um elemento especial.
`void dispose()`	Método chamado quando o usuário sai da aplicação. Sua implementação deve liberar todos os recursos que porventura tenham sido alocados.

Tabela 7.1: Interface `Controller`

7.4 Interface `UserImplementedViewRenderer`

A classe que implementa a interface gráfica de elementos do tipo `UserImplemented` deve implementar a interface `UserImplementedViewRenderer`. Seus métodos estão descritos na tabela 7.3.

Note que esta interface não define nenhum método que faça a implementação da interface gráfica. Isto acontece porque os parâmetros necessários variam muito de plataforma para plataforma. Cada gerador provê uma interface que estende

Nome do método	Descrição
`void setup( ApplicationInfo applicationInfo, Controller controller, UserImplementedViewRenderer uivr)`	Método chamado logo antes de este `ViewRenderer` ser utilizado pela primeira vez. `applicationInfo` é o objeto que descreve a aplicação, `Controller` é o objeto que implementa a lógica da aplicação e `uivr` é o objeto que implementa os elementos do tipo `UserImplemented`.
`void dispose()`	Método chamado quando o usuário sai da aplicação. Deve desalocar todos os recursos compartilhados que foram alocados por este objeto.

Tabela 7.2: Interface `ViewRenderer`

Nome	Descrição
`void setup()`	Método que é chamado quando o usuário entra na aplicação.
`void dispose()`	Método chamado quando o usuário sai da aplicação. Sua implementação deve liberar todos os recursos que porventura tenham sido alocados.

Tabela 7.3: Interface `UserImplementedViewRenderer`

`UserImplementedViewRenderer` e define um método que faz a implementação da interface gráfica de elementos do tipo `UserImplemented`.

Capítulo 8

Geradores de aplicações

Foram implementados dois geradores de aplicações para o MultiMAD neste trabalho: um para a plataforma WAP [37, 38] e outro para Java 2 Micro Edition™(J2ME) [57]. Ambos geram aplicações que seguem a arquitetura MobileVC, descrita no capítulo 7.

A seção 8.1 trata das várias características em comum entre os dois geradores. A seção 8.2 apresenta o gerador para a plataforma WAP e a seção 8.3 discorre sobre o gerador para a plataforma J2ME.

8.1 Características comuns

Os dois geradores são subclasses de uma mesma classe abstrata, `thiagohp.multimad.mobilevc.MobileVCGenerator`. Ela implementa todas as características comuns de geradores de aplicações que seguem a arquitetura MobileVC.

A classe `MobileVCGenerator` gera as seguintes classes e interfaces:

- `GeneratedApplicationInfo`: implementação da interface `ApplicationInfo`.
- `Constants`: interface que contém uma constante do tipo `String` para cada um dos elementos da aplicação e para cada um dos elementos especiais do

modelo de aplicações seguido pela aplicação.

- `GeneratedController`: classe que implementa parcialmente a interface `Controller`. Descrita na seção 8.1.2.

Estas classes e interfaces são geradas de forma que o usuário não precise e nem deva alterá-las. Caso o projeto seja gerado novamente, elas são sobrescritas e todas as alterações feitas pelo usuário são perdidas. Cada uma delas é gerada com avisos a respeito.

Para evitar confusão entre código gerado e código implementado pelo usuário, todo código gerado é colocado no subpacote `generated` do pacote onde o usuário colocas suas classes. Exemplo: se o pacote onde o usuário coloca suas classes (configuração `packageName`) é `multimad.exemplo`, o código gerado é colocado em `multimad.exemplo.generated`, fazendo com que o código gerado e o escrito pelo usuário fiquem em diretórios diferentes.

A classe `MobileVCGenerator` tem como uma única dependência externa o motor de modelos (*template engine*) Velocity [3].

Outra característica comum entre os geradores de aplicações para a plataforma WAP 1.1 e 2.0 e Java 2 Micro Edition é que ambos geram *scripts* Ant [2] e os executam após a geração de código. Estes *scripts* automatizam a compilação, empacotamento e, no caso do gerador de aplicações para J2ME, a execução da aplicação em um emulador.

8.1.1 Configurações

Todo gerador é uma subclasse de `MobileVCGenerator` e possui o conjunto mínimo de configurações apresentado na lista a seguir. Todas elas são obrigatórias.

- `packageName`: Nome do pacote Java que conterá as implementações de `Controller` e `UserImplementedViewRenderer` feitas pelo usuário do MultiMAD. Não tem valor inicial.

- `controllerClassName`: Nome da classe, escrita pelo usuário, que implementará `Controller`. Valor inicial: `ControllerImpl`.

- `userImplementedViewRendererClassName`: Nome da classe, escrita pelo usuário que implementará `UserImplementedViewRenderer`. Valor inicial: `UserImplementedViewRendererImpl`.
- `viewRendererClassName`: Nome completo da classe que implementará `ViewRenderer`. Valor inicial:
  - Gerador de aplicações para WAP: `thiagohp.multimad.mobilevc.wml.core.GenericWMLViewRenderer`
  - Gerador de aplicações para J2ME: `thiagohp.multimad.mobilevc.midp.core.GenericMIDPViewRenderer`
- `okString`: String a ser utilizada como rótulo em botões do tipo `OK`. Valor inicial: `OK`.
- `submitString`: String a ser utilizada como rótulo em botões do tipo `Submit` em formulários. Valor inicial: `Submit`.
- `trueString`: String a ser utilizada para mostrar um valor booleano verdadeiro em formulários. Valor inicial: `true`.
- `falseString`: String a ser utilizada para mostrar um valor booleano falso em formulários. Valor inicial: `false`.

8.1.2 `GeneratedController`

A classe abstrata `GeneratedController`, subclasse de `Controller`, é gerada automaticamente e implementa o relacionamento entre os elementos da aplicação de acordo com sua especificação no MultiMAD. Esta classe define métodos abstratos para lidar com os aspectos dinâmicos da aplicação.

A implementação da lógica da implementação, feita pelo usuário da ferramenta, deve ser uma subclasse concreta de `GeneratedController`.

Esta classe implementa os métodos `process`, `getParameter` e `getMainElementName` definidos por `Controller`. Além destes, ela declara os métodos abstratos descritos na lista a seguir. Eles são a integração entre

o código gerado pelo MultiMAD e o código implementado pelo usuário da ferramenta.

- `void doProcess(ElementInfo elementInfo, Object data)`: Este método será chamado no início da execução de `process` e é o método que o usuário do MultiMAD deve implementar para processar ações do usuário.

- Para cada elemento cujo conteúdo é dinâmico (exceto aqueles do tipo `UserImplemented`), é declarado um método `TipoConteudoXXX nomeDoMetodo()`, onde `TipoConteudoXXX` é o tipo que representa o conteúdo do elemento e `nomeDoMetodo` é o nome, escolhido pelo usuário no MultiMAD, do método que retorna o conteúdo do elemento. Estes métodos são usados pela classe `GeneratedController` para obter o conteúdo de elementos dinâmicos. As classes utilizadas para representar o conteúdo dos tipos de elementos do modelo genérico de aplicações estão descritas na seção 8.1.4.

- Para cada elemento do tipo `UserImplemented`, é declarado um método `Object getXXXParameter()`, onde `XXX` é o nome do elemento. Este método é usado pela classe `GeneratedController` para obter o parâmetro a ser usado em elementos deste tipo.

- Para cada elemento que não tenha seu próximo elemento definido, é declarado um método `String getXXXNextElementName()`, onde `XXX` é o nome do elemento. Este método é usado pelo `GeneratedController` para obter o próximo elemento de elementos que não têm um próximo elemento definido.

A classe `GeneratedController` implementa os aspectos estáticos da aplicação (relacionamentos entre elementos e conteúdo de elementos estáticos). Os métodos abstratos são uma espécie de lacunas que serão preenchidas pelo usuário da ferramenta, implementando a lógica da implementação (método `doProcess`), os parâmetros a serem passados a elementos dinâmicos e o elemento a ser mostrado ao usuário depois de elementos que não têm um próximo elemento definido.

8.1.3 Geração de protótipos funcionais da classes implementadas pelo usuário

A classe `MobileVCGenerator` também gera um protótipo funcional da classe na qual o usuário implementa a lógica da aplicação. Nela, todos os métodos abstratos declarados em `GeneratedController` têm uma implementação que nada faz.

Os métodos que retornam o conteúdo de elementos dinâmicos retornam um conteúdo temporário e que avisa ao usuário da aplicação que aquele elemento sendo mostrado não tem uma implementação definitiva. Os métodos que retornam o próximo elemento de elementos que não têm um próximo elemento definido retornam o valor retornado pelo método `getMainElementName`.

A implementação do método `getSpecialElementNextElementName` retorna o nome do elemento especial `EXIT` se o elemento especial for `EXIT` e `getMainElementName` caso contrário.

Os dois geradores também geram uma protótipo da implementação da interface `UserImplementedViewRenderer` caso o projeto tenha algum elemento do tipo `UserImplemented`.

Quando é requisitada a geração de código, o gerador nunca sobrescreve implementações de `Controller` e `UserImplementedViewRenderer` com protótipos. Quando elas já existem, os protótipos são gerados para um arquivo cujo nome é o do arquivo original mais o sufixo `.generated`. Por exemplo, se o arquivo que contém a classe na qual o usuário implementa `GeneratedController` se chama `ControllerImpl.java` e a que implementa `UserImplementedViewRenderer` se chama `UserImplementedViewRendererImpl.java`, os protótipos são gerados para os arquivos `ControllerImpl.java.generated` e `UserImplementedViewRendererImpl.java.generated`, respectivamente.

A geração destes protótipos têm pelo menos duas utilidades. A primeira é possibilitar a geração de um protótipo funcional da aplicação sem que o usuário precise sair do MultiMAD e sem escrever uma única linha de código. A segunda utilidade é facilitar o trabalho do usuário da ferramenta, já que todos os métodos que ele deve implementar já estão declarados, bastando trocar a implementação temporária pela definitiva.

8.1.4 Classes que representam o conteúdo de elementos dinâmicos

A lista seguinte mostra as classes utilizadas para representar o conteúdo dinâmico de tipos de elementos do modelo genérico de aplicações.

- `Text`: `java.lang.String`
- `Menu`: vetor de instâncias da classe `thiagohp.multimad.mobilevc.core.elementinfo.MenuItem` Seus métodos estão descritos na tabela 8.1.
- `List`: vetor de instâncias da classe `thiagohp.multimad.mobilevc.core.elementinfo.MenuItem` Seus métodos estão descritos na tabela 8.2.

Nome do método	**Descrição**
`void setLabel(String label)`	Altera o rótulo deste item para `label`.
`String getLabel(String label)`	Retorna o rótulo deste item.
`void setNextElementName( String nextElementName)`	Altera o próximo elemento deste item para aquele cujo nome é `nextElementName`.
`String getNextElementName()`	Retorna o nome do próximo elemento deste item.

Tabela 8.1: Classe `MenuItem`

Nome do método	**Descrição**
`void setLabel(String label)`	Altera o rótulo deste item para `label`.
`String getLabel(String label)`	Retorna o rótulo deste item.
`void setValue(String value)`	Altera o valor deste item para `value`.
`String getValue()`	Retorna o valor deste item.

Tabela 8.2: Classe `ListItem`

8.1.5 Implementação de `ViewRenderer`

As implementações de `ViewRenderer` providas pelos dois geradores usam os valores descritos na tabela 8.3 para descreverem a interação do usuário da aplicação com os seus elementos (segundo parâmetro do método `process` de `Controller`).

Tipo de elemento	**Valor usado para descrever a interação**
`Text`	O valor `null`, já que a única interação possível entre elementos deste tipo é ir para o próximo elemento.
`Menu`	Uma `String` contendo o nome do elemento selecionado.
`List`	Uma `String` contendo o valor selecionado.
`Form`	Um `Hashtable` contendo pares (nome do campo, valor escolhido pelo usuário). Tanto o nome do campo quanto o valor são `String`s.

Tabela 8.3: Valores usados para descrever a interação do usuário da aplicação com os seus elementos

8.1.6 Diretórios

Antes da geração do código-fonte, os geradores criam um diretório `src/java` (no caso de sistemas operacionais que usam a barra invertida como separador de diretórios, `src\java`) dentro do diretório de saída do projeto e lá colocam o código gerado. Neste mesmo diretório o usuário do MultiMAD deve colocar seus fontes.

Arquivos de recursos (`resources`) que porventura sejam usados na implementação da aplicação devem ser colocados no diretório `src/resources`. Este diretório é tipicamente criado por *scripts* Ant [2] criados pelos geradores.

8.2 Gerador de aplicações para a plataforma WAP 1.1 e 2.0

O gerador de aplicações para a plataforma WAP [37, 38]. É baseado no modelo genérico e não suporta nenhum outro modelo. As aplicações por ele geradas podem ser utilizadas em navegadores WAP 1.1 e 2.0 de forma completamente transparente, bastando que o usuário do MultiMAD implemente os elementos do tipo `UserImplemented`, caso existam, de forma que retornem documentos em um formato suportado pelo do navegador do usuário da aplicação. Uma forma é aproveitar o fato de que a maioria absoluta dos navegadores WAP 2.0 também suporta WAP 1.1.

A escolha do formato do documento retornado para o navegador do usuário final para elementos que não sejam do tipo `UserImplemented` é determinado pelo campo `Accept` da requisição HTTP [16]. Se ele disser que o navegador do cliente aceita arquivos XHTML, são retornados documentos na linguagem WML 2.0 [39], que é baseada no XHTML Basic [69] e é a utilizada por WAP 2.0. Caso contrário, são retornados documentos na linguagem WML 1.1, utilizada por WAP 1.1.

8.2.1 Configurações

Além das comuns a todos os geradores que herdam de `MobileVCGenerator`, este gerador tem uma configuração, `servletContext`, sem valor inicial, que define em qual contexto de *servlet* a aplicação gerada será encontrada [24]. Exemplo: se o servidor onde a aplicação será hospedada tem URL `http://www.exemplo.com.br` e o contexto do *servlet* é `teste`, a aplicação Web será acessável em `http://www.exemplo.com.br/teste/`. Na verdade, a aplicação implementada pelo usuário do MultiMAD é acessável em `http://www.exemplo.com.br/teste/application`. Isto foi feito para que fosse possível adicionar documentos estáticos que não façam parte da aplicação. Isto será explicado na seção 8.2.4.

8.2.2 Implementação de `ViewRenderer`

A implementação de `ViewRenderer` provida por este gerador é baseada na classe `thiagohp.multimad.mobilevc.wml.core.GenericWMLViewRenderer`. Ela provê métodos que podem ser muito úteis na implementação de `UserImplementedViewRenderer`. Eles estão listados na tabela 8.4.

Também é provido um *servlet* que recebe as requisições HTTP feitas pelo navegador do usuário da aplicação e as transforma em chamadas para o método `process` da implementação de `Controller` escrita pelo usuário do MultiMAD. Além disso, ele cria uma nova instância das implementações de `Controller` e de `UserImplementedViewRenderer` para cada nova sessão de usuário. Este *servlet* é configurado pelo arquivo `web.xml`, gerado automaticamente por este gerador. Ele é sobrescrito quando o projeto é gerado novamente e não deve ser alterado pelo usuário.

Quando o usuário final requisita a saída da aplicação, seu navegador é direcionado para a URL `http://[dominio]/[servletContext]`. Por exemplo: se a aplicação é hospedada num servidor cuja URL é `http://www.exemplo.com.br` e a configuração `servletContext` tem valor `teste`, o navegador do usuário final é direcionado para `http://www.exemplo.com.br/teste` quando ele requisita a saída da aplicação.

Esta implementação de `ViewRenderer` e o *servlet* estão empacotados no arquivo `mobilevc-wml-runtime-1.0.jar` e têm uma única dependência externa, Velocity, utilizado para se montar dinamicamente os documentos que são retornados ao navegador do usuário da aplicação. Ambos estão incluídos na instalação do MultiMAD e são copiados para o diretório `web/WEB-INF/lib` pelo *script* Ant.

8.2.3 Implementação de `UserImplementedViewRenderer`

A implementação de `UserImplementedViewRenderer` escrita pelo usuário do MultiMAD deve implementar a interface `thiagohp.multimad.mobilevc.wml.core.WMLUserImplementedViewRenderer`. Seus métodos estão descritos na tabela 8.5.

Um aspecto muito importante da implementação do

`WMLUserImplementedViewRenderer` é que os documentos por ele gerados devem obrigatoriamente utilizar os métodos `createURL` e `createListItemURL` de `GenericWMLViewRenderer` para criar URLs que não sejam de endereços fora da aplicação. Além destes métodos facilitarem muito a implementação, as URLs geradas contêm informações necessárias para o correto funcionamento da aplicação.

8.2.4 Diretórios

A estrutura de diretórios criada pelo *script* Ant está descrita na tabela 8.6. Esta estrutura é a definida pela especificação de *servlets* para ser utilizada em aplicações Web implementadas em Java [24].

8.2.5 *Script* Ant

Este gerador cria um *script* Ant, de nome `build.xml`, que a cria a estrutura de diretórios definida pela especificação de *servlets*, compila e empacota a aplicação. Caso o projeto seja gerado de novo e já exista um `build.xml`, este é renomeado para `build.xml.old` antes da geração do novo. Isto acontece com o objetivo de o *script* poder ser alterado e gerado novamente sem que se perca as alterações.

O *script* cria a estrutura de diretórios descrita na seção anterior e copia alguns arquivos necessários para a compilação e execução da aplicação. A tarefa padrão do *script* (comando `ant` na linha de comando) compila e empacota a aplicação em um arquivo cujo nome é `[servletContext].war`. Exemplo: se o valor da configuração `servletContext` é `teste`, o nome do arquivo será `teste.war`. Este arquivo pode ser instalado (*deployed*) diretamente em um contâiner de *servlets* tal como o Tomcat [4].

Para que o *script* possa compilar o código implementado pelo usuário, é imprescindível que a Servlet API esteja no diretório `dependencies` da instalação do MultiMAD.

Método	**Descrição**
`String escape(String string)`	Faz o escape de caracteres que poderiam causar problemas em um documento WML ou XHTML.

`String createListItemURL(`
`ElementInfo elementInfo,`
`String value,`
`HttpServletResponse response)`
	Retorna uma URL para ser usada como uma ligação (*link*) que, caso o usuário a utilize, ele estará selecionando um determinado valor de uma lista.

`String createURL(`
`ElementInfo elementInfo,`
`String nextElementName,`
`HttpServletResponse response)`
	Retorna uma URL para ser usada como uma ligação (*link*) que leva a um outro elemento da aplicação. Utilizado para gerar os itens de elementos do tipo `Menu`.

`String createURL(`
`ElementInfo elementInfo,`
`HttpServletResponse response)`
	Retorna uma URL para ser usada como uma ligação (*link*) que leva ao próximo elemento do elemento corrente. Utilizado para montar a URL em botões `OK` de elementos do tipo `List` e `Text`.

`String getOkString()`	Retorna a `String` a ser utilizada como rótulo em botões do tipo `OK`.
`String getSubmitString()`	Retorna a `String` a ser utilizada como rótulo em botões do tipo `Submit` em formulários.
`String getTrueString()`	Retorna a `String` a ser utilizada para mostrar um valor booleano verdadeiro em formulários.
`String getFalseString()`	Retorna a `String` a ser utilizada para mostrar um valor booleano falso em formulários.

Tabela 8.4: Alguns métodos da classe `GenericWMLViewRenderer`

Método	**Descrição**
`void render( UserImplementedElementInfo userImplementedElementInfo, Object parameter, GenericWMLViewRenderer viewRenderer, WMLViewRendererHelper helper, HttpServletRequest request, HttpServletResponse response)`	Método que o usuário do MultiMAD deve implementar para montar o documento que seja a visualização do elemento `userImplementElementInfo`. `parameter` é o parâmetro do elemento. `viewRenderer` é a instância de `GenericWMLViewRenderer` que possui este objeto. `request` é a instância de `HttpServletRequest` que representa a requisição HTTP feita pelo navegador do usuário da aplicação. `response` é a instância de `HttpServletResponse` que representa a resposta à requisição HTTP feita pelo navegador do usuário da aplicação.
`Object getData( UserImplementedElementInfo userImplementedElementInfo, HttpServletRequest request)`	Método que, após o usuário da aplicação interagir com um elemento do tipo `UserImplemented`, retorna um objeto descrevendo esta interação. Ele será usado como o segundo parâmetro do método `process` de `Controller`. O primeiro será `userImplementedElementInfo`.

Tabela 8.5: Interface `WMLUserImplementedViewRenderer`

Diretório	**Descrição**
`web`	Diretório onde são colocados documentos estáticos que não estão contidos na aplicação.
`web/WEB-INF`	Diretório onde é colocado o arquivo de configuração `web.xml`.
`web/WEB-INF/lib`	Diretório onde devem ser colocados arquivos JAR que contém bibliotecas necessárias para a compilação e execução da aplicação. Aqueles nos quais o `ViewRenderer` depende são copiadas pela tarefa `copyDependencies` do *script* Ant.
`web/WEB-INF/classes`	Diretório para onde é compilado o código-fonte escrito pelo usuário e para onde os arquivos contidos no diretório `src/resources` são copiados pelo *script* Ant.

Tabela 8.6: Estrutura de diretórios de aplicações WAP geradas pelo MultiMAD

8.3 Gerador de aplicações para a plataforma Java 2 Micro Edition™(J2ME)

O gerador de aplicações para a plataforma Java 2 Micro Edition™(J2ME) [57], além de prover uma implementação pronta de `ViewRenderer`, gera automaticamente um *script* Ant que compila, empacota e executa a aplicação no J2ME Wireless Toolkit [58], inclusive criando e preenchendo todos os atributos necessários do arquivo JAD (*Java Descriptor*, Descritor Java) e do `MANIFEST.MF`. O JAD é utilizado por dispositivos e emuladores para obter informações necessárias para a execução da aplicação. Este gerador é baseado no modelo de aplicações específico para J2ME e o código gerado é direcionado para o Mobile Information Device Profile (MIDP) [59].

Uma característica interessante deste gerador é que, após a geração da aplicação, ele executa o *script* Ant gerado. Este, por sua vez, compila, empacota e executa a aplicação no J2ME Toolkit sem que o usuário do MultiMAD saia da ferramenta e sem escrever uma única linha de código.

Todas as classes citadas nesta seção pertencem ao pacote `thiagohp.multimad.mobilevc.midp.core` salvo indicação em contrário.

8.3.1 Propriedades

Este gerador possui uma única propriedade, `j2me.wireless.toolkit.dir`, que deve ser preenchida com o diretório onde o J2ME Wireless Toolkit está instalado. Caso esta propriedade não esteja preenchida ou esteja preenchida incorretamente, o *script* Ant gerado não irá funcionar.

8.3.2 Configurações

Além das comuns a todos os geradores que herdam de `MobileVCGenerator`, este gerador tem as configurações descritas na tabela 8.7.

Nome	**Descrição**	**Valor inicial**
`midletName`	Valor do atributo `MIDlet-Name` do JAD (Java Descriptor). *String* que aparece como nome da aplicação no dispositivo.	Não tem valor inicial
`midletVendor`	Valor do atributo `MIDlet-Vendor` do JAD. Nome de quem construiu a aplicação (pessoa ou empresa).	Não tem valor inicial
`midletVersion`	Valor do atributo `MIDlet-Version` do JAD. Versão da aplicação.	1.0
`midletIcon`	Valor do atributo `MIDlet-Icon` do JAD. Nome da figura que é o ícone desta aplicação. A figura deve estar no formato PNG e estar no diretório `src/resources`.	Não tem valor inicial
`cldcVersion`	Valor do atributo `MicroEdition-Configuration` do `MANIFEST.MF` do JAR contendo a aplicação. Versão do CLDC (`Connected Limited Device Configuration`) [50] necessária para a execução da aplicação. Dois valores possíveis: 1.0 e 1.1.	1.0
`midletVersion`	Valor do atributo `MicroEdition-Profile` do `MANIFEST.MF` do JAR contendo a aplicação. Versão do MIDP (`Mobile Information Device Profile`) [59] necessária para a execução da aplicação. Dois valores possíveis: 1.0 e 2.0.	1.0

Tabela 8.7: Configurações específicas do gerador de aplicações para a plataforma J2ME

8.3.3 `GeneratedController`

A classe abstrata `GeneratedController` gerada é marcada como implementando a interface `MIDletLifecycle`. Esta interface define métodos relacionados ao ciclo de vida de um MIDlet (aplicação J2ME). Eles estão descritos na tabela 8.8. Como `GeneratedController` não implementa nenhum dos métodos de `MIDletLifecycle`, a implementação de `Controller` feita pelo usuário deve implementar esta interface. O protótipo de implementação gerado provê uma implementação vazia dos métodos de `MIDletLifecycle`.

Nome do método	Descrição
`void startApp()`	Método chamado quando a aplicação entra no estado `Active`.
`void pauseApp()`	Método chamado quando a aplicação entra no estado `Paused`.

Tabela 8.8: Interface `MIDletLifecycle`

Campos de formulário do tipo `Gauge` devem ter um valor máximo. Desta forma, caso a aplicação tenha algum campo deste tipo, a classe abstrata `GeneratedController` gerada também é marcada como implementando a interface `GaugeFieldHandler`. Ela define um único método, `int getMaximumGaugeFieldValue(String formName, String fieldName)`, que retorna o valor máximo do campo de nome `fieldName` do formulário de nome `formName`. O protótipo da classe que implementa a lógica da aplicação tem uma implementação temporária deste método.

8.3.4 Implementação de `ViewRenderer`

Classes que implementam a interface gráfica de algum tipo de elemento devem ser subclasses da classe abstrata `Renderer`. Os métodos cujo nome começam com `notify` nada fazem e devem ser sobrescritos caso o objeto sendo implementado precise reagir a algum tipo de evento notificado por estes métodos. Seus métodos estão listados na tabela 8.9.

Método	**Descrição**
`abstract Displayable getDisplayable()`	Retorna a instância de `javax.microedition.lcdui.Displayable` (classe pertencente à API de MIDP) que mostra o elemento na tela.
`void notifyMadeCurrent()`	Método chamado pelo `GenericMIDPViewRenderer` quando este `Renderer` passa a ser o corrente e o `Displayable` retornado pelo método `getDisplayable` deste objeto passa a ser mostrado para o usuário.
`void notifyDismissed()`	Método chamado pelo `GenericMIDPViewRenderer` quando este `Renderer` deixa de ser o corrente e o `Displayable` retornado pelo método `getDisplayable` deste objeto deixa de ser mostrado para o usuário.
`void notifyPauseApp()`	Método chamado pelo `GenericMIDPViewRenderer` quando esta aplicação entra no estado `Paused` (pausado).
`void notifyStartApp()`	Método chamado pelo `GenericMIDPViewRenderer` quando esta aplicação entra no estado `Active` (ativo).
`void dispose()`	Método chamado pelo `GenericMIDPViewRenderer` quando esta aplicação está sendo terminada. Esta implementação nada faz e deve ser sobrescrita caso a subclasse tenha que liberar algum recurso que ela tenha alocado.

Tabela 8.9: Interface `Renderer`

A classe que implementa `ViewRenderer`, `GenericMIDPViewRenderer`, provê métodos necessários na implementação de `Renderer`s. Eles estão descritos na tabela 8.10.

`void process( ElementInfo elementInfo, Object data)`	Método que deve ser chamado depois que o usuário final interagiu com um elemento da aplicação. O primeiro parâmetro é o `ElementInfo` que descreve o elemento com o qual o usuário final interagiu e o segundo um objeto descrevendo a interação.
`String getOkString()`	Retorna a *string* a ser utilizada como rótulo em botões do tipo `OK`.
`String getSubmitString()`	Retorna a *string* a ser utilizada como rótulo em botões de submissão de formulários.
`String getTrueString()`	Retorna a *string* a ser utilizada para mostrar um valor booleano verdadeiro em formulários.
`String getFalseString()`	Retorna a *string* a ser utilizada para mostrar um valor booleano falso em formulários.

Tabela 8.10: Alguns métodos de `GenericMIDPViewRenderer`

Além dos arquivos gerados por `MobileVCGenerator`, este gerador gera uma classe, `LauncherMIDlet`, subclasse de `javax.microedition.midlet.MIDlet`, que é o ponto de partida da aplicação gerada. Esta classe cria as instâncias de `ViewRenderer` e das implementações de `Controller` e `UserImplementedViewRenderer` escritas pelo usuário e repassa a elas os eventos de ciclo de vida de aplicações J2ME através dos métodos da interface `MIDletLifecycle`.

8.3.5 Implementação de `UserImplementedViewRenderer`

A implementação de `UserImplementedViewRenderer` a ser escrita pelo usuário deve implementar a interface `MIDPUserImplementedViewRenderer`, descrita na tabela 8.11. Esta interface estende `MIDletLifecycle`.

Nome do método	Descrição
`void startApp()`	Método herdado de `MIDletLifecycle`.
`void pauseApp()`	Método herdado de `MIDletLifecycle`.
`Renderer getRenderer( UserImplementedElementInfo info, Object parameter, GenericMIDPViewRenderer viewRenderer)`	Método que retorna a instância de `Renderer` que irá implementar a interface gráfica do elemento descrito por `info` usando o parâmetro `parameter`. O terceiro parâmetro deste método é a instância de `GenericMIDPViewRenderer` que está sendo executada.

Tabela 8.11: Interface `MIDPUserImplementedViewRenderer`

8.3.6 Diretórios

O diretório `lib`, dentro do diretório onde a aplicação foi gerada, serve como local onde são colocados os arquivos JAR que contêm código no qual a aplicação gerada depende. Eles são automaticamente colocados no *classpath* na compilação pelo *script* Ant e são incluídos no arquivo JAR que contém a aplicação. Não deve-se colocar no diretório `lib` JARs contendo a implementação de CLDC ou MIDP.

8.3.7 *Script* Ant

O *script* Ant criado por este gerador utiliza o pacote Antenna [42] para compilar, empacotar e executar a aplicação no J2ME Wireless Toolkit [58], um emulador de dispositivos J2ME fornecido gratuitamente pela Sun Microsystems. Para isto, é necessário que o usuário do MultiMAD preencha a propriedade `j2me.wireless.toolkit.dir` com o diretório onde o J2ME Wireless Toolkit está instalado.

Caso o projeto seja gerado de novo e já exista um `build.xml`, este é renomeado

para `build.xml.old` antes da geração do novo. Isto acontece com o objetivo de o *script* poder ser alterado e gerado novamente sem que se perca as alterações.

A tarefa padrão (comando `ant` na linha de comando) compila, empacota e executa a aplicação no J2ME Wireless Toolkit. São gerados os arquivos `[nome do projeto]-[midletVersion].jad` (descritor da aplicação) e `[nome do projeto]-[midletVersion].jar` (JAR contendo a aplicação). Exemplo: se o nome do projeto for `teste` e o valor da configuração `midletVersion` for `1.2.3`, serão gerados os arquivos `teste-1.2.3.jad` e `teste-1.2.3.jar`.

A tarefa `clean` (comando `ant clean` na linha de comando) remove todos os arquivos e diretórios criados pelo *script* e pela compilação da aplicação.

Capítulo 9

Exemplo passo a passo de utilização do MultiMAD

Nesta seção será apresentado um exemplo de utilização do MultiMAD na especificação e implementação de uma aplicação simples na qual o usuário pode cadastrar e controlar seus gastos do dia-a-dia. Este exemplo não tem a intenção de ser completo nem de ser um modelo a ser seguido, já que há diversas possibilidades de implementação de uma mesma aplicação. Será seguido o modelo genérico de aplicações.

9.1 Especificação da aplicação

A aplicação deve prover uma forma de seu usuário cadastrar seus gastos, sendo que cada gasto é representado por uma trio (descrição, valor, data). Deve-se apresentar uma lista com todos os gastos de um determinado dia e um resumo sobre todos os gastos. Num primeiro momento, esta aplicação não tem uma plataforma definida, seguindo assim o modelo genérico de aplicações. Como este não provê tipos de campos de formulário específicos para valores monetários nem para datas (vide seção 6.1), todos os campos do formulário que o usuário utilizará para entrar gastos terão tipo `String`.

A partir da descrição acima, especifica-se os elementos da aplicação:

Nome do elemento: `MenuPrincipal`
Título: Menu principal
Tipo: `Menu`
Conteúdo: Itens deste menu:

- Cadastrar gasto: vai para Cadastro.
- Listar gastos: vai para ListaDiasComGastos.
- Resumo dos gastos: vai para Resumo.
- Sair: sai da aplicação.

Próximo elemento: De acordo com o item selecionado pelo usuário.

Nome do elemento: `Cadastro`
Título: Cadastro de gasto
Tipo: `Form`
Conteúdo: Campos deste formulário:

- Descrição do gasto (tipo `String`)
- Valor (tipo `String`)
- Data (tipo `String`)

Próximo elemento: Se o cadastro for realizado com sucesso: MensagemCadastroOK; caso contrário: MensagemCadastroErro

Nome do elemento: `MensagemCadastroOK`
Título: Cadastro OK
Tipo: `Text`
Conteúdo: "O cadastro de gasto foi realizado com sucesso."
Próximo elemento: MenuPrincipal

Nome do elemento: `MensagemCadastroErro`
Título: Erro no cadastro
Tipo: `Text`
Conteúdo: Texto dinâmico descrevendo o erro
Próximo elemento: Cadastro

Nome do elemento: `ListaDiasComGastos`

Título: Escolha um dia para visualisar seus gastos
Tipo: `List`
Conteúdo: Lista dinâmica contendo um item para cada dia com gastos cadastrados
Próximo elemento: ListaGastosDeUmDia

Nome do elemento: `ListaGastosDeUmDia`
Título: Lista de gastos
Tipo: `List`
Conteúdo: Lista dinâmica contendo todos os gastos de um determinado dia
Próximo elemento: MenuPrincipal

Nome do elemento: `Resumo`
Título: Resumo dos gastos
Tipo: `Text`
Conteúdo: Texto dinâmico mostrando informações sobre os gastos tais como total e média por dia
Próximo elemento: MenuPrincipal

9.2 Descrição da aplicação no MultiMAD

Agora que já temos uma especificação da nossa aplicação, podemos descrevê-la no MultiMAD. Quando executamos a ferramenta, não há nenhum projeto aberto. A figura 9.1 mostra a interface gráfica do MultiMAD nesta situação.

O primeiro passo, então, é criar um novo projeto que iremos chamar de ControladorDeGastos. Para isso, clicamos no botão `Novo projeto` ou usamos a opção homônima no menu `Arquivo`. A figura 9.2 mostra o resultado.

Todo novo projeto segue inicialmente o modelo genérico de aplicações. Como este é exatamente o seguido por este projeto, não se faz necessário defini-lo.

O primeiro passo é entrarmos na aba `Informações sobre o projeto`. Preencheremos por enquanto apenas o campo `Nome` com `ControladorDeGastos` e uma pequena descrição do projeto. Salvaremos o

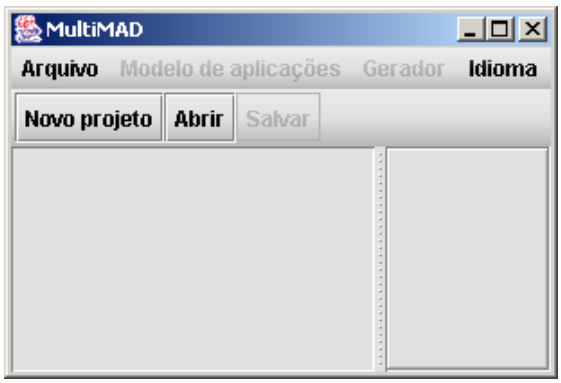

Figura 9.1: Janela inicial

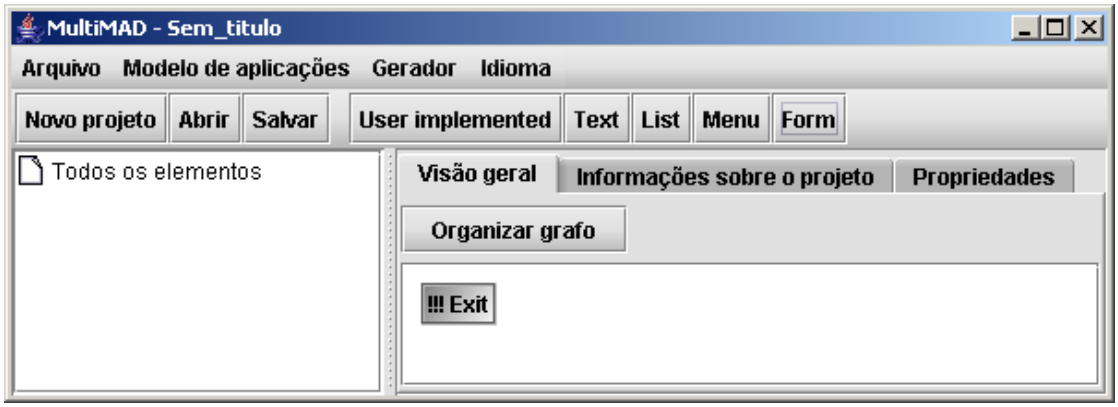

Figura 9.2: Novo projeto

projeto no diretório `E:\thiago\aula\mestrado\texto\projeto` com nome `controladorDeGastos.mmad` (`.mmad` é o sufixo dos arquivos do MultiMAD). A figura 9.3 mostra este passo.

Vamos agora criar o primeiro elemento do projeto. Podemos começar por qualquer um, então começaremos pelo primeiro a ser mostrado ao usuário da aplicação: `MenuPrincipal`. Para criá-lo, clicamos no botão `Menu` na barra de ferramentas (figura 9.4). A figura 9.5 mostra a janela de diálogo que pergunta qual é o nome do novo elemento sendo criado. Preenchemos com `MenuPrincipal` e damos o OK.

Aparece então uma nova aba onde será especificado o `MenuPrincipal` (figura 9.6). Estes passos são os mesmos para a criação de novos elementos de quaisquer tipos, por isso eles não serão repetidos para os outros elementos deste

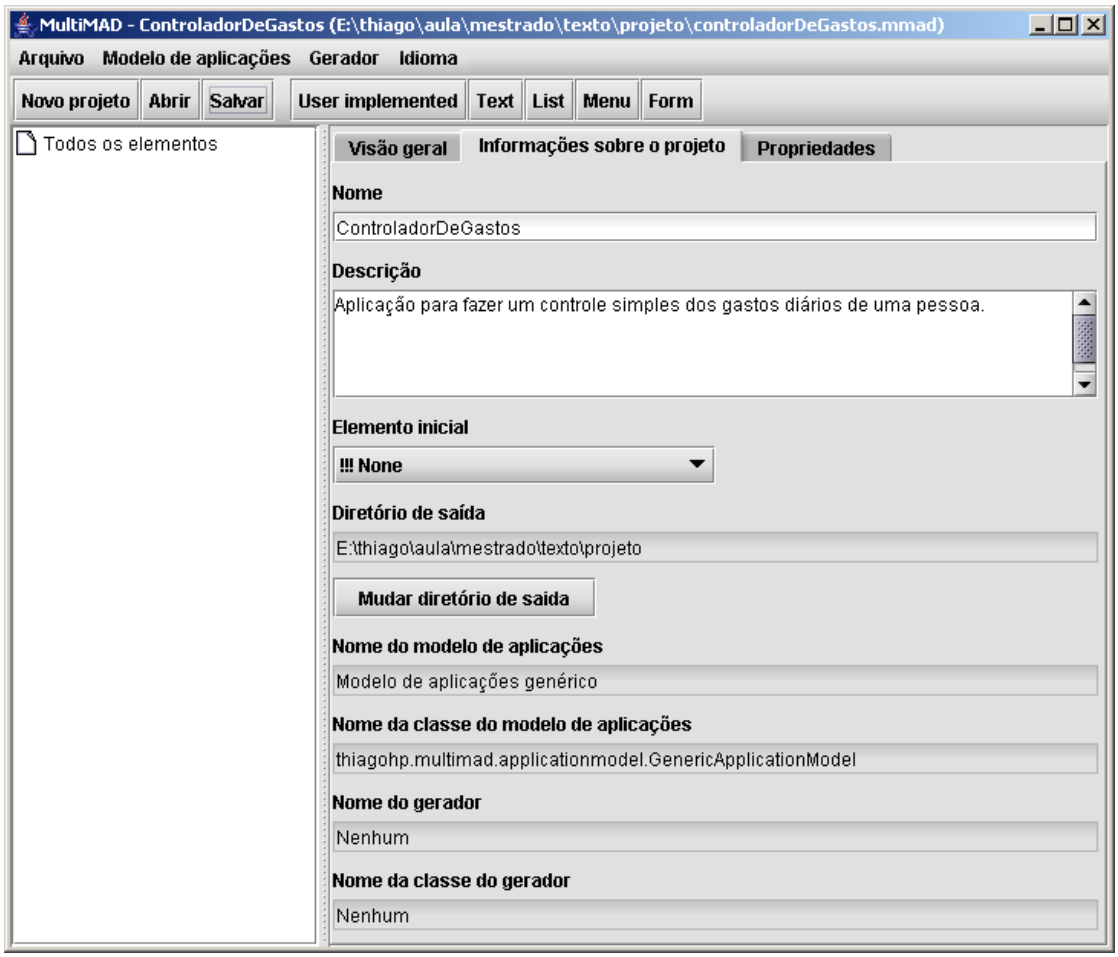

Figura 9.3: Informações iniciais sobre o projeto

Figura 9.4: Localização do botão de novo elemento do tipo `Menu`

projeto.

O título de um elemento é inicialmente igual ao seu nome. O nome é utilizado no código gerado e o título é o que mostrado ao usuário da aplicação. Preenchemos então o título com "Menu principal". Como os elementos para os quais os itens do menu principal apontam ainda não foram criados (exceto Sair), vamos, por

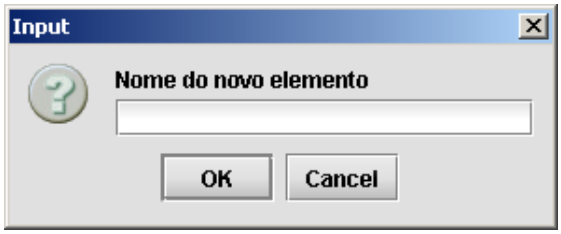

Figura 9.5: Diálogo de novo elemento do tipo `Menu`

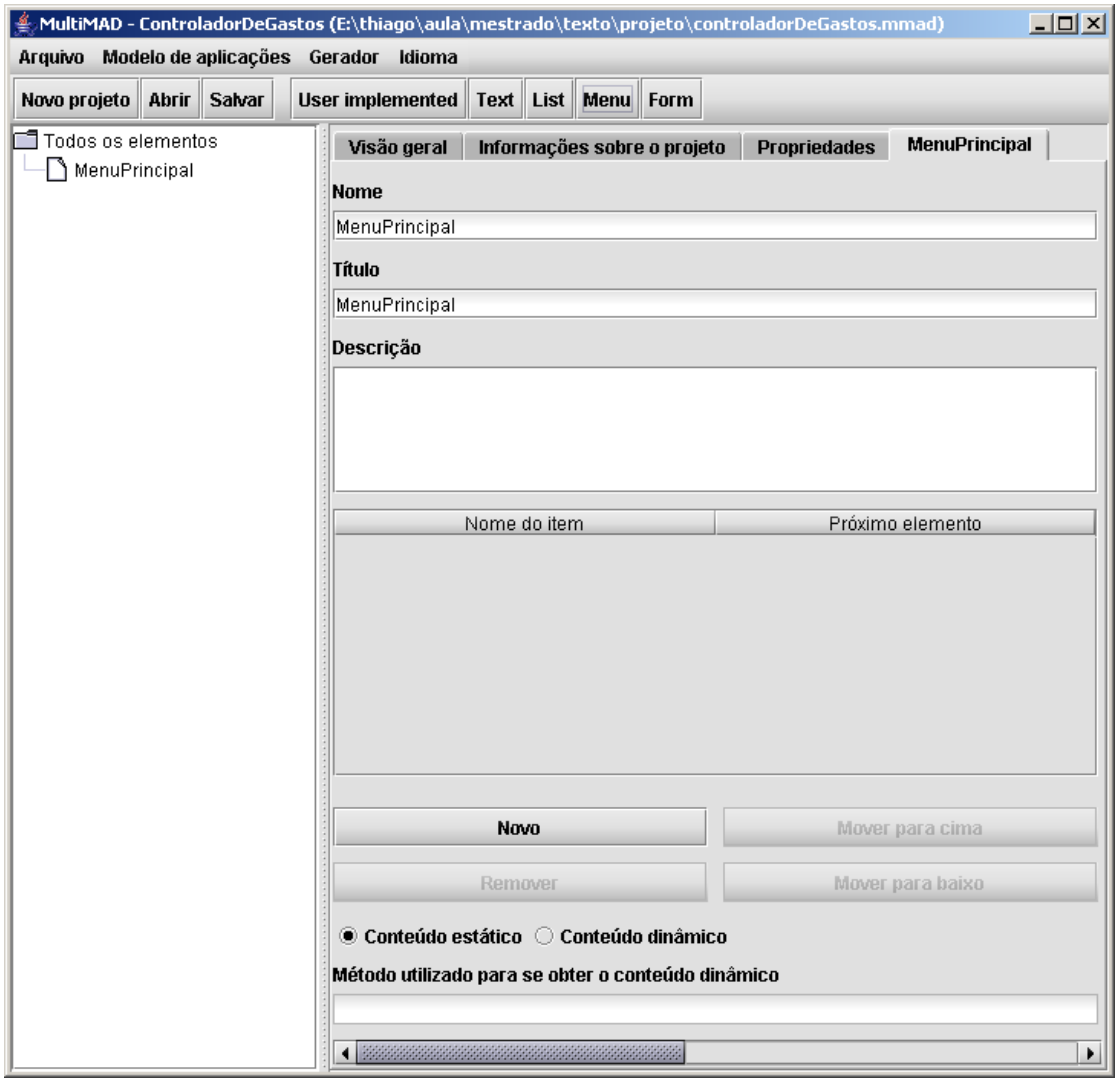

Figura 9.6: `MenuPrincipal` ainda não especificado

enquanto, colocar apenas os rótulos dos itens. Para criar um item, devemos clicar no botão `Novo`. Aparece então uma linha de tabela onde a primeira célula é um campo de texto onde se entra o rótulo e a segunda célula é uma *combo box* com as opções `!!! None` (próximo elemento não definido ainda), `!!! Exit` (sair da aplicação) e mais uma opção para elemento do projeto. A figura 9.7 mostra o resultado deste passo. Note que, no painel de elementos, à esquerda, o menu `MenuPrincipal` aparece em vermelho por não estar pronto para ser gerado: todos os itens tem que ter o próximo elemento definido, o que ainda não acontece.

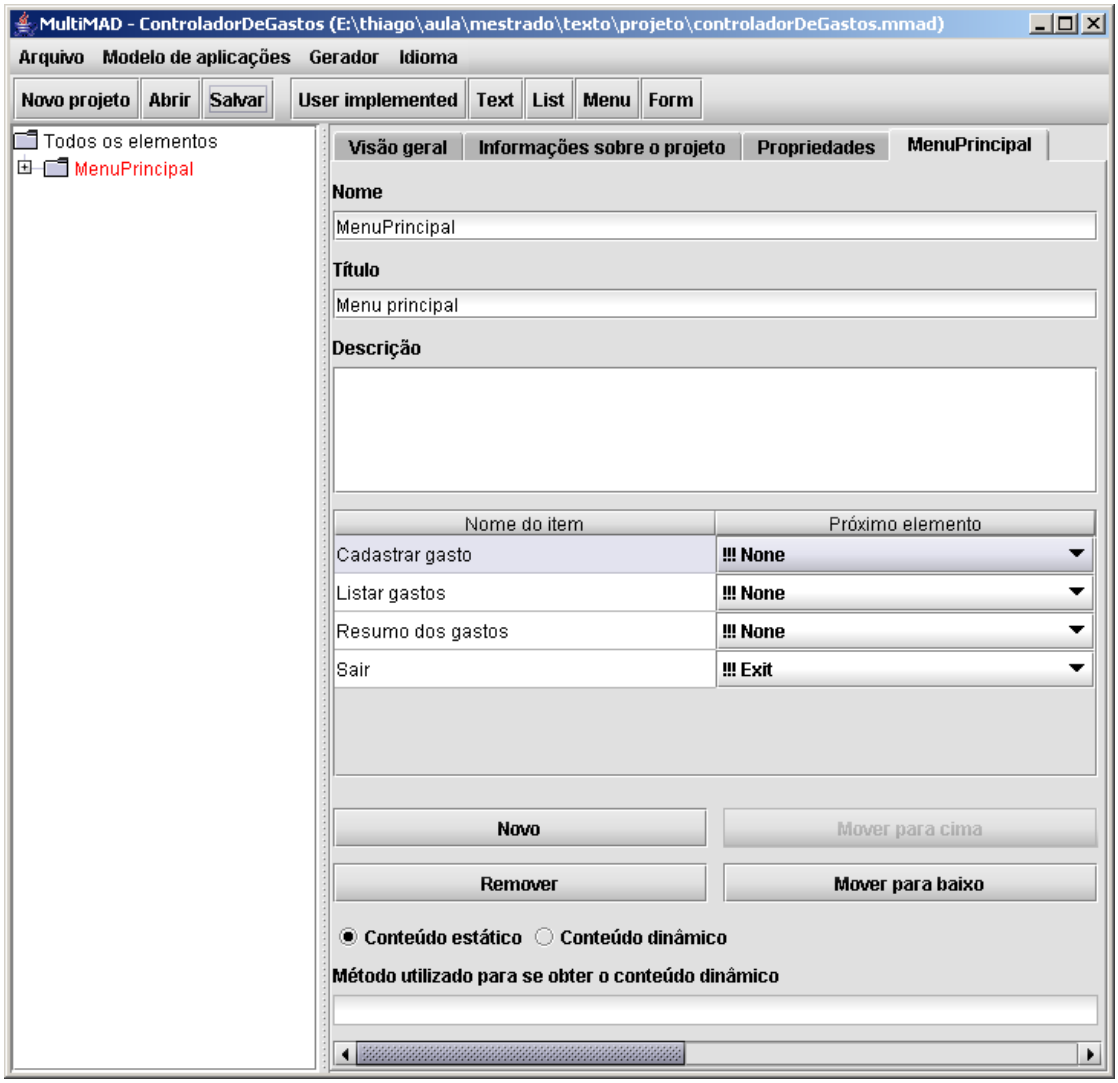

Figura 9.7: Especificação parcial de `MenuPrincipal`

Especifiquemos agora o `Cadastro` criando um novo formulário. Para cada campo devemos informar seu nome (utilizado apenas no código gerado), seu rótulo (descrição que aparece ao lado do campo), seu tipo e, opcionalmente, seu valor inicial. Não daremos valor inicial a nenhum campo. Como o próximo elemento de `Cadastro` é definido de forma dinâmica, ele não é definido na sua descrição no MultiMAD (opção `!!! None`). A figura 9.8 mostra o estado atual do elemento `Cadastro` no MultiMAD.

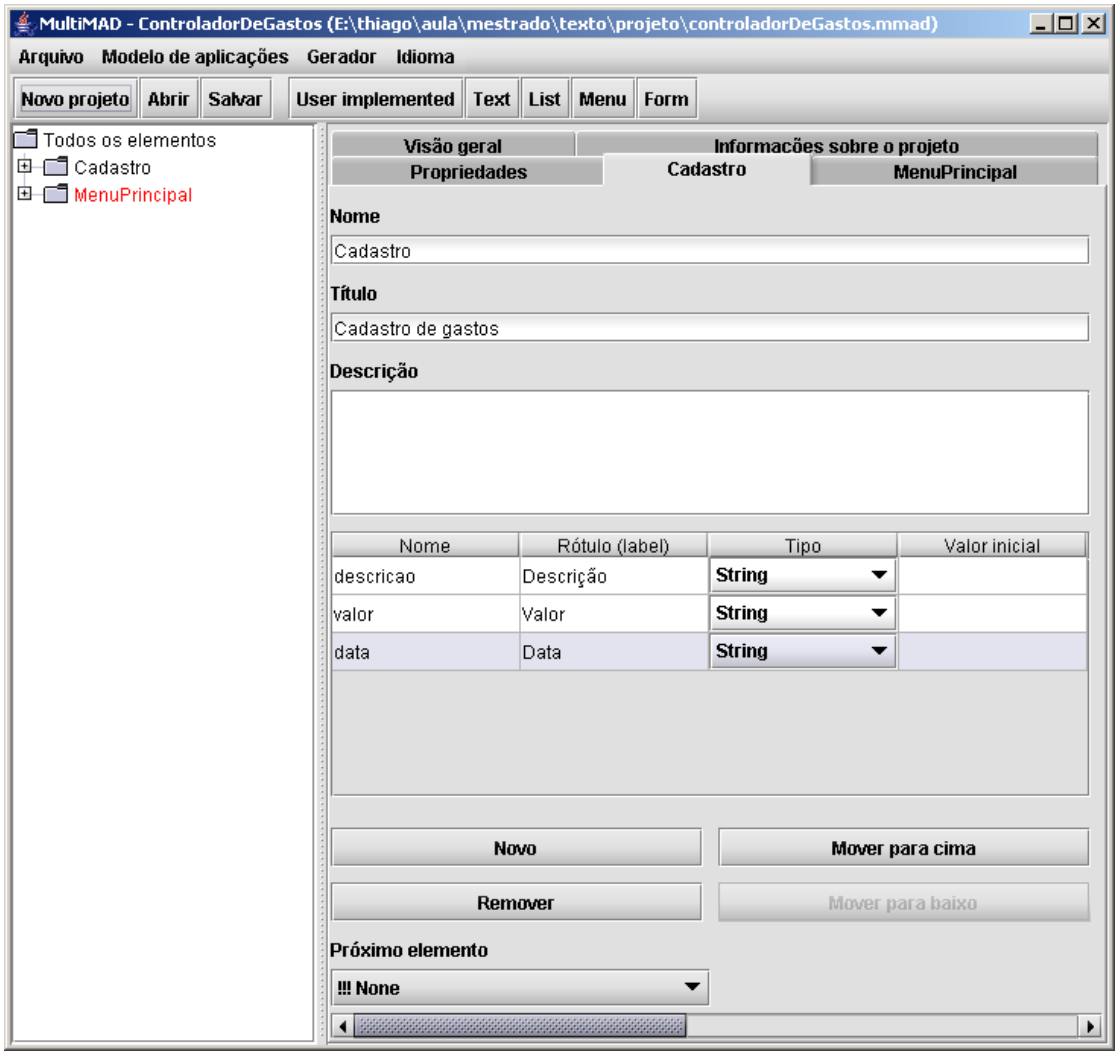

Figura 9.8: Especificação parcial de `Cadastro`

Em seguida, deve-se criar o elemento `MensagemCadastroOK`. Depois de criá-lo,

sendo o texto deste elemento estático, basta preencher o campo Texto. Definimos agora que o elemento a ser mostrado depois desse é `MenuPrincipal` clicando-se na *combo box* `Próximo elemento` e escolhendo a opção `MenuPrincipal`, tal como mostrado na figura 9.9.

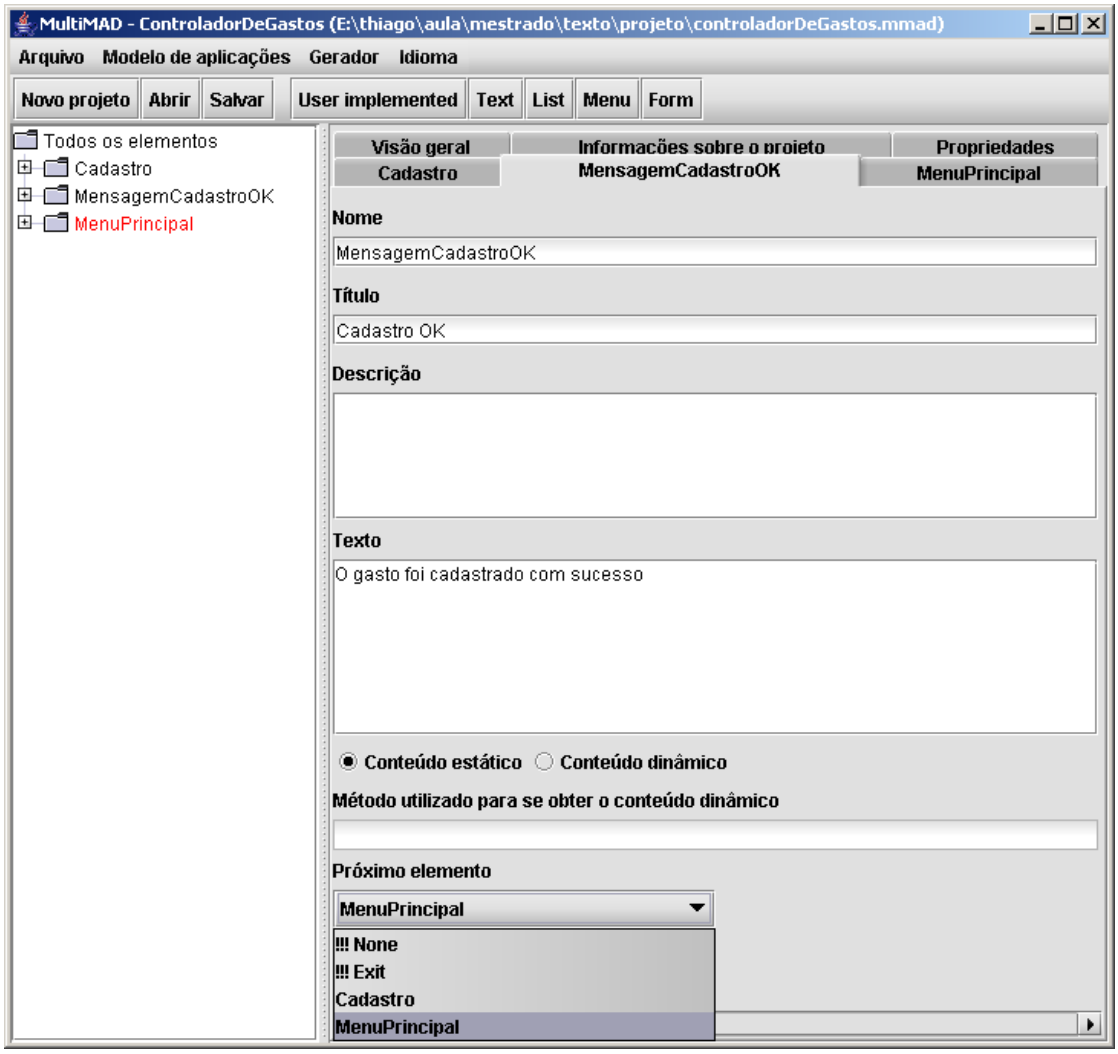

Figura 9.9: Especificação final de `MensagemCadastroOK`

Criemos agora o elemento `MensagemCadastroErro`. Como seu texto varia de acordo com o erro realizado pelo usuário no cadastro de um gasto, marcamos o conteúdo deste elemento como dinâmico e definimos que o texto será obtido através da chamada a um método de nome `obterDescricaoDeErroDeCadastro`, que será

implementado futuramente. Note que este elemento aparece em azul no painel de elementos já que ele precisará de código externo (implementado pelo usuário do MultiMAD).

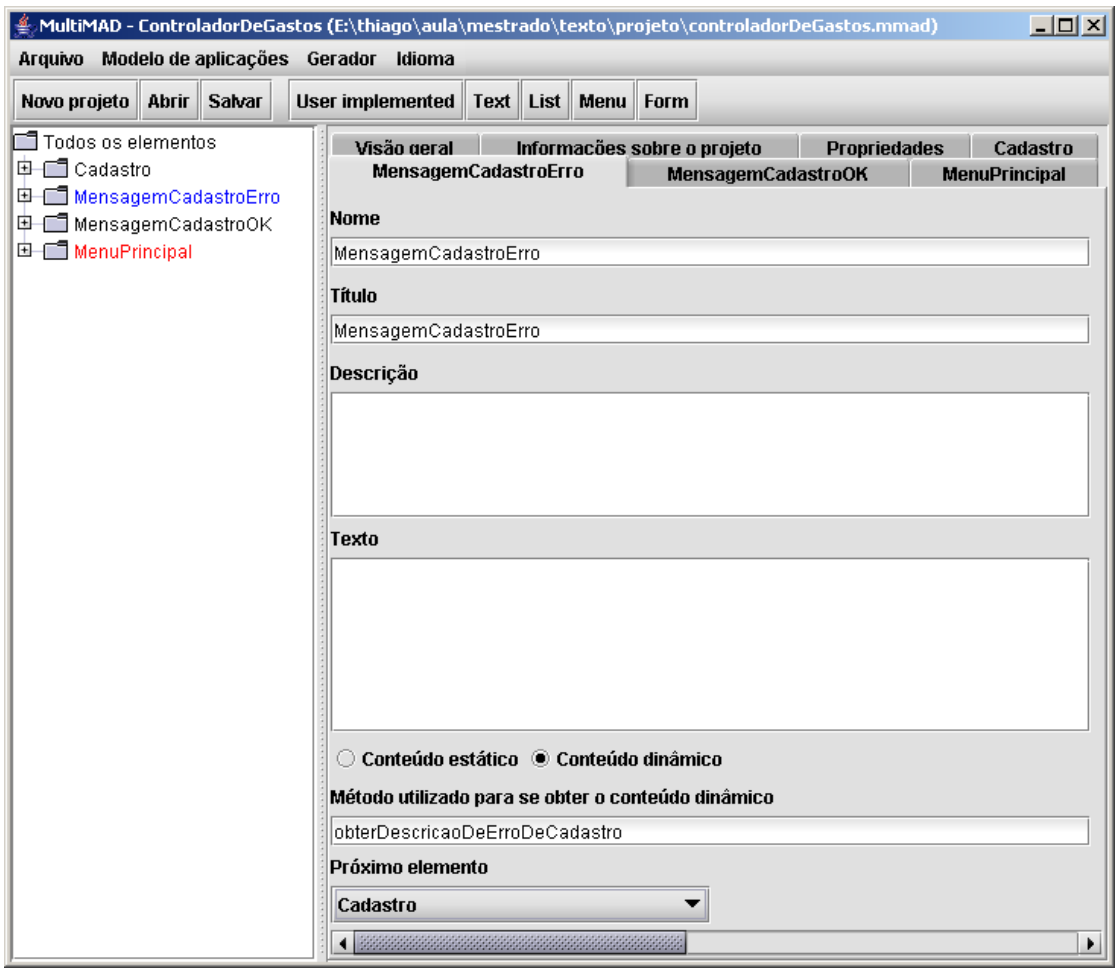

Figura 9.10: Especificação final de `MensagemCadastroErro`

Adicionemos agora a `ListaGastosDeUmDia`. Como o conteúdo de sua lista não é estático, marcamos o conteúdo deste elemento como dinâmico e definimos que o texto será obtido através da chamada a um método de nome `obterGastosDeUmDia`. O resultado deste passo pode ser visto na figura 9.11. O elemento `ListaGastosDeUmDia` é especificado de forma análoga.

O último elemento a ser criado é `Resumo`, especificado de forma análoga a `MensagemCadastroErro`.

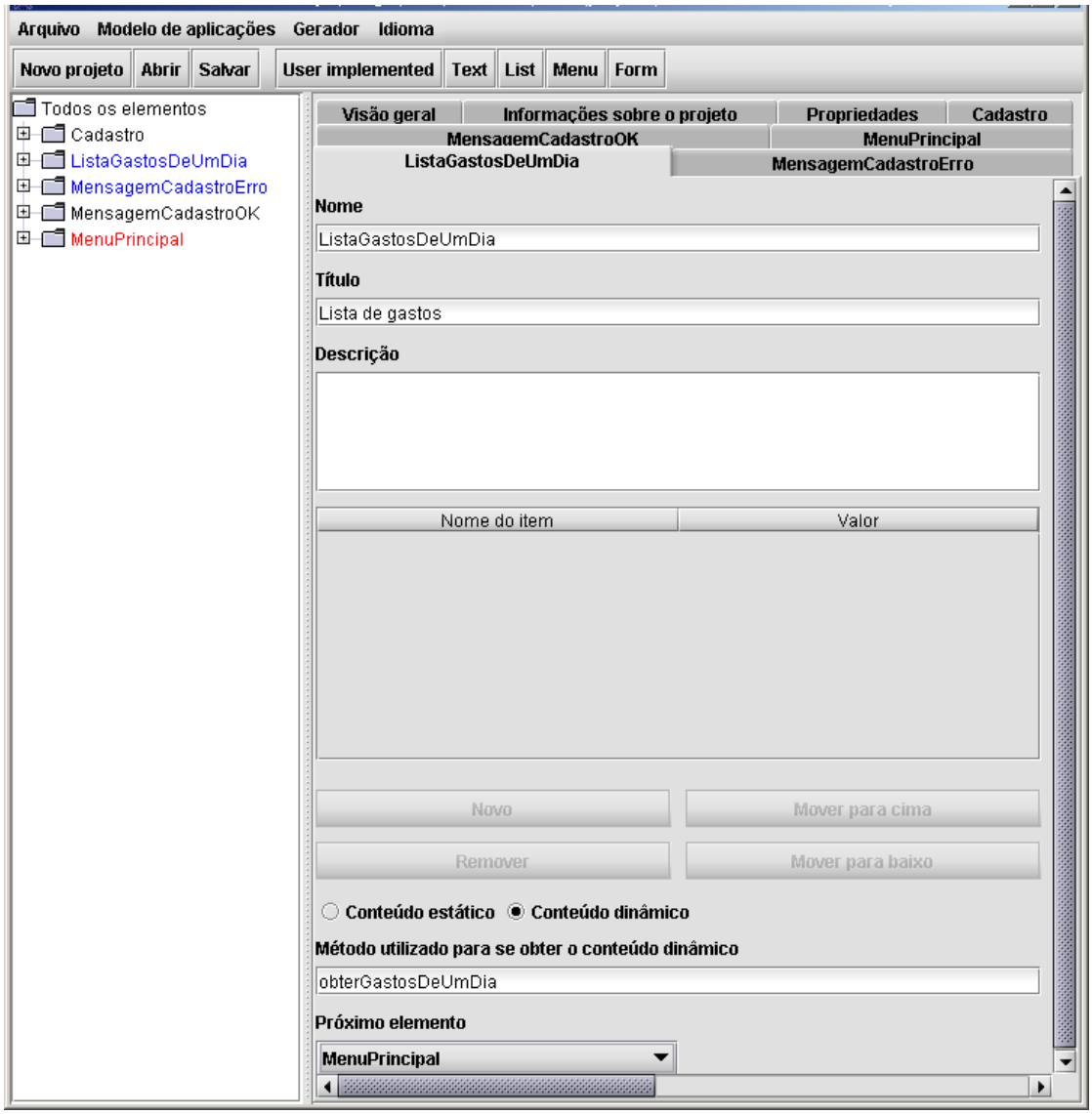

Figura 9.11: Especificação final de `ListaGastosDeUmDia`

Como todos os elementos que `MenuPrincipal` referencia já estão criados, podemos terminar sua especificação.

A especificação de todos os elementos já está pronta. Devemos agora voltar à aba `Informações sobre o projeto` e definir o elemento principal da aplicação (aquele que é o primeiro a ser mostrado). Ele é escolhido na janela que aparece quando o botão `Diretório de saída` é clicado (figura 9.12).

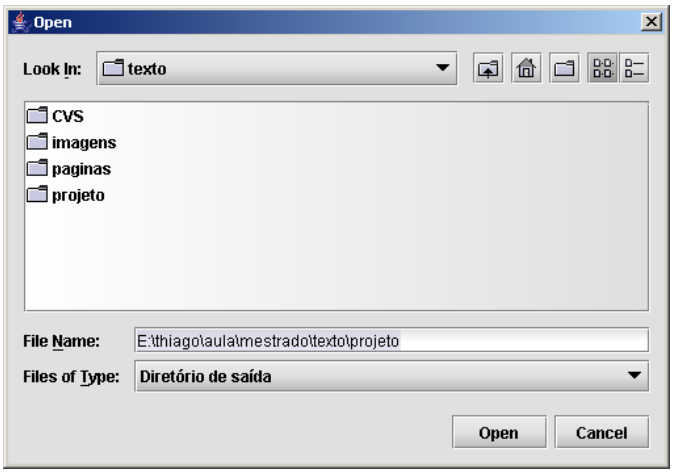

Figura 9.12: Escolha do diretório onde o projeto será gerado

Para termos uma visualização dos relacionamentos entre os elementos e, por conseqüência, dos possíveis fluxos de execução da aplicação, entramos na aba `Visão geral`. Num primeiro momento, o grafo está um pouco desorganizado. Depois de algumas operações de arrastar e soltar elementos do grafo, obtemos a visualização do projeto mostrada na figura 9.13.

9.3 Geração da aplicação

Por enquanto, ainda não foi definido o gerador utilizado por este projeto. Devemos então escolher um gerador dentre os disponíveis na ferramenta através do menu `Gerador`. Para este exemplo, utilizaremos o gerador de aplicações para Java 2 Micro Edition (seção 8.3).

Como o gerador escolhido é configurável, aparecerá na área de trabalho uma nova aba de nome `Configuração do gerador`. Preenchemos então os campos desta aba com os seguintes valores:

- `packageName`: *thiagohp.controladordegastos*
- `viewRendererClassName`: *thiagohp.multimad.mobilevc.midp.core.GenericMIDPViewRenderer* (valor sugerido pelo gerador)

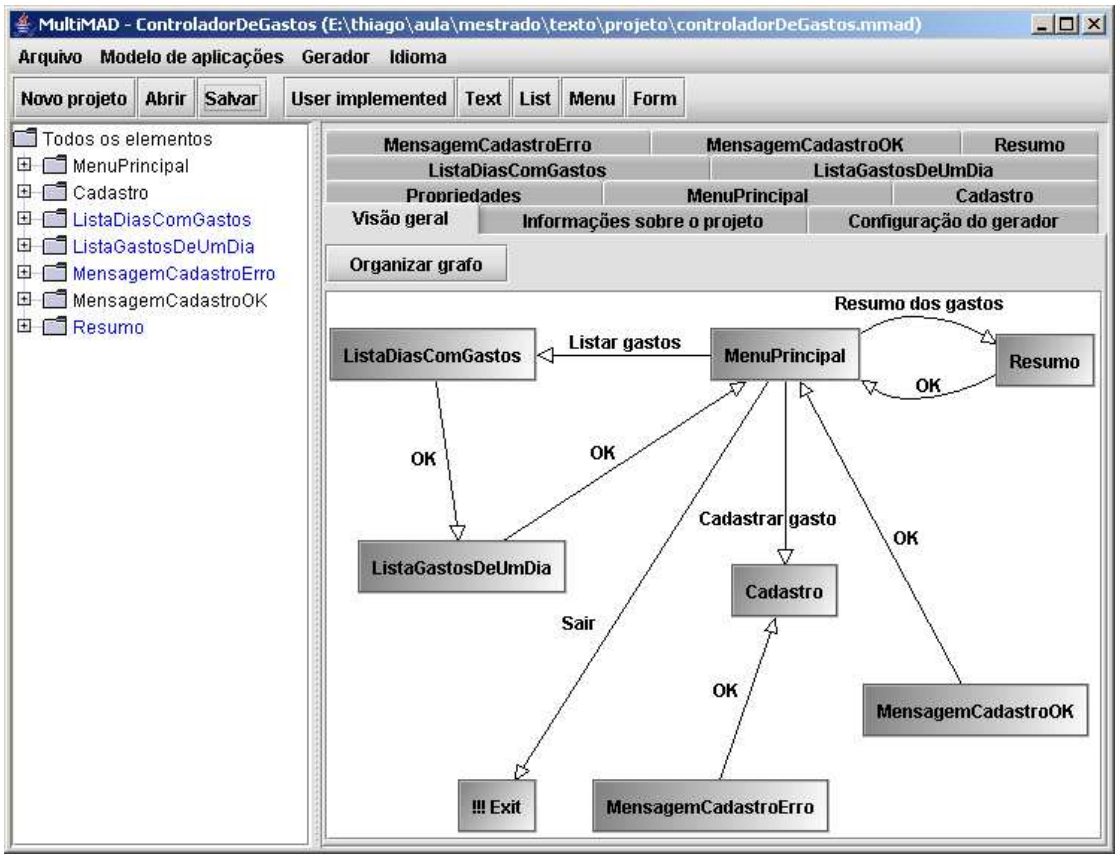

Figura 9.13: Visão geral do projeto

- `userImplementedViewRendererClassName`: *UserImplementedViewRendererImpl* (valor sugerido pelo gerador). Não vai ser utilizado já que este projeto não tem elementos do tipo `UserImplemented`
- `controllerClassName`: *ControllerImpl* (valor sugerido pelo gerador)
- `okString`: *OK* (valor sugerido pelo gerador)
- `submitString`: *Submeter*
- `trueString`: *true* (valor sugerido pelo gerador). Não será utilizado já que este projeto não tem campos de formulário do tipo `Boolean`)
- `falseString`: *false* (valor sugerido pelo gerador). Não será utilizado já que este projeto não tem campos de formulário do tipo `Boolean`)

- `midletName`: *Controlador de Gastos*
- `midletVendor`: *Thiago H. de Paula Figueiredo*
- `midletVersion`: *1.0* (valor sugerido pelo gerador)
- `cldcVersion`: *1.0* (valor sugerido pelo gerador)
- `midletVersion`: *1.0* (valor sugerido pelo gerador)
- `midletIcon`: campo não preenchido, já que não escolhemos um ícone para a aplicação.

A figura 9.14 mostra as configurações do gerador já preenchidas (exceto `midletIcon`, que não coube na tela).

O gerador de aplicações para a plataforma Java 2 Micro Edition (implementado pela classe `thiagohp.multimad.mobilevc.midp.GenericMIDPMobileVCGenerator`) possui uma propriedade, `j2me.wireless.toolkit.dir`, que deve ser preenchida com o diretório onde o Java 2 Micro Edition Wireless Toolkit (J2MEWTK) [58] para que o *script* Ant [2] gerado consiga funcionar. Para preenchermos a propriedade acima, clicamos na aba `Propriedades` da área de trabalho. Neste caso, o J2MEWTK está instalado no diretório `C:\j2meWirelessToolkit2.2`. A figura figura 9.15 mostra o preenchimento da propriedade.

Através do menu `Gerador` ou da combinação de teclado `Control-G`, requisitamos a geração do projeto. O gerador gera o código-fonte e um *script* Ant. Em seguida, executa o *script*. Aparece então uma janela mostrando as mensagens geradas pelo Ant, tal como pode ser visto na figura 9.16. O *script*, quando a compilação e empacotamento são realizados com sucesso, executa a aplicação no J2MEWTK.

A primeira tela mostrada pelo J2MEWTK apresenta uma lista contendo todas as aplicações disponíveis no dispositivo simulado (apenas a que estamos desenvolvendo), o que pode ser visto na figura 9.17. `MenuPrincipal`, que é totalmente estático (conteúdo e próximo elemento), pode ser visto na figura 9.18. A figura 9.19 mostra o elemento `Cadastro`. Os outros elementos têm conteúdo estático e tiveram uma implementação temporária gerada pelo MultiMAD. Esta implementação

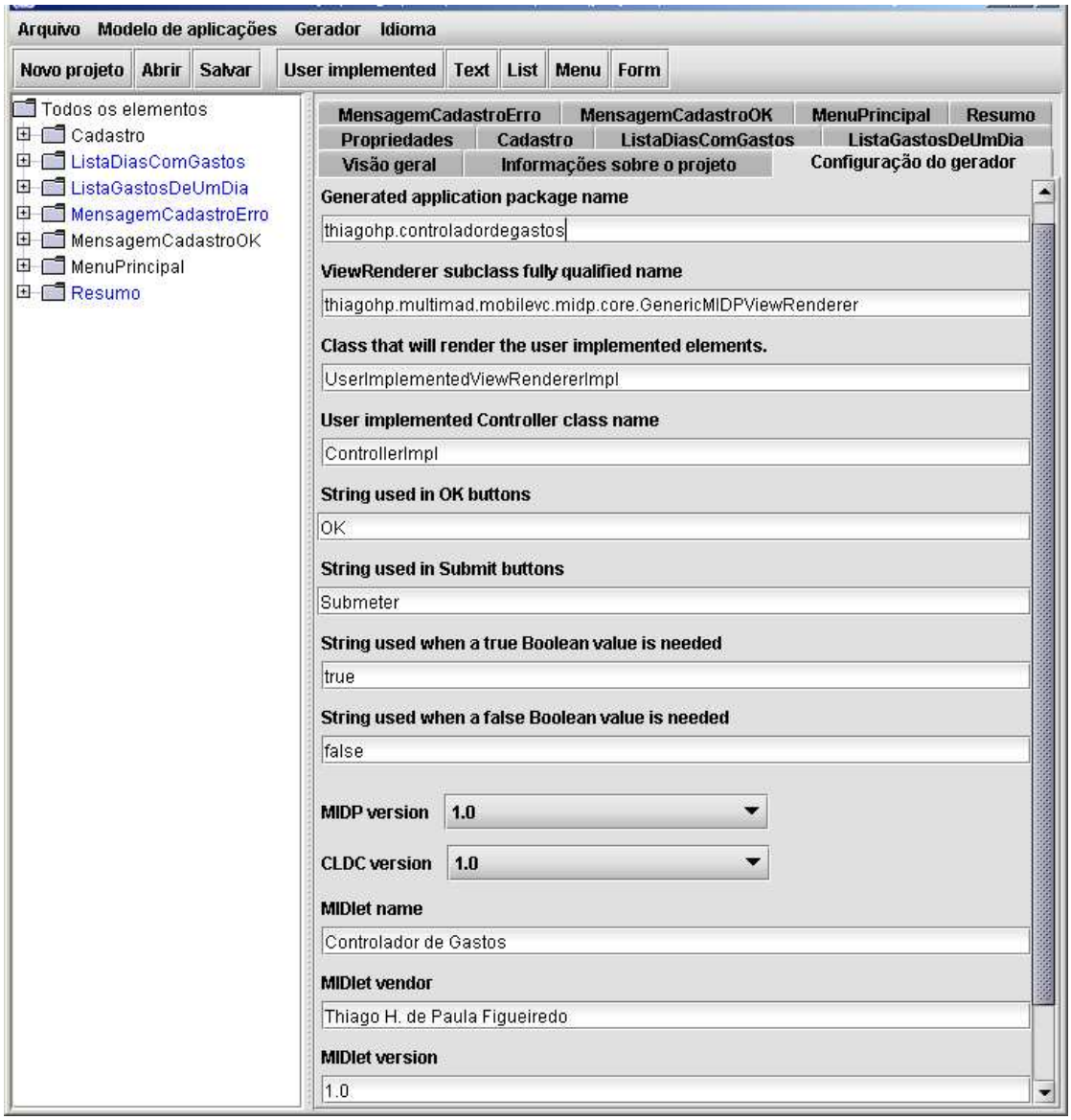

Figura 9.14: Configurações do gerador para a plataforma Java 2 Micro Edition

tipicamente mostra uma mensagem dizendo que aquele elemento ainda não foi implementado. Um exemplo é `Resumo` (figura 9.20).

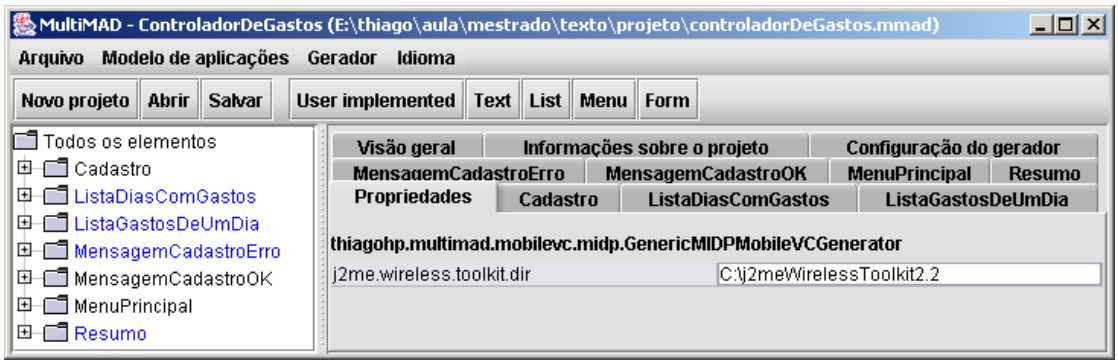

Figura 9.15: Propriedade do gerador para a plataforma Java 2 Micro Edition

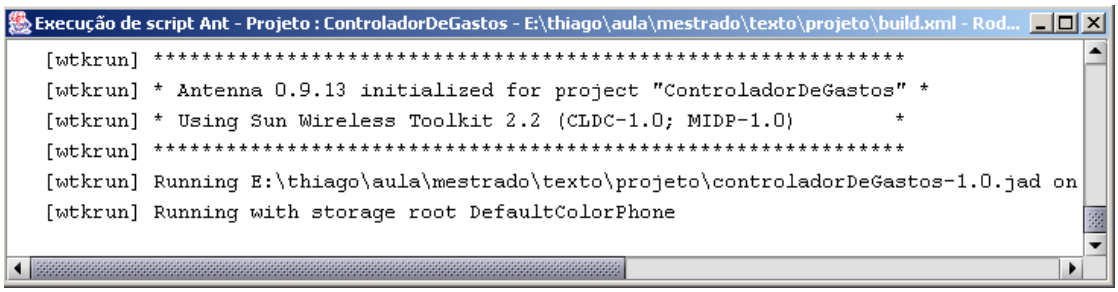

Figura 9.16: Mensagens geradas pela execução do *script* Ant

9.4 Implementação da lógica da aplicação

A implementação gerada pelo MultiMAD de `thiagohp.controladordegastos.ControllerImpl`, a classe que implementa a lógica da aplicação, já contém todos os métodos necessários para que a aplicação compile. Falta implementar a lógica da aplicação e trocar a implementação temporária dos métodos por uma implementação definitiva, o que foge do escopo deste documento.

Figura 9.17: Tela de seleção de aplicação no Java 2 Micro Edition Wireless Toolkit

Figura 9.18: `MenuPrincipal` no Java 2 Micro Edition Wireless Toolkit

Figura 9.19: Implementação temporária de `Cadastro` no Java 2 Micro Edition Wireless Toolkit

Figura 9.20: Implementação temporária de `Resumo` no Java 2 Micro Edition Wireless Toolkit

Capítulo 10

Conclusões e trabalhos futuros

Este capítulo discute as conclusões obtidas ao final do desenvolvimento desta dissertação (seção 10.1) e os trabalhos futuros (seção 10.2).

10.1 Conclusões

Este trabalho teve como objetivo a especificação e a construção de uma uma ferramenta de desenvolvimento de aplicações para dispositivos móveis. A ferramenta construída, denominada MultiMAD (*Multimodel Mobile Application Development Tool*, Ferramenta Multimodelo de Desenvolvimento de Aplicações para Dispositivos Móveis), implementa a maioria dos requisitos desejáveis citados no capítulo 4. Os requisitos não satisfeitos são incluídos na lista de trabalhos futuros.

O maior diferencial do MultiMAD é a possibilidade de adição de novos modelos de aplicações e geradores. Ela, ao contrário de todas as outras ferramentas similares descritas no capítulo 3 (exceto BeanWatcher, uma ferramenta específica para aplicações de monitoração), não tem uma visão única de quais são os blocos de construção de aplicações disponíveis. Os conceitos de modelo de aplicações e de gerador fazem com que o MultiMAD seja, ao mesmo tempo, uma ferramenta genérica e específica em relação a plataformas de aplicações para dispositivos móveis. Genérica porque o MultiMAD provê um modelo genérico, descrito na seção 6.1, cujos tipos de elementos foram definidos com o objetivo de serem su-

portados pela maioria (idealmente, pela totalidade) das plataformas de aplicações para dispositivos móveis. Específica porque a ferramenta permite que modelos de aplicações e geradores específicos para determinadas plataformas sejam adicionados sem que ela precise ser alterada.

Outra característica muito importante do MultiMAD é a geração de um protótipo funcional das classes que implementam a lógica de aplicação e a interface gráfica de elementos implementados pelo usuário da ferramenta. Ela permite que um protótipo da aplicação seja executado em um emulador sem que o usuário saia do MultiMAD e sem escrever sequer uma linha de código. O código-fonte do protótipo também serve como ponto de partida para a implementação real da aplicação, fazendo com que o usuário tenha apenas que substituir o corpo dos métodos gerados pela sua implementação real.

Uma grande vantagem do MultiMAD é liberdade que ela dá ao usuário. A ferramenta permite a ele decidir, para cada tela de sua aplicação, se ele irá implementá-la ou se ele utilizará um modelo pronto (tipo de elemento provido por um modelo de aplicações, exceto o tipo `UserImplemented`). Desta forma, o trabalho do usuário não fica limitado ao que a ferramenta oferece. Além disso, caso escolha apenas modelos prontos, o usuário não precisará escrever nenhuma linha de código de interface gráfica.

10.2 Trabalhos futuros

Como qualquer ferramenta de desenvolvimento, é possível imaginar várias outras funcionalidades que poderiam ser melhoradas ou adicionadas ao MultiMAD. A lista abaixo contém alguns dos trabalhos futuros mais interessantes:

- Definir modelos de aplicações e geradores para outras plataformas de aplicações para dispositivos móveis. Neste trabalho foram implementados o modelo genérico, um modelo específico para a plataforma Java 2 Micro Edition (MIDP 1.0) e geradores para as plataformas WAP (1.1 e 2.0) e Java 2 Micro Edition (MIDP 1.0).

- Adicionar suporte a alertas (*alerts*) e *tickers* no modelo de aplicações para

a plataforma Java 2 Micro Edition (MIDP 1.0) e criar um modelo específico para a plataforma Java 2 Micro Edition (MIDP 2.0).

- Implementar um emulador de projetos na interface gráfica do MultiMAD, aumentando a sua capacidade de prototipagem de aplicações.

- Utilizar as informações contidas na descrição da aplicação para realizar análises de consistência tais como verificação formal, elementos não alcançáveis, número de interações necessárias para se chegar a um determinado elemento. Uma possibilidade é a geração de um arquivo de entrada para a ferramenta de verificação formal NuSMV [10].

- Estender o conceito de modelo de aplicações para incluir formas de descrição de estruturas de dados de aplicações. Entre outros aspectos, isto permitiria que geradores pudessem gerar código de definição, armazenamento em memória secundária e transmissão de dados.

- Permitir que o usuário indique quais são as possibilidades de elemento a ser mostrado depois de elementos que não tenham essa informação definida. Ela é necessária para que a visualização em grafo e que eventuais futuras verificações de consistência possam ser feitas de maneira mais completa e precisa.

- Adicionar suporte a tipos enumerados de campos de formulário.

- Permitir que aplicações também sejam especificadas graficamente através de arrastar e soltar (*drag and drop*), tal como no Mobility Pack do NetBeans [51].

- Adicionar internacionalização de projetos ao MultiMAD. Isto permitiria a geração de aplicações que podem ser apresentadas ao usuário em mais de uma língua e/ou geração de versões de uma mesma aplicação em mais de uma língua.

- Estender o gerador de WAP/WML para gerar páginas em outras linguagens de marcação tais como VoiceXML [72] e cHTML [25].

- Permitir que cada projeto tenha mais de uma configuração de modelos de aplicações, uma por modelo utilizado. O mesmo vale para configurações de geradores.

- Definir na arquitetura MobileVC alguma forma de se obter dinamicamente os valores iniciais de campos de formulários.

- Otimizar, em termos de tamanho, a implementação de `ViewRenderer` provida pelo gerador de aplicações para a plataforma Java 2 Micro Edition. A implementação atual tem 33 kilobytes, enquanto o tamanho máximo de aplicações aceito por dispositivos está, neste momento, por volta de 64 kilobytes. Há pelo menos duas formas possíveis de se diminuir o tamanho de aplicações J2ME. A primeira é usar o menor número possível de classes, sacrificando a elegância da implementação em favor de um menor uso de memória. A outra é o uso de obfuscadores. O ProGuard [15], por exemplo, retira de um JAR classes, métodos e campos que não são utilizados, faz otimizações de código e altera o nome de classes, campos e métodos para *strings* muito curtas. Este último passo, além de dificultar a engenharia reversa de código-objeto, diminui bastante o tamanho do JAR, pois o código-objeto Java contém várias referências aos nomes de campos, métodos e classes.

- Implementar o suporte a elementos especiais. No presente momento, o MultiMAD supõe que todos os modelos de aplicação possuem um único elemento especial, o `EXIT` (sair).

- Internacionalização do modelo de aplicações para a plataforma Java 2 Micro Edition e ambos os geradores. Suas mensagens e *strings* estão disponíveis apenas em inglês até agora.

Apêndice A

Glossário e acrônimos

Ant Ferramenta de automatização de processos de *software* [2]. Semelhante à ferramenta `make`, porém é escrita na linguagem Java e o formato de seus *scripts* é XML ao invés do formato Makefile.

API Application Programming Interface (API). Conjunto de definições de como um componente de *software* pode se comunicar com outro.

Assistente Vide *wizard*.

BREW Binary Runtime for Wireless Environment [43]. Vide seção 2.2.6.

CDC Connected Device Configuration [49]. Vide seção 2.2.1.

cHTML Compact HyperText Markup Language [25]. Vide seção 2.1.4.

CLDC Connected Limited Device Configuration [50]. Vide seção 2.2.1.

Framework Um conjunto de classes, interfaces e padrões dedicados a resolver uma classe de problemas através de uma arquitetura extensível e flexível.

Gateway Nó de uma rede cuja função é a conversão de dados entre dois protocolos diferentes.

HDML Handheld Device Markup Language [26]. Linguagem de marcação propietária derivada de HTML e específica para dispositivos móveis. O protocolo

de transporte utilizado era o HDTP (*Handheld Device Transport Protocol*, Protocolo de Transporte para Dispositivos de Mão). Foi apenas utilizada na América do Norte e foi substituída por WML e WAP.

HTML HyperText Markup Language [68]. Vide seção 2.1.1.

IDE *Integrated Development Environment*. Tipo de ferramenta de desenvolvimento de *software* que tipicamente inclui um editor de código-fonte, um depurador (*debugger*) e recursos de compilação de aplicações.

Internacionalização Processo de desenho de uma aplicação de modo que ela possa ser adaptada para várias línguas e regiões.

Java 2 Standard Edition Versão de Java [7] específica para computadores pessoais.

J2ME Java 2 Micro Edition [57]. Vide seção 2.2.1.

J2MEWTK Java 2 Micro Edition Wireless Toolkit [58].

MAB Mobile Application Builder [32]. Vide seção 3.1.

MIDP Mobile Information Device Profile. Vide seção 2.2.1

Obfuscador Ferramenta que altera código-objeto de forma a dificultar ou impossibilitar a realização de engenharia reversa sobre ele.

Plataforma Conjunto de especificações, protocolos, componentes de *software* e/ou sistema operacional sobre o qual aplicações são construídas.

Plugin Componente de *software* que adiciona funcionalidades a uma ferramenta sem que ela seja alterada, recompilada, reempacotada ou reinstalada.

***Script* Ant** Documento XML que descreve que tarefas o Ant deve executar.

Servlet Classe Java que produz conteúdo dinâmico em um servidor Web.

VoiceXML Voice Extensible Markup Language [72]. Vide seção 2.1.5

XHTML Extensible HyperText Markup Language. [70].

XML Extensible Markup Language [71].

WAP Wireless Application Protocol [38]. Vide seção 2.1.3.

WCA Web Clipping Application. Vide seção 2.2.2.

Wizard Função ou parte de uma ferramenta que que ajuda o usuário a realizar, passo a passo, alguma tarefa complexa.

WML Wireless Markup Language [39]. Vide seção 2.1.3.

Referências Bibliográficas

[1] Amiga, Inc., *The Amiga Operating System*. http://www.amiga.com/amigaos/. Último acesso em 27 de março de 2005.

[2] Apache Software Foundation, *Apache Ant*. http://ant.apache.org/, 2004. Último acesso em 27 de março de 2005.

[3] ——, *Velocity*. http://jakarta.apache.org/velocity/, 2004. Último acesso em 27 de março de 2005.

[4] ——, *Apache Jakarta Tomcat*. http://jakarta.apache.org/tomcat/, 2005. Último acesso em 27 de março de 2005.

[5] Apple Computer, Inc., *Mac OS X*. http://www.apple.com/macosx/. Último acesso em 27 de março de 2005.

[6] ARM Ltd., *ARM Processors*. http://www.arm.com/products/CPUs/. Último acesso em 1 de abril de 2005.

[7] K. Arnold, J. Gosling, and D. Holmes, *The Java^{TM} Language Specification*, Addison-Wesley Professional, 3rd ed., June 2000.

[8] AVIDWireless, *AVIDRapidTools Release 2*. http://avidwireless.com/ARTRelease20.htm. Último acesso em 14 de março de 2005.

[9] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, *NuSMV 2: An OpenSource tool for symbolic model checking*, Conference on Computer-Aided Verification (CAV), (2002).

[10] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri, *NuSMV: A new symbolic model checker*, International Journal on Software Tools for Technology Transfer, 2 (2000), pp. 410–425.

[11] Data Representations, Inc, *Simplicity for Mobile Devices*. `http://www.datarepresentations.com/products/mobile/index.shtml`. Último acesso em 14 de março de 2005.

[12] ——, *Simplicity for Mobile Servers*. `http://www.datareps.com/products/mobileServers/index.jsp`. Último acesso em 14 de março de 2005.

[13] ——, *Simplicity for PalmOS Platform*. `http://www.datarepresentations.com/products/palm/index.shtml`. Último acesso em 14 de março de 2005.

[14] A. L. L. de Aquino, *BeanWatcher: uma ferramenta para o desenvolvimento de aplicações de monitoração em ambientes sem fio*, master's thesis, Departamento de Ciência da Computação, UFMG, Brasil, Novembro 2003.

[15] Eric Lafortune, *ProGuard*. `http://proguard.sourceforge.net/`. Último acesso em 3 de abril de 2005.

[16] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, *RFC 2616: Hypertext Transfer Protocol – HTTP/1.1*. `http://www.ietf.org/rfc/rfc2616.txt`, Junho 1999. Último acesso em 27 de março de 2005.

[17] G. H. Forman and J. Zahorjan, *The challenges of mobile computing*, IEEE Computer, 27 (1994), pp. 38–47.

[18] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns – Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, 1st ed., Jan. 1995.

[19] Hewlett-Packard Development Company, L.P., *HP handheld devices*. `http://welcome.hp.com/country/us/en/prodserv/handheld.html`. Último acesso em 27 de março de 2005.

[20] J. Hunter and B. McLaughlin, *JDOM*. `http://www.jdom.org`, 2004.

[21] IBM Corporation, *OS/2 Warp*. http://www-306.ibm.com/software/os/warp/. Último acesso em 27 de março de 2005.

[22] IDC, *Worldwide handheld market experiences third straight year of decline, according to IDC*. http://www.idc.com/getdoc.jsp?containerId=pr2005_01_28_185111, Fevereiro 2005. Último acesso em 27 de março de 2005.

[23] International Organization for Standardization, *Information Processing, Text and Office Systems, Standard Generalized Markup Language (SGML) = Traitement de l'information, systemes bureautiques, langage standard généralisé de balisage (SGML)*, International Organization for Standardization, Geneva, Switzerland, Oct. 1986. International Standard ISO 8879-1986.

[24] Java Community Process (JCP), *Java^{TM}Servlet 2.4 Specification – Final Release*. http://jcp.org/aboutJava/communityprocess/final/jsr154/index.html, 2004. Último acesso em 27 de março de 2005.

[25] T. Kamada, *Compact HTML for Small Information Appliances*. http://www.w3.org/TR/1998/NOTE-compactHTML-19980209/, Fevereiro 1998. World Wide Web Consortium (W3C) Note, Último acesso em 28 de março de 2005.

[26] P. King and T. Hyland, *Handheld Device Markup Language Specification*. http://www.w3.org/TR/NOTE-Submission-HDML-spec.html, Maio 1997. World Wide Web Consortium (W3C) Note, Último acesso em 14 de março de 2005.

[27] Microsoft Corporation, *ActiveSync*. http://msdn.microsoft.com/library/en-us/wceappenduser5/html/wce50oriactivesync.asp. Último acesso em 1 de abril de 2005.

[28] ——, *Microsoft .NET Compact Framework*. http://msdn.microsoft.com/mobility/gettingstarted/netcf/default.aspx. Último acesso em 1 de abril de 2005.

[29] ——, *Visual C++ Developer Center*. http://msdn.microsoft.com/visualc/. Último acesso em 29 de março de 2005.

[30] ——, *Windows CE.NET*. http://msdn.microsoft.com/embedded/windowsce/default.aspx. Último acesso em 1 de abril de 2005.

[31] ——, *Windows Family Home Page*. http://www.microsoft.com/windows/. Último acesso em 28/03/2005.

[32] A. C. O. Minelli, *Uma ferramenta para desenvolvimento de aplicações para dispositivos móveis*, master's thesis, Departamento de Ciência da Computação, UFMG, Brasil, Janeiro 2002.

[33] O. Nierstrasz, G. Arévalo, S. Ducasse, R. Wuyts, A. Black, P. Müller, C. Zeidler, T. Genssler, and R. van den Born, *A component model for field devices*, in First International IFIP/ACM Working Conference on Component Deployment, ACM, Berlin, Germany, June 2002.

[34] NTT DoCoMo, Inc., *i-mode*. http://www.nttdocomo.com/corebiz/imode/, Dezembro 2004. Último acesso em 28 de março de 2005.

[35] Open Mobile Alliance Ltd. http://www.openmobilealliance.org. Último acesso em 27 de março de 2005.

[36] ——, *SyncML Sync Protocol, version 1.0.1*. http://www.openmobilealliance.org/tech/affiliates/syncml/spec1-0-1.zip. Último acesso em 1 de abril de 2005.

[37] ——, *Wireless Application Protocol Architecture Specification Version 1.1*. http://www.openmobilealliance.org/tech/affiliates/wap/technical_1_1_20010720[1].zip, 2001. Último acesso em 15 de março de 2005.

[38] ——, *Wireless Application Protocol Architecture Specification Version 2.0*. http://www.openmobilealliance.org/tech/affiliates/wap/wap-210-waparch-20010712-a.pdf, 2001. Último acesso em 15 de março de 2005.

[39] ——, *Wireless Markup Language Version 2.0*. http://www.openmobilealliance.org/tech/affiliates/wap/wap-238-wml-20010911-a.pdf, 2001. Último acesso em 27 de março de 2005.

[40] Opera Software ASA, *Opera's Small-Screen Rendering^{TM}*. http://www.opera.com/products/mobile/smallscreen/. Último acesso em 27 de março de 2005.

[41] PalmSource, Inc, *PalmOS*. http://www.palmsource.com/palmos/. Último acesso em 27 de março de 2005.

[42] J. Pleumann, *Antenna*. http://antenna.sourceforge.net/, 2004. Último acesso em 27 de março de 2005.

[43] Qualcomm Incorporated, *Qualcomm BREW Home*. http://www.qualcomm.com/brew/. Último acesso em 27 de março de 2005.

[44] D. Raggett, *HTML 3.2 Reference Specification*. http://www.w3.org/TR/REC-html32, Janeiro 1997. W3C Recommendation, Último acesso em 27 de março de 2005.

[45] Seasam Time Oy, *A revolutionary wireless application development tool for creating wireless services*. http://www.seasam.com/products/brochures/Seasam_Time_White_Paper.pdf, 2003. White Paper, Último acesso em 14 de março de 2005.

[46] B. Smith, *Battle for the browser*. http://www.wirelessweek.com/index.asp?layout=article\&articleid=CA247030, Setembro 2002. Último acesso em 27 de março de 2005.

[47] Speedware Ltee./Ltd., *MobileDev*. http://www.speedware.com/solutions/wireless/mobile_development_tools/MobileDev/, 2005. Último acesso em 14 de março de 2005.

[48] Sun Microsystems, Inc., *Abstract Window Toolkit (AWT)*. http://java.sun.com/products/jdk/awt/. Último acesso em 28 de março de 2005.

[49] ——, *Connected Device Configuration (CDC)*. http://java.sun.com/products/cdc/. Último acesso em 27 de março de 2005.

[50] ——, *Connected Limited Device Configuration (CLDC)*. http://java.sun.com/products/cldc/. Último acesso em 27 de março de 2005.

[51] ——, *J2ME MIDP Development for NetBeans IDE 4.1*. http://www.netbeans.org/kb/41/mobility.html. Último acesso em 1 de abril de 2005.

[52] ——, *J2ME MIDP Development for NetBeans IDE 4.1 Quick Start Guide*. http://www.netbeans.org/kb/41/quickstart-mobility.html. Último acesso em 1 de abril de 2005.

[53] ——, *Java^{TM} 2 Platform, Enterprise Edition*. http://www.java.sun.com/j2ee. Último acesso em 1 de abril de 2005.

[54] ——, *Java^{TM} 2 Platform, Standard Edition*. http://java.sun.com/j2se/. Último acesso em 27 de março de 2005.

[55] ——, *NetBeans*. http://www.netbeans.org. Último acesso em 1 de abril de 2005.

[56] ——, *Solaris 10*. http://www.sun.com/software/solaris/. Último acesso em 27 de março de 2005.

[57] ——, *Java^{TM} 2 Platform, Micro Edition*. http://java.sun.com/j2me/docs/j2me-ds.pdf, 1999. White paper, Último acesso em 27 de março de 2005.

[58] ——, *Java^{TM} 2 Platform, Micro Edition Wireless Toolkit*. http://java.sun.com/j2me/docs/j2me_wireless_tlkit.pdf, 1999. White paper, Último acesso em 27/03/2005.

[59] ——, *Mobile Information Device Profile (MIDP)*. http://java.sun.com/products/midp/midp-ds.pdf, 1999. White paper, Último acesso em 27 de março de 2005.

[60] ——, *Over The Air User Initiated Provisioning Recommended Practice for the Mobile Information Device Profile*. http://java.sun.com/products/midp/OTAProvisioning-1.0.pdf, Maio 2001. Último acesso em 1 de abril de 2005.

[61] ——, *JAR File Specification*. http://java.sun.com/j2se/1.4.2/docs/guide/jar/jar.html, 2003.

[62] ——, *Duke heads to Cannes for the 3GSM World Congress*. http://java.com/en/promotions/3gsm.jsp, Fevereiro 2005. Último acesso em 27 de março de 2005.

[63] ——, *Sun Microsystems Shows Solutions to Monetize 3G at 3GSM World Congress*. http://java.sun.com/j2me/events/3gsm05/index.jsp, Fevereiro 2005. Último acesso em 27 de março de 2005.

[64] Symbian Ltd., *Symbian OS – the mobile operating system*. http://www.symbian.com/. Último acesso em 1 de abril de 2005.

[65] Tekrati Inc., *Wireless PDAs, Business Buyers Drive PDA Market to Record Revenues, Says Gartner*. http://www.tekrati.com/T2/Analyst_Research/ResearchAnnouncementsDetails.asp?Newsid=4517, Fevereiro 2005. Último acesso em 27 de março de 2005.

[66] Wabasoft Corporation, *Wabasoft*. http://www.wabasoft.com. Último acesso em 27 de março de 2005.

[67] World Wide Web Consortium (W3C). http://www.w3.org. Último acesso em 19 de março de 2005.

[68] ——, *HTML 4.01 Specification*. http://www.w3.org/TR/html4/, Dezembro 1999. Último acesso em 14 de março de 2005.

[69] ——, *XHTML^{TM} Basic*. http://www.w3.org/TR/xhtml-basic/, Dezembro 2000. W3C Recommendation, Último acesso em 27 de março de 2005.

[70] ——, *XHTML^{TM} 1.0 The Extensible HyperText Markup Language (Second Edition)*. http://www.w3.org/TR/xhtml1/, Agosto 2002. W3C Recommendation, Último acesso em 14 de março de 2005.

[71] ——, *Extensible Markup Language (XML) 1.0 (Third Edition)*. http://www.w3.org/XML/, 2004. W3C Recommendation.

[72] ——, *Voice Extensible Markup Language (VoiceXML) Version 2.0*. `http://www.w3.org/TR/voicexml20/`, Março 2004. W3C Recommendation, Último acesso em 28 de março de 2005.