SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Desenvolvimento de aplicações multi-plataformas para dispositivos móveis

Jane Dirce Alves Monteiro

Orientadora: *Profa. Dra. Maria da Graça Campos Pimentel*

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP como parte dos requisitos para obtenção do título de Mestre em Ciências de Computação e Matemática Computacional.

USP – São Carlos
Maio/2006

Desenvolvimento de aplicações
multi-plataformas para dispositivos móveis

Jane Dirce Alves Monteiro

DEDICATÓRIA

Dedico esse trabalho aos meus pais, Donaldo e Fátima, pelo incentivo que me deram para chegar até aqui, sempre me apoiando em todos os momentos, e ao meu namorado Pedro pelo carinho, apoio e dedicação.

AGRADECIMENTOS

Agradeço a Deus por ter me dado forças e me ajudado a não desistir diante das dificuldades encontradas durante a realização deste trabalho.

Agradeço à minha família, em especial aos meus irmãos Tânia e Marcos, pelo carinho e apoio que me dedicaram durante toda minha vida, principalmente neste período que passei longe de casa.

Agradeço à professora Maria da Graça Campos Pimentel, minha orientadora, pelo ótimo trabalho realizado.

Agradeço à professora Renata pela atenção, conselhos e contribuições dadas a este trabalho.

Agradeço todos os meus amigos Silvana, Claudia, Renata Porto, Carlos (Patrão), Elaine, Taciana, Renato, Izabella e demais colegas do Intermídia.

Agradeço ao Projeto Tidia-Ae e à FAPESP pela oportunidade, pelo aprendizado proporcionado e pelo apoio financeiro.

Agradeço ao ICMC-USP e aos seus professores e funcionários pelo apoio na realização deste trabalho.

E a todos que contribuíram direta ou indiretamente para a realização desse trabalho, ou simplesmente conviveram comigo durante este período.

RESUMO

Devido à sua constante evolução, associada ao surgimento contínuo de novas tecnologias, a computação tem se tornado cada vez mais presente no cotidiano das pessoas, sendo utilizada não só no meio acadêmico e corporativo, mas também no entretenimento e no auxílio às atividades pessoais, entre outros. Nesse contexto, vem ocorrendo um grande avanço em termos de redução de custo e de aumento da variedade de dispositivos do tipo *Personal Digital Assistants* (PDAs). Esse avanço aumenta a demanda por aplicações que explorem esses equipamentos, mas são poucas as ferramentas que auxiliam no desenvolvimento dessas aplicações. O objetivo da presente pesquisa foi investigar o desenvolvimento de aplicações para dispositivos móveis, especificamente para PDAs, com independência de plataforma. A abordagem adotada foi a extensão da ferramenta MCards, um construtor de aplicações para PDAs, com o propósito de permitir a geração de aplicações para dispositivos móveis com a utilização da tecnologia J2ME (*Java 2 Micro Edition*). Para tanto, foi necessário, inicialmente, um entendimento do protótipo disponível da ferramenta MCards realizado através da aplicação de um processo de Engenharia Reversa na ferramenta MCards, baseando-se no modelo FUSION-RE/I. Foram executadas, também, manutenções na ferramenta com a finalidade de corrigir imperfeições e problemas de execução e de melhorar suas funcionalidades. Após o processo de Engenharia Reversa e manutenção da ferramenta, foi implementado e incorporado à ferramenta MCards o módulo de geração de aplicações J2ME. O módulo de geração de aplicações J2ME transforma a especificação abstrata, gerada pela manipulação do usuário no editor visual da ferramenta MCards, em código para a tecnologia J2ME com a utilização do perfil CDC / Personal Profile, compilando o código gerado para dar origem aos arquivos da aplicação J2ME que serão utilizados para instalação no dispositivo móvel.

Palavras-Chave: Computação Ubíqua, Computação Móvel, PDA e J2ME

ABSTRACT

Mobile devices such as Personal Digital Assistants (PDAs) have grown up and number and models, while their cost has been continuously decreasing. Although this scenario indicates that there is a demand for tools that support the development of applications for mobile devices, the literature does not report platform-independent tools. The aim of this research was to investigate the development of applications for mobile devices in a platform-independent way. We have elected to extend an existing prototype tool that supported the development of Palm OS applications, MCards, so that it supported the development of applications using J2ME (Java 2 Micro Edition). We first carried out a reverse engineering process, using FUSION-RE/I, which lead to the understanding of the prototype along with the production of a new, more stable and documented version. We then investigated the process of application generation for the J2ME platform and proposed an extension to MCards that supported both Palm OS and J2ME. In the extended version, the J2ME module transforms an abstract specification, generated with the MCards visual editor, in J2ME code using the CDC/Personal profile. As a result, the extended MCards tool supports the development of applications for the two most common platforms for mobile devices, Palm Os, Pocket PC and Symbian OS.

Keywords: Ubiquitous Computing, Mobile Computing, PDA and J2ME

SUMÁRIO

1.	INTRODUÇÃO	1
1.1.	MOTIVAÇÃO	1
1.2.	PROBLEMA	2
1.3.	OBJETIVO	2
1.4.	METODOLOGIA	2
1.5.	ORGANIZAÇÃO DO TEXTO	3
2.	COMPUTAÇÃO UBÍQUA E COMPUTAÇÃO MÓVEL	4
2.1.	CONSIDERAÇÕES INICIAIS	4
2.2.	COMPUTAÇÃO UBÍQUA	5
2.2.1.	INTERFACES NATURAIS	6
2.2.2.	COMPUTAÇÃO CIENTE DE CONTEXTO	8
2.2.3.	CAPTURA E ACESSO DE INFORMAÇÕES	10
2.3.	COMPUTAÇÃO MÓVEL	11
2.3.1.	DISPOSITIVOS MÓVEIS	11
2.3.2.	PLATAFORMAS PARA PDAS	12
2.3.3.	TECNOLOGIAS DE DESENVOLVIMENTO JAVA	15
2.4.	CONSIDERAÇÕES FINAIS	20
3.	TRABALHOS RELACIONADOS	21
3.1.	CONSIDERAÇÕES INICIAIS	21
3.2.	HYPERCARD	22
3.3.	XIML	23
3.4.	LIQUIDUI	25
3.5.	TERESA	26
3.6.	MOBIFORMS	28
3.7.	SIMPLICITY	28
3.8.	CONSIDERAÇÕES FINAIS	29
4.	MCARDS	31
4.1.	CONSIDERAÇÕES INICIAIS	31
4.2.	CONCEITO DE CARDS	31
4.3.	A FERRAMENTA MCARDS	33
4.4.	EDITOR VISUAL	34

4.5.	GERAÇÃO DE CÓDIGO _____	36
4.6.	CONTROLE DE MENSAGENS _____	37
4.7.	CONSIDERAÇÕES FINAIS _____	38

5. MCARDS 2.0: ENGENHARIA REVERSA E EVOLUÇÃO DA FERRAMENTA MCARDS **40**

5.1.	CONSIDERAÇÕES INICIAIS _____	40
5.2.	ENGENHARIA REVERSA DA FERRAMENTA MCARDS _____	41
5.2.1.	CONCEITOS BÁSICOS DE ENGENHARIA REVERSA _____	41
5.2.2.	MÉTODO FUSION-RE/I _____	45
5.2.3.	APLICAÇÃO DO MÉTODO FUSION-RE/I NA FERRAMENTA MCARDS _____	48
5.3.	MANUTENÇÃO DA FERRAMENTA MCARDS _____	50
5.4.	NOVA ARQUITETURA DA FERRAMENTA MCARDS _____	53
5.5.	EVOLUÇÃO DA FERRAMENTA MCARDS _____	55
5.5.1.	CONFIGURAÇÃO E PERFIL SELECIONADO _____	56
5.5.2.	MÓDULO DE GERAÇÃO DE APLICAÇÕES J2ME _____	60
5.6.	EXEMPLO DE APLICAÇÃO GERADA COM MCARDS 2.0 _____	62
5.7.	DIFICULDADES E LIMITAÇÕES _____	66
5.8.	CONSIDERAÇÕES FINAIS _____	68

6. CONCLUSÃO **70**

6.1.	DISCUSSÃO DOS RESULTADOS _____	70
6.2.	TRABALHOS FUTUROS _____	72

REFERÊNCIAS **74**

GLOSSÁRIO DE TERMOS **76**

APÊNDICE A. ENGENHARIA REVERSA DA FERRAMENTA MCARDS 1.0 **80**

A.1.	INTRODUÇÃO _____	80
A.2.	PRIMEIRA ETAPA: RECUPERAÇÃO DAS VISÕES FUNCIONAIS _____	80
A.2.1.	DESCRIÇÃO DA FERRAMENTA _____	81
A.2.1.1.	SOFTWARE E SEUS COMPONENTES PRINCIPAIS _____	81
A.2.1.2.	DESCRIÇÃO DOS MÓDULOS _____	81
A.2.1.3.	MISSÃO DA FERRAMENTA _____	82
A.2.1.4.	TERMOS E DEFINIÇÕES _____	83
A.2.2.	REQUISITOS FUNCIONAIS E CASOS DE USO _____	83
A.2.2.1.	FUNÇÕES DA FERRAMENTA _____	83
A.2.2.2.	COMO USAR A FERRAMENTA _____	83
A.2.2.3.	INTERFACES DE USUÁRIO _____	84
A.2.2.4.	DIAGRAMA DE CASOS DE USO _____	88
A.2.2.5.	CASOS DE USO _____	88
A.2.3.	MODELO DE CICLO DE VIDA _____	96
A.3.	SEGUNDA ETAPA: RECUPERAÇÃO DAS VISÕES ESTRUTURAIS _____	99

APÊNDICE B. MANUAL DA FERRAMENTA MCARDS 2.0 **107**

B.1.	INTRODUÇÃO	107
B.2.	INSTALAÇÃO DA FERRAMENTA MCARDS	107
B.3.	UTILIZAÇÃO DA FERRAMENTA MCARDS	108
B.3.1	CRIANDO UMA NOVA APLICAÇÃO	108
B.3.2	GERANDO CÓDIGO E COMPILANDO A APLICAÇÃO	113
B.3.3	EDITANDO UMA APLICAÇÃO CRIADA ANTERIORMENTE	114
B.3.4	CRIANDO UMA NOVA APLICAÇÃO A PARTIR DE UM TEMPLATE	114
B.4.	INSTALAÇÃO DA MÁQUINA VIRTUAL	115
B.4.1	SUPERWABA	115
B.4.2	J2ME	116
B.5.	INSTALAÇÃO DA APLICAÇÃO GERADA	116
B.5.1	SUPERWABA	116
B.5.2	J2ME	117

APÊNDICE C. EXEMPLO DE CÓDIGO J2ME GERADO **118**

C.1.	APLICAÇÃO CRIADA	118
C.2.	ESTUTURA DO CÓDIGO J2ME GERADO	121
C.2.1.	CABEÇALHO E CRIAÇÃO DA APLICAÇÃO	121
C.2.2.	GERAÇÃO DE INTERFACE	122
C.2.3.	GERENCIAMENTO DE EVENTOS	126
C.2.4.	GERENCIAMENTO E ENVIO DE <i>MESSAGEML</i>	129
C.2.5.	ARMAZENAMENTO E GERENCIAMENTO DE DADOS	132
C.2.6.	GERENCIAMENTO DE PONTOS E COMPONENTE STROKE	134

LISTA DE FIGURAS

Figura 2.1: Dispositivos ubíquos: PDA, Tablet, Laptop, Lousa eletrônica e Mimio	4
Figura 2.2: EnhancedWall (NAKANISHI, SATO e KOIKE, 2002, p. 2)	7
Figura 2.3: <i>Aware Home</i> - Visão externa e interiores da casa (GATECH, 2005)	9
Figura 2.4: Exemplos de interface do Symbian OS para aplicativos de agenda e navegador de Internet	15
Figura 2.5: Exemplos de interface para celulares com telas pequenas, mostrando aplicativo para e-mail e tela de entrada	15
Figura 2.6: J2SE, J2EE e J2ME (<i>Profiles e Configurations</i>) (TOPLEY, 2002)	17
Figura 2.7: SuperWaba funcionamento	19
Figura 3.1: HyperCard	22
Figura 3.2: Telas do Supercard de edição de <i>card</i> (a) e edição de script (b)	23
Figura 3.3: Tela de criação de aplicação no LiquidUI com criação da interface gráfica (a) e visualização da especificação UIML(b)	25
Figura 3.4: Camadas e transformações realizadas pelo TERESA	27
Figura 3.5: Exemplo de interface de usuário de uma aplicação gerada no TERESA para o ambiente Web e celular	27
Figura 3.6: Desenvolvimento de uma aplicação móvel usando MobiForms	28
Figura 4.1: Exemplo de navegação de uma aplicação gerada pelo MCards (BOCCARDO, 2005, p. 38)	32
Figura 4.2: Diagrama de Caso de Uso da ferramenta MCards (BOCCARDO, 2005, p. 36)	33
Figura 4.3: Editor Visual da ferramenta MCards (BOCCARDO, 2005, p. 40)	35
Figura 4.4: Exemplo de aplicação construída no MCards	36
Figura 4.5: Exemplo de arquivo <i>forML</i> gerado pela ferramenta MCards	36
Figura 4.6: Funcionamento de JAXB no MCards (BOCCARDO, 2005, p. 43)	37
Figura 4.7: Exemplo de arquivo <i>messageML</i>	38
Figura 5.1: Engenharia Progressiva / Engenharia Reversa	42
Figura 5.2: Níveis de entendimento do software de acordo com o ciclo de vida	43
Figura 5.3: Categorias da Engenharia Reversa relacionadas ao ciclo de vida	44
Figura 5.4: Síntese do Método FUSION-RE/I (COSTA, 1997, p.83)	46

Figura 5.5: Nova arquitetura da Ferramenta MCards versão 2.0	54
Figura 5.6: Exemplos de interfaces possíveis com o perfil MIDP	58
Figura 5.7: Exemplos de interfaces possíveis com o perfil PP	60
Figura 5.8: Arquitetura MCards com Módulo de Geração de Aplicações J2ME	61
Figura 5.9: Módulo de Geração de Aplicações J2ME	62
Figura 5.10: Criação das telas da aplicação pra cadastro de clientes	63
Figura 5.11: Telas da aplicação pra cadastro de clientes	64
Figura 5.12: Arquivo <i>forML</i> gerado para a aplicação de cadastro de clientes	65
Figura 5.13: Mensagem a ser enviada para URL durante a execução da aplicação de cadastro de clientes	65
Figura 5.14: Trecho do código J2ME gerado para a aplicação de cadastro de clientes	66
Figura A.1: Arquitetura da Ferramenta MCards	81
Figura A.2: Tela inicial	85
Figura A.3: Tela de criação de aplicação - Editor Visual	85
Figura A.4: Exemplo de aplicação sendo criada	86
Figura A.5: Tela de criação de arquivos necessários para compilação - Deploy	86
Figura A.6: Tela About	87
Figura A.7: Tela de ajuda - Help	87
Figura A.8: Diagrama de Casos de Uso da Ferramenta MCards	88
Figura B.1: Estrutura do diretório da ferramenta	108
Figura B.2: Tela para escolha de novo projeto	108
Figura B.3: Tela inicial da ferramenta	109
Figura B.4: Menu superior: escolha de tecnologia	110
Figura B.5: Menu superior: escolha do tamanho da tela	110
Figura B.6: Menu superior: definição da URL	110
Figura B.7: Adicionando um componente	110
Figura B.8: Menu de configurações do componente "Edit"	111
Figura B.9: Menu de configuração de <i>cards</i>	112
Figura B.10: Tela de uma aplicação criada no MCards	112
Figura B.11: Salvar aplicação	113
Figura B.12: Gerar código para aplicação	113
Figura B.13: Barra de Interação	113
Figura B.14: Tela para escolha de editar um projeto	114
Figura B.15: Tela para escolha de abrir um template	114
Figura C.1: Criação das telas da aplicação pra cadastro de clientes	119

Figura C.2: Telas da aplicação pra cadastro de clientes _____	119
Figura C.3: Arquivo <i>forML</i> gerado para a aplicação de cadastro de clientes _____	120
Figura C.4: Mensagem a ser enviada para URL durante a execução da aplicação de cadastro de clientes _____	121

LISTA DE TABELAS

Tabela 2.1: Comparações de Ferramentas utilizando a plataforma Palm (Fonte: SuperWaba, 2004) _____	13
Tabela 3.1: Tabela comparativa entre ferramentas selecionadas _____	30
Tabela 4.1: Tabela comparativa entre ferramentas e ferramenta MCards _____	38
Tabela 5.1: Comparação entre J2SE 1.3.1 e CDC/Personal Profile _____	59
Tabela 5.2: Tabela comparativa entre ferramentas e as duas versões da ferramenta MCards _____	68
Tabela A.1: Termos e definições utilizados na ferramenta MCards _____	83
Tabela A.2: Descrição das Funções da Ferramenta MCards _____	83
Tabela A.3: Modelo de Ciclo de Vida da Ferramenta MCards _____	97
Tabela A.4: Relação de pacotes e arquivos do MCards _____	99
Tabela A.5: Índice de Procedimentos da MCards _____	100
Tabela B.1: Componentes x opções de menu de configuração _____	111

1. INTRODUÇÃO

1.1. Motivação

Devido à sua evolução e ao surgimento de novas tecnologias, a computação tem se tornado cada vez mais presente no cotidiano das pessoas, sendo utilizada não só no ambiente de trabalho e estudo, mas também no entretenimento e no auxílio às atividades domésticas, entre outros. Diante desse contexto, surgiu um novo paradigma: computação ubíqua, que nos trouxe a idéia de tornar os serviços computacionais tão persistentes e integrados ao ambiente, que estes se tornariam transparentes aos indivíduos (WEISER, 1991). As pesquisas realizadas em computação ubíqua apontam grandes desafios para a ciência da computação em geral, uma vez que requerem o desenvolvimento de aplicações que suportem continuamente o usuário em sua vida cotidiana (ABOWD e MYNATT, 2000). Segundo Abowd e Mynatt, existem três aspectos fundamentais que devem ser observados no desenvolvimento de aplicações ubíquas:

- Interfaces transparentes ou naturais: deve-se buscar alternativas apropriadas de interface, que possam substituir o tradicional desktop, de forma que o usuário não tenha a necessidade de mudar seus hábitos para fazer uso da aplicação;
- Sensibilidade ao contexto: a aplicação deve possuir a habilidade de modificar seu próprio comportamento baseada no conhecimento do contexto de seu uso;
- Captura automática e acesso: deve-se capturar automaticamente as experiências do usuário e possibilitar a sua recuperação ao longo do tempo.

Sob esse novo paradigma, observa-se o crescente aumento do número de dispositivos móveis, tanto na área acadêmica como no mercado corporativo e no uso pessoal. Nesse ambiente aumenta, cada vez mais, a necessidade de pesquisas e de desenvolvimento que explorem as possibilidades desse recente recurso computacional. Tais dispositivos móveis

possuem uma gama de novas explorações em campos diversos como Interação Humano-Computador, Hipermídia, Engenharia Web, entre outros.

1.2. Problema

Devido à contemporaneidade dos PDAs (*Personal Digital Assistants*), existem poucas ferramentas que auxiliam o processo de desenvolvimento para tal dispositivo, o que dificulta e, principalmente, acarreta um maior tempo de desenvolvimento. Agrava-se o problema quando se tem um mercado dominado, principalmente, por apenas duas grandes plataformas: a Palm (Palm OS) e a Microsoft (WinCe e Pocket PC), demandando ambientes de desenvolvimento diferentes, contando cada uma com linguagens de desenvolvimento específicas. Uma das soluções é a utilização de máquinas virtuais, proporcionando uma maior homogeneização no desenvolvimento. Mesmo essa abordagem demanda um elevado custo de aprendizado, e por consequência tempo de desenvolvimento. Assim, existe uma demanda por mecanismos que auxiliem no processo de desenvolvimento de aplicações de software para PDAs.

1.3. Objetivo

O objetivo do presente projeto de mestrado foi investigar o desenvolvimento de aplicações para dispositivos móveis, especificamente para PDAs, com independência de plataforma, analisando e expandindo a ferramenta MCards, um construtor de aplicações para PDAs, que foi desenvolvida durante o mestrado de Danilo Rogério Boccardo no Instituto de Ciências Matemáticas e de Computação / USP, com o propósito de permitir a geração de aplicações para dispositivos móveis com a utilização da tecnologia J2ME (*Java 2 Micro Edition*).

1.4. Metodologia

Para alcançar o objetivo proposto foi necessário, inicialmente, um entendimento do protótipo disponível da ferramenta MCards. Para tanto, foi necessária a aplicação de um processo de Engenharia Reversa na ferramenta MCards, e assim tomou-se como base o modelo FUSION-RE/I (COSTA, 1997). Além disso, foram executadas manutenções na ferramenta com a finalidade de corrigir imperfeições e problemas de execução da ferramenta e de melhorar suas funcionalidades trazendo assim, maiores benefícios para o usuário ao fazer

uso da mesma. Após o processo de Engenharia Reversa e de manutenção da ferramenta, foi implementada uma extensão para a mesma com a inserção do módulo de geração de aplicações J2ME.

1.5. Organização do Texto

Neste capítulo foram apresentados o contexto no qual se insere o trabalho, a motivação que impulsionou sua realização, seu objetivo e sua metodologia. No Capítulo 2 serão discutidos os temas Computação Ubíqua e Computação Móvel e os desafios nessas áreas; no Capítulo 3 são descritos os trabalhos relacionados à ferramenta MCards, objeto de estudo da presente pesquisa; no Capítulo 4 é detalhada a ferramenta MCards; no Capítulo 5 são descritas as atividades realizadas durante a realização deste trabalho de mestrado, incluindo o processo de Engenharia Reversa realizado para auxiliar no entendimento da ferramenta MCards, as atividades de manutenção realizadas para o melhoramento das funções da ferramenta e a evolução da ferramenta com a inserção do módulo de geração de aplicações J2ME.

2. COMPUTAÇÃO UBÍQUA E COMPUTAÇÃO MÓVEL

2.1. Considerações Iniciais

A história dos sistemas computacionais é marcada por transições de tecnologia sempre acompanhadas por novos paradigmas de interação, como se pode constatar na transição do *mainframe* para o computador pessoal. Weiser introduziu o conceito de computação ubíqua ao vislumbrar uma nova transição (WEISER, 1991). Ele previu que o paradigma teclado-mouse-display seria substituído pela proliferação de dispositivos computacionais de variados tamanhos e propósitos — pequenos e médios de uso pessoal e grandes de uso coletivo. Essa proliferação se verifica, na atualidade, com os dispositivos já relativamente comuns como PDAs (pequenos), *tablets* e laptops (médios), e lousas eletrônicas e *mimios*¹ (grandes) (Figura 2.1).

Figura 2.1: Dispositivos ubíquos: PDA, Tablet, Laptop, Lousa eletrônica e Mimio

Embora esses dispositivos ubíquos estejam cada vez mais presentes, o esforço de interação por parte do usuário vem sendo minimizado, pois tem explorado o modo como o ser humano interage com o mundo. Isso denota um novo paradigma de interação, no qual a tecnologia torna-se transparente ao usuário. Assim, recursos computacionais são incorporados aos ambientes alterando muito pouco o comportamento das pessoas e provendo facilidades

¹ *Mimio* é um digitalizador de lousa eletrônica que funciona com ultra-som.

como informações e serviços em qualquer momento e em qualquer lugar para auxiliar as atividades humanas.

Weiser define computação ubíqua como “tecnologia integrada ao ambiente porém de forma não intrusiva” (WEISER, 1993). Para prover transparência aos dispositivos computacionais, existe a necessidade de novas aplicações, geralmente associadas aos três principais temas apontados na literatura (ABOWD e MYNATT, 2000): interfaces naturais; computação ciente de contexto; captura e acesso de experiências. Pesquisas na área de computação ubíqua abordam principalmente esses três temas que, embora sejam tratados separadamente a seguir, têm forte relação entre si.

2.2. Computação Ubíqua

O objetivo principal da computação ubíqua é permitir que a tecnologia seja integrada ao ambiente que cerca o usuário sem fazer com que o usuário modifique o seu comportamento natural (ABOWD et al., 1998). Sendo assim, o usuário não tem que se deslocar para um ambiente próprio de tecnologia para se beneficiar com o uso dela.

A tecnologia de computação ubíqua pode ser explorada para gravar informações em situações cotidianas para fazer com que elas estejam disponíveis para revisão posterior. Sendo assim, tal tecnologia pode ser utilizada para gravar continuamente informações de um usuário, apesar de apenas uma pequena parte dessa informação ser fundamental para a realização de uma determinada tarefa. Uma analogia a esse comportamento pode ser feita, por exemplo, com o backup de dados de um servidor, onde os usuários geralmente precisam de apenas um sub-conjunto de arquivos.

A tecnologia de computação ubíqua propõe que muitos computadores possam ser utilizados em ambientes físicos, deixando-os transparentes para os usuários (WEISER, 1993).

Estudos antropológicos sobre o cotidiano das pessoas no trabalho (WEISER, 1993) indicam que o computador é destinado a trabalhos específicos e não integra o dia-a-dia do trabalho de muitas pessoas. Em outras palavras, o computador é visto como um “centro de atenções” e não como um outro componente qualquer dentro do ambiente de trabalho normal de uma pessoa. Segundo Weiser (WEISER, 1993) o trabalho das pessoas pode melhorar se o computador deixar de ser um “centro de atenções” e passar a estar inserido no ambiente normal de trabalho, deixando as pessoas se preocuparem com o que elas têm a fazer, sem modificar suas atitudes por causa da presença da tecnologia.

Pesquisas na área de computação ubíqua iniciaram-se em 1988 no Xerox PARC (*Palo Alto Research Center*) (WEISER, 1993). Naquela época, havia interesse na investigação de ambientes computacionais que seriam utilizados na próxima geração da computação.

A proposta era direcionada à utilização de vários computadores interligados sem fio, com os quais cada pessoa realizaria interações de modo transparente, isto é, a interação deveria acontecer sem a utilização explícita de um computador. Com isso, surgiu a necessidade de novos tipos de computadores, de diferentes formas e tamanhos, mas com capacidade e potência razoáveis. Para tais computadores, surgiram novas aplicações e novos paradigmas de interação inspirados pelo amplo e crescente acesso a informações.

2.2.1. Interfaces Naturais

A computação ubíqua demanda o desenvolvimento de aplicações cujo objetivo é a não utilização do paradigma convencional (teclado, *mouse* e *display*). Dessa forma, esforços são concentrados na construção de aplicações capazes de explorar o modo com que os humanos interagem com o mundo físico. Humanos falam, gesticulam e escrevem para se comunicarem entre si. Ações naturais como essas poderiam ser usadas, explícita ou implicitamente, como entrada para aplicações ubíquas (ABOWD e MYNATT, 2000).

Nos últimos anos, surgiram vários sistemas e dispositivos capazes de suportar formas naturais de comunicação em termos de reconhecimento de escrita, voz e gestos. Esses sistemas são caracterizados pela facilidade de aprendizagem e de uso e também são utilizados por pessoas que apresentam dificuldades de manuseio de teclado e mouse. No entanto, ainda são necessários mecanismos robustos para reconhecimento e tratamento de erros relacionados às novas interfaces.

Higel, O'Donnell e Wade (HIGEL, O'DONNELL e WADE, 2003) propõem a utilização de uma interface unificada para dispositivos com os quais usuários geralmente interagem. O sistema desenvolvido por Higel, denominado TSUNAMI, monitora implicitamente informações de usuários, tais como gestos ou comentários, e as utiliza para prever que tipo de assistência o usuário requer. Vários tipos de origem produzem dados para esse sistema. Sensores dispersos pelo ambiente fornecem dados sobre manipulação de objetos, gestos ou voz. Calendários agregam informações de tempo. A biografia do usuário auxilia a predição de fatos, baseando-se em preferências e comportamentos prévios. Lousas eletrônicas e interfaces tradicionais permitem que a entrada de dados seja feita de maneira não ambígua.

Uma vez que a predição esteja formada, uma requisição é construída e enviada para um serviço adaptativo, de modo que uma solução possa ser preparada.

Na tentativa de permitir a manipulação direta de objetos reais e projetados, Nakanishi, Sato e Koike (NAKANISHI, SATO e KOIKE, 2002) desenvolveram o EnhancedDesk, um sistema com características inovadoras, tais como rastreamento rápido e preciso da posição das mãos e dos dedos, e o cadastramento e reconhecimento de objetos com gestos manuais. O EnhancedWall (Figura 2.2), outro sistema desenvolvido pelo grupo de pesquisa de Nakanishi, é constituído por mecanismos de rastreamento de faces e por aplicações responsáveis pela interatividade com o usuário. Quando projetadas na parede, essas aplicações permitem que o usuário especifique informações de interesse, exibindo determinados itens com maior ou menor detalhamento, através de movimentos com a própria face. Esses movimentos são detectados com o uso de uma câmera de vídeo que consegue captar e interpretar para onde o usuário destina sua atenção.

Figura 2.2: EnhancedWall (NAKANISHI, SATO e KOIKE, 2002, p. 2)

Em muitos sistemas, o fato de o usuário ter que dirigir sua atenção para a interface pode comprometer aspectos de usabilidade em ambientes que possuem muitos dispositivos.

Por outro lado, os *Ambient Displays* são exemplos de sistemas que não possuem essa propriedade, ou seja, eles são projetados para conduzir informações que o usuário pode ou

não desejar obter em um determinado momento. Caracterizam-se por trabalharem na periferia da atenção do usuário, movendo-se para o centro apenas quando apropriado ou desejado.

Muitas das abordagens utilizadas em interfaces naturais utilizam informações contextuais. Os trabalhos de Higell e Nakanishi evidenciam esse fato. De um modo geral, os *Ambient Displays* também são projetados para conduzir informações de contexto para os usuários. Na próxima seção, apresentamos como essas informações podem influir no comportamento de sistemas de acordo com o conhecimento do ambiente.

2.2.2. Computação Ciente de Contexto

Segundo Dey (DEY, 2001), contexto é qualquer informação que pode ser usada para caracterizar a situação de uma entidade. Uma entidade é uma pessoa, lugar, ou objeto que é considerado relevante para a interação entre o usuário e uma aplicação, incluindo os próprios usuários e aplicações. Por exemplo, em uma casa, informações como quem entra, quando entra, o que faz e para onde vai são informações de contexto que podem ser capturadas por fontes de contexto.

Um sistema é ciente de contexto se utiliza informações de contexto para fornecer serviços relevantes para o usuário, sendo a relevância dependente das tarefas do usuário (DEY, 2001). Assim, espera-se que sistemas computacionais cientes de contexto não somente respondam quanto ao estado social e cognitivo do usuário, mas também se antecipem às suas necessidades (SIEWIOREK, 2002). Por exemplo, em uma casa que possui dispositivos cientes de contexto, pode-se identificar que um determinado morador está procurando, na biblioteca, livros sobre determinado assunto e fornecer sugestões de leitura. Ainda nesse cenário, a sala de leitura da casa pode adequar o ambiente (luz natural ou artificial, mais ou menos luz, entre outros) para as preferências de quem está lendo.

Para suportar pesquisas nessa área, a *Aware Home* (Figura 2.3) começou a ser construída em 1999 pelos pesquisadores do *Georgia Institute of Technology* (GATECH, 2005). Trata-se de um ambiente dotado de dispositivos e de serviços computacionais com capacidade para reconhecer informações sobre si mesma e sobre a localização e as atividades de seus habitantes (KIDD et al., 1999). Por exemplo, existem sensores adaptados ao solo que podem identificar e rastrear indivíduos em uma grande área. São várias as aplicações para a tecnologia de sensores, incluindo suporte para idosos ou para encontrar objetos perdidos.

Figura 2.3: *Aware Home* - Visão externa e interiores da casa (GATECH, 2005)

Entretanto, o progresso na área de sensores precisa ser acompanhado pelo desenvolvimento rápido de aplicações que utilizam informações sensoriais. Nesse sentido, Salber, Dey e Abowd (SALBER, DEY e ABOWD, 1999) desenvolveram uma infra-estrutura de software para auxiliar a construção de aplicações cientes de contexto. Trata-se do *Context Toolkit*, um conjunto de ferramentas baseado no conceito de *widgets* de contexto. Tais como *widgets* de interface estão entre a aplicação e o usuário, os *widgets* de contexto estão localizados entre a aplicação e seu ambiente de operação e, além disso, possuem características de ocultar a complexidade dos sensores utilizados pela aplicação, de abstrair informações de contexto e de prover reusabilidade.

O *Context Toolkit* foi utilizado, por exemplo, nos trabalhos de Covington et al. (COVINGTON et al., 2001) durante o desenvolvimento de um modelo de controle de acesso, visando à segurança de aplicações cientes de contexto. No caso da *Aware Home*, políticas de segurança podem restringir o acesso a informações ou recursos de acordo com vários fatores, incluindo atributos sobre o indivíduo — adultos ou crianças, proprietários ou hóspedes —, ou sobre o ambiente — temperatura ou horário. Pode haver vários cenários de uso de políticas de segurança, contanto que existam informações relevantes em um ambiente que possam ser capturadas e usadas para restringir o acesso a recursos do sistema. Uma outra aplicação que utiliza o *Context Toolkit* é o *Family Intercom* (NAGEL et al., 2001).

O protótipo inicial foi instalado na *Aware Home* com o objetivo de explorar informações de contexto para dar suporte à comunicação dentro da própria casa entre membros de uma família. O modo de interação com a aplicação é por meio de voz e, portanto, existem equipamentos de áudio espalhados pela casa, incluindo alto-falantes, microfones e

comutadores, além de sistemas de identificação de localização. O segundo protótipo foi construído para explorar a comunicação entre famílias de casas distintas. Nesse cenário, existem monitores sensíveis ao toque (com formato semelhante ao de um porta-retrato), equipamentos de áudio e um sistema para identificação de indivíduos. Através do porta-retrato digital, os membros de uma família se comunicam entre si e são capazes de perceber informações contextuais sobre a qualidade de vida de seu parente. Por exemplo, condições do ambiente, relacionamento com outras pessoas, atividades físicas e ocorrência de eventos especiais (MYNATT et al., 2001) são informações que podem indicar o bem-estar de uma pessoa idosa.

2.2.3. Captura e Acesso de Informações

Uma das características de ambientes de computação ubíqua é sua utilização visando a captura de experiências do cotidiano e tornando os registros disponíveis para acesso pelos usuários (ABOWD, 1999). Na maior parte do tempo, as pessoas encontram-se registrando, com maior ou menor precisão, os eventos que estão a sua volta. Posteriormente, de acordo com a necessidade do usuário, deseja-se que parte das informações possa ser recuperada. Nesse sentido, a utilização de ferramentas especializadas na tarefa de registrar informações possibilita que as pessoas se concentrem na síntese e compreensão da experiência propriamente dita, com confiança de que os detalhes estão sendo registrados e serão disponibilizados para futuras consultas.

Captura e acesso são definidos como armazenamento de um registro de determinada atividade em tempo real, e a possibilidade de acesso a essas informações posteriormente (TRUONG, ABOWD e BROTHERTON, 2001). Em aplicações de captura e acesso é importante conhecer tanto o espaço de projeto em que a aplicação está sendo desenvolvida, quanto à estruturação do problema a ser resolvido. Truong, Abowd e Brotherton (TRUONG, ABOWD e BROTHERTON, 2001) sugerem cinco maneiras de conhecer o espaço de projeto em que a aplicação está situada:

- Quem são os usuários durante a fase de captura e acesso;
- O que é capturado e acessado;
- Quando ocorrem a captura e acesso;
- Onde ocorrem a captura e acesso;
- Como são feitos a captura e acesso;

Similarmente Abowd et al. (ABOWD et al., 1996) identificam quatro fases de estruturação do problema de captura e acesso:

- Pré-produção: preparação de materiais para a sessão de captura;
- Gravação ao vivo: captura sincronizada dos fluxos de informação;
- Pós-produção: integração dos fluxos de informação capturados;
- Acesso: visualização dos dados capturados pelos usuários;

As experiências do cotidiano podem ser vistas como geradoras de rico conteúdo multimídia. Prover mecanismos automatizados para a captura, integração e acesso a esses registros multimídia é um dos desafios da computação ubíqua.

2.3. Computação Móvel

O crescimento ocorrido nesta década nas áreas de telefonia, redes locais sem fio e serviços via satélite permitem que informações e recursos possam ser acessados e utilizados em qualquer lugar e em qualquer momento. Dado o atual crescimento do segmento de computadores pessoais e PDAs, estima-se que em poucos anos, dezenas de milhões de pessoas terão um laptop, palmtop, ou algum tipo de PDA. Independentemente do tipo de dispositivo portátil, parte desses equipamentos deve ter a capacidade de se comunicar com a parte fixa da rede e, possivelmente, com outros computadores móveis. A esse ambiente de computação se dá o nome de computação móvel ou computação nômade.

Assim, computação móvel representa um novo paradigma computacional. Surge como uma quarta revolução na computação, antecedida pelos grandes centros de processamento de dados da década de sessenta, o surgimento de terminais nos anos setenta, e as redes de computadores na década de oitenta (MATEUS e LOUREIRO, 1998).

2.3.1. Dispositivos Móveis

Existe uma forte tendência à integração entre PDAs e celulares, sendo que já é possível se comprar celulares com função de PDAs e vice-versa. É grande a diversidade de interfaces de entrada e saída, capacidade de processamento e armazenamento, autonomia de funcionamento, e suporte a protocolos de comunicação sem fio existentes entre os dispositivos móveis, assim, a escolha de um modelo que melhor se adeque à tarefa a ser realizado é fundamental para a um bom resultado.

2.3.2. Plataformas para PDAs

A definição de plataforma se confunde muitas vezes com a de sistema operacional e de fato essas palavras podem ser vistas, de certa forma, como sinônimas no âmbito da computação móvel, pois aparelhos que utilizam a plataforma Palm, por exemplo, foram fabricados para utilizar o sistema operacional Palm OS, o mesmo ocorre com as demais plataformas existentes. A seguir são apresentadas as principais plataformas para PDAs disponíveis no mercado.

2.3.2.1. Palm

O Palm OS² é um sistema operacional desenvolvido pela PalmSource³ para os PDAs manufaturados por diversas empresas que adquiriram a licença de fabricação. A maior licenciadora do Palm OS é a Palm Inc⁴. Essa plataforma foi desenvolvida com a utilização da linguagem C, portanto C é a linguagem nativa do Palm OS e com ela o desenvolvedor pode extrair o máximo dos recursos da plataforma para escrever aplicações eficientes e compactas. No entanto, várias outras linguagens de programação podem ser usadas para o desenvolvimento em Palm OS, como C++, Pascal, Visual Basic, Lua⁵ ou Java.

Alguns programas importantes para o desenvolvimento nesta plataforma são:

- Palm OS SDK contém as bibliotecas para desenvolvimento nesta plataforma.
- Palm OS Emulator simula um dispositivo que roda a plataforma Palm OS em um computador desktop.
- Palm OS Simulator é a versão do emulador exclusivamente para a versão 5 do Palm OS ou superior, mostra também mensagens de erro de uma forma mais clara.

Outro item importante para iniciar o desenvolvimento em Palm OS é a escolha da tecnologia de desenvolvimento; para isso é preciso levar em consideração o custo dessa ferramenta, o tempo de desenvolvimento e o seu impacto na performance final da aplicação. Na Tabela 2.1 são citadas algumas ferramentas de desenvolvimento, com seus custos e descrições.

² Palm OS. [http://www.palmsource.com/Palm OS/](http://www.palmsource.com/Palm%20OS/)

³ PalmSource. <http://www.palmsource.com/>

⁴ Palm Inc. <http://www.palm.com/>

⁵ Lua é uma linguagem de programação poderosa e leve, projetada para estender aplicações. Foi desenvolvida pela PUC-Rio. <http://www.lua.org/>

Tabela 2.1: Comparações de Ferramentas utilizando a plataforma Palm (Fonte: SuperWaba, 2004)

Ferramenta	Licença	Linguagem	Velocidade	Descrição
CodeWarrior (MetroWerks)	U\$499.00	C/C++	Rápido	Era ferramenta oficial da Palm até meados de 2003. www.metrowerks.com
PocketStudio	R\$ 233,00 - Standard R\$ 829,00 - Professional	Pascal	Rápido	Uma das ferramentas mais utilizadas por desenvolvedores. www.clubepalm.com.br
Palm Developer Studio	U\$ 399.00	C/C++	Rápido	http://www.falch.net
Palm OS Developer Suite	Grátis	C/C++	Rápido	Ferramenta oficial de desenvolvimento para Palm OS indicada pela Palm Inc.
SuperWaba	Grátis	Java	Médio	VM tem uma boa performance e é muito utilizada.
Mobile VB (AppForge)	U\$ 900.00	Visual Basic	Lento	
Satellite Forms (Pumatech)	U\$ 899.00	Script	Lento	www.pumatech.com
NS Basic	U\$ 445.00	Basic	Lento	

2.3.2.2. Pocket PC

A plataforma Pocket PC⁶, de propriedade da Microsoft, vem ganhando a cada dia mais espaço no mercado de PDAs, sendo a principal concorrente da plataforma Palm. A Microsoft conta com um grupo de sistemas operacionais destinados a dispositivos móveis, como: Windows CE, Pocket PC 2000, Pocket PC 2002 e Pocket PC 2003.

A Microsoft fornece também uma solução própria para desenvolvimento para a plataforma Pocket PC, *.Net Compact Framework*⁷. Pelo fato dessa plataforma ser um subconjunto da *.Net Framework*, ela reduz o tempo de desenvolvimento de aplicações, pois os programadores podem desenvolver para PDAs de maneira similar a quando desenvolvem para PC's, utilizando um subconjunto das mesmas API's.

A *.Net Compact Framework* possui algumas API's para: Manipulação de objetos de interação como botões, campos, imagens, etc; Manipulação de I/O; Manipulação de banco de dados; Manipulação de arquivos; Manipulação de conjunto de dados; Manipulação gráfica; Manipulação de rede e Internet; e Manipulação de Web Services.

Duas outras opções de desenvolvimento para a plataforma da Microsoft são:

⁶ <http://www.microsoft.com/windowsmobile/pocketpc/>

- *Microsoft eMbedded Visual C++ Tools* possibilitando o desenvolvedor escrever código nativo em C++;
- *Microsoft Visual Studio .Net. 2003*, que possibilita a escrita de código não nativo, em C# ou *Visual Basic .Net*.

2.3.2.3. Symbian

Surgido a partir de um consórcio entre as principais empresas de telefonia móvel do mundo (Nokia⁸, Motorola⁹ e Ericsson¹⁰), a Symbian¹¹ se tornou uma empresa privada e independente em Julho de 1998, fornecendo licenças para a plataforma Symbian, cujo sistema operacional é o Symbian OS.

Em 2004, a Symbian recebeu aportes de capital de várias empresas de telefonia, e em Junho de 2004 a divisão de ações da Symbian era a seguinte: Nokia com 47.9%; Ericsson com 15.6%; Sony-Ericsson¹² com 13.1%; Panasonic¹³ com 10.5%; Siemens¹⁴ com 8.4 %; Samsung¹⁵ com 4.5%

Sua linguagem nativa é o C++ e com ela se pode extrair o máximo dos recursos dos dispositivos, possibilitando construir aplicações eficientes e compactas, podendo-se também optar por outras linguagens como Java e Visual Basic.

O Symbian OS é um sistema operacional multitarefa, aberto e criado para ser padrão para *smartphones* e celulares com recursos avançados. É um sistema totalmente modular, e permite que cada empresa crie sua própria interface. Portanto este sistema não tem uma cara definida. Pode ser um simples sistema de textos em telas monocromáticas, ou um completo sistema operacional tão potente como o Palm OS ou Pocket PC que já pode ser encontrado nos *smartphones* da Nokia e Ericsson. A Figura 2.4 mostra exemplos de interfaces, para aplicativos de agenda e navegador de Internet, desenvolvidas para celulares e *smartphones* com Symbian OS, e a Figura 2.5 ilustra exemplos de interfaces para aplicativos de e-mail e para tela de entrada de celulares Symbian OS com telas com tamanho reduzido.

⁷ <http://msdn.microsoft.com/netframework/programming/netcf/>

⁸ <http://www.nokia.com>

⁹ <http://www.motorola.com>

¹⁰ <http://www.ericsson.com>

¹¹ <http://www.symbian.com>

¹² <http://www.sonyericsson.com>

¹³ <http://www.panasonic.com>

¹⁴ <http://www.siemens.com>

¹⁵ <http://www.samsung.com>

Figura 2.4: Exemplos de interface do Symbian OS para aplicativos de agenda e navegador de Internet

Figura 2.5: Exemplos de interface para celulares com telas pequenas, mostrando aplicativo para e-mail e tela de entrada

O futuro do sistema operacional da Symbian é bastante promissor. Espera-se que seus acionistas lancem produtos cada vez mais avançados, rivalizando diretamente com o Palm OS e o Pocket PC. No momento, a empresa somente fornece sistemas operacionais para produtos com função de telefone, mas fica claro que isto não é uma regra. Portanto, em 2005, o mercado de sistemas operacionais de equipamentos portáteis está nas mãos da PalmOne, Microsoft com o Pocket PC e a Symbian correndo por fora com o Symbian OS.

2.3.3. Tecnologias de Desenvolvimento Java

Nesta seção é feita uma introdução sobre as principais tecnologias Java para desenvolvimento para PDAs existentes no mercado.

2.3.3.1. J2ME

A plataforma J2ME¹⁶ (*Java 2 Micro Edition*) é a plataforma Java para dispositivos compactos, como celulares, *smartphones*, PDAs, controles remotos, e uma outra gama de dispositivos móveis. Assim como a versão Enterprise (J2EE), a Standard (J2SE) e a Smart Card (*Java Card*), a plataforma J2ME é uma coleção de APIs do Java definidas através do JCP¹⁷ (*Java Community Process*), como ilustra a Figura 2.6. O J2ME possui o poder e os benefícios da tecnologia Java, incluindo uma interface flexível, um modelo robusto de segurança e suporte a aplicações em rede ou off-line (TOPLEY, 2002).

A arquitetura J2ME define configurações (*configurations*), perfis (*profiles*) e pacotes (*APIs*) opcionais como elementos para construir ambientes de execução completos que preenchem os requisitos para um maior número de dispositivos possível. Cada combinação é otimizada para a capacidade de memória, processamento e I/O de uma determinada categoria de dispositivos. O resultado é uma plataforma comum, que é compatível com a maioria dos dispositivos móveis do mercado.

As configurações são compostas de uma VM (*Virtual Machine*) e um conjunto mínimo de bibliotecas. Elas fornecem a funcionalidade básica para um número particular de dispositivos que possuem características similares, como conectividade e memória. Atualmente, existem duas configurações no J2ME:

- Connected Limited Device Configuration (CLDC): é a de menor capacidade entre as duas configurações, destinada a dispositivos com conexões de rede intermitentes, processadores lentos e memória limitada (como celulares, pagers, e PDAs). Esses dispositivos possuem, geralmente, processadores de 16 ou 32 bits e memória disponível entre 128KB e 512KB para a implementação da plataforma Java e suas aplicações;
- Connected Device Configuration (CDC): foi construída para dispositivos que possuem conexões mais rápidas e maior capacidade de memória e de processamento, como controles-remoto, gateways residenciais, etc. A CDC inclui uma full-featured JVM e acesso a mais interfaces do J2SE que o CLDC. A maioria dos dispositivos compatíveis com CDC possui processadores de 32 bits e um mínimo de 2MB de memória.

¹⁶ Java 2 Micro Edition. <http://www.Java.sun.com/j2me>

¹⁷ Java Community Process. <http://jcp.org/en/home/index>

Figura 2.6: J2SE, J2EE e J2ME (Profiles e Configurations) (TOPLEY, 2002)

Para fornecer um ambiente completo para cada categoria de dispositivos, as configurações devem ser combinadas com perfis, ou seja, com um conjunto de *APIs* de alto nível que definirão o modelo do ciclo de vida da aplicação, a interface do usuário e o acesso a propriedades específicas do dispositivo. Os perfis CDCs podem ser combinados, ou seja, esses perfis podem ser adicionados quando preciso para fornecer a funcionalidade para diferentes tipos de dispositivos. A seguir estão listados os principais perfis:

- Mobile Information Device Profile (MIDP): especificado para celulares e PDAs, e oferece a base para a funcionalidade requerida pelas aplicações móveis, incluindo interfaces do usuário, conexões com a rede, persistência de dados e controle de aplicações. Combinado com o CLDC, o MIDP fornece um ambiente de execução Java completo que alavanca a capacidade processamento de handhelds e minimiza o consumo de memória e energia.
- Foundation Profile (FP): é o nível mais baixo de perfil do CDC. Fornece uma implementação de rede do CDC que pode ser usada para construir aplicações sem interface com o usuário. Pode também ser combinado com o PBP (Personal Basis Profile) e o PP (Personal Profile) para dispositivos que necessitem de uma interface com o usuário.
- Personal Profile (PP): é o perfil CDC utilizado em dispositivos que necessitam de um suporte completo para interface ou *applet*, como PDAs e consoles para jogos.

Ele inclui a biblioteca AWT completa e é fiel ao ambiente Web, executando facilmente *applets* construídos para ambientes desktop.

- Personal Basis Profile (PBP): O Personal Basis Profile (PBP), uma subdivisão do PP, fornece um ambiente para dispositivos conectados que suportem um nível básico de apresentação gráfica ou necessitem do uso de toolkits específicos para aplicações.

2.3.3.2. SuperWaba

O SuperWaba¹⁸ é uma tecnologia para o desenvolvimento de aplicações para PDAs que utiliza a linguagem Java. É uma tecnologia brasileira criada no início de 2000 por Guilherme C. Hazan¹⁹, sendo derivado de outro projeto de software livre chamado Waba²⁰. Essa tecnologia é constituída de duas partes: as classes Java, que compõem a biblioteca com a qual são construídas as aplicações e a sua máquina virtual, que interpreta o código Java. O SuperWaba possui uma Máquina Virtual (SWVM), que é responsável pela inter-relação entre o PDA e a aplicação SuperWaba (SW), e aumenta a portabilidade das aplicações desenvolvidas. Assim, o SuperWaba consiste em:

- Bibliotecas de classes Java (SW API) – componentes gráficos, classes de controle do sistema operacional, entrada e saída de dados, banco de dados, criptografia, GPS (*Global Positioning System*), API especial para jogos, etc.
- Máquina virtual Java (SWVM) – desenvolvida para proporcionar bom desempenho das aplicações, portabilidade e ocupar pouca memória (308k).

Além disso, o SW possui os programas Warp e Exegen que são usados para gerar as aplicações a serem importadas para o PDA.

O funcionamento do SuperWaba não envolve somente a interpretação do código Java pela sua Máquina Virtual – existem outros processos que estão envolvidos nesta ação, o que acaba sendo uma peculiaridade dessa tecnologia.

¹⁸ SuperWaba: <http://www.superwaba.com.br/>

¹⁹ Guilherme Hazan contato: <http://www.guich.com/resume.html>

²⁰ WabaSoft : <http://www.wabasoft.com>

Figura 2.7: SuperWaba funcionamento

Os passos que correspondem à execução completa do processo de transformação do código Java em aplicações para dispositivos móveis:

- As classes Java, que compõem a biblioteca com a qual são construídas as aplicações fazem parte da API SW;
- O compilador transforma o código Java (classes) em bytecodes que são empacotados em um arquivo que possui formato pdb (Palm DataBase);
- O arquivo em formato pdb é interpretado pela máquina virtual SuperWaba quando rodado no PDA ;
- O aplicativo exegen gera os arquivos executáveis (extensão .prc) para o PDA;

O SuperWaba oferece uma biblioteca de classes Java, a qual encontra-se dividida em pacotes, os quais possuem funções diferentes, auxiliando o usuário no desenvolvimento de aplicações que atendam as suas necessidades. As principais APIs são:

- waba.ui: Controles da interface com o usuário, janelas popup, eventos.
- waba.io: Entrada e saída.
- waba.fx: Gráficos, som e fontes de caracteres.
- waba.sys: Conversão de dados, acesso à máquina virtual.
- waba.util: data, vetor e tabela hash.

2.4. Considerações Finais

Aplicações de computação ubíqua compartilham várias características funcionais. Há um grande esforço para que essas aplicações suportem mecanismos de transparência de interação, adaptem seu comportamento de acordo com as mudanças de contexto e forneçam serviços automatizados para captura de experiências ao vivo e acesso às informações capturadas. Em paralelo, tem-se observado um grande avanço no número de componentes de hardware de custo e tamanho reduzidos, favorecendo a criação de novos produtos embutidos em objetos convencionais do dia-a-dia. Enquanto o uso ubíquo da computação pode trazer benefícios, ao mesmo tempo ele impõe novos desafios no projeto de interfaces e de aplicações responsáveis por capturar informações contextuais. Isso leva a uma mudança na relação existente entre humanos e computadores. Ao fornecer modelos de interação contínua com o computador, a computação ubíqua transforma os dispositivos em uma ferramenta de presença constante no ambiente.

Neste capítulo foi apresentada uma introdução sobre a computação móvel, um breve resumo de sua história, assim como problemas relacionados a essa área. Mostrou-se também algumas formas de comunicação existentes no mundo dos dispositivos móveis, bem como as plataformas mais utilizadas em PDAs.

3. TRABALHOS RELACIONADOS

3.1. Considerações Iniciais

Devido à atual situação da heterogeneidade dos dispositivos disponíveis no mercado e com a necessidade de interfaces de usuário que, em vez de ter aplicações separadas para cada dispositivo, troca-se somente dados básicos, há um aumento no número de pesquisas que avancem na descrição abstrata de aplicações e no desenvolvimento de ambientes capazes de sugerir um projeto apropriado para dispositivos variados.

Dentro desse contexto, com a expansão significativa do número de dispositivos móveis, tanto na área acadêmica como no mercado corporativo, surge, cada vez mais, a necessidade de pesquisas e desenvolvimento fazendo uso de tais equipamentos. Devido ao tema ser relativamente novo, existem poucas ferramentas que auxiliam nesse desenvolvimento. Uma dessas ferramentas, a MCards, é um construtor de interface para PDAs com geração de código.

Neste capítulo serão descritas algumas ferramentas que apóiam a construção de interface com o usuário enfatizando, em cada uma delas, os pontos similares à ferramenta MCards. Entre os trabalhos relacionados foram selecionados:

- HyperCard, devido ao seu conceito de *cards* e tratamento de persistência;
- XIML, LiquidUI e TERESA, por serem ferramentas que possuem linguagens específicas de marcação baseadas em XML para definir a interface para diversos dispositivos;
- MobiForms e Simplicity, por serem ferramentas para desenvolvimento rápido (RAD) de aplicações Java para dispositivos móveis, especificamente PDAs.

Vale lembrar que no segmento das ferramentas que apóiam o desenvolvimento para PDAs existem diversas ferramentas como, por exemplo, *Microsoft Visual Studio*, *Palm*

Developer Studio, entre outras, mas que todavia geram aplicações para apenas um tipo de dispositivo, Pocket PC ou Palm OS, em diversas linguagens como Java, C/C++, VB e Pascal.

3.2. HyperCard

HyperCard é um ambiente de programação gráfico produzido pela Apple para rodar no sistema operacional *MacOS*²¹. Assemelha-se, conceitualmente, a uma aplicação de banco de dados, que armazena a informação, mas ao contrário dos sistemas tradicionais da banco de dados HyperCard é gráfico, muito flexível e fácil de modificar. Inclui também uma poderosa linguagem de programação, HyperTalk, para manipular dados e a interface de usuário. HyperCard é usado freqüentemente como aplicação para o desenvolvimento rápido (RAD).

HyperCard é baseado no conceito de uma "pilha" de "cartões virtuais", ou *cards* que podem incluir campos para armazenamento de dados e elementos visuais. Entre os elementos visuais possíveis num *card* estão figuras, botões, textos, campos de texto, e outros elementos comuns de uma interface gráfica. Seu sistema é similar ao conceito de "formulário" usado na maioria das aplicações de desenvolvimento (RAD), tais como Delphi²² ou Visual BASIC²³. No HyperCard pode-se também definir navegação entre os *cards* através da utilização de links de ligação. A Figura 3.1 ilustra a interface visual do HyperCard.

Figura 3.1: HyperCard

²¹ MacOS. <http://www.apple.com/macosx/>

²² Borland Delphi. <http://www.borland.com/br/products/delphi/>

²³ Microsoft Visual BASIC. <http://msdn.microsoft.com/vbasic/>

Usuários do HyperCard podem construir bases de dados com uso de editores gráficos que permitem a manipulação de itens visuais, de modo que o controle das operações relacionadas ao armazenamento e recuperação dos dados fica a cargo do ambiente, ou seja, em tempo de execução de uma aplicação criada pelo HyperCard, o usuário pode utilizar operações básicas como *search*, *add*, *delete*, proporcionando um conjunto de funcionalidades necessárias para a execução da aplicação.

Devido ao seu conceito inovador, foram surgindo várias ferramentas no mercado seguindo a mesma linha que o HyperCard, como:

- SuperCard: uma versão colorida do HyperCard para MacOS com características adicionais, um toolkit com mais componentes de interface (Figura 3.2);
- WinPlus: similar ao HyperCard com versões para Windows e MacOS;
- OracleCard: rebatizado para Oracle Media Object, usado para o acesso a bases de dados;
- Revolution: incorporou todo o HyperTalk (a linguagem de script do HyperCard), e uma parte grande das características do HyperCard, expandindo-o ao oferecer um conjunto mais avançado de componentes de interface. Compatível com MacOS, Windows, e Unix.

Figura 3.2: Telas do Supercard de edição de *card* (a) e edição de script (b)

3.3. XIIML

O XIIML²⁴ (*eXtensible Interface Markup Language*) (PUERTA e EISENSTEIN, 2006) (VANDERDONCKT, LIMBOURG e FLORINS, 2001) é uma linguagem de

²⁴ XIIML. <http://www.xiiml.org/>

representação baseada em XML. O foco principal da sua abordagem é o desenvolvimento fundamentado em modelos. XIML oferece um mecanismo padrão para o intercâmbio de dados entre ferramentas e aplicações, desde o projeto até a operação e manutenção da interface. Isto é, permite que os modelos de projeto possam ser transformados em soluções de implementação multi-plataformas.

A descrição de uma interface utilizando essa linguagem é composta de alguns detalhes em um nível abstrato e de alguns em um nível mais concreto, voltados a detalhes de implementação da interface. O XIML define três modelos que são utilizados para o desenvolvimento de interfaces:

- O modelo de tarefa é uma representação estruturada das tarefas que usuário poderá realizar na aplicação. Esse modelo é decomposto hierarquicamente em sub tarefas que contém informações relacionadas aos objetivos, as pré-condições e as pós-condições de cada uma delas.
- O modelo de plataforma representa todos os elementos de cada plataforma específica, e cada elemento, por sua vez, contém atributos que descrevem suas características e restrições.
- O modelo de apresentação detalha a aparência visual da interface. Nele são incluídas informações que descrevem a hierarquia dos elementos que compõem a interface, conhecidos como *widgets*. Esses *widgets* são definidos de acordo com uma relação entre AIOs (objetos de interação abstrata) e CIOs (objetos de interação concreta). Os AIOs são elementos que permitem ao usuário de uma aplicação visualizar e manipular a informação, enquanto os CIOs são elementos de interação que são executáveis em alguma plataforma com um processamento adicional. É interessante observar que os AIOs não são executáveis em nenhuma plataforma e, por isso, são completamente portáteis, permitindo a implementação de uma mesma interface em inúmeras plataformas.

- Java Renderer: gera as interfaces de usuário de UIML para plataforma Java (compatível apenas com JDK 1.3.x);
- HTML Renderer: gera as interfaces de usuário de UIML para HTML, e visualizar a interface gerada em navegadores Web;
- WML Renderer: gera as interfaces de usuário de UIML para WML, ou seja, interface para telefones celulares com WML;
- VoiceXML Renderer: gera as interfaces de usuário de UIML para VoiceXML, para dispositivos de voz como telefones;
- UIML Proxies: interliga as interfaces de usuário UIML com outras aplicações;
- UIML Server: disponibiliza as interfaces de usuário em UIML na Internet para acesso externo;
- UIML Browser: renderiza as interfaces de um documento UIML;

3.5. TERESA

TERESA (*Transformation Environment for inteRactivE Systems representAtions*) (MORI, PATERNÒ e SANTORO, 2003) é uma ferramenta que fornece um ambiente completo, semi-automático, para modelagem e geração de interface para aplicações, com base na linguagem XML, para vários tipos de plataforma.

Essa ferramenta, mediante uma série de transformações, divididas em quatro camadas, permite gerar interfaces para múltiplos dispositivos. A primeira camada, Figura 3.4(a), trata-se de uma descrição abstrata das atividades, mediante uma notação hierárquica de tarefas a serem executadas pelo usuário final e os objetos manipulados para tal objetivo. É utilizada a notação *ConcurTaskTrees* (MORI, PATERNÒ e SANTORO, 2004).

Após a criação desse modelo de alto nível, na segunda camada, Figura 3.4(b), muda-se o foco para objetos de interação necessários à execução das tarefas pretendidas. Para tal, é produzida uma especificação da interface compreendendo um conjunto de apresentações abstratas, cada um representado por elementos de interfaces. Cada apresentação abstrata é construída por *interactors* que são objetos de interação abstratos identificados em termos de seu significado, ou seja, a tarefa básica que assistem.

Em seguida, vem a camada de concretização dos *interactors*, Figura 3.4(c), em que são substituídos por objetos de interação concretos, de acordo com o tipo da plataforma. Tal camada permite a atribuição de diversos atributos relacionados à aparência e ao comportamento. Enfim, na última camada, Figura 3.4(d), a interface concreta é traduzida para uma determinada linguagem. A Figura 3.5 mostra uma interface de usuário de uma aplicação gerada no TERESA para o ambiente desktop e celular.

Figura 3.4: Camadas e transformações realizadas pelo TERESA

Figura 3.5: Exemplo de interface de usuário de uma aplicação gerada no TERESA para o ambiente Web e celular

3.6. MobiForms

MobiForms²⁶ é uma ferramenta para desenvolvimento rápido (RAD) de aplicações Java para dispositivos móveis que utiliza tecnologia J2ME. MobiForms possui um ambiente gráfico para desenvolvimento de interface usando a técnica *drag and drop*. Adicionalmente, essa ferramenta pode também ser usada para criar aplicações Java para desktops. Dentre os componentes disponíveis na ferramenta para a criação de interface estão: *Button*; *CheckBox*; Paleta de cores; *Data Types*; *Field*; *Label*; *List*; e *Drawing* (componente de escrita manual).

MobiForms pode construir aplicações para Windows CE, Pocket PC, EPOC/Symbian e Palm, sendo possível rodar as aplicações geradas com qualquer máquina virtual J2ME, como NSIcom CreEme 3.25, Jeode Runtime 1.9, PersonalJava ou outras. Por ser uma ferramenta comercial, possui licença para uso que, em fevereiro de 2006, estava custando U\$178,00 (Fonte: <http://www.mobiforms.com>).

Figura 3.6: Desenvolvimento de uma aplicação móvel usando MobiForms

3.7. Simplicity

O Simplicity é uma ferramenta comercial RAD para desenvolvimento Java da Data Representation²⁷. O pacote Simplicity tem ferramentas para desenvolvimento Java para Web, desktop e dispositivos móveis, dentre esta última destacam-se:

²⁶ MobiForms. <http://www.mobiforms.com>

²⁷ Data Representation. <http://www.datareprs.com>

- Simplicity for Palm OS é uma ferramenta para desenvolvimento de aplicações em J2ME usando a configuração CDC/MIDP que funcionam em dispositivos Palm OS. O Simplicity para Palm OS inclui emuladores para dispositivos como Tungsten, Treo e muito outros. Assim, pode-se construir e testar os MIDlets²⁸ usando a técnica de arrastar e soltar (*drag and drop*) os componentes para montar a interface visual. Além da interface de usuário os desenvolvedores podem adicionar imagens, temporizadores e comunicações de rede. Sua licença, em fevereiro de 2006, estava custando por volta de U\$395,00 (Fonte: <http://www.datareps.com>).
- Simplicity for Mobile Devices é uma ferramenta para construir aplicações de J2ME para todos os dispositivos que aceitam a configuração CDC/MIDP, tais como telefones celulares, PDAs e pagers. As aplicações Java podem ser escritas para Pocket PC, Palm OS, Nokia, Symbian e todos outros dispositivos que permitem MIDP. Os desenvolvedores podem, visualmente, construir programas robustos em poucas horas arrastando e soltando componentes. Uma emulador MIDP flexível ajuda os desenvolvedores a criar e testar seus MIDlets enquanto adaptam a aplicação para o dispositivo desejado. Sua licença é mais acessível que o Simplicity for Palm OS, custando cerca de U\$195,00, em fevereiro de 2006 (Fonte: <http://www.datareps.com>).

3.8. Considerações finais

Este capítulo apresentou alguns trabalhos relacionados à ferramenta MCards, sendo esses: HyperCard, XIML, LiquidUI, TERESA, MobiForms e Simplicity.

Além desses trabalhos selecionados, muitos outros se assemelham à presente pesquisa, seja devido à abordagem de construção de aplicações por intermédio de editores visuais, pela utilização de linguagem de marcação baseadas em XML ou por desenvolver aplicações para dispositivos móveis. Assim, para a seleção dos mesmos foram levadas em consideração três características básicas incluídas na ferramenta MCards:

- Estrutura de persistência e organização da interface baseadas no conceito de *cards*;
- Utilização de linguagem de marcação baseadas em XML para definir a interface para diversos dispositivos;

²⁸ MIDlets são aplicativos em J2ME para PDA desenvolvido com a configuração DLDC/MIDP.

- Desenvolvimento rápido (RAD) de aplicações Java para dispositivos móveis, especificamente PDAs.

A Tabela 3.1 ilustra uma comparação entre as ferramentas selecionadas levando em consideração parâmetros como:

- Persistência de dados: se baseada em *cards*, definida pelo usuário ou não existente;
- Linguagem utilizada: possui linguagem intermediária para a geração da aplicação, por exemplo, linguagens baseadas em XML;
- Editor visual: possui Editor visual para auxiliar o usuário durante o processo de desenvolvimento da aplicação;
- Tecnologia: tecnologia utilizada na criação da aplicação final, como por exemplo Java ou HTML;
- Plataforma ou dispositivo: plataforma ou dispositivo para o qual a aplicação gerada se destina;
- Licença: se necessita de pagamento de licença para a utilização da ferramenta.

Tabela 3.1: Tabela comparativa entre ferramentas selecionadas

Ferramenta	Persistência	Linguagem Utilizada	Editor Visual	Tecnologia	Plataforma / Dispositivo	Licença	Cronologia
HyperCard	cards	HyperTalk	Sim	-	Desktop - MacOS	Sim	Foi lançado em 1987 e se manteve até 2004
XIML	Definida pelo usuário	XIML	Não	-	Várias	Não	Final da década de 1990
LiquidUI	Não	UIML	Sim	J2SE 1.3, HTML, WML e VoiceXML	Desktop e celulares	Sim	Final da década de 1990
TERESA	Não	XML	Sim	XHTML, WML e VoiceXML	Desktop e celulares	Não	Recente: surgiu por volta de 2002
MobiForms	Definida pelo usuário	-	Sim	J2ME	Pocket PC, Symbian OS e Palm OS	Sim	Recente: lançado por volta de 2002
Simplicity	Definida pelo usuário	-	Sim	J2ME	Desktop, Web, Symbian OS, Pocket PC e Palm OS	Sim	Recente: lançado por volta de 2003

Como se pode notar, a ferramenta faz uso de conceitos consagrados como *cards* através da ferramenta Hypercard, além de conceitos relativamente recentes como definição de interface de usuário com o uso de definição abstrata (XML) e desenvolvimento de aplicações para PDAs.

4. MCARDS

4.1. Considerações Iniciais

Com a expansão significativa do número de dispositivos móveis, tanto na área acadêmica como no mercado corporativo, surge, cada vez mais, a necessidade de pesquisas sobre o uso de tais equipamentos. Devido ao tema ser relativamente novo, existem poucas ferramentas que auxiliam nesse desenvolvimento.

Nesse contexto, foi desenvolvido um trabalho durante o mestrado de Danilo Rogério Boccardo no Instituto de Ciências Matemáticas e de Computação / USP, com a finalidade de investigar as ferramentas existentes que auxiliam no desenvolvimento para assistentes pessoais digitais, especificamente as que apóiam a construção de interfaces e, como resultado, foi construída a ferramenta MCards, um construtor de interfaces para PDAs com correspondente geração de código. Neste capítulo são descritas a ferramenta MCards e suas principais funcionalidades.

4.2. Conceito de *Cards*

A ferramenta MCards busca auxiliar os usuários no desenvolvimento de aplicações baseadas em *cards*, ou cartões virtuais, para dispositivos móveis. O uso de *cards* permite separar contextos dentro da aplicação visando auxiliar no processo de construção ao separar o desenvolvimento da aplicação em partes. Trata-se do conceito de dividir para conquistar, amplamente utilizado na programação convencional e na ferramenta MCards tal abordagem ocorre de forma visual. Para concretizar essa idéia, usa-se uma série de mecanismos que vão desde um editor visual para manipulação dos componentes de interface, passando pela transformação em uma linguagem abstrata, posteriormente para o código da aplicação na linguagem desejada, até o resultado final que corresponde à geração da aplicação de acordo com a linguagem escolhida (BOCCARDO, 2005).

MCards se diferencia, em termos de criação de interface, em comparação com a maioria dos sistemas existentes, devido ao conceito de "cartões virtuais", ou *cards* que podem incluir campos para armazenamento de dados e componentes visuais. Entre os elementos visuais possíveis num *card* dessa ferramenta estão botões, textos, campos de texto, ligações (*links*), *strokes* (componente para escrita manual), e outros elementos comuns de uma interface gráfica. Além disso, é possível o estabelecimento de vínculos por intermédio de ligações virtuais entre os *cards*. Para cada *card* criado, há o controle de persistência, de acordo com os componentes escolhidos, sendo tal funcionalidade transparente ao usuário. Por exemplo, para cada *card* a ferramenta gera uma barra navegacional para, que em tempo de execução, controlar as várias instâncias dentro de cada *card*. Além disso, é possível estabelecer ligações entre os *cards* através da utilização de ligações e outros componentes com eventos. A Figura 4.1 ilustra um exemplo de ligação entre *cards* e várias instâncias para cada *card*.

Figura 4.1: Exemplo de navegação de uma aplicação gerada pelo MCards (BOCCARDO, 2005, p. 38)

Além disso, a ferramenta MCards possui o módulo de geração de código responsável por gerar o artefato executável, ou seja, a aplicação para o dispositivo móvel. Na versão original da ferramenta, é possível apenas gerar aplicações para a linguagem SuperWaba. O código produzido espelha a manipulação ocorrida no editor visual, de modo transparente ao usuário, ou seja, este não precisa manusear o código. Além do módulo de geração de código e o editor visual, MCards possui também um módulo cujo objetivo é a comunicação entre a aplicação que está sendo produzida e uma determinada aplicação Web definida pelo usuário. Para tal fim, estabelece um formato de mensagem a ser trocada e os meios necessários para a recuperação desses dados no lado do servidor.

4.3. A Ferramenta MCards

A ferramenta MCards tem por objetivo a construção de aplicações baseadas em *cards* para PDAs com geração de código. A ferramenta disponibiliza ao desenvolvedor um ambiente visual com um conjunto de elementos de interface básicos (*Label*, *Edit*, *ListBox*, *ComboBox*, *TextArea*), um elemento de navegação (*Link*), e um elemento específico para PDAs (Entrada para *Strokes*). Para a manipulação desses elementos, MCards explora a técnica *drag and drop* para seleção, criação, e posicionamento dos elementos na interface, objetivando, com isso, tornar o processo intuitivo, bem como ajudar a prevenir erros de programação. A Figura 4.2 lista as principais funcionalidades da ferramenta MCards, ilustradas em cada caso de uso de primeiro nível.

Figura 4.2: Diagrama de Caso de Uso da ferramenta MCards (BOCCARDO, 2005, p. 36)

Assim, têm-se o processo de especificação da interface da aplicação a ser gerada, abrangendo: (a) a manipulação de *cards* e componentes (*widgets*) e de parâmetros relacionados como a linguagem em que o código será gerado, como o dispositivo para qual será gerada a aplicação e como o tamanho da tela; (b) a geração de código-fonte baseado na especificação definida durante a criação da aplicação; e (c) a integração com aplicações Web onde se define uma URL com a qual a aplicação criada poderá trocar informações.

4.4. Editor Visual

O MCards tem como foco auxiliar ao usuário na criação de aplicações para dispositivos móveis, especificamente PDAs, fazendo com que o mesmo não necessite conhecer a linguagem de programação nem ter conhecimento prévio das possibilidades do dispositivo alvo. Para tal finalidade, o MCards possui um módulo denominado Editor Visual que permite ao usuário a inserção de componentes de interface em uma tela virtual que representa o dispositivo pretendido, de modo que a posição, o tamanho e outras propriedades dos componentes possam ser manipulados diretamente através da técnica de manipulação de componentes *drag and drop*. Além disso, demais parâmetros como tamanho da tela virtual, dispositivo alvo, linguagem pretendida e URL de comunicação podem ser manipulados através desse módulo de edição visual.

O resultado do processo de criação da aplicação no editor visual corresponde a um artefato XML que reflete os componentes inseridos, seus atributos, *cards* associados e ligações entre eles, quando existentes. A vantagem dessa abordagem é a separação entre a especificação da aplicação e o código a ser gerado, facilitando assim a inserção de novas linguagens para a produção de códigos e aplicações, ou seja, permite a utilização de n transformadores que interpretam e transformam esse documento XML para n linguagens diferentes, desvincula-se assim a construção da aplicação do dispositivo alvo e da linguagem de programação.

A Figura 4.3 apresenta o editor visual onde em (a) tem-se a barra de componentes visuais que contém componentes como *Label*, *Text*, *Edit*, *ComoBox*, *List* e *Button*, além de elementos como *Strokes*, para dispositivos baseados em caneta, e *Link* para o estabelecimento de ligações entre contextos distintos. O elemento de ligação, *Link* é fundamental para a ferramenta MCards por que permite a criação de vínculos entre os *cards*. Ligações também podem ser definidas por outros elementos, tais como *Button*, *List* e *ComboBox* (BOCCARDO, 2005).

Quando o usuário seleciona um componente da barra de componentes, tal elemento é inserido na área indicada em (b), na Figura 4.3, onde é possível arrastar e dimensionar o componentes e definir atributos para o mesmo. Essa tela virtual permite visualizar a interface da aplicação a ser gerada. Em (c) é mostrado o menu do editor visual que possui como opções para configurações de parâmetros necessários para a geração do código da aplicação como dispositivo alvo, linguagem para a qual o código será gerado tamanho da área de edição (tela

virtual) e URL para a comunicação. Em (d) está a barra de ferramentas que contem o atributo *store* que informa aos transformadores de código a necessidade ou não de se controlar a persistência.

Figura 4.3: Editor Visual da ferramenta MCards (BOCCARDO, 2005, p. 40)

Além disso é possível visualizar o arquivo XML (*forML*) gerado até então para a aplicação e o arquivo XML que representa a mensagem a ser transmitida pela aplicação gerada a uma aplicação Web (*messageML*). Logo abaixo, têm-se os botões para a visualização dos códigos gerados, inicialmente a ferramenta MCards gera aplicações apenas para SuperWaba. Por fim, tem-se em (e) o botão que inicia o processo no qual a instância de *forML* é enviada aos transformadores para a geração do código. A Figura 4.4 mostra a tela de uma aplicação construída no MCards, e a Figura 4.5 mostra o documento *forML*, ou seja, a especificação XML correspondente.

Figura 4.4: Exemplo de aplicação construída no MCards

```
<?xml version="1.0" encoding="UTF-8" ?>
<forML>
  <menu about="" help="" />
  <parameters height="320" width="240" platform="PALM" code="SUPERWABA" title="Cadastro" url="" store="true" />
  <components>
    <form id="0" backgroundColor="202,223,216" navigatorbar="true">
      <myedit name="Edit_Nome" backgroundColor="255,255,255" foregroundcolor="0,0,0" lines="1" position="76,32" size="93,20" />
      <mylabel name="MyLabel_Nome" foregroundcolor="0,0,0" position="31,27" text="Nome:" />
      <myedit name="Edit_Telefone" backgroundColor="255,255,255" foregroundcolor="0,0,0" lines="1" position="80,124" size="85,20" />
      <mylabel name="MyLabel_Telefone" foregroundcolor="0,0,0" position="16,116" text="Telefone:" />
      <mybutton name="mybutton3423158" backgroundColor="236,233,216" foregroundcolor="0,0,0" text="Enviar" position="61,211" event="null" option="null" />
      <mycombobox backgroundColor="210,210,210" foregroundcolor="0,0,0" position="80,75" items="Masculino~NULL#Feminino~NULL#" name="Combo_Sexo" events="GOTO" />
      <mylabel name="MyLabel_Sexo" foregroundcolor="0,0,0" position="31,70" text="Sexo:" />
    </form>
  </components>
</forML>
```

Figura 4.5: Exemplo de arquivo *forML* gerado pela ferramenta MCards

4.5. Geração de Código

O módulo de geração de código encarrega-se de transformar o artefato XML, denominado *forML* gerado pela manipulação do usuário no editor visual, em código para uma determinada linguagem. No código gerado para aplicação tem-se informações necessárias para a formação da interface com o usuário, controle de persistência no dispositivo, controle de eventos e comunicação HTTP para envio de dados encapsulados num padrão XML definido por MCards, porém personalizado pelo usuário.

Inicialmente o MCards possui apenas o tradutor para a linguagem SuperWaba, mas o módulo de geração de código foi construído de maneira a facilitar a inserção de novos tradutores.

Normalmente, a manipulação de dados XML necessita de SAX²⁹ (*Simple API for XML*) ou DOM³⁰ (*Document Object Model*), sendo essas APIs amplamente utilizadas no tratamento de dados XML. Embora o uso consagrado dessas APIs, MCards faz uso de JAXB³¹ (*Java Architecture for XML Binding*) para tratar o *forML*, devido ao seu relacionamento intrínseco com a plataforma Java que consiste na tecnologia empregada na construção da ferramenta MCards.

JAXB permite que a partir de um esquema, no caso XMLSchema, gere-se classes Java correspondentes em que cada elemento torna-se uma classe e cada atributo, uma variável de instância. Mediante o processo denominado *Unmarshal* permite-se a associação de dados XML, comumente por intermédio de um arquivo XML, com classes geradas. A vantagem de JAXB, quando comparado a outras APIs, é que o acesso aos dados XML pode ser realizado instanciando classes e métodos representativos.

Figura 4.6: Funcionamento de JAXB no MCards (BOCCARDO, 2005, p. 43)

A Figura 4.6 mostra que os tradutores não trabalham diretamente na especificação abstrata da aplicação a ser gerada, no caso o documento XML, mas sim em objetos Java provenientes desse documento, gerados pelo processo de *Unmarshal*, já descrito anteriormente. Além do processo de *Unmarshal*, a Figura 4.6 ilustra também o processo de *Marshal* em que um objeto Java pode ser transformado num documento XML.

4.6. Controle de Mensagens

O resultado da manipulação do editor visual não consiste apenas em código executável pra uma determinada linguagem. Além de gerar aplicações do tipo formulário com tratamento

²⁹ SAX: <http://www.saxproject.org>

³⁰ DOM: <http://www.w3.org/DOM/>

³¹ JAXB: <http://Java.sun.com/xml/jaxb/>

de persistência, MCards provê, também, um tratamento dos dados gerados em tempo de execução pelo aplicativo, estabelecendo um formato de mensagem e meios de comunicação via HTTP para o envio dos dados pra uma determinada aplicação Web definida pelo usuário. Ou seja, os potenciais dados a serem produzidos pela aplicação em execução no dispositivo são considerados por MCards no momento da edição da aplicação. Ao fazer isso, o usuário que desenvolve a aplicação, ao utilizar a ferramenta MCards, tem ciência das informações que podem ser enviadas pela aplicação que está sendo produzida.

Bem como o formato utilizado para a especificação da aplicação a ser gerada é o formato XML, a representação da mensagem a ser enviada também faz uso de XML devido às vantagens já mencionadas. Este documento de especificação da mensagem a ser enviada denomina-se *messageML*. O uso desse padrão permite ao usuário desenvolvedor integrar aplicações geradas pelo MCards com sistemas legados residentes em ambientes Web. Tal abordagem faz da ferramenta MCards um gerador de aplicações que podem funcionar como mecanismo de entrada para sistemas pré-existent. A Figura 4.7 mostra um exemplo de arquivo *messageML*:

```
<?xml version="1.0" encoding="UTF-8" ?>
<MessageML isStoreinPda="true" aplicacion="Cadastro">
<Data>
<Container id="0">
<Record id="0">
<edit name="Edit_Nome">Fulano de Tal</edit>
<edit name="Edit_Telefone">16 5566-9966</edit>
<combobox name="Combo_Sexo">Masculino</combobox>
</Record>
<Record id="1">
<edit name="Edit_Nome">Ciclana</edit>
<edit name="Edit_Telefone">16 6633-3663</edit>
<combobox name="Combo_Sexo">Feminino</combobox>
</Record>
</Container>
</Data>
</MessageML>
```

Figura 4.7: Exemplo de arquivo *messageML*

4.7. Considerações Finais

A Tabela 4.1 inclui a ferramenta MCads na comparação apresentada no Capítulo 3 (Tabela 3.1).

Tabela 4.1: Tabela comparativa entre ferramentas e ferramenta MCards

Ferramenta	Persistência	Linguagem Utilizada	Editor Visual	Tecnologia	Plataforma / Dispositivo	Licença	Cronologia
HyperCard	cards	HyperTalk	Sim	-	Desktop - MacOS	Sim	Foi lançado em 1987 e se manteve até 2004

XIML	Definida pelo usuário	XIML	Não	-	Várias	Não	Final da década de 1990
LiquidUI	Não	UIML	Sim	J2SE 1.3, HTML, WML e VoiceXML	Desktop e celulares	Sim	Final da década de 1990
TERESA	Não	XML	Sim	XHTML, WML e VoiceXML	Desktop e celulares	Não	Recente: surgiu por volta de 2002
MobiForms	Definida pelo usuário	-	Sim	J2ME	Pocket PC, Symbian OS e Palm OS	Sim	Recente: lançado por volta de 2002
Simplicity	Definida pelo usuário	-	Sim	J2ME	Desktop, Web, Symbian OS, Pocket PC e Palm OS	Sim	Recente: lançado por volta de 2003
MCards	cards	XML	Sim	SuperWaba	Pocket PC e Palm OS	Não	Início: 2003

Pode-se notar, ao analisar a Tabela 4.1, que a ferramenta MCards possui persistência de dados baseada em *cards*, a linguagem intermediária usada para a geração da aplicação é XML, além de possuir um editor visual para a construção das aplicações que serão geradas com o uso da tecnologia SuperWaba, para os dispositivos Pocket PC e Palm OS.

5. MCARDS 2.0: ENGENHARIA REVERSA E EVOLUÇÃO DA FERRAMENTA

MCARDS

5.1. Considerações Iniciais

O objetivo do presente projeto de mestrado foi investigar o desenvolvimento de aplicações para dispositivos móveis, especificamente para PDAs, com independência de plataforma, analisando e expandindo a ferramenta MCards, com o propósito de permitir a geração de aplicações para dispositivos móveis com a utilização da tecnologia J2ME. Assim, no início do desenvolvimento das atividades referentes ao presente projeto de mestrado, o primeiro passo foi o levantamento de novos requisitos, seguido da análise da demanda por expansões na ferramenta para assim definir as expansões e as modificações necessárias de serem executadas na ferramenta. O objetivo principal era definir quais os pontos favoráveis para a integração de um módulo de geração de aplicações J2ME, de modo a demandar menos esforços, para o usuário, durante o uso da ferramenta e a respectiva geração de aplicações para PDAs.

Durante o estudo das funções e da implementação já existentes na ferramenta, foram constatados os primeiros problemas: erros na execução da ferramenta e falhas durante o processo de geração da aplicação para PDAs. Além disso, foi constatada a ausência de documentação tanto para o usuário final quanto para o desenvolvedor que fosse realizar futuras manutenções na ferramenta MCards. Essa ausência de documentação dificultou o processo de entendimento da ferramenta tanto no seu uso quanto no desenvolvimento das atividades de evolução da mesma. É importante observar que esses problemas se justificam, uma vez que o trabalho original tinha por objetivo investigar mecanismos para a construção

de aplicações para dispositivos do tipo PDA que fizesse uso de manipulação direta de elementos de interação, assim o protótipo desenvolvido se confirma como uma prova de conceito da proposta correspondente.

Diante desses fatores, e tendo também como objetivo facilitar o uso da ferramenta, foi constatado que antes de investigar um mecanismo de geração de aplicações J2ME para dispositivos móveis, havia a necessidade de resolver os problemas já existentes. Nesse capítulo são descritas as atividades desenvolvidas no contexto da Engenharia Reversa, das manutenções executadas na ferramenta e da expansão da ferramenta MCards com a implementação do módulo de geração de aplicações J2ME.

5.2. Engenharia Reversa da Ferramenta MCards

A Engenharia Reversa é uma linha de pesquisa da engenharia de software que inverte o ciclo de vida tradicional do software caracterizado pelo processo tradicional de engenharia de software conhecido como Engenharia Progressiva (CHIKOFSKY e CROSS, 1990), ou seja, a Engenharia Reversa parte do nível mais baixo de abstração visando o nível mais alto para o entendimento do software.

Conceitos de Engenharia Reversa foram aplicados na ferramenta MCards com a finalidade de recuperar o nível mais alto de abstração da mesma, ou seja, suas especificações e requisitos. Assim, nesta seção serão tratados os conceitos básicos de Engenharia Reversa, o modelo de Engenharia Reversa FUSION-RE/I e sua aplicação na ferramenta MCards.

5.2.1. Conceitos Básicos de Engenharia Reversa

Segundo Pressman (PRESSMAN, 2002), a Engenharia Reversa é o processo de analisar um software num esforço de criar uma representação do mesmo, em um nível de abstração mais alto do que o código-fonte. Assim sendo, pode-se dizer que o objetivo da Engenharia Reversa é a produção de informações que possam aumentar o conhecimento geral de sistemas de software. Essas informações podem ser utilizadas em atividades como manutenção, reuso, teste e controle de qualidade de software.

A Figura 5.1 ilustra a relação entre a Engenharia Progressiva e a Engenharia Reversa de acordo com as fases do ciclo de vida do software, sendo que essas fases podem ser agrupadas em três atividades fundamentais:

- **Sistema** (engenharia de sistemas): envolve o contexto em que o sistema está operando, ou seja, o porquê do sistema ser desenvolvido;
- **Requisitos** (análise): são estabelecidos os serviços a serem fornecidos pelo sistema e as restrições sob as quais ele deve operar, ou seja, o que o sistema deve fazer e sob quais circunstâncias;
- **Desenvolvimento** (projeto, codificação e testes): cria-se um planejamento da solução, ou seja, como o sistema cumprirá o que foi estabelecido na atividade de requisitos e a implementação dessa solução, incluindo a codificação, os testes, a depuração e a entrega do sistema. A fase de manutenção é vista como reiteração das atividades prévias.

Figura 5.1: Engenharia Progressiva / Engenharia Reversa

Um dos objetivos principais da Engenharia Reversa é a recuperação de informações para serem utilizadas na fase de manutenção do software. Pode-se observar que a facilidade de manutenção (manutenibilidade), caracterizada principalmente pelo entendimento do sistema, está fortemente relacionada à disponibilidade de informações sobre o sistema. Essas informações podem ser produzidas pela Engenharia Reversa facilitando, primeiramente, o entendimento e, posteriormente, a modificação e revalidação do sistema, aumentando assim a manutenibilidade do mesmo (CHIKOFSKY e CROSS, 1990).

Segundo Harandi e Ning (HARANDI e NING, 1990), um sistema pode ser visualizado a partir de diferentes níveis de entendimento. Baseado nos níveis de abstração, as visões são classificadas em quatro categorias (Figura 5.2):

- **Visão em Nível Implementacional:** abstrai características da linguagem de programação e, especificamente, da implementação. Exemplos de visões em nível implementacional são informações a respeito da sintaxe e da semântica da linguagem, e informações de implementação;
- **Visão em Nível Estrutural:** abstrai detalhes da implementação para revelar sua estrutura a partir de diferentes perspectivas. O resultado é uma representação explícita das dependências entre os componentes do sistema;
- **Visão em Nível Funcional:** abstrai a função de um componente do sistema de modo a identificar o que o componente faz. Relaciona partes do programa à suas funções a fim de revelar as relações lógicas entre elas;
- **Visão em Nível de Domínio:** abstrai o contexto em que o sistema está operando, ou seja, relata o porquê do sistema ter sido desenvolvido.

Figura 5.2: Níveis de entendimento do software de acordo com o ciclo de vida

É relevante ressaltar que uma forma de representação extraída do código pode ser diferente de uma representação similar que foi desenvolvida no processo de engenharia

progressiva. A forma extraída irá refletir as particularidades do código muito mais do que a representação original, que reflete compreensão do problema.

- Para obter as diversas visões do software, usualmente é necessário acrescentar às informações contidas no código outras informações provenientes de conhecimento e de experiência humana. De acordo com o nível de entendimento obtido do sistema e o escopo das informações fornecidas, há uma categorização das técnicas de Engenharia Reversa: Redocumentação e Recuperação de Projeto (CHIKOFSKY e CROSS, 1990). Na Figura 5.3, são apresentadas as categorias de Engenharia Reversa, relacionadas às fases do ciclo de vida.

Figura 5.3: Categorias da Engenharia Reversa relacionadas ao ciclo de vida

- **Redocumentação:** é a criação ou revisão de representações semanticamente equivalentes em um mesmo nível de abstração. O processo de redocumentação cria as representações a partir de informações obtidas apenas da análise do código-fonte. Por exemplo: a partir do código-fonte, pode-se analisar e extrair a arquitetura do programa, a estrutura de controle, a estrutura de dados, etc. As formas obtidas das representações são consideradas visões alternativas, tendo como objetivo melhorar a compreensão do sistema de maneira global;
- **Recuperação de Projeto ou Entendimento do Programa:** Tem como objetivo entender o sistema ao invés de simplesmente fornecer visões alternativas para auxiliar o usuário a entendê-lo, distinguindo-se, porém, da redocumentação. Esse entendimento vai além do escopo do conhecimento em nível implementacional e

estrutural. A ênfase é obter conhecimento em nível funcional e até mesmo em nível de domínio (ambiente de operação do sistema).

Para auxiliar a atividade de manutenção de software, Costa (COSTA, 1997) desenvolveu o método de Engenharia Reversa FUSION-RE/I que, partindo do código-fonte, de aspectos operacionais do sistema e de dados da interface, recupera visões funcionais e estruturais do sistema. As visões funcionais são recuperadas partindo-se de informações apresentadas na interface e de aspectos operacionais; as visões estruturais são recuperadas utilizando as visões funcionais produzidas e o código-fonte do sistema. Na seção a seguir são detalhadas as etapas do método de Engenharia Reversa FUSION-RE/I, que foi utilizado neste trabalho de mestrado.

5.2.2. Método FUSION-RE/I

O método FUSION-RE/I (FUSION – Reverse Engineering / Interface) (COSTA, 1997) é um método para a realização de Engenharia Reversa que, partindo do código-fonte, de aspectos operacionais do sistema e de dados da interface, recupera as visões funcionais e estruturais do sistema.

As visões funcionais são recuperadas partindo-se de informações apresentadas na interface e de aspectos operacionais; as visões estruturais são recuperadas utilizando as visões funcionais produzidas e o código-fonte do sistema. O método FUSION-RE/I é constituído de duas etapas:

- **1ª Etapa – Recuperar Visões Funcionais:** parte-se de considerações “lógicas”, obtidas por meio de aspectos operacionais e visões de dados disponíveis via interface, para a recuperação de visões funcionais do sistema, ou seja, essa primeira etapa tem como objetivo a abstração da funcionalidade do sistema, representada por meio dos modelos de ciclo de vida, de operações e de objetos do sistema, segundo o método de Engenharia Reversa FUSION de Coleman et al (COLEMAN et al., 1994);
- **2ª Etapa – Recuperar Visões Estruturais:** parte-se de considerações “físicas”, obtidas por meio do código-fonte, e das visões funcionais, para a recuperação de visões estruturais do sistema, sendo que o objetivo dessa fase é a identificação dos procedimentos que implementam as operações da interface do sistema,

representados através dos quadros de chamadas de procedimentos e de operações-procedimentos de implementação.

Na Figura 5.4 é esquematizada uma síntese do método de Engenharia Reversa FUSION-RE/I:

Figura 5.4: Síntese do Método FUSION-RE/I (COSTA, 1997, p.83)

Conforme ilustrado na Figura 5.4, a Etapa de Recuperação de Visões Funcionais é subdividida em dois passos: o primeiro envolve a reunião de todas as informações existentes sobre o sistema em estudo e o segundo envolve a recuperação de informações da fase de análise do sistema. No primeiro passo deve-se:

- Reunir toda a documentação envolvendo a informação textual existente sobre o sistema (manuais, livros, artigos, listagem do código-fonte, etc.) e a informação técnica relacionada aos conceitos pertinentes (domínio do sistema, aplicação, linguagem de implementação do sistema, da interface, etc.);
- Identificar os usuários do sistema, pois informações importantes podem não estar documentadas e um modo de recuperá-las é a partir do conhecimento dos desenvolvedores e de usuários. Esses conhecedores do sistema também são requeridos para auxiliar na validação dos modelos obtidos no processo de aplicação do método FUSION-RE/I;
- Analisar os dados a fim de identificar informações relacionadas aos requisitos do sistema, ao projeto arquitetural, ao projeto de dados e procedimental, ao ambiente onde o sistema é executado, à linguagem de implementação do sistema, à organização dos arquivos no disco, etc.

O segundo passo da Etapa de Recuperação de Visões Funcionais do método FUSION-RE/I, envolve a recuperação das informações da fase de análise do sistema. Para representar

essas informações são utilizados os modelos da fase de análise do método FUSION (COLEMAN et al., 1994) de desenvolvimento de software orientado a objeto. A fase de análise do método FUSION gera dois modelos: (1) o modelo de objetos, que descreve a estrutura do sistema e (2) o modelo de interface, que descreve o comportamento do sistema. O modelo de interface é composto de dois modelos que capturam diferentes aspectos do comportamento: o modelo de ciclo de vida, o qual caracteriza seqüências permitidas de operações e eventos do sistema, e o modelo de operações, o qual caracteriza o efeito de cada operação do sistema em termos de mudanças de estado e eventos gerados. Esse segundo passo consiste em:

- Elaborar o Modelo de Ciclo de Vida: para tanto, se faz necessário possuir um bom conhecimento do domínio e da aplicação do sistema submetido à Engenharia Reversa, incluindo estudo da documentação existente e realização de entrevistas com usuários e, se possível, com os desenvolvedores do sistema. A partir disso, e principalmente do uso do sistema, é possível definir as seqüências de operações permitidas e os eventos de entrada e de saída que o sistema aceita;
- Elaborar o Modelo de Operações: para a elaboração do modelo de operações do sistema, parte-se do modelo de ciclo de vida obtido anteriormente. Esse modelo de ciclo de vida apresenta uma visão geral da funcionalidade das operações do sistema, as quais, para elaboração do modelo de operações, devem ser melhor estudadas, por meio do uso intensivo do sistema, de modo que possam ser especificadas detalhadamente;
- Elaborar o Modelo de Objetos: definindo, primeiramente, assuntos com os quais a funcionalidade do sistema se relaciona. O objetivo da definição de assuntos é identificar os diferentes níveis de abstração da funcionalidade do sistema e os possíveis agrupamentos de operações de um mesmo assunto.

Na Etapa de Recuperação de Visões Estruturais do método FUSION-RE/I, os passos a serem desenvolvidos são subdivididos em:

- Elaborar o Quadro de Procedimentos;
- Elaborar o Quadro das Operações/Procedimentos de Implementação.

O primeiro passo da Etapa de Recuperação das Visões Estruturais envolve o entendimento da implementação do sistema, por meio da identificação dos procedimentos de

implementação, da funcionalidade de cada procedimento identificado e da sequência de chamadas desses procedimentos. A apresentação dos procedimentos de cada arquivo do sistema é realizada por meio de um conjunto de quadros de chamadas, que vem acompanhado por um índice geral dos procedimentos do sistema. Esse passo, então, consiste em:

- Elaborar o Quadro de Chamadas de Procedimentos: para cada arquivo do sistema, deve ser elaborado um quadro de chamadas dos procedimentos de implementação. Nesse quadro de chamadas devem ser apresentados todos os procedimentos contidos no arquivo, fazendo-se referência aos procedimentos utilizados (CHAMA) e utilizadores (CHAMADO POR);
- Elaborar o Quadro Índice de Procedimentos: para isso, utilizam-se os quadros de chamadas de procedimentos de todos os arquivos.

O segundo passo da Etapa de Recuperação de Visões Estruturais do método FUSION-RE/I consiste em identificar, entre todos os procedimentos considerados no quadro de chamadas, aqueles que implementam as operações da interface do sistema (identificadas na primeira etapa), e alocá-los, de acordo com sua funcionalidade, à interface ou a um dos assuntos do sistema (identificados na primeira etapa).

5.2.3. Aplicação do Método FUSION-RE/I na Ferramenta MCards

A aplicação do processo de Engenharia Reversa na ferramenta MCards baseou-se no método FUSION-RE/I para recuperação das visões funcionais e visões estruturais da ferramenta. Como dito anteriormente, o método FUSION-RE/I é dividido em duas etapas: a Recuperação de Visões Funcionais e a Recuperação das Visões Estruturais. Cada etapa é subdividida em passos cujas aplicações na ferramenta MCards serão detalhados a seguir.

A realização do primeiro passo da Etapa de Recuperação de Visões Funcionais do método na ferramenta MCards envolveu a realização das seguintes atividades que encontram-se detalhadas no Apêndice A, “Engenharia Reversa da Ferramenta MCards 1.0”:

- Leitura da dissertação do trabalho de mestrado que deu origem à ferramenta (BOCCARDO, 2005);
- Uso intensivo da ferramenta para familiarização com a interface;
- Elaboração do roteiro “Como usar o software”, presente no documento de especificação da ferramenta MCards;

- Estudo da linguagem Java para entendimento do código-fonte;
- Identificação dos requisitos funcionais da ferramenta, presente no documento de especificação da ferramenta MCards;
- Identificação dos atores e elaboração do diagrama de casos de uso, presente no documento de especificação da ferramenta MCards.

A realização do segundo passo da Etapa Recuperação de Visões Funcionais do método FUSION-RE/I na ferramenta MCards compreendeu a realização das seguintes atividades (Apêndice A):

- Identificação dos objetos da ferramenta e elaboração de uma tabela de termos e definições a respeito desses objetos;
- Identificação da arquitetura da ferramenta, seus módulos e as interdependências entre eles;
- Elaboração dos casos de uso da ferramenta, detalhando o fluxo principal de eventos e os fluxos alternativos;
- Elaboração do modelo de ciclo de vida da ferramenta.

Durante a execução do primeiro passo da Etapa de Recuperação de Visões Estruturais do método FUSION-RE/I, para o entendimento da implementação da ferramenta MCards, ao invés de elaborar o quadro de chamadas, atividade que demandaria grande esforço e tempo, procurou-se conhecer a implementação da ferramenta, entendendo a estrutura de pacotes e identificando-se os métodos e classes de implementação em cada arquivo de código, elaborando a relação de pacotes e arquivos e o índice geral de procedimentos da ferramenta MCards (Apêndice A), construindo assim o modelo mental da estrutura do sistema.

O segundo passo da Etapa de Recuperação de Visões Estruturais consiste em identificar os procedimentos que implementam as operações da interface do sistema, e alocá-los, de acordo com sua funcionalidade, à interface ou a um dos temas do sistema. Todavia, este passo, como não foi elaborado no passo anterior o quadro de chamadas, não foi executado para a Engenharia Reversa da ferramenta MCards por se tratar de um modelo para realizar engenharia reversa em softwares orientados a procedimentos para a transferência para o paradigma orientado a objetos sendo, a ferramenta MCards, um software desenvolvido sob o paradigma de orientação a objetos.

Vale ressaltar que, além do estudo realizado, foram necessários alguns encontros com o aluno responsável pela implementação da ferramenta MCards para o esclarecimento de dúvidas. Foi realizada também a documentação do procedimento de instalação da ferramenta MCards, do manual de utilização e dos procedimentos de instalação da máquina virtual SuperWaba e J2ME, além do procedimento de instalação da aplicação gerada pela ferramenta MCards que se encontra disponível no “Manual da Ferramenta MCards 2.0” (Apêndice B).

5.3. Manutenção da Ferramenta MCards

O termo manutenção significa muito mais do que simplesmente consertar erros, significa manter, consertar e até atualizar, ou seja, após a sua construção um software irá inevitavelmente sofrer modificações (PRESSMAN, 2002). Uma modificação acontece quando erros são encontrados, quando o software precisa ser adaptado, ou quando o cliente deseja melhoramentos funcionais ou de desempenho. Quatro tipos de manutenção são identificados por Swanson (SWANSON, 1976):

- **Manutenção Corretiva:** dá-se em partes do sistema que não estejam funcionando de forma satisfatória, ou seja, a manutenção corretiva modifica o software para corrigir defeitos.
- **Manutenção Adaptativa:** é o ajuste do software para acomodar mudanças no seu ambiente externo. Por exemplo: adaptação a um novo sistema operacional, mudanças de versão de tecnologia, entre outros.
- **Manutenção Perfectiva:** aprimora o software além dos requisitos funcionais originais, ou seja, adiciona funções que trarão benefícios para seu uso.
- **Manutenção Preventiva:** também chamada de reengenharia de software, a manutenção preventiva faz modificações no software de modo que ele possa ser mais facilmente corrigido, adaptado e melhorado.

Nesse contexto, durante o processo de manutenção da ferramenta MCards, foram necessárias correções no código para um melhor funcionamento da ferramenta, bem como alterações na para facilitar o uso da mesma, entre outras atividades. As principais atividades realizadas no contexto de cada tipo de manutenção foram:

Manutenção Corretiva:

- Script de Geração de Aplicação: o script de geração de executável (.bat) para Superwaba encontrava-se com erro e não compilava o código, pois o *class_path* estava incorreto e com isso não gerava o arquivo .pdb, ou seja, os executáveis gerados eram inconsistentes;
- Contador de Registros: correção do contador de registros durante a execução da aplicação gerada;
- Suporte a *caracteres ISO-8859-1*: não era possível a criação de rótulos de texto com acentos e pontuação nos componentes como *Link*, *Label*, entre outros;
- Componente *Stroke*: correções no componente *Stroke* pois o mesmo estava com cor de fundo relativamente escura e o traçado (linha da escrita) estava desregulado, o que impossibilitava o seu uso;
- Diversas correções no código como, por exemplo, a comparação de *String* com sintaxe errônea, o que causava erros durante a execução da ferramenta MCards;
- Padronização de idioma: variáveis e comentários no código foram traduzidos para o Inglês e a interface com o usuário (menus, mensagens de erro, entre outros) foram padronizadas para o idioma Inglês;
- XMLSchema (*forML.xsd*): correção de inconsistências no schema, pois o mesmo não estava refletindo o código gerado;
- Dimensão dos componentes: inclusão da dimensão dos componentes no schema e no código gerado. A ferramenta MCards não armazenava a dimensão dos componentes definidos pelo usuário ao criar uma aplicação, alterando portando a interface no momento da geração do código e execução da aplicação.

Manutenção Adaptativa:

- Foi realizada a migração da versão do SuperWaba SDK do módulo de geração de código SuperWaba de 2.0 para 5.61 que se destaca pela rapidez e confiabilidade de suas aplicações, além de ter suporte também a Linux e Symbian OS. Esta mudança acarretou em alterações no módulo de geração de código SuperWaba pois o código gerado não era compatível com a nova versão do SuperWaba SDK.

Manutenção Perfectiva:

- Instalação: criação de um único pacote para instalação, seguindo como modelo o Eclipse³² que possui um *workspace* (área de trabalho da aplicação) para criação dos projetos. Este pacote já possui o Java SDK e o SuperWaba SDK, o que poupa o usuário de ter que instalá-los;
- *Workspace*: com a criação do *workspace* não é mais necessária a escolha de diretório onde será armazenado o projeto: o mesmo será criado no *workspace* com o nome da aplicação que está sendo gerada. Vale ressaltar que, na versão original da ferramenta MCards, era necessário criar um diretório com exatamente o mesmo nome da aplicação e, posteriormente, selecioná-lo, o que acarretava vários problemas para o usuário como, por exemplo, perda de dados;
- Salvar e Editar Projeto: a ferramenta MCards não salvava o projeto, portanto não era possível fazer a edição do mesmo, o que trazia vários transtornos ao usuário durante a criação de uma aplicação utilizando a ferramenta MCards;
- Script de Geração de Aplicação SuperWaba: foi modificado o script de geração de aplicação para que o mesmo possa também realizar a geração de instaláveis para Palm OS, Pocket PC, Linux e Symbian OS, além dos arquivos *.prc* e *.pdb*;
- Navegação: reorganização da navegação da ferramenta: (1) não é necessária a identificação do usuário (dado que esta informação não é utilizada na criação de uma aplicação); (2) ao iniciar a ferramenta, a tela de edição de aplicação é mostrada ao usuário; (3) remoção do menu de escolha do dispositivo, tendo em vista que ambas as linguagens (Superwaba e J2ME) geram aplicações para ambos os dispositivos; (4) Substituição da tela de “*Deploy*” por botão “*Generate Code*” na tela de criação da aplicação para simplificar a interação do usuário com a ferramenta (detalhes em “Manual da Ferramenta MCards 2.0” - Apêndice B);
- Geração do Executável ou Instalador para a Aplicação: com a adição do botão “*Compile*” na tela de edição de aplicação, não é mais necessário que o usuário execute um script de geração de executável (arquivo *.bat*), pois a própria ferramenta Mcards encarrega-se de gerar os executáveis ou instaladores para a aplicação que está sendo criada, ou seja, torna a tarefa de geração da aplicação mais transparente, transferindo essa complexidade do usuário para a ferramenta.

³² Eclipse Project: <http://www.eclipse.org/>

5.4. Nova Arquitetura da Ferramenta MCards

De acordo com a apresentação original da ferramenta, sumarizada no Capítulo 4, a ferramenta MCards possui uma arquitetura dividida em: módulo do editor visual e módulo de geração de códigos, sendo que este último pode conter tradutores para várias linguagens.

Com a aplicação do processo de Engenharia Reversa na ferramenta, constatou-se que a arquitetura da mesma encontra-se dividida em módulos e supermódulos (Apêndice A - Engenharia Reversa da Ferramenta MCards 1.0). Realizado o processo de manutenção, onde ocorreram modificações não apenas corretivas mas também perfectivas (manutenções que aprimoram a ferramenta além dos requisitos funcionais originais), houve uma alteração da arquitetura original da ferramenta MCards.

A nova arquitetura também é composta de módulos e supermódulos como a anterior, todavia a nomenclatura e funções de alguns módulos foram alteradas. Os módulos da nova arquitetura e a descrição de suas funções são:

- **Supermódulo Criador de Aplicações:** responsável pela manipulação visual da ferramenta. É através desse módulo que o usuário interage com a ferramenta para a criação de uma aplicação para dispositivo móvel. Esse módulo é composto pelos módulos de Edição Visual, Geração *forML* e Geração *messageML*.
 - **Módulo de Edição Visual:** responsável pela interação visual entre o usuário e a ferramenta. Nesse módulo é possível construir uma aplicação para dispositivos móveis através da utilização da técnica *drag and drop* (arrastar e soltar) para manipulação de componentes de interface.
 - **Módulo de Geração *forML* e Geração *messageML*:** responsáveis pela geração dos arquivos XML para especificação da aplicação a ser gerada e da mensagem a ser enviada por essa aplicação para um endereço Web.
- **Supermódulo Gerador de Aplicações:** esse módulo transforma o documento XML de especificação da aplicação a ser gerada, *forML*, em documentos contendo códigos de acordo com os tradutores existentes, além de gerar também os instaladores ou executáveis para a aplicação que está sendo gerada. Esse módulo é composto pelo módulo JAXB e pelos módulos geradores de aplicações para linguagens específicas.

- **Módulo JAXB:** responsável pela conversão do arquivo *forML* em objetos Java que serão utilizados pelos módulos de geração de aplicações disponíveis.
- **Módulo Gerador de Aplicações SuperWaba:** responsável pela tradução dos objetos Java gerados pelo JAXB para a linguagem SuperWaba e geração dos instaladores ou executáveis para a aplicação que está sendo gerada, ou seja, os arquivos para Palm OS (*.prc* e *.pdb*), Pocket PC (*.exe*), Linux (*.sh*) ou Symbian OS (*.sis*).

A Figura 5.5 mostra a nova arquitetura da ferramenta Mcards dividida em módulos e supermódulos:

Figura 5.5: Nova arquitetura da Ferramenta MCards versão 2.0

Na nova arquitetura ilustrada pela Figura 5.5, o usuário pode especificar a aplicação a ser gerada através do supermódulo criador de aplicações interagindo apenas com o módulo de edição visual. Ao especificar uma aplicação, o usuário define, como na arquitetura anterior, não apenas os componentes de interface, mas também um conjunto com parâmetros relacionados como a tecnologia para a qual será gerado o código, entre outros.

Durante, e após, o processo de criação da aplicação o usuário pode executar a geração do código para visualização, instanciando assim o supermódulo gerador de aplicações que, por sua vez, irá instanciar o módulo JAXB. O módulo JAXB faz uso do documento *forML* para a construção dos objetos Java que serão utilizados pelo gerador de aplicações da

tecnologia escolhida. Esse gerador de aplicações irá construir o arquivo de código-fonte em questão que poderá ser visualizado pelo usuário através do editor visual.

Além da geração do código para visualização, o usuário tem a possibilidade de solicitar a geração dos arquivos de instalação ou executáveis para aplicação que está sendo construída de acordo com a tecnologia escolhida. Para tanto, o supermódulo gerador de aplicações irá efetuar os passos descritos anteriormente para a construção do código-fonte da aplicação, sendo que este será utilizado pelo próprio gerador de aplicações da tecnologia para a compilação e construção dos arquivos necessários para a execução da aplicação ou sua instalação no dispositivo alvo. Esses arquivos ficam a disposição do usuário pois são gerados no diretório do projeto da aplicação que está sendo criada, ou seja, dentro do diretório *workspace* da ferramenta MCards.

5.5. Evolução da Ferramenta MCards

Como o objetivo do presente projeto de mestrado foi investigar o desenvolvimento de aplicações para PDAs com independência de plataforma, analisando a ferramenta MCards expandindo suas funcionalidades com a construção de um mecanismo que possibilite a geração de aplicações J2ME para dispositivos móveis que interagem com sistemas Web, foi necessário, além do entendimento da ferramenta e a execução de modificações para seu perfeito funcionamento, um estudo detalhado sobre a tecnologia de desenvolvimento para dispositivos móveis J2ME.

Como apresentado no Capítulo 2, a arquitetura J2ME define configurações e perfis. Cada combinação de configuração e perfil é otimizada para a capacidade de memória e processamento de uma determinada categoria de dispositivos.

Se, por um lado, o J2ME possui a vantagem de ser uma plataforma comum que é compatível com a maioria dos dispositivos móveis do mercado, por outro lado, possui a desvantagem de demandar esforços para a escolha da combinação mais apropriada para cada caso, pois uma aplicação gerada com um par de configuração e perfil específico para um tipo de dispositivo não pode ser facilmente utilizada para outro dispositivo de tipo diferente. As configurações e perfis utilizam conceitos distintos e conseqüentemente pacotes (APIs) distintos, o que gera códigos incompatíveis entre si.

Nesse contexto, a Seção 5.5.1 discute a problemática da definição do melhor par de configuração e perfil a ser utilizado, levando em consideração os dispositivos alvos da

ferramenta MCards, e a Seção 5.5.2 discute o módulo de geração de aplicações J2ME implementado na ferramenta MCards.

5.5.1. Configuração e Perfil Selecionado

Devido à particularidade da tecnologia J2ME de produzir códigos incompatíveis para diversos dispositivos, foi necessário um estudo minucioso das configurações e perfis existentes e uma definição cuidadosa dos dispositivos alvos da ferramenta MCards.

O módulo de geração de aplicação para SuperWaba gera aplicações para dispositivos com variado poder de memória e de processamento, além de variada combinação de tamanhos de tela e cores possíveis na interface; entretanto seu uso é restrito para PDAs. Deste modo, optou-se por gerar aplicações para os mesmos tipos de dispositivos: Pocket PC e Palm OS.

Definido o dispositivo alvo, o próximo passo foi estudar a tecnologia J2ME para assim poder definir a configuração e o perfil mais apropriado.

Após um estudo inicial, foram feitos testes com alguns perfis e configurações criando aplicações simples para poder avaliar a complexidade de desenvolvimento, a confiabilidade, desempenho e qualidade da interface das aplicações geradas. Para a execução de tais testes, foi necessária uma pré-seleção dos perfis que possibilitavam o desenvolvimento para PDAs, dentre os quais foram selecionados:

- Mobile Information Device Profile (MIDP): perfil para a configuração CLDC que oferece a base para a funcionalidade requerida pelas aplicações móveis, incluindo interfaces do usuário, conexões com a rede, persistência de dados e controle de aplicações.
- Personal Profile (PP): é o perfil CDC utilizado em dispositivos que necessitam de um suporte completo para interface ou *applet*, como PDAs e consoles para jogos. Ele inclui a biblioteca AWT completa e é fiel ao ambiente Web, executando facilmente *applets* feitos para ambientes desktop.

Após a análise de cada perfil, e a criação de aplicações com cada perfil, chegou-se à conclusão que o par de configuração e perfil mais apropriado para o desenvolvimento de aplicações para PDAs, mais especificamente, Pocket PC e Palm OS, é CDC/PP (Connected Device Configuration com Personal Profile).

Essa escolha se deu devido a grande distância entre os dois perfis com relação à criação de interface de usuário. O perfil MIDP possibilita a construção apenas de interfaces simples mais voltadas a celulares e PDAs antigos, enquanto o perfil PP oferece maiores recursos de interface, além de ser este o perfil mais indicado para PDAs mais poderosos em relação a processamento e memória, o que se encaixa com o Pocket PC e os novos dispositivos com Palm OS.

Abaixo segue abaixo uma descrição desses perfis analisados:

5.5.1.1. Mobile Information Device Profile - MIDP

O perfil MIDP traz as classes de interface com o usuário e acesso à rede, de maneira semelhante as *applets* e aos *servlets*, um aplicativo MIDP é chamado de *MIDlet*, por herdar de uma classe denominada *MIDlet* (*javax.microedition.midlet.MIDlet*). Com essa herança, a classe passa a ter a capacidade de se comunicar com o Gerenciador de Aplicativos que os dispositivos MIDP possuem e de interagir com a interface do aparelho.

Um aplicativo MIDP é formado por um conjunto de classes que implementam seu funcionamento, reunidas em um pacote chamado MIDlet Suite. O aplicativo é desenvolvido em um computador desktop, compilado, pré-verificado, empacotado (JAR) e então instalado em um dispositivo para execução ou em um emulador. O arquivo JAR que contém o aplicativo, é acompanhado por um descritor da aplicação, um arquivo JAD (*Java Application Descriptor*), que é processado pelo gerenciador de aplicativos do dispositivo, fornecendo informações sobre os requisitos de funcionamento da MIDlet Suíte e permitindo que os usuários decidam pela sua instalação ou não. O desenvolvimento de todos esses passos pode ser executado manualmente, ou otimizado por alguma ferramenta que execute todos esses passos de forma quase transparente.

O MIDP oferece ao programador algumas bibliotecas básicas para o desenvolvimento de aplicações. Essas bibliotecas abrangem:

- Bibliotecas Nucleares: o perfil MIDP adiciona algumas classes e funcionalidades aos pacotes *Java.lang* e *java.util* da CLDC;
- Ciclo de Vida da Aplicação: define o ciclo de vida da aplicação e a forma como é controlada pelo dispositivo (*javax.microedition.midlet*);
- Interface Gráfica: possui funções para construir interfaces gráficas e para obter o input do utilizador (*javax.microedition.lcdui*);

- Armazenamento Persistente: apesar de não existir o conceito de arquivo em MIDP, existem, no entanto, funções para ler e escrever dados num sistema de armazenamento persistente (*javax.microedition.rms*);
- Comunicação: o MIDP disponibiliza funções para atender ao sistema de comunicação do dispositivo, possibilita o uso de conexões HTTP, acesso a portas seriais, entre outros (*javax.microedition.io*);
- Multimídia: possui funções multimídia como geração de tons, notas MIDI, reprodução de arquivos áudio e vídeo, etc (*javax.microedition.media*);
- Segurança: possui uma biblioteca relacionada com certificados usados em ligações seguras (*javax.microedition.pki*).

A Figura 5.6 mostra exemplos de interfaces de aplicações criadas com a utilização do MIDP. Como se pode notar, as interfaces são relativamente simples pois o perfil MIDP é voltado a dispositivos com baixo poder pra interfaces gráficas.

Figura 5.6: Exemplos de interfaces possíveis com o perfil MIDP

5.5.1.2. Personal Profile - PP

O Personal Profile é um dos perfis da configuração CDC destinado a dispositivos mais poderosos que um celular, normalmente um PDA. Esta API é basicamente uma parte do J2SDK 1.3, incluindo boa parte da API de AWT para o desenvolvimento de aplicações gráficas.

Como inclui a biblioteca AWT completa e é fiel ao ambiente Web, executa facilmente *applets* feitos para ambientes desktop, o que aumenta as possibilidades de criação de interface. Seu desenvolvimento assemelha-se ao desenvolvimento de um *applet*, o que facilita seu aprendizado e o processo de desenvolvimento de aplicações. A API padrão do CDC/Personal Profile não inclui nenhuma API de acesso a dados, mas inclui um pacote opcional de JDBC, sem incluir todas as classes do pacote *Java.SQL* padrão. A Tabela 5.1 mostra uma comparação entre as APIs do J2SE 1.3.1 e APIs que estão disponíveis no perfil Personal Profile. E a Figura 5.7 mostra exemplos de interfaces de aplicações criadas com a utilização do PP. As interfaces possíveis são bem variadas seguindo o padrão AWT.

Tabela 5.1: Comparação entre J2SE 1.3.1 e CDC/Personal Profile

Pacote	J2SE 1.3.1	PP 1.0
java.applet.*	Suporte total ao J2SE 1.3.1	Suporte total ao J2SE 1.3.1
java.awt.*	Suporte total ao J2SE 1.3.1	Suporte total ao J2SE 1.3.1
java.beans.*	Suporte total ao J2SE 1.3.1	Suporte parcial ao J2SE 1.3.1
java.io.*	Suporte total ao J2SE 1.3.1	Suporte total ao J2SE 1.3.1
java.lang.*	Suporte total ao J2SE 1.3.1	Suporte total ao J2SE 1.3.1
java.math	Suporte total ao J2SE 1.3.1	Suporte total ao J2SE 1.3.1
java.net.*	Suporte total ao J2SE 1.3.1	Suporte total ao J2SE 1.3.1
java.rmi.*	Suporte total ao J2SE 1.3.1	Substituído por um pacote opcional
java.security.*	Suporte total ao J2SE 1.3.1	Suporte total ao J2SE 1.3.1
java.sql.*	Suporte total ao J2SE 1.3.1	Substituído por um pacote opcional
java.text.*	Suporte total ao J2SE 1.3.1	Suporte total ao J2SE 1.3.1
java.util.*	Suporte total ao J2SE 1.3.1	Suporte total ao J2SE 1.3.1
javax.accessibility.*	Suporte total ao J2SE 1.3.1	Não suportado
javax.naming.*	Suporte total ao J2SE 1.3.1	Não suportado
javax.rmi.*	Suporte total ao J2SE 1.3.1	Não suportado
javax.sound.*	Suporte total ao J2SE 1.3.1	Não suportado
javax.swing.*	Suporte total ao J2SE 1.3.1	Não suportado
javax.transaction.*	Suporte total ao J2SE 1.3.1	Não suportado
org.omg.*	Suporte total ao J2SE 1.3.1	Não suportado
javax.microedition.io.*	Não suportado.	Adicionado no PP
javax.microedition.xlet.*	Não suportado.	Adicionado no PP

Figura 5.7: Exemplos de interfaces possíveis com o perfil PP

5.5.2. Módulo de Geração de Aplicações J2ME

Após a definição de qual perfil e configuração do J2ME deveriam ser utilizados, foi realizada a implementação do módulo de geração de aplicações J2ME. Esse módulo é responsável por realizar a transformação da especificação da aplicação criada pelo usuário, o artefato *forML*, em código para a linguagem J2ME e seus respectivos arquivos executáveis e instaladores.

O módulo de geração de aplicações J2ME encarrega-se de transformar os objetos Java gerados pelo módulo JAXB através da manipulação do *forML* (artefato XML gerado pela manipulação do usuário no editor visual) em código para a tecnologia J2ME com a utilização da configuração CDC e do perfil Personal Profile, e compilar o código gerado dando origem aos arquivos da aplicação J2ME que serão utilizados para que a mesma seja instalada no dispositivo.

Como descrito na Seção 5.4, após o processo de Engenharia Reversa e manutenção da ferramenta MCards, foi definida uma nova arquitetura para a ferramenta, sendo que esta arquitetura foi novamente modificada com a inserção deste novo módulo de geração de aplicações (Figura 5.8).

Figura 5.8: Arquitetura MCards com Módulo de Geração de Aplicações J2ME

Nessa nova arquitetura, durante a especificação da aplicação através do supermódulo criador de aplicações, o usuário define para qual tecnologia a aplicação será gerada — nesta segunda versão da ferramenta MCards o usuário pode escolher entre SuperWaba e J2ME — e, ao iniciar os processos de geração de código e geração de executável para a aplicação, o supermódulo gerador de aplicações instancia o módulo de geração de aplicações para a tecnologia escolhida. Este, por sua vez, irá disponibilizar para o usuário o resultado de suas execuções: o arquivo de código para ser visualizado dentro do próprio editor visual e os arquivos de instalação da aplicação gerada que podem ser acessados para instalação no dispositivo pretendido.

O módulo de geração de aplicações J2ME está dividido em dois submódulos. A Figura 5.9 ilustra essa subdivisão do módulo de geração de aplicações J2ME:

- **Tradutor J2ME:** que faz a geração do código Java;
- **Gerador J2ME:** que utiliza o código recém gerado para a geração da aplicação para ser instalada no dispositivo alvo.

O tradutor J2ME faz uso dos objetos Java gerados pelo módulo JAXB para a composição do código-fonte da aplicação que está sendo criada. Nesse código, assim como no código gerado para outras linguagens, têm-se os elementos necessários para a formação de: interface com o usuário; controle de persistência no dispositivo; controle de eventos; e comunicação HTTP para envio de dados encapsulados no padrão XML.

Figura 5.9: Módulo de Geração de Aplicações J2ME

O gerador J2ME é responsável por receber o código-fonte recém gerado pelo tradutor, compilá-lo e gerar os arquivos necessários para a execução da aplicação no dispositivo. Para tanto faz uso de um compilador Java.

Os arquivos gerados para uma aplicação J2ME consistem em um arquivo executável Java (.jar) e um arquivo de atalho para execução do arquivo JAR (.lnk). Informações de como fazer uso desses arquivos nos diversos tipos de dispositivos podem ser encontradas no “Manual da Ferramenta MCards 2.0”(Apêndice B).

5.6. Exemplo de Aplicação Gerada com MCards 2.0

Como descrito no Capítulo 4, a ferramenta MCards apóia o desenvolvimento de aplicações para dispositivos móveis, especialmente para PDAs, com geração de aplicações baseadas em *cards*. Dentro desse contexto, essa seção ilustra uma aplicação desenvolvida com o uso da ferramenta MCards, versão 2.0 explorando suas funcionalidades para a construção de interface gráfica e armazenamento de dados, sendo que a aplicação deverá ser gerada para a tecnologia J2ME.

A aplicação criada trata-se de um cadastro de clientes de uma empresa. Os funcionários dessa empresa querem cadastrar seus clientes de forma rápida, preenchendo

apenas dados básicos como nome, endereço, telefone e se o cliente deseja receber informações sobre os produtos da empresa. Para tal, a empresa possui alguns PDAs nos quais será instalada a aplicação a ser criada.

A Figura 5.10 ilustra a criação das telas da aplicação de cadastro de clientes através do uso da ferramenta MCards.

Figura 5.10: Criação das telas da aplicação pra cadastro de clientes

Foram criadas duas telas para a aplicação de cadastro de clientes (Figura 5.11): (a) a primeira tela possui os campos para preenchimento do nome, endereço, telefone e e-mail que utilizam o componente *Edit*, além do campo para a escolha do sexo com a utilização do componente *ComboBox*, e o campo para recolhimento da assinatura do cliente que utiliza o componente *Stroke*; (b) a segunda tela é composta pelos dois componentes *ComboBox* e *List* para o preenchimento de duas perguntas relativas ao recebimento de informações sobre produtos da empresa pelo usuário. Além dos componentes para preenchimento de dados, as duas telas possuem também componentes para os rótulos de texto relativos aos componentes de dados: os *Labels*.

CadastroCliente (a)

Nome:

Sexo: [Masculino... ▼]

Endereco:

Telefone:

e-mail:

Assinatura: Strokes

Próxima Tela

CadastroCliente (b)

Deseja receber informações sobre nossos produtos?

[Sim... ▼]

Se desejar, escolha um meio para receber essas informações:

[Carta, NULL]
[e-mail, NULL]
[Telemarketing,...]

Voltar

save (c) <- -> del

Figura 5.11: Telas da aplicação pra cadastro de clientes

Os mecanismos de gerenciamento e armazenamento dos dados são transparentes ao desenvolvedor que está criando a aplicação, pois são gerados pela ferramenta MCards. O armazenamento e o acesso aos registros armazenados são possíveis através da barra de navegação presente nas telas que possuem campos de dados (Figura 5.11 c).

A Figura 5.12 mostra o arquivo *forML* de especificação gerado para a aplicação que está sendo criada. Este arquivo possui a especificação abstrata de todas as telas definidas para a aplicação e dos componentes inseridos em cada tela com seus respectivos parâmetros. Possui, também, parâmetros necessários para a geração da aplicação como o tamanho da tela e a tecnologia para a qual o código e a aplicação serão gerados. Entre os parâmetros definidos está a URL para a qual a aplicação poderá enviar os dados, neste caso essa URL é o endereço Web do servidor da empresa para onde os dados serão enviados e serão armazenados no banco de dados da empresa. A Figura 5.13 ilustra uma mensagem enviada pela aplicação em execução contendo apenas um registro, ou seja, o cadastro de um cliente. Não há restrição do número de registros que podem ser enviados em uma mensagem.

```

<?xml version="1.0" encoding="ISO-8859-1" ?>
<forML>
<menu about="Cadastro de Clientes - Versão 1.0" help="Para salvar e navegar entre registros utilize a barra de navegação." />
<parameters height="320" width="240" code="J2ME" title="CadastroCliente" url="http://www.empresaX.com.br" store="true" />
<components>
<form id="0" backgroundColor="201,218,231" navigatorbar="true">
<myLabel name="lbl_Nome" foregroundcolor="0,51,102" position="8,6" Text="Nome:" dimension="35,30" />
<myEdit name="edt_Nome" backgroundColor="255,255,255" foregroundcolor="0,0,0" lines="1" position="65,6" size="169,20"
dimension="169,20" />
<myLabel name="lbl_Sexo" foregroundcolor="0,51,102" position="8,36" Text="Sexo:" dimension="35,30" />
<myComboBox backgroundColor="210,210,210" foregroundcolor="0,0,0" position="65,36"
items="Masculino~NULL#Feminino~NULL#" name="cmb_Sexo" events="GOTO" dimension="82,20" />
<myLabel name="lbl_End" foregroundcolor="0,51,102" position="8,66" Text="Endereco:" dimension="67,30" />
<myEdit name="edt_end" backgroundColor="255,255,255" foregroundcolor="0,0,0" lines="1" position="65,66" size="163,20"
dimension="163,20" />
<myLabel name="lbl_Fone" foregroundcolor="0,51,102" position="8,96" Text="Telefone:" dimension="63,30" />
<myEdit name="edt_Fone" backgroundColor="255,255,255" foregroundcolor="0,0,0" lines="1" position="65,96" size="79,20"
dimension="79,20" />
<myLabel name="lbl_email" foregroundcolor="0,51,102" position="8,126" Text="e-mail:" dimension="49,30" />
<myEdit name="edt_email" backgroundColor="255,255,255" foregroundcolor="0,0,0" lines="1" position="65,126" size="79,20"
dimension="79,20" />
<myLabel name="Label_Ass" foregroundcolor="0,51,102" position="8,156" Text="Assinatura:" dimension="77,30" />
<myStrokes position="65,156" color="null" option="1" dimension="162,58" />
<mybutton name="btn_next" backgroundColor="236,233,216" foregroundcolor="0,51,102" Text="Próxima Tela"
position="146,244" dimension="86,25" event="GOTO" option="0" />
</form>
<form id="1" backgroundColor="214,228,209" navigatorbar="true">
<myLabel name="lbl_info" foregroundcolor="0,94,47" position="8,6" Text="Deseja receber informações sobre nossos produtos?"
dimension="217,60" />
<myComboBox backgroundColor="210,210,210" foregroundcolor="0,0,0" position="16,56" items="Sim~NULL#Não~NULL#"
name="cmb_info" events="GOTO" dimension="50,20" />
<myLabel name="lbl_meio" foregroundcolor="0,94,47" position="8,90" Text="Se desejar, escolha um meio para receber essas
informações:" dimension="280,60" />
<myList name="lst_meio" backgroundColor="255,255,255" foregroundcolor="0,0,0" position="16,140" events="GOTO"
dimension="92,54" items="Carta~NULL# e-mail~NULL# Telemarketing~NULL#" />
<mybutton name="btn_back" backgroundColor="236,233,216" foregroundcolor="0,94,47" Text="Voltar" position="161,244"
dimension="72,25" event="GOTO" option="0" />
</form>
</components>
</forML>

```

Figura 5.12: Arquivo *forML* gerado para a aplicação de cadastro de clientes

```

<?xml version="1.0" encoding="ISO-8859-1" ?>
<MessageML isStoreinPda="true" application="CadastroCliente">
<Data>
<Container id="0">
<Record id="1">
<ink>
<trace>0,4,85,5,86,582,64,58,25,36,23,65,68,69,32,45,88,-1</trace>
<trace>80,25,26,99,0,5,-1</trace>
</ink>
<edit name="edt_Nome">Fulano de Tal</edit>
<combobox name="cmb_Sexo">Masculino</combobox>
<edit name="edt_end">Rua das Avenidas, 88</edit>
<edit name="edt_Fone">3366-6633</edit>
<edit name="edt_email">fulano@email.com</edit>
</Record>
</Container>
<Container id="1">
<Record id="1">
<combobox name="cmb_info">Sim</combobox>
<list name="lst_meio">e-mail</list>
</Record>
</Container>
</Data>
</MessageML>

```

Figura 5.13: Mensagem a ser enviada para URL durante a execução da aplicação de cadastro de clientes

A Figura 5.14 mostra um trecho do arquivo de código J2ME gerado para a aplicação de cadastro de clientes.

```

//Code generated automatically for communication between PDA
//and Web Application;
//The following parameters had been defined for the creation
//of the interface
//Store          = true
//Code           = J2ME
//Windows        = 2
//Labels         = 10
//Edits          = 4
//InputStroke    = 1
//Button         = 2
//ComboBox       = 2
//ListBox        = 1
//TextArea       = 0
//Links          = 0
//*****
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;
import java.io.*;
import java.net.*;

public class CadastroCliente extends Frame {
    private static final long serialVersionUID = 1L;
    static final String CREATOR_ID = "1788";
    static final String NAME = "Cada";
    static final String CATALOG_NAME = NAME+"."+CREATOR_ID;
    static Vector panels = new Vector(6);
    private boolean laidOut = false;
    Strokes stroke;
    HandlerStrokes handlerStrokes = new HandlerStrokes();
    HandlerRecord handlerrecord = new HandlerRecord();
    MenuBar menubar;
    Button btn_next;
    Choice cmb_Sexo;
    List lst_meio;
    Button btn_back;
    Choice cmb_info;

    public CadastroCliente() {
        super();

        ...

        setLayout(null);
        setTitle("CadastroCliente");
        Menu menu = new Menu("Menu");
        menu.add(new MenuItem("Help"));
        menu.add(new MenuItem("About"));
        menu.add(new MenuItem("Send"));
        menu.add(new MenuItem("Exit"));
        Menu communication = new Menu("Communication");
        communication.add(new CheckboxMenuItem("Wireless", true));
        menubar = new MenuBar();
        menubar.add(menu);
        menubar.add(communication);
        setMenuBar(menubar);
    }

    public static void main(String argv[]) {
        CadastroCliente app = new CadastroCliente();
        app.setFont(new Font("Arial", Font.PLAIN, 11));
        app.setSize(240, 320);
        app.show();
    }

    ...
}

```

Figura 5.14: Trecho do código J2ME gerado para a aplicação de cadastro de clientes

5.7. Dificuldades e Limitações

Dentre as dificuldades encontradas, pode-se definir como principal a falta de documentação do protótipo original da ferramenta MCards, o que dificultou seu entendimento tanto na visão de usuário, ou seja, aprender a fazer uso da mesma, quanto na visão de

desenvolvedor no entendimento do código em si para extensão da ferramenta. Essa ausência de documentação foi causada pelo fato da ferramenta ser um protótipo desenvolvido durante um projeto de mestrado para a prova de conceito de um objetivo maior, e levou à necessidade da aplicação de um processo de Engenharia Reversa para que, partindo apenas do código-fonte e da interface de usuário, se obtivesse um modelo abstrato da ferramenta.

Inicialmente não foi aplicado modelo algum de Engenharia Reversa para a recuperação das visões da ferramenta. Todavia, durante o processo de entendimento da ferramenta, notou-se a necessidade da utilização de um modelo de processo de Engenharia Reversa específico para ser seguido. O modelo de processo escolhido foi o FUSION-RE/I devido ao mesmo partir apenas da interface e do código-fonte e ser indicado para ferramentas desenvolvidas sob o modelo de orientação de objetos.

Superadas as dificuldades de ausência de documentação e entendimento da ferramenta, e com base nos requisitos levantados e análise da demanda por expansões na ferramenta, decidiu-se por executar manutenções corretivas para um melhor funcionamento da mesma. Durante o trabalho de manutenção corretiva foram realizadas diversas entrevistas informais com os usuários, tanto antes quanto durante a implementação das correções, para conhecer os principais problemas enfrentados por eles durante o uso da ferramenta. Tais discussões com usuários trouxeram a necessidade de outras manutenções como a adaptativa, que atualizou a ferramenta perante as novas versões da tecnologia SuperWaba, e a perfectiva, que aprimorou a ferramenta além dos requisitos funcionais originais visando tornar a ferramenta MCards mais atrativa para o usuário.

Outro ponto importante durante a implementação do módulo de geração de aplicações J2ME foi a necessidade de conhecer os perfis e configurações, inclusive implementando aplicações testes com os perfis mais relevantes, a fim de definir o par de perfil e configuração mais adequado às necessidades da ferramenta MCards, dificuldade esta descrita na Seção 5.5.

Infelizmente, mesmo com a implementação das manutenções, não foi possível sanar todos os problemas da ferramenta por limitações de tempo. Para tanto seria interessante a realização de uma avaliação de usabilidade com os usuários da ferramenta MCards a fim de identificar os pontos a serem alterados como, por exemplo, a inserção de novos componentes de interface e de dados para auxiliar na construção das aplicações, possibilitando ao usuário uma maior gama de opções durante o processo de criação de aplicações, pode-se citar como exemplos de componentes os *RadioButton*, *CheckBoxs* e imagens.

5.8. Considerações Finais

O principal resultado obtido com a realização da Engenharia Reversa foi o entendimento da ferramenta e documentação da mesma, o que auxiliou no processo reconhecimento de requisitos para manutenção da ferramenta MCards, dando uma base maior para a especificação e implementação da evolução da mesma, ou seja, a inserção de um mecanismo para possibilitar a geração de aplicações J2ME. Essas atividades deram origem ao módulo de geração de aplicações J2ME.

O módulo de geração de aplicações J2ME transforma a especificação abstrata, gerada pela manipulação do usuário no editor visual da ferramenta MCards, em código para a tecnologia J2ME com a utilização do par de configuração e perfil CDC / Personal Profile, compilando o código gerado dando origem aos arquivos da aplicação J2ME que serão utilizados para instalação no dispositivo móvel

A nova versão da ferramenta MCards está, em comparação com a primeira versão, mais fácil de usar e mais confiável, ou seja, menos suscetível a falhas de execução. A Tabela 5.2 mostra a tabela comparativa entre ferramentas mostrada nos Capítulos 3 e 4, atualizada com a nova versão da ferramenta MCards:

Tabela 5.2: Tabela comparativa entre ferramentas e as duas versões da ferramenta MCards

Ferramenta	Persistência	Linguagem Utilizada	Editor Visual	Tecnologia	Plataforma / Dispositivo	Licença	Cronologia
HyperCard	cards	HyperTalk	Sim	-	Desktop - MacOS	Sim	Foi lançado em 1987 e se manteve até 2004
XIML	Definida pelo usuário	XIML	Não	-	Várias	Não	Final da década de 1990
LiquidUI	Não	UIML	Sim	J2SE 1.3, HTML, WML e VoiceXML	Desktop e celulares	Sim	Final da década de 1990
TERESA	Não	XML	Sim	XHTML, WML e VoiceXML	Desktop e celulares	Não	Recente: surgiu por volta de 2002
MobiForms	Definida pelo usuário	-	Sim	J2ME	Pocket PC, Symbian OS e Palm OS	Sim	Recente: lançado por volta de 2002
Simplicity	Definida pelo usuário	-	Sim	J2ME	Desktop, Web, Symbian OS, Pocket PC e Palm OS	Sim	Recente: lançado por volta de 2003
MCards	cards	XML	Sim	SuperWaba	Pocket PC e Palm OS	Não	Início: 2003
MCards 2.0	cards	XML	Sim	SuperWaba e J2ME	Pocket PC, Symbian OS e Palm OS	Não	Início: 2005

Pode-se notar, ao analisar a Tabela 5.2, que a nova versão da ferramenta MCards destaca-se perante a versão original por gerar aplicações não apenas com a tecnologia SuperWaba mas também com J2ME, gerando aplicações para os dispositivos com Palm Os e Pocket PC, além de gerar aplicações também para os dispositivos que possui o sistema operacional Symbian OS.

6. CONCLUSÃO

6.1. Discussão dos Resultados

Verifica-se, na atualidade, um grande avanço no número de componentes de hardware de custo e tamanho reduzidos como PDAs, celulares e *smartphones* e, conseqüentemente, está ocorrendo também um grande avanço no uso desses dispositivos, não só no meio acadêmico, mas também no mundo corporativo e para uso pessoal. Essa proliferação demanda, cada vez mais, por aplicações que façam uso de tais equipamentos. Todavia, existem poucas ferramentas que auxiliam o processo de desenvolvimento para tais dispositivos, o que dificulta e, principalmente, acarreta um maior tempo de desenvolvimento.

Um dos fatores que contribuem para esse baixo número de ferramenta que auxiliam tal desenvolvimento é o fato de o mercado ainda ser dominado, principalmente, por apenas duas grandes plataformas: a Palm (Palm OS) e a Microsoft (WinCe e Pocket PC), sendo que cada uma possui ambientes de desenvolvimento diferentes com linguagens de desenvolvimento específicas. Dentro desse contexto, vem crescendo a cultura de desenvolvimento com a linguagem Java com a utilização não apenas da tecnologia J2ME, mas também com tecnologias alternativas como o SuperWaba.

Uma das ferramentas que segue essa tendência é a MCards, um ambiente para desenvolvimento rápido (RAD) de aplicações Java para dispositivos móveis, especificamente PDAs. A ferramenta MCards utiliza uma estrutura de persistência e organização da interface baseadas no conceito de cards e faz uso de linguagem de marcação baseadas em XML para definir a interface, gerando aplicações com o uso da tecnologia SuperWaba para os dispositivos Pocket PC e Palm OS.

O objetivo do presente projeto de mestrado foi investigar o desenvolvimento de aplicações para dispositivos móveis, especificamente para PDAs, com independência de plataforma, analisando e expandindo a ferramenta MCards com o propósito de permitir a

geração de aplicações para dispositivos móveis com a utilização da tecnologia J2ME. Para tanto, foi necessária, inicialmente, a aplicação de um processo de Engenharia Reversa na ferramenta MCards de modo a possibilitar um melhor entendimento da ferramenta e assim realizar o levantamento de novos requisitos e definir as expansões necessárias na mesma. O processo de Engenharia Reversa, utilizado para recuperar as visões funcionais e estruturais da ferramenta MCards, foi baseado no modelo FUSION-RE/I, por se tratar de um modelo de processo de Engenharia Reversa indicado para ferramentas desenvolvidas sob o modelo de orientação de objetos e partir apenas da interface e do código-fonte para a recuperação das visões. Além do processo de Engenharia Reversa, foi necessária a execução de manutenções na ferramenta a fim de assegurar um melhor funcionamento e inserir melhorias nas suas funcionalidades.

Destaca-se como a principal atividade o estudo da tecnologia J2ME com a problemática da definição do par de configuração e perfil mais apropriado para, então, prover o mecanismo para construção de aplicações J2ME, o que ocasionou na implementação do módulo de geração de aplicações J2ME e sua incorporação à ferramenta MCards. O módulo de geração de aplicações J2ME transforma a especificação abstrata, gerada pela manipulação do usuário no editor visual, em código para a tecnologia J2ME com a utilização da configuração CDC e do perfil Personal Profile e compila o código gerado dando origem aos arquivos da aplicação J2ME que serão utilizados para que a mesma seja instalada no dispositivo.

Com o processo de Engenharia Reversa, as manutenções executadas na ferramenta e a implementação do módulo de geração de aplicações J2ME ocorreram várias modificações na arquitetura da ferramenta MCards, sendo a nova arquitetura dividida em módulos e supermódulos: Supermódulo Criador de Aplicações; Módulos de Edição Visual, de Geração *forML* e de Geração *messageML*; Supermódulo Gerador de Aplicações; Módulos JAXB, Gerador de Aplicações SuperWaba e o Módulo Gerador de Aplicações J2ME.

A nova versão da ferramenta MCards está, em comparação com a primeira versão, mais fácil de usar e mais confiável, ou seja, menos suscetível a falhas de execução. Essa nova versão destaca-se, perante a versão original, por gerar aplicações não apenas com a tecnologia SuperWaba mas também com J2ME, além de gerar aplicações também para os dispositivos que possui o sistema operacional Symbian OS, que vem crescendo e conquistando cada vez mais lugar no mercado.

6.2. Trabalhos Futuros

As possíveis atividades que podem dar continuidade a este trabalho são:

- Construção de novos módulos geradores de código e de aplicações como, por exemplo, o módulo para a construção de aplicações com a utilização da tecnologia J2ME com o par de perfil e configuração CLDC/MIDP (*MIDlet*) para a geração de aplicações para telefones celulares e dispositivos dotados de menor poder de memória e de processamento. Outra opção seria o módulo gerador de *applets* Java, ou seja, para que as aplicações também possam ser executadas em ambiente Web;
- Inclusão de novos componentes de interface e armazenamento de dados como *RadioButton*, *CheckBoxs*, imagem e mensagens *pop-up*;
- Inclusão de novos parâmetros de configuração de componentes como fontes e tamanhos de fontes;
- Adição de funcionalidades para alinhamento de componentes e outras funcionalidades que auxiliem ao usuário desenvolvedor na construção da interface das aplicações;
- Incorporação de emuladores ou simuladores para os dispositivos na ferramenta MCards para que as aplicações criadas possam ser executadas e testadas sem a necessidade de instalação da mesma em um dispositivo;
- Inclusão de um editor XML que possibilite ao usuário desenvolvedor executar alterações diretamente no documento *forML*, dando ao mesmo uma maior interação com o código XML da aplicação especificada;
- Implementação de um sistema de reorganização e adaptação da interface em caso de redimensionamento de tela, ou mesmo para que o usuário tenha a possibilidade de criar aplicações adaptáveis às configurações de tela possíveis entre os vários dispositivos disponíveis no mercado;
- Avaliação com os usuários da ferramenta MCards a fim de identificar novos requisitos para a ferramenta;
- Avaliação da interface com a finalidade de verificar seus níveis de usabilidade. Após essa avaliação, alterações na interface podem ser necessárias com o objetivo de melhorar a usabilidade da ferramenta;

- Testes e usos intensivos da ferramenta devem ser estimulados, nos cursos de graduação e pós-graduação do ICMC / USP, e para desenvolvedores de modo geral.

REFERÊNCIAS

- Abowd, G. D. (1999). **Classroom 2000**: an experiment with the instrumentation of a living educational environment. IBM Systems Journal 38(4), p. 508–530.
- Abowd, G. D. et al. (1998). **Investigating the capture, integration and access problem of ubiquitous computing in an educational setting**. Proc. CHI'98, ACM, New York, p. 440-447.
- Abowd, G. D. et al., (1996). **Teaching and learning as multimedia authoring**: The classroom 2000 project. In Proceedings of the 1996 ACM Multimedia Conference, p. 187–198.
- Abowd, G. D.; Mynatt, E. D. (2000). **Charting past, present, and future research in ubiquitous computing**. ACM Transactions on Computer-Human Interaction (TOCHI), 7(1):29–58.
- Abrams, M. et al. (1999). **UIML**: An Appliance-Independent XML User Interface Language. Proc. 8th WWW Conf., 1999.
- Boccardo, D. R. (2005). **MCards**: Ferramenta de apoio à construção de aplicativos para assistentes pessoas digitais. Dissertação (Mestrado) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos.
- Chikofsky, E. J.; Cross, J. H. (1990). **Reverse engineering and design recovery**: A taxonomy. IEEE Software, 7(1): p.13–17.
- Coleman, D. et al. (1994). **Object-Oriented Development**: The FUSION Method. Prentice Hall.
- Costa, R. M. (1997). **Um método de engenharia reversa para auxiliar a manutenção de software**. Dissertação (Mestrado) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos.
- Covington, M. J. et al. (2001). **Securing context-aware applications using environment roles**. Em Proceedings of the Sixth ACM Symposium on Access control models e technologies, p. 10–20. ACM Press.
- Dey, A. K. (2001). **Understanding and Using Context**. Special issue on Situated Interaction and Ubiquitous Computing, 5(1): p. 4-7, 2001.
- Harandi, M. T.; Ning, J. Q. (1990). **Knowledge-based program analysis**. IEEE Software, 7: p. 74–81.
- Higel, S.; O'Donnell, T.; Wade, V. (2003). **Towards a natural interface to adaptive service composition**. Em Proceedings of the 1st International Symposium on Information e communication technologies, p. 169–174. Trinity College Dublin.

- Kidd, C. D. et al. (1999). **The aware home**: A living laboratory for ubiquitous computing research. Em Cooperative Buildings, p. 191–198.
- Mateus, G. R.; Loureiro, A. A. F. (1998). **Introdução à Computação Móvel**.
- Mori, G.; Paternò, F.; Santoro, C. (2004). **Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions**. IEEE Transactions on Software Engineering (August 2004, p.507-520).
- Mori, G.; Paternò F.; Santoro C. (2003). **Tool Support for Designing Nomadic Applications**. Proceedings of the 2003 international conference on Intelligent user interfaces
- Mynatt, E. D. et al. (2001). **Digital family portraits**: supporting peace of mind for extended family members. Em Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, p. 333–340. ACM Press.
- Nakanishi, Y.; Sato, Y.; Koike, H. (2002). **Enhanceddesk and Enhancedwall**: Augmented desk and wall interfaces with real-time tracking of user's motion. Em Ubicomp – Workshop on Collaborations with Interactive Walls and Tables, p. 27–30.
- Nagel, K. et al. (2001). **The family intercom**: Developing a context-aware audio communication system. Em Proceedings of the 3rd International Conference on Ubiquitous Computing, p. 176–183. Springer-Verlag.
- Pressman, R. S. (2002). **Engenharia de Software**. Makron Books.
- Puerta, A.; Eisenstein, J. (2006). **XIML**: A Universal Language for User Interfaces. Disponível em: <http://www.xml.org/documents/XimlWhitePaper.pdf> e acessado em Janeiro de 2006.
- Salber, D.; Dey, A. K.; Abowd, G. A. (1999). **The Context Toolkit**: aiding the development of context-enabled applications. In Proceedings of the SIGCHI conference on Human factors in computing systems, p. 434–441. ACM Press.
- Siewiorek, D. P. (2002). **New frontiers of application design**. Commun. ACM, 45(12): p.79–82.
- Swanson, E.B., (1976). **The Dimensions of Maintenance**, Proc. 2nd Intl. Conf. Software Engineering, IEEE, pp. 492-497.
- Topley, K (2002) **J2ME in a Nutshell**, Ed. O'Reilly.
- Truong, K. N.; Abowd, G. D.; Brotherton, J. A. (2001). **Who, what, when, where, how**: Design issues of capture & access applications. In Proceedings of the 2001 International Conference on Ubiquitous Computing, p. 209– 224.
- Weiser, M. (1993). **Some Computer Science Issues in Ubiquitous Computing**. Communications of the ACM, 36(7):75–84.
- Weiser, M. (1991). **The computer for the 21st century**. In Scientific American 265, 3, p. 94–104.
- Vanderdonckt J., Limbourg Q., Florins M. (2001). **Synchronized Model-Based Design of Multiple User Interfaces**. Proceedings of Workshop on Multiple User Interfaces over the Internet: Engineering and Applications Trends, A. Seffah, T. Radhakrishnan & G.Canals (Eds.), Lille, 2001.

GLOSSÁRIO DE TERMOS

.NET - Framework desenvolvido pela Microsoft para rodar aplicações escritas em diferentes linguagens transparentemente, umas para as outras. Este framework permite que qualquer aplicação seja executada de qualquer lugar do mundo, em qualquer dispositivo.

API - *Application Programming Interface* é um conjunto de rotinas e padrões estabelecidos por um software para utilização de suas funcionalidades. De modo geral, a API é composta por uma série de funções acessíveis somente por programação, e que permitem utilizar características do software menos evidentes ao usuário tradicional.

Bluetooth – É uma tecnologia de baixo custo para a comunicação sem fio entre dispositivos móveis. É usado para comunicação entre pequenos dispositivos de uso pessoal, como PDAs, telefones celulares de nova geração, computadores portáteis, mas também é utilizado para a comunicação de periféricos, como impressoras, scanners, e qualquer dispositivo dotado de um chip *Bluetooth*.

BSS – *Basic Service Set*, unidade de arquitetura do protocolo IEEE 802.11.

Bytecodes – Os fontes da linguagem Java são compilados para uma forma intermediária de código denominada *bytecodes*, que são interpretados pelas Máquinas Virtuais Java (JVM). É essa característica que faz com que os programas Java sejam independentes de plataforma, executando em qualquer sistema que possua uma JVM.

CDC – *Connected Device Configuration*, configuração do J2ME destinada a dispositivos com conexões de rede intermitentes.

CLDC – *Connected Limited Device Configuration*, configuração do J2ME destinada a dispositivos com conexões mais rápidas e que possuem maior capacidade de memória e de processamento.

DOM – *Document Object Model* (Modelo de Objeto de Documentos) é uma API para a Internet (W3C) para acesso e modificação de documentos XML.

FP - *Foundation Profile*, perfil do J2ME que pode ser usado para construir aplicações sem interface do usuário.

GPS – *Global positioning system* (Sistema de Posicionamento Global), um sistema de navegação por satélite utilizado para determinação da posição na superfície da Terra ou em órbita. É constituído por uma constelação de 24 satélites.

IEEE 802.11 – Protocolo padrão em redes sem fio para microcomputadores, trabalha na banda de frequências de 900Mhz, não necessitando de autorização especial para funcionamento (a mesma frequência dos telefones sem fio domésticos). Tornou-se padrão nos últimos anos trabalhando nas seguintes modalidades 802.11a, 802.11b e 802.11g que correspondem respectivamente a 5 Mbits, 11 Mbits e 54 Mbits de velocidade máxima de transferência.

J2EE – *Java 2 Enterprise Edition*.

J2ME – *Java 2 Micro Edition*, plataforma Java para dispositivos compactos.

J2SE – *Java 2 Second Edition*.

Java – Linguagem de programação orientada a objetos, projetada e implementada por um pequeno grupo de pessoas coordenado por James Gosling na empresa Sun Microsystems em 1991. Criada inicialmente com a finalidade de ser utilizada em eletrodomésticos, se tornou popular pelo seu uso na Internet.

JAXB – *Java Architecture for XML Binding*, API para mapeamento Java-XML usada para gerar código Java a partir de documentos XML.

JVM – *Java Virtual Machine* (Máquina Java Virtual) carrega e executa os aplicativos, convertendo os bytecodes em código executável de máquina. A JVM é responsável pelo gerenciamento dos aplicativos, na medida em que são executados.

Lua – É uma linguagem de programação poderosa e leve, projetada para estender aplicações, também é frequentemente usada como uma linguagem de propósito geral. Foi criada por um time de desenvolvedores da PUC-Rio, a princípio, para ser usada em um projeto da Petrobrás. Devido à sua eficiência, clareza e facilidade de aprendizado, passou a ser usada em diversos ramos da programação, como no desenvolvimento de jogos, controle de robôs, processamento de texto, etc. <http://www.lua.org/>

MIDlet – É um aplicativo em J2ME para dispositivos móveis, desenvolvido com a configuração DLDC/MIDP.

MIDP – *Mobile Information Device Profile*, perfil do J2ME destinado a celulares e PDAs.

PAN – *Personal Area Network* (Rede de Área Pessoa), rede de computadores pessoais, formadas por nós (dispositivos conectados à rede) muito próximos ao usuário (geralmente em metros). Estes dispositivos podem ser pertencentes ao usuário ou não.

PBP – *Personal Basis Profile*, perfil do J2ME destinado a dispositivos que conectados que suportem um nível básico de apresentação gráfica.

PDA – *Personal Digital Assistant* (Assistente Pessoal Digital) é um computador de dimensões reduzidas, dotado de grande capacidade computacional.

pdb – *Palm DataBase*, arquivo de dados para Palm.

PP – *Personal Profile*, perfil do J2ME utilizado em dispositivos que necessitam de um suporte completo para interfaces, como PDAs e consoles para jogos.

prc – *Palm Resource*, arquivo de execução para Palm.

RAD – *Rapid Application Development* (Desenvolvimento Rápido de Aplicação).

Stroke – Componente para escrita manual presente na ferramenta MCards.

SW – SuperWaba, tecnologia para o desenvolvimento de aplicações para PDAs que utiliza a linguagem Java.

SWVM – *SuperWaba Virtual Machine* (Máquina virtual para SuperWaba).

UIML – *User Interface Markup Language* é uma linguagem declarativa baseada em XML, que descreve interfaces para o usuário.

VM – *Virtual Machine* (Maquina Virtual), um computador fictício criado por um programa de simulação.

Wireless – Sem fio ou *Wi-fi* (*Wireless Fidelity*) é o termo usado para receptores de rádios. Um protocolo de comunicação sem fios desenhado com o objetivo de criar redes wireless de alta velocidade e que não faz mais do que transferir dados por ondas de rádio em frequências não licenciadas.

WLAN – *Wireless Local Network* (Rede Local Sem Fio), um sistema que interliga computadores em uma área de alcance bastante restrita, através de tecnologias de rede sem fio.

XML – *Extensible Markup Language* é uma recomendação da W3C para gerar linguagens de marcação para necessidades especiais. É um subtipo de SGML (*Standard Generalized Markup Language - Padrão de Marcação Generalizada*) capaz de descrever diversos tipos de dados. Seu propósito principal é a facilidade de compartilhamento de informações através da Internet.

Apêndice A. ENGENHARIA REVERSA DA FERRAMENTA MCards 1.0

A.1. Introdução

Este documento foi elaborado aplicando conceitos de Engenharia Reversa, com o objetivo de obter um entendimento da ferramenta MCards, bem como realizar a documentação dessa ferramenta a fim de permitir manutenções futuras na mesma, sejam essas manutenções corretivas, evolutivas, etc. Para tanto, baseou-se no método de Engenharia Reversa FUSION-RE/I para recuperação das visões funcionais e visões estruturais da ferramenta, todavia não sendo fiel ao método. A escolha do método FUSION-RE/I se orientou pelo fato de que o mesmo leva à produção das visões do sistema, ou seja, auxilia no entendimento e reconhecimento do seu modelo abstrato, sob o paradigma de Orientação a Objetos (OO).

A.2. Primeira Etapa: Recuperação das Visões Funcionais

Com o objetivo de se obter informações sobre as funcionalidades da ferramenta, realizou-se uma pesquisa sobre as informações já existentes da MCards e o uso intensivo da ferramenta, visando seu entendimento e elaboração da documentação contendo casos de uso, recuperação dos requisitos funcionais e modelo de ciclo de vida.

Relação da documentação existente sobre a ferramenta MCards:

- Dissertação de mestrado que deu origem à ferramenta MCards: “MCards: Ferramenta de apoio à construção de aplicativos para assistentes pessoas digitais” Boccardo, D. R. (2005). MCards: Ferramenta de apoio à construção de aplicativos

para assistentes pessoas digitais. Dissertação (Mestrado) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos.

A.2.1. Descrição da Ferramenta

A.2.1.1. Software e seus Componentes Principais

A ferramenta *MCards* é composta pelos seguintes módulos (Figura A.1):

- Supermódulo *Editor Visual*
 - Módulo de Edição Visual
 - Módulo de Geração *forML*
 - Módulo de Geração *messageML*
- Supermódulo de Geração de Código
 - Módulo JAXB
 - Módulo Tradutor SuperWaba

Figura A.1: Arquitetura da Ferramenta MCards

A.2.1.2. Descrição dos Módulos

- **Supermódulo Editor Visual:** Responsável pela manipulação visual da ferramenta. É através desse módulo que o usuário interage com a ferramenta pra a criação de

uma aplicação para dispositivo móvel. Este módulo é composto pelos módulos de Edição Visual, Geração de *forML* e Geração *messageML*.

- **Módulo de Edição Visual:** Responsável pela interação visual entre o usuário e a ferramenta. Neste módulo é possível construir uma aplicação para dispositivos móveis através da utilização da técnica *drag and drop* (arrastar e soltar) para manipulação de componentes de interface.
- **Módulo de Geração *forML* e Geração *messageML*:** Responsáveis pela geração dos arquivos XML para especificação da aplicação a ser gerada e da mensagem a ser enviada por essa aplicação para um endereço Web.
- **Supermódulo de Geração de Código:** Esse módulo transforma o documento XML de especificação da aplicação a ser gerada, *forML*, em documentos contendo códigos de acordo com os tradutores existentes. Esse módulo é composto pelo módulo JAXB e pelos módulos tradutores para linguagens específicas, nesta versão da ferramenta MCards está disponível apenas o tradutor para a linguagem SuperWaba.
 - **Módulo JAXB:** Responsável pela conversão do arquivo *forML* em objetos Java.
 - **Módulo Tradutor SuperWaba:** Responsável pela tradução dos objetos Java gerados pelo JAXB para a linguagem SuperWaba.

A.2.1.3. Missão da Ferramenta

A ferramenta MCards apóia a construção de aplicações baseadas em *cards* para PDAs com geração de código. A ferramenta dispõe ao desenvolvedor um ambiente visual com um conjunto de elementos de interface básicos (*Label*, *Edit*, *ListBox*, *ComboBox*, *TextArea*), um elemento de navegação (*Link*), e um elemento específico para PDAs (Entrada para *Strokes*). Para a manipulação destes elementos, utiliza-se da técnica *drag and drop* para seleção, criação, e posicionamento dos elementos na interface, objetivando, com isso, tornar o processo mais intuitivo, bem como, ajudar a prevenir erros de programação.

Assim, tem-se o processo de especificação da interface da aplicação a ser gerada abrangendo a manipulação de *cards* e componentes (*widgets*) em conjunto com parâmetros relacionados como a linguagem em que o código será gerado, dispositivo para qual será gerada a aplicação e tamanho da tela. Nesse código tem-se:

- Interface com o usuário
- Controle de persistência no dispositivo;
- Controle de eventos;

A.2.1.4. Termos e Definições

Tabela A.1: Termos e definições utilizados na ferramenta MCards

Termo	Definição
Usuário Desenvolvedor	Usuário que utiliza a ferramenta MCards para desenvolver uma aplicação para dispositivo móvel.
card	Cartão virtual, conceito similar ao formulário. Tela para inserção de componentes de interface.
Componente de Interface	Elemento visual possível num card. Pode ser botão, texto, campo de texto, ou outros elementos comuns de uma interface gráfica.
forML	Documento XML de especificação da aplicação a ser gerada.
messageML	Documento XML de especificação da mensagem que a aplicação a ser gerada poderá enviar para um endereço Web durante sua execução.
SuperWaba	Tecnologia para o desenvolvimento de aplicações para PDAs que utiliza a linguagem Java. Sigla: SW.
Stroke	Componente para escrita manual.
Java	Linguagem de programação orientada a objetos.
JAXB	API para acesso e transformação de documentos XML em objetos Java e transformação de objetos Java em documentos XML.
PDA	<i>Personal Digital Assistant</i> (Assistente Pessoal Digital) é um computador de dimensões reduzidas, dotado de grande capacidade computacional.

A.2.2. Requisitos Funcionais e Casos de Uso

A.2.2.1. Funções da Ferramenta

Tabela A.2: Descrição das Funções da Ferramenta MCards

Função	Descrição
Especificação da Interface	Permite a manipulação de componentes de interface e configuração de parâmetros para a criação de uma aplicação.
Geração de Código	Permite a conversão da especificação definida para a nova aplicação em código para uma determinada linguagem.
Integração com Sistemas Web	Permite a integração da aplicação gerada com sistemas Web.

A.2.2.2. Como Usar a Ferramenta

Para iniciar o uso da ferramenta, é necessário o preenchimento dos dados do usuário como nome do usuário. Para a criação de uma nova aplicação, é necessário seguir os passos listados a seguir sem necessariamente obedecer a uma sequência pré-estabelecida nem obrigatoriedade de sua execução:

- Definir parâmetros para a configuração da aplicação:
- Dispositivo para qual se deseja gerar a aplicação;
- Linguagem utilizada para a geração do código;
- Dimensões da janela virtual, tendo em vista o dispositivo pretendido;
- URL para a o envio de dados durante a execução da aplicação gerada;
- Inserção de componentes na tela virtual:
 - Clicar no componente desejado para que o mesmo apareça na tela virtual;
 - Corrigir posicionamento do componente inserido;
 - Definir propriedades do componente inserido, como nome, cores, textos (para componentes como *Label*, *Link* e *Button*) e eventos, se houver;
- Visualizar documento de especificação da aplicação, *forML*;
- Visualizar documento de especificação da mensagem que pode ser enviada pela aplicação, *messageML*;
- Gerar o código para a aplicação especificada até então e visualizá-lo;

Após a especificação da aplicação, o usuário pode gerar os arquivos necessários para compilar a aplicação, sendo, para isso, necessária a definição do diretório onde se encontra uma versão do SDK do SuperWaba e definição do diretório onde será criada a aplicação, sendo este último com o mesmo nome da aplicação. Em seguida deve-se gerar os arquivos necessários para compilar a aplicação (botão “Create Files”).

Para compilar a aplicação é necessário ir ao diretório no qual foram gerados os arquivos e executar o arquivo “compila.bat”.

A.2.2.3. Interfaces de usuário

Abaixo estão algumas imagens que ilustram o layout da ferramenta MCards.

Figura A.2: Tela inicial

Figura A.3: Tela de criação de aplicação - Editor Visual

Figura A.4: Exemplo de aplicação sendo criada

Figura A.5: Tela de criação de arquivos necessários para compilação - Deploy

Figura A.6: Tela About

Figura A.7: Tela de ajuda - Help

A.2.2.4. Diagrama de Casos de Uso

Figura A.8: Diagrama de Casos de Uso da Ferramenta MCards

A.2.2.5. Casos de Uso

a) Casos de Uso de Especificação da Interface:

Configura *card*:

Nome do Caso de Uso:	Configura <i>card</i>
Ator:	Usuário.
Descrição:	O usuário manipula (adiciona, remove, configura) <i>cards</i> para a criação da interface da aplicação a ser gerada.
Pré-condições:	1.O usuário iniciou a criação de uma nova aplicação.
Pós-condições:	1. Existência de uma especificação para a aplicação construída, documento <i>forML</i> .

Fluxo Básico de Eventos	
Ações do Ator:	Ações do Sistema:
1. O usuário acessa a ferramenta MCards.	2. O sistema oferece na interface uma tela virtual para a criação da interface da

	aplicação a ser.
3. O usuário clica com botão direito na tela virtual, ou seja, no <i>card</i> .	4. O sistema mostra as opções possíveis.
5. O usuário seleciona a opção “Create Screen”	6. O sistema adiciona um novo <i>card</i> no final da pilha.

Fluxo Alternativo de Eventos: Remover

Ações do Ator:	Ações do Sistema:
5. O usuário clica <i>card</i> na tela virtual e seleciona a opção “Remover”.	6. O sistema remove da o <i>card</i> da pilha.

Fluxo Alternativo de Eventos: Configurar Cor de Fundo

Ações do Ator:	Ações do Sistema:
5. O usuário clica <i>card</i> na tela virtual e seleciona a opção “BackgroundColor”	6. O sistema mostra uma janela para seleção da cor.
7. O usuário seleciona uma cor.	8. O sistema atualiza o <i>card</i> com a cor atualizada.

Fluxo Alternativo de Eventos: Configurar Título da Aplicação

Ações do Ator:	Ações do Sistema:
3. O usuário clica no título da aplicação na tela virtual	4. O sistema mostra o cursor para edição do título da aplicação.
5. O usuário digita o novo título.	6. O sistema atualiza a tela virtual com o título atualizado.

Fluxo Alternativo de Eventos: Configurar Menu Help da Aplicação

Ações do Ator:	Ações do Sistema:
3. O usuário clica no título da aplicação na tela virtual	4. O sistema mostra o menu com as opções disponíveis.
5. O usuário seleciona a opção “Help”.	6. O sistema mostra o menu com as opções disponíveis.
7. O usuário seleciona a opção “New” ou “Edit”	8. O sistema mostra janela para preenchimento dos dados.
9. O usuário digita o conteúdo do Help e fecha a janela.	8. O sistema atualiza os dados.

Fluxo Alternativo de Eventos: Configurar Menu About da Aplicação

Ações do Ator:	Ações do Sistema:
3. O usuário clica no título da aplicação na tela virtual	4. O sistema mostra o menu com as opções disponíveis.
5. O usuário seleciona a opção “About”.	6. O sistema mostra o menu com as opções disponíveis.
7. O usuário seleciona a opção “New” ou “Edit”	8. O sistema mostra janela para preenchimento dos dados.
9. O usuário digita o conteúdo do About e fecha a janela.	8. O sistema atualiza os dados.

Outras Informações: (opcional)	1. O usuário pode adicionar quantos <i>cards</i> quiser.
-----------------------------------	--

Manipula Componentes:

Nome do Caso de Uso:	Manipula Componentes
Ator:	Usuário.
Descrição:	O usuário manipula (adiciona, remove, configura) componentes para a criação da interface da aplicação a ser gerada.
Pré-condições:	1.O usuário iniciou a criação de uma nova aplicação.
Pós-condições:	1. Existência de uma especificação para a aplicação construída, documento <i>forML</i> .

Fluxo Básico de Eventos

Ações do Ator:	Ações do Sistema:
1. O usuário acessa a ferramenta MCards.	2. O sistema oferece na interface uma tela virtual para a criação da interface da aplicação a ser gerada e uma barra de ferramentas com a lista de componentes possíveis.
3. O usuário seleciona um componente.	4. O sistema mostra uma instância do componente seleciona na tela virtual.

Fluxo Alternativo de Eventos: Adicionar outro Componente

Ações do Ator:	Ações do Sistema:
3. O usuário seleciona um componente.	4. O sistema mostra uma instância do componente seleciona na tela virtual.

Fluxo Alternativo de Eventos: Remover Componente

Ações do Ator:	Ações do Sistema:
5. O usuário clica na componente adicionado na tela virtual e seleciona a opção “Remover”.	6. O sistema remove da tela virtual o componente selecionado.

Fluxo Alternativo de Eventos: Configurar Cor do Componente	
Ações do Ator:	Ações do Sistema:
5. O usuário clica na componente adicionado na tela virtual e seleciona a opção “BackgroundColor” ou “ForegroundColor”	6. O sistema mostra uma janela para seleção da cor.
7. O usuário seleciona uma cor.	8. O sistema atualiza o componente com a cor atualizada.

Fluxo Alternativo de Eventos: Configurar Nome do Componente	
Ações do Ator:	Ações do Sistema:
5. O usuário clica na componente adicionado na tela virtual e seleciona a opção “Rename”	6. O sistema mostra uma janela para preenchimento do novo nome.
7. O usuário digita o novo nome.	8. O sistema atualiza o componente com o nome atualizado.

Fluxo Alternativo de Eventos: Configurar Texto do Componente	
Ações do Ator:	Ações do Sistema:
5. O usuário clica na componente adicionado na tela virtual e seleciona a opção “Edit Label”	6. O sistema mostra uma janela para preenchimento do texto.
7. O usuário digita o novo texto.	8. O sistema atualiza o componente com o texto atualizado.

Fluxo Alternativo de Eventos: Configurar Evento para o Componente	
Ações do Ator:	Ações do Sistema:
5. O usuário clica na componente adicionado na tela virtual e seleciona a opção “Event” ou “Goto”	6. O sistema mostra uma janela para configuração do evento.
7. O usuário configura o evento.	8. O sistema atualiza o componente com o evento atualizado.

Outras Informações: (opcional)	1. O usuário pode adicionar quantos componentes quiser. 2. O usuário pode fazer qualquer combinação de configuração em um componente.
-----------------------------------	--

Determina Linguagem:

Nome do Caso de Uso:	Determina Linguagem
Ator:	Usuário.
Descrição:	O usuário seleciona a linguagem para a geração do código da aplicação a ser gerada.
Pré-condições:	1.O usuário iniciou a criação de uma nova aplicação.
Pós-condições:	1. Existência de uma especificação para a aplicação construída, documento <i>forML</i> .

Fluxo Básico de Eventos

Ações do Ator:	Ações do Sistema:
1. O usuário acessa a ferramenta MCards.	2. O sistema oferece na interface um menu de opções de configurações.
3. O usuário seleciona o menu “Target Code”.	4. O sistema mostra as opções do menu com .as linguagens possíveis.
5. O usuário seleciona uma linguagem.	6. O sistema atualiza o menu deixando destacada a linguagem selecionada.

Determina Dispositivo:

Nome do Caso de Uso:	Determina Dispositivo
Ator:	Usuário.
Descrição:	O usuário seleciona o dispositivo para o qual a aplicação será gerada.
Pré-condições:	1.O usuário iniciou a criação de uma nova aplicação.
Pós-condições:	1. Existência de uma especificação para a aplicação construída, documento <i>forML</i> .

Fluxo Básico de Eventos

Ações do Ator:	Ações do Sistema:
1. O usuário acessa a ferramenta MCards.	2. O sistema oferece na interface um menu de opções de configurações.
3. O usuário seleciona o menu “Target Device”.	4. O sistema mostra as opções do menu com .os dispositivos disponíveis.
5. O usuário seleciona um dispositivo.	6. O sistema atualiza o menu deixando destacado o dispositivo selecionado.

Determina Tamanho de Tela:

Nome do Caso de Uso:	Determina Tamanho de Tela
Ator:	Usuário.
Descrição:	O usuário seleciona o tamanho da tela para a construção da aplicação a ser gerada.
Pré-condições:	1.O usuário iniciou a criação de uma nova aplicação.
Pós-condições:	1. Existência de uma especificação para a aplicação construída, documento <i>forML</i> .

Fluxo Básico de Eventos

Ações do Ator:	Ações do Sistema:
1. O usuário acessa a ferramenta MCards.	2. O sistema oferece na interface um menu de opções de configurações.
3. O usuário seleciona o menu “Device Size”.	4. O sistema mostra as opções do menu com tamanhos de tela disponíveis.
5. O usuário seleciona um tamanho.	6. O sistema atualiza o menu deixando destacado o tamanho selecionado.

b) Casos de Uso de Geração de Código:

Visualiza *forML*:

Nome do Caso de Uso:	Visualiza <i>forML</i>
Ator:	Usuário.
Descrição:	O usuário visualiza o documento XML de especificação da aplicação a ser gerada.
Pré-condições:	1. O usuário iniciou a criação de uma nova aplicação. 2. Existência de uma especificação para a aplicação construída, documento <i>forML</i> .
Pós-condições:	

Fluxo Básico de Eventos

Ações do Ator:	Ações do Sistema:
1. O usuário acessa a ferramenta MCards.	2. O sistema oferece na interface uma barra de ferramentas com botões para visualização dos documentos gerados para o arquivo <i>forML</i> : “XML” e “Schema”.
3. O usuário clica no botão “XML”.	4. O sistema mostra uma nova janela com conteúdo do arquivo selecionado.

Fluxo Alternativo de Eventos: Visualizar Schema

Ações do Ator:	Ações do Sistema:
3. O usuário clica no botão “Schema”.	4. O sistema mostra uma nova janela com conteúdo do arquivo selecionado.

Define Pasta da Aplicação:

Nome do Caso de Uso:	Define pasta da aplicação
Ator:	Usuário.
Descrição:	O usuário define pasta onde serão gerados os arquivos necessários para a compilação da nova aplicação.
Pré-condições:	1. O usuário criou nova aplicação. 2. Existência de uma especificação para a aplicação construída, documento <i>forML</i> .
Pós-condições:	

Fluxo Básico de Eventos	
Ações do Ator:	Ações do Sistema:
1. O usuário seleciona na aba inferior, da tela de edição da ferramenta, a opção “Deploy”	2. O sistema mostra a tela de deploy da aplicação com o botão “Aplication” para definir a o diretório onde serão salvos os arquivos necessários para a compilação da nova aplicação.
3. O usuário clica no botão “Aplication”.	4. O sistema abre a janela para seleção de diretório.
5. O usuário seleciona o diretório desejado.	

Define Pasta do SuperWaba:

Nome do Caso de Uso:	Define pasta do SuperWaba
Ator:	Usuário.
Descrição:	O usuário define pasta onde está o SDK do Superwaba.
Pré-condições:	1. O usuário criou nova aplicação.
Pós-condições:	

Fluxo Básico de Eventos	
Ações do Ator:	Ações do Sistema:
1. O usuário seleciona na aba inferior, da tela de edição da ferramenta, a opção “Deploy”	2. O sistema mostra a tela de deploy da aplicação com o botão “SuperWabaSDK” para definir a o diretório onde está o JDK

	do SuperWaba.
3. O usuário clica no botão “SuperWabaSDK”.	4. O sistema abre a janela para seleção de diretório.
5. O usuário seleciona o diretório desejado.	

Gera arquivos:

Nome do Caso de Uso:	Gera Arquivos
Ator:	Usuário.
Descrição:	O usuário gera os arquivos necessários para a compilação da nova aplicação.
Pré-condições:	1. O usuário criou nova aplicação. 2. Existência de uma especificação para a aplicação construída, documento <i>forML</i> . 3. O usuário definiu o diretório da aplicação. 4. O usuário definiu o diretório do SDK do SuperWaba
Pós-condições:	1. Existência de um arquivo de código Java (nome_da_aplicacao.java) e um arquivo para compilar e gerar a aplicação SuperWaba (compila.bat).

Fluxo Básico de Eventos	
Ações do Ator:	Ações do Sistema:
1. O usuário seleciona na aba inferior, da tela de edição da ferramenta, a opção “Deploy”	2. O sistema mostra a tela de deploy da aplicação com o botão “Create Files” para definir a o diretório onde serão salvos os arquivos necessários para a compilação da nova aplicação.
3. O usuário clica no botão “Create Files”.	4. O sistema gera os arquivos necessários no diretório da aplicação.

c) Casos de Uso de Integração com Aplicação Web:

Define URL:

Nome do Caso de Uso:	Define URL
Ator:	Usuário.
Descrição:	O usuário define um endereço Web (URL) para o qual a aplicação a ser gerada enviará os dados durante sua execução.
Pré-condições:	1.O usuário iniciou a criação de uma nova aplicação.
Pós-condições:	1. Existência de uma especificação para a aplicação construída, documento <i>forML</i> .

Fluxo Básico de Eventos	
Ações do Ator:	Ações do Sistema:
1. O usuário acessa a ferramenta MCards.	2. O sistema oferece na interface um menu de opções de configurações.
3. O usuário seleciona o menu “URL”.	4. O sistema mostra as opções do menu com campo de texto para inserção da URL.
5. O usuário digita a URL desejada.	

Visualiza *messageML*:

Nome do Caso de Uso:	Visualiza <i>messageML</i>
Ator:	Usuário.
Descrição:	O usuário visualiza o documento XML de especificação para a mensagem a ser enviada durante a execução da aplicação gerada.
Pré-condições:	1. O usuário iniciou a criação de uma nova aplicação. 2. Existência de uma especificação para a mensagem a ser enviada durante a execução da aplicação gerada, documento <i>messageML</i> .
Pós-condições:	

Fluxo Básico de Eventos	
Ações do Ator:	Ações do Sistema:
1. O usuário acessa a ferramenta MCards.	2. O sistema oferece na interface uma barra de ferramentas com botões para visualização dos documentos gerados para o arquivo <i>messageML</i> : “XML” e “Schema”
3. O usuário clica no botão “XML”.	4. O sistema mostra uma nova janela com conteúdo do arquivo selecionado.

Fluxo Alternativo de Eventos: Visualizar Schema	
Ações do Ator:	Ações do Sistema:
3. O usuário clica no botão “Schema”.	4. O sistema mostra uma nova janela com conteúdo do arquivo selecionado.

A.2.3. Modelo de Ciclo de Vida

O Modelo de Ciclo de Vida é composto por expressões regulares que definem a sequência de eventos a que o sistema pode interagir durante todo o período que está em

execução. Ele descreve o comportamento completo de como o sistema se comunica com o ambiente, desde seu início até o seu término. Para facilitar a compreensão do modelo, foi adotada uma convenção para a escrita do ciclo de vida:

- nomes em *itálico*: não são constantes da interface do sistema, ou seja, não aparecem de fato na interface. Esses nomes, ou são ações do usuário (como selecionar elementos de uma lista), ou são recursos de abstração para facilitar a escrita e o entendimento do ciclo de vida;

- nomes precedidos por “b_” aparecem na interface como botões;

- nomes precedidos por “m_” aparecem na interface como opções de menus;

- nomes que aparecem em letras maiúsculas são aqueles que possuem uma sentença que descreve com mais detalhes essa opção;

- nomes que aparecem em letras minúsculas são símbolos terminais, que podem ser uma operação ou uma entrada;

- nomes precedidos por “#” representam os eventos de saída, tais como mensagens.

- x . y : x seguido de y

- x | y : x ou y

- [x] : opcional

- x^{*} : zero ou mais ocorrências de x

- x⁺ : uma ou mais ocorrências de x

- x || y: intercalação arbitrária dos elementos de x e y.

As opções presentes na interface do sistema formam a expressão principal do modelo de ciclo de vida. A partir das opções apresentadas na expressão principal, são construídas novas expressões, uma para cada opção, mostrando as seqüências de operações permitidas a partir daquele ponto.

Tabela A.3: Modelo de Ciclo de Vida da Ferramenta MCards

Modelo de Ciclo de Vida : MCards
Life cycle MCards: (usuario . b_Ok . BEGIN)
BEGIN = ((EDITOR HELP DEPLOY ABOUT)*)
ABOUT = ((BEGIN EDITOR HELP DEPLOY)*)
HELP = (documentacao (BEGIN EDITOR DEPLOY ABOUT)*)

```

DEPLOY = ((b_Aplicacion . CHOSEFOLDER . b_SuperWabaSDK . CHOSEFOLDER . b_CreateFiles .
(#mensagem_erro_cricao_arquivo)) | (BEGIN | EDITOR | HELP | ABOUT)*)

CHOSEFOLDER = (folder. (b_Ok . (#mensagem_erro_pasta_invalida) | b_Cancel))

EDITOR = (((m_TARGET_DEVICE | m_TARGET_CODE | m_DEVICE_SIZE | m_URL)* | | (b_Label .
COMPONET_LABEL | b_Text . COMPONET_TEXT | b_Edit . COMPONET_EDIT | b_ComboBox .
COMPONET_COMBOBOX | b_List . COMPONET_LIST | b_Strokes . COMPONET_STROKES |
b_Link . COMPONET_LINK | b_Button . COMPONET_BUTTON) | | VIRTUAL-SCREEN | |
(b_forML_Schema . FORML-SCHEMA | b_forML_XML . FORML-XML | b_messageML_Schema .
MESSAGEML-SCHEMA | b_messageML_XML . MESSAGEML-XML) | (b_Generate .
[b_SuperWaba . CODESUPERWABA])) | (BEGIN | HELP | DEPLOY | ABOUT)*)

TARGET_DEVICE = (Palm | Pocket PC)

TARGET_CODE = (SuperWaba | J2me)

DEVICE_SIZE = (size_160x160 | size_240x320 | size_320x320 | size_320x480 | OTHERSIZE)

OTHERSIZE = ((Width . (#mensagem_erro_invalido) . (b_Ok | b_Cancel)) . (Height .
(#mensagem_erro_invalido) . (b_Ok | b_Cancel)))

URL = (url)

VIRTUAL-SCREEN = ((BACKGROUNDCOLOR | Create_Screen | Configure_URL)* | |
(COMPONET_LABEL | COMPONET_TEXT | COMPONET_EDIT | COMPONET_COMBOBOX |
COMPONET_LIST | COMPONET_STROKES | COMPONET_LINK | COMPONET_BUTTON)* )

COMPONET_LABEL = ((RENAME | EDIT_TEXT | FOREGROUNDCOLOR | Remove)*)

COMPONET_TEXT = ((RENAME | FOREGROUNDCOLOR | BACKGROUNDCOLOR | Remove)*)

COMPONET_EDIT = ((RENAME | FOREGROUNDCOLOR | BACKGROUNDCOLOR | Remove)*)

COMPONET_COMBOBOX = (ADD-ITENS . (RENAME | BACKGROUNDCOLOR |
FOREGROUNDCOLOR | Remove | EVENT)*)

COMPONET_LIST = (ADD-ITENS . (RENAME | BACKGROUNDCOLOR | FOREGROUNDCOLOR |
Remove | EVENT)*)

COMPONET_STROKES = ([Remove])

COMPONET_LINK = ((RENAME | EDIT_TEXT | FOREGROUNDCOLOR | Remove | GOTO)*)

COMPONET_BUTTON = ((RENAME | BACKGROUNDCOLOR | FOREGROUNDCOLOR |
EDIT_TEXT | Remove | EVENT)*)

RENAME = (name . (b_Ok | b_Cancel))

EDIT_TEXT = (Text . (b_Ok | b_Cancel))

FOREGROUNDCOLOR = (COLOR)

BACKGROUNDCOLOR = (COLOR)

COLOR = (color . (b_Ok | b_Cancel | b_Reset))

EVENT = ((null | GOTO | send). (b_Ok | b_Cancel))

GOTO = (chose_screen . (b_Ok | b_Cancel))

ADD-ITENS = (items . (b_Ok | b_Cancel))

```

FORML-SCHEMA = (SHOW-FILE)
FORML-XML = (SHOW-FILE)
MESSAGEML-SCHEMA = (SHOW-FILE)
MESSAGEML-XML = (SHOW-FILE)
CODESUPERWABA = (SHOW-FILE)
SHOW-FILE = (file)

A.3.Segunda Etapa: Recuperação das Visões Estruturais

Nessa etapa procurou-se conhecer a implementação da ferramenta, entendendo a estrutura de pacotes e identificando-se os métodos e classes de implementação em cada arquivo de código. Para tanto, foram elaborados a relação de pacotes e arquivos (Tabela A.4) e o índice de procedimentos (Tabela A.5) da ferramenta MCards.

Tabela A.4: Relação de pacotes e arquivos do MCards

Pacote	Arquivo
MCards: Pacote geral da ferramenta.	MCards.java
MCards / Componentes: Classes responsáveis pelos componentes de interface e de dados que podem ser inseridos na construção de uma aplicação.	Componentes.java MyButton.java MyComboBox.java MyEdit.java MyLabel.java MyLink.java MyList.java MyStrokes.java MyTextArea.java
MCards / Generate: Possui classes responsáveis pela geração de código.	generateCode.java LanguageAbstract.java ProcessLanguageSuperwaba.java
MCards / HandlerMessage: classe para ser usada no lado servidor quando estiver sendo comunicado dados do dispositivo	AbstractMessage.java
MCards / JAXBXML: classes geradas pelo JAXB previamente a partir de xml schema	Arquivos correspondentes ao XML Schema (forML.xsd).
MCards / Main: Classes responsáveis pelo editor visual e o módulo de geração de código.	Deploy.java Editor.java HelpMain.java SendParameters.java
MCards / Menu: Possui classes que gerenciam o menu do aplicativo a ser gerado.	About.java Help.java

MCards / Parameters: classes que controlam parâmetros de tela, tamanho, URL, etc.	HandlePanelDesign.java Parameters.java
MCards / util:	Wrapper.java
MCards / xml: Possui classes que controlam a manipulação e criação de documentos xml.	CreateforML.java CreatemessageML.java XMLSerializer.java

O índice de procedimentos da Tabela A.5 refere-se a todos os procedimentos da ferramenta MCards, identificando a localização no código-fonte de cada procedimento em termos do diretório e do arquivo a que pertencem. Para melhor entendimento foram adotadas algumas representações como:

- Método: nomeMetodo e arquivo Arquivo.java: significa que o método nomeMetodo pertence à classe Arquivo;
- Método: Classe.nomeMetodo e arquivo Arquivo.java: significa que o método nomeMetodo pertence à classe Classe, sendo que esta pertence à classe Arquivo;
- O construtor de cada classe não está na lista de métodos da classe;

Tabela A.5: Índice de Procedimentos da MCards

Método	Diretório / Arquivo
addBegin	MCards / MCards.java
addDesign	MCards / MCards.java
addHelp	MCards / MCards.java
addDeploy	MCards / MCards.java
addComm	MCards / MCards.java
addAbout	MCards / MCards.java
setName	MCards / Componentes / Componentes.java
getName	MCards / Componentes / Componentes.java
getBackgroundColor	MCards / Componentes / Componentes.java
getForegroundColor	MCards / Componentes / Componentes.java
getPosition	MCards / Componentes / Componentes.java
toXML	MCards / Componentes / Componentes.java
setName	MCards / Componentes / MyButton.java
getName	MCards / Componentes / MyButton.java
setEvent	MCards / Componentes / MyButton.java
getEvent	MCards / Componentes / MyButton.java
setOption	MCards / Componentes / MyButton.java
getOption	MCards / Componentes / MyButton.java
getPosition	MCards / Componentes / MyButton.java
getBackgroundColor	MCards / Componentes / MyButton.java
getForegroundColor	MCards / Componentes / MyButton.java

toXML	MCards / Componentes / MyButton.java
setTargetItem	MCards / Componentes / MyComboBox.java
getTargetItem	MCards / Componentes / MyComboBox.java
getBackgroundColor	MCards / Componentes / MyComboBox.java
getPosition	MCards / Componentes / MyComboBox.java
getItems	MCards / Componentes / MyComboBox.java
setName	MCards / Componentes / MyComboBox.java
getName	MCards / Componentes / MyComboBox.java
setEvent	MCards / Componentes / MyComboBox.java
getEvent	MCards / Componentes / MyComboBox.java
setGoto	MCards / Componentes / MyComboBox.java
getGoto	MCards / Componentes / MyComboBox.java
getForegroundColor	MCards / Componentes / MyComboBox.java
toXML	MCards / Componentes / MyComboBox.java
setName	MCards / Componentes / MyEdit.java
getName	MCards / Componentes / MyEdit.java
setLines	MCards / Componentes / MyEdit.java
getLines	MCards / Componentes / MyEdit.java
getSizeComponent	MCards / Componentes / MyEdit.java
getPosition	MCards / Componentes / MyEdit.java
getBackgroundColor	MCards / Componentes / MyEdit.java
getForegroundColor	MCards / Componentes / MyEdit.java
toXML	MCards / Componentes / MyEdit.java
getPosition	MCards / Componentes / MyLabel.java
getForegroundColor	MCards / Componentes / MyLabel.java
setName	MCards / Componentes / MyLabel.java
getName	MCards / Componentes / MyLabel.java
getBackgroundColor	MCards / Componentes / MyLabel.java
toXML	MCards / Componentes / MyLabel.java
getPosition	MCards / Componentes / MyLink.java
getForegroundColor	MCards / Componentes / MyLink.java
setName	MCards / Componentes / MyLink.java
getName	MCards / Componentes / MyLink.java
setEvent	MCards / Componentes / MyLink.java
getEvent	MCards / Componentes / MyLink.java
setGoto	MCards / Componentes / MyLink.java
getGoto	MCards / Componentes / MyLink.java
toXML	MCards / Componentes / MyLink.java
setTargetItem	MCards / Componentes / MyList.java
getTargetItem	MCards / Componentes / MyList.java
getBackgroundColor	MCards / Componentes / MyList.java
getForegroundColor	MCards / Componentes / MyList.java
getPosition	MCards / Componentes / MyList.java
getSizeComponent	MCards / Componentes / MyList.java
getName	MCards / Componentes / MyList.java
setEvent	MCards / Componentes / MyList.java
getEvent	MCards / Componentes / MyList.java

setGoto	MCards / Componentes / MyList.java
getGoto	MCards / Componentes / MyList.java
getItfens	MCards / Componentes / MyList.java
toXML	MCards / Componentes / MyList.java
getBackgroundColor	MCards / Componentes / MyStrokes.java
getForegroundColor	MCards / Componentes / MyStrokes.java
getPosition	MCards / Componentes / MyStrokes.java
getSizeComponent	MCards / Componentes / MyStrokes.java
toXML	MCards / Componentes / MyTextArea.java
setName	MCards / Componentes / MyTextArea.java
getName	MCards / Componentes / MyTextArea.java
setLines	MCards / Componentes / MyTextArea.java
getLines	MCards / Componentes / MyTextArea.java
getPosition	MCards / Componentes / MyTextArea.java
getBackgroundColor	MCards / Componentes / MyTextArea.java
getForegroundColor	MCards / Componentes / MyTextArea.java
toXML	MCards / Componentes / MyTextArea.java
getXmlRoot	MCards / Generate / generateCode.java
getCodeSuperwaba	MCards / Generate / generateCode.java
addLine	MCards / Generate / LanguageAbstract.java
processContainers	MCards / Generate / LanguageAbstract.java
processComponents	MCards / Generate / LanguageAbstract.java
getTableItfens	MCards / Generate / LanguageAbstract.java
getParameters	MCards / Generate / LanguageAbstract.java
getVectorContainer	MCards / Generate / LanguageAbstract.java
getComents	MCards / Generate / LanguageAbstract.java
getGenerateVariables	MCards / Generate / LanguageAbstract.java
getVectorEventVariables	MCards / Generate / LanguageAbstract.java
getForm	MCards / Generate / LanguageAbstract.java
getComponentsofForm	MCards / Generate / LanguageAbstract.java
getItensofBox	MCards / Generate / LanguageAbstract.java
getSizeStroke	MCards / Generate / LanguageAbstract.java
getNumberOf	MCards / Generate / LanguageAbstract.java
setStrokesize	MCards / Generate / LanguageAbstract.java
getStrokesize	MCards / Generate / LanguageAbstract.java
getNumberOfData	MCards / Generate / LanguageAbstract.java
hasNavigator	MCards / Generate / LanguageAbstract.java
isStore	MCards / Generate / LanguageAbstract.java
getUser	MCards / Generate / LanguageAbstract.java
getTitle	MCards / Generate / LanguageAbstract.java
getMenu	MCards / Generate / LanguageAbstract.java
getUrl	MCards / Generate / LanguageAbstract.java
getSyntaxList	MCards / Generate / LanguageAbstract.java
setSyntaxList	MCards / Generate / LanguageAbstract.java
getSyntaxCombo	MCards / Generate / LanguageAbstract.java
setSyntaxCombo	MCards / Generate / LanguageAbstract.java
getSyntaxLabel	MCards / Generate / LanguageAbstract.java

setSyntaxLabel	MCards / Generate / LanguageAbstract.java
getSyntaxButton	MCards / Generate / LanguageAbstract.java
setSyntaxButton	MCards / Generate / LanguageAbstract.java
getCode	MCards / Generate / ProcessLanguageSuperwaba.java
getImports	MCards / Generate / ProcessLanguageSuperwaba.java
getMain	MCards / Generate / ProcessLanguageSuperwaba.java
getConstants	MCards / Generate / ProcessLanguageSuperwaba.java
getVariables	MCards / Generate / ProcessLanguageSuperwaba.java
getConstructor	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodonStart	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodonExit	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodonPaint	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodchangeScreen	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodCreateContainers	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodgetControl	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodCreateNavigator	MCards / Generate / ProcessLanguageSuperwaba.java
getComponents	MCards / Generate / ProcessLanguageSuperwaba.java
addMyLabel	MCards / Generate / ProcessLanguageSuperwaba.java
addMyLink	MCards / Generate / ProcessLanguageSuperwaba.java
addMyEdit	MCards / Generate / ProcessLanguageSuperwaba.java
addMyStrokes	MCards / Generate / ProcessLanguageSuperwaba.java
addMyComboBox	MCards / Generate / ProcessLanguageSuperwaba.java
addMyTextArea	MCards / Generate / ProcessLanguageSuperwaba.java
addMyList	MCards / Generate / ProcessLanguageSuperwaba.java
addButton	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodOnEvents	MCards / Generate / ProcessLanguageSuperwaba.java
getEventMenu	MCards / Generate / ProcessLanguageSuperwaba.java
getItemHelp	MCards / Generate / ProcessLanguageSuperwaba.java
getItemAbout	MCards / Generate / ProcessLanguageSuperwaba.java
getItemSend	MCards / Generate / ProcessLanguageSuperwaba.java
getItemCommunication	MCards / Generate / ProcessLanguageSuperwaba.java
getEventPrior	MCards / Generate / ProcessLanguageSuperwaba.java
getEventNext	MCards / Generate / ProcessLanguageSuperwaba.java
getEventSave	MCards / Generate / ProcessLanguageSuperwaba.java
getEventDelete	MCards / Generate / ProcessLanguageSuperwaba.java
getEventStrokes	MCards / Generate / ProcessLanguageSuperwaba.java
getEventListCombo	MCards / Generate / ProcessLanguageSuperwaba.java
getEventLinks	MCards / Generate / ProcessLanguageSuperwaba.java
getEventButton	MCards / Generate / ProcessLanguageSuperwaba.java
getClassStrokes	MCards / Generate / ProcessLanguageSuperwaba.java
getClassSerialConnection	MCards / Generate / ProcessLanguageSuperwaba.java
getVariablesClassSerialConnection	MCards / Generate / ProcessLanguageSuperwaba.java
getConstructorClassSerialConnection	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodClassSerialConnectionwrite	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodClassSerialConnectionread	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodClassSerialConnectionclose	MCards / Generate / ProcessLanguageSuperwaba.java
getClassHttpConnection	MCards / Generate / ProcessLanguageSuperwaba.java

getVariablesClassHttpConnection	MCards / Generate / ProcessLanguageSuperwaba.java
getConstructorClassHttpConnection	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodClassHttpConnectionopen	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodClassHttpConnectiongetProtocol	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodClassHttpConnectiongetUri	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodClassHttpConnectiongetHost	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodClassHttpConnectionaddHeader	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodClassHttpConnectionpost	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodClassConnectionaddParameter	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodClassHttpConnectionclose	MCards / Generate / ProcessLanguageSuperwaba.java
getClassHandlerStrokes	MCards / Generate / ProcessLanguageSuperwaba.java
getVariablesClassHandlerStrokes	MCards / Generate / ProcessLanguageSuperwaba.java
getConstructorClassHandlerStrokes	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodClassHandlerStrokescreateStrokes	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodClassHandlerStrokesclearStrokes	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodClassHandlerStrokesinicializaBuffer	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodClassHandlerStrokesgetPontos	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodClassHandlerStrokesaddPoints	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodClassHandlerStrokesdrawPoints	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodClassHandlerStrokesgetPoints	MCards / Generate / ProcessLanguageSuperwaba.java
getClassHandlerRecord	MCards / Generate / ProcessLanguageSuperwaba.java
getVariablesClassHandlerRecord	MCards / Generate / ProcessLanguageSuperwaba.java
getConstructorClassHandlerRecord	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodClassHandlerRecordclearScreen	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodClassHandlerRecordsaveRecords	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodClassHandlerRecorddeleteRecord	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodClassHandlerRecordloadRecord	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodSendMessage	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodshowMessage	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodCreateMessage	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodFillMessageRecord	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodSaveData	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodLoadData	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodConstructor	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodAdd	MCards / Generate / ProcessLanguageSuperwaba.java
getMethodBounds	MCards / Generate / ProcessLanguageSuperwaba.java
processRequest	MCards / HandlerMessage / AbstractMessage.java
doGet	MCards / HandlerMessage / AbstractMessage.java
doPost	MCards / HandlerMessage / AbstractMessage.java
getServletInfo	MCards / HandlerMessage / AbstractMessage.java
generateCode	MCards / HandlerMessage / AbstractMessage.java
getXmlRoot	MCards / HandlerMessage / AbstractMessage.java
Gerado automaticamente pelo JAXB	MCards / JAXBXML
getPath	MCards / Main / Deploy.java
createFile	MCards / Main / Deploy.java
createScript	MCards / Main / Deploy.java
addTopPanel	MCards / Main / Deploy.java

addCenterPanel	MCards / Main / Deploy.java
addBottomPanel	MCards / Main / Deploy.java
initializeHandlers	MCards / Main / Editor.java
addWidgetsRightMenu	MCards / Main / Editor.java
addWidgetsLeftMenu	MCards / Main / Editor.java
addMenuMain	MCards / Main / Editor.java
addBottomBar	MCards / Main / Editor.java
addTitle	MCards / Main / Editor.java
getComponentTitle	MCards / Main / Editor.java
getNativeInterface	MCards / Main / Editor.java
addName	MCards / Main / Editor.java
Configure	MCards / Main / Editor.java
setGoto	MCards / Main / Editor.java
addEvents	MCards / Main / Editor.java
addItens	MCards / Main / Editor.java
updateNavigator	MCards / Main / Editor.java
addPanelButton	MCards / Main / Editor.java
ButtonClick . actionPerformed	MCards / Main / Editor.java
ButtonPanelHandler . actionPerformed	MCards / Main / Editor.java
MouseHandler . mouseClicked	MCards / Main / Editor.java
MouseHandler . mouseExited	MCards / Main / Editor.java
MouseHandlerMotion . mouseDragged	MCards / Main / Editor.java
MouseHandlerMotion . mouseMoved	MCards / Main / Editor.java
RadioHandler . itemStateChanged	MCards / Main / Editor.java
createNodes	MCards / Main / HelpMain.java
getTexto	MCards / Main / HelpMain.java
getItem	MCards / Main / HelpMain.java
getConnection	MCards / Main / SendParameters.java
sendObject	MCards / Main / SendParameters.java
receiveObject	MCards / Main / SendParameters.java
*** Possui apenas construtor ***	MCards / Menu / About.java
*** Possui apenas construtor ***	MCards / Menu / Help.java
setBackColor	MCards / Parameters / HandlePanelDesign.java
getBackColor	MCards / Parameters / HandlePanelDesign.java
setHasNavigator	MCards / Parameters / HandlePanelDesign.java
hasNavigator	MCards / Parameters / HandlePanelDesign.java
getAttributeNavigator	MCards / Parameters / HandlePanelDesign.java
getCount	MCards / Parameters / HandlePanelDesign.java
getComponentes	MCards / Parameters / HandlePanelDesign.java
getCountComp	MCards / Parameters / HandlePanelDesign.java
setHeight	MCards / Parameters / Parameters.java
getHeight	MCards / Parameters / Parameters.java
setStore	MCards / Parameters / Parameters.java
getStore	MCards / Parameters / Parameters.java
getUser	MCards / Parameters / Parameters.java
setUser	MCards / Parameters / Parameters.java
setCode	MCards / Parameters / Parameters.java

getCode	MCards / Parameters / Parameters.java
setWidth	MCards / Parameters / Parameters.java
getWidth	MCards / Parameters / Parameters.java
setPlataforma	MCards / Parameters / Parameters.java
getPlataforma	MCards / Parameters / Parameters.java
setCodigoPDA	MCards / Parameters / Parameters.java
getCodigoPDA	MCards / Parameters / Parameters.java
getURL	MCards / Parameters / Parameters.java
getComponentUrl	MCards / Parameters / Parameters.java
setComponentTitle	MCards / Parameters / Parameters.java
getTitle	MCards / Parameters / Parameters.java
getNameApp	MCards / Parameters / Parameters.java
inputSize	MCards / Parameters / Parameters.java
readFile	MCards / util / Wrapper.java
getforML	MCards / xml / CreateforML.java
readSchema	MCards / xml / CreateforML.java
getMenu	MCards / xml / CreateforML.java
getParameters	MCards / xml / CreateforML.java
getComponents	MCards / xml / CreateforML.java
getDocument	MCards / xml / CreateforML.java
getMessageML	MCards / xml / CreateMessageML.java
readSchema	MCards / xml / CreateMessageML.java
getContainers	MCards / xml / CreateMessageML.java
getDocument	MCards / xml / CreateMessageML.java
setLineSeparator	MCards / xml / XMLSerializer.java
serialize	MCards / xml / XMLSerializer.java
serializeNode	MCards / xml / XMLSerializer.java

Apêndice B. MANUAL DA FERRAMENTA

MCARDS 2.0

B.1. Introdução

Este manual tem o objetivo de facilitar o uso da ferramenta desde sua instalação, utilização e instalação da aplicação gerada no dispositivo. Assim, o mesmo está dividido em quatro partes, sendo essas:

- Instalação da Ferramenta MCards
- Utilização da Ferramenta MCards
- Instalação da Máquina Virtual
- Instalação da Aplicação Gerada

B.2. Instalação da Ferramenta MCards

Descompactar a arquivo zip MCards.zip e está pronto para usar!

Estrutura do pacote(Figura B.1):

- MCards: diretório que contém o arquivo .jar da aplicação MCards, além das API's necessárias para sua execução como Java 1.3 (*j2sdk1.3.1*), Java 1.4 (*j2sdk1.4.2_07*), SuperWaba 5.61 (*SuperWabaSDK_5.61*) e J2ME (*j2me*).
- Schema: contém os schemas utilizados para gerar o arquivo xml de projeto e da mensagem a ser enviada pela aplicação gerada.
- Workspace: diretório de trabalho da aplicação MCards. Neste diretório ficam todos os projetos criados dentro da aplicação.

- Templates: diretório com templates para projetos MCards. Neste diretório ficam todos os templates que podem ser usados para criação de um novo projeto na ferramenta MCards.
- MCards.jar: arquivo executável Java para a aplicação MCards.
- MCards.bat: arquivo para a execução do executável Java para a aplicação MCards.

Mcards		Pasta de arquivos
schema		Pasta de arquivos
template		Pasta de arquivos
workspace		Pasta de arquivos
MCards	1 KB	Arquivo em lotes do MS-DOS
MCards	893 KB	Executable Jar File

Figura B.1: Estrutura do diretório da ferramenta

B.3. Utilização da Ferramenta MCards

Para iniciar a ferramenta MCards basta executar o arquivo MCards.bat presente na raiz do diretório da ferramenta.

As sessões seguintes ilustram como criar uma aplicação para dispositivo móvel através da ferramenta MCards, como ditar posteriormente uma aplicação já criada, como gerar seu respectivo código e os arquivos de instalação, entre outras funções da ferramenta.

B.3.1 Criando uma nova aplicação

Ao iniciar a ferramenta MCards, o usuário deve escolher entre criar um novo projeto, abrir um projeto já existente ou abrir um template. Para criar uma nova aplicação, deve-se escolher a opção “New Project” (Figura B.2).

Figura B.2: Tela para escolha de novo projeto

Ao escolher a opção “New Project”, a tela apresentada é a tela do Editor Visual (Figura B.3) que possui (a) um menu superior para configuração de parâmetros para construção da aplicação, (b) um conjunto de componentes que podem ser usados para

construção de uma aplicação, (c) uma tela para construção da aplicação, (d) um conjunto de operações que podem ser executadas e (e) um conjunto de botões para visualização dos arquivos gerados para a aplicação. Entraremos em detalhes sobre cada um desses itens durante o decorrer desse manual.

Figura B.3: Tela inicial da ferramenta

Para a configuração dos parâmetros da aplicação, deve-se acessar o menu superior presente na tela do Editor (Figura B.3 a) e escolher a linguagem a com a qual será gerado o código, o tamanho da tela do dispositivo de destino e a URL para envio dos dados enquanto a aplicação estiver sendo executada.

A tecnologia usada para geração do código e conseqüentemente a geração da aplicação pode ser SuperWaba ou J2ME, como mostra a Figura B.4:

Figura B.4: Menu superior: escolha de tecnologia

As opções de tamanho de tela são mostradas na Figura B.5. Além desses tamanhos previamente definidos, pode-se também definir outros tamanhos através da opção “Other”.

Figura B.5: Menu superior: escolha do tamanho da tela

No menu URL pode ser preenchida uma URL para a qual, durante a execução da aplicação que está sendo criada, a mesma poderá enviar seus dados (Figura B.6). Esse parâmetro é opcional.

Figura B.6: Menu superior: definição da URL

Para criação da interface da nova aplicação, basta clicar no componente desejado da barra de componentes e, ao clicar na tela, uma instância do mesmo será adicionada na mesma, na posição desejada. A Figura B.7 mostra (a) a barra de componentes com o componente “Edit” selecionado, e (b) componente “Edit” sendo inserido na tela. Essa sequência de passos pode ser executada para inserção de qualquer componente na tela, em qualquer quantidade. Apenas o componente “Stroke” possui a restrição de apenas um por aplicação.

Figura B.7: Adicionando um componente

Para a configuração das propriedades de um componente inserido, basta clicar com o botão direito do mouse para ter acesso ao menu de configurações do componente. Este menu varia de componente para componente, pois algumas propriedades como cor de fundo e/ou eventos podem estar presentes ou não num componente. Segue abaixo a tabela com a relação de cada componente e suas possíveis opções de menu:

Tabela B.1: Componentes x opções de menu de configuração

Opção de Menu	Label	Text	Edit	ComboBox	List	Strokes	Link	Button
Rename	*	*	*	*	*		*	*
Edit Text	*						*	*
BackColor		*	*	*	*			*
ForeColor	*	*	*	*	*		*	*
Event				*	*			*
Goto							*	
Remove	*	*	*	*	*	*	*	*

A Figura B.8 mostra o componente “*Edit*” inserido na tela e, ao clicar como botão direito, o seu menu de opções:

Figura B.8: Menu de configurações do componente "Edit"

Pode-se também adicionar novos *cards*, ou seja, novas telas para a aplicação que está sendo criada. Além disso, é possível criar ligações entre essas telas, por exemplo através de links ou eventos em componentes, como, por exemplo, o componente botão.

Para criar uma nova tela, basta clicar com botão direito do mouse na tela e escolher a opção “Create Screen” no menu de opções, como mostra a Figura B.9. Além da opção de adicionar novas telas à aplicação, pode-se também definir cores de fundo (“Background Color”), remover a barra de navegação para armazenamento de registros durante a execução da aplicação (“Remove Navigator”). OBS: A opção “Remove Navigator” só estará disponível quando foi inserido algum componente de dados na tela, sendo esses componentes de armazenamento: *Edit*, *ComboBox*, *List* e *Stroke*.

Figura B.9: Menu de configuração de *cards*

A Figura B.10 mostra a tela de uma aplicação criada na ferramenta MCards:

A imagem mostra a interface de uma aplicação web intitulada "Cadastro Cliente". O formulário contém campos para "Nome:", "Sexo:" (com uma lista suspensa mostrando "[Masculino...]", "Endereco:", "Telefone:" e "e-mail:". Abaixo dos campos, há um botão "Enviar". Na base da interface, há uma barra com quatro botões: "save", "<-", "->" e "del".

Figura B.10: Tela de uma aplicação criada no MCards

Para salvar a aplicação criada, deve-se clicar no botão “Save Project” (Figura B.11), sendo que o projeto será salvo no diretório “workspace/Nome_da_Aplicação”).

Figura B.11: Salvar aplicação

B.3.2 Gerando código e compilando a aplicação

Após criar/editar e salvar a aplicação, pode-se gerar o código para a mesma de acordo com a tecnologia escolhida (SuperWaba ou J2ME) clicando no botão “Generate Code” (Figura B.12).

Figura B.12: Gerar código para aplicação

O código gerado pode ser visualizado, através do botão “Code” (Figura B.13 a). Além disso, também é possível, a qualquer momento do processo de construção da aplicação, definir se os dados serão armazenados na aplicação através da opção “Store” (Figura B.13 b) e a visualização dos arquivos *forML* que define a especificação abstrata da ferramenta (Figura B.13 c), *messageML* que exemplifica o xml que será enviado pela aplicação durante sua execução para a URL configurada (Figura B.13 d).

Figura B.13: Barra de Interação

B.3.3 Editando uma aplicação criada anteriormente

Para editar uma aplicação criada anteriormente basta selecionar a opção “Open a project” quando iniciar a ferramenta (Figura B.14), e em seguida selecionar o projeto que deseja editar.

Figura B.14: Tela para escolha de editar um projeto

Para editar as configurações e/ou componentes que compõem a aplicação, deve-se seguir os mesmos passos mostrados na Seção B.3.1 “Criando uma nova aplicação”.

Para salvar as alterações feitas na aplicação, deve-se seguir o mesmo procedimento descrito na Seção B.3.1, ou seja, clicando no botão “Save Project” (Figura B.11), sendo que o projeto será salvo no diretório “workspace/Nome_da_Aplicação”.

B.3.4 Criando uma nova aplicação a partir de um template

Para criar uma aplicação a partir de um template basta selecionar a opção “Open a template” quando iniciar a ferramenta (Figura B.15), e em seguida selecionar o template que seja usar para criar uma nova aplicação.

Figura B.15: Tela para escolha de abrir um template

Para editar as configurações e/ou componentes que compõem a aplicação, deve seguir os mesmos passos mostrados na Seção B.3.1 “Criando uma nova aplicação”.

Para salvar as alterações feitas na aplicação deve-se seguir o mesmo procedimento descrito na Seção B.3.1, ou seja, clicando no botão “Save Project” (Figura B.11), sendo que o projeto será salvo no diretório “workspace/Nome_da_Aplicação”.

B.4. Instalação da Máquina Virtual

Para que a aplicação gerada possa ser executada no dispositivo móvel é necessária, inicialmente, a instalação da máquina virtual (*Virtual Machine*) específica para a plataforma e para a tecnologia em questão.

Por exemplo, se o dispositivo for um Palm, com o Palm OS versão 5.0 e a tecnologia usada para a geração do código for SuperWaba, a máquina virtual a ser instalada neste dispositivo deverá ser a SuperWaba Virtual Machine (SWVM) para Palm OS 5.0.

Se for um Pocket Pc é necessário saber a plataforma e o processador que seu dispositivo usa e seleccionar a opção correta. As possíveis plataformas são:

- HPC2000 (Hand PC 2000);
- HPC211 (Hand PC v. 2.11);
- PPC211 (Palm-sized PC); e
- Pocket PC (AKA Windows CE 3.0).

Os possíveis processadores são:

- MIPS;
- SH3; e
- ARM.

B.4.1 SuperWaba

A máquina virtual SuperWaba (SWVM) está disponível online em <http://www.SuperWaba.org/install> e também no pacote da ferramenta MCards no diretório “*/MCards/SuperWabaSDK_5.61/VirtualMachines/*”.

A parte da máquina virtual SuperWaba que necessita ser instalada no dispositivo Palm OS consiste em três arquivos:

- SuperWaba.pdb: contem as classes disponíveis em “*SuperWabaSDK_5.61/src/waba/**” e “*SuperWabaSDK_5.61/src/java/**”;
- SuperWaba.prc: contem o intérprete de bytecode Java;

- SWNatives.prc: contem a implementação para os métodos nativos (ex: Graphics.drawLine);

Para a instalação da maquina virtual no Pocket PC é necessário único arquivo de instalação “superwaba.PLATAFORMA_PROCESSADOR.cab” .

No Linux consiste no arquivo:

- superwaba_XXX.deb: para o Debian que necessita ser renomeado para deb;
- superwaba-XXX.rpm: para RedHat e Fedora.

Para Symbian consiste em um arquivo superwaba_XXX.cab.

B.4.2 J2ME

Existem algumas maquinas virtuais Java para dispositivos móveis disponíveis no mercado como NSIcom CreEme 3.25, Jeode Runtime 1.9, PersonalJava, IBM J9 entre outras, das quais utilizaremos a J9 JVM (*Java Virtual Machine*) da IBM (<http://www.ibm.com/us/>).

A J9 JVM está disponível online para várias plataformas em http://www-306.ibm.com/software/wireless/wctme_fam/ e no pacote da ferramenta MCards no diretório “/MCards/j2me/VirtualMachines/” apenas para Windows Mobile 2003 for Pocket PC. Para outros dispositivos é necessário fazer o download pelo site.

Para a instalação é necessário que o dispositivo esteja conectado ao computador de mesa, então basta executar a instalação da J9 JVM e seguir os passos para instalar também no dispositivo. Além disso, é necessária a criação do diretório no dispositivo móvel onde serão instalados os executáveis Java (.jar), por exemplo “/J2ME”.

B.5. Instalação da Aplicação Gerada

B.5.1 SuperWaba

Os arquivos gerados para uma aplicação SuperWaba consistem:

- “MinhaAplicação”.prc: arquivo executável para Palm;
- “MinhaAplicação”.pdb: arquivo de dados para Palm;

- “MinhaAplicação”.PLATAFORMA_PROCESSADOR.cab: arquivo de instalação para o Pocket PC para cada tipo de dispositivo Pocket PC (plataforma x processador);
- “MinhaAplicação”.sh: arquivo para Linux;
- epoc: diretório com arquivo para Symbian OS (“MinhaAplicação”.sis)
- s60: diretório com arquivo para Symbian OS para celulares Nokia serie 60 (“MinhaAplicação”.sis)

Para a utilização, basta instalar os arquivos necessários para o Palm OS (.prc e .pdb), Pocket PC (.exe), Linux (.sh) ou Symbian OS (.sis) e executar a aplicação.

B.5.2 J2ME

Os arquivos gerados para uma aplicação J2ME consistem em um arquivo executável Java (.jar) e um arquivo de link para execução do arquivo JAR (.lnk).

Para a instalação é necessário copiar o arquivo .jar no diretório “/J2ME” e copiar o arquivo de atalho (.lnk) no diretório de sua preferência, ex: “Meus Documentos”. Para executar basta clicar no arquivo de atalho.

Apêndice C. EXEMPLO DE CÓDIGO

J2ME GERADO

C.1. Aplicação Criada

A ferramenta MCards apóia o desenvolvimento de aplicações para dispositivos móveis, especialmente para PDAs, com geração de aplicações baseadas em *cards*. Essa seção ilustra uma aplicação desenvolvida com o uso da ferramenta MCards, versão 2.0, explorando suas funcionalidades para a construção de interface gráfica e armazenamento de dados, sendo que a aplicação deverá ser gerada para a tecnologia J2ME.

A aplicação criada trata-se de um cadastro de clientes de uma empresa. Os funcionários dessa empresa querem cadastrar seus clientes de forma rápida, preenchendo apenas dados básicos como nome, endereço, telefone e se o cliente deseja receber informações sobre os produtos da empresa. Para tal, a empresa possui alguns PDAs nos quais será instalada a aplicação a ser criada.

A Figura C.1 ilustra a criação das telas da aplicação de cadastro de clientes através do uso da ferramenta MCards.

Foram criadas duas telas para a aplicação de cadastro de clientes (Figura C.2): (a) a primeira tela possui os campos para preenchimento do nome, endereço, telefone e e-mail que utilizam o componente *Edit*, além do campo para a escolha do sexo com a utilização do componente *ComboBox*, e o campo para recolhimento da assinatura do cliente que utiliza o componente *Stroke*; (b) a segunda tela é composta pelos dois componentes *ComboBox* e *List* para o preenchimento de duas perguntas relativas ao recebimento de informações sobre produtos da empresa pelo usuário. Além dos componentes para preenchimento de dados, as duas telas possuem também componentes para os rótulos de texto relativos aos componentes de dados: os *Labels*.

Figura C.1: Criação das telas da aplicação pra cadastro de clientes

Figura C.2: Telas da aplicação pra cadastro de clientes

Os mecanismos de gerenciamento e armazenamento dos dados são transparentes ao desenvolvedor que está criando a aplicação, pois são gerados pela ferramenta MCards. O

armazenamento e o acesso aos registros armazenados são possíveis através da barra de navegação presente nas telas que possuem campos de dados (Figura C.2 c).

A Figura C.3 mostra o arquivo *forML* de especificação gerado para a aplicação que está sendo criada. Este arquivo possui a especificação abstrata de todas as telas definidas para a aplicação e dos componentes inseridos em cada tela com seus respectivos parâmetros. Possui, também, parâmetros necessários para a geração da aplicação como o tamanho da tela e a tecnologia para a qual o código e a aplicação serão gerados. Entre os parâmetros definidos está a URL para a qual a aplicação poderá enviar os dados, neste caso essa URL é o endereço Web do servidor da empresa para onde os dados serão enviados e serão armazenados no banco de dados da empresa. A Figura C.4 ilustra uma mensagem enviada pela aplicação em execução contendo apenas um registro, ou seja, o cadastro de um cliente. Não há restrição do número de registros que podem ser enviados em uma mensagem.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<forML>
<menu about="Cadastro de Clientes - Versão 1.0" help="Para salvar e navegar entre registros utilize a barra de navegação." />
<parameters height="320" width="240" code="J2ME" title="CadastroCliente" url="http://www.empresaX.com.br" store="true" />
<components>
<form id="0" backgroundColor="201,218,231" navigatorbar="true">
<myLabel name="lbl_Nome" foregroundcolor="0,51,102" position="8,6" Text="Nome:" dimension="35,30" />
<myEdit name="edt_Nome" backgroundColor="255,255,255" foregroundcolor="0,0,0" lines="1" position="65,6" size="169,20"
dimension="169,20" />
<myLabel name="lbl_Sexo" foregroundcolor="0,51,102" position="8,36" Text="Sexo:" dimension="35,30" />
<myComboBox backgroundColor="210,210,210" foregroundcolor="0,0,0" position="65,36"
itens="Masculino~NULL#Feminino~NULL#" name="cmb_Sexo" events="GOTO" dimension="82,20" />
<myLabel name="lbl_End" foregroundcolor="0,51,102" position="8,66" Text="Endereco:" dimension="67,30" />
<myEdit name="edt_end" backgroundColor="255,255,255" foregroundcolor="0,0,0" lines="1" position="65,66" size="163,20"
dimension="163,20" />
<myLabel name="lbl_Fone" foregroundcolor="0,51,102" position="8,96" Text="Telefone:" dimension="63,30" />
<myEdit name="edt_Fone" backgroundColor="255,255,255" foregroundcolor="0,0,0" lines="1" position="65,96" size="79,20"
dimension="79,20" />
<myLabel name="lbl_email" foregroundcolor="0,51,102" position="8,126" Text="e-mail:" dimension="49,30" />
<myEdit name="edt_email" backgroundColor="255,255,255" foregroundcolor="0,0,0" lines="1" position="65,126" size="79,20"
dimension="79,20" />
<myLabel name="Label_Ass" foregroundcolor="0,51,102" position="8,156" Text="Assinatura:" dimension="77,30" />
<myStrokes position="65,156" color="null" option="1" dimension="162,58" />
<mybutton name="btn_next" backgroundColor="236,233,216" foregroundcolor="0,51,102" Text="Próxima Tela"
position="146,244" dimension="86,25" event="GOTO" option="0" />
</form>
<form id="1" backgroundColor="214,228,209" navigatorbar="true">
<myLabel name="lbl_info" foregroundcolor="0,94,47" position="8,6" Text="Deseja receber informações sobre nossos produtos?"
dimension="217,60" />
<myComboBox backgroundColor="210,210,210" foregroundcolor="0,0,0" position="16,56" itens="Sim~NULL#Não~NULL#"
name="cmb_info" events="GOTO" dimension="50,20" />
<myLabel name="lbl_meio" foregroundcolor="0,94,47" position="8,90" Text="Se desejar, escolha um meio para receber essas
informações:" dimension="280,60" />
<myList name="lst_meio" backgroundColor="255,255,255" foregroundcolor="0,0,0" position="16,140" events="GOTO"
dimension="92,54" itens="Carta~NULL# e-mail~NULL#Telemarketing~NULL#" />
<mybutton name="btn_back" backgroundColor="236,233,216" foregroundcolor="0,94,47" Text="Voltar" position="161,244"
dimension="72,25" event="GOTO" option="0" />
</form>
</components>
</forML>
```

Figura C.3: Arquivo *forML* gerado para a aplicação de cadastro de clientes

```

<?xml version="1.0" encoding="ISO-8859-1" ?>
<MessageML isStoreinPda="true" application="CadastroCliente">
  <Data>
    <Container id="0">
      <Record id="1">
        <ink>
          <trace>0,4,85,5,86,582,64,58,25,36,23,65,68,69,32,45,88,-1</trace>
          <trace>80,25,26,99,0,5,-1</trace>
        </ink>
        <edit name="edt_Nome">Fulano de Tal</edit>
        <combobox name="cmb_Sexo">Masculino</combobox>
        <edit name="edt_end">Rua das Avenidas, 88</edit>
        <edit name="edt_Fone">3366-6633</edit>
        <edit name="edt_email">fulano@email.com</edit>
      </Record>
    </Container>
    <Container id="1">
      <Record id="1">
        <combobox name="cmb_info">Sim</combobox>
        <list name="lst_meio">e-mail</list>
      </Record>
    </Container>
  </Data>
</MessageML>

```

Figura C.4: Mensagem a ser enviada para URL durante a execução da aplicação de cadastro de clientes

C.2. Estrutura do Código J2ME Gerado

O código J2ME está dividido em 5 partes, sendo essas:

- Cabeçalho e Criação da Aplicação;
- Geração de Interface;
- Gerenciamento de Eventos;
- Gerenciamento e Envio de *messageML*;
- Armazenamento e Gerenciamento de Dados; e
- Gerenciamento de Pontos e Componente Stroke.

C.2.1. Cabeçalho e Criação da Aplicação

Trecho do código composto pelo cabeçalho com um resumo dos componentes que formam a aplicação, a importação das bibliotecas necessárias, a instanciação da aplicação com definição de telas, menus e outros componentes básicos para a formação da aplicação.

```

//Code generated automatically for communication between PDA
//and Web Application;
//The following parameters had been defined for the creation
//of the interface
//Store          = true
//Code           = J2ME
//Windows        = 2
//Labels         = 10
//Edits          = 4
//InputStroke    = 1
//Button         = 2
//ComboBox       = 2
//ListBox        = 1

```

```

//TextArea          = 0
//Links             = 0
//*****

import java.awt.*;
import java.awt.event.*;
import java.util.Vector;
import java.io.*;
import java.net.*;

public class CadastroCliente extends Frame {
    private static final long serialVersionUID = 1L;

    static final String CREATOR_ID = "1788";

    static final String NAME = "Cada";

    static final String CATALOG_NAME = NAME + "." + CREATOR_ID;

    static Vector panels = new Vector(6);

    private boolean laidOut = false;

    Strokes stroke;

    HandlerStrokes handlerstrokes = new HandlerStrokes();

    HandlerRecord handlerrecord = new HandlerRecord();

    MenuBar menubar;

    Choice cmb_Sexo;

    Button btn_next;

    List lst_meio;

    Button btn_back;

    Choice cmb_info;

    public CadastroCliente() {
        super();
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                dispose();
            }

            public void windowClosed(WindowEvent e) {
                saveData();
                System.exit(0);
            }
        });
        setLayout(null);
        setTitle("CadastroCliente");
        Menu menu = new Menu("Menu");
        menu.add(new MenuItem("Help"));
        menu.add(new MenuItem("About"));
        menu.add(new MenuItem("Send"));
        menu.add(new MenuItem("Exit"));
        Menu communication = new Menu("Communication");
        communication.add(new CheckboxMenuItem("Wireless", true));
        menubar = new MenuBar();
        menubar.add(menu);
        menubar.add(communication);
        setMenuBar(menubar);
    }

    public static void main(String argv[]) {
        CadastroCliente app = new CadastroCliente();
        app.setFont(new Font("Arial", Font.PLAIN, 11));
        app.setSize(240, 320);
        app.show();
    }
}

```

C.2.2. Geração de Interface

Trecho do código responsável pela construção da interface de usuário, entre eles: os cards, seus componentes e barra de navegação de registros.

```

/**
 * To define the user interface
 */
public void paint(Graphics g) {
    Vector names, data;
    if (!laidOut) {
        laidOut = true;
        Insets insets = insets();
        data = new Vector();
        names = new Vector();

        // Creates the card 0 and its components
        Panel mainPanel0 = new Panel();
        add(mainPanel0);
        mainPanel0.setBounds(insets.left, insets.top, 240, 320);
        mainPanel0.setVisible(false);
        mainPanel0.setBackground(new Color(219, 231, 240));

        Label lbl_Fone = new Label("Telefone:");
        lbl_Fone.setForeground(new Color(0, 51, 102));
        lbl_Fone.setBounds(4, 96, 60, 30);
        mainPanel0.add(lbl_Fone);

        Label lbl_Sexo = new Label("Sexo:");
        lbl_Sexo.setForeground(new Color(0, 51, 102));
        lbl_Sexo.setBounds(4, 33, 35, 30);
        mainPanel0.add(lbl_Sexo);

        Label lbl_End = new Label("Endereco:");
        lbl_End.setForeground(new Color(0, 51, 102));
        lbl_End.setBounds(4, 66, 60, 30);
        mainPanel0.add(lbl_End);

        TextField edt_Fone = new TextField(1);
        edt_Fone.setBackground(new Color(255, 255, 255));
        edt_Fone.setForeground(new Color(0, 0, 0));
        mainPanel0.add(edt_Fone);
        edt_Fone.setBounds(70, 96, 79, 20);
        names.add(new String("edt_Fone"));

        TextField edt_end = new TextField(1);
        edt_end.setBackground(new Color(255, 255, 255));
        edt_end.setForeground(new Color(0, 0, 0));
        mainPanel0.add(edt_end);
        edt_end.setBounds(70, 66, 150, 20);
        names.add(new String("edt_end"));

        stroke = new Strokes();
        mainPanel0.add(stroke);
        stroke.setBounds(70, 156, 150, 58);
        names.add(new String("stroke"));

        Label Label_Ass = new Label("Assinatura:");
        Label_Ass.setForeground(new Color(0, 51, 102));
        Label_Ass.setBounds(4, 156, 60, 30);
        mainPanel0.add(Label_Ass);

        cmb_Sexo = new Choice();
        cmb_Sexo.add("Masculino");
        cmb_Sexo.add("Feminino");
        mainPanel0.add(cmb_Sexo);
        cmb_Sexo.setBackground(new Color(210, 210, 210));
        cmb_Sexo.setForeground(new Color(0, 0, 0));
        cmb_Sexo.setBounds(70, 36, 82, 20);
        names.add(new String("cmb_Sexo"));

        btn_next = new Button("Próxima Tela");
        mainPanel0.add(btn_next);
        btn_next.setName("btn_next");
        btn_next.setBackground(new Color(236, 233, 216));
        btn_next.setForeground(new Color(0, 51, 102));
        btn_next.setBounds(150, 220, 86, 25);

        TextField edt_Nome = new TextField(1);
        edt_Nome.setBackground(new Color(255, 255, 255));
        edt_Nome.setForeground(new Color(0, 0, 0));
        mainPanel0.add(edt_Nome);
        edt_Nome.setBounds(70, 6, 150, 20);
        names.add(new String("edt_Nome"));

        TextField edt_email = new TextField(1);
        edt_email.setBackground(new Color(255, 255, 255));
        edt_email.setForeground(new Color(0, 0, 0));
        mainPanel0.add(edt_email);
        edt_email.setBounds(70, 126, 79, 20);
        names.add(new String("edt_email"));

        Label lbl_email = new Label("e-mail:");

```



```

lbl_email.setForeground(new Color(0, 51, 102));
lbl_email.setBounds(4, 126, 49, 30);
mainPanel0.add(lbl_email);

Label lbl_Nome = new Label("Nome:");
lbl_Nome.setForeground(new Color(0, 51, 102));
lbl_Nome.setBounds(4, 6, 35, 30);
mainPanel0.add(lbl_Nome);

mainPanel0.add(createNavigator(insets.top));
panels.addElement(mainPanel0);
panels.addElement(names);
panels.addElement(data);
data = new Vector();
names = new Vector();

// Creates the card 1 and its components
Panel mainPanel1 = new Panel();
add(mainPanel1);
mainPanel1.setBounds(insets.left, insets.top, 240, 320);
mainPanel1.setVisible(false);
mainPanel1.setBackground(new Color(225, 247, 225));

lst_meio = new List(3, true);
lst_meio.add("Carta");
lst_meio.add(" e-mail");
lst_meio.add("Telemarketing");
mainPanel1.add(lst_meio);
lst_meio.setBackground(new Color(255, 255, 255));
lst_meio.setForeground(new Color(0, 0, 0));
lst_meio.setBounds(16, 140, 92, 54);
names.add(new String("lst_meio"));

Label lbl_meio2 = new Label("essas informações:");
lbl_meio2.setForeground(new Color(0, 94, 47));
lbl_meio2.setBounds(8, 110, 126, 30);
mainPanel1.add(lbl_meio2);

Label lbl_info2 = new Label("nossos produtos?");
lbl_info2.setForeground(new Color(0, 94, 47));
lbl_info2.setBounds(8, 26, 112, 30);
mainPanel1.add(lbl_info2);

btn_back = new Button("Voltar");
mainPanel1.add(btn_back);
btn_back.setName("btn_back");
btn_back.setBackground(new Color(236, 233, 216));
btn_back.setForeground(new Color(0, 94, 47));
btn_back.setBounds(150, 220, 72, 25);

Label lbl_meio1 = new Label(
    "Se desejar, escolha um meio para receber");
lbl_meio1.setForeground(new Color(0, 94, 47));
lbl_meio1.setBounds(8, 90, 280, 30);
mainPanel1.add(lbl_meio1);

Label lbl_info1 = new Label("Deseja receber informações sobre");
lbl_info1.setForeground(new Color(0, 94, 47));
lbl_info1.setBounds(8, 6, 217, 30);
mainPanel1.add(lbl_info1);

cmb_info = new Choice();
cmb_info.add("Sim");
cmb_info.add("Não");
mainPanel1.add(cmb_info);
cmb_info.setBackground(new Color(210, 210, 210));
cmb_info.setForeground(new Color(0, 0, 0));
cmb_info.setBounds(16, 56, 50, 20);
names.add(new String("cmb_info"));

mainPanel1.add(createNavigator(insets.top));
panels.addElement(mainPanel1);
panels.addElement(names);
panels.addElement(data);
mainPanel0.setVisible(true);

// Loads the stored data, in case that it has
loadData();
int k = 0;
while (k < 6) {
    Component[] components = ((Panel) panels.elementAt(k))
        .getComponents();
    for (int i = 0; i < components.length; i++)
        if (components[i] instanceof Panel) {
            Component[] navComponents = ((Panel) components[i])
                .getComponents();
            for (int j = 0; j < navComponents.length; j++)
                if (navComponents[j] instanceof Label) {

```

```

        if (handlerrecord.currentSlide == ((Vector) panels
            .elementAt(k + 2)).size())
            ((Label) navComponents[j])
                .setText("-/"
                    + ((Vector) panels
                        .elementAt(k + 2))
                        .size());
        else
            ((Label) navComponents[j])
                .setText((handlerrecord.currentSlide + 1)
                    + "/"
                    + ((Vector) panels
                        .elementAt(k + 2))
                        .size());

        .elementAt(k + 2))
        .size());
    }
    k = k + 3;
}
}
((CheckboxMenuItem) (menubar.getMenu(1)).getItem(0)).setState(true);

// Set the first register
Component[] components = ((Panel) panels
    .elementAt(handlerrecord.currentScreen)).getComponents();
for (int i = 0; i < components.length; i++)
    if (components[i] instanceof Panel) {
        Component[] navComponents = ((Panel) components[i])
            .getComponents();
        for (int j = 0; j < navComponents.length; j++)
            if (navComponents[j] instanceof Label) {
                if (handlerrecord.currentSlide == ((Vector) panels
                    .elementAt(handlerrecord.currentScreen + 2))
                    .size())
                    ((Label) navComponents[j])
                        .setText("-/"
                            + ((Vector) panels
                                .elementAt(handlerrecord.currentScreen + 2))
                                .size());
                else
                    ((Label) navComponents[j])
                        .setText((handlerrecord.currentSlide + 1)
                            + "/"
                            + ((Vector) panels
                                .elementAt(handlerrecord.currentScreen + 2))
                                .size());
            }
    }
}

/**
 * Create the navigator bar to navigate into registers
 *
 * @param top
 *       the panel
 * @return
 */
public Panel createNavigator(int top) {
    Panel panel_nav = new Panel();
    panel_nav.setBounds((int) LEFT_ALIGNMENT, top + 320 - 72, 240, 25);
    panel_nav.setBackground(new Color(245, 245, 245));

    Button save = new Button("Save");
    save.setName("navcontrolbuttonsave");
    panel_nav.add(save);
    save.setBounds((240 / 2) - 32 - 42, 0, 40, 20);

    Button prior = new Button("<");
    prior.setName("navcontrolbuttonprior");
    panel_nav.add(prior);
    prior.setBounds((240 / 2) - 32, 0, 30, 20);

    Button next = new Button(">");
    next.setName("navcontrolbuttonnext");
    panel_nav.add(next);
    next.setBounds(240 / 2, 0, 30, 20);

    Button delete = new Button("Del");
    delete.setName("navcontrolbuttondelete");
    panel_nav.add(delete);
    delete.setBounds((240 / 2) + 32, 0, 40, 20);

    Label status = new Label("");
    status.setName("navcontrollabelstatus");
    panel_nav.add(status);
    status.setBounds((240 / 2) + 32 + 42, 0, 35, 20);

```

```

    }
    return panel_nav;
}

```

C.2.3. Gerenciamento de Eventos

Trecho do código responsável pelo gerenciamento de eventos como troca de cards, clique de botões, menus, links e outros componenstes com eventos, além de eventos para desenho no componente de escrita manual.

```

/**
 * Controls aplicacion events
 */
public boolean handleEvent(Event event) {
    if (event.id == Event.WINDOW_DESTROY) {
        dispose();
    } else if (event.id == Event.ACTION_EVENT) {
        if (event.target instanceof MenuItem) {
            String target = new String(((MenuItem) event.target).getLabel());
            if (target.equals("Help"))
                showMessage("Help", "");

            if (target.equals("About"))
                showMessage("About", "");

            if (target.equals("Send"))
                SendMessage();

            if (target.equals("Exit")) {
                saveData();
                dispose();
                System.exit(0);
            }

            if (target.equals("Wireless")) {
                ((CheckboxMenuItem) menubar.getMenu(1)).getItem(0)
                    .setState(true);
            }
        }

        if (event.target instanceof Button) {
            String target = new String(((Button) event.target).getName());
            if (target.equals("navcontrolbuttonprior")) {
                if (handlerrecord.currentSlide > 0) {
                    handlerrecord.currentSlide--;
                    handlerrecord.clearScreen();
                    stroke.clear();
                    handlerrecord.loadRecord(handlerrecord.currentScreen,
                        handlerrecord.currentSlide);
                } else
                    showMessage("Mensagem", " Primeiro Registro ");
            }

            if (target.equals("navcontrolbuttonnext")) {
                if (handlerrecord.currentSlide < ((Vector) panels
                    .elementAt(handlerrecord.currentScreen + 2)).size()) {
                    handlerrecord.currentSlide++;
                    handlerrecord.clearScreen();
                    stroke.clear();
                    if (handlerrecord.currentSlide != ((Vector) panels
                        .elementAt(handlerrecord.currentScreen + 2))
                        .size())
                        handlerrecord.loadRecord(
                            handlerrecord.currentScreen,
                            handlerrecord.currentSlide);
                    repaint();
                } else
                    showMessage("Mensagem", "Última posição");
            }

            if (target.equals("navcontrolbuttonsave")) {
                if (handlerrecord.currentSlide == ((Vector) panels
                    .elementAt(handlerrecord.currentScreen + 2)).size()) {
                    handlerrecord.saveRecord(handlerrecord.currentScreen,
                        handlerrecord.currentSlide, 0);
                    handlerrecord.clearScreen();
                    stroke.clear();
                }
            }
        }
    }
}

```

```

    } else {
        handlerrecord.saveRecord(handlerrecord.currentScreen,
                                handlerrecord.currentSlide, 1);
        if (handlerrecord.currentSlide + 1 != ((Vector) panels
            .elementAt(handlerrecord.currentScreen + 2))
            .size())
            handlerrecord.loadRecord(
                handlerrecord.currentScreen,
                handlerrecord.currentSlide + 1);
        else {
            handlerrecord.clearScreen();
            stroke.clear();
        }
    }
    handlerrecord.currentSlide++;
    repaint();
}

if (target.equals("navcontrolbuttondelete")) {
    if (((Vector) panels
        .elementAt(handlerrecord.currentScreen + 2)).size() > 0) {
        if (handlerrecord.currentSlide != ((Vector) panels
            .elementAt(handlerrecord.currentScreen + 2))
            .size()) {
            handlerrecord.deleteRecord(
                handlerrecord.currentScreen,
                handlerrecord.currentSlide);
            handlerrecord.clearScreen();
            stroke.clear();
            if (handlerrecord.currentSlide != ((Vector) panels
                .elementAt(handlerrecord.currentScreen + 2))
                .size())
                handlerrecord.loadRecord(
                    handlerrecord.currentScreen,
                    handlerrecord.currentSlide);
        } else
            showMessage("Mensagem",
                "Não há registros nessa posição");
    } else
        showMessage("Mensagem", "Não há Registros");
    repaint();
}

if (target.equals("btn_next")) {
    changeScreen(1);
}
if (target.equals("btn_back")) {
    changeScreen(0);
}
}

if (event.target instanceof List || event.target instanceof Choice) {
    if (event.target == cmb_Sexo) {
    }
    if (event.target == lst_meio) {
    }
    if (event.target == cmb_info) {
    }
}

}

if (event.id == Event.MOUSE_DRAG && event.target == stroke) {
    handlerstrokes.x = event.x - (70 + 4);
    handlerstrokes.y = event.y - (156 + 1);
    handlerstrokes.addPoints(0, handlerstrokes.x, handlerstrokes.y);
    stroke.drawLine(handlerstrokes.oldx, handlerstrokes.oldy,
        handlerstrokes.x, handlerstrokes.y);
    handlerstrokes.oldx = handlerstrokes.x;
    handlerstrokes.oldy = handlerstrokes.y;
    repaint();
} else if (event.id == Event.MOUSE_UP && event.target == stroke) {
    ;
    handlerstrokes.addPoints(1, 0, 0);
} else if (event.id == Event.MOUSE_DOWN && event.target == stroke) {
    handlerstrokes.oldx = event.x - (70 + 2);
    handlerstrokes.oldy = event.y - (156 + 1);
    handlerstrokes.addPoints(0, handlerstrokes.oldx,
        handlerstrokes.oldy);
}

return super.handleEvent(event);
}

/**
 * Load a card with a card id
 *
 * @param id
 *         the card id

```

```

*/
public void changeScreen(int id) {
    int i = 0;
    while (i < 6) {
        ((Panel) panels.elementAt(i)).setVisible(false);
        i = i + 3;
    }
    ((Panel) panels.elementAt(id * 3)).setVisible(true);
    handlerrecord.currentSlide = 0;
    handlerrecord.currentScreen = id * 3;
    if (((Vector) panels.elementAt(handlerrecord.currentScreen + 2)).size() > 0)
        handlerrecord.loadRecord(handlerrecord.currentScreen, 0);
}

/**
 * Show a message box for user
 *
 * @param title
 *         the title os window
 * @param message
 *         the shown message
 */
public void showMessage(String title, String message) {
    MsgBox mb = new MsgBox(this, title, message, false);
    requestFocus();
    mb.dispose();
}

/**
 * class MsgBox
 *
 * the class that generates the message box
 */
public static class MsgBox extends Dialog implements ActionListener {
    boolean id = false;

    Button ok, can;

    MsgBox(Frame frame, String title, String msg, boolean okcan) {
        super(frame, "Message", true);
        setLayout(new BorderLayout());
        setTitle(title);
        add("Center", new Label(msg));
        addOKCancelPanel(okcan);
        createFrame();
        pack();
        setVisible(true);
    }

    void addOKCancelPanel(boolean okcan) {
        Panel p = new Panel();
        p.setLayout(new FlowLayout());
        createOKButton(p);
        if (okcan == true)
            createCancelButton(p);
        add("South", p);
    }

    void createOKButton(Panel p) {
        p.add(ok = new Button("OK"));
        ok.addActionListener(this);
    }

    void createCancelButton(Panel p) {
        p.add(can = new Button("Cancel"));
        can.addActionListener(this);
    }

    void createFrame() {
        Dimension d = getToolkit().getScreenSize();
        setLocation(d.width / 3, d.height / 3);
    }

    public void actionPerformed(ActionEvent ae) {
        if (ae.getSource() == ok) {
            id = true;
            setVisible(false);
        } else if (ae.getSource() == can) {
            setVisible(false);
        }
    }
}

```

C.2.4. Gerenciamento e Envio de *messageML*

Trecho do código responsável pela criação da mensagem a ser enviada (*messageML*) à URL definida na aplicação, bem como a criação de conexões necessárias e envio da mesma.

```
/**
 * Generate the MessageML to send
 *
 * @return the xml
 */
public String getMessageXml() {
    StringBuffer message = new StringBuffer();
    message.append("<?xml version='1.0' encoding='UTF-8'>");
    message
        .append("<MessageML application='CadastroCliente' isStoreinPda='true'>");
    message.append("<data>");
    message.append(fillMessageXmlRecord());
    message.append("</data>");
    message.append("</MessageML>");
    return message.toString();
}

/**
 * Fill the MessageML with the datas
 *
 * @return the part of the xml
 */
public String fillMessageXmlRecord() {
    StringBuffer message = new StringBuffer();
    for (int i = 0; i < panels.size(); i = i + 3) {
        message.append("<container id='");
        message.append(new Integer(i).toString());
        message.append(">\n");
        Vector panel = (Vector) panels.elementAt(i + 2);
        Vector tags = (Vector) panels.elementAt(i + 1);
        for (int j = 0; j < panel.size(); j++) {
            message.append("<record number='");
            message.append(String.valueOf(j));
            message.append(">\n");
            Vector slide = (Vector) panel.elementAt(j);
            for (int item = 0; item < slide.size(); item++) {
                Object o = slide.elementAt(item);
                if (o instanceof String) {
                    int type = new Integer(((String) o).substring(0, 1))
                        .intValue();
                    switch (type) {
                        case 0:
                            message.append("<edit name = \""
                                + (String) tags.elementAt(item) + "\">\n");
                            message.append("<value>");
                            message.append(((String) o).substring(1));
                            message.append("</value>\n");
                            message.append("</edit>\n");
                            break;
                        case 1:
                            message.append("<list name = \""
                                + (String) tags.elementAt(item) + "\">\n");
                            message.append("<value>");
                            message.append(((String) o).substring(2));
                            message.append("</value>\n");
                            message.append("</list>\n");
                            break;
                        case 2:
                            message.append("<combobox name = \""
                                + (String) tags.elementAt(item) + "\">\n");
                            message.append("<value>");
                            message.append(((String) o).substring(2));
                            message.append("</value>\n");
                            message.append("</combobox>\n");
                            break;
                    }
                }
            }
            if (o instanceof Vector) {
                message.append("<ink>\n");
                for (int h = 0; h < ((Vector) o).size(); h++) {
                    message.append("<trace>");
                    while (((Integer) ((Vector) o).elementAt(h))
                        .intValue() != -1) {
                        message.append(((Integer) ((Vector) o)
                            .elementAt(h)));
                        message.append(" ");
                        h++;
                    }
                }
            }
        }
    }
}
```

```

        message.append("</trace>\n");
    }
    message.append("</ink>\n");
    }
    message.append("</record>\n");
    }
    message.append("</container>\n");
    }
    return message.toString();
}

/**
 * Sent the MessageML to a URL
 *
 */
public void SendMessage() {
    if (((CheckboxMenuItem) (menubar.getMenu(1)).getItem(0)).getState()) {
        try {
            HttpURLConnection con = new HttpURLConnection(
                "http://143.107.231.196/", 80);
            con.addHeader("Host", con.getHost());
            con.addHeader("User-Agent", "PDA");
            con.addParameter("message", getMessageXml());
            if (con.isOpen()) {
                con.post();
                con.close();
                showMessage("Mensagem", "Dados Enviados");
            }
        } catch (Exception e) {
            showMessage("Mensagem", "" + e);
        }
    }
}

/**
 * class HttpURLConnection
 *
 * Creates the HTTP Connection to send the MessageML
 *
 */
public class HttpURLConnection {
    private Socket sock = null;

    private URL url = null;

    private String uri = null;

    private int port = 80;

    private PrintWriter ds = null;

    private Vector headers = new Vector(20);

    private Vector parameters = new Vector(20);

    public HttpURLConnection(String url, int port) throws Exception {
        int pos;
        this.port = port;
        this.url = new URL(url);
        pos = url.indexOf("http://");
        pos = url.indexOf("/", pos + 7);
        if (pos == -1) {
            throw new Exception("MalFormedURL");
        }
        uri = url.substring(pos + 1);
    }

    public boolean isOpen() {
        boolean result = false;
        try {
            sock = new Socket(url.getHost(), port);
            result = true;
        } catch (UnknownHostException e) {
        } catch (IOException e) {
        }

        if (result)
            try {
                ds = new PrintWriter(new BufferedWriter(
                    new OutputStreamWriter(sock.getOutputStream()),
                    true);

                result = true;
            } catch (IOException e) {
                result = false;
            }
    }

    if (!result)

```

```

        showMessage("Mensagem", "Não foi possível conectar.");

        return result;
    }

    public boolean close() {
        ds.close();
        try {
            sock.close();
        } catch (IOException e) {
        }
        sock = null;
        ds = null;
        return true;
    }

    public void post() {
        String postdata = "";

        Vector xkeys = new Vector();
        for (int i = 0; i < parameters.size(); i = i + 2)
            xkeys.add(parameters.get(i));

        int sz = xkeys.size();
        if (sz > 0) {
            for (int x = 0; x < sz - 1; x++) {
                String key = (String) xkeys.get(x);
                postdata = postdata + key + "="
                    + parameters.get((x * 2) + 1) + "&";
            }
            postdata = postdata + xkeys.get(sz - 1) + "="
                + parameters.get(((sz - 1) * 2) + 1);
        }

        String header = "POST /" + uri + " HTTP/1.0\n"
            + "Content-type: application/x-www-form-urlencoded\n"
            + "Content-length: " + postdata.length() + "\n";

        xkeys = new Vector();
        for (int i = 0; i < headers.size(); i = i + 2)
            xkeys.add(headers.get(i));

        sz = xkeys.size();
        if (sz > 0) {
            for (int x = 0; x < sz - 1; x++) {
                String key = (String) xkeys.get(x);
                header = header + key + ": " + headers.get((x * 2) + 1)
                    + "\n";
            }
            header = header + xkeys.get(sz - 1) + ": "
                + headers.get(((sz - 1) * 2) + 1) + "\n\n";
        } else {
            header = header + "\n";
        }
        ds.println(header);
        ds.println(postdata);
    }

    public void addHeader(String name, String value) {
        headers.add(name);
        headers.add(value);
    }

    public String getProtocol() {
        return url.getProtocol();
    }

    public String getUri() {
        return uri;
    }

    public String getHost() {
        return url.getHost();
    }

    public void addParameter(String name, String value) {
        parameters.add(name);
        parameters.add(value);
    }
}

```


C.2.5. Armazenamento e Gerenciamento de Dados

Trecho do código responsável pelo armazenamento dos registros de cada card, bem como o acesso aos registros armazenados.

```
/**
 * Save the instance of a card in device
 *
 */
public void saveData() {
    FileOutputStream fos = null;

    try {
        fos = new FileOutputStream(CATALOG_NAME + ".PROPERTIES");

        DataOutputStream dos = new DataOutputStream(fos);

        for (int i = 0; i < panels.size(); i = i + 3) {

            Vector panel = (Vector) panels.elementAt(i + 2);
            Vector tags = (Vector) panels.elementAt(i + 1);

            dos.writeChar('\n');
            dos.writeInt(panel.size());

            for (int j = 0; j < panel.size(); j++) {
                Vector slide = (Vector) panel.elementAt(j);
                dos.writeInt(slide.size());
                for (int item = 0; item < slide.size(); item++) {
                    Object obj = slide.elementAt(item);
                    if (obj instanceof String) {
                        dos.writeUTF("<" + (String) tags.elementAt(item)
                            + ">");
                        dos.writeUTF((String) obj);
                        dos.writeUTF("<" + (String) tags.elementAt(item)
                            + ">");
                    }
                    if (obj instanceof Vector) {
                        dos.writeUTF("strokeintvector");
                        dos.writeInt(((Vector) obj).size());
                        for (int h = 0; h < ((Vector) obj).size(); h++)
                            dos.writeInt(((Integer) ((Vector) obj)
                                .elementAt(h)).intValue());
                    }
                }
            }

        }

    } catch (IOException e) {
    } finally {
        if (fos != null) {

            try {
                fos.close();
            } catch (IOException e) {
            }

        }
    }
}

/**
 * Get registers stored in the device
 *
 */
public void loadData() {
    FileInputStream fis = null;

    try {

        fis = new FileInputStream(CATALOG_NAME + ".PROPERTIES");

        DataInputStream dis = new DataInputStream(fis);

        String line = dis.readLine();
        int record = 0;
        while (line != null) {

            int sizeofslides = dis.readInt();
            for (int slides = 0; slides < sizeofslides; slides++) {
```

```

        Vector slide = new Vector();
        int sizeofcomp = dis.readInt();
        for (int comp = 0; comp < sizeofcomp; comp++) {
            String aux = dis.readUTF();
            if (aux.equals("strokeintvector")) {
                int size = dis.readInt();
                Vector intvector = new Vector();
                for (int pontos = 0; pontos < size; pontos++) {
                    intvector.add(new Integer(dis.readInt()));
                }
                slide.add(intvector);
            } else {
                slide.add(dis.readUTF());
                String consome = dis.readUTF();
            }
        }
        ((Vector) panels.elementAt(3 * record + 2)).add(slide);
    }

    line = dis.readLine();
    record++;
}

} catch (IOException e) {
} finally {
    if (fis != null) {
        try {
            fis.close();
        } catch (IOException e) {
        }
        handlerrecord.loadRecord(handlerrecord.currentScreen,
                                handlerrecord.currentSlide);
    }
}

}

/**
 * class HandlerRecord
 *
 * Handler record to store and get the registers
 *
 */
public class HandlerRecord {
    int currentScreen;

    int currentSlide;

    public HandlerRecord() {
    }

    public void clearScreen() {
        Component[] components = ((Panel) panels.elementAt(currentScreen))
            .getComponents();
        for (int i = 0; i < components.length; i++)
            if (components[i] instanceof TextField)
                ((TextField) components[i]).setText("");
            else if (components[i] instanceof Choice)
                ((Choice) components[i]).select(0);
            else if (components[i] instanceof List)
                for (int k = 0; k < ((List) components[i]).getItemCount(); k++)
                    ((List) components[i]).deselect(k);
    }

    public void saveRecord(int screen, int numberofslide, int mode) {
        Vector slide = new Vector();
        if (mode == 1)
            handlerstrokes.getPoints(screen, numberofslide);
        Panel panel = (Panel) panels.elementAt(screen);
        Component[] components = panel.getComponents();
        for (int i = 0; i < components.length; i++) {
            if (components[i] instanceof TextField) {
                String value = "0" + ((TextField) components[i]).getText();
                slide.add(value);
            }
            if (components[i] instanceof List) {
                String value = "1"
                    + ((List) components[i]).getSelectedIndex()
                    + ((List) components[i]).getSelectedItem();
                slide.add(value);
            }
            if (components[i] instanceof Choice) {
                String value = "2"
                    + ((Choice) components[i]).getSelectedIndex()
                    + ((Choice) components[i]).getSelectedItem();
                slide.add(value);
            }
            if (components[i] instanceof Strokes) {
                slide.add(handlerstrokes.points);
            }
        }
    }
}

```

```

        handlerstrokes.initializeBuffer();
    }
}
if (numberofslide == ((Vector) panels.elementAt(currentScreen + 2))
    .size())
    ((Vector) panels.elementAt(screen + 2)).add(slide);
else
    ((Vector) panels.elementAt(screen + 2)).setElementAt(slide,
        numberOfslide);
}

public void loadRecord(int screen, int slide) {
    Vector aux = (Vector) ((Vector) panels.elementAt(screen + 2))
        .elementAt(slide);
    Panel panel = (Panel) panels.elementAt(screen);
    Component[] components = panel.getComponents();
    int j = 0;
    for (int i = 0; i < components.length; i++) {
        if (components[i] instanceof TextField) {
            String value = ((String) aux.elementAt(j)).substring(1);
            ((TextField) components[i]).setText(value);
            j++;
        }
        if (components[i] instanceof List) {
            for (int k = 0; k < ((List) components[i]).getItemCount(); k++)
                ((List) components[i]).deselect(k);
            String value = ((String) aux.elementAt(j)).substring(1, 2);
            if (!value.equals("-"))
                ((List) components[i]).select((new Integer(value))
                    .intValue());
            j++;
        }
        if (components[i] instanceof Choice) {
            String value = ((String) aux.elementAt(j)).substring(1, 2);
            ((Choice) components[i]).select((new Integer(value))
                .intValue());
            j++;
        }
        if (components[i] instanceof Strokes) {
            Vector vectorint = (Vector) aux.elementAt(j);
            stroke.drawPoints(vectorint);
            j++;
        }
    }
    repaint();
}

public void deleteRecord(int screen, int slide) {
    ((Vector) panels.elementAt(screen + 2)).removeElementAt(slide);
    handlerstrokes.initializeBuffer();
}
}

```

C.2.6. Gerenciamento de Pontos e Componente Stroke

Trecho do código composto por duas partes: 1) implementação da classe do componente de stroke (escrita manual) e 2) gerenciamento e desenho dos pontos gerados no componente stroke.

```

/**
 * class HandlerStrokes
 *
 * Control the generated points in one stroke
 */
public class HandlerStrokes {
    Vector points = new Vector();

    int x, y, oldx, oldy;

    public HandlerStrokes() {
        initializeBuffer();
    }

    public void initializeBuffer() {
        points = new Vector();
    }
}

```

```

        public Vector getPoints() {
            return points;
        }

        public void addPoints(int mode, int px, int py) {
            if (mode == 0) {
                points.add(new Integer(px));
                points.add(new Integer(py));
            } else {
                points.add(new Integer(-1));
            }
        }

        public void getPoints(int screen, int slide) {
            Vector aux = (Vector) ((Vector) panels.elementAt(screen + 2))
                .elementAt(slide);
            for (int i = 0; i < aux.size(); i++) {
                if (aux.elementAt(i) instanceof Vector) {
                    Vector vectorint = (Vector) aux.elementAt(i);
                    for (int j = 0; j < vectorint.size(); j++)
                        points.add(vectorint.elementAt(j));
                }
            }
        }
    }

    /**
     * end class HandlerStrokes
     *
     * Gerenate the component of draw the strokes
     *
     */

    /** class Strokes */
    class Strokes extends Canvas {
        Color linecolor = Color.black;

        public Strokes() {
            super();
            setBackground(Color.white);
        }

        public void paint(Graphics g) {
            Dimension d = getSize();
            g.drawRect(0, 0, d.width - 1, d.height - 1);
        }

        public void clear() {
            Graphics g = getGraphics();
            Dimension d = getSize();
            Color c = Color.white;
            g.setColor(c);
            g.fillRect(1, 1, d.width - 2, d.height - 2);
        }

        public void drawLine(int x, int y, int xl, int yl) {
            Graphics g = getGraphics();
            g.setColor(linecolor);
            g.drawLine(x, y, xl, yl);
        }

        public void drawPoints(Vector points) {
            clear();
            int i = 0;
            Graphics g = getGraphics();
            g.setColor(linecolor);

            while (i < points.size()) {
                while (((Integer) points.get(i)).intValue() != -1) {
                    if (((Integer) points.get(i + 2)).intValue() != -1)
                        g.drawLine((((Integer) points.elementAt(i)).intValue(),
                            ((Integer) points.elementAt(i + 1)).intValue(),
                            ((Integer) points.elementAt(i + 2)).intValue(),
                            ((Integer) points.elementAt(i + 3)).intValue()));
                    else
                        g.drawLine((((Integer) points.elementAt(i)).intValue(),
                            ((Integer) points.elementAt(i + 1)).intValue(),
                            ((Integer) points.elementAt(i)).intValue(),
                            ((Integer) points.elementAt(i + 1)).intValue()));
                    i += 2;
                }
                i += 1;
            }
        }
    }
}

```