UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

MAGREGISTER 1.0: GERADOR DE INTERFACES DE COLETAS DE DADOS PARA PDA'S

GILSON CHEQUETO

BLUMENAU
2007

2007/5-14

GILSON CHEQUETO

MAGREGISTER 1.0: GERADOR DE INTERFACES DE COLETAS DE DADOS PARA PDA'S

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Adilson Vahldick, Especialista

BLUMENAU
2007

2007/5-14

MAGREGISTER 1.0: GERADOR DE INTERFACES DE COLETAS DE DADOS PARA PDA'S

Por

GILSON CHEQUETO

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Adilson Vahldick, Especialista – Orientador, FURB
Membro:	Prof. Jomi Fred Hübner, Doutor – FURB
Membro:	Prof. Francisco Adell Péricas, Mestre – FURB

Blumenau, 11 de julho de 2007

Dedico este trabalho aos meus familiares, amigos e professores do curso, especialmente aqueles que me ajudaram diretamente na realização deste.

AGRADECIMENTOS

Aos meus pais, Geraldo e Guiza, pelo incentivo para que eu fizesse um curso de graduação.

Ao meu filho Júlio César, pois sem ao menos conseguir entender o motivo pelo qual o pai estava tão ausente, teve que abdicar da minha presença em muitos momentos.

À empresa Senior Sistemas Corporativos por ter me apoiado em situações em que precisei de um tempo extra para me dedicar a este trabalho.

Aos meus amigos e companheiros desta trajetória, André e Maicon, por estarem sempre disponíveis para ajudar no desenvolvimento deste, assim como pelo apoio e incentivo de um modo em geral e por terem compartilhado comigo, por cinco anos, um sonho em comum.

A Deus, por ter me dado força em momentos que pensei que não fosse conseguir e por ter me acompanhado nesta trajetória.

Ao meu orientador, Adilson Vahldick, por todo apoio prestado e por ter acreditado na conclusão deste trabalho.

A vida só pode ser entendida olhando-se para trás. Mas só pode ser vivida olhando-se para frente.

S. Kierkegaard

RESUMO

Este trabalho apresenta a especificação e o desenvolvimento de um software gerador de interfaces para coletas de dados em PDA's. Nesse protótipo definem-se modelos de tabelas e interfaces que serão utilizadas no PDA. Isso facilita a geração de programas para dispositivos móveis através da simples especificação do modelo de dados, pois torna-se desnecessário o uso de uma linguagem de programação. O modelador de interfaces foi desenvolvido com Java para interfaces gráficas, utilizando *Swing*. Os programas gerados para executarem no PDA são compatíveis com a especificação *Java Micro Edition* (JME) e utilizam o banco de dados DB2 Everyplace.

Palavras-chave: Dispositivos móveis. Gerador. Java. JME. Interfaces. Coleta de dados.

ABSTRACT

This work presents the specification and the development of a software to generate interfaces for collect data in PDA' s. In this prototype, models of tables and interfaces are defined that will be used in the PDA. This facilitates the generation of programs for mobile devices through the simple specification of the model of data, therefore the use of a programming language becomes unnecessary. The modeller of interfaces was developed with Java for graphical interfaces, using *Swing*. The generated programs to execute in the PDA are compatible with the specification *Java Micro Edition* (JME) and uses the DB2 Everyplace data base.

Key-words: Mobile devices. Generator. Java. JM. Interfaces. Data collect.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

API – *Application Programming Interface*

ASP – *Active Server Pages*

AWT – *Abstract Window Toolkit*

CD – *Compact Disc*

CDC – *Connected Device Configuration*

CE – *Compact Edition*

CLDC – *Connected Limited Device Configuration*

GUI – *Graphical User Interface*

HPC – *Handheld Personal Computer*

HTML – *Hyper Text Markup Language*

HTTP – *Hipertext Transfer Protocol*

JAD – *Java Application Descriptor*

JAR – *Java Archive*

JDBC – *Java Database Conectivity*

JEE – *Java Platform, Enterprise Edition*

JME – *Java Micro Edition*

JSE – *Java Platform, Standard Edition*

JVM – *Java Virtual Machine*

MIDP – *Mobile Information Device Profile*

MP3 – *MPEG Audio Layer-3*

MSDE – *Microsoft SQL Server Desktop Engine*

MVC – *Model View Controller*

OS – *Operational System*

PC – *Personal Computer*

PDA – *Personal Digital Assistant*

PRC – *Palm Resource*

RMI – *Remote Method Invocation*

RTF – *Rich Text Format*

SQL – *Structured Query Language*

URL – *Universal Resource Locator*

VTL – *Velocity Template Language*

XML - *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Segundo Mattos (2005, p. 3), a tecnologia de desenvolvimento de sistemas para equipamentos móveis não pára de evoluir. Ao observar os últimos modelos de telefones celulares é perceptível esta evolução, pois são inúmeros os recursos que eles possuem. Modelos de *Personal Digital Assistants* (PDAs), televisores digitais e até refrigeradores ganham a cada dia interfaces mais sofisticadas, que permitem total interação e facilidade no acesso às informações. Atualmente, a produção de dispositivos móveis em todo o mundo já é bem superior à de *Personal Computers* (PCs), o que criou um grande mercado de trabalho em potencial para desenvolvedores de sistemas para dispositivos móveis e fez com que muitos programadores buscassem especializações nesta área. Mattos (2005, p. 3) afirma que dentre as ferramentas utilizadas, a principal delas é o *Java Micro Edition* (JME), a plataforma Java para desenvolvimento de sistema para aparelhos portáteis, o que inclui softwares para dispositivos como celulares e *pagers*, passando por refrigeradores, televisores digitais e até automóveis.

De acordo com PalmLand (2007), PDA's utilizam geralmente o sistema operacional Windows CE ou Palm OS, que permitem uma imensa gama de opções tanto em software como em multimídia. São muito práticos na vida profissional e podem ser um excelente companheiro para diversão, pois os últimos modelos já incorporam fone de ouvido onde se pode escutar músicas com qualidade de *Compact Disc* (CD) utilizando o formato MPEG Audio Layer-3 (MP3). Agendas de números e endereços, anotações rápidas, gravador de sons, alto falantes, tela colorida de alta resolução são as maiores diferenças para uma agenda eletrônica normal. Os PDA's em um futuro próximo serão integrados com os *smartphones*[1], o que tornará mais fácil o acesso a estes equipamentos.

Em virtude da expansão das tecnologias de desenvolvimento de softwares para dispositivos móveis e da constatação da necessidade do mercado corporativo em utilização de softwares para coletas de dados, principalmente para serem utilizados em processos de inventários de estoque e patrimoniais, surge a idéia de desenvolver nesse trabalho uma ferramenta de geração de interfaces para a coleta de dados em ambientes externos ao ambiente de operacionabilidade de um sistema de gestão empresarial. O principal objetivo deste sistema é permitir que sejam gerados códigos fontes de modelos de telas para serem executadas em dispositivos móveis, para a finalidade de serem efetuadas coletas de dados

[1] Celulares com funcionalidades estendidas através de programas que executam sem seus sistemas operacionais.

externos e pertinentes a processos do sistema de gestão. Para tornar o software compatível com os diversos sistemas de gestão que estão no mercado, esta ferramenta é capaz de coletar informações nos mais variados formatos de dados, bem como exportar os dados coletados para PCs em *eXtensible Markup Language* (XML), tornando assim o sistema no dispositivo móvel flexível e tolerante às exigências de formatos do sistema de gestão. Trata-se de uma ferramenta de geração de interfaces para PDA's, através da qual é possível informar quais campos farão parte da tela de coleta de dados. Com base nos campos informados na modelagem do sistema, são criadas estruturas no banco de dados do dispositivo móvel, para que possam ser gravadas as informações coletadas através da tela.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver um sistema gerador de interfaces para dispositivos móveis. Essas interfaces são compatíveis com a especificação JME.

Os objetivos específicos do trabalho são:

a) disponibilizar um modelador de interfaces a ser desenvolvido na linguagem Java e executado na forma de uma aplicação *desktop,* que gravará a modelagem dos leiautes no formato XML;
b) disponibilizar na mesma aplicação *desktop* do modelador de interfaces, um gerador de código que siga a especificação JME capaz de carregar o leiaute da interface gravada em formato XML, gerar o código fonte da tela que executará no dispositivo móvel, chamar a compilação do código gerado e transferir a aplicação Java para o dispositivo móvel;
c) disponibilizar, através do código fonte gerado em JME, uma tela a ser executada no dispositivo móvel que contenha os campos informados no leiaute configurado na aplicação *desktop*, permitindo que os dados sejam coletados e gravados em um banco de dados DB2 Everyplace;
d) permitir que os dados coletados no PDA sejam transferidos para PCs através do uso da linguagem XML.

1.2 ESTRUTURA DO TRABALHO

A estrutura deste trabalho está dividida em quatro capítulos. O segundo capítulo apresenta as tecnologias utilizadas no desenvolvimento do protótipo, demonstrando conceitos teóricos que justificam as suas utilizações no desenvolvimento deste trabalho. O terceiro capítulo apresenta o desenvolvimento do protótipo, sua especificação, ferramentas utilizadas e implementação, assim como os resultados e discussões. Por fim, o quarto capítulo apresenta as limitações, extensões e conclusões.

2 FUNDAMENTAÇÃO TEÓRICA

Nas seções seguintes são apresentados alguns aspectos teóricos relacionados aos recursos e tecnologias utilizadas neste trabalho, tais como: tecnologia Java, plataforma JME, máquina virtual J9, *Servlets* Java, *templates*, motor de *templates Velocity* e banco de dados DB2 Everyplace.

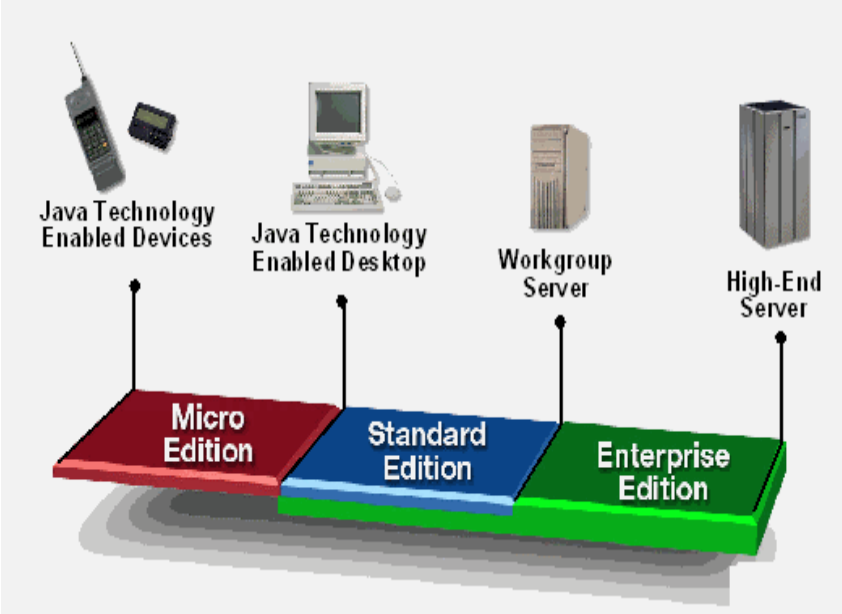
2.1 TECNOLOGIA JAVA

Conforme Sun (2007), a tecnologia Java define uma linguagem de programação e uma plataforma de software, cujo principal objetivo é a portabilidade do código. Isto significa que um programa escritos apenas uma vez e em apenas uma linguagem, podem ser executados em diferentes sistemas operacionais, assim como também em diferentes tipos de dispositivos, tais como *desktops*, servidores, PDAs, celulares e outros. De acordo com Schmitt Junior (2004), os programas criados com a linguagem Java não geram códigos nativos para os sistemas operacionais. Ao invés disso geram código compilado, chamado de *bytecode*, que são interpretados através de outro programa chamado *Java Virtual Machine* (JVM), que por sua vez, interpreta o código compilado e gera o código binário para o sistema operacional onde está sendo executado o programa.

De acordo com Sun (2007), a plataforma Java é dividida em três edições:

a) *Java Platform, Standard Edition* (JSE): contém a *Application Programming Interface* (API) básica de todos os programas Java;
b) *Java Platform, Enterprise Edition* (JEE): conjunto de especificações para atender a arquitetura de sistemas corporativos;
c) *Java Platform, Micro Edition* (JME): possibilita a construção de aplicações voltadas para dispositivos móveis como celulares e *handhelds*.

A figura 1 mostra como está dividida a plataforma Java.



Fonte: Souza (2003).

Figura 1 – A plataforma Java

2.1.1 *Java Micro Edition*

De acordo com Mattos (2005, p. 16), a plataforma JME é uma edição da linguagem Java, destinada a dispositivos com recursos limitados de memória, vídeo e processamento, que satisfaz a necessidade dos seguintes usuários:

a) consumidores e fabricantes de equipamentos eletrônicos computadorizados;
b) provedores de serviços que desejam distribuir o conteúdo de suas soluções por meio de dispositivos móveis;
c) criadores de conteúdos que pretendem construir produtos agradáveis para dispositivos pequenos e limitados, com o menor esforço possível.

A JME possui um conjunto de tecnologias que podem ser usadas na construção de aplicativos, incluindo:

a) JVMs que se enquadram em uma larga quantidade de dispositivos computacionais;
b) bibliotecas especializadas para cada tipo de dispositivo;
c) ferramentas para o desenvolvimento e implantação de softwares e configuração de dispositivos.

Segundo Mattos (2005, p. 18), a plataforma JME enfoca duas categorias de dispositivos:

a) dispositivos para consumidores de alto padrão: essa categoria está agrupada na

divisão de tipos de dispositivos móveis *Connected Device Configuration* (CDC), que inclui equipamentos como aparelhos televisores, videofones com internet, comunicadores sem fio e sistemas de entretenimento e navegação automobilística. Esses tipos de equipamentos possuem boa capacidade de interação, memória e conexão com a internet. Apesar de terem sido projetados há muito tempo, esses equipamentos ainda não integram a realidade brasileira;

b) dispositivos para consumidores de baixo padrão: essa categoria está agrupada na divisão de tipos de dispositivos móveis *Connected Limited Device Configuration* (CLDC), incluindo celulares, *pagers*, organizadores pessoais, etc. Esta categoria possui capacidade de conexão, armazenamento, memória reduzida e conectividade de baixa velocidade, se comparadas às taxas de computadores pessoais

2.1.2 Configurações e perfis

Segundo Sun (2007), as tecnologias JME são baseadas em configurações e perfis. Uma configuração define um conjunto mínimo de classes e bibliotecas que estão disponíveis para grupos de dispositivos. Um perfil define o conjunto de APIs disponíveis para uma família de dispositivos em particular. Por exemplo, o perfil para telefones celulares é separado do perfil para organizadores pessoais, mas ambos perfis trabalham com a mesma configuração.

Conforme Silva (2005, p. 19), as configurações do JME definem uma plataforma Java para uma determinada faixa de dispositivos, definidos por uma série de características como:

a) informações sobre a memória do dispositivo;
b) tipo e velocidade do processador;
c) conectividade do aparelho.

Os perfis do JME podem ser descritos como extensões das configurações, que definem bibliotecas para o desenvolvimento de aplicações para um determinado dispositivo. Estas bibliotecas podem ser de interface gráfica, armazenamento persistente, segurança e conectividade. Cada configuração da JME pode ter um ou mais perfis associados. Conforme Schmitt Junior (2004), alguns perfis associados à configuração CDC são :

a) *Foundation Profile*: conjunto de APIs Java que fornece uma implementação de rede do CDC que pode ser usada para construir aplicações sem interface com o

usuário

b) *Personal Profile:* perfil CDC utilizado em dispositivos que necessitam de um suporte completo para interfaces, como PDAs e consoles para jogos. Ele inclui a biblioteca *Abstract Window Toolkit* (AWT) completa e é fiel ao ambiente web, executando facilmente *applets*[2] feitos para ambientes desktop;

c) *Remote Method Invocation* (RMI) *Profile*: perfil projetado para as plataformas que suportam a configuração do CDC;

d) *Personal Basis Profile*: perfil que suporta pacotes adicionais.

São perfis associados à configuração CLDC, os perfis *Mobile Information Device Profile* (MIDP), que trata-se de um conjunto de pré-requisitos que são implementados com determinada configuração e *Personal Digital Assistant Profile* (PDAP).

A figura 2 demonstra as camadas da arquitetura JME.

Fonte: Fernando Andriolli (2007).

Figura 2 – A arquitetura JME

2.1.3 Perfil MIDP

Conforme Wilding-McBride (2003, p. 31), a especificação do perfil MIDP é voltada

[2] pequenos programas feitos em Java, que se transferem com as páginas web e que o navegador executa no espaço da página.

para dispositivos que suportam a configuração CLDC. Normalmente são telefones celulares e PDAs. De acordo com a especificação do perfil MIDP, o dispositivo deve ter pelo menos 128 kb de memória não volátil para o armazenamento da implementação, 32 kb de memória volátil para a pilha Java e 8 kb de memória não volátil para armazenamento persistente. O dispositivo deve ainda ter algum recurso de entrada de dados como um teclado, tela por toque ou área de teclas como telefones celulares. É necessário ainda a possibilidade de conexão a algum tipo de rede e um visor de pelo menos 96 x 54 pixels.

Segundo Sun (2007), o perfil MIDP combinado com a configuração CLDC disponibiliza recursos para dispositivos como telefones celulares, PDAs, etc., provendo funcionalidades requeridas por aplicações móveis, incluindo interfaces com usuários, conectividade com a internet e armazenamento de dados.

Conforme Schmitt Junior (2004, p. 23), *MIDlet* é uma aplicação que implementa o perfil MIDP e roda sobre a máquina virtual KVM proposta pela configuração CLDC. Um *MIDlet* pode utilizar tanto funções oferecidas no perfil MIDP como funções que o MIDP herda da CLDC. Uma aplicação MIDP deve conter no mínimo uma classe *MIDlet*, podendo em alguns casos conter mais de um, sendo chamada de *MIDlet Suite*. A aplicação que segue o MIDP, para ser instalada no dispositivo, precisa ser empacotada, criando um Java Archive (JAR) que obrigatoriamente contém um arquivo de manifesto, englobando as informações sobre o *MIDlet* contido no pacote JAR. Este arquivo contendo as informações sobre a aplicação, configuração e perfil utilizado na mesma, é denominado *Java Application Descriptor* (JAD).

De acordo com Muchow (2001, p. 444), para que possa ser executado no Palm OS, este par de arquivos deve ser convertido em um único arquivo denominado *Palm Resource* (PRC). Segundo Rodhes (1998, p. 89), uma aplicação destinada ao sistema operacional Palm é armazenada em forma de recursos, contendo o código da aplicação, interfaces, ícones, textos e assim por diante.

2.1.4 IBM J9 VM

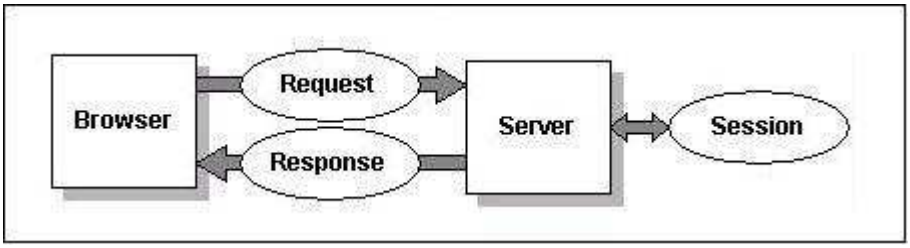
Conforme IBM (2007), a máquina virtual J9 implementa uma arquitetura configurável e compacta que provê uma interface comum para aplicações executarem em diferentes dispositivos e sistemas operacionais. A J9 executa nos sistemas operacionais (PalmOS, PocketPC, QNX, Linux embarcado, OSE, ITRON, etc.) e gerencia as interfaces específicas

com o sistema operacional e com o hardware do dispositivo. Além disto, esta máquina virtual possui o suporte às configurações CLDC 1.1, MIDP 2.0 além de outras configurações e perfis não disponibilizados pela Sun Microsystems para a plataforma Palm.

Pelo fato de terem sido utilizados recursos da configuração CLDC 1.1 que as máquinas virtuais da Sun não suportam, foi utilizada a IBM J9 VM no desenvolvimento deste trabalho.

2.2 SERVLETS JAVA

Segundo Schmitt Junior (2004, p.24), *servlet* Java é uma aplicação Java semelhante a uma página da internet, para ser executada num servidor e atender a requisições web. O *servlet* recebe a requisição com seus parâmetros, processa e devolve a resposta. O acesso ao *servlet* pode ser executado através do protocolo *Hipertext Transfer Protocol* (HTTP). Os parâmetros podem ser enviados juntamente com o endereço *Universal Resource Locator* (URL) executado. O resultado do processamento do *servlet* é dado em informações literais, podendo ser uma página internet ou dados no formato XML. A figura 3 apresenta uma visão macro a respeito do processo de requisições a um *servlet*.



Fonte: Sun (2007).

Figura 3 – Visão macro de uma requisição a um servlet

O quadro 1 apresenta o código fonte de um *servlet* que recebe uma requisição e retorna um código fonte *Hyper Text Markup Language* (HTML) para o *browser*. Ao receber o retorno do *servlet*, o *browser* irá imprimir uma mensagem na tela.

```
import java.io.*;
import java.net.*;

import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldServlet extends HttpServlet {

    protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hello World</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("</body>");
        out.println("</html>");
        out.close();
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    public String getServletInfo() {
        return "Short description";
    }
}
```

Quadro 1 – Exemplo de um servlet Java

2.3 GERADORES DE CÓDIGO

Segundo Herrington (2003, p. 3), a geração do código é a técnica utilizada para a construção de código através da utilização de programas. Os geradores de código podem trabalhar em forma de linha de comando ou usar uma interface gráfica, assim como podem construir código para várias linguagens de programação, de uma única vez ou em etapas. As entradas e saídas são definidas conforme a necessidade, sendo que o desenvolvedor especifica o código de saída de forma manual.

De acordo com Herrington (2003, p. 15-16), a utilização de geradores de código traz benefícios substanciais ao desenvolvimento do projeto, aumentando a produtividade e a qualidade do software. As vantagens são alcançadas nos seguintes aspectos:

a) qualidade: grandes quantidades de código escritos manualmente tendem a possuir uma baixa qualidade. A qualidade do código gerado está relacionada a qualidade do código definido nos *templates*. Quanto maior a qualidade do código utilizado pelo gerador, maior a qualidade do código resultante;
b) consistência: o código gerado pelo gerador de código é consistente. A padronização da nomenclatura de classes, métodos e variáveis o torna de fácil entendimento e uso;
c) único ponto de conhecimento: caso se faça uso de vários geradores, ou de um gerador que tenha como finalidade a geração de código para várias camadas de uma aplicação, basta que o desenvolvedor conheça o ponto onde a alteração deverá ser efetuada. Uma vez alterada a entrada do gerador, ela será propagada para os processos conseguintes;
d) maior tempo para o projeto: o cronograma de um projeto que utiliza a geração de código é completamente diferente do cronograma de um projeto onde o desenvolvimento é totalmente manual. Em um projeto que adota o desenvolvimento manual, o uso inadequado de uma API pode ocasionar na reescrita de uma grande quantidade de código. Com o uso de geradores de código, os desenvolvedores podem modificar os *templates* para adequar-se a API e executar o gerador novamente, e todo o código gerado estará modificado contemplando as necessidades;
e) decisões atípicas do projeto: o alto nível das regras de negócio são perdidas em minuciosas implementações de código. Como geradores de código utilizam arquivos pequenos, que especificam o “esqueleto” do código a ser gerado, as pequenas exceções são mais fáceis de serem tratadas, não sendo necessário avaliar um resultado de centenas de linha de código;
f) abstração: os geradores de código podem construir códigos a partir de modelos abstratos do código alvo. Essa abstração permite que o código gerado possa ser facilmente migrado para outras linguagens e plataformas e a análise de negócio revista no escopo da abstração e não no código de execução;
g) agilidade no desenvolvimento: uma característica chave da geração de código é a maleabilidade do código de saída. A nível de negócio, o software será de mais fácil manutenção e implementação de novas funcionalidades ao longo do tempo.

Herrington (2003, p. 77) afirma que a construção de um gerador de código pode seguir as seguintes etapas de desenvolvimento:

a) identificação do que se deseja obter como saída do gerador;
b) definição de qual será a entrada e como a mesma será analisada;
c) interpretação e recuperação as informações da entrada, definindo a formatação e a geração da saída;
d) geração da saída a partir das informações extraídas do arquivo de entrada.

Stephens (2002) afirma que as possibilidades de geração de código são ilimitadas, mas não significa que deva ser utilizada em qualquer situação ou para qualquer projeto. É mais comum o uso de geradores de código em áreas específicas, incluindo geradores de interfaces, de acesso à base de dados, de documentação a partir de comentários presentes no código-fonte, de comandos para a linguagem SQL, de componentes e bibliotecas JSP

2.4 *TEMPLATES*

Conforme Takecian et al. (2004, p. 6), *templates* são arquivos que descrevem uma estrutura de formatação, que contém variáveis, blocos especiais e diretivas, indicando valores a serem inseridos e ações a serem executadas pelo sistema. Possuem a finalidade de gerar outros arquivos que contenham a formatação previamente definida no arquivo base. São muito utilizados em geração de código, pois o código de saída tem a tendência de seguir um padrão estabelecido.

De acordo Schvepe (2006, p. 11), *templates* podem conter códigos estáticos e dinâmicos. Código dinâmico é aquele que passa por transformações no processo de geração do código de saída e código estático é definido literalmente e não sofre alterações pelo motor de *templates*.

2.5 MOTOR DE *TEMPLATES* VELOCITY

Segundo Steil (2007), o Velocity é um motor de template amplamente utilizado no desenvolvimento de aplicações web, onde o código Java fica totalmente separado do código HTML, tornando assim a aplicação muito mais modularizada e fácil de manter. Steil (2007) afirma que apesar do seu maior uso ser nesta área, não está limitado a apenas isso, muito pelo

contrário. O Velocity pode ser usado para quase tudo o que possa ser imaginado em relação a processamento de textos, como por exemplo formatação de mensagens com base em um template, criação de documentos *Rich Text Format* (RTF), *Structured Query Language* (SQL), XML e assim por diante.

Conforme Moura e Cruz (2002), o Velocity não é um programa, mas um conjunto de classes Java, utilizado principalmente no desenvolvimento de aplicações web. Possui uma linguagem de manipulação de *templates*, a *Velocity Template Language* (VTL), que permite fazer a inserção de informações de maneira dinâmica dentro do *template*. A VTL define as referências das variáveis no *template*, possui controles de fluxo de execução, como *loops* e comandos condicionais, permite a definição de macros e uma estrutura básica de substituição de valores de variáveis.

Segundo Moura e Cruz (2002), os dados formatados na forma textual são gerados a partir dos dados brutos e de uma descrição da formatação. Os dados brutos devem estar encapsulados em classes Java e serem acessíveis por meio de uma interface pública (métodos públicos). O Velocity infere, em tempo de execução, esta interface pública e através dela pode recuperar os dados armazenados nas classes.

2.6 DB2 EVERYPLACE

Conforme Goya (2004), o IBM DB2 Everyplace é um banco de dados pago, disponível para as plataformas Palm OS e Windows CE, EPOC-32, Neutrino, Linux e Win32. O DB2 Everyplace é o menor banco de dados do mundo e ocupa aproximadamente 100k de espaço. Foi desenvolvido para ser utilizado em dispositivos de baixo custo, com pouco poder de processamento e com poucos recursos gráficos, como por exemplo PDAs e *Handheld Personal Computers* (HPCs). Segundo Goya (2004), os dados nos dispositivos móveis podem ser sincronizados com outros bancos de dados DB2 e até mesmo com banco de dados de outros fabricantes, como Oracle e Microsoft, desde que haja um software de sincronismo da IBM chamado Mobile Connect. Este sincronismo pode ser feito para um outro banco de dados local ou remoto, sendo que neste último caso também é necessário um software especial para fazer a integração.

Apesar de ser o menor banco de dados do mundo, o DB2 Everyplace tem várias funcionalidades interessantes como: suporte a sub-consultas, criação de visões, criação de

gatilhos (*triggers*), criação de procedimentos armazenados (*stored procedures*) e criação de funções definidas pelo usuário.

Entretanto, possui as seguintes limitações:

a) 15 índices por tabela;
b) 8 chaves estrangeiras por tabela;
c) 8 colunas por índice;
d) 8 colunas por chave primária;
e) número máximo de registros por tabela depende do tamanho da tabela;
f) 128 colunas por tabela;
g) 32Kb de tamanho máximo de colunas caracteres, alfanuméricas e de arquivos;
h) 31 dígitos para números inteiros.

O DB2 Everyplace está disponível para ser utilizado na plataforma JME através das classes conhecidas por *Java Database Conectivity* (JDBC), que podem ser adicionadas ao projeto do software do dispositivo móvel.

2.7 TRABALHOS CORRELATOS

Existem aplicações com características semelhantes à desenvolvida neste trabalho. Com relação a aplicativos para dispositivos móveis, pode-se citar o aplicativo para representante comercial em PDA usando a tecnologia JME e banco de dados (SILVA, 2005), assim como o protótipo de software para dispositivos móveis utilizando Java ME para cálculo de regularidade em rally (DEPINÉ, 2002)**.** Com relação a ferramentas de geração de código, tem-se o sistema de gerenciamento customizável baseado em PDA's (LESSNAU, 2002).

2.7.1 Aplicativo para representante comercial em dispositivo móvel (PDA) usando a tecnologia JME e banco de dados

Silva (2005) desenvolveu um protótipo de software para representantes comerciais utilizando dispositivos móveis. A aplicação disponibiliza dados relevantes da empresa para os seus representantes, visando auxiliar a troca de informações entre a empresa e funcionários que trabalham fora dela. O prótipo faz uso da tecnologia JME voltada para estes dispositivos e

uso de banco de dados em dispositivos móveis.

As principais funcionalidades do protótipos são:

a) disponibilizar dados de clientes e situação da carteira de vendas para usuários que utilizam PDA;
b) importar dados disponibilizados pela empresa através do uso da linguagem XML;
c) permitir alterações dos dados no dispositivo móvel;
d) disponibilizar dados do dispositivo móvel para a empresa através do uso da linguagem XML;

2.7.2 Sistema de gerenciamento customizável baseado em PDA's

Lessnau (2002) desenvolveu um sistema de gerenciamento de dados customizável baseado em PDAs. O sistema apresenta uma arquitetura cliente/servidor, tendo como clientes PDA's que possuam suporte a aplicações Java, independentemente do hardware ou do sistema operacional. As aplicações atendidas são os sistemas que necessitem de coleta móvel de dados, como controle de venda, sistemas de pesquisa de opinião, sistemas de requisição de produtos, entre outros. Para a implementação do módulo servidor foi utilizada a tecnologia *Active Server Pages* (ASP), com gerenciamento de dados via banco de dados *Microsoft SQL Server Desktop Engine* (MSDE)[3].

A aplicação servidora possui as seguintes funcionalidades:

a) criação de um novo sistema, ou seja, o primeiro passo para a sua instalação;
b) definição de configurações e customizações de alguns parâmetros;
c) disponibilização do software que será instalado no PDA;

A aplicação final a ser executada nos PDA's oferece as seguintes funcionalidades:

a) interface de entrada de dados com gravação dos dados;
b) opção de comunicação com a base central (feita em lotes);
c) transferência dos dados para o módulo servidor;

[3] Banco de dados proprietário desenvolvido pela Microsoft.

2.7.3 Protótipo de software para dispositivos móveis utilizando Java ME para cálculo de regularidade em rally.

Depiné (2002) desenvolveu um protótipo de software para dispositivos móveis utilizando JME para cálculo de regularidade em *rally*. Este protótipo objetivou demonstrar a tecnologia JME e verificar suas limitações quando aplicada à equipamentos portáteis, especificar e implementar um protótipo de software para cálculo de planilha de *rally* para dispositivos móveis e entrar com as informações sobre a planilha do *rally* no celular, retornando os tempos ideais de passagem nos trechos. As principais funcionalidades do protótipo são:

a) armazenar em memória as informações de cada linha dos trechos da planilha de *rally*;
b) obedecer a lógica do livro de bordo;
c) permitir informar o tempo ideal em horas, minutos e segundos;
d) permitir que o usuário altere o tempo ideal calculado pelo protótipo;
e) demonstrar os tempos calculados.

3 DESENVOLVIMENTO DO TRABALHO

O resultado deste trabalho é um protótipo de uma ferramenta destinada para a geração de interfaces de coletas de dados para PDA's. A ferramenta gera programas para dispositivos móveis que possuem suporte a JME, com o perfil MIDP 2 e configuração CLDC 1.1.

O protótipo utiliza a tecnologia Java tanto na parte da ferramenta quanto na parte dos programas gerados. Os programas gerados para os dispositivos móveis fazem uso de um banco de dados relacional para a persistência dos dados.

Os programas gerados pela ferramenta se conectam a um servidor de internet por meio de um *servlet*, através do qual podem enviar solicitações de dados para que sejam persistidos nas tabelas auxiliares e também para enviarem os dados coletados. O *servlet* foi implementado manualmente, ou seja, não é gerado pela ferramenta. A troca das informações é feita através da transferência de arquivos no formato XML.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Nesta seção são apresentados os requisitos funcionais e não funcionais que detalham as principais funcionalidades que a ferramenta deve ter, assim com também detalham questões referente ao ambiente de execução, recursos da ferramenta e dos *softwares* que ela permite gerar.

3.1.1 Requisitos funcionais

A ferramenta deverá:

a) permitir que sejam cadastrados os leiautes de dados através de uma aplicação *desktop*. Esses leiautes deverão ser compostos por tabelas, com seus respectivos campos, tipos, formatos e relacionamentos;
b) permitir que sejam cadastradas as interfaces de coletas de dados que a ferramenta deverá gerar. No cadastro destas interfaces, deverá ser possível indicar qual leiaute estará associado à interface, bem como definir uma ordem para disposição dos

campos no programa que a ferramenta irá gerar e também efetuar associações deste campos com campos bases de outros leiautes para sugestão automática de valores;

c) permitir que sejam geradas aplicações JME, contendo itens de menu para cada um dos leiautes associados a interfaces de coletas de dados. Na geração das interfaces, os componentes visuais deverão ser de acordo com o tipo dos campos, assim como dispostos na ordem que foram definidos na ferramenta. Campos com relação para outras tabelas, também deverão ter um tipo específico de componente visual;
d) disponibilizar na aplicação JME a criação das tabelas relacionadas à interface de coleta, logo na primeira vez em que o software seja executado no dispositivo móvel;
e) disponibilizar um recurso de importação de tabelas auxiliares, que são aqueles que se relacionam à tabela da interface;
f) disponibilizar para que sejam feitas coletas de dados nos dispositivos móveis através do programa gerado e armazenados num banco de dados;
g) permitir que os dados coletados nos dispositivos móveis sejam transferidos entre dispositivo móvel e computador.

3.1.2 Requisitos não funcionais

A ferramenta deverá:

a) ser desenvolvida em Java;
b) gerar programas para dispositivos móveis na linguagem Java para a plataforma JME;
c) utilizar o banco de dados DB2 Everyplace;
d) disponibilizar nos dispositivos móveis interfaces de boa usabilidade.

3.2 ESPECIFICAÇÃO

A especificação desta ferramenta foi feita através da utilização dos diagramas da

Unified Model Language (UML). Foram elaborados, na ferramenta Enterprise Architect, dois diagramas de casos de uso, um diagrama de atividades e um diagrama de classes, que demonstram a especificação da regra de negócio da ferramenta. Os diagramas de casos de uso demonstram as possíveis operações executadas pelo usuário gerador do projeto no uso da ferramenta. O diagrama de atividades mostra a seqüência do processo macro de criação de um projeto. Por fim, o diagrama de classes explica como as classes de negócio estão estruturadas no sistema.

3.2.1 Diagramas de casos de uso

No protótipo deste trabalho foram definidos nove casos de usos demonstrados em dois diagramas. O primeiro diagrama apresenta casos de uso que demonstram as operações que o usuário exerce sob a ferramenta e o segundo diagrama apresenta casos de uso que demonstram as operações do usuário sob o um exemplo de *software* para dispositivos móveis gerado pela ferramenta.

A figura 4 apresenta o primeiro diagrama de casos de uso.

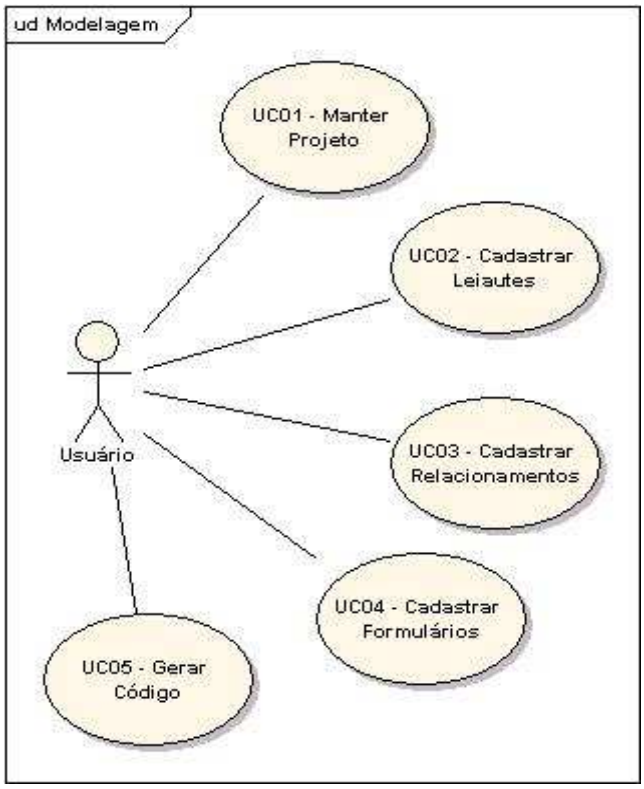


Figura 4 – Diagrama de casos de uso da ferramenta

O quadro 2 apresenta a descrição do caso de uso UC01 – Criar projeto.

Manter Projeto

Sumário:
O usuário efetua as operações referentes à definição do projeto, criando novos projetos, alterando e salvando as suas configurações.

Ator primário:
Usuário.

Fluxo principal: Criar novo projeto
1) O Usuário clica no botão “Novo”;
2) Os campos da tela são habilitados para digitação;
3) O usuário preenche as configurações do projeto: Descrição, data e observações;
4) O usuário clica no botão “Salvar”;
5) O sistema verifica se todas as informações foram preenchidas corretamente;
6) O usuário informa o nome do arquivo para salvar o projeto;
7) O sistema salva as informações do projeto em um arquivo XML.

Fluxo alternativo: Abrir projeto existente
No passo 1, caso o usuário opte por abrir um projeto previamente salvo:
1.1) Usuário clica no botão “Abrir”;
1.2) Usuário seleciona o projeto desejado;
1.3) Os campos da tela são habilitados e carregados com as definições do projeto;
1.4) Retorna ao passo 3.

Pós-condições:
É criado um arquivo XML contendo as definições do projeto.

Quadro 2 – Caso de uso UC01 - Criar projeto

O quadro 3 apresenta a descrição do caso de uso UC02 – Cadastrar leiautes.

Cadastrar leiautes

Sumário:
O usuário efetua as operações referentes ao cadastro dos leiautes do projeto, criando novos leiautes, alterando e excluindo.

Ator primário:
Usuário.

Pré-condições:
Um projeto deve ter sido criado ou aberto.

Fluxo principal: Criar novo leiaute
1) O usuário clica na página leiautes;
2) O usuário informa o nome de um leiaute ainda não cadastrado;
3) O sistema habilita os campos da tela referente ao leiaute;
4) O usuário cadastra os campos do leiaute, informando seus nomes, tipos e tamanhos;

Fluxo alternativo: Alterar um leiaute
No passo 1, caso o usuário opte por abrir um leiaute previamente cadastrado:
1.1) O usuário informar o nome ou selecionar um leiaute já cadastrado;
1.2) O sistema carrega a grade com os campos já incluídos no leiaute e habilita a tela para digitação;
1.3) Retorna ao passo 4.

Quadro 3 – Caso de uso UC02 – Cadastrar leiautes

O Quadro 4 apresenta a descrição do caso de uso UC03 – Cadastrar relacionamentos.

Cadastrar Relacionamentos

Sumário:
O usuário efetua as operações referentes ao cadastro dos relacionamentos entre os leiautes, criando novos relacionamentos e excluindo.

Ator primário:
Usuário.

Pré-condições:
Devem ter sido cadastrados ao menos dois leiautes com campos compatíveis para que o relacionamento seja cadastrado. Um dos leiautes deve ter pelo menos um campo definido como sendo chave primária.

Fluxo principal: Criar novo relacionamento
1) O usuário clica na página relacionamentos;
2) O usuário seleciona um leiaute origem;
3) O usuário seleciona um leiaute destino. Este leiaute deve conter uma chave primária para que o relacionamento possa ser feito;
4) O sistema adiciona automaticamente os campos que compões a chave primária do leiaute destino;
5) O usuário seleciona os campos do leiaute origem que se relacionarão com os campos do leiaute destino;
6) O usuário clica no botão de confirmação.

Fluxo alternativo: Excluir relacionamento
No passo 1, caso o usuário opte por abrir excluir um relacionamento previamente cadastrado:
1.1) O usuário seleciona o relacionamento na lista à esquerda da tela;

1.2) O usuário clica no botão de exclusão.

Fluxo de exceção: Validar informações
No passo 5, caso o usuário selecione campos que não são compatíveis com os respectivos campos da chave primária do leiaute destine:
5.1) O sistema apresenta uma mensagem ao usuário informando que o campo não pode relacionar-se;
5.3) Retorna ao passo 5.

Quadro 4 – Caso de uso UC03 – Cadastrar relacionamentos

O quadro 5 apresenta a descrição do caso de uso UC04 – Cadastrar formulários.

Cadastrar Formulários

Sumário:
O usuário efetua as operações referentes ao cadastro das interfaces de coletas de dados, criando novas interfaces, alterando e excluindo.

Ator primário:
Usuário.

Pré-condições:
Deve ter sido cadastrado ao menos um leiaute, para que o mesmo possa ser associado ao formulário que estará sendo criado.

Fluxo principal: Criar novo formulário
1) O usuário clica na página formulários;
2) O usuário informa o nome do formulário;
3) O usuário informa o leiaute ao qual o formulário se refere;
4) O usuário informa o título do formulário que deverá aparecer na janela do programa no dispositivo móvel
5) O usuário adiciona os campos do leiaute que estarão disponíveis no formulário;
5) Para cada campo do formulário o usuário seleciona os relacionamentos e campos bases que farão com que seja possível que o programa gerado carregue valores automaticamente.

Fluxo alternativo: Excluir formulário
No passo 1, caso o usuário opte por excluir um formulário previamente cadastrado:
1.1) O usuário informa o nome de um formulário já cadastrado ou seleciona-lo na lista
1.2) O usuário clica no botão de exclusão.

Fluxo alternativo: Alterar campos do formulário
No passo 1, caso o usuário opte por alterar os campos de um formulário previamente cadastrado:
1.1) O usuário informa o nome de um formulário já cadastrado ou seleciona-lo na lista
1.2) O usuário manipula os campos, excluindo-os ou adicionando-os na lista de campos do formulário

Quadro 5 – Caso de uso UC04 – Cadastrar formulários

O quadro 6 apresenta a descrição do caso de uso UC05 – Gerar código.

Gerar Código
Sumário: O usuário efetua as operações referentes a geração de código do programa modelado na ferramenta. **Ator primário:** Usuário. **Pré-condições:** Deve ter sido cadastrado ao menos um formulário. **Fluxo principal: Gerar Código** 1) O usuário clica no botão “Gerar código”; 2) O sistema gera o código fonte da aplicação na pasta “C:\TCC\MagRegister\PDAMagRegister\src\”, pré-definida pela ferramenta.

Quadro 6 – Caso de uso UC05 – Gerar código

A figura 5 apresenta o segundo caso de uso, que refere-se ao processo de operação do programa gerado pela ferramenta.

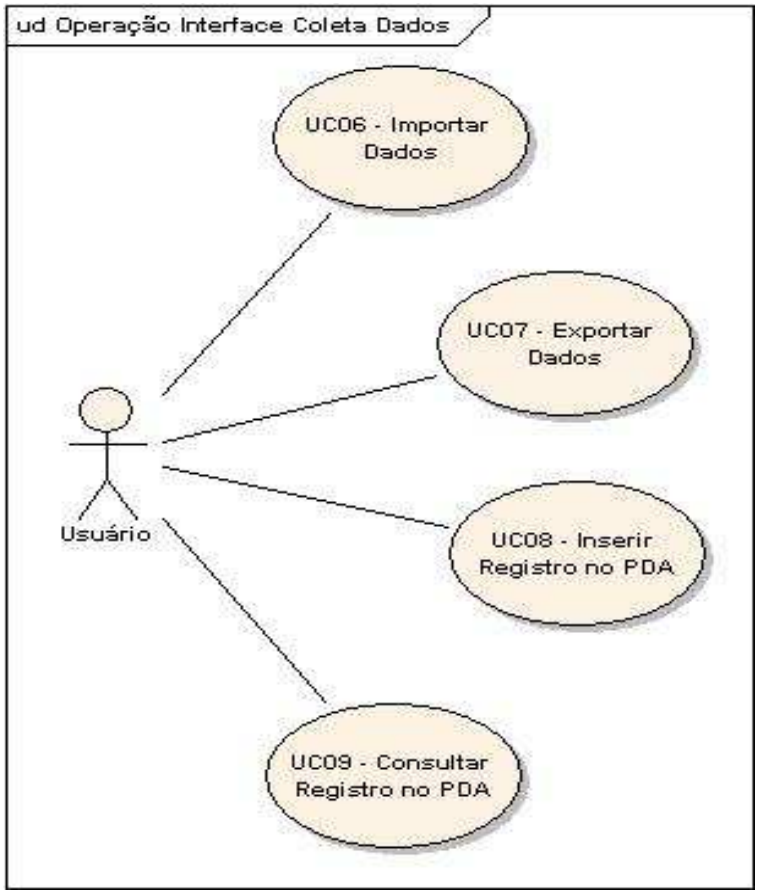


Figura 5 – Diagrama de casos de uso do programa gerado pela ferramenta.

O quadro 7 apresenta a descrição do caso de uso UC06 – Importar dados.

Importar dados

Sumário:
O usuário efetua a operação referente a importação de dados para a base de dados do programa instalado no dispositivo móvel, gerado pela ferramenta.

Ator primário:
Usuário.

Fluxo principal: Importar Dados
1) O usuário seleciona a opção de menu “Importar dados”;
2) O sistema importa os dados na base através de um arquivo XML recebido via requisição HTTP feita ao *servlet* disponibilizado no endereço eletrônico
“http://localhost:8084/MagServlet/MagRegisterServlet”

Fluxo de exceção: Validar informações
No passo 1, caso o sistema identifique problemas com a importação dos dados:
1.1) O sistema pode identificar problemas de conexão HTTP para importação dos dados e apresentar uma mensagem ao usuário informando tal situação.

Quadro 7 – Caso de uso UC06 – Importar dados

O quadro 8 apresenta a descrição do caso de uso UC07 – Exportar dados.

Exportar Dados

Sumário:
O usuário efetua a operação referente a exportação de dados da base de dados do programa instalado no dispositivo móvel, gerado pela ferramenta.

Ator primário:
Usuário.

Fluxo principal: Exportar Dados
1) O usuário seleciona a opção de menu “Exportar dados”;
2) O sistema exporta os dados através de uma requisição HTTP feita ao *servlet* disponibilizado no endereço eletrônico “http://localhost:8084/MagServlet/MagRegisterServlet”

Fluxo de exceção: Validar informações
No passo 1, caso o o sistema identifique problemas com a exportação dos dados
1.1) O sistema pode identificar problemas com a conexão HTTP para exportação dos dados e apresentar uma mensagem ao usuário informando tal situação.

Quadro 8 – Caso de uso UC07 – Exportar dados

O quadro 9 apresenta a descrição do caso de uso UC08 – Inserir registro no PDA.

Inserir registro no PDA

Sumário:
O usuário efetua a operação referente a inclusão de um registro na base de dados do programa instalado no dispositivo móvel, gerado pela ferramenta.

Ator primário:
Usuário.

Fluxo principal: Inserir novo registro no PDA
1) O usuário seleciona uma opção de menu referente a uma tela de cadastro gerada;
2) O usuário informa um valor que não está cadastrado na base de dados no campo referente a chave primária do leiaute;
3) O usuário preenche os demais campos referentes à tela de cadastro;
4) O usuário clica em “Gravar”;
5) O sistema grava o registro no banco de dados.

Fluxo de exceção: Validar informações
No passo 3, caso o o sistema identifique problemas com os dados informados;
3.1) O sistema pode identificar problemas com a máscara dos valores informados e apresentar uma mensagem informando tal situação;
3.2) Volta ao passo 3.

Quadro 9 – Caso de uso UC08 – Inserir registro no PDA

O quadro 10 apresenta a descrição do caso de uso UC09 – Consultar registro no PDA.

Consultar registro no PDA

Sumário:
O usuário efetua a operação referente a consulta de um registro na base de dados do programa instalado no dispositivo móvel, gerado pela ferramenta.

Ator primário:
Usuário.

Fluxo principal: Consultar registro do PDA
1) O usuário seleciona uma opção de menu referente a uma tela de cadastro gerada;
2) O usuário informa um valor que está previamente cadastrado na base de dados no campo referente a chave primária do leiaute, ou então seleciona o registro através do botão “Selecionar”;
3) O sistema apresenta os dados do registro na tela do dispositivo móvel.

Fluxo alternativo: Alterar registro do PDA
No passo 3, caso o o usuário opte por alterar os dados do registro carregado.
3.1) O usuário altera os campos do cadastro;
3.2) O usuário clica em “Gravar”;
3.3) O sistema grava as informações alteradas;
3.4) Retorna ao passo 2.

Fluxo alternativo: Excluir registro do PDA
No passo 3, caso o o usuário opte por excluir um registro carregado.
3.2) O usuário clica em “Excluir”;
3.3) O sistema exclui o registro;
3.4) Retorna ao passo 2.

Fluxo de exceção: Validar informações
No passo 3.3, caso o usuário deseja alterar um registro.
3.3.1) O sistema pode identificar problemas com a máscara dos valores informados e apresentar uma mensagem informando tal situação;
No passo 3.3, caso o usuário deseja excluir um registro.
3.3.1) O sistema pode identificar que o registro já está referenciado em outro cadastro e apresentar mensagem alertando sobre a situação.
3.3.2) Volta ao passo 2.

Quadro 10 – Caso de uso UC09 – Consultar registro no PDA

3.2.2 Diagrama de atividades

A figura 6 descreve o processo principal desempenhado pela ferramenta, desde a criação ou abertura do projeto, até a geração de código. São demonstradas as etapas necessárias para que se chegue ao processo final, que é a geração do programa para dispositivos móveis.

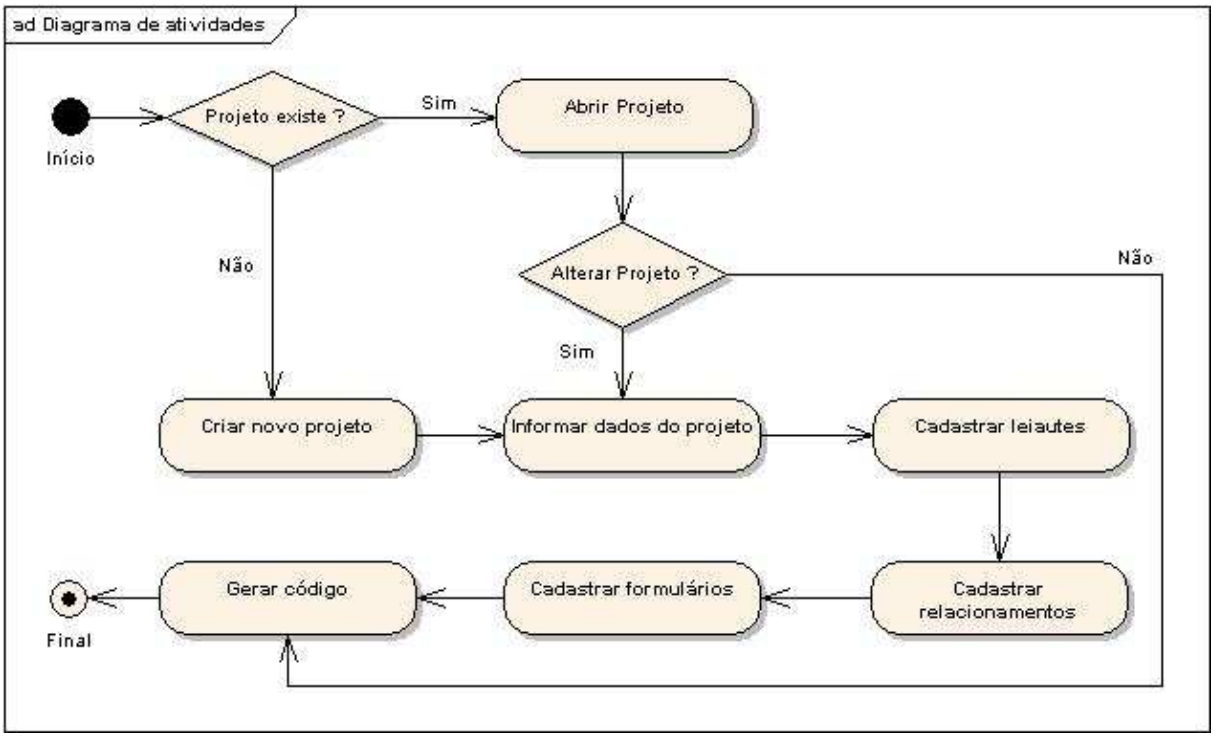


Figura 6 – Diagrama de atividades

Para que seja possível gerar uma aplicação de coleta de dados para dispositivos móveis, o usuário necessita inicialmente criar um projeto. Estando o projeto criado, opcionalmente podem ser informados os dados do cabeçalho, tais como nome do projeto, data de criação e observações. Após esta etapa, ficam disponíveis as demais páginas da tela de configuração, para que sejam criados os leiautes, relacionamentos e formulários. Somente após a criação e configuração dos formulários é que a aplicação permite que o código fonte da aplicação seja gerado.

3.2.3 Diagrama de classes

A figura 7 demonstra o diagrama das classes da camada de modelo da ferramenta, demonstrando as suas estruturas e as ligações estabelecidas entre elas.

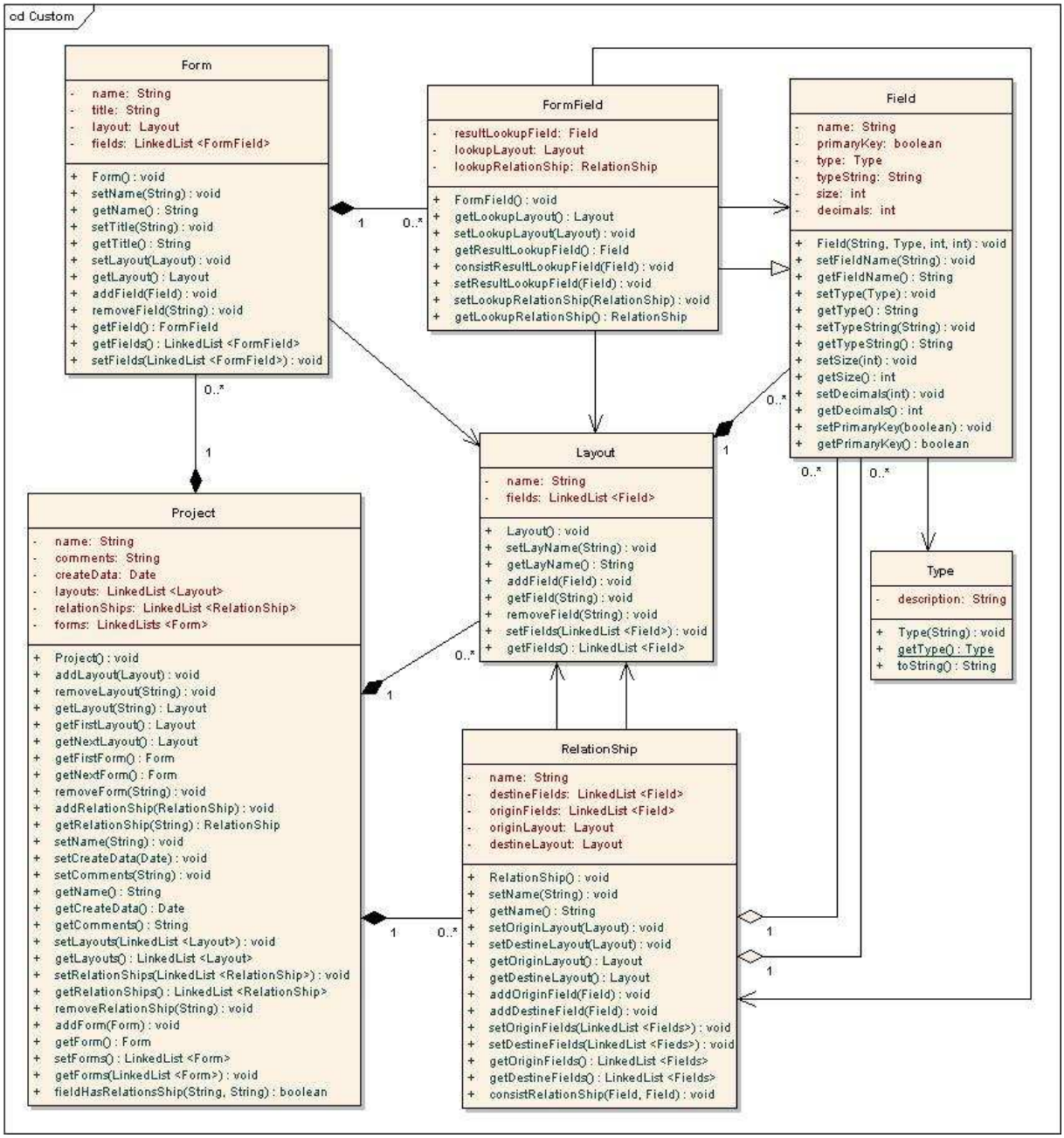


Figura 7 – Diagrama de classes

O diagrama de classes apresentado na figura 7 representa a estrutura de dados que armazena as principais informações que dão origem aos programas para dispositivos móveis que a ferramenta permite gerar. Todas as classes apresentadas estão definidas na camada de modelo e implementam a interface `java.io.Serializable`, permitindo que seus objetos possam ser salvos em disco.

No Quadro 11 é apresentado o detalhamento das classes relacionadas na figura 7, descrevendo o papel de cada classe na estrutura.

Classe	Objetivos
Type	Objetos instanciados a partir desta classe representam os tipos de dados dos campos que são adicionados aos leiautes do projeto. Esta classe possui uma propriedade que pode conter um dos seguintes valores: (Numérico, Alfa Numérico, Inteiro, Caractere).
Field	Objetos desta instanciados a partir desta classe representam os campos que são adicionados aos leiautes do projeto.
Layout	Objetos instanciados a partir desta classe representam os leiautes do projeto.
RelationShip	Objetos instanciados a partir desta classe representam os relacionamentos entre os leiautes. Através destes objetos é possível indicar leiautes origens e destino, bem como as ligações entre os seus campos.
FormField	Objetos instanciados a partir desta classe representam os campos que farão parte das interfaces de coletas de dados. Esta classe estende da classe Field e possui alguns atributos a mais, que indicam as ligações com outros leiautes para carga automática de valores.
Form	Objetos instanciados a partir desta classe representam as interfaces para coletas de dados que serão geradas para os dispositivos móveis. Através desta classe são parametrizados os dados das interfaces, tais como leiaute, título, campos dos leiautes que farão parte das interfaces e a ordenação da disposição dos campos na tela.
Project	Objetos instanciados a partir desta classe são responsáveis por armazenar os leiautes, relacionamentos e formulários que fazem parte do projeto. Esta é a classe principal da ferramenta, pois através dela é administrada toda a estrutura do os componentes do software que será gerado.

Quadro 11 - Descrição das classes da camada de modelo

3.3 IMPLEMENTAÇÃO

Nessa seção são descritas as ferramentas e técnicas utilizadas no desenvolvimento da ferramenta e é demonstrada a sua operacionalidade.

3.3.1 Técnicas e ferramentas utilizadas

Para o desenvolvimento desta ferramenta foi utilizada a linguagem Java, SDK versão 1.5.0. Para compilação, depuração e execução da ferramenta, foi utilizado o ambiente de desenvolvimento NetBeans 5.5. As telas da ferramenta foram desenvolvidas utilizando-se a biblioteca gráfica Swing e foi adotada a arquitetura *Model-View-Controller* (MVC). A ferramenta gera os códigos fontes das aplicações para dispositivos móveis mediante a utilização do motor de *templates* Velocity através da biblioteca `velocity-1.4.jar`.

Os programas gerados utilizam a plataforma JME e fazem acesso ao banco de dados DB2 Everyplace. Para a compilação destes programas também foi utilizado o ambiente de desenvolvimento NetBeans 5.5, através do acoplamento com o *plugin Mobility Pack*. Para a execução deste programas, foi necessário utilizar o simulador do sistema operacional Palm OS, o Palm OS Simulator Tungsten 2. A geração e interpretação dos dados no formato XML para a transmissão de dados entre PC e PDA, foi feita através da utilização da biblioteca KXmlParser.

O contêiner de aplicações web Apache Tomcat foi utilizado para a execução do *servlet*, criado para fazer a troca de dados entre dispositivos móveis e PCs através da transferência de arquivos no formato XML.

3.3.2 Implementação da ferramenta

Esta sessão apresenta a implementação das classes que compõem a ferramenta, mostrando trechos de código fonte e explicação sobre os mesmos. São apresentadas as implementações das camadas de visão, modelo e controle, assim como é detalhada a implementação da rotina que gera o código das aplicações para os dispositivos móveis e dos *templates*. Também é detalhada a estrutura dos programas que a ferramenta gera, demonstrando-se as principais classes e funcionalidades.

3.3.2.1 Gravação dos projetos

Os projetos criados pela ferramenta são gravados em arquivos no formato XML, sendo que cada projeto corresponderá a um arquivo. A gravação dos arquivos XML são feitas mediante serialização dos objetos criados na ferramenta. Para que esta serialização fosse possível ser feita, todas as classes que fazem parte da estrutura dos projetos implementam a interface `java.io.Serializable` da biblioteca do Java, e utilizam métodos das classes `java.beans.XMLDecoder` e `java.beans.XMLEncoder`. A classe principal que armazena todas as informações dos projetos é a `model.Project`.

Os objetos instanciados a partir desta classe, armazenam lista de outros objetos que fazem parte do projeto, tais como objetos de leiautes, relacionamentos e formulários, que respectivamente são instanciados a partir das classes `model.Layout`, `model.RelationShip` e `model.Form`. A figura 8 apresenta o conteúdo de um arquivo XML com *tags* minimizadas, gravado a partir de um projeto criado na ferramenta.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <java version="1.5.0_11" class="java.beans.XMLDecoder">
 - <object class="model.Project">
  - <void property="comments">
      <string>Este projeto tem por finalidade gera um programa para coleta de dados</string>
    </void>
  - <void property="createData">
    - <object class="java.util.Date">
        <long>1179889200000</long>
      </object>
    </void>
  + <void property="forms">
  - <void property="layouts">
    - <void method="add">
        <object idref="Layout0" />
      </void>
    - <void method="add">
        <object idref="Layout2" />
      </void>
    + <void method="add">
    - <void method="add">
        <object idref="Layout3" />
      </void>
    + <void method="add">
    </void>
  - <void property="name">
      <string>Projeto CONTROLE INVENTÁRIOS ESTOQUES</string>
    </void>
  + <void property="relationShips">
   </object>
</java>
```

Figura 8 – Exemplo de XML gerado a partir de um projeto criado pela ferramenta

3.3.2.2 Implementação da camada de modelo

As classes da camada de modelo estão todas agrupadas no pacote `model` e referem-se aos objetos que armazenam a estrutura dos projetos de softwares para dispositivos móveis. As classes que fazem parte desta camada são `Type`, `Field`, `Layout`, `RelationShip`, `Project`, `FormField` e `Form`.

A classe `model.Type` é uma classe do tipo enumeração e não pode ser instanciada. O método construtor é utilizado apenas pela própria classe no momento da adição dos elementos na lista. Ela foi utilizada para a criação de uma lista de *Strings* que contém os tipos de dados disponíveis para os campos que são criados nos leiautes dos projetos. Foram afixadas nesta classe uma lista de *Strings* contendo o seguinte conteúdo: "Numérico", "Alfa numérico", "Caractere", e "Inteiro". O quadro 12 demonstra os principais métodos desta classe.

```
package model;

/**
 * Enumeração dos tipos de dados.
 * @author Gilson Chequeto
 * @version 08/03/2007
 */
public enum Type {
    // Possibilidades de privilégios de objeto
    numeric("Numérico"),
    alfanumeric("Alfa numérico"),
    character("Caractere"),
    integer("Inteiro");

    /**
     * Descrição do privilégio de objeto.
     */
    private final String description;

    /**
     * Construtor da enumeração Type.
     */
    private Type(String description) {
        this.description = description;
    }

    .
    .

    /**
     * Retorna o tipo de dados de acordo com a String
     */
    public static Type getType(String strAux) {
        if  (strAux.equalsIgnoreCase("Numérico")) {
            return Type.valueOf("numeric");
        } else {
            if  (strAux.equalsIgnoreCase("Alfa numérico")) {
                return Type.valueOf("alfanumeric");
            } else {
                if   (strAux.equalsIgnoreCase("Caractere")) {
                    return Type.valueOf("character");
                } else return Type.valueOf("integer");
            }
        }
    }
}
```

Quadro 12 – Implementação da eumeração model.Type

A classe `model.Field` implementa também a interface `controller.FieldInterface`, além da interface `java.io.Serializable`. Isto foi feito para que durante a interação entre a camada de visão e a camada de modelo, através da camada controladora, fosse possível serem feitas trocas de dados e invocação de métodos, preservando-se a integridade dos dados e a garantia de que métodos não permitidos sejam invocados em momentos impróprios. A interface `controller.FieldInterface` declara apenas métodos da classe `model.Field` que podem ser acessados pela camada de visão, que basicamente são os métodos de acessos. Sendo assim, os métodos da camada de controle que são chamados pela camada de visão para adição de campos no leiaute, sempre retornarão para a camada de visão objetos do tipo `controller.FieldInterface`, ao invés de objetos do tipo

`model.Field`.

Conforme já descrito na seção referente a especificação, objetos da classe `model.Layout` são responsáveis por armazenar as estruturas de dados (tabelas) que serão criadas nos dispositivos móveis. Estes objetos armazenam uma lista de objetos da classe `model.Field` através de uma lista do tipo `java.util.LinkedList`. O quadro 13 apresenta os principais métodos desta classe.

```
package model;

import java.io.Serializable;
import java.util.LinkedList;

public class Layout implements Serializable {


    private String name;
    private LinkedList <Field> fields;

    /* Construtor da classe */
    public Layout() {
        super();
        fields = new LinkedList <Field>();
    }

    /* Método que adiciona os campos no leiaute */
    public void addField(Field field) throws Exception {
        for (int i = 0; i < this.fields.size(); i++) {
            if
       (this.fields.get(i).getFieldName().equalsIgnoreCase(field.getFieldName())) {
                throw new Exception("Campo já cadastrado no layout !");
            }
        }
        fields.add(field);
    }

    /* Método que remove os campos do leiaute */
    public boolean removeField(String fieldName) {
        for (int i = 0; i < this.fields.size(); i++)
            if      (this.fields.get(i).getFieldName().equalsIgnoreCase(fieldName))
{
            this.fields.remove(i);
            return true;
            }
        return false;
    }

    /* Método que resgata os campos do leiaute */
    public Field getField(String fieldName) {
        Field fieldAux = null;
        for (int i = 0; i < this.fields.size(); i++) {
            fieldAux = this.fields.get(i);
            if      (fieldAux.getFieldName().equalsIgnoreCase(fieldName)) {
                return fieldAux;
            }
        }
        return fieldAux;
    }
.
.
.
}
```

Quadro 13 – Implementação da classe model.Layout

As demais classes da camada de modelo foram implementadas nos mesmos moldes da classe `model.Layout`, ou seja, possuem métodos para atribuir e resgatar os valores dos atributos, lista de objetos filhos e métodos para manipulação destes objetos.

3.3.2.3 Implementação da camada de controle

O controlador implementado na classe `controller.Controller` tem a responsabilidade de instanciar objetos do modelo, inicializar suas propriedades e retornar as mesmas para a camada de visão nas formas primitivas de dados ou até mesmo através de interfaces de objetos que disponibilizam apenas métodos que podem ser visíveis na camada de visão. Nos métodos em que a troca de dados é feita através de tipos primitivos, normalmente um objeto das classse `java.util.HashMap` ou `java.util.LinkedList` é passado como parâmetro da camada de visão para a camada de controle, ou retornado como resultado de um método. No Apêndice A é apresentada a implementação classe controladora `controller.Controller` e seus principais métodos.

3.3.2.4 Implementação da camada de visão

A camada de visão é responsável pela interação entre os usuários e a ferramenta. A camada de visão possui apenas uma classe responsável por exibir toda a apresentação gráfica da ferramenta, mesmo porque a ferramenta é formada por apenas uma tela principal para manipulação dos projetos. Esta tela está divida em diversas páginas, sendo que cada uma delas é destinada para a manipulação dos principais objetos do projeto. Esta única classe `view.MainInterface` interage diretamente com a classe controlador `controller.Controller` passando e resgatando parâmetros. Pode-se citar como exemplo a abertura de um projeto. Ao abrir um projeto, o caminho do arquivo XML a ser aberto é passado como parâmetro para o controlador, que abre o projeto e retorna um objeto da classe `java.util.HashMap` contendo os atributos do projeto aberto. A camada de visão utiliza os valores retornados e carrega os componentes da tela. O Quadro 14 mostra esta operação implementada na classe `view.MainInterface`.

```
    .
    .
    .
   /** evento de um botão na camada de visão, responsável por abrir um
       projeto */

   private void jbAbrirActionPerformed(java.awt.event.ActionEvent evt) {
       try {
           JFileChooser chooser = new JFileChooser();
           MyFileFilter fileFilter = new MyFileFilter();
           fileFilter.addExtension("xml");
           fileFilter.setDescription("Arquivos XML");
           chooser.setFileFilter(fileFilter);
           chooser.showOpenDialog(this);
           String path = chooser.getSelectedFile().getPath();
           if  (!(path.equals(" "))) {
               HashMap <String, String> values = iController.getProject(path);
               this.setName(values.get(iController.PROJECT_NAME));
               SimpleDateFormat simpleFormat = new SimpleDateFormat("dd/MM/yyyy");

               this.setCreateData(simpleFormat.parse(
                                  values.get(iController.PROJECT_CREATEDATA)));
               this.setComments(values.get(iController.PROJECT_COMMENTS));
               this.setAddingItems(true);
               this.chargeJcbLayoutName();
               this.setAddingItems(false);
               this.setLayName(values.get(iController.PROJECT_FIRST_LAYOUT));
               this.chargeLayout(this.getLayName());
               this.enableDefaultControls(true);
           }

       } catch (Exception e) {
           JOptionPane.showMessageDialog(null, "Não foi possível abrir o projeto:
           "+e.getMessage(), "Erro", JOptionPane.ERROR_MESSAGE);
       }

   }
   .
   .
   .
```

Quadro 14 – Implementação da abertura de um projeto na camada de visão view.MainInterface

3.3.2.5 Implementação dos *templates* e geração de código

A estrutura dos códigos fontes das aplicações geradas por esta ferramenta, basicamente estão divididas em cinco tipos de classes:

a) Classe do *MIDLet* da aplicação;

b) Classe do menu principal da aplicação;

c) Classes de objetos relacionados às interfaces de coletas de dados;

d) Classes de objetos relacionados aos leiautes;

e) Classe de conexão com o *servlet* para troca de informações.

A classe do *MIDLet* da aplicação é responsável por conectar-se com o banco de dados, criar as tabelas na primeira execução do aplicativo e criar os formulários que fazem parte da

aplicação a medida em que eles vão sendo solicitados. Esta classe possui métodos para todas as operações acima citadas e o objeto instanciado através dela é passado como parâmetro para todos os formulários da aplicação, para que os mesmos possam fazer uso da conexão com o banco de dados, assim como acessar um método para exibição de mensagens aos usuários, que também está implementado nesta classe. O quadro 15 apresenta parte da definição do template que é utilizado para a geração da classe do *MIDLet* e o quadro 16 apresenta os principais métodos do código gerado.

```
import java.sql.Connection;
.
.

public class MainMidlet extends MIDlet implements CommandListener {

    /** Creates a new instance of MainMidlet */
    public MainMidlet() {
        initialize();
    }

    #foreach($form in $project.getForms())
    private Frm$convert.convertUpperFirst($form.getName())
    frm$convert.convertUpperFirst($form.getName());
    #end
    .
    .
    private void createTables() {
        try {
            #foreach($form in $project.getForms())
            #set($formName = $convert.convertUpperFirst($form.getName()))
            #set($layout = $project.getForm($formName).getLayout())
            #set($stringCreateTable = "$convert.aspas()CREATE TABLE
                 $convert.convertUpperAll($layout.getLayName()) (")


            .
            .
            #end
            showAlert("Tabelas criadas com sucesso.");
        } catch (SQLException ex) {
            showAlert("Tabelas inicializadas.");
        }
    }

    /* Faz a conexão com o banco de dados no Palm */
    private void conecta(){
        try {
            // Carrega driver do DB2
            Class.forName("com.ibm.db2e.jdbc.DB2eDriver");
            // Faz conexao com banco TCCJ2ME
            con = DriverManager.getConnection("jdbc:db2e:TCCJ2ME");
            st = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                    ResultSet.CONCUR_UPDATABLE);
            // Verifica se há a necessidade de criar as tabelas
            createTables();
        } catch (Exception ex) {
            showAlert("Não foi possível conectar no banco de dados.");
        }
    }
```

```
public static String generateXMLFromData(MainMidlet midlet) {

        StringBuffer xmlString = new StringBuffer();

        xmlString.append("<?xml version=\"1.0\" encoding=\"UTF-8\" ?>");
        xmlString.append("<register>");

        ResultSet rs;
        try {
            rs = midlet.st.executeQuery("SELECT * FROM
                  ${convert.convertUpperAll(${layout.getLayName()})}");
            while (rs.next()) {
                xmlString.append("   <record>\r\n");

                #foreach($field in $layout.getFields())
                #set($fieldName = $convert.convertUpperFirst(
                                                        $field.getFieldName()))
                xmlString.append("      <field>\r\n");
                xmlString.append("         <name>${fieldName}</name>\r\n");
                xmlString.append("         <value>"+rs.getString
                                           (rs.findColumn("${fieldName}"))+
                                            "</value>\r\n");
                xmlString.append("      </field>\r\n");
                #end

                xmlString.append("   </record>\r\n");
            }

            xmlString.append("</register>");

        } catch (SQLException ex) {
            midlet.showAlert(ex.getMessage());
        }

        return xmlString.toString();
    }

}
```

Quadro 15 – Template para geração do *MIDLet*

```
import java.sql.Connection;
.
.
.
public class MainMidlet extends MIDlet implements CommandListener {

    public MainMidlet() {
        initialize();
    }

    private FrmFiliais frmFiliais;
    .
    .
    private FrmInventarios frmInventarios;

    private MainMenu mainMenu;
    private Displayable currentForm;
    public Connection con;
    public Statement st;
    Command cmdAlertOk;

    /** método responsável por setar a tela de menu como sendo a tela principal da
        Aplicação */
    private void initialize() {
        setCurrentForm(get_mainMenu());
    }
    private void createTables() {
        try {
             .
             .
            st.executeUpdate("CREATE TABLE INVENTARIOS (CODIGO DECIMAL(5,1),
                              PRODUTO VARCHAR(10), FILIAL DECIMAL(5,1),
                              DEPARTAMENTO DECIMAL(5,1), DEPOSITO DECIMAL(5,1),
                              QUANTIDADE DECIMAL(7,2))");
            showAlert("Tabelas criadas com sucesso.");
        } catch (SQLException ex) {
            showAlert("Tabelas inicializadas.");
        }
    }

/** Método responsável por fazer a conexão com o banco de dados no Palm */
    private void conecta(){
        try {
            // Carrega driver do DB2
            Class.forName("com.ibm.db2e.jdbc.DB2eDriver");
            // Faz conexao com banco TCCJ2ME
            con = DriverManager.getConnection("jdbc:db2e:TCCJ2ME");
            st = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                    ResultSet.CONCUR_UPDATABLE);
            // Verifica se há a necessidade de criar as tabelas
            createTables();
        } catch (Exception ex) {
            showAlert("Não foi possível conectar no banco de dados.");
        }
    }

    public void commandAction(Command command, Displayable displayable) {
        if  ((command == cmdAlertOk)) {
            setCurrentForm(getCurrentForm());
        }
    }

    public static String generateXMLFromData(MainMidlet midlet) {

        StringBuffer xmlString = new StringBuffer();

        xmlString.append("<?xml version=\"1.0\" encoding=\"UTF-8\" ?>");
        xmlString.append("<register>");
```

```
        ResultSet rs;
        try {
            rs = midlet.st.executeQuery("SELECT * FROM DEPOSITOS");
            while (rs.next()) {
                xmlString.append("   <record>\r\n");
                .
                .
                xmlString.append("   </record>\r\n");
            }

            xmlString.append("</register>");

        } catch (SQLException ex) {
            midlet.showAlert(ex.getMessage());
        }

        return xmlString.toString();
    }

}
```

Quadro 16 - Implementação do *MIDLet*

O quadro 17 apresenta parte do *template* utilizado para gerar o menu principal da aplicação e o quadro 18 apresenta para do código gerado.

```
import javax.microedition.lcdui.Command;
.
.
import javax.microedition.midlet.MIDlet;


public class MainMenu extends Form implements CommandListener{

    #foreach($form in $project.getForms())
    private Command cmd$convert.convertUpperFirst($form.getName());
    #end

    private Command cmdDivision1;
    .
    .
    private Command cmdSair;

    private MainMidlet midlet;
    private Display display; // Formsr


    public MainMenu(MainMidlet mainMidlet) {
        //Monta tela
        super("Escolha a opção");

        display = mainMidlet.getDisplay();
        midlet = mainMidlet;

        //Adiciona comandos do formulário principal
        #foreach($form in $project.getForms())
        cmd$convert.convertUpperFirst($form.getName()) = new
        Command("$convert.convertUpperFirst($form.getName())", Command.ITEM, 1);
        #end
        cmdDivision1 = new Command("---------------", Command.ITEM, 1);
        cmdDivision2 = new Command("---------------", Command.ITEM, 1);
        cmdDropTables = new Command("Excluir Tabelas", Command.ITEM, 1);
        cmdImportar = new Command("Importar Dados", Command.ITEM, 1);
        cmdExportar = new Command("Exportar Dados", Command.ITEM, 1);
        cmdSair = new Command("Sair", Command.ITEM,1);

        #foreach($form in $project.getForms())
            addCommand(cmd$convert.convertUpperFirst($form.getName()));
        #end
            .
            .
            addCommand(cmdSair);
            setCommandListener(this);
    }

    public void commandAction(Command command, Displayable displayable) {

        #foreach($form in $project.getForms())
        if  (command == cmd$convert.convertUpperFirst($form.getName())) {
midlet.setCurrentForm(midlet.get_frm$convert.convertUpperFirst($form.getName())());
        }
        #end
        .
        .
        if  ((command == cmdSair)) {
            midlet.exitMIDlet();
        }
    }
}
```

Quadro 17 – Template para geração do menu principal

```
import javax.microedition.lcdui.Command;
.
.
import javax.microedition.midlet.MIDlet;


public class MainMenu extends Form implements CommandListener{


    private Command cmdDivision1;
    .
    .
    private Command cmdSair;

    private MainMidlet midlet;
    private Display display; // Formsr


    public MainMenu(MainMidlet mainMidlet) {
        //Monta tela
        super("Escolha a opção");

        display = mainMidlet.getDisplay();
        midlet = mainMidlet;

        //Adiciona comandos do formulário principal
                cmdFiliais = new Command("Filiais", Command.ITEM, 1);
                cmdDepositos = new Command("Depositos", Command.ITEM, 1);
                cmdDepartamentos = new Command("Departamentos", Command.ITEM, 1);
                cmdProdutos = new Command("Produtos", Command.ITEM, 1);
                cmdInventarios = new Command("Inventarios", Command.ITEM, 1);
                cmdDivision1 = new Command("---------------", Command.ITEM, 1);

        .
        .
        cmdExportar = new Command("Exportar Dados", Command.ITEM, 1);
        cmdSair = new Command("Sair", Command.ITEM,1);

        addCommand(cmdSair);
        .
        .
    }

    public void commandAction(Command command, Displayable displayable) {

         .
         .
         if  (command == cmdInventarios) {
             midlet.setCurrentForm(midlet.get_frmInventarios());
         }
         .
         .
         if  (command == cmdDropTables) {
             midlet.dropTables();
         }
         if  ((command == cmdSair)) {
             midlet.exitMIDlet();
         }
    }
}
```

Quadro 18 – Implementação do menu principal

As classes de objetos relacionados às interfaces de coletas de dados referem-se a

classes que representam as interfaces para coletas de dados que são disponibilizadas na aplicação para o dispositivo móvel. Estas classes são geradas a partir dos formulários cadastrados na ferramenta, sendo que cada formulário cadastrado representa uma classe gerada. Possuem implementações para a edição dos dados, bem como para interação com objetos relacionados aos leiautes, efetuando a troca de informações entre os componentes visuais e os objetos que fazem a persistência dos dados. O quadro 19 apresenta parte da implementação do *template* e o quadro 20 apresenta parte da implementação do código gerado.

```
import java.sql.Connection;
.
.
import javax.microedition.lcdui.Image;


public class Frm$convert.convertUpperFirst($form.getName()) extends Form
implements CommandListener {
    private Command cmdVoltar;
    private Command cmdSalvar;
    private Command cmdSelecionar;

    Command cmdSelectionBack;
    Command cmdSelectionOk;

    private MainMidlet midlet;
    private Displayable lastForm;

    #foreach($field in $form.getFields())
    #if (!($project.getRelationedLayout(${form.getLayout().
          getLayName()},${field.getFieldName()}).equals(" ")))
    private ChoiceGroup cg$convert.convertUpperFirst($field.getFieldName());
    #else private TextField txt$convert.convertUpperFirst($field.getFieldName());
    #end
    #end


    #set($layoutName = $convert.convertUpperFirst($form.getLayout().getLayName()))

    //Lista para consultar os registros
    private List searchList = new List("Selecione um registro:",List.EXCLUSIVE);

    public Frm$convert.convertUpperFirst($form.getName())(MainMidlet mainMidlet,
                                                          Displayable lastFrm) {
        //Monta tela
        super("$convert.convertUpperFirst($form.getTitle())");

        midlet = mainMidlet;
        lastForm = lastFrm;

        ResultSet rs;

        #foreach($field in $form.getFields())
        #set($fieldName = $convert.convertUpperFirst($field.getFieldName()))
        #set($relationedLayout = $project.getRelationedLayout(${form.getLayout().
             getLayName()},${field.getFieldName()}))
        #set($relationedField = $project.
                                getRelationedField(${form.getLayout().
                                                    getLayName()},
                                                    ${field.getFieldName()}))
        #if (!($relationedLayout.equals(" ")))
        cg$fieldName = new ChoiceGroup("$convert.
                           convertUpperFirst($field.getFieldName()):",
             Choice.POPUP, new String[0], new Image[0]);
        try {
            rs = midlet.st.executeQuery("SELECT ${relationedField}
                                         FROM ${relationedLayout} ORDER BY
                                         ${relationedField}");
            while (rs.next()) {
                cg${fieldName}.append(rs.getString(1),null);
            }
        } catch (SQLException ex) {
            midlet.showAlert(ex.getMessage());
        }
        append(cg$fieldName);
        #else
        txt$fieldName = new TextField("$convert.
                         convertUpperFirst($field.getFieldName()):",
```

```
                        "", $field.getSize(), TextField.ANY);
        append(txt$fieldName);
        #end
        #end

        //Adiciona comandos do formulário principal
        cmdVoltar = new Command("Voltar", Command.BACK,1);
        .
        .
        setCommandListener(this);

        //Adiciona comandos da tela de pesquisa
        cmdSelectionBack = new Command("Voltar", Command.BACK,1);
        cmdSelectionOk = new Command("OK", Command.ITEM,1);
        searchList.addCommand(cmdSelectionBack);
        searchList.addCommand(cmdSelectionOk);
        searchList.setCommandListener(this);
    }

    private void saveObj_$layoutName() {
        try {
            $layoutName obj$layoutName = new $layoutName();
            #foreach($field in $form.getFields())
            #set($fieldName = $convert.convertUpperFirst($field.getFieldName()))
            #if (!($project.getRelationedLayout(${form.getLayout().getLayName()},
                   ${field.getFieldName()}).equals(" ")))
            obj${layoutName}.set${fieldName}(cg${fieldName}.
                             getString(cg${fieldName}.getSelectedIndex()));
            #else
            obj${layoutName}.set${fieldName}(txt${fieldName}.getString());
            #end
            #end
            obj${layoutName}.save(midlet.st);
        } catch (Exception ex) {
            midlet.showAlert("Não foi possível gravar os dados: verifique a
                              máscara correta dos campos.");
        }
    }

    public void commandAction(Command command, Displayable displayable) {

        if  (command == cmdSelecionar) {
            chargeRegisterSearch();
        }
        .
        .
        if (command == cmdSalvar) {
            saveObj_$layoutName();
        }
    }
}
```

Quadro 19 – Template para geração das classes relacionadas às interfaces para coletas de dados

```
import java.sql.Connection;
.
.
import javax.microedition.lcdui.Image;


public class FrmDepositos extends Form implements CommandListener {
    .
    .
    private MainMidlet midlet;
    private Displayable lastForm;

    private TextField txtCodigo;
    private TextField txtNome;



    //Lista para consultar os registros
    private List searchList = new List("Selecione um registro:",List.EXCLUSIVE);

    public FrmDepositos(MainMidlet mainMidlet, Displayable lastFrm) {
        //Monta tela
        super("Cadastro de depósitos");

        midlet = mainMidlet;
        lastForm = lastFrm;

        ResultSet rs;

        txtCodigo = new TextField("Codigo:", "", 5, TextField.ANY);
        append(txtCodigo);
        .
        .
        addCommand(cmdSelecionar);
        setCommandListener(this);

        //Adiciona comandos da tela de pesquisa
        cmdSelectionBack = new Command("Voltar", Command.BACK,1);
        .
        .
    }

    .
    .
    private void saveObj_Depositos() {
        try {
            Depositos objDepositos = new Depositos();
            objDepositos.setCodigo(txtCodigo.getString());
            objDepositos.setNome(txtNome.getString());
            objDepositos.save(midlet.st);
        } catch (Exception ex) {
            midlet.showAlert("Não foi possível gravar os dados: verifique a máscara
                              correta dos campos.");
        }
    }

}
.
.
.
```

Quadro 20 – Implementação de classes relacionada à uma interface para coletas de dados

As classes de objetos relacionados aos leiautes, referem-se a classes que são criadas para armazenar os dados de um registro de um leiaute. A ferramenta gera uma classe para cada leiaute cadastrado no projeto. Basicamente estas classes possuem atributos para armazenar os valores dos campos do leiaute, assim como métodos para atribuir e resgatar os atributos, além de um método responsável por fazer a inserção do registro no banco de dados. Para a geração destas classes, assim como para cada uma das categorias de classes da aplicação gerado, foi criado um *template* específico. O quadro 21 apresenta parte da definição do *template* que é utilizado para a geração do código da classe de objetos relacionados aos leiautes e o quadro 22 apresenta um exemplo de código gerado.

```
import java.sql.SQLException;
import java.sql.Statement;
public class $convert.convertUpperFirst($layout.getLayName()) {

       ##Declaração dos atributos da classe##
       #foreach($field in $layout.getFields())
       private $convert.convertType($field.getTypeString())
        $convert.convertLowerAll($field.getFieldName());
       #end

       ##Declaração do primeiro construtor da classe##
       public $convert.convertUpperFirst($layout.getLayName())() {
             super();
       }
      ##Declaração do segundo construtor da classe##
       #set($counter = 0)
       public $convert.convertUpperFirst($layout.getLayName())(
       #foreach($field in $layout.getFields()) #set($counter = $counter + 1)
          #if ($counter == $layout.getFields().size())
              $convert.convertType($field.getTypeString())
              $convert.convertLowerAll($field.getFieldName())) {
         #else
              $convert.convertType($field.getType())
              $convert.convertLowerAll($field.getFieldName()),
         #end #end #set($counter = 0)
              super();

         #foreach($field in $layout.getFields())
            this.$convert.convertLowerAll($field.getFieldName()) =
            $convert.convertLowerAll($field.getFieldName());
         #end
    }
    ##Implementação dos métodos sets ##
    #foreach($field in $layout.getFields())
    public void set$convert.convertUpperFirst($field.getFieldName())(String
        $convert.convertLowerAll($field.getFieldName())) {
        #set($fieldType = "$field.getTypeString()")
        #if ($fieldType.equalsIgnoreCase("Numérico"))
            this.$convert.convertLowerAll($field.getFieldName()) =
            Float.parseFloat($convert.convertLowerAll($field.getFieldName()));
        #end
    }
  .
  .
  .
  .
```

```
    ##Implementação do método para gravar o registro ##
    public void save(Statement st) throws SQLException {
       #set($stringInsert = "$convert.aspas()INSERT INTO
       $convert.convertUpperFirst($layout.getLayName()) VALUES ($convert.aspas()")
       #set($counter = 0)
       #foreach($field in $layout.getFields()) #set($counter = $counter + 1)
       #set($fieldType = "$field.getTypeString()")

       #if($counter != 1)
       #set($campo = $convert.concat("+$convert.aspas(),$convert.aspas()",$campo))
       #end
       #if($counter == $layout.getFields().size())
       #set($campo = $convert.concat($campo,"+"))
       #set($campo = $convert.concat($campo,$convert.aspas()))
       #set($campo = $convert.concat($campo,")"))
       #set($campo = $convert.concat($campo,$convert.aspas()))
       #end
       #set($stringInsert = $convert.concat($stringInsert,$campo))
       #end
       st.executeUpdate($stringInsert);
    }
}
```

Quadro 21 – Template para geração das classes relacionadas aos leiautes

```
import java.sql.SQLException;
import java.sql.Statement;
public class Produtos {

    private String codigo;
    .
    .
    private double quantidade;

    public Produtos() {
        super();
    }
    public Produtos(string codigo,double filial,double departamento,
            double deposito,
            String descricao,
            double quantidade) {
        super();
        this.codigo = codigo;
        .
        .
        this.quantidade = quantidade;
    }

    public void setCodigo(String codigo) {
        this.codigo = codigo;
    }

    public String getCodigo() {
        return this.codigo;
    }

    .
    .
    public void save(Statement st) throws SQLException {
        st.executeUpdate("INSERT INTO Produtos VALUES ("+ "'" +
                         this.getCodigo()+"'"+","+this.getFilial()+","+
                         this.getDepartamento()+","+this.getDeposito()+","+ "'" +
                         this.getDescricao()+"'"+","+this.getQuantidade()+")");
    }
}
```

Quadro 22 – Exemplo de classe de objetos relacionados aos leiautes

As demais classes das aplicações que podem ser geradas, seguem o mesmo princípio e não utilizam recursos diferentes dos apresentado nas classes acima, exceto a classe de conexão com *servlet* para importação de dados, que possue um recurso para conexão via protocolo HTTP. Esta classe implementa a conexão com um *servlet* e executa um *parse* do arquivo XML enviado pelo *servlet.*através da biblioteca KXmlParser, que é um *parser* simplificado de XML que contém métodos para leitura e verificação da sintaxe de dados. O Apêndice B mostra as suas principais funcionalidades. Um único *servlet* foi criado para atender as requisições de importação e exportação de dados. O software no dispositivo móvel envia uma requisição HTTP, passando parâmetros indicando qual o tipo da operação (importação/exportação) e qual a tabela a ser importada ou exportada. O funcionamento do processo de exportação se dá através da varredura dos dados nas tabelas no banco de dados para a montagem de um arquivo XML e envio para o *servlet.* Cada classe que representa um modelo de leiaute possui a implementação de um método para a montagem do XML para exportação, conforme demonstrado no quadro 15. O quadro 23 apresenta o *servlet* responsável por receber as requisições de importação e exportação.

```
//imports omitidos


public class MagRegisterServlet extends HttpServlet {
    protected void processRequest(HttpServletRequest request,
                                  HttpServletResponse response)
                                  throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        if  ((request.getParameter("operation") != null) &&
            (request.getParameter("operation").equalsIgnoreCase("importOnPDA"))) {
            FileReader fileReader = new
            FileReader("C:/tcc/MagRegister/ExpXMLGenerator/
                        XMLExample/"+request.getParameter("tableName")+".xml");
            BufferedReader buff = new BufferedReader(fileReader);

            response.setContentType("text/xml");
            String linha = null;
            while((linha = buff.readLine()) !=null){
                out.println(linha);
            }
            out.close();
        } else {
            boolean receive = false;
            response.setContentType("text/plain");
            BufferedReader br = request.getReader();
            String buf;
            FileWriter fr = new FileWriter("C:/tcc/MagRegister/
                                           ExpXMLGenerator/
                                           XMLExample/"+
                                       request.getParameter("tableName")+".xml");
            while ((buf = br.readLine()) != null) {
                receive = true;
                fr.write(buf);
            }
            fr.flush();
            fr.close();
    }
}
//outros métodos omitidos
```

Quadro 23 – Implementação do servlet para troca de dados

3.3.3 Operacionalidade da implementação

Nesta sessão é apresentada a operacionalidade da ferramenta, através da criação de um projeto para geração de um programa de coleta de dados para inventário de estoques. São criados leiautes de filiais, departamentos, depósitos, produtos e inventários, bem como suas respectivas telas para entrada de dados. São demonstrados todos os passos para a criação do projeto, desde a sua elaboração, passando pela geração de código e demonstração do programa gerado pela ferramenta.

Inicialmente deve ser criado um projeto e informados os dados do cabeçalho, conforme apresentado na Figura 9.

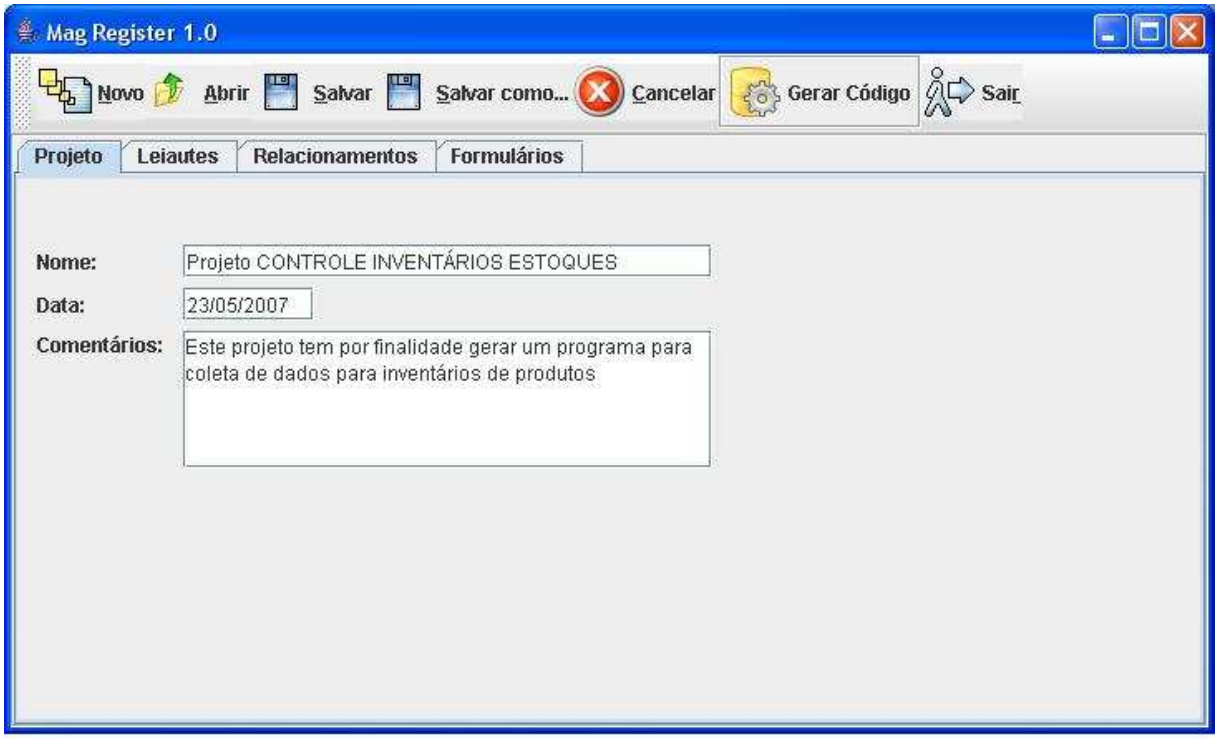


Figura 9 – Tela principal (Dados do cabeçalho)

Em seguida, devem ser cadastrados os leiautes que farão parte do projeto. Para isto, devem ser informados os nomes do leiautes e adicionados os campos que fazem parte, informando se os mesmos fazem parte da chave primária, o seus nomes, tipos, tamanhos e quantidade de decimais. A Figura 10 ilustra este passo.

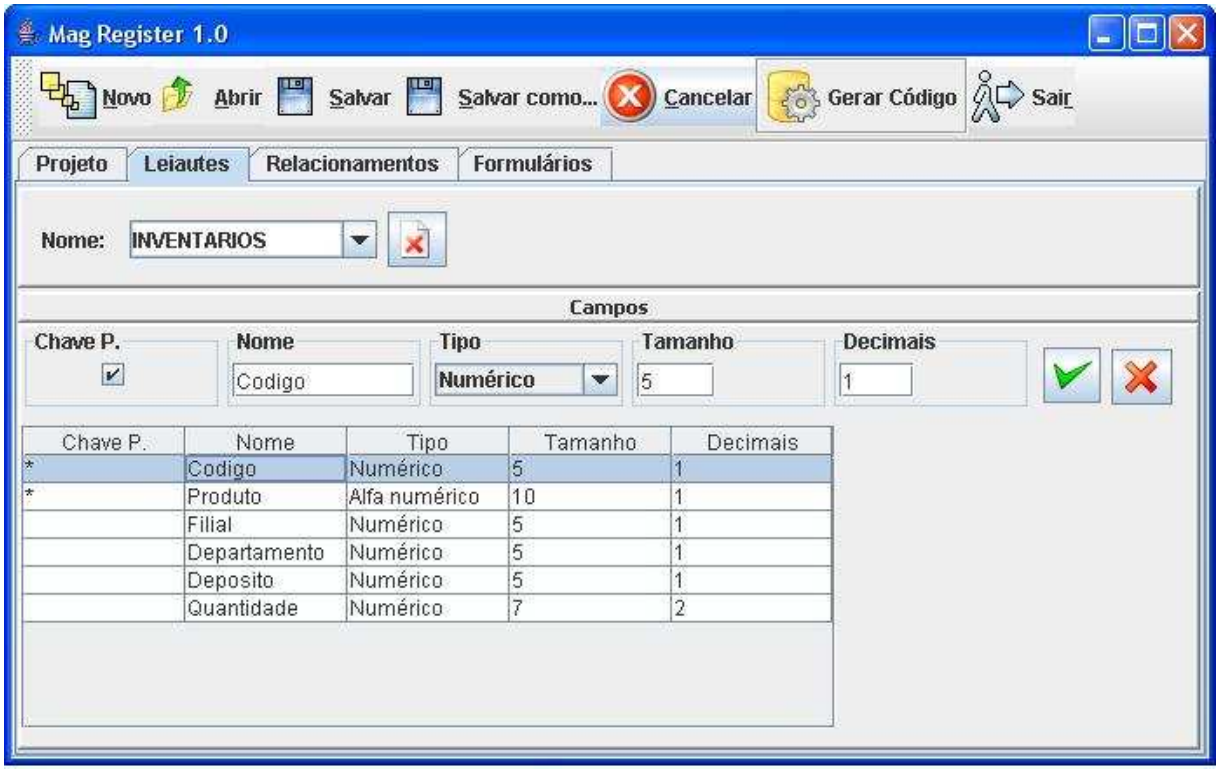


Figura 10 – Página “Leiautes” (Cadastramento dos leiautes do projeto)

O próximo passo a ser seguido, é efetuar o cadastramento dos relacionamentos entre os leiautes. Este cadastramento é feito na página “Relacionamentos” e para isto, devem ser informadas as relações “De > Para” entre os leiautes e os campos que os compõe. A Figura 11 demonstra a ilustração desta operação.

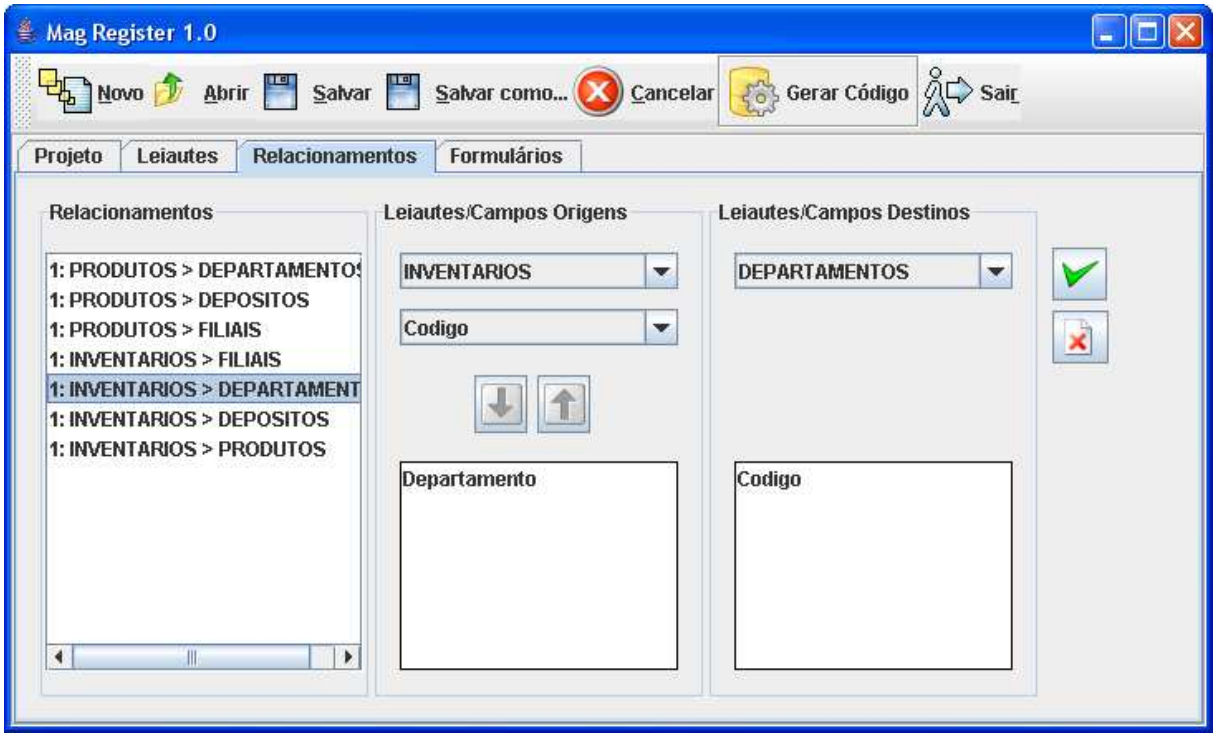


Figura 11 – Página “Relacionamentos” (Ligações entre os leiautes)

Por fim, devem ser cadastrados os formulários do projeto, que nada mais são do que as interfaces de coleta de dados que a ferramenta irá gerar. Na página de formulários devem ser informados o nome do formulário, a qual leiaute ele se refere, título que será apresentado na tela do dispositivo móvel e os campos do leiaute que farão parte do formulário. Estes cadastramentos devem ser feitos na página “Formulários”, conforme ilustra a Figura 12.

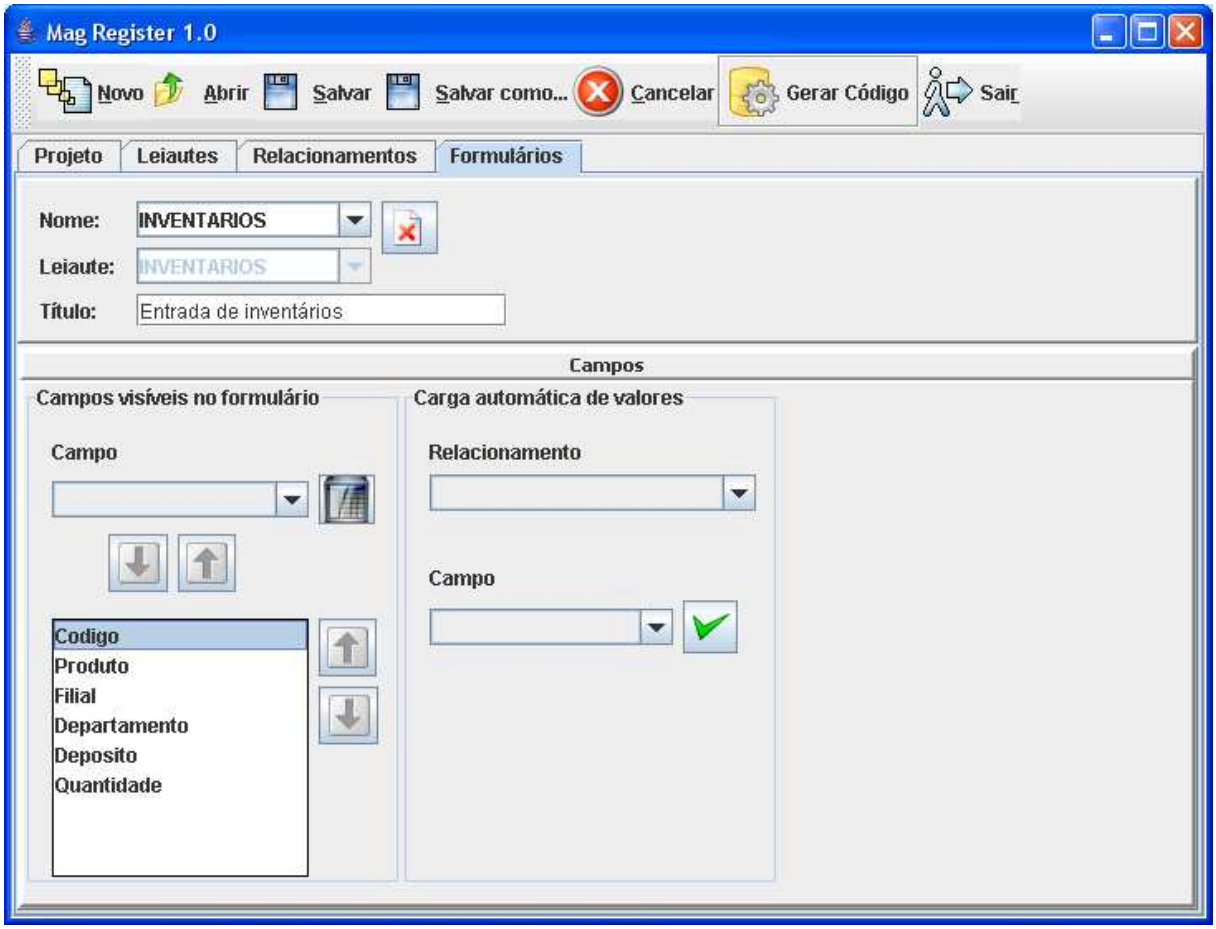


Figura 12 – Página “Formulários” (Interfaces de coletas de dados)

A partir do momento em que o projeto foi modelado, basta que seja gerado o código. Com base no exemplo seguido, a seguir é apresentado o resultado final da ferramenta, que é a aplicação para dispositivo móvel. A figura 13 demonstra o menu principal da aplicação, que foi gerado de acordo com os formulários criados, além das opções de excluir tabelas, importar e exportar dados.

Figura 13 – Menu principal no dispositivo móvel

A figura 14 apresenta um exemplo de uma interface para coleta de dados. Neste caso, a tela de entrada de inventários.

Figura 14 – Interface de coleta de dados

A figura 15 demonstra um exemplo da tela de pesquisa de registros.

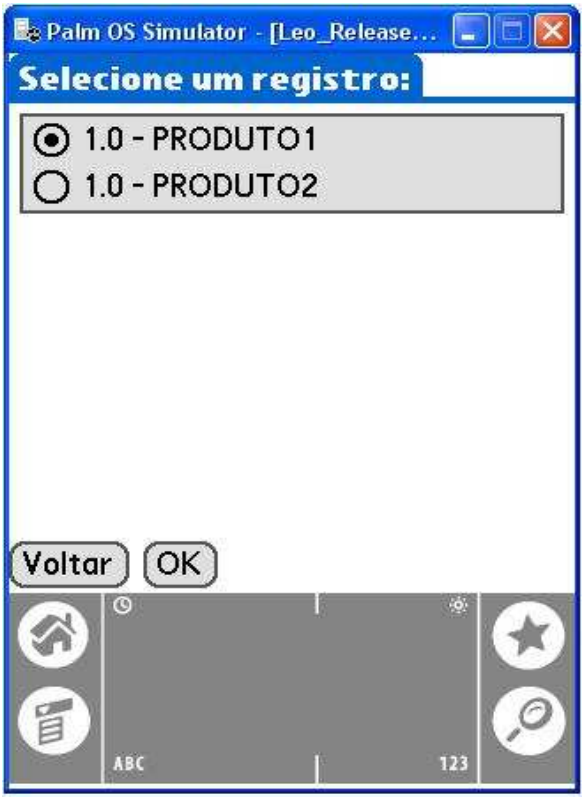


Figura 15 – Tela de pesquisa de registros

Os processos de importação e exportação são executados através do acesso às opções de menu para tal finalidade e apresentam mensagens confirmando o sucesso das operações, assim como eventuais mensagens de erro, tais como falha na conexão, erro do *parser* XML e assim por diante.

3.4 RESULTADOS E DISCUSSÃO

Para os testes da ferramenta, foi montado um cenário de geração de um programa para inventários de estoques. Foram criados leiautes de cadastro de filiais, departamentos, depósitos, produtos e inventários, assim como formulários para cada um desse leiautes. Mediante a utilização deste cenário de testes, foi possível testar a operacionabilidade dos casos de uso apresentados na seção da especificação. Foi constatado que a ferramenta atendeu aos requisitos especificados.

A geração do código da aplicação para o dispositivo móvel mostrou-se bastante rápida e segura. O uso do motor de templates Velocity facilitou bastante o processo de geração de código. Após terem sido definidos os *templates*, percebeu-se que muito improvavelmente um

código gerado terá problemas de compilação, pois os *templates* estão bem definidos e funcionando, independentemente da estrutura de leiautes e formulários que forem gerados.

A aplicação gerada pela ferramenta através do cenário de testes montado, foi validada apenas em um simulador do aparelho Palm Tungsten E2, utilizando-se o sistema operacional Palm OS e a máquina virtual J9 da IBM. Os testes efetuados na aplicação do dispositivo móvel também foram executados com sucesso e a máquina virtual funcionou perfeitamente sem ser necessário efetuar qualquer tipo de configuração. Todas as telas geradas pela ferramenta foram validadas através da inserção, consulta, alteração e exclusão de registros. Os *servlets* mostraram-se viáveis para a troca dos dados entre o PDA e o PC. O uso do banco de dados IBM DB2 Everyplace mostrou-se viável, tornando simples o processo de recuperação e gravação das informações na base de dados do dispositivo móvel. Nos testes realizados na aplicação gerada pela ferramenta, o banco de dados recebeu uma carga de dados relativamente pequena, fazendo com que não fosse possível ser avaliada a performance da aplicação com relação ao volume de dados.

Analisando as tecnologias adotadas para o desenvolvimento deste trabalho, percebe-se que há uma tendência na utilização da tecnologia JME para o desenvolvimento de aplicativos para dispositivos móveis. Exemplos desta afirmação são os trabalhos de Depiné (2002), Lessnau (2002) e Silva (2005) que respectivamente desenvolveram um software para cálculo de regularidade em rallies, um sistema de gerenciamento de dados customizável baseado em PDA’s e um protótipo de software para representantes comerciais.

Embora pensa-se em futuramente estender este trabalho para que possam ser geradas interfaces mestre-detalhes conforme em Silva (2005), onde é possível fazer um cadastro de pedidos e seus itens, inicialmente a implementação deste trabalho não é capaz de criar aplicações com leiautes deste tipo.

Em Lessnau (2002), é possível criar aplicações para dispositivos móveis de acordo com as necessidades, pois as aplicações são customizáveis. É neste aspecto que este trabalho assemelha-se com o trabalho citado, pois ambos têm a finalidade de gerar interfaces de acordo com as necessidades. Apesar disto, há uma principal diferença entre os dois, pois diferentemente do protótipo desenvolvido em Lessnau (2002), este trabalho irá utilizar banco de dados para a gravação dos dados e não arquivos textos.

Em Depiné (2002), a aplicação executa interfaces para entrada de dados, armazena temporariamente os dados nos PDA's e transfere posteriormente para um módulo servidor, o que torna a aplicação semelhante aos programas gerados por esta ferramenta.

O quadro 24 demonstra um comparativo entre este trabalho e os trabalhos correlatos.

Característica	Mag Register	Silva (2005)	Lessnau (2002)	Depiné (2002)
Geração de programas de forma customizável	X		X	
Banco de dados no dispositivo móvel	X	X		
Utilização plataforma JME	X	X	X	X
Telas no formato mestre-detalhe		X	X	

Quadro 24 - Comparativo entre esta ferramenta e os trabalhos correlatos

4 CONCLUSÕES

Mediante a constatação da necessidade de mercado de um software que permitisse coletar dados de produtos para a finalidade de inventários de estoques, surge a idéia de criar uma ferramenta que possibilitasse a criação de programas para esta finalidade. No amadurecer da idéia, percebeu-se que uma ferramenta que fosse capaz de gerar programas para inventários com base em leiautes configuráveis, poderia abranger muitos outros nichos e áreas de negócio, tais como programas para áreas de vendas, pesquisas de campo, etc.

Tem-se como resultado desta implementação, uma ferramenta flexível e portável, capaz de gerar programas com leiautes configuráveis. Consegue-se esta portabilidade, pois a linguagem Java está bastante difundida neste quesito e o banco de dados IBM DB2 Everyplace é portável a maioria dos sistemas operacionais para PDAs, tais como Palm OS e Windows CE. No desenvolvimento desta ferramenta foram utilizados vários métodos e tecnologias recentes, o que contribuiu muito para o desenvolvimento profissional. Dentre os assuntos abordados, os que tiveram maior ênfase foram a utilização da própria linguagem Java, experiência com *templates* e geração de código, JME, servidores de internet e padrão de desenvolvimento MVC.

Entretanto, é importante ressaltar que foram enfrentadas diversas dificuldades, principalmente nos momentos em que foi necessário optar por uma ou outra ferramenta e tecnologia. Cada um dos temas abordados neste trabalho foram amplamente pesquisados e validados. A maior parte das dificuldades se deram no momento de colocar em uso o banco de dados e os programas gerados para executarem no simulador de PDA’s. As formas de como se proceder com as instalações dos softwares nos PDAs não estão muito bem documentadas, o que fez com que fosse gasto bastante tempo com a montagem dos ambientes de testes.

4.1 LIMITAÇÕES

Os programas gerados pela ferramenta para os dispositivos não podem conter leiautes de dados com mais de um campo fazendo parte da chave primária. Outra limitação é o fato de não estar sendo possível definir como serão montadas as telas de pesquisas de registros nas interfaces de coletas de dados. A ferramenta está sempre gerando as telas de pesquisas

listando o primeiro campo do leiaute seguido de um hífen, seguido do segundo campo. Também não está sendo possível definir a URL de conexão com o *servlet* para a troca dos dados, pois a mesma está pré-fixada na ferramenta.

4.2 EXTENSÕES

Sugere-se como extensão deste trabalho a implementação de leiautes mestre-detalhes, para que, por exemplo, seja possível em uma única interface de coleta de dados poderem ser alimentadas duas tabelas no banco de dados simultaneamente. Esta implementação permitirá que sejam gerados programas mais avançados, como programas para entrada de pedidos de venda, entre outros.

Outra extensão poderia ser feita, no sentido de disponibilizar para que seja possível efetuar as trocas de dados entre o PC e o PDA através de uma conexão via cabos. Até então é possível efetuar a troca das informações apenas via protocolo HTTP. Em se tratando de troca de informações, também seria interessante implementar para que fosse possível definir modelos de arquivos textos para a troca de informações. Até o presente momento, a ferramenta permite que sejam mapeados arquivos no formato XML.

Também sugere-se que seja implementado algum recurso para que os usuários geradores dos programas, possam de alguma forma associar regras de negócio ao programa, tais como consistências de usuário, alimentação de outras tabelas em determinados eventos, entre outras.

REFERÊNCIAS BIBLIOGRÁFICAS

ANDRIOLLI, G. F. **J2ME MIDP:** Os primeiros passos para o desenvolvimento de aplicativos para celulares. [S.1], 2007. Disponível em: <http://www.devmedia.com.br/articles/viewcomp.asp?comp=120>. Acesso em: 14 jun. 2007.

DEPINÉ, F. M. **Protótipo de software para dispositivos móveis utilizando Java ME para cálculo de regularidade em rally.** 2002. 54f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais. Universidade Regional de Blumenau, Blumenau.

GOYA, M. **A Família DB2.** São Paulo, [2004]. Disponível em: <http://www.imasters.com.br/artigo.php?cn=2215&cc=60>. Acesso em: 10 fev. 2007.

HERRINGTON, J. **Code generation in action**. Greenwich: Manning Puplications, 2003.

IBM. **WebSphere everyplace micro environment**. [S.1], 2007. Disponível em <http://www-304.ibm.com/jct03001c/software/wireless/weme/features.html>. Acesso em: 28 maio 2007.

LESSNAU, T. A. **Sistema de gerenciamento customizável baseado em PDA's.** 2002. 84f. Trabalho de Conclusão de Curso (Engenharia da Computação) – Centro Universitário Positivo, UNICENP, Curitiba.

MATTOS, E. T. de. **Programação Java para wireless:** aprenda a desenvolver sistemas em JME. São Paulo: Digerati Books, 2005.

MOURA, M. F.; CRUZ, S. A. B. **Formatação de dados usando a ferramenta Velocity**. Campinas, 2002. Disponível em: <http://www.cnptia.embrapa.br/modules/tinycontent3/index.php?id=2>. Acesso em: 27 maio 2007.

MUCHOW, John W. **Core J2ME technology & MIDP**. Palo Alto: Prentice Hall. 1. ed. 2001.

PALMLAND. **PalmTops**. [S.1], 2007. Disponível em: <http://www.palmland.com.br/palmtops/>. Acesso em: 27 fev. 2007.

RODHES, Neil.;MCKEEHAN, Julie. **Palm programing**: the developer´s guide. O´Reilly Media. 1.ed. 1998.

SCHMITT JUNIOR, A. J. **Protótipo de front end de controle de acesso usando J2ME**. 2004. 70 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SCHVEPE, C. **Gerador de código Java a partir de arquivos do Oracle 6I.** 2006. 75 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SILVA, R. L. da **Aplicativo para representante comercial em dispositivo móvel (PDA) usando a tecnologia JME e banco de dados.** 2005. 72f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro Universitário Positivo Núcleo de Ciências Exatas e de Tecnologia, Curitiba.

SOUZA, B. P. F. de. **A plataforma Java 2**. [S.l.],[2003]. Disponível em: <http://www.javaman.com.br/apres/java&jini/index.html>. Acesso em: 22 mai. 2007.

STEIL, R. **Introdução ao Velocity**. [S.l.], 2007. Disponível em: <http://www.guj.com.br/java.tutorial.artigo.18.1.guj>. Acesso em: 28 maio 2007.

STEPHENS, Matt. **Software reality**: automated code generation. [S.l.], 2002. Disponível em: <http://www.softwarereality.com/programming/code_generation4.jsp>. Acesso em: 20 maio 2007.

SUN. **Java technology**. [S.l.], 2007. Disponível em: <http://java.sun.com/about>. Acesso em: 22 maio 2007.

TAKECIAN, Pedro L. et al. **Sistema de gerenciamento de workflows**. 2004. 35 f. Trabalho de Formatura Supervisionado (Bacharelado em Ciência da Computação) – Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo. Disponível em: <http://gul.linux.ime.usp.br/~plt/mac499/monografia.pdf >. Acesso em: 19 maio 2007.

WILDING-MCBRIDE, Daryl. **Java development on PDAs**: building applications for PocketPC and Palm devices. Boston: Addison-Wesley. 1. ed. 2003.

APÊNDICE A – Implementação da classe controller.Controller

```
package controller;

import controller.GenerateCode;
.
.

public class Controller {

    public static final String PROJECT_NAME = "ProjName";
    public static final String PROJECT_CREATEDATA = "ProjCreateData";
    .
    .

    private Project project;
    private Layout currentLayout;
    private int currentIndexField;
    private int currentIndexLayout;
    private Form currentForm;
    private int currentIndexForm;


    public Controller() {
    }

    /** Metodo responsável por criar um novo projeto */
    public void newProject() {
        project = new Project();
    }

    /** Metodo responsável por salvar um projeto */
    public void saveProject(String fullPath, String aName, Date aCreateData, String
aComments) throws FileNotFoundException {
        .
        .
        .
        xml.writeObject(this.project, fullPath);
    }

    /** Metodo responsável por carregar um projeto */
    public HashMap <String, String> getProject(String fullPath) throws
           FileNotFoundException {
        HashMap <String, String> values = null;
        this.project = (Project) xml.readObject(fullPath);
        if  (this.project != null) {
            .
            .
            .
            Layout layoutAux = this.getFirstLayout();
            if  (layoutAux != null) {
                values.put(this.PROJECT_FIRST_LAYOUT,layoutAux.getLayName());
            } else {
                values.put(this.PROJECT_FIRST_LAYOUT," ");
            }
        }
        return values;
    }
```

```
/** Metodo responsável por carregar um leiaute */
    public int checkLayout(String aLayoutName) throws Exception {
        Layout layoutAux = project.getLayout(aLayoutName);
        if  (layoutAux == null) {
            layoutAux = new Layout();
            layoutAux.setLayName(aLayoutName);
            project.addLayout(layoutAux);
            this.currentLayout = layoutAux;
            return 0;
        } else {
            this.currentLayout = layoutAux;
            return 1;
        }
    }

    .
    .
    .

    /** método reponsável por retornar a lista de campos relacionados */
    public LinkedList <String> getResultLookup(String aFieldName) {
        LinkedList <String> values = new LinkedList <String> ();
        FormField fieldAux = this.currentForm.getField(aFieldName);
        if  (fieldAux != null) {
            RelationShip relationShipAux = fieldAux.getLookupRelationShip();
            if  (relationShipAux != null) {
                Field lookupField = fieldAux.getResultLookupField();
                if  (lookupField != null) {
                    values.add(relationShipAux.getName());
                    values.add(lookupField.getFieldName());
                }
            }
        }
        return values;
    }

    // método responsável por chamar a execução da geração de código
    public void generateCode() {
        try {
            GenerateCode.generateLayerModel(this.project);
            GenerateCode.generateForms(this.project);
            GenerateCode.generateMainMidlet(this.project);
            GenerateCode.generateMainMenu(this.project);
            GenerateCode.generateMainXMLGenerator(this.project);
        } catch (ResourceNotFoundException ex) {
            ex.printStackTrace();
        } catch (MethodInvocationException ex) {
            ex.printStackTrace();
        } catch (ParseErrorException ex) {
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
    .
    .
    .
}
```

Quadro 25 – Implementação da classe controller.Controller

APÊNDICE B – Exemplo de classe do parser de XML

```
import java.io.*;
.
.
import org.xmlpull.v1.XmlPullParserException;

public class XMLParser extends Thread {

    .
    .

    public XMLParser(MainMidlet aMidlet, final String aUrl) {
        mainMidlet = aMidlet;
        url = aUrl;
        persistantLayout = null;
    }

    public void run() {
        HttpConnection hc;
        try {

            DatabaseMetaData metadata = mainMidlet.con.getMetaData();
            ResultSet rs = metadata.getTables(null, null, null, new
                           String[]{"TABLE"});
            while (rs.next()) {
                xmlRecord = new XMLRecord();
                tableName = rs.getString("TABLE_NAME");
                recordCount = 0;
                // set up the network connection
                hc = null;
                try {
                    try {
                        hc = (HttpConnection)Connector.open(url+
                             "?operation=importOnPDA&tableName="+tableName);
                        InputStream in = hc.openInputStream();
                        if  (in != null) {
                            mainMidlet.showAlert("Importando dados da tabela " +
                                                 tableName.toUpperCase());
                            parse(in);
                        } else mainMidlet.showAlert("Não foi possível carregar o
                                                arquivo XML a partir do servlet.");
                    } catch (Exception ex) {
                        Exception e = new Exception(ex.getMessage());
                    }
                } finally {
                    try {
                        if (hc != null) {
                            hc.close();
                        }
                    } catch (IOException ex) {
                        mainMidlet.showAlert("Não foi possível fechar a conexão com
                                                o servlet: "+ex.getMessage());
                    }
                }
            }
        } catch (Exception ex) {
            mainMidlet.showAlert("Não foi possível importar os dados:
                                 "+ex.getMessage());
        }
        mainMidlet.showAlert("Dados importados com sucesso.");
```

```
   .
   .
   .
   public void parse(InputStream in) throws Exception {

        KXmlParser parser = new KXmlParser();
        parser.setInput(new InputStreamReader(in));

        parser.nextTag();

        //posiciona na Tag register
        parser.require(XmlPullParser.START_TAG, null, "register");

        while   (parser.nextTag() != XmlPullParser.END_TAG) {
            //posiciona na Tag record
            parser.require(XmlPullParser.START_TAG, null, "record");

            //Cria o objeto que armazenará os dados
            createObject();

            while   (parser.nextTag() != XmlPullParser.END_TAG) {

                //posiciona na Tag record
                parser.require(XmlPullParser.START_TAG, null, "field");
                XMLRecord xmlRecord = new XMLRecord();
                while   (parser.nextTag() != XmlPullParser.END_TAG) {
                    //posiciona na Tag name
                    parser.require(XmlPullParser.START_TAG, null, null);
                    String name = parser.getName();
                    String text = parser.nextText();

                    if  ((!(name.equalsIgnoreCase("name"))) &&
                        (!(name.equalsIgnoreCase("value")))) {
                        Exception e = new Exception("MagRegister: Arquivo XML
                                                    possui formato inválido.");
                    } else {
                        if  (name.equalsIgnoreCase("name")) {
                            xmlRecord.setFieldName(text);
                        } else {
                            xmlRecord.setFieldValue(text);
                            persistantLayout.setFromXMLRecord(xmlRecord);
                        }
                    }
                    parser.require(XmlPullParser.END_TAG, null, name);
                }

                parser.require(XmlPullParser.END_TAG, null, "field");
            }

            parser.require(XmlPullParser.END_TAG, null, "record");
            persistantLayout.save(mainMidlet.st);
        }
    }
}
```

Quadro 26 – Exemplo de classe do parser de XML