Uma Ferramenta para Desenvolvimento de Aplicações para Dispositivos Móveis

André C. O. Minelli
minelli@dcc.ufmg.br

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

Belo Horizonte - MG
21 de fevereiro de 2003

Resumo

A convergência entre telefonia celular, Internet e computação móvel tem trazido novos desafios em praticamente todas as áreas da Ciência da Computação. O principal objetivo é permitir o acesso a informações em qualquer lugar, a qualquer momento.

Muitas promessas têm sido feitas, várias tecnologias têm sido propostas e desenvolvidas, mas uma questão ainda não foi inteiramente resolvida: como desenvolver aplicações para este novo ambiente, que está evoluindo e se modificando rápida e continuamente. É nesta questão que este trabalho é focado.

Após rever e analisar brevemente as alternativas de desenvolvimento existentes atualmente, é proposta e especificada uma ferramenta capaz de ajudar projetistas de aplicações para dispositivos móveis. Uma possível arquitetura e implementação, baseadas nesta especificação, são descritas e os resultados obtidos são apresentados.

Abstract

The convergence among cellular telephony, Internet and mobile computing has brought new challenges to almost every Computer Science area. The main goal is to allow information access anywhere anytime.

Many promises has been made, many technologies has been proposed and developed, but an issue was not fully resolved yet: how to develop applications to this new environment which is quickly and continuously evolving and changing. This work is focused on this issue.

After briefly review and analyze existing development alternatives a tool capable of help mobile devices applications designers is proposed and specified. A feasible architecture and implementation based on this specification are described and the obtained results are presented.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A área de computação móvel está gerando uma nova e ampla frente de pesquisa e desenvolvimento, não só em Ciência da Computação mas em outras áreas de conhecimento. A crescente proximidade da computação móvel com a telefonia celular e com a Internet abriu ainda mais as possibilidades para a criação de novas tecnologias e, em conseqüência, de novos serviços e produtos [1].

Atualmente, dispositivos computacionais móveis com as mais diferentes características estão se tornando populares pelo mundo. Celulares com acesso à Internet, assistentes pessoais (PDA's), *laptops* e vários tipos de *handhelds* proliferam-se em diversos setores e atividades, fazendo com que serviços e informações possam ser acessados a qualquer momento e em qualquer lugar. Estes dispositivos fazem parte da área de computação móvel, ou computação ubíqua [2]. Relógios, carros e até eletrodomésticos também deverão fazer parte deste conjunto em um futuro próximo.

Existem muitas projeções sendo feitas e expectativas sendo criadas com base nestas novas tecnologias. De acordo com o *DC Research* e *The Yankee Group* [3], em 2005 o número de usuários de telefones com acesso à Internet somarão aproximadamente 1 bilhão de pessoas no mundo, sendo que esta marca deverá ser alcançada ainda em 2003 se considerarmos a utilização de outros dispositivos móveis. Por outro lado, o *Gartner Group* [4] prevê que tecnologias sem fio irão tornar os trabalhadores móveis cerca de 30% mais produtivos. Existem muitos outros números semelhantes a estes, o que parece apontar para, no mínimo, uma grande massificação dos dispositivos móveis.

Apesar de todos os possíveis e prováveis benefícios, estes dispositivos possuem diversas limitações se comparados aos computadores disponíveis atualmente:

1. CPU menos poderosa
2. Menos memória
3. Consumo de energia restrito
4. Telas menores
5. Mecanismos de entrada de dados restritos

6. Largura de banda reduzida
7. Maior latência das respostas
8. Menor estabilidade de conexões

Enquanto os cinco primeiros itens são características de dispositivos móveis, os três últimos são devidos às restrições de um ambiente de comunicação sem fio. Apesar de todas estas restrições, estes dispositivos trazem consigo a conveniência de poderem acessar informações a qualquer momento e em qualquer lugar.

Podemos então distinguir duas formas principais de aplicações para dispositivos móveis operarem, em escala crescente de complexidade:

1. *Stand-alone*, onde o dispositivo móvel funciona como um "canivete suíço", incorporando sistemas utilitários diversos, como por exemplo calculadora, conversor de medidas e jogos. A principal característica destas aplicações é o isolamento: todo processamento feito é mantido no dispositivo, sem qualquer troca ou alimentação de entidades externas. Embora aplicações possam ser carregadas através de uma rede de comunicação, nenhum processamento depende de recursos externos.

2. *Cliente/servidor*, onde o dispositivo móvel deve comunicar-se com entidades externas para enviar ou receber dados. Podemos distinguir três tipos de comunicação, dependendo da maneira e frequência com que o dispositivo realiza transferências de dados:

   - *Off-line*, quando dados podem ser carregados e processados de forma independente no dispositivo, mas que precisam ser atualizados e descarregados no sistema computacional fixo. É caracterizado pela necessidade de sincronização: o dispositivo móvel deve trocar e sincronizar informações com um sistema central fixo de tempos em tempos. Organizadores pessoais e bancos de dados são exemplos típicos.
   - *On-line*, quando o dispositivo pode comunicar-se com o sistema computacional fixo a qualquer momento, normalmente através de um enlace sem fio. Qualquer processamento depende da existência desta comunicação. Compreende aplicações como leitor de e-mail, envio de mensagens de texto através de *Short Message Service* (SMS), serviços baseados em localização, acesso à bases de dados e *m-commerce*.
   - *Agent-based*, baseado em agentes de *software* móveis, uma forma mista das duas anteriores. Aqui a comunicação pode ser interrompida sem prejuízo e inclui não apenas dados, mas também o próprio processamento [5, 7]. Através de um agente, um processamento é iniciado no terminal móvel e é capaz de ser transferido para o sistema computacional fixo, onde coleta informações e recursos necessários para completar sua tarefa e então retornar para o terminal, quando

este estiver disponível. Muito útil em aplicações que necessitam de recursos além dos disponíveis nos dispositivos móveis. Qualquer aplicação dos outros tipos pode ser desenvolvida com este paradigma.

As aplicações cliente/servidor são as mais interessantes e úteis, tendo como dispositivos típicos os *laptops*, os celulares e os PDA's. Celulares e PDA's oferecem um desafio à criação destas aplicações por três razões principais: são mais limitados computacionalmente se comparados aos *laptops*; existem em grande variedade; possuem uma enorme gama de ferramentas para desenvolvimento de aplicações.

1.1 Objetivos

O objetivo principal deste trabalho é especificar e desenvolver uma ferramenta capaz de definir fontes de dados, entradas, saídas e interfaces externas de aplicações para dispositivos móveis, de maneira independente destes dispositivos. A idéia é atrasar, tanto quanto possível, a inclusão no projeto de peculiaridades dos dispositivos-alvo na aplicação. Desta forma, um mesmo projeto poderá ser reutilizado para outros dispositivos. Esta ferramenta deverá ainda oferecer flexibilidade para incorporar outras tecnologias (como por exemplo fontes de dados, interfaces de usuário e equipamentos móveis).

A validação e avaliação da ferramenta e deste trabalho serão feitas através de duas aplicações de exemplo, que servirão ainda para ilustrar a utilização da ferramenta.

Para alcançar os objetivos propostos, a utilização de padrões abertos ou estabelecidos será usada sempre que possível. Esta estratégia potencialmente aumenta a compatibilidade e facilidade em geral da aplicação, já que a massa crítica tanto de conhecimento quanto de ferramentas será bastante considerável. Exemplos destes padrões são o protocolo HTTP, muito bem conhecido e utilizado para comunicação entre entidades, e a linguagem de marcação XML [9], amplamente utilizada para intercâmbio de dados. Esta última é especialmente interessante devido à sua capacidade de descrever o conteúdo e isolar a informação de sua apresentação.

1.2 Contribuições

Este trabalho traz três contribuições principais:

- a revisão das tecnologias de desenvolvimento que mostra o panorama atual das opções disponíveis em cada plataforma. As principais características de ferramentas e ambientes de desenvolvimento para diferentes plataformas de celulares e PDA's são brevemente apresentadas. Três tecnologias que em breve deverão mudar este panorama, trazendo maior flexibilidade, são também apresentadas.

- a especificação dos requisitos desejáveis para uma ferramenta capaz de atender ao desenvolvimento de aplicações para as várias plataformas de dispositivos móveis. Estes requisitos foram levantados de modo a atender amplamente às demandas de desenvolvedores de aplicações para dispositivos móveis.

- a implementação de ferramenta em si. Toda uma arquitetura de desenvolvimento foi especificada, atendendo à um sub-conjunto dos requisitos obtidos. Como resultado obteve-se uma ferramenta funcional, capaz de auxiliar desenvolvedores de aplicações para várias plataformas de dispositivos móveis.

1.3 Estrutura da dissertação

Este trabalho está organizado da seguinte forma. O capítulo 2 traz uma breve apresentação das principais opções existentes para o desenvolvimento de aplicações para dispositivos móveis. O capítulo 3 apresenta alguns dos trabalhos relacionados à esta dissertação, mostrando as ferramentas existentes atualmente no mercado. No capítulo 4 são traçados os principais requisitos necessários para uma ferramenta capaz de auxiliar desenvolvedores de aplicações para dispositivos móveis. O capítulo 5 traz a descrição da arquitetura e implementação de uma ferramenta desenvolvida em concordância com os requisitos propostos. Os testes realizados usando duas aplicações de exemplo são apresentados no capítulo 6. Finalmente, no capítulo 7 são apresentadas as conclusões desta dissertação e perspectivas de futuros trabalhos. O apêndice A traz os documentos de definição dos formatos XML especificados neste trabalho. No apêndice B encontra-se um glossário dos principais termos utilizados neste documento.

Capítulo 2

Plataformas de desenvolvimento para dispositivos móveis

A diversidade de dispositivos móveis traz consigo uma variedade de plataformas e, junto com esta, uma gama enorme de ferramentas de desenvolvimento. A seguir serão apresentadas as características das principais tecnologias existentes atualmente.

2.1 Desenvolvimento para Celulares

O desenvolvimento de aplicações para celulares tem se baseado na utilização de linguagens de marcação, principalmente WML, sub-conjunto do WAP [11]; HDML, precursora do WAP, desenvolvida e mantida pela Openwave, anteriormente conhecida como Phone.com; e CHTML, subconjunto do HTML padrão, utilizado no sistema i-Mode, no Japão. Pode-se ainda incluir nesta lista o VoiceXML, linguagem de marcação para aplicações baseadas em comandos de voz.

2.1.1 WML

O padrão WAP, mantido pelo Open Mobile Alliance (OMA [12], anteriormente conhecido como WAP Forum, um consórcio internacional de fabricantes e desenvolvedores do "mundo sem fio"), é o padrão *de facto* para a criação de aplicações para telefones celulares e terminais de acesso sem fio. Para isto, o WAP define uma pilha de protocolos, onde WML está situado na camada de aplicação. WML é responsável pela marcação de como o conteúdo será exibido para estes dispositivos.

As principais características do padrão WML são:

- linguagem baseada em XML.

- organização do conteúdo utiliza a metáfora de um baralho (*deck*) de cartas (*cards*). Cada *card* corresponde à uma interação com o usuário. Um *deck* corresponde à um documento WML, contendo um ou mais *cards*.

- possibilidade de declarar variáveis dentro de um *deck*, as quais são substituídas por seu valor pelo navegador.

- existência de eventos e tarefas, que são acionados quando o navegador muda de estado, como por exemplo quando um novo *card* é carregado.

Toda a pilha WAP foi adaptada e otimizada para operação sem fio, o que já traz vantagens para o desenvolvedor. Especificamente no que se refere a WML, podemos citar a codificação binária do conteúdo (garantindo a utilização reduzida da largura de banda), previsibilidade do comportamento (um navegador compatível com uma versão de WAP tem a garantia de apresentar o conteúdo daquela versão) e disponibilidade de eventos e temporizadores (oferecendo flexibilidade, mesmo em conteúdos estáticos).

Porém WML ainda não está bem adaptado para dispositivos como os PDA's, cujos usuários possuem expectativas de interfaces ainda não cobertas pelo padrão WAP [13]. Além disso, existe ainda a questão da interpretação da especificação, uma vez que o padrão, em busca da compatibilidade, deixa em aberto para a implementação alguns elementos. Por exemplo, não especifica como elementos de interface devem aparecer ou se comportar.

2.1.2 HDML

HDML foi a primeira linguagem de marcação disponível para telefones celulares. Desenvolvida pela Openwave [14], tem sido amplamente licenciada para fabricantes de aparelhos, principalmente na América do Norte. Embora seja um padrão aberto, o controle é mantido pela Openwave. Sua sintaxe é a mesma do HTML e, inclusive, algumas marcações são idênticas. Porém apresenta uma organização do documento e recursos de variáveis e eventos semelhantes à WML.

2.1.3 CHTML

Compact HTML (CHTML) é um sub-conjunto da linguagem HTML 4.0, e foi desenvolvida pela empresa japonesa Access Company Ltd [15]. No início de 1998 foi submetida ao organismo que regula as atividades da Internet, o W3C (World Wide Web Consortium), que a aceitou [**?**]CHTML-W3C). A CHTML utiliza o protocolo HTTP sobre TCP/IP. Os celulares i-Mode da NTT DoCoMo possuem *minibrowsers* CHTML que já suportam gráficos, GIF's animadas e arquivos MIDI para download.

CHTML não suporta alguns dos recursos disponíveis na *Web*, e que requerem maiores quantidades de memória ou de processamento. Desta maneira, não são suportados *frames*,

imagens JPEG, tabelas, mapas de imagens, múltiplas fontes e estilos de caracteres e folhas de estilos (*style sheets*).

2.1.4 VoiceXML

Mais uma linguagem de marcação, mas que, ao contrário das demais, permite a criação de aplicações baseadas em comandos de voz. Desenvolvida pelo VoiceXML Forum [17], seu principal objetivo é integrar serviços de voz e dados usando o paradigma cliente/servidor. A linguagem descreve a interação homem-máquina de um sistema de resposta audível.

Usando VoiceXML, um serviço de voz é definido como uma seqüencia de diálogos (audíveis) de interação com o usuário, especificados em um conjunto de documentos. Um diálogo em VoiceXML é conduzido por um interpretador VoiceXML, que também é responsável pela entrada de dados (via voz ou caracteres - DTMF), assinalar valores a variáveis e tomar decisões (definidos nos documentos).

VoiceXML é uma opção interessante para prover aplicações para praticamente qualquer telefone, seja ele móvel ou não, uma vez que nenhum recurso extra é necessário. Funciona muito bem para serviços de navegação em geral, sendo que aplicações mais sofisticadas são limitadas apenas pela plataforma de suporte ao reconhecimento de voz.

2.2 Desenvolvimento para PDA's

O maior poder computacional dos PDA's, se comparados aos celulares, permite o desenvolvimento de aplicações com mais recursos. Linguagens de marcação ainda estão disponíveis (e, na prática, são bastante utilizadas), sendo em geral subconjuntos de HTML. Mas neste segmento de dispositivos já é possível desenvolver aplicações nativas, utilizando todos os recursos que uma determinada plataforma ofereça.

Podemos separar os *handhelds* em três grandes plataformas: o *PalmOS* [18], desenvolvido pela Palm Inc., o *Windows CE* [19] (que faz parte da família *Windows Embedded*) desenvolvido pela Microsoft, e o Symbian OS/EPOC [20], da Symbian. Existe ainda uma versão de Linux para PDA's [21], porém sua aceitação ainda é muito tímida.

2.2.1 Aplicações nativas

O desenvolvimento de aplicações nativas tanto para dispositivos móveis quanto para computadores pessoais possuem as mesmas dificuldades: a variedade de ambientes de desenvolvimento e a incompatilidade entre eles e entre as plataformas. Somente o PalmOS, sistema operacional com penetração de mais 72% do mercado de *handhelds* no final de 2000 [10], possui mais de 15 ambientes de desenvolvimento reconhecidos pela Palm Inc.

PalmOS

As ferramentas atuais de desenvolvimento para a plataforma Palm são baseadas em extensões da linguagem C (como o Codewarrior [22], a ferramenta indicada pela Palm para desenvolvimento em sua plataforma de *handhelds*) ou Basic (por exemplo AppForge [23] e NSBasic [24]), ou sub-conjuntos de Java (Waba [25]). Estas ferramentas oferecem grande controle sobre a aplicação, porém sua curva de aprendizado tende a ser acentuada. Além disso, normalmente a compatibilidade com outras plataformas é muito pobre. Outras ferramentas fornecem mecanismos que procuram libertar o desenvolvedor de qualquer linguagem de programação, baseando todo o desenvolvimento em manipulação direta de objetos e propriedades (como os populares *Satellite Forms* [26] e *PDA Toolbox* [27]). Embora o desenvolvimento com estas ferramentas e o aprendizado das mesmas sejam rápidos, as aplicações normalmente são bastante limitadas, e em geral requerem um *run-time engine* próprio.

Através do *HotSync Manager* [28], um PDA pode sincronizar e trocar dados com um computador pessoal, além de efetuar e restaurar cópias de segurança (*backups*) e instalar novas aplicações. Para isso são utilizados *conduits*, instalados como *plugins*. A comunicação pode ser feita através de uma conexão serial, infra-vermelho, modem ou rede.

Windows CE

A plataforma Windows CE foi desenvolvida para suportar sistemas embutidos em geral, não só PDA's. Por isso, possui o *Platform Builder* [29], um sistema para juntar todos os componentes necessários ao funcionamento de uma determinado dispositivo sendo desenvolvido. O desenvolvimento de aplicações conta com o *eMbedded Visual Tools* [30], derivado do *Visual Studio* [31], com versões dos ambientes integrados do Visual Basic e Visual C++ otimizados para aplicações embutidas. Utiliza o modelo de programação popularizado pelo Windows (Win32).

A conexão entre um dispositivo com Windows CE e um computador pessoal é feita utilizando a tecnologia ActiveSync [32] utilizando uma conexão serial, infra-vermelho, modem, USB ou Ethernet. Provê sincronização e conversão de arquivos de dados entre o PDA e computador pessoal.

Symbian OS/EPOC

A plataforma EPOC é a única que possui suporte (do próprio desenvolvedor da plataforma) a Java, além de C++ e OPL [33] (uma linguagem criada pela Symbian, baseada no Basic). Disponibiliza SDK's gratuitamente, que podem ser utilizados isoladamente ou em conjunto com outros ambientes de desenvolvimento (Microsoft Visual C++, por exemplo).

A conectividade externa é feita utilizando o *Symbian Connect*, que permite a sincronização de dados com um computador pessoal, e ainda que, através deste, possa-se gerenciar arquivos, imprimir e instalar aplicações.

2.2.2 HTML

Embora o poder computacional dos PDA's sejam reduzidos quando comparados aos computadores pessoais, a disponibilidade de uma tela de tamanho razoável (colorida em alguns modelos) e capacidade de processamento razoável permite a criação de aplicações utilizando subconjuntos ou adaptações de HTML. Obviamente, muitos dos recursos de HTML disponíveis nos computadores pessoais não podem ser facilmente utilizados nos PDA's, como por exemplo *frames*, *applets* e *scripts*.

Um exemplo é a arquitetura Web Clipping Application (WCA [34]) da Palm, anteriormente conhecida como PQA. Uma WCA é algo como um mini *web site* instalado no Palm. Uma aplicação típica contém um formulário HTML ou uma lista de hiperligações, os quais requisitam informações adicionais local ou remotamente. Existe também a possibilidade de executar outras aplicações do PalmOS a partir de uma WCA. Nesta arquitetura, uma servidor *proxy* é responsável por receber as requisições da aplicação, contactar o servidor real, receber e compactar o conteúdo solicitado, e, finalmente, retorná-lo para o dispositivo que originou a requisição.

As aplicações na arquitetura WCA utilizam um subconjunto da especificação HTML 3.2. Uma vez prontas as páginas e gráficos HTML, estes são convertidos (compilados e compactados) em uma WCA utilizando o *Web Clipping Application Builder*, uma ferramenta disponibilizada gratuitamente pela Palm. A aplicação gerada é então exibida no Palm através de um visualizador, *Web Clipping Application Viewer*. A figura 2.1 mostra um exemplo de aplicação desta arquitetura.

Figura 2.1: Uma *Web Clipping Application*

2.3 Novas tecnologias

Existem pelo menos três tecnologias que poderão tornar bem mais ágil o desenvolvimento de aplicações para dispositivos móveis: *Java 2 Micro Edition* [35], BREW [36] e XHTML [37]. As três foram desenvolvidas visando a compatibilidade com uma variedade de dispositivos.

2.3.1 Java 2 Micro Edition (J2ME)

Espera-se que, em breve, dispositivos como celulares e *pages* sejam muito mais personalizáveis do que hoje. Ao invés de virem com um conjunto de instruções praticamente imutável, este novos dispositivos permitirão aos usuários incluir novos serviços ou aplicações (provavelmente direto da Internet) [38].

Para atender à esta demanda, a Sun estendeu a tecnologia Java com a introdução da plataforma J2ME. Esta plataforma foi desenvolvida para ser modular e escalável, de modo a atender a grande variedade de dispositivos e configurações existentes. Baseia-se em três camadas (figura 2.2):

- *Java Virtual Machine Layer*, uma implementação da máquina virtual Java adaptada para cada dispositivo.
- *Configuration Layer*, que define as características mínimas de uma máquina virtual e bibliotecas Java disponíveis para uma categoria de dispositivos.
- *Profile Layer*, que define o conjunto mínimo de *Application Programming Interfaces* (API's) disponíveis para uma família particular de dispositivos.

A relação entre uma máquina virtual e uma configuração é extremamente próxima, capturando as capacidades essenciais de uma categoria de dispositivos. Maiores diferenciações são oferecidas através de API's adicionais incluídas em um perfil específico, podendo ser incrementadas com bibliotecas adicionais.

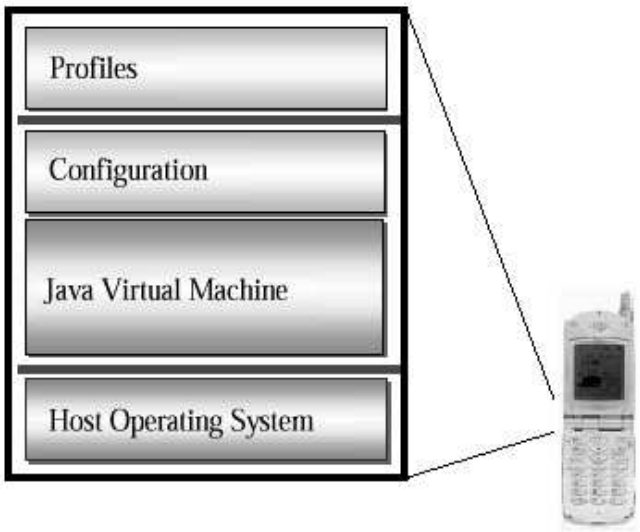

Figura 2.2: Camadas de *software* J2ME

Para dispositivos móveis e demais sistemas com limitações de recursos semelhantes, J2ME define a *Connected Limited Device Configuration* [40] (CLDC). Esta configuração é

composta pela KVM [41] (*K Virtual Machine*) e um conjunto de bibliotecas que poderão ser usados em dispositivos como celulares, *pages* e organizadores pessoais, por exemplo. Um perfil também foi definido para esta família de dispositivos, *Mobile Information Device Profile* [42] (MIDP), e a Sun já disponibilizada uma ferramenta para desenvolvimento específica, *J2ME Wireless Toolkit* [43].

A vantagem de utilizar-se a plataforma J2ME está na portabilidade do código, além das vantagens da programação orientada a objetos. Uma aplicação poderá ser desenvolvida para toda uma família de dispositivos que sejam compatíveis com KVM/CLDC/MIDP, o que deverá incluir desde celulares até PDA's. Atualmente está disponível para a plataforma Palm a CLDC [44], que já conta com bibliotecas adicionais de interface gráfica.

2.3.2 BREW

Criada e mantida pela Qualcomm, é a primeira plataforma para aplicações móveis que incorpora toda a solução de distribuição e modelo de negócio de aplicações para desenvolvedores, fabricantes de dispositivos e operadoras de redes de telecomunicações.

O *kit* de desenvolvimento é distribuído gratuitamente. Baseada em C/C++, a plataforma pode ser incorporada em qualquer tipo de dispositivo a um custo relativamente baixo. Apesar de ter sido notadamente criada para funcionar em celulares, oferecendo por exemplo bibliotecas para interação com envio de mensagens de texto e agenda de telefones, qualquer dispositivo móvel pode incorporar o ambiente de execução BREW.

Embora seja aberta e extensível, permitindo que outros ambientes sejam incluídos como a própria máquina virtual Java, toda a cadeia de valores de uma aplicação é centralizada na Qualcomm. Qualquer aplicação deve ser testada e certificada pela Qualcomm, envolvendo o respectivo pagamento por este serviço, para então ficar disponível em um catálogo. A distribuição destas aplicações, que envolve a publicação, *download over the air* e cobrança para clientes, e a divisão de ganhos entre desenvolvedores e operadoras, também é gerenciado por um sistema que compõe a plataforma.

Desta forma, embora o custo financeiro de desenvolvimento possa ser mais elevado em BREW, a plataforma oferece uma solução fim a fim para a distribuição de aplicações. Captura praticamente todas as necessidades do mercado e é capaz de oferecer mais segurança a todos os envolvidos:

- usuários, que poderão personalizar seus dispositivos através de um ponto central de distribuição. Além de facilitar a descoberta de aplicações, poderão contar com aplicações seguras e confiáveis;
- desenvolvedores de aplicações, que podem criar aplicações para um grande conjunto de dispositivos e oferecê-las para um conjunto de operadoras simultaneamente;
- fabricantes de dispositivos, que podem oferecer dispositivos mais personalizáveis e úteis, e com maior apelo junto aos consumidores;

- operadoras de telecomunicação, que poderão oferecer um catálogo crescente de aplicações, com toda a solução de cobrança e distribuição pronta.

2.3.3 XHTML

XHTML 1.0 [45] é uma especificação do W3C que define um tipo de documento rico o bastante para ser usado para criação de conteúdo e formatação de documentos, mantendo ainda a acessibilidade para diversas classes de dispositivos.

XHTML é baseado na especificação HTML 4.1, adicionando-lhe modularidade e aderência ao XML. A apresentação do documento é descrita através de folhas de estilo (*Cascading Style Sheets* - CSS), sem sacrificar a independência do navegador. E como um documento XML, permitirá a transformação de conteúdo através da aplicação de XSLT.

XHTML e CHTML são muito similares, exceto que CHTML não é um documento XML e não suporta CSS. Além disso, a especificação WAP 2.0 irá evoluir do WML para o XHTML *Basic* [46], uma adaptação do XHTML 1.0 apropriada para dispositivos com telas pequenas. Isto significa que, futuramente, XHTML será compatível com uma ampla variedade de dispositivos móveis, oferecendo grande flexibilidade para o desenvolvimento de aplicações para a Internet móvel e para diferentes clientes.

2.4 Considerações finais

Os fabricantes de celulares oferecem ferramentas de desenvolvimento específicas para seus aparelhos, dos quais podemos citar Nokia WAP Toolkit [47], Openwave SDK [48], Motorola Mobile Application Development Kit [49]. Todas estas ferramentas possuem um ambiente integrado para edição e depuração de documentos (WML ou HDML, dependendo da tecnologia suportada nos celulares do fabricante), com emuladores de vários aparelhos para visualização e teste das aplicações. São ferramentas excelentes quando o dispositivo-destino da aplicação é uma única família de celulares de um único fabricante.

Uma vantagem óbvia ao desenvolver aplicações baseadas em linguagens de marcação é a capacidade de aproveitar-se a infraestrutura de fornecimento de documentos (sendo a Internet, juntamente com servidores Web, a mais popular), bem como o conhecimento e ferramentas existentes em tecnologias de produção dinâmica de conteúdo no servidor, como CGI (*Common Gateway Interface*), ASP (*Active Server Pages*), PHP (*PHP Hyper-Processor*), JSP (*Java Server Pages*), *Cold Fusion*, e várias outras.

Entre as novas tecnologias, J2ME e BREW deverão ser os ambientes utilizados para oferecer aplicações com mais recursos, tanto de processamento quanto de interatividade, para dispositivos de maior capacidade. Já existem dispositivos com estas plataformas e várias aplicações podem ser encontradas.

Pode-se perceber que utilizando as tecnologias disponíveis atualmente, uma determinada aplicação desenvolvida para um dispositivo em uma plataforma utilizando uma dada

ferramenta, em geral, somente poderá ser portada para um outro ambiente através de um processo custoso. Pouco do trabalho já realizado poderá ser aproveitado na nova plataforma. Uma ferramenta capaz de facilitar o desenvolvimento de aplicações para esta gama de dispositivos, que não ignore os futuros avanços (ou tendências) tecnológicos, e que possa atender às várias formas de aplicações será extremamente útil para integradores e desenvolvedores.

Capítulo 3

Trabalhos Relacionados

A complexidade para o desenvolvimento de aplicações para dispositivos móveis é bastante clara para qualquer desenvolvedor. Exatamente por isso, ferramentas que procuram criar aplicações que possam ser utilizadas em qualquer dispositivo já estão disponíveis no mercado.

Muitos trabalhos foram desenvolvidos baseando a criação de aplicações em algum formato de documento XML, que então era transformado para um outro formato mais adaptado ao dispositivo [50]. Esta é a base teórica da grande maioria das ferramentas disponíveis atualmente.

3.1 Speedware MobiledDev

O *Mobiledev* [51] é um ambiente de desenvolvimento para WAP (WML e WMLScript) e HDML, através da geração de *scripts* em ASP ou Perl, permitindo acesso a dados através de conexões ODBC. Aplicações WAP podem ser testadas facilmente através do simulador da Yospace [53] (baseado no aparelho Nokia 7160) e do servidor Web Xitami [52] (onde os *scripts* gerados são executados), ambos integrados ao pacote. Outros simuladores podem ser incluídos (como o simulador da Openwave [48], para testes de aplicações em HDML), assim como outros servidores Web podem ser utilizados (Apache [54], por exemplo).

Esta ferramenta possui uma interface baseada em assistentes e manipulação gráfica de objetos e folhas de propriedades para a criação de aplicações (figura 3.1). Além disso, possui um editor de código com ajuda sensível ao contexto, capaz de exibir propriedades de marcações. Possui ainda suporte para colaboração, através de um mecanismo próprio de bloqueio ou usando o *SourceSafe* da Microsoft [55].

MobileDev conta com bons recursos para a criação de aplicações, podendo ser considerada uma ferramenta bastante completa e fácil de usar. Porém é voltada unicamente ao desenvolvimento para celulares (suporte a WML e HDML apenas), e o acesso a dados fica limitado a conexões ODBC.

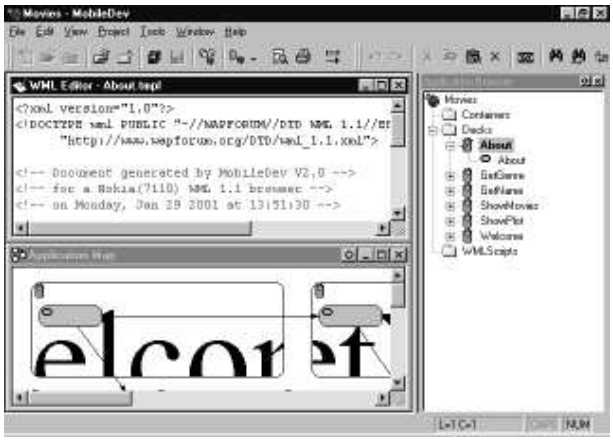

Figura 3.1: MobileDev 2.0

3.2 Oracle9iAS Mobile Studio

Esta ferramenta é disponibilizada gratuitamente pela Oracle sob a forma de um *site* [56]. É um ambiente completo para construção, teste e entrega de aplicações móveis. A aplicação é desenvolvida utilizando o MobileXML, um conjunto de marcações desenvolvido para serem independente de dispositivo. A figura 3.2 mostra um pequeno exemplo de MobileXML usando a costumeira aplicação "Hello World".

```
<?xml version="1.0" encoding="UTF-8"?> <SimpleContainer>
  <SimpleText>
    <SimpleTextItem>Hello World</SimpleTextItem>
    <Action label="Reload" type="SOFT1" task="CALL" target="helloworld.xml">
    </Action>
  </SimpleText>
</SimpleContainer>
```

Figura 3.2: "Hello World", em MobileXML

A geração de conteúdo em MobileXML pode ser feita utilizando qualquer tecnologia de geração dinâmica de conteúdo, como CGI, JSP ou ASP. Suporta qualquer dispositivo capaz de acessar a Internet, compatíveis com um dos formatos CHTML, HTML, WML e HDML.

É uma ferramenta bastante interessante por gerar automaticamente conteúdo adaptado ao dispositivo cliente, desde que este suporte um dos formatos disponíveis. Ao usar uma descrição em XML da aplicação, permite aproveitar toda a experiência pré-existente de criação de conteúdo dinâmico para a Web.

3.3 AVIDRapidTools

Esta ferramenta [57] é inteiramente desenvolvida em Java. Seu núcleo é um pacote de classes que encapsulam a criação de aplicações e a adaptação do conteúdo para os vários dispositivos. Suporta os dispositivos e linguagens WAP/WML, *i-Mode/Microsoft Mobile Explorer/Compact HTML*, Palm VII/PQA e HTML.

Fazem parte do pacote um servidor Web compatível com Servlets e JSP (*Jo! Java Server Engine* [58]), um SGBD (*HyperSonic SQL* [59]), ambos gratuitos e escritos em Java, e um simulador WML (YoSpace). O acesso a dados é feito utilizando JDBC (*Java Database Connectivity*).

Nenhuma interface é fornecida com a ferramenta. A idéia desta ferramenta é que um desenvolvedor Java experiente possa utilizar seu ambiente de desenvolvimento preferido e criar aplicações sem ter nenhum conhecimento de dispositivos ou marcações específicas. É composto por classes para acesso a banco de dados, gerenciamento de configuração, utilitários e gerenciamento de telas e de exibição. O processo de navegação entre telas é definido através de um arquivo de texto que define o fluxo de controle da aplicação (semelhante à uma máquina de estados). Este arquivo, chamado *ScreenFlowFile*, é baseado na metáfora *Screen, Mode, Result*, onde a próxima janela e modo a serem exibidos são baseados na janela, modo e resultado atuais.

Esta ferramenta é particularmente interessante devido à utilização do *ScreenFlowFile*. Sua estruturação, semelhante à uma máquina de estados finitos, pode permitir que técnicas de verificação formal sejam aplicadas ao desenvolvimento de aplicações.

3.4 Microsoft Mobile Internet Toolkit

Integrado à recém lançada plataforma .NET, este *toolkit* permite a criação de aplicações capazes de enviar conteúdo na linguagem de marcação correta para uma grande quantidade de dispositivos móveis [60].

É composto por um conjunto de controles (*Mobile Web Forms Controls*), capazes de gerar elementos de interface adequados ao dispositivo; um ambiente integrado de desenvolvimento (*Mobile Internet Designer*) integrado ao Visual Studio .NET; e um conjunto de definições de capacidades de dispositivos (indicando por exemplo tipo de linguagem de marcação de cada dispositivo, o tamanho da tela, número de linhas, etc).

Pode ainda ser extendido, através da inclusão de novas definições de capacidades de aparelhos e da criação de novos controles a partir dos controles existentes.

3.5 HiddenMind Mobility Platform

Plataforma de desenvolvimento de aplicações móveis para dispositivos com suporte a voz ou linguagens de marcação [61], baseada em Java. Para isto uma ferramenta, *Active Studio*, permite a criação visual de documentos em XML independente de dispositivo.

Esta ferramenta permite que os dados sejam retirados de fontes como XML, HTML, bancos de dados, POP3, IMAP e LDAP, e ainda que adaptadores para outras fontes sejam criados. Definida a aplicação, um *engine* transforma sua representação em XML para os formatos específicos dos dispositivos, como WML, HTML, VoiceXML e Palm Clippings. A figura 3.3 ilustra esta plataforma.

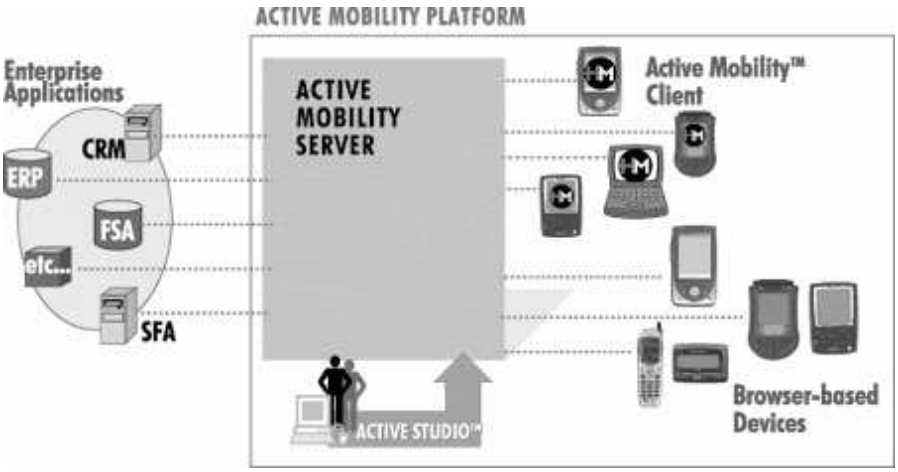

Figura 3.3: HiddenMind Mobility Platform

Possui ainda recursos de personalização onde, através de uma interface *Web*, pode-se configurar e gerenciar informação sobre os dispositivos clientes, bem como qual conteúdo cada um deverá receber. Além disso permite que alertas e notificações sejam agendadas e enviadas de acordo com cada perfil.

3.6 Everypath Studio

É um ambiente de programação desenhado para facilitar o desenvolvimento de aplicações móveis [62]. Possui um ambiente gráfico capaz de selecionar dados de várias fontes de dados, usando uma metodologia de apontar-e-clicar, para criar aplicações e estruturar o fluxo de trabalho. Este ambiente possui ainda a habilidade de testar e depurar aplicações usando simuladores de dispositivos.

Pode ser integrada as plataformas proprietárias para gerenciamento e personalização (Everypath Administration Console), extração avançada de dados (Everypath Connectors), instalação de clientes inteligentes nos dispostivos (Everypath Smart Client) e sincronização (Everypath Server).

3.7 Considerações finais

As duas últimas ferramentas citadas, HiddenMind Mobility Platform e Everypath Studio estão disponíveis apenas comercialmente. Aparentemente oferecem plataformas de desenvolvimento bastante completas, oferecendo acesso a várias fontes de dados e suportando vários dispositivos. Porém, infelizmente, as empresas que as desenvolveram não disponibilizam nenhuma forma de demonstração, nem oferecem informações mais detalhadas dos produtos.

Um fato importante é que todas as ferramentas citadas baseiam-se inteiramente na geração de aplicações baseadas em linguagens de marcação. Nenhuma propõe-se a gerar código nativo ou intermediário (que executa sobre algum tipo de *run-time*).

Aplicações baseadas em uma linguagem de marcação oferecem as vantagens de serem facilmente atualizáveis e manipuláveis, podendo gerar de modo simples outras marcações. Porém é necessário que o dispositivo esteja sempre conectado à uma rede de dados para poder acessar estas aplicações. Muitas vezes este acesso não está disponível, seja por problemas de interferência ou sinal fraco, ou mesmo a simples ausência de infra-estrutura. Nestes casos faz-se imprescindível a geração de aplicações nativas, capazes de processar dados e, posteriormente, sincronizá-los com um sistema computacional central. Esta é uma lacuna que a ferramenta proposta neste trabalho procurará cobrir.

Capítulo 4

Análise de Requisitos

Dado o panorama do desenvolvimento de aplicações para dispositivos móveis, os principais requisitos para uma ferramenta que pudesse ser útil a projetistas e desenvolvedores seriam:

1. Capacidade de descrever aplicações independente do dispositivo móvel que será utilizado, de modo que a mesma descrição possa ser utilizada em qualquer dispositivo. Esta descrição deverá gerar uma aplicação nativa (da plataforma de desenvolvimento de cada dispositivo) ou então uma aplicação intermediária (que executará sobre um *engine*, uma máquina virtual ou um protocolo de aplicação). Além disto, a descrição deverá ser modular o suficiente para que aspectos específicos da interface de um dispositivo possam ser definidos separadamente, para cada dispositivo suportado.

   Descrever aplicações de modo independente do dispositivo não é uma novidade. Vários estudos e trabalhos já foram feitos buscando alcançar este objetivo.

   A maior parte das propostas são baseadas em linguagens de marcação, com destaque para XML (WML, HDML, XHTML, e todas as demais linguagens de marcação citadas nos capítulos 2 e 3). Uma vantagem clara da utilização de XML é a possibilidade de gerar diretamente outros formatos de documentos através de XSL.

   Existem ainda propostas que utilizam interfaces de programação com esta finalidade. Nesta categoria encontramos desde conjuntos de ferramentas de *software* (como por exemplo o pacote *AVIDRapidTools*) até especificações inteiras de plataformas independentes de dispositivos (especificação J2ME). Estas soluções oferecem grande flexibilidade no desenvolvimento de aplicações. Porém esbarram no problema da portabilidade destas interfaces para os diversos dispositivos.

2. A descrição também deverá permitir a marcação de pontos de processamento da aplicação, onde código externo deverá ser inserido pelo projetista (usuário da ferramenta). Estas marcações podem ser parte da aplicação como um todo (isto é, será a mesma para todos os dispositivos) ou podem ser específicas para determinado dispositivo.

Uma ferramenta fechada em si mesma terá pouca utilidade, pois sua aplicabilidade será restrita. É muito importante permitir que o projetista possa inserir código próprio. Isto dará mais versatilidade à ferramenta, permitindo que o projetista possa aplicá-la em várias situações.

Esta propriedade é encontrada em várias tipos de aplicações (em toda a área de computação, não só a móvel) sob a forma de *plugins* instaláveis ou similares. É uma maneira bastante conveniente de estender o escopo de aplicações em geral. Para o caso da aplicações baseadas em linguagens de marcação, têm sido utilizados *scripts*. Dependendo da situação, estes *scripts* podem executar no próprio cliente ou no ambiente do servidor.

3. O código externo poderá ser inserido utilizando uma linguagem particular ou através de tecnologias de componentes de *software* (objetos). No caso de uma linguagem, deverá prover ao projetista um método o mais uniforme possível de incluir seu código, seja para dispositivos móveis ou para o sistema central fixo. Uma linguagem já existente, adaptada à aplicação, poderia ser utilizada.

   Baseadas nos conceitos de definições de interfaces e encapsulamento, as tecnologias de componentes permitem que o desenvolvedor possa usar seu ambiente de desenvolvimento preferido para escrever seu código. Oferecem uma arquitetura muito conveniente para aplicações distribuídas e multi-camadas. CORBA, COM e JavaBeans são os principais representantes destas tecnologias. Porém estas tecnologias ainda não podem ser aplicadas diretamente no desenvolvimento de aplicações para dispositivos móveis. Os mecanismos necessários à invocação destes objetos ainda é muito complexo para fazerem parte da maior parte das plataformas. Entretanto, o Windows CE, a partir de sua versão 3.0, incorpora suporte à COM e mais recentemente foi lançada a especificação do padrão CORBA para terminais móveis [70]. Este é basicamente uma adaptação do padrão de modo a utilizar a pilha WAP para realizar a comunicação entre objetos, em vez da pilha TCP/IP, além de uma arquitetura para dar suporte otimizado em redes sem fio. Independente da complexidade inerente destas tecnologias, as especificações de definição de interfaces e técnicas de programação por contrato utilizadas por estas tecnologias podem ser bastante úteis.

   Pode ainda ser utilizada uma linguagem intermediária, a partir da qual seria gerado código adequado à plataforma. A princípio poderia ser usado um subconjunto de qualquer linguagem de programação.

   Uma outra opção, que no futuro poderá ser bastante promissora, será utilizar a linguagem Java. Sua característica de portabilidade aliada ao lançamento da plataforma J2ME permitirão que o desenvolvedor utilize a mesma linguagem tanto para os dispositivos móveis quanto para os sistemas fixos.

4. Deve possuir facilidades para a criação de mecanismos de sincronização, como por exemplo a geração de *stubs*, para o caso de aplicações *off-line*. Tais facilidades tam-

bém deverão existir para a criação da aplicação correspondente no sistema central fixo.

Este aspecto de sincronização tem sido ignorado pela maior parte das ferramentas disponíveis atualmente. Apenas aquelas voltadas exclusivamente para PDA's possuem, em alguns casos, tais facilidades. Mesmo que no futuro exista uma alta disponibilidade de redes de comunicação sem fio, é importante não negligenciar a disponibilização de mecanismos de sincronização de aplicações *off-line*, principalmente por questões de compatibilidade com dispositivos que não disponham deste acesso *on-line* (que em geral são mais baratos).

5. Os protocolos de comunicação mais populares devem estar disponíveis para a criação de aplicações *on-line*, como as pilhas TCP/IP e WAP. Mecanismos que permitam incorporar outros protocolos de comunicação são altamente desejáveis.

   TCP/IP e WAP são importantes padrões mundiais no que refere-se a protocolos de comunicação de dados. Não é possível imaginar uma ferramenta que não seja capaz de dar suporte a eles. Por outro lado, é importante que seja possível a inclusão de suporte a novos protocolos.

   Em breve deverão estar disponíveis equipamentos suportando *Bluetooth*[1] e protocolos para telefonia celular 3G[2], que provavelmente farão surgir novas aplicações as quais, em conseqüência, demandarão ferramentas que suportem estes protocolos.

6. Deve ser possível adquirir dados e conteúdo de fontes diversas como XML, bancos de dados e protocolos de aplicação da Internet. Mecanismos que permitam a inclusão de outras fontes são altamente desejáveis.

   Informações podem estar armazenadas sob diversas formas, e a grande promessa da computação móvel é a possibilidade de acessar estas informações a qualquer momento, de qualquer lugar. Daí a importância de não limitar o acesso a dados e, mais ainda, de oferecer meios para incorporar à ferramenta o acesso a novas fontes de dados.

   Uma forma cada vez mais aceita para troca de dados é a utilização de XML [73]. Através da geração de DTD's contendo a descrição dos dados, é possível obter um documento auto-contido para intercâmbio de informações. Já existem inclusive propostas de linguagens de consulta a dados contidos em documentos XML [63].

7. Ser extensível, possibilitando que novas interfaces, novas aplicações e novos dispositivos sejam incorporados facilmente.

   A área de computação móvel é muito ativa. Muito ainda está sendo desenvolvido e poucos padrões existem realmente. Seria muito interessante a possibilidade de

[1] Especificação de enlaces sem fio de baixo consumo de energia, voltada para interconexão de equipamentos

[2] Terceira geração de celulares, que suportará taxas de transferência capazes de atender à aplicações que necessitem de bandas mais largas (ex.: multimídia)

acrescentar à ferramenta novas opções de desenvolvimento, sem ter que reescrevê-la novamente. Aqui também a capacidade de instalação de *plugins* e tecnologias de componentes de *software* aparecem como opções bastante apropriadas.

8. Incluir facilidades de verificação da aplicação sendo gerada. Por exemplo, um modelo como um grafo ou uma máquina de estados finitos poderia ser gerado. Desta forma o projetista poderá avaliar o comportamento desta aplicação e suas propriedades. A própria ferramenta poderá realizar tal verificação, ou exportar em uma representação capaz de ser analisada por outra, desenvolvida por terceiros.

   Esta facilidade será muito importante para projetos grandes ou com requisitos rígidos quanto à correção do fluxo, onde vários estados podem ser atingidos dentro da aplicação. Um exemplo seria uma aplicação bancária. Através de mecanismos de verificação, o projetista poderá atestar a correção do projeto ou, em caso contrário, descobrir erros.

9. Possuir facilidades para depuração de aplicações. Recursos como emuladores, depurador passo-a-passo e visualizador de estado da aplicação seriam muito úteis, juntamente com o item anterior.

   Qualquer projeto deve passar por testes de aceitação, para os quais devem existir recursos que auxiliem o projetista. Ao permitir a inclusão de pontos de verificação (*checkpoints*) ou parada (*breakpoints*), onde o estado da aplicação possa ser visto ou analisado, a ferramenta possibilita a redução do tempo de desenvolvimento. Estes recursos estão presentes na grande maioria dos ambientes de desenvolvimento integrado de várias plataformas, principalmente as fixas.

   No caso de dispositivos móveis, emuladores mostram-se bastante úteis. Permitem que testes sejam executados localmente, ainda no ambiente de desenvolvimento, além de serem capazes de oferecer recursos extras (não presentes nos dispositivos reais) que facilitem a depuração.

   A geração de históricos (*logs*) de operação também contribuem para a depuração de aplicações, além de aumentar a confiabilidade destas ao permitir a monitoração de atividades.

10. Permitir recursos de internacionalização, ou seja, oferecer meios para que aplicações possam ser entregues em outros idiomas sem grande esforço.

    Uma vez que no mundo móvel as informações podem ser acessadas de qualquer lugar, passa a ser importante oferecer recursos de internacionalização. A detecção automática de idiomas preferenciais é suportada na Internet, por exemplo, mas um recurso que permita escolher o idioma (entre alguns previamente disponibilizados e suportados) já seria satisfatório.

11. Curva de aprendizado aceitável: a ferramenta não deve exigir esforços exagerados para sua utilização.

Este é um requisito ligado à usabilidade. A interface da ferramenta deve ser intuitiva o suficiente para que o projetista possa usá-la facilmente, promovendo a retenção do conhecimento associado à sua utilização.

Estes requisitos são bastante genéricos, de modo a atender amplamente às demandas de desenvolvedores de aplicações para dispositivos móveis.

A tabela 4.1 mostra um resumo das características das ferramentas descritas no capítulo anterior, relacionando-as com os requisitos apresentados neste capítulo.

Pode-se observar que nenhuma das ferramentas consegue implementar todos os requisitos listados. Entretanto a maioria possui meios de descrever uma aplicação de modo independente da plataforma de destino, seja através de objetos de programação (como AVID RapidTools e Mobile Internet Toolkit) ou de documentos XML (Oracle Mobile Studio, HiddenMind Mobility Platform e Everypath Studio). Da mesma forma, a capacidade de incluir lógica externa é uma constante, caso contrário as ferramentas teriam uma utilização muito restrita.

Já a capacidade de estender-se para atender novos dispositivos ou aplicações, sem necessidade de desenvolver-se uma nova versão da ferramenta, é um requisito que ainda não recebeu muita atenção: apenas AVID RapidTools e Mobile Internet Toolkit possuem esta facilidade. Recursos para internacionalização, que é também uma forma de extensibilidade, são ainda menos encontrados (apenas AVID RapidTools). A verificação formal de aplicações ainda não está disponível em nenhuma das ferramentas.

Finalmente, cabe fazer duas considerações sobre este conjunto de requisitos. Primeiro, a implementação de apenas parte deste conjunto poderá ainda resultar em uma ferramenta bastante útil para determinados cenários de desenvolvimento. Segundo, mesmo que haja uma implementação de todos estes requisitos, ainda assim poderão existir aplicações que não poderão ser desenvolvidas usando tal ferramenta, provavelmente devido a singularidades de dispositivos, ambientes ou mesmo da própria aplicação.

Requi-sitos	MobileDev	Oracle9iAS Mobile Studio	AVID RapidTools	MS Mobile Internet Toolkit	HiddenMind Mobility Plat.	Everypath Studio
1	○	●	●	●	●	●
2	±	●	●	●	?	?
3	●	●	●	●	?	?
4	○	○	○	○	?	●
5	WAP	WAP TCP-IP	WAP TCP-IP	WAP TCP-IP	●	●
6	±	●	●	●	●	●
7	○	○	●	±	?	?
8	○	○	○	○	○	○
9	±	±	±	●	?	●
10	○	○	●	○	?	?
11	●	±	±	●	±	±

Legenda: ○ - não possui ± - aproxima-se ? - não se sabe ● - possui

1. Descrição independente de dispositivos
2. Inserção de código externo
3. Flexibilidade na inserção de código externo
4. Facilidades de sincronização
5. Disponibilidade de protocolos de comunicação
6. Facilidades na aquisição de dados de fontes diversas
7. Extensibilidade
8. Facilidades para verificação formal de aplicações
9. Facilidades para depuração de aplicações
10. Recursos para internacionalização
11. Facilidade de aprendizado e utilização

Tabela 4.1: Comparação das principais características das ferramentas existentes

Capítulo 5

Implementação da ferramenta

5.1 Decisões de projeto

A análise de requisitos feita no capítulo anterior foi feita de modo a atender amplamente às questões relacionadas ao desenvolvimento de aplicações para dispositivos móveis.

Uma ferramenta capaz de atender a todos os requisitos especificados demandaria recursos acima daqueles disponíveis dentro do escopo deste trabalho. Desta forma, somente um subconjunto destes requisitos serão selecionados e implementados nesta ferramenta, batizada como MAB (*Mobile Application Builder*). Apenas os requisitos relativos à verificação, depuração e internacionalização (respectivamente itens 8, 9 e 10) não serão implementados, mas foram considerados durante a etapa de projeto.

Outra decisão de implementação é a limitação do número de dispositivos de destino. A ferramenta implementada irá apenas gerar aplicações para as plataformas PalmOS utilizando CLDC e WAP 1.1 . Desta forma, foi coberto o desenvolvimento baseado em uma linguagem de marcação (WML 1.1) e na geração de código intermediário baseado em um *run-time* (KVM). Com isso consegue-se ainda cobrir as duas classes de dispositivos-alvo deste trabalho, os celulares e os PDA's, e atingir uma quantidade considerável dos aparelhos disponíveis atualmente.

Quanto às fontes de dados, serão suportadas apenas a aquisição de dados baseada em XML. Qualquer outra fonte deverá ser inserida através da transformação dos dados em um documento XML. A linguagem XML oferece uma interface flexível o bastante para permitir que qualquer fonte de dados seja importada para a ferramenta. Nesta implementação será utilizada a importação a partir de bancos de dados relacionais.

Para manter a regularidade e facilidade de desenvolvimento para o projetista (usuário da ferramenta), o código gerado pela ferramenta será Java. Além das vantagens de utilizar-se uma linguagem orientada a objetos, altamente portável e reusável, mantém-se a consistência ao utilizar a mesma linguagem para as aplicações em PDA's, baseado na J2ME, e celulares, onde poder-se-á utilizar *Java Servlets* [64] (capaz de gerar conteúdo em qualquer

linguagem de marcação).

Códigos externos serão definidos através de uma classe abstrata Java, a qual possuirá a assinatura dos métodos definidos pelo projetista que ficará responsável por sua implementação.

Para a incorporação e importação de dados para as aplicações será utilizado XML. Usando-se esse formato para a leitura de dados, adaptadores para praticamente qualquer fonte de dados poderão ser criados e incorporados à ferramenta. A aplicação poderá acessar estes dados através de requisições HTTP, por exemplo, que então retornarão XML da fonte desejada. Estas requisições poderão ainda serem baseadas no envio de parâmetros ou utilizar SOAP[1] (*Simple Object Access Protocol* [65]). A vantagem de se permitir a utilização de SOAP via HTTP é que, desta forma, o projetista terá liberdade para utilizar outra linguagem de programação em sua implementação. Independente do modo que os dados serão buscados, o formato do documento XML retornado será bem definido *a priori*.

5.2 Arquitetura

Uma vez que linguagens de marcação e a arquitetura cliente/servidor são (e provavelmente continuarão sendo) utilizadas na criação de aplicações para dispositivos móveis, torna-se bastante evidente que um servidor *Web* possa ser usado como base da ferramenta. Esta arquitetura tem sido usada largamente na Internet, tanto fixa quanto móvel, e é capaz de atender perfeitamente aos requisitos propostos. A arquitetura proposta tem a vantagem de, potencialmente, ser escalável e portável, uma vez que é baseada inteiramente no modelo cliente/servidor e no protocolo HTTP, cuja escalabilidade horizontal através de *web farms* está bastante madura atualmente [66, 67].

A figura 5.1 ilustra a arquitetura da ferramenta.

A ferramenta é baseada em uma interface de manipulação, onde o usuário define o fluxo de aplicação através de recursos de arrastar-e-soltar e folhas de propriedades. Foram definidas três unidades básicas de interação com o usuário, com as quais o projetista irá desenvolver uma aplicação:

- Listas: utilizadas para exibir um menu de comandos ou opções, geralmente para fins de navegação.
- Textos: utilizados para exibir informações para o usuário.
- Formulários: utilizados para receber entrada de dados, que pode consistir de combinações de itens como caixas de texto, campos de data ou hora, listas de seleção de opções e outros.

[1]SOAP é um protocolo baseado em XML para intercâmbio de informações através de chamadas de procedimentos remotos (*remote procedure calls*) em ambientes distribuídos. Pode ser usado em combinação com uma variedade de protocolos, mas a especificação mais madura o utiliza em conjunto com HTTP, o que será bastante conveniente dada a arquitetura proposta neste trabalho.

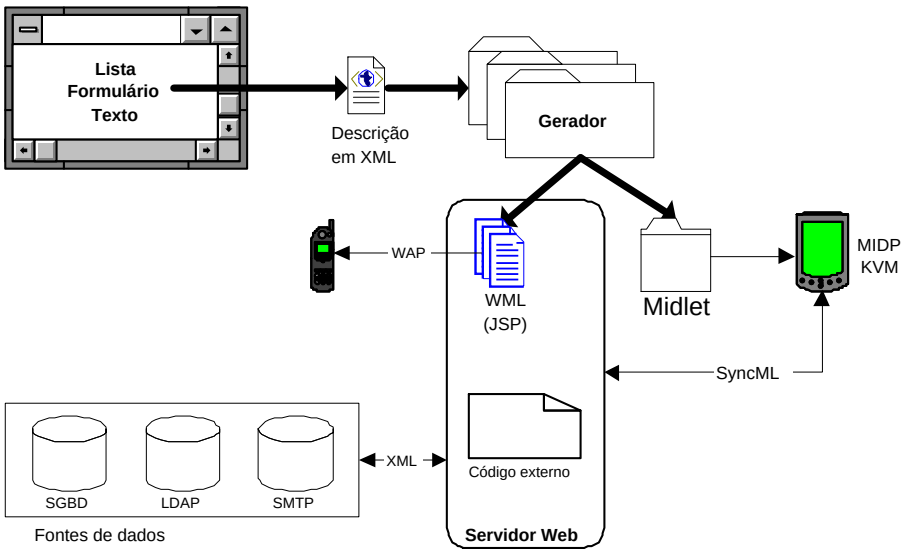

Figura 5.1: Arquitetura proposta

Estas três unidades de interação são defendidas em [39] e também utilizadas na ferramenta MobileDev. Seguem o conceito de usabilidade de que as interações com os usuários devem ser simples. Isto torna-se mais relevante ao se considerar que as telas dos dispositivos têm tamanho reduzido e, em geral, possuem poucos recursos. Assim, apenas uma tarefa deve ser exibida por vez, de modo a tornar a navegação óbvia e limpa.

5.2.1 Lista

Possui uma lista de itens disponíveis, geralmente para uma navegação ou em um menu de opções. Cada item consiste de uma descrição curta, a ação seguinte (próxima unidade a ser exibida), e, opcionalmente, alguns parâmetros (figura 5.2).

Os itens da lista podem ser estáticos ou dinâmicos. Os itens de uma lista gerada dinamicamente serão recuperados através de uma chamada a uma rotina externa. Esta rotina deverá conter quatro parâmetros de retorno obrigatórios:

- `descriptions` - vetor que conterá a descrição que será exibida ao usuário de cada item da lista. Os elementos do vetor serão separados pelo caracter `|`.
- `actions` - vetor que conterá o nome da unidade que deverá ser exibida para cada item da lista. Os elementos do vetor serão separados pelo caracter `|`.
- `parameters` - vetor que conterá a lista de parâmetros que será passada para cada item da lista. Esta lista tem a forma `nome=valor`, onde cada parâmetro é separado por ponto-e-vírgula. Os elementos do vetor serão separados pelo caracter `|`.

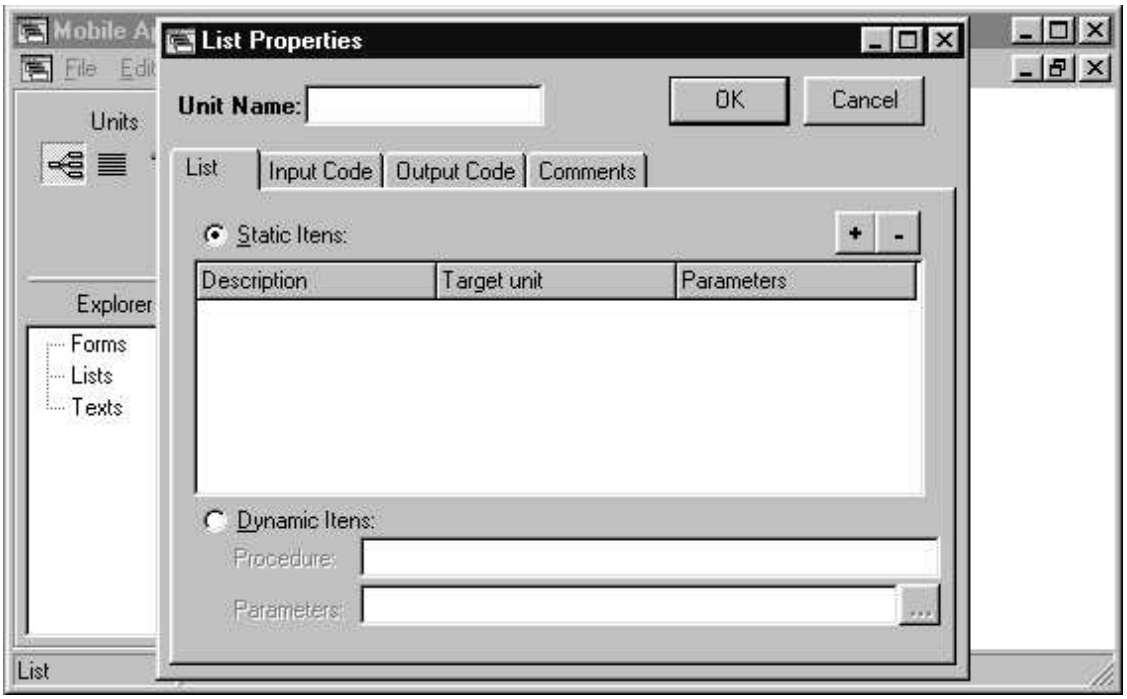

Figura 5.2: Propriedades para uma lista

Cada elemento destes vetores é do tipo `string`. O número de elementos destes três vetores deve sempre ser igual. No caso de uma ação não necessitar de parâmetros, sua posição no vetor deverá possuir um caracter de espaço.

5.2.2 Texto

Pode ser definido de forma estática utilizando substituição de parâmetros, ou de forma dinâmica (figura 5.3).

Cada parâmetro recebido pela unidade ou retornado por chamada à código externo poderá ser exibido. Para isto, cada parâmetro deverá ser informado usando a sintaxe `$nome`. O valor do parâmetro indicado será substituído por seu respectivo valor.

Por exemplo, suponha que o texto da unidade seja "Bem vindo, `$usuario`", e que esta unidade tenha recebido como parâmetro da unidade que a precedeu (talvez uma unidade para autenticação) o parâmetro `usuario=Minelli`. O texto final desta unidade seria "Bem vindo, Minelli".

Para o caso de texto dinâmico deverá ser chamada uma rotina externa, que retornará o texto a ser exibido por este unidade. Esta rotina deverá conter um parâmetro de retorno obrigatório chamado `text`, o qual conterá o texto final que será exibido.

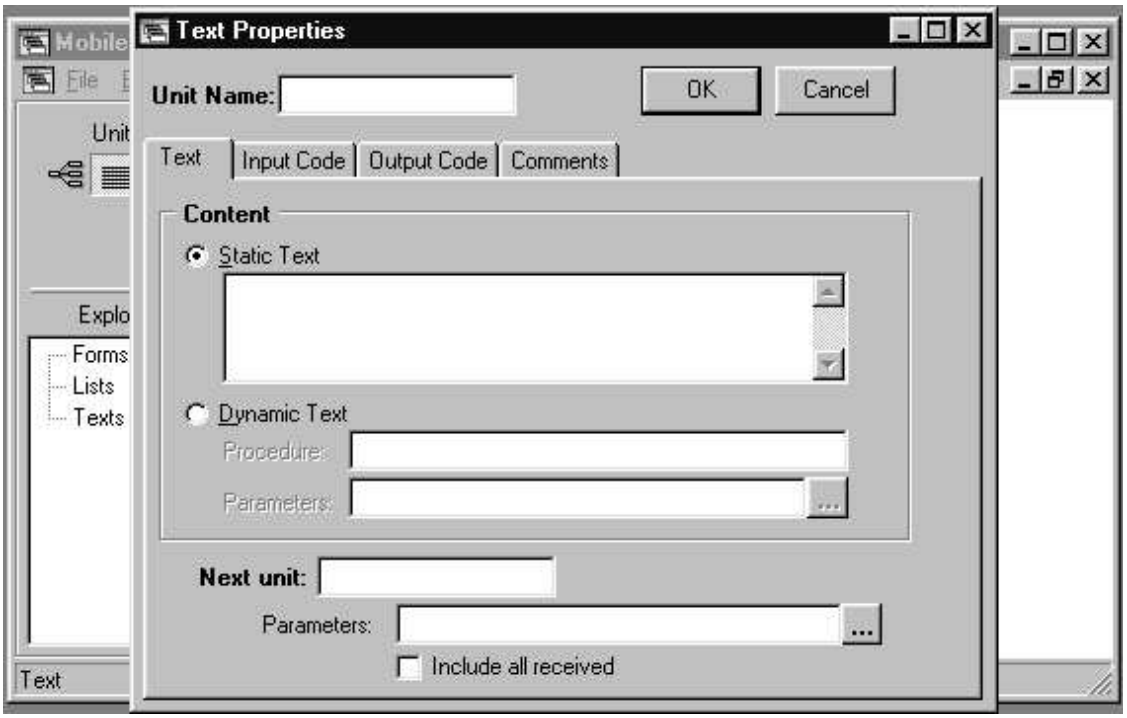

Figura 5.3: Propriedades para um texto

5.2.3 Formulário

Possui uma lista de campos para serem preenchidos. Cada campo consiste de uma descrição curta, um nome identificador e o tipo ou domínio do campo (figura 5.4).

É importante que o tipo do campo seja informado para que a interface possa ser gerada adequadamente, permitindo aumentar a usabilidade da aplicação para o usuário final. Os tipos possíveis são: inteiro, real, texto, booleano e lista enumerada.

Um formulário deve especificar ainda:

- rotina de tratamento, responsável pelo processamento dos campos do formulário. Como toda rotina, deverá retornar um código de sucesso (`code`) de sua operação.
- próxima unidade em caso de sucesso, ou seja, qual unidade deverá ser exibida se a rotina de processamento for bem sucedida.
- próxima unidade em caso de falha, isto é, qual unidade deverá ser exibida se a rotina de processamento falhar.

5.2.4 Descrição de parâmetros

Cada unidade pode receber parâmetros de uma unidade anteriormente exibida, e passar parâmetros para a próxima unidade a ser mostrada. Dentro de cada uma destas unidades

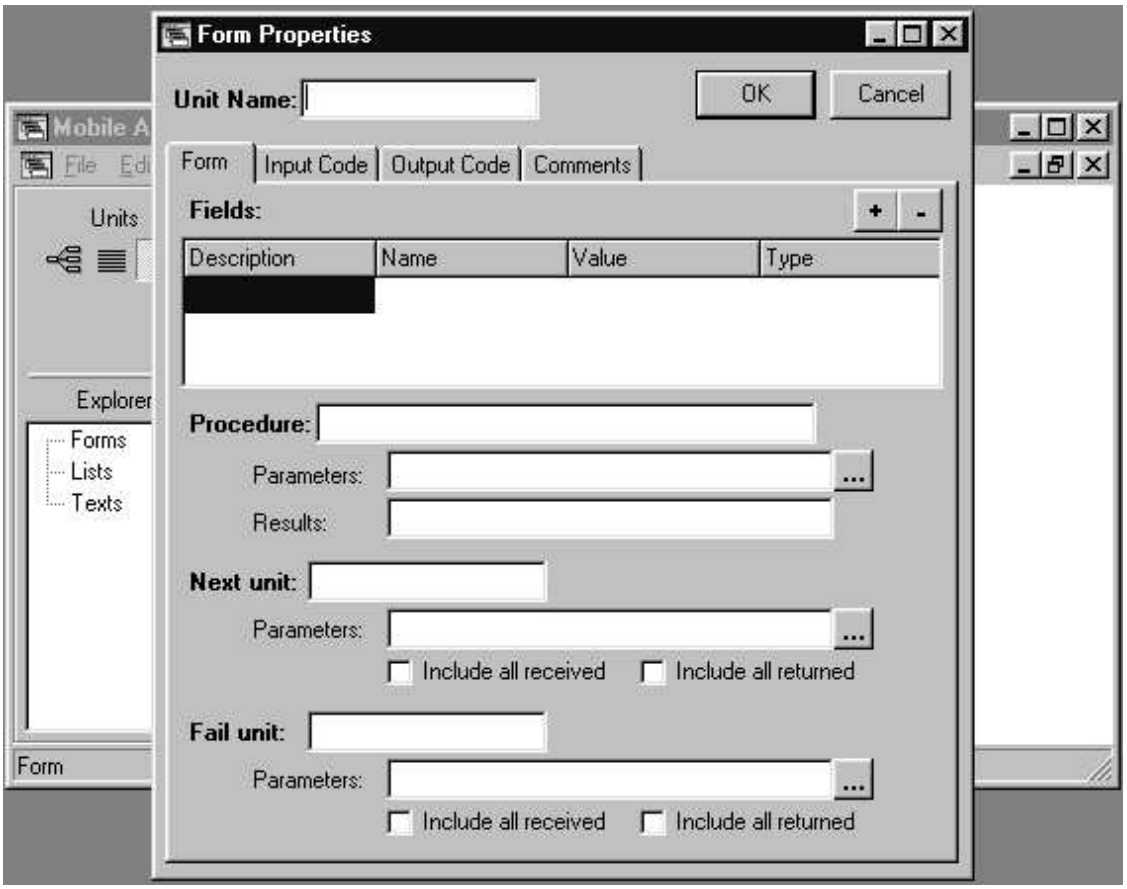

Figura 5.4: Propriedades para um formulário

o projetista pode definir um processamento de entrada, realizado antes que a unidade de interação seja exibida, e um processamento de saída, realizado após a finalização daquela unidade de interação. Estes processamentos consistirão de chamadas a código externo.

Para cada chamada poderão ser definidos parâmetros, que serão passados ao código externo, e os resultados esperados, retornados após processamento da chamada (parâmetros de retorno ou saída). Não serão definidos explicitamente os tipos (como tipo inteiro ou texto, por exemplo) destes dados: todos serão manipulados como `strings`. O projetista será responsável por fazer as devidas conversões de tipos em seu código.

Parâmetros serão definidos como uma lista da forma `nome=valor`, separados por ponto-e-vírgula. No caso dos resultados, apenas os nomes dos dados retornados serão definidos. Toda chamada retorna um número inteiro indicando o sucesso da operação (valor `0`) ou seu fracasso (qualquer outro valor).

Este código de retorno poderá ser repassado para outras unidades através da palavra-chave `$code`.

Os parâmetros recebidos por uma unidade podem ser repassados diretamente para um

chamada à código externo. Para isso será utilizado um par especial `$$=$$`. De modo semelhante, os resultados de uma chamada podem ser passados para a unidade seguinte, utilizando-se o par `$RESULT$=$$`

5.2.5 Ações pré-definidas

Finalmente, duas ações com semântica especial foram definidas. A primeira é identificada pela palavra-chave `!SYNC`. Indica que uma rotina de sincronização deverá ser chamada. A segunda é identificada pela palavra-chave `!EXIT`. Esta ação marca o final da execução da aplicação.

Estas ações podem ser utilizadas em qualquer parte onde uma rotina ou unidade puder ser especificada na ferramenta. Por exemplo, como ação para um (último) item da uma lista de comandos.

5.3 Formato de armazenamento

Aplicações geradas usando a ferramenta proposta são armazenadas através de documentos em XML. Uma das vantagens de utilizar-se XML é a facilidade de leitura (por pessoas) associada à facilidade de processamento (por máquinas). Além disso XML é um padrão internacional muito bem aceito atualmente.

A estrutura deste documento é mostrada a seguir:

```
<?xml version="1.0"?>
 <maml id="MyApplication">
    <application>
        <unit id="list1">
            <input>
                <routine id="in-list1">
                    <param name="par1" value="val1"/>
                    <!-- redundante: toda rotina deve retornar "code" -->
                    <result name="code"/>
                </routine>
            </input>
            <list>
                <item action="text1" description="Item 1">
                    <param name="par1" value="val1"/>
                </item>
                <item action="text2" description="Item 2">
                    <param name="par2" value="val2"/>
                </item>
            </list>
            <output/>
            <comment/>
        </unit>
        <unit id="text1">
```

```
        <input/>
        <text>Static content</text>
        <output/>
        <comment/>
        <next action="list1">
            <param name="par1" value="val1"/>
        </next>
    </unit>
    <unit id="text2">
        <input/>
        <text>
            <routine id="content-text2">
                <param name="par1" value="val1"/>
                <!-- redundante: toda rotina deve retornar "code" -->
                <result name="code"/>
                <!-- redundante: rotinas dentro de elementos text devem retornar "text" -->
                <result name="text"/>
            </routine>
        </text>
        <output/>
        <next action="list1">
            <param name="par2" value="val2"/>
        </next>
        <comment/>
    </unit>
    <unit id="form1">
        <input/>
        <form>
            <field name="f1" type="string" value="">Field 1</field>
            <field name="f2" type="integer" value="">Field 2</field>
            <field name="f3" type="float" value="">Field 3</field>
            <field name="f4" type="boolean" value="">Field 4</field>
            <field name="f5" type="enum1" value="">Field 5</field>
            <routine id="process-form1">
                <param name="par3" value="val3"/>
                <!-- redundante: toda rotina deve retornar "code" -->
                <result name="code"/>
            </routine>
        </form>
        <output/>
        <next action="list1">
            <param name="par3" value="val3"/>
        </next>
        <fail action="list1">
            <param name="par4" value="val4"/>
        </fail>
        <comment/>
    </unit>
</application>
<resource>
    <enumeration id="enum1" semantic="radio">
```

```
            <value id="Value 1" value="1"/>
            <value id="Value 2" value="2"/>
        </enumeration>
    </resource>
</maml>
```

Basicamente o documento é formado por um conjunto de unidades definidas pelo elemento `unit`. MAB recupera as unidades definidas em uma aplicação e insere os elementos no documento XML, de acordo com as propriedades de cada unidade. A seguir é detalhado cada elemento do documento XML. O documento de definição de tipos (DTD) para este formato pode ser encontrado no apêndice A.

5.3.1 maml

Utilização: elemento raiz do documento

Atributos: `id` - nome da aplicação

Elementos pais: Nenhum

Elementos filhos: `application, resource`

5.3.2 application

Utilização: marca o início da descrição da aplicação

Atributos: nenhum

Elementos pais: `maml`

Elementos filhos: `unit`

5.3.3 unit

Utilização: descreve uma unidade da aplicação. Pode ser uma lista, texto ou formulário. Cada elemento `unit` pode possuir apenas um único elemento `list`, `text` ou `form`, que indicará o tipo desta unidade.

Atributos: `id` - nome desta unidade

Elementos pais: `application`

Elementos filhos: `input,output,comment,list,text,form,next,fail`

5.3.4 input

Utilização: define a rotina que deverá ser executada antes de exibir uma unidade. Opcional.

Atributos: Nenhum

Elementos pais: `unit`

Elementos filhos: `routine`

5.3.5 output

Utilização: define a rotina que deverá ser executada após a exibição de uma unidade. Opcional.

Atributos: Nenhum

Elementos pais: `unit`

Elementos filhos: `routine`

5.3.6 comment

Utilização: marcação usada para comentários relativos a uma unidade. Opcional.

Atributos: Nenhum

Elementos pais: `unit`

Elementos filhos: Nenhum

5.3.7 text

Utilização: usado para definir uma unidade do tipo texto. Para textos dinâmicos usa-se uma rotina, através do elemento `routine`, que retornará o conteúdo do texto via parâmetro de retorno obrigatório `text`.

Atributos: Nenhum

Elementos pais: `unit`

Elementos filhos: `routine`

5.3.8 list

Utilização: usado para definir uma unidade do tipo lista. No caso de itens gerados dinamicamente usa-se uma rotina, através do elemento `routine`, que os retornará via parâmetros de retorno obrigatórios `descriptions`, `actions` e `parameters`.

Atributos: Nenhum

Elementos pais: `unit`

Elementos filhos: `item, routine`

5.3.9 form

Utilização: usado para definir uma unidade do tipo formulários.

Atributos: Nenhum

Elementos pais: `unit`

Elementos filhos: `field, routine`

5.3.10 next

Utilização: define a unidade que deverá ser exibida após a unidade atual. Apenas unidades do tipo texto e formulário podem utilizar este elemento. No caso de um formulário, este elemento especifica qual unidade será exibida no caso de sucesso no processamento de seus campos.

Atributos: `action` - identificador da unidade que deverá ser exibida.

Elementos pais: `unit`

Elementos filhos: `param`

5.3.11 fail

Utilização: define a unidade que deverá ser exibida após o formulário atual em caso de falha em seu processamento. Apenas formulários podem utilizar este elemento.

Atributos: `action` - identificador da unidade que deverá ser exibida.

Elementos pais: `unit`

Elementos filhos: `param`

5.3.12 item

Utilização: define um item de uma unidade do tipo lista. Pode conter, opcionalmente, parâmetros que serão enviados a unidade associada a este item.

Atributos: `action` - identificador da unidade que deverá ser exibida se este item da lista for selecionado; `description` - texto descritivo para ser exibido ao usuário da aplicação

Elementos pais: `list`

Elementos filhos: `param`

5.3.13 field

Utilização: define um campo de uma unidade do tipo formulário.

Atributos: `name` - nome do campo; `value` - valor do campo (atributo opcional); `type` - tipo do informação do campo, podendo ser *string*, inteiro, ponto flutuante, booleano ou uma enumeração definida pelo desenvolvedor.

Elementos pais: `form`

Elementos filhos: Nenhum

5.3.14 routine

Utilização: descreve a assinatura de uma rotina de processamento externa à aplicação. Deve sempre possuir no mínimo um elemento `result` com nome igual a `code`, além de opcionalmente elementos `param`.

Atributos: `id` - identificador da rotina

Elementos pais: `input, output, text, form, list`

Elementos filhos: `param, result`

5.3.15 param

Utilização: define um parâmetro que será passado para uma rotina de processamento ou unidade.

Atributos: `name` - nome do parâmetro; `value` - valor do parâmetro

Elementos pais: `routine, item, next`

Elementos filhos: Nenhum

5.3.16 result

Utilização: define um resultado (parâmetro de saída ou retorno) de uma rotina de processamento.

Atributos: `name` - nome associado a este resultado

Elementos pais: `routine`

Elementos filhos: Nenhum

5.3.17 resource

Utilização: marca o início da descrição de tipos e recursos definidos pelo usuário. Neste trabalho definirá apenas o tipo enumeração, através do elemento `enumeration`.

Atributos: Nenhum

Elementos pais: `maml`

Elementos filhos: `enumeration`

5.3.18 enumeration

Utilização: descreve uma enumeração definida pelo usuário.

Atributos: `id` - identificador (único) da enumeração

Elementos pais: `resource`

Elementos filhos: `value`

5.3.19 value

Utilização: define um elemento da enumeração. Cada elemento associa um valor a um identificador.

Atributos: `id` - identificador do valor definido pela enumeração; `value` - valor propriamente dito ao qual corresponde o identificador

Elementos pais: `enumeration`

Elementos filhos: Nenhum

5.4 Geração de código

O código será gerado a partir do documento XML especificado anteriormente. Cada plataforma de destino deverá possuir um gerador associado. Um gerador é um componente de *software* com uma interface padronizada, que receberá e enviará informações básicas de e para a ferramenta além de, provavelmente, solicitar ao projetista informações extras, essenciais à geração de código para a plataforma a que se destina.

Um gerador possuirá um *parser XML*, capaz de ler o documento armazenado e manipulá-lo de modo a gerar código para a plataforma a que se destina. Deverá ainda implementar a seguinte interface:

- `getName` - recupera uma sequência de caracteres com o nome do gerador para exibição na ferramenta
- `getVersion` - recupera a versão atual do gerador para exibição na ferramenta
- `getTargetPlataform` - recupera uma seqüência de caracteres com uma pequena descrição da plataforma de destino para exibição na ferramenta
- `needSync` - este método indica se o código gerado necessita de um mecanismo de sincronização. Deve retornar `1` se for necessário ou `0` em caso contrário.
- `initializeGenerator` - responsável por capturar informações necessárias à geração do código. Recebe dois parâmetros: o caminho e o nome da aplicação corrente. O comportamento deste método é altamente dependente da implementação e da plataforma de destino. Poderá, por exemplo, abrir janelas de diálogo para obter do projetista informações diversas, como opções de geração de código, local de destino do código gerado, ou qualquer outra que seja relevante para o funcionamento do gerador. Retorna 0 se a chamada for bem sucedida.
- `generateCode` - este método só é chamado após o retorno a uma chamada do método anterior. É o local onde todo o trabalho de geração de código é realmente feito. É recomendável que ofereça ao usuário algum tipo de retorno visual do processamento. As seguintes padronizações também são recomendadas para o funcionamento de um gerador:
  - criar um diretório dentro da pasta onde o projeto corrente está armazenado, dentro do qual deverá gerar seus arquivos
  - criar todas as assinaturas e demais métodos auxiliares das rotinas externas definidas no projeto na classe `Routine`, definida no arquivo `Routine.java`

  Este método retorna 0 se a chamada for bem sucedida.

Um gerador qualquer que atenda a estes requisitos poderá, assim, ser registrado na ferramenta e apresentado como opção de plataforma de destino da aplicação desenvolvida pelo projetista. Esta interface deve ser implementada através de um servidor de automação ActiveX.

Neste trabalho foram implementados dois geradores, um para a plataforma WAP 1.1 e outro para dispositivos que utilizam J2ME/CLDC/MIDP (figura 5.5). Este geradores foram criados a partir de um modelo básico, desenvolvido para o ambiente Delphi, contendo todos os métodos necessários bem como funções de `parsing` para cada tipo de unidade. Este modelo também está disponível na ferramenta, tornando a crição de novos geradores bastante ágil.

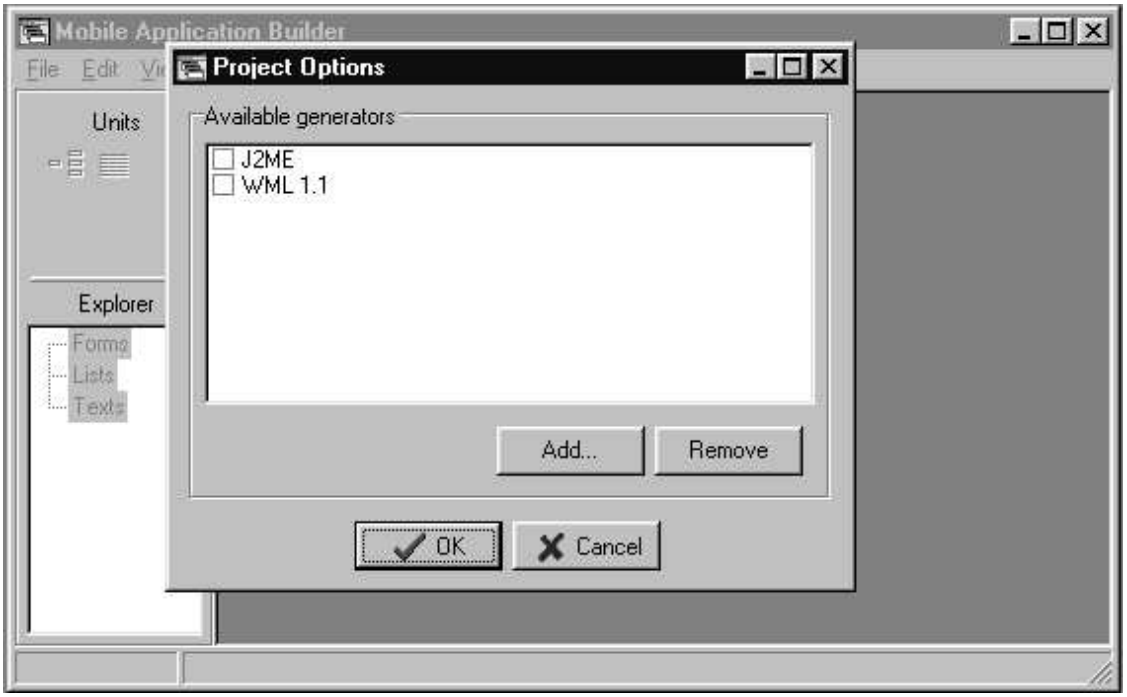

Figura 5.5: Geradores disponíveis no MAB

5.4.1 Extensão do formato de armazenamento

A inclusão de código externo pelo projetista seria bastante trabalhosa no caso de alterações no projeto, pois a cada geração todos os arquivos são criados novamente. Desta maneira, o trabalho de codificação eventualmente já realizado teria que ser feito de novo. Mesmo que isso signifique apenas "recortar e colar" o código de um lugar para outro, isto demanda tempo.

Para facilitar o trabalho do projetista (usuário da ferramenta) foi criada uma extensão

através de uma anotação[2] no documento XML de armazenamento.

Esta anotação será identificada pelo texto `@GENERATOR-OPTIONS`. Em seguida, cada um em uma linha, será indicado o nome do gerador de destino - o mesmo retornado pelo método `getName`, através do texto `@@GENERATOR-NAME`, e o nome do arquivo que implementa as rotinas externas, usando o texto `@@ROUTINES-CLASS`. Assim, esta anotação teria o seguinte formato dentro do documento de armazenamento:

```
<!--@@GENERATOR-OPTIONS
@GENERATOR-NAME WML
@ROUTINES-CLASS MyRoutine
-->
```

A interpretação desta anotação será feita pelo gerador. Este deverá utilizar a classe de rotinas externas definida na anotação contendo seu nome, se esta existir.

5.5 Sincronização

Para o caso de plataformas que necessitem de um mecanismo de sincronização com um servidor fixo será utilizado SyncML [71].

SyncML provê um *framework* para sincronização de dispositivos que não possuem conexão permanente a uma rede de comunicações qualquer. Para isso define um documento XML para fazer o intercâmbio de dados no processo de sincronização, além de um protocolo para diferentes procedimentos de sincronização.

Será utilizado o protocolo HTTP como mecanismo de transporte para os procedimentos de sincronização[3].

SyncML define sete tipos de sincronização:

- duas vias (*two-way sync*): cliente e servidor trocam informações sobre os dados modificados em cada parte. O cliente envia suas modificações primeiro.
- lenta (*slow sync*): uma variação da sincronização de duas vias em que todos os itens são comparados uns com os outros, campo a campo. Ou seja, o cliente envia todos os seus dados para o servidor, e este faz uma análise de sincronização contra seus dados.
- uma via originada no cliente (*one-way sync from client only*): o cliente envia suas modificações para o servidor, mas este não envia as suas de volta ao cliente.
- atualização originada no cliente (*refresh sync from client only*): o cliente envia todos os seus dados para o servidor, o qual deverá substituir todos os seus dados com os dados recebidos do cliente.

[2] Uma anotação é uma marcação com uma semântica especial definida geralmente dentro de comentários.
[3] A especificação de SyncML prevê alguns mecanismos de transporte, entre eles HTTP, WSP e OBEX.

- uma via originada no servidor (*one-way sync from server only*): o cliente recebe todas as modificações do servidor, mas não envia envia suas modificações para este.
- atualização originada no servidor (*refresh sync from server only*): o servidor envia todos os seus dados para o cliente, o qual deverá substituir todos os seus dados com os dados recebidos do servidor.
- requisitada pelo servidor (*server alerted sync*): o servidor solicita ao cliente que inicie uma sincronização de um dos tipos anteriores.

Neste trabalho não será descrita a especificação SyncML (para maiores informações consulte [71]).

A implementação de *engines*, tanto cliente como servidor, que implementem o protocolo proposto também está fora do escopo deste trabalho. Em princípio, qualquer uma das setes formas pode ser implementada e utilizada junto com a ferramenta.

5.6 Acesso a dados

Acessos a bases de dados serão sempre realizadas a partir do código externo. Desta forma pretende-se conseguir um isolamento entre interface com usuário, lógica de negócio e banco de dados, seguindo o modelo de programação em camadas.

Qualquer tipo de dado, vindo de qualquer fonte, deverá ser retornado como um documento XML. O formato básico deste documento será:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<data type="<tipo>">
  ...
</data>
```

O atributo `type` indicará o tipo do dado contido neste documento e, desta maneira, como este dado deve ser acessado e manipulado.

Inicialmente só será definido o tipo `recordset`, para acesso a bancos de dados relacionais. O documento retornado então terá a seguinte forma:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<data type="recordset">
  <meta>
    <field name="nome" type="string"/>
    <field name="sobrenome" type="string"/>
  </meta>
  <record id="1" nome="André" sobrenome="Minelli" />
  <record id="2" nome="Antônio" sobrenome="Loureiro" />
</data>
```

Uma descrição de cada um dos elementos deste documento é dada a seguir. O documento de definição de tipos para este formato pode ser encontrado no apêndice A.

5.6.1 data

Utilização: elemento raiz do documento

Atributos: `type` - indica o tipo de dados contidos no documento.

Elementos pais: Nenhum

Elementos filhos: `meta, record`

5.6.2 meta

Utilização: descreve os campos que fazem parte deste documento. Esta descrição pode ser informada como um DTD anexado ao documento. A inclusão do DTD é opcional, mas altamente recomendável.

Atributos: Nenhum

Elementos pais: `data`

Elementos filhos: `field`

5.6.3 field

Utilização: descreve um campo do conjunto de registros que está sendo recuperado.

Atributos: `name` - identifica o nome do campo; `type` - identifica o tipo de dado contido no campo

Elementos pais: `meta`

Elementos filhos: Nenhum

5.6.4 record

Utilização: representa um registro de informação recuperado.

Atributos: `id` - identificador do registro (a chave primária de um banco de dados relacional, tipicamente); demais atributos são fornecidos pelos elementos `field` ou pelo DTD anexado ao documento.

Elementos pais: `data`

Elementos filhos: Nenhum

A implementação de interfaces que recuperem dados utilizando o documento XML proposto nesta seção está fora do escopo deste trabalho.

5.7 Plataforma de desenvolvimento

O ambiente Borland Delphi foi utilizado para o desenvolvimento da ferramenta propriamente dita. Esta escolha foi motivada pela maior familiaridade com este ambiente, além da grande disponibilidade de componentes de *software* gratuitos (que proporcionam um desenvolvimento acelerado).

Desta forma a ferramenta estará disponível apenas para a plataforma Windows. Mas existe a possibilidade de poder ser disponibilizada também para plataformas Linux, com o lançamento do pacote Borland Kylix[4].

O servidor Web escolhido para este trabalho foi o Tomcat [74] versão 3.2, uma implementação das especificações *Java Servlets API* 2.2 e *Java Server Pages* 1.1 derivada do projeto Apache.

Um componente fundamental deste trabalho é o *parser* XML da Microsoft [75], disponível gratuitamente. Este componente ActiveX possui bom desempenho e oferece bons recursos para recuperação de elementos e atributos de documentos XML, tendo facilitado bastante esta parte do desenvolvimento.

A ferramenta em si resultou em um código relativamente pequeno, em torno de 5.456 linhas, gerando um arquivo executável de 980 kb. Os geradores precisaram de 1.239 linhas de código, para WML, e 1.457 linhas, para J2ME, além de mais algumas dezenas de linhas em cada um, correpondentes ao código comum do modelo de gerador (*template*)utilizado no desenvolvimento de ambos.

A utilização de objetos COM e bibliotecas de ligação dinâmica (DLLs), que por um lado aumenta a complexidade e diminui um pouco a robustez, por outro aumenta bastante a flexibilidade e extensibilidade da ferramenta.

5.8 Como utilizar MAB

Será descrito nesta seção as etapas necessárias para modelar uma aplicação e implementá-la utilizando MAB. Esta é apenas uma sugestão de um processo simples, que também será utilizado no desenvolvimento das aplicações de exemplo do capítulo seguinte. Obviamente que outros processos de desenvolvimento podem ser utilizados ou adaptados para a ferramenta. A especificação de um processo completo está fora do escopo deste trabalho.

Basicamente são utilizados quatro passos para implementar uma aplicação qualquer:

- Especificação da aplicação
- Mapeamento das unidades
- Geração

[4]Kylix é, na prática, uma versão do Delphi portada para o ambiente Linux

- Inserção de código externo

5.8.1 Especificação da aplicação

A especificação da aplicação começa com a definição de requisitos. Como exemplo, a ser usado no próximo capítulo, tem-se a seguinte definição de requisitos: a aplicação permitirá gerenciar uma agenda de contatos telefônicos. Um contato é composto simplesmente por um nome e respectivo telefone. O usuário poderá listar os contatos, e alterar, adicionar ou remover um determinado contato.

Dada esta especificação, procede-se uma etapa de modelagem tradicional. Em princípio qualquer técnica poderá ser usada para isto.

5.8.2 Mapeamento das unidades

Uma vez definida a aplicação será necessário mapear as interações com o usuário em uma ou mais unidades do MAB. As unidades podem ser de três tipos básicos: lista (estática ou dinâmica), texto (estático ou dinâmico) e formulário.

Geralmente sempre será possível fazer um mapeamento direto de uma interação qualquer para uma dessas unidades. Porém recomenda-se fazer, a seguir, uma análise mais cuidadosa deste primeiro mapeamento, pois invariavelmente será possível realizar otimizações através de parametrização de unidades e rotinas e da utilização de listas e textos dinâmicos.

5.8.3 Geração

O passo seguinte é gerar as aplicações, ou seja executar os geradores disponíveis. Para isto deve ser escolhida a opção *Build*, disponível no menu *Project* (a tecla de atalho F9 também pode ser usada).

Cada gerador criará um diretório dentro da pasta do projeto corrente, onde os arquivos gerados serão salvos. O arquivo mais importante possui extensão java e é chamado `Routine.java`. Este arquivo conterá as assinaturas de todas as rotinas externas definidas no projeto, e que deverão ser implementadas pelo projetista.

5.8.4 Inserção de código externo

Por fim, o projetista deverá inserir seu código, implementando as assinaturas de cada rotina definida no projeto.

Isso pode ser feito de duas formas:

- Incluindo código diretamente na classe de rotinas (geralmente `Routine.java`) criada por cada gerador. O único inconveniente é que será necessário editar estas classes

a cada nova geração do projeto, pois nesse momento essas classes são recriadas, sobrescrevendo arquivos pré-existentes.

- Implementando uma sub-classe da classe de rotinas criada por cada gerador. Desta maneira o único trabalho após um nova geração será o de ajustar na sub-classe as assinaturas dos métodos herdados, caso necessário. Neste caso, o projetista deve indicar o nome do arquivo desta sub-classe, para cada gerador. Isto é feito através da opção *Build options...* disponível no menu *Project*. É recomendado que este arquivo esteja dentro do diretório criado por cada gerador dentro da pasta do projeto.

O segundo método geralmente será preferido por usuários da ferramenta dada sua praticidade.

Capítulo 6

Exemplos de Utilização

Para testar o trabalho desenvolvido, duas aplicações foram especificadas e implementadas utilizando os recursos oferecidos pela ferramenta.

A primeira aplicação é uma agenda de contatos telefônicos. Foi criada como um protótipo para demonstrar como pode ser feita a modelagem de uma aplicação utilizando MAB e como é feita sua implementação. Este processo é descrito em detalhes na próxima seção.

A segunda é uma aplicação de *m-commerce* com uma complexidade maior. Foi criada para abranger várias das etapas de desenvolvimento que um projetista, eventualmente, enfrentaria ao utilizar esta ferramenta.

6.1 Aplicação 1: Agenda de contatos

6.1.1 Especificação da aplicação

Tem-se a seguinte definição de requisitos: a aplicação permitirá gerenciar uma agenda de contatos telefônicos. Um contato é composto simplesmente por um nome e respectivo telefone. O usuário poderá listar os contatos, e alterar, adicionar ou remover um determinado contato.

Dada esta especificação, procede-se uma etapa de modelagem tradicional utilizando-se UML, por exemplo. O diagrama de casos de uso obtido é mostrado na figura 6.1.

6.1.2 Mapeamento das unidades

Uma vez definida a aplicação será necessário mapear os casos de uso em uma ou mais unidades do MAB. As unidades podem ser de três tipos básicos: lista (estática ou dinâmica), texto (estático ou dinâmico) e formulário. Um possível mapeamento, sem realizar uma análise mais detalhada, pode ser feito como mostrado na tabela 6.1.

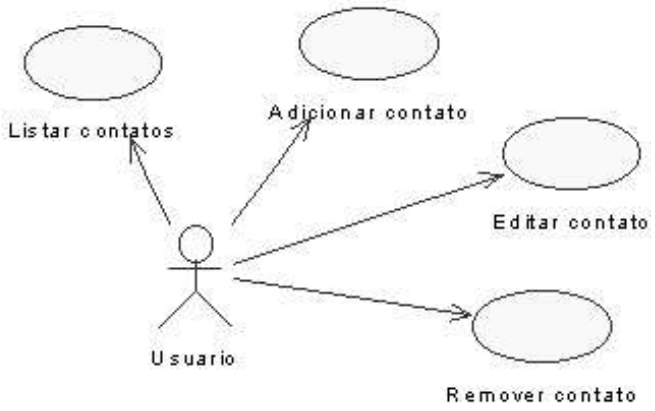

Figura 6.1: Aplicação 1 - Diagrama de Casos de uso

Na unidade inicial da aplicação, *MainList*, estão presentes as duas ações especiais !SYNC, para indicar o ponto onde uma rotina de sincronização será chamada, e !EXIT, que indica que aplicação deverá ser finalizada.

Este primeiro mapeamento poderia ser usado na ferramenta sem problemas. Mas analisando-o com mais cuidado é possível realizar algumas otimizações através de parametrização de unidades e rotinas. Por exemplo, as listas dinâmicas *ShowList*, *EditList* e *DeleteList* exibem o mesmo resultado – a lista de contatos, embora sejam funcionalmente diferentes. Assim poderemos agrupá-las em uma única lista dinâmica a qual receberá um parâmetro, que será chamado `action`, indicando qual será a função desempenhada pela lista. O mapeamento da tabela 6.2 incorpora esta modificação e mais algumas outras (boa parte decorrentes desta).

6.1.3 Codificação na ferramenta

Feito o mapeamento, inicia-se MAB para inserir as unidades obtidas. Cada uma das três unidades é representada por um ícone. Basta selecionar o tipo de unidade desejada, clicando sobre o respectivo ícone, e clicar dentro da área de trabalho da ferramenta para inserir uma unidade no projeto. A figura 6.2 mostra a inserção da primeira unidade da aplicação, *MainList*, uma lista contendo o menu principal de opções da agenda. Isso é feito para cada unidade da tabela 6.2. A figura 6.3 ilustra o projeto final, já desenhado na ferramenta e salvo com o nome *Contact*.

O passo seguinte é gerar as aplicações, ou seja executar os geradores disponíveis. Para isto deve ser escolhida a opção *Build*, disponível no menu *Project* (a tecla de atalho F9 também pode ser usada).

O primeiro gerador criará o diretório WML e, dentro deste, vários arquivos com extensão jsp e um arquivo com extensão java, chamado `Routine.java`. Este último conterá as assinaturas de todas as rotinas externas definidas no projeto, e que deverão ser implemen-

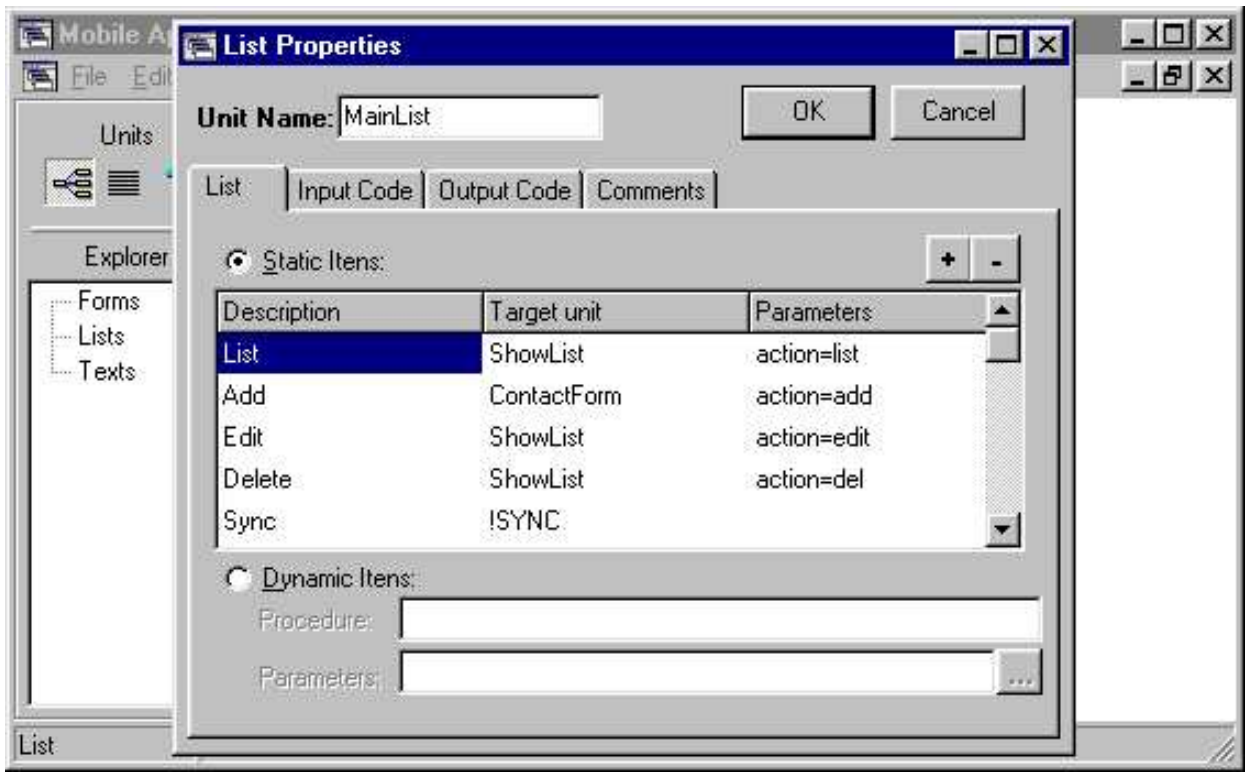

Figura 6.2: Inserindo a primeira unidade - projeto Contact

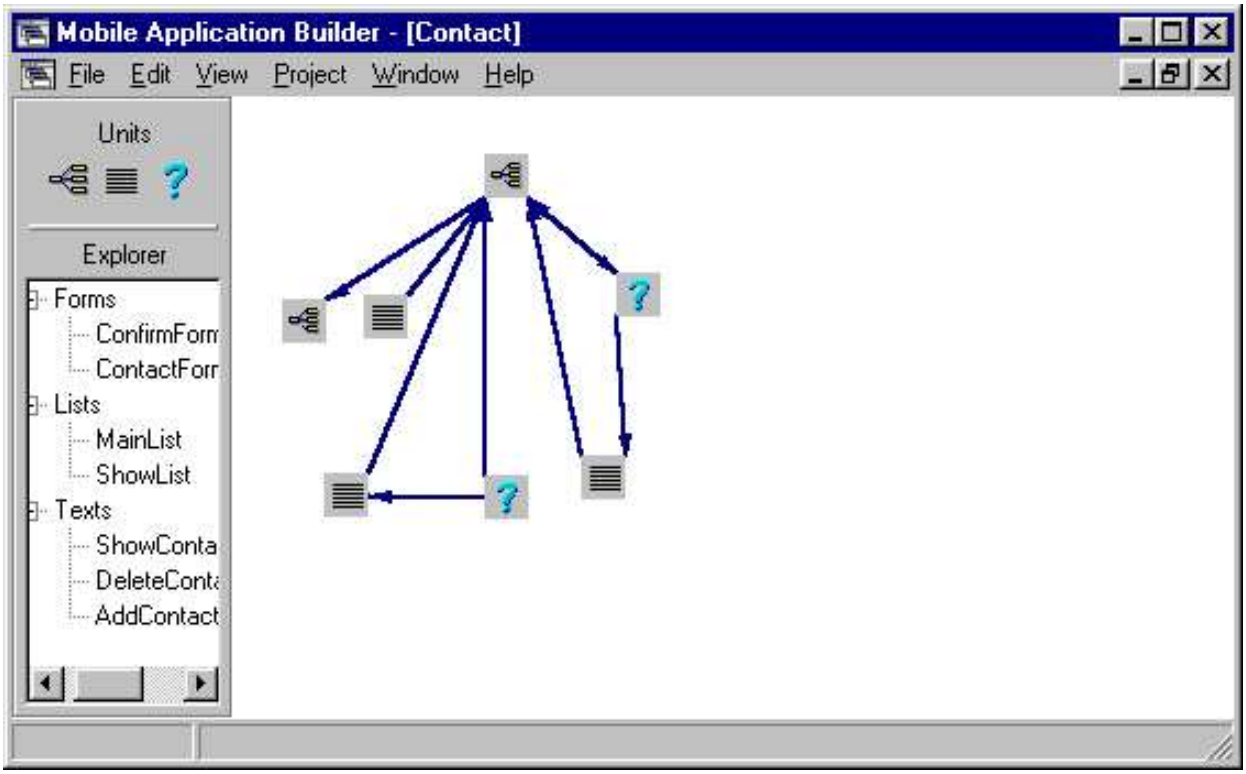

Figura 6.3: Unidades mapeadas no MAB - projeto Contact

<table>
<tr><td>Unidades</td><td colspan="4">Descrição da unidade</td></tr>
<tr><td>MainList</td><td colspan="4">Lista estática</td></tr>
<tr><td rowspan="7"></td><td>Descrição</td><td colspan="2">Ação</td><td>Parâmetros</td></tr>
<tr><td>List</td><td colspan="2">ShowList</td><td></td></tr>
<tr><td>Add</td><td colspan="2">AddForm</td><td></td></tr>
<tr><td>Edit</td><td colspan="2">EditList</td><td></td></tr>
<tr><td>Delete</td><td colspan="2">DeleteList</td><td></td></tr>
<tr><td>Sync</td><td colspan="2">!SYNC</td><td></td></tr>
<tr><td>Exit</td><td colspan="2">!EXIT</td><td></td></tr>
<tr><td>ShowList</td><td colspan="4">Lista dinâmica</td></tr>
<tr><td></td><td colspan="4">Rotina: getContactList (mostra contatos armazenados)
Obs:
-cada item da lista conterá um parâmetro id, correspondendo ao identificador daquele contato na agenda
-a ação para todos os itens será ShowContactText, que receberá o parâmetro id</td></tr>
<tr><td>AddForm</td><td colspan="4">Formulário</td></tr>
<tr><td rowspan="4"></td><td>Descrição</td><td>Nome</td><td>Valor</td><td>Tipo</td></tr>
<tr><td>Name</td><td>name</td><td></td><td>String</td></tr>
<tr><td>Telephone</td><td>tel</td><td></td><td>String</td></tr>
<tr><td colspan="4">Rotina: addContact($name,$tel)
Sucesso: MainList
Falha: AddContactErrText</td></tr>
<tr><td>EditList</td><td colspan="4">Lista dinâmica</td></tr>
<tr><td></td><td colspan="4">Rotina: getContactList2 (mostra contatos armazenados)
Obs:
-cada item da lista conterá um parâmetro id, correspondendo ao identificador daquele contato na agenda
-a ação para todos os itens será EditForm, que receberá o parâmetro id</td></tr>
<tr><td>EditForm</td><td colspan="4">Formulário</td></tr>
<tr><td rowspan="5"></td><td colspan="4">Entrada: getContact ($id,$name,$tel)</td></tr>
<tr><td>Descrição</td><td>Nome</td><td>Valor</td><td>Tipo</td></tr>
<tr><td>Name</td><td>name</td><td>$name</td><td>String</td></tr>
<tr><td>Telephone</td><td>tel</td><td>$tel</td><td>String</td></tr>
<tr><td colspan="4">Rotina: updateContact($id,$name,$tel)
Sucesso: MainList
Falha: EditContactErrText</td></tr>
<tr><td>DeleteList</td><td colspan="4">Lista dinâmica</td></tr>
<tr><td></td><td colspan="4">Rotina: getContactList3 (mostra contatos armazenados)
Obs:
-cada item da lista conterá um parâmetro id, correspondendo ao identificador daquele contato na agenda
-a ação para todos os itens será ConfirmForm, que receberá o parâmetro id</td></tr>
</table>

Tabela 6.1: Primeiro mapeamento da agenda de contatos

ConfirmForm	Formulário			
	Descrição	Nome	Valor	Tipo
	Delete	del		Booleano
	Rotina: delContact($id,$del) Sucesso: MainList Falha: DeleteContactErrText			
ShowContactText	Texto dinâmico			
	Rotina: showContact ($id) Próxima: MainList			
AddContactErrText	Texto dinâmico			
	Rotina: getAddContactErrMsg ($code) Próxima: MainList			
EditContactErrText	Texto dinâmico			
	Rotina: getEditContactErrMsg ($code) Próxima: MainList			
DeleteContactErrText	Texto dinâmico			
	Rotina: getDeleteContactErrMsg ($code) Próxima: MainList			

Tabela 6.1: Primeiro mapeamento da agenda de contatos (continuação)

tadas pelo projetista.

O segundo gerador criará o diretório J2ME, onde estarão dois arquivos com extensão java. Um arquivo terá o mesmo nome do projeto, no caso `Contact.java`, e contém o *midlet* correspondente à aplicação J2ME gerada. O segundo arquivo, `Routine.java`, é a classe que deverá ser implementada pelo projetista, contendo as assinaturas das rotinas externas definidas no projeto[1].

6.1.4 Inserção de código externo

Por fim, o projetista deverá inserir seu código, implementando as assinaturas de cada rotina definida no projeto.

Isso pode ser feito de duas formas:

- Incluindo código diretamente na classe de rotinas (geralmente `Routine.java`) criada por cada gerador. O único inconveniente é que será necessário editar estas classes a cada nova geração do projeto, pois nesse momento essas classes são recriadas, sobrescrevendo arquivos pré-existentes.

- Implementando uma sub-classe da classe de rotinas criada por cada gerador. Desta maneira o único trabalho após um nova geração será o de ajustar na sub-classe as assinaturas dos métodos herdados, caso necessário. Neste caso, o projetista deve indicar o nome do arquivo desta sub-classe, para cada gerador. Isto é feito através da

[1]Embora os nomes das duas classes que devem ser implementadas possuam o mesmo nome e possuam interfaces idênticas, isto não é uma regra - embora seja recomendado para todos os geradores!

Unidades	Descrição da unidade			
MainList	Lista estática			
	Descrição	Ação	Parâmetros	
	List	ShowList	action=list	
	Add	ContactForm	action=add	
	Edit	ShowList	action=edit	
	Delete	ShowList	action=del	
	Sync	!SYNC		
	Exit	!EXIT		
ShowList	Lista dinâmica			
	Rotina: getContactList (mostra contatos armazenados) Obs: -cada item da lista conterá um parâmetro *id*, correspondendo ao identificador daquele contato na agenda -a ação, que receberá o parâmetro *id*, para cada item será: ShowContactText se action=list ContactForm se action=edit ConfirForm se action=del			
ContactForm	Formulário			
	Entrada: getContact ($id,$name,$tel)			
	Descrição	Nome	Valor	Tipo
	Name	Name	$name	String
	Telephone	Tel	$tel	String
	Rotina: processContact($id,$action,$name,$tel) Sucesso: MainList Falha: AddContactErrText			
ConfirmForm	Formulário			
	Descrição	Nome	Valor	Tipo
	Delete	del		Booleano
	Rotina: delContact($id,$del) Sucesso: MainList Falha: DeleteContactErrText			
ShowContactText	Texto dinâmico			
	Rotina: showContact ($id) Próxima: MainList			
AddContactErrText	Texto dinâmico			
	Rotina: getUpdateErrMsg ($code) Próxima: MainList			
DeleteContactErrText	Texto dinâmico			
	Rotina: getDeleteContactErrMsg ($code) Próxima: MainList			

Tabela 6.2: Mapeamento final da agenda de contatos

opção *Build options...* disponível no menu *Project*. É recomendado que este arquivo esteja dentro do diretório criado por cada gerador dentro da pasta do projeto.

O segundo método geralmente será preferido por usuários da ferramenta dada sua praticidade. Exatamente por isso foi escolhido neste exemplo. Dentro da pasta WML foi criado o arquivo `MyRoutine.java`, derivado da classe `Routine` definida no arquivo `Routine.java`. O mesmo foi feito na pasta J2ME.

Tendo todo o código pronto, segue-se a etapa de entrega para cada plataforma de destino. Para a aplicação WAP será necessário publicar os arquivos gerados no servidor Web (neste trabalho, o Tomcat [74]). A aplicação J2ME precisará ser compilada primeiro, utilizando um JDK (ou equivalente) adequado, e depois instalado no dispositivo de destino.

As figuras 6.4, 6.5, 6.6 e 6.7 ilustram a execução da agenda de contatos em emuladores das duas plataformas geradas. Nas figuras 6.4 e 6.6 temos a listagem e exibição de contatos armazenados na agenda. As figuras 6.5 e 6.7 mostram os passos utilizados para a inclusão de um novo contato.

Figura 6.4: Aplicação 1 - Listando contatos (WML)

O código fonte externo implementado para esta aplicação foi de apenas 122 linhas, além de mais 89 linhas geradas automaticamente pela ferramenta. Para a aplicação WML foram gerados automaticamente 9 arquivos JSP num total de 327 linhas. Foram geradas também 526 linhas de código para a aplicação J2ME.

6.2 Aplicação 2: *M-Commerce* B2C

6.2.1 Especificação da aplicação

A aplicação permitirá que um cliente faça compras dentro de um conjunto de lojas, como um shopping center, por exemplo.

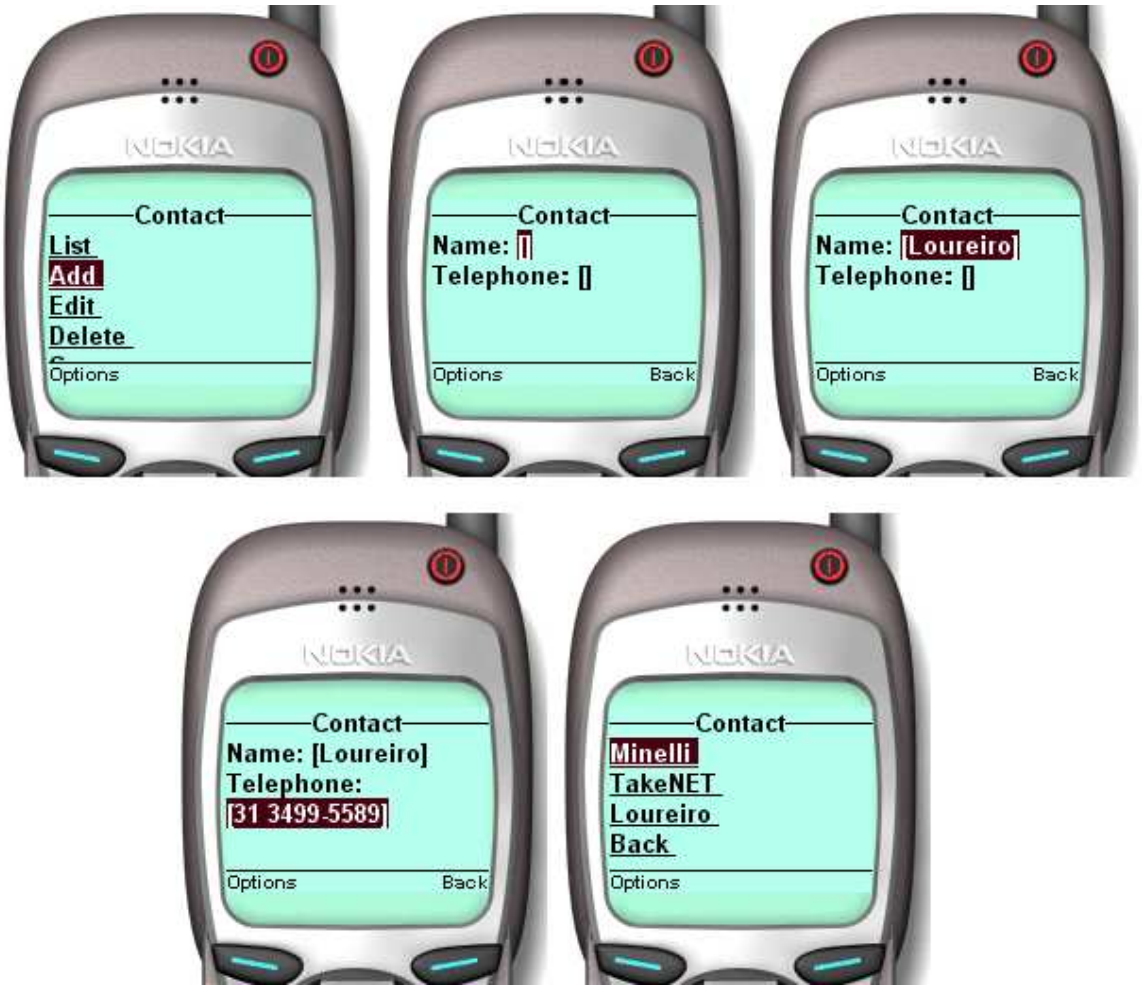

Figura 6.5: Aplicação 1 - Adicionando novo contato (WML)

Figura 6.6: Aplicação 1 - Listando contatos (J2ME)

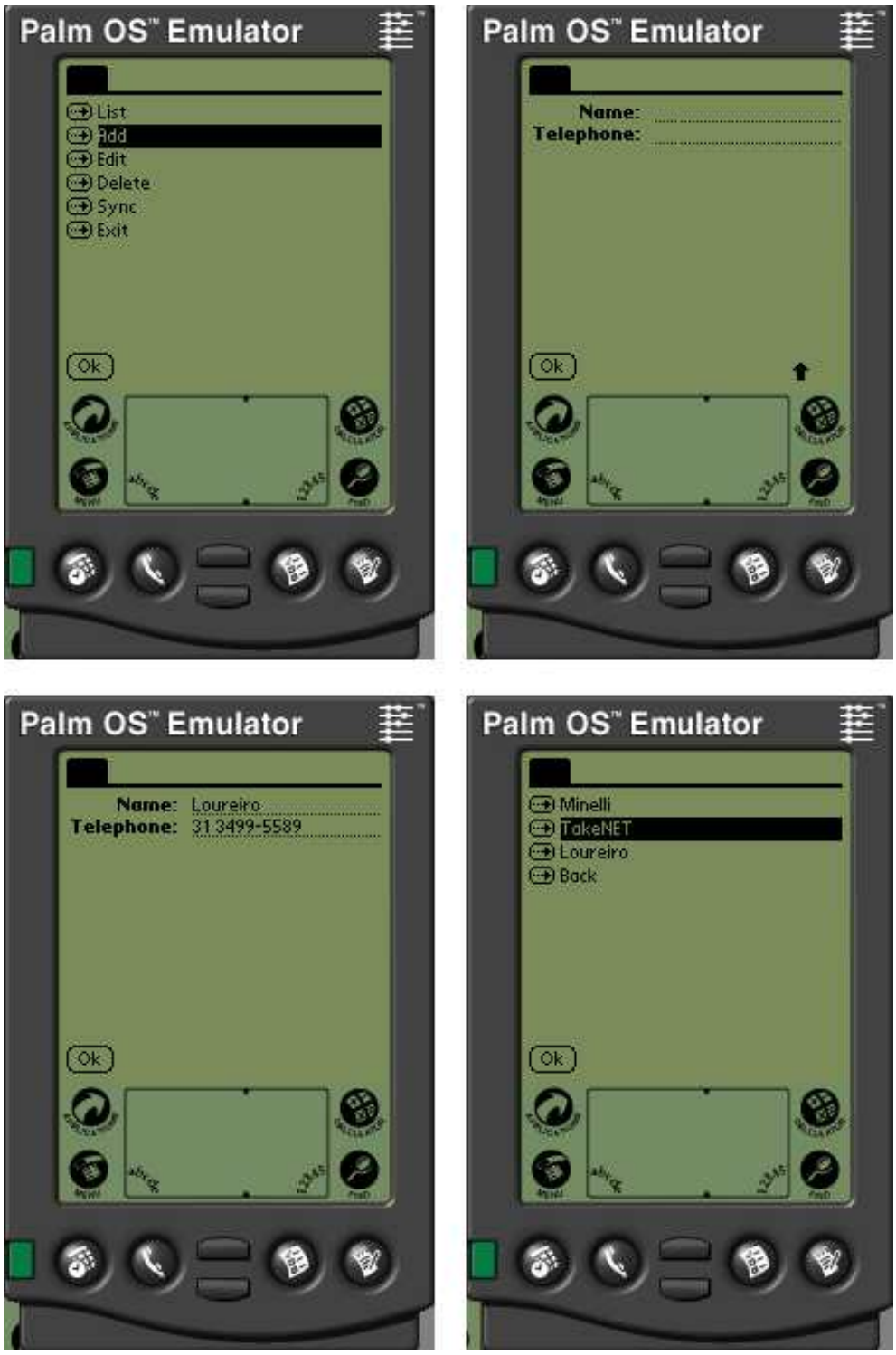

Figura 6.7: Aplicação 1 - Adicionando novo contato (J2ME)

Para tanto cada comprador deverá cadastrar-se, informando dados básicos de identificação e escolhendo um perfil segundo suas preferências. Poderão existir perfis pré-configurados ou o usuário poderá ajustar seu próprio perfil.

Feito o cadastro, o usuário iniciará a utilização da aplicação identificando-se previamente. A partir de então, poderá realizar consultas sobre os produtos disponíveis. As consultas possíveis, potencialmente, dependerão do perfil do usuário. Serão basicamente feitas a partir de características de cada produto como por exemplo código identificador, loja, fabricante, categoria, preço, ou ainda uma combinação destas. Detalhes dos produtos consultados, como formas de pagamento e descrição do fabricante, poderão ser mostrados ao usuário. Este poderá então incluir um determinado produto em sua "cesta de compras", indicando ainda a quantidade desejada.

Por fim, o usuário poderá confirmar sua intenção de adquirir os produtos em sua cesta, efetuando realmente a compra.

Novamente foi realizada uma etapa de modelagem utilizando-se UML. O diagrama de casos de uso e de classes obtidos são mostrados nas figuras 6.8 e 6.9, respectivamente.

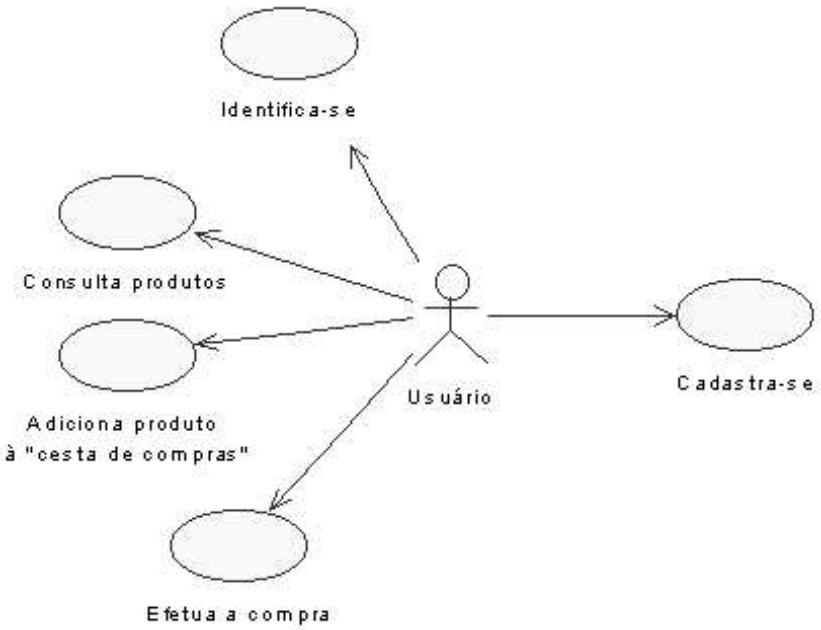

Figura 6.8: Aplicação 2 - Diagrama de Casos de uso

6.2.2 Mapeamento das unidades

Tendo a definição e modelagem da aplicação, é necessário mapeá-la em uma ou mais unidades do MAB. Após realizar algumas etapas sucessivas de mapeamento e posterior otimização, foi obtido o mapeamento final mostrado na tabela 6.3.

Unidades	Descrição da unidade			
WelcomeText	Texto estático: Welcome to Shop, a sample application for MAB			
	Próxima: LoginForm			
LoginForm	Formulário			
	Descrição	Nome	Valor	Tipo
	CPF	cpf		String
	Password	password		String
	Rotina: validate($cpf,$password) Sucesso: MainList Falha: LoginErrText			
MainList	Lista estática			
	Descrição	Ação	Parâmetros	
	Search product	SearchList		
	Add product to cart	AddProductForm		
	Close your cart	FinishForm		
	Exit	ConfirmForm		
LoginErrText	Texto dinâmico			
	Rotina: showLoginErrMsg ($code) Próxima: LoginForm			
SearchList	Lista estática			
	Descrição	Ação	Parâmetros	
	By store	ChoosingForm	type=store	
	By manufacturer	ChoosingForm	type=manufacturer	
	By cathegory	ChoosingForm	type=cathegory	
AddProductForm	Formulário			
	Descrição	Nome	Valor	Tipo
	Product ID	prodid		String
	Rotina: (none) Sucesso: ProductConfirmationForm Falha: (none)			
ProductConfirmationForm	Formulário			
	Entrada: retrieveProduct ($prodid, name, store, manufacturer, price)			
	Descrição	Nome	Valor	Tipo
	Product ID	prodid	$prodid	String
	Name	name	$name	String
	Store	store	$store	String
	Manufacturer	manufacturer	$manufacturer	String
	Price	Price	$price	String
	Rotina: addProductToCart Sucesso: MainList Falha: GenericSearchErrText			
FinishList	Lista dinâmica			
	Rotina: showAvailabePayments (mostra formas de pagamento disponíveis) Obs: -cada item da lista conterá um parâmetro *paytype*, correspondendo ao identificador daquele tipo de pagamento -a ação para todos os itens será ConclusionText, que receberá o parâmetro *paytype*			

Tabela 6.3: Unidades mapeadas no MAB - projeto Shop

<table>
<tr><td>ConclusionText</td><td colspan="4">Texto dinâmico</td></tr>
<tr><td></td><td colspan="4">Entrada: closeCart($paytype)</td></tr>
<tr><td></td><td colspan="4">Rotina: showConclusionMsg ($code)
Próxima: MainList</td></tr>
<tr><td>ConfirmForm</td><td colspan="4">Formulário</td></tr>
<tr><td></td><td>Descrição</td><td>Nome</td><td>Valor</td><td>Tipo</td></tr>
<tr><td></td><td>Clear your cart and exit?</td><td>exit</td><td></td><td>Booleano</td></tr>
<tr><td></td><td colspan="4">Rotina: logout($exit)
Sucesso: !EXIT
Falha: WelcomeText</td></tr>
<tr><td>ChoosingForm</td><td colspan="4">Formulário</td></tr>
<tr><td></td><td>Descrição</td><td>Nome</td><td>Valor</td><td>Tipo</td></tr>
<tr><td></td><td>With this name (type nothing for all)</td><td>key</td><td></td><td>String</td></tr>
<tr><td></td><td colspan="4">Rotina: (none)
Sucesso: DisplayTypesList
Falha: (none)</td></tr>
<tr><td>DisplayTypesList</td><td colspan="4">Lista dinâmica</td></tr>
<tr><td></td><td colspan="4">Rotina: getSearchResults($type,$key)
(mostra resultados da busca, de acordo com o tipo – store, manufacturer ou cathegory – e a chave)
Obs:
-cada item da lista conterá os parâmetros id e type, correspondendo ao identificador e tipo da lista
-a ação para todos será ShowProductLis</td></tr>
<tr><td>ShowProductsList</td><td colspan="4">Lista dinâmica</td></tr>
<tr><td></td><td colspan="4">Rotina: showProducts($type,$id)
(mostra produtos)
Obs:
-cada item da lista conterá um parâmetro prodid, correspondendo ao identificador do produto
-a ação para todos os itens será ProductConfirmationForm</td></tr>
<tr><td>GenericSearchErrText</td><td colspan="4">Texto dinâmico</td></tr>
<tr><td></td><td colspan="4">Rotina: showSearchError ($code)
Próxima: SearchList</td></tr>
</table>

Tabela 6.3: Unidades mapeadas no MAB - projeto Shop (continuação)

6.2.3 Codificação na ferramenta

Dado o mapeamento, usa-se MAB para inserir as unidades obtidas. A figura 6.10 ilustra o projeto final, desenhado na ferramenta e salvo com o nome *Shop*.

6.2.4 Inserção de código externo

Finalmente o projetista deverá inserir seu código, implementando as assinaturas de cada rotina definida no projeto. Porém, neste exemplo, não foi utilizado o gerador para J2ME, uma vez que seria necessário todo um mecanismo de sincronização de base de dados, não implementado neste trabalho. Embora já existam PDAs e até mesmo celulares com suporte a J2ME[2], a capacidade de memória disponível atualmente não passa de poucos megabytes, o que impossibilita armazenar uma base como esta, salvo o caso de otimizações. Desta Assim, foi implementada apenas uma sub-classe da classe de rotinas criada pelo gerador para WML.

Para armazenar e gerenciar as informações referentes à aplicação (produtos, lojas, usuários, etc.), foi criado um banco de dados em um servidor SQL Server [76] usando o esquema relacional mostrado na figura 6.11. A seguir as tabelas foram povoadas com uma massa de dados fictícia para fins de demonstração. Como o acesso a dados utilizando XML proposto na arquitetura não foi implementado neste trabalho, foram utilizadas conexões usando JDBC [77] nesta implementação.

Uma vez que o código esteja pronto, segue-se a etapa de entrega para a plataforma de destino. Neste caso, os arquivos foram publicados no servidor Tomcat. As figuras 6.12 e 6.13 ilustram a sequência de interações com a aplicação para a compra de um produto.

A figura 6.12 exemplifica uma compra através de navegação pela aplição. Inicialmente é mostrada a tela de boas vindas, que passa para o diálogo de autenticação. Um vez bem-sucedida esta autenticação, o cliente executa uma busca por categoria de produto, inserindo a palavra *software*. São exibidos então os fabricantes disponíves. Escolhido um fabricante, os produtos disponíveis são mostrados. Um produto é escolhido, após o diálogo de detalhamento e confirmação, para então a compra ser efetivada.

A figura 6.13 como seria uma compra feita diretamente pelo código (identificador) de um produto. Após a etapa de autenticação (não mostrada), o diálogo de detalhamento e confirmação é mostrado. Em seguida a compra é efetivada como no exemplo anterior.

O código fonte externo implementado para esta aplicação foi de 683 linhas, além de mais 112 linhas geradas automaticamente pela ferramenta. Foram gerados automaticamente 22 arquivos JSP num total de 855 linhas de código.

[2] dispositivos baseados no PalmOS e celulares Siemens S45 e Motorola T720, por exemplo

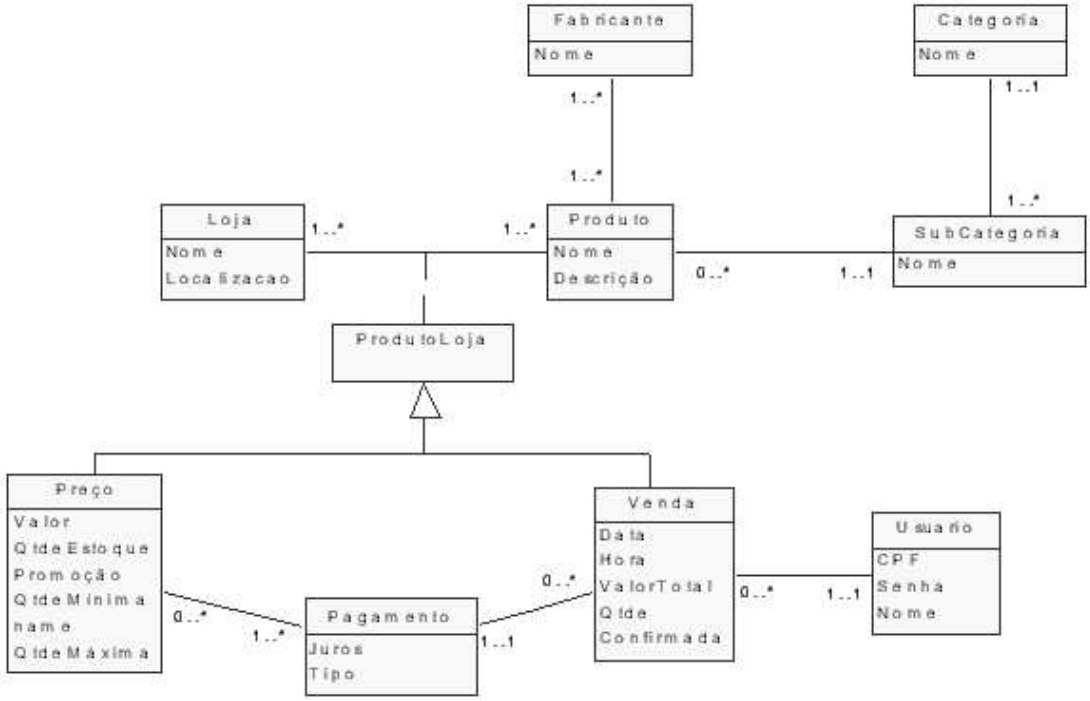

Figura 6.9: Aplicação 2 - Diagrama de Classes

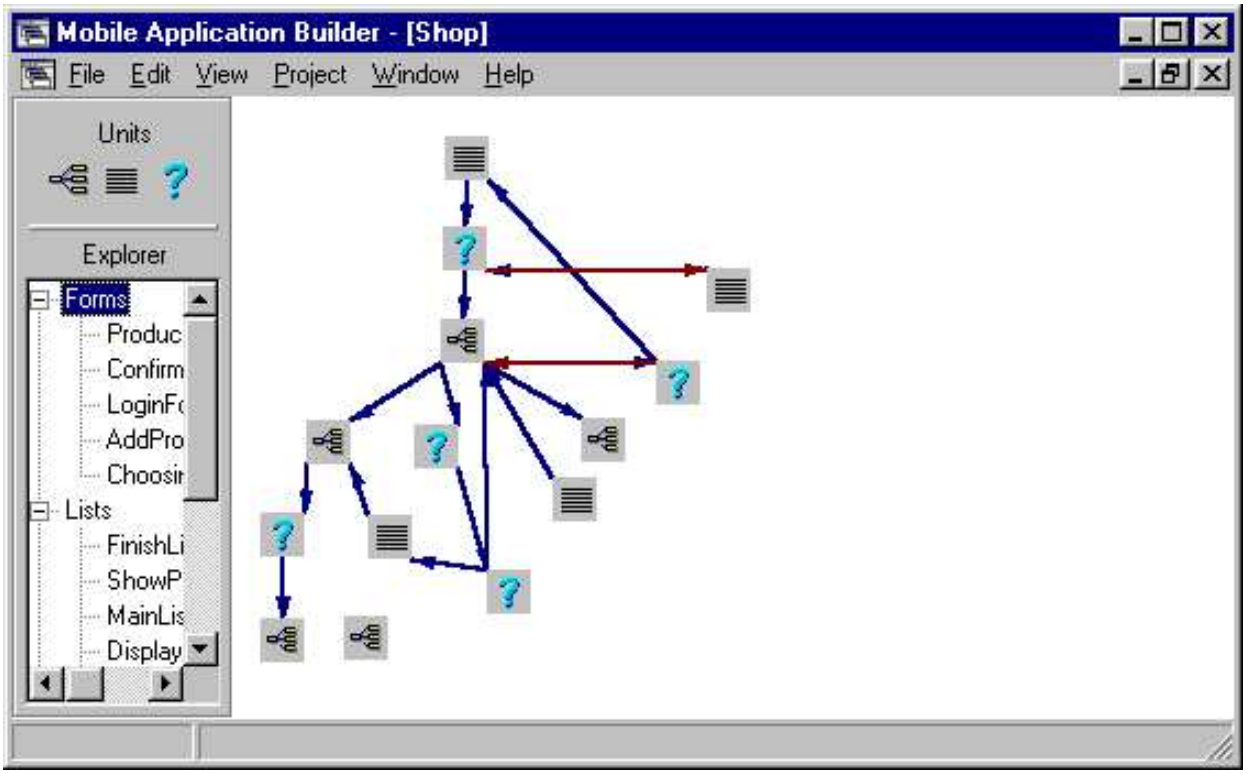

Figura 6.10: Unidades mapeadas no MAB - projeto Shop

Figura 6.11: Aplicação 2 - Esquema relacional

6.3 Comentários finais

Desenvolver aplicações utilizando MAB é bastante simples para um programador Java experiente. Apesar de relativamente recentes, JSP e J2ME serão bastante familiares.

Uma característica interessante da ferramenta está na possibilidade de reutilização de unidades. Muitas vezes é possível agrupar o processamento de duas ou mais unidades, com similaridades funcionais ou de interação, em uma única unidade parametrizada. É basicamente a aplicação do princípo da generalização/especialização, comumente utilizada em programação orientada a objetos.

Por fim pode-se observar que, para aplicações mais complexas em J2ME e que exijam uma capacidade de armazenamento maior, será necessário utilizar mecanismos otimizados de armazenamento interno e de sincronização dos dispositivos. Desta forma, será ainda necessário quebrar um pouco da regularidade do código externo, que deverá tratar de maneira diferente o acesso a dados de uma aplicação J2ME e o acesso feito por uma aplicação WML através de JSP. Ou seja, não será possível utilizar o mesmo código de rotinas externas em todas as plataformas.


Shop
Welcome to Shop, a sample application for Mobile Application Manager
Options
Shop
CPF: [1234567890]
Password: [mypass]
Options
Back
Shop
Search product
Add product to cart
Close your cart
Exit
Options
Shop
By Store
By Manufacturer
By Cathegory
Options
Back
Shop
With this name (type nothing for all):
[software]
Options
Back
Shop
Borland
Microsoft
Symantec
Back
Options
Shop
Delphi 5 Developer Edition
JBuilder 5 Enterprise Edition
Options
Back
Shop
Product ID: [2]
Name:
[Delphi 5 Develope...]
Store: [R2V2]
Options
Back
Shop
Manufacturer:
[Borland]
Price: [150]
Accept: [Yes]
Options
Back
Shop
Search product
Add product to cart
Close your cart
Exit
Options
Shop
Money
Credit card
Check
Options
Back
Shop
1 product
Payment type: money
Total: $150.00
Please, go to the
Options

Figura 6.13: Aplicação 2 - Comprando um item diretamente pelo identificador

Capítulo 7

Conclusões e trabalhos futuros

A convergência entre computação móvel e telecomunicações, com destaque para a telefonia celular e a Internet, criou um desafio para os desenvolvedores de aplicações: a diversidade de plataformas e dispositivos, e a variedade de ferramentas e ambientes de desenvolvimento.

Embora já existam propostas que procurem facilitar o desenvolvimento de aplicações para dispositivos móveis, na maioria é trabalhado apenas um domínio específico. Os produtos geram apenas linguagens de marcação, seja WML, HTML, cHTML e outras, geralmente para celulares, ou então geram apenas código nativo para uma determinada plataforma, utilizado em assistentes pessoais.

7.1 Conclusões

Este trabalho levantou um conjunto básico de requisitos para uma ferramenta que pudesse ajudar na criação de aplicações para os dispositivos móveis existentes e para aqueles que poderão surgir com a chegada da novas tecnologias, como por exemplo 3G. Baseada num subconjunto destes requisitos, a ferramenta MAB foi especificada e implementada.

Baseada na geração de código em Java, esta ferramenta mostrou ser capaz de gerar aplicações para boa parte dos dispositivos móveis existentes atualmente. A mesma aplicação criada pela ferramenta atende a celulares compatíveis com WAP 1.1, servidos através de documentos JSP, e PDA's, desde que tenham suporte à J2ME. Nos testes realizados com MAB pode-se observar que aplicações relativamente complexas podem ser implementadas, como a aplicação de *m-commerce* proposta no capítulo anterior.

MAB tem como vantagem adicional a capacidade de criar protótipos rapidamente. Após especificar uma aplicação, a ferramenta criará toda a interface para as plataformas de destino. Desta maneira bastará ao projetista a criação de *stubs* nos pontos de inserção de código externo.

A extensibilidade de MAB também é uma característica importante. Geradores de código para outras plataformas podem ser desenvolvidos independentemente e depois in-

corporados à ferramenta. Mais ainda, é possível gerar qualquer tipo de código. Java foi a opção neste trabalho, mas não há nada que impeça um desenvolvedor de criar um gerador para código nativo de uma determinada plataforma, por exemplo.

7.2 Trabalhos futuros

No desenvolvimento deste trabalho foram encontrados vários temas que poderão subsidiar estudos e trabalhos futuros. A seguir são descritos brevemente estes temas:

Estudo de usabilidade A usabilidade tanto da ferramenta em si quanto das aplicações geradas para cada plataforma de destino podem ser objeto de estudo. Modelos de usabilidade podem ser propostos e incorporados.

Personalização de interfaces de destino Seria interessante que o projetista pudesse fazer uma personalização da interface final gerada pela ferramenta. A princípio essa tarefa estaria a cargo de cada gerador, pois MAB não possui qualquer conhecimento das plataformas de destino e suas facilidades. Apesar disso podem ser criadas extensões com essa finalidade. Por exemplo, recursos como anotações (marcações com semântica especial feitas dentro de comentários) poderiam ser usados com esse objetivo.

Personalização do usuário Mecanismos que permitam que a interface da aplicação seja personalizada de acordo com o perfil do usuário podem ser incluídos na ferramenta. Poderia permitir a personalização por usuário ou por perfis pré-existentes de determinadas características da aplicação.

Formulários dinâmicos Uma vez que unidades do tipo texto e lista possuem essa opção de serem geradas dinamicamente, provavelmente será interessante que um formulário também possa ser gerado da mesma maneira.

Novos geradores Existem várias outras plataformas e tecnologias disponíveis que podem ser incorporadas à ferramenta através de criação de novos geradores. Por exemplo, poderia ser criado um gerador para VoiceXML utilizando JSP, ou um gerador para aplicações utilizando SMS em Java, para citar apenas algumas possibilidades.

***Engines* SyncML** Uma contribuição muito útil seria o desenvolvimento de um mecanismo que implementasse o protocolo definido por SyncML. Embora tenha sido definido na arquitetura a utilização de SyncML para as tarefas de sincronização, sua implementação por si só equivaleria a um trabalho de mestrado (dadas os vários tipos de sincronização e tratamentos possíveis). Na implementação atual da ferramenta, este mecanismo deverá possuir um *engine* J2ME no cliente bem como um *engine* no servidor fixo.

Encapsulamento de acesso a SGBD's Como foi definida a utilização de XML para acesso a dados na arquitetura da ferramenta, mecanismos que permitam o acesso a dados de maneira semelhante a JDBC seriam bastante úteis. Embora a utilização de XML favoreça a manipulação de dados, uma API baseada em um subconjunto de JDBC pode ser criada, favorecendo a regularidade de programação para o projetista. Essa interface seria particularmente interessante para aplicações em J2ME, onde o armazenamento é feito através de um mecanismo simples baseado em registros.

Importação de outros formatos Por fim, para aumentar a extensibilidade da ferramenta, seria interessante se esta pudesse ler diretamente modelos de outros formatos. Por exemplo, especificações em UML feitas no Rose[1]

Suporte a outros paradigmas A utilização de agentes móveis, por exemplo, seria bastante interessante devido às restrições encontradas nos dispositivos. Um agente poderia migrar para a plataforma fixa quando a bateria se esgotasse, ou retornar o resultado de um processamento quando o dispositivo se conectasse novamente à rede sem fio.

[1] ferramenta de modelagem e desenvolvimento desenvolvida pela Rational.

Bibliografia

[1] PALEN, Leysia, SALZMAN, Marilyn and YOUNGS, Ed. *Going Wireless: Behavior & Practice of New Mobile Phone Users*

[2] WEISER, M. *The computer for the 21st century.* 1991, Sci. Am. 265, 3 (Sept.), 94-104

[3] WAPMetrics
http://uolwap.uol.com.br/metrics1.php

[4] *How to Build a Wireless Office: The Next Wireless Revolution*
http://www.gartnertechwatch.com/public/static/techwatch/sponsor_100900.html

[5] KOTZ, D., et al. *Mobile Agents for Mobile Internet Computing*, July/August 1997, IEEE Internet Computing, vol 1, no 4, pp 58-67.

[6] Aglets
http://www.trl.ibm.com/aglets/index_e.htm

[7] MAHMOUD, Qusay H. *MobiAgent: A Mobile Agent-based Approach to Wireless Information Systems*

[8] MILOJICIC, Dejan S., LAFORGE, William and CHAUHAN, Deepika. *Mobile Objects and Agents (MOA)*

[9] *XML - Extensible Markup Language*
http://www.w3.org/XML/

[10] Palm Brasil - Imprensa 2001//http://www.palm.com/br/press2001/102901.html

[11] *WAP - Wireless Application Protocol*
http://www.wapforum.com/what/index.htm

[12] *Open Mobile Alliance*
http://http://www.openmobilealliance.org/

[13] RISCHPATER, R. *Desenvolvendo Wireless para Web.* Makron Books, 2001.

[14] Openwave Systems Inc.
http://www.openwave.com/

[15] Access Company Ltd.
http://www.access.co.jp

[16] Compact HTML for Small Information Appliances
http://www.w3.org/TR/1998/NOTE-compactHTML-19980209/

[17] *VoiceXML Forum*
http://www.voicexml.org

[18] Palm OS Platform
http://www.palmos.com/

[19] Microsoft Windows CE
http://www.microsoft.com/windows/embedded/ce.net/

[20] Symbian OS/EPOC
http://www.symbian.com/technology/symbos-v6x-det.html

[21] Embedded Linux Consortium
http://www.embedded-linux.org/

[22] Metrowerks Codewarrior
http://www.codewarrior.com/

[23] AppForge
http://www.appforge.com/

[24] NS Basic Corporation
http://www.nsbasic.com/

[25] Wabasoft
http://wabasoft.com/

[26] *Pumatech Satellite Forms*
http://www.pumatech.com/Satellite_Forms_Enterprise.html

[27] *PDA Toolbox*
http://www.pdatoolbox.com/

[28] *Hotsync Manager*
http://www.palmos.com/dev/tech/conduits/

[29] *Microsoft Platform Builder*
http://www.microsoft.com/windows/embedded/ce.net/previous/evaluation/tools/overview/

[30] *Microsoft eMbedded Visual Tools*
http://msdn.microsoft.com/vstudio/device/overview.asp

[31] *Microsoft Visual Studio*
http://msdn.microsoft.com/vstudio/

[32] *Microsoft ActiveSync*
http://www.microsoft.com/mobile/pocketpc/downloads/activesync35.asp

[33] Symbian OPL
http://www.symbian.com/developer/downloads/er5oplsdk.html

[34] *Web Clipping Application*
http://www.palmos.com/dev/tech/webclipping/

[35] *Java 2 Platform, Micro Edition*
http://java.sun.com/j2me/

[36] *Binary Runtime Environment for Wireless*
http://www.qualcomm.com/brew/

[37] W3C HyperText Markup Language Home Page//http://www.w3.org/MarkUp/

[38] *Java 2 Platform Micro Edition (J2ME) Technology for Creating Mobile Devices - White Paper. Sun Microsystems*, 2000.

[39] *Applications for Mobile Information Devices - White Paper. Sun Microsystems*, 2000.

[40] *Connected Limited Device Configuration*
http://java.sun.com/products/cldc/

[41] *K Virtual Machine*
http://java.sun.com/products/cldc/

[42] *Mobile Information Device Profile*
http://java.sun.com/products/midp/

[43] *JavaTM 2 Platform Micro Edition, Wireless Toolkit*
http://java.sun.com/products/j2mewtoolkit/

[44] *Palm CLDC Reference Implementation*
http://java.sun.com/products/cldc/

[45] XHTML 1.0
http://www.w3.org/TR/xhtml1

[46] XHTML *Basic*
http://www.w3.org/TR/xhtml-basic

[47] Nokia WAP Toolkit
http://www.nokia.com/corporate/wap/sdk.html

[48] Openwave SDK
http://www.openwave.com/products/developer_products/sdk/index.html

[49] Motorola Mobile Application Development Kit
http://www.motorola.com/MIMS/PSD/products/developer/index.html

[50] MESSIAS, E. *Adaptação automática de interfaces para dispositivos WAP utilizando XSL*. Dissertação de Mestrado - DCC/UFMG, 2001.

[51] MobileDev 2.0
http://mobiledev.speedware.com/

[52] Xitami Web Server
http://www.imatix.com/html/xitami/index.htm

[53] Yospace Phone Emulator
http://www.yospace.com

[54] Apache HTTP Server Project
http://httpd.apache.org/

[55] *Microsoft Visual SourceSafe*
http://msdn.microsoft.com/ssafe/

[56] Oracle9iAS Mobile Studio
http://studio.oraclemobile.com/

[57] AvidRapidTools
http://www.avidwireless.com/AVIDRapidTools.htm

[58] *Jo! Java Server Engine*
http://www.tagtraum.com/jo.html

[59] *HyperSonic SQL*
http://hsqldb.sourceforge.net/

[60] *Mobile Web Development with ASP.NET*
http://msdn.microsoft.com/vstudio/device/mobilecontrols/default.asp

[61] HiddenMind Mobility Platform
http://www.hiddenmind.com/mobility_platform.html

[62] Everypath Studio
http://www.everypath.com/products/studio.shtml

[63] XML:DB Projects
http://www.xmldb.org/projects.html

[64] *Java Servlet Technology*
http://java.sun.com/products/servlet/

[65] *Simple Object Access Protocol*
http://www.w3.org/TR/SOAP

[66] WOLF, Joel L., YU, Philip S. *On balancing the load in a clustered web farm.* ACM Transactions on Internet Technology. Volume 1 , Issue 2 (November 2001). 231-261

[67] CARDELLINI, V., COLAJANNI, M., and YU, P. *Dynamic load balancing on Web-server systems.* 1999. IEEE Internet Comput. 28-39.

[68] *Common Object Request Broker Archictecture.* OMG, Julho de 1995.

[69] *Component Object Model Technologies*
http://www.microsoft.com/com/

[70] *Wireless Access and Terminal Mobility in CORBA Specification.* OMG, Setembro de 2002.

[71] *SyncML Protocol*
http://www.syncml.org/

[72] *JavaBeans Component Architecture*
http://www.java.sun.com/products/javabeans/

[73] TURAU, V. *Making legacy data accessible for XML applications.*

[74] *The Jakarta Project - Tomcat*
http://jakarta.apache.org/tomcat/

[75] Microsoft XML Core
http://www.microsoft.com/xml/

[76] Microsoft SQL Server
http://www.microsoft.com/sql

[77] *Java Database Connection*
http://java.sun.com/products/jdbc/

Apêndice A

Documentos de descrição de tipos

A.1 Descrição e armazenamento de aplicações

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT application (unit+)>
<!ELEMENT comment EMPTY>
<!ELEMENT enumeration (value+)>
<!ATTLIST enumeration
    id CDATA #REQUIRED
    semantic CDATA #REQUIRED
>
<!ELEMENT field (#PCDATA)>
<!ATTLIST field
    name CDATA #REQUIRED
    type (CDATA | boolean | float | integer | string) #REQUIRED
>
<!ELEMENT form (field+, routine)>
<!ELEMENT input (routine?)>
<!ELEMENT item (param)>
<!ATTLIST item
    action CDATA #REQUIRED
    description CDATA #REQUIRED
>
<!ELEMENT list (item+ | routine)>
<!ELEMENT maml (application, resource)>
<!ATTLIST maml
    id CDATA #REQUIRED
>
<!ELEMENT next (param*)>
<!ATTLIST next
    action CDATA #REQUIRED
>
<!ELEMENT fail (param*)>
<!ATTLIST next
    action CDATA #REQUIRED
```

```
>
<!ELEMENT output (routine?)>
<!ELEMENT param EMPTY>
<!ATTLIST param
    name CDATA #REQUIRED
    value CDATA #REQUIRED
>
<!ELEMENT resource (enumeration)>
<!ELEMENT result EMPTY>
<!ATTLIST result
    file CDATA #REQUIRED
>
<!ELEMENT routine (param, result*)>
<!ATTLIST routine
    id CDATA #REQUIRED
>
<!ELEMENT text (#PCDATA | routine)*>
<!ELEMENT unit (input, list?, text?, form?, output, comment, next?, fail?, comment)>
<!ATTLIST unit
    id CDATA #REQUIRED
>
<!ELEMENT value EMPTY>
<!ATTLIST value
    id CDATA #REQUIRED
    value CDATA #REQUIRED
>
```

A.2 Acesso a dados

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT data (meta,record+)>
<!ATTLIST data
    type (recordset | CDATA) #REQUIRED
>
<!ELEMENT field EMPTY>
<!ATTLIST field
    name CDATA #REQUIRED
    type (string | integer | date | boolean | float) #REQUIRED
>
<!ELEMENT meta (field*)>
<!ELEMENT record EMPTY>
<!ATTLIST record
    id CDATA #REQUIRED
    nome CDATA #REQUIRED
    sobrenome IDREF #REQUIRED
>
```

Apêndice B

Glossário de Termos

ActiveX - tecnologia desenvolvida pela Microsoft para criação de componentes de *software* independente de linguagem. Baseada no modelo COM, também desenvolvido pela Microsoft.

API - *Application Programming Interface* - um conjunto de funções e procedimentos usados para acessar os serviços de uma entidade qualquer, como por exemplo um sistema operacional, um banco de dados ou *middleware*.

ASP - *Active Server Pages* - tecnologia criada pela Microsoft que permite a inclusão de *scripts* dentro de páginas HTML, de modo a criar seu conteúdo dinamicamente.

BREW - *Binary Runtime Environment for Wireless* - plataforma para execução de aplicações em dispositivos sem fio, desenvolvida e mantida pela Qualcomm.

CGI - *Common Gateway Interface* - interface que permite a execução de programas em um servidor Web, interagindo com páginas HTML.

CHTML - *Compact Hypertext Markup Language* - linguagem de marcação utilizada pelo sistema celular I-mode (amplamente utilizado no Japão). É basicamente um subconjunto do HTML.

CLDC - *Connected Limited Device Configuration* - camada intermediária da pilha especificada em J2ME. Define o conjunto de bibliotecas e características da máquina virtual Java para um conjunto específico de dispositivos.

COM - *Component Object Model* - modelo de código binário reusável definido pela Microsoft. Possibilita o desenvolvimento de objetos de *software* que podem ser acessados por qualquer aplicação que siga este modelo.

CORBA - *Component Object Request Broker Architecture* - arquitetura para a criação de objetos de *software* distribuídos, independente de plataforma, fabricante e linguagem.

CSS - *Cascade Style Sheet* - padrão suportado pelo W3C que permite a formatação de documentos HTML.

DTD - *Definition Type Document* - documento que descreve as marcações (*tags*), estrutura e símbolos de uma linguagem de marcação.

HDML - *Handheld Device Markup Language* - linguagem de marcação utilizada para formatar conteúdo para dispositivos móveis com acesso à Internet. Linguagem proprietária, desenvolvida pela Openwave (anteriormente conhecida como Phone.com, que era anteriormente conhecida como Unwired Planet).

HTML - *Hypertext Markup Language* - linguagem de marcação utilizada na WWW para a disponibilização de conteúdo. Sua padronização é feita pelo W3C.

HTTP - *Hypertext Transfer Protocol* - protocolo, baseado na arquitetura cliente/servidor, utilizado para a transferência de dados através da WWW.

IMAP - *Internet Mail Access Protocol* - protocolo utilizado para a manipulação e gerenciamento de mensagens em um servidor remoto.

J2ME - *Java 2 Micro Edition* - versão da plataforma Java otimizada para uma variedade de dispositivos com características de processamento modestas, como *pages*, telefones celulares, *handhelds* e outros dispositivos móveis e embutidos.

JDBC - *Java Database Connection* - interface de programação que permite o acesso a bancos de dados utilizando a linguagem Java.

JDK - *Java Development Kit* - um conjunto de ferramentas e utilitários para o desenvolvimento de programas na linguagem Java. Designação criada e utilizada pela Sun para seu SDK.

JSP - *Java Server Pages* - tecnologia para criação dinâmica de conteúdo Web, que utiliza *scripts* e objetos Java criados no servidor.

KVM - *Kilo Virtual Machine* - máquina virtual que compõe a camada inferior da arquitetura J2ME. É otimizada para executar em dispositivos com recursos limitados.

LDAP - *Light-weight Directory Access Protocol* - protocolo para acesso a informações em diretórios de dados.

Middleware - camada de *software* que atua como intermediário entre uma aplicação e seu ambiente operacional. Permite a distribuição da aplicação, que pode estar em ambientes ou plataforma diferentes.

MIDP - *Mobile Information Device Profile* - camada superior da pilha J2ME. Define a interface com o usuário e métodos de armazenamento e comunicação para um conjunto específico de dispositivos.

ODBC - *Open Database Connection* - interface de programação da Microsoft que permite o acesso a fontes de dados.

PDA - *Personal Digital Assistant* - computador portátil pessoal que oferece serviços de agenda de compromissos, bloco de notas e outras funções para organização pessoal.

PHP - *PHP: Hypertext Processor* - linguagem de *script* para a criação de conteúdo dinâmico para a Web. É executada no servidor e embutida dentro de marcações em páginas HTML.

POP3 - *Post Office Protocol version 3* - protocolo que permite o acesso a mensagens em um servidor remoto.

PQA - *Palm Query Application* - aplicação desenvolvida para a plataforma PalmOS. É uma versão compilada de um conjunto de páginas HTML especialmente escritas para acessar um serviço através da Internet. Ver WCA.

SDK - *Software Development Kit* - um conjunto de ferramentas e utilitários para o desenvolvimento de aplicações de *software*.

SGBD - Sistema de Gerenciamento de Banco de dados - conjunto de programas que permite criar e manipular bases de dados.

SMS - *Short Message Service* - serviço geralmente disponível em redes de telefonia celular para envio ou recebimento de mensagens de texto entre 160 e 255 caracteres.

SOAP - *Simple Object Access Protocol* - protocolo para a comunicação entre aplicações através da Internet, através de troca de documentos XML.

SyncML - *framework* para sincronização de dados entre um cliente e um servidor, através de troca de documentos XML.

VoiceXML - *Voice Extensible Markup Language* - linguagem de marcação que descreve aplicações baseadas em comandos de voz.

W3C - *World Wide Web Consortium* - organização sem fins lucrativos responsável pela padronização na Web.

WAP - *Wireless Application Protocol* - protocolo para acesso a informações da Internet através de redes sem fio, utilizando dispositivos móveis como celulares, *pagers* e PDA's.

WCA - *Web Clipping Application* - anteriormente conhecida como PQA, é uma arquitetura onde uma aplicação cliente em um dispositivo executando o sistema operacional PalmOS recebe conteúdo baseado em um sub-conjunto do HTML através de um *proxy* através da Internet.

WML - *Wireless Markup Language* - linguagem de marcação utilizada como interface com o usuário e para acesso a conteúdo por dispositivos WAP.

XHTML - *Extensible Hypertext Markup Language* - versão do HTML 4.0 compatível com a especificação XML.

XML - *Extensible Markup Language* - linguagem de marcação que permite que a estrutura do documento seja definida de modo a representar informações de maneira organizada e específica para cada situação.

XSLT - *Extensible Style Sheet: Transformation* - padrão que define como transformar um documento XML em outro documento XML com uma estrutura diferente.