PEDRO AUGUSTO FERREIRA ROCHA

MAPEAMENTO DE BANCO DE DADOS RELACIONAL PARA COMPONENTES DE INTERFACE GRÁFICA EM DISPOSITIVOS MÓVEIS

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação, para a obtenção do título de bacharel.

Orientador

Prof. Antônio Maria Pereira de Resende

LAVRAS
MINAS GERAIS - BRASIL
2009

PEDRO AUGUSTO FERREIRA ROCHA

MAPEAMENTO DE BANCO DE DADOS RELACIONAL PARA COMPONENTES DE INTERFACE GRÁFICA EM DISPOSITIVOS MÓVEIS

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação, para a obtenção do título de bacharel.

APROVADA em __ de _____ de 2009.

_____
Ana Rubélia Mendes de Lima Resende

_____
Prof. Juliana Galvani Greghi

_____
Prof. Antônio Maria Pereira de Resende
(Orientador)

LAVRAS
MINAS GERAIS – BRASIL
2009

DEDICATÓRIA

Dedico este trabalho primeiramente aos meus pais João e Marisa e ao meu irmão Eduardo por me ajudarem nos estudos, por sempre me motivarem a estudar e seguir este caminho. Dedico também ao meu irmão Rafael que sempre me apoiou em Lavras tanto nos estudos e em várias outras situações. A todos os meus amigos que fiz nesta caminhada, meus companheiros de turma, o pessoal da República Taverna, a toda minha família e também a minha namorada Ariana que me ajudou bastante na elaboração deste trabalho e me apoiou nos momentos mais difíceis. Que nunca falte paz, alegria e esperança para todos eles.

Dedico especialmente aos meus amigos e companheiros que viveram comigo ao longo destes quatro anos, a República Ninho do Amor e República Galo Doido.

Queria também dedicar ao meu orientador Antônio Maria que me ajudou bastante nesse projeto e em outros trabalhos. Por fim, a todos os outros que estão seguindo o mesmo caminho e que também irão encontrar as mesmas dificuldades que eu encontrei em minha jornada.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Atualmente, muitas empresas utilizam dispositivos móveis como celulares, *Personal Digital Assistants* (PDA) e *smarthphones* para coleta de dados aplicáveis na estatística, vendas, pagamentos, dentre outros. Essa grande adesão a dispositivos móveis para coleta de dados despertou o interesse de empresas no desenvolvimento de ferramentas do tipo *rapid application development* (RAD) para automatização da programação de códigos comum a esse tipo de aplicação. Este trabalho de pesquisa desenvolveu um mapeamento de um modelo de banco de dados existente para os componentes de uma interface gráfica de dispositivos móveis, de acordo com os atributos requeridos na aplicação. A partir do mapeamento, construiu-se um protótipo denominado Gerador de Códigos para Interface de Dispositivos Móveis (GACIDIM) capaz de gerar automaticamente o código fonte da interface a partir das características dos relacionamentos entre as tabelas de um banco de dados, seus campos e atributos.

Palavras-Chave: *Rapid Application Development*, Ferramenta Case para dispositivos móveis, Coletores de Dados, Mapeamento de Banco de Dados Relacional.

1. INTRODUÇÃO

De grandes a pequenas empresas e também pessoas físicas utilizam hoje em dia algum tipo de dispositivo móvel, seja um celular, PDA, *smartphones*, *laptops*, dentre outros. Além de prover um meio de comunicação e entretenimento para seus usuários, estes dispositivos são também utilizados para cálculos financeiros, comércio, pesquisa, e várias outras funcionalidades.

Muitos *softwares* para pequenos dispositivos têm por base a coleta de dados. Estes podem ser utilizados em estatísticas de uma empresa, no cadastro de uma venda realizada, ou até mesmo para uso pessoal, como coletar todos os produtos comprados no dia a dia afim de obter um balanço geral dos gastos.

Aplicativos com intuito de coletar dados, na grande maioria dos casos, têm por trás um *software desktop* ou um *software* via WEB que realiza todo o controle, transformação da informação coletada, e a persistência de dados através de um banco de dados relacional. Com base neste escopo, surgiu a necessidade de uma ferramenta que automatize grande parte da criação do código desses coletores a partir de um banco de dados previamente estabelecido. Contudo, para a elaboração desta ferramenta foi necessário um estudo de como mapear uma interface gráfica para dispositivos móveis.

2. REFERENCIAL TEÓRICO

2.1. Dispositivos e Computação Móvel

Computação Móvel é um conceito que simboliza o acesso a vários tipos de serviços e informações, independente de sua localização. Seja esse acesso feito via *internet*, via rede telefônica ou até mesmo por redes pessoais utilizando *Bluetooth*[1]. A Figura 1 exibe um cenário de computação móvel (JOHNSON; 2007).

Dispositivos móveis são dispositivos que se enquadram no cenário da computação móvel. Classificam-se em três grupos. O primeiro é composto pelos *notebooks*, cuja capacidade computacional é equivalente a um computador *desktop*. O segundo grupo compõe-se pelos PDAs, cuja capacidade computacional é bem menor que os *notebooks*, porém possuem mais recursos quando comparados a celulares. E o terceiro grupo é composto pelos celulares, que possuem um recurso computacional bem pequeno quando comparado com os dispositivos dos outros grupos. Porém, hoje em dia já existem vários celulares que possuem capacidade de processamento e outras ferramentas como GPS[2], equivalente ou superior aos PDAs. São chamados de *Smartphones* (JOHNSON; 2007).

---

[1] Tecnologia baseada em comunicação por frequência de rádio de curto alcance, independente de custos para tráfego de dados e com baixo custo de energia. (HAARTSEN, Ericsson; 2000)

[2] Global Positioning System (BULUSU, HEIDEMANN, ESTRIN; 2000 apud HOFMANN-WELLENHOF, LICHTENEGGER, COLLINS; 1997)

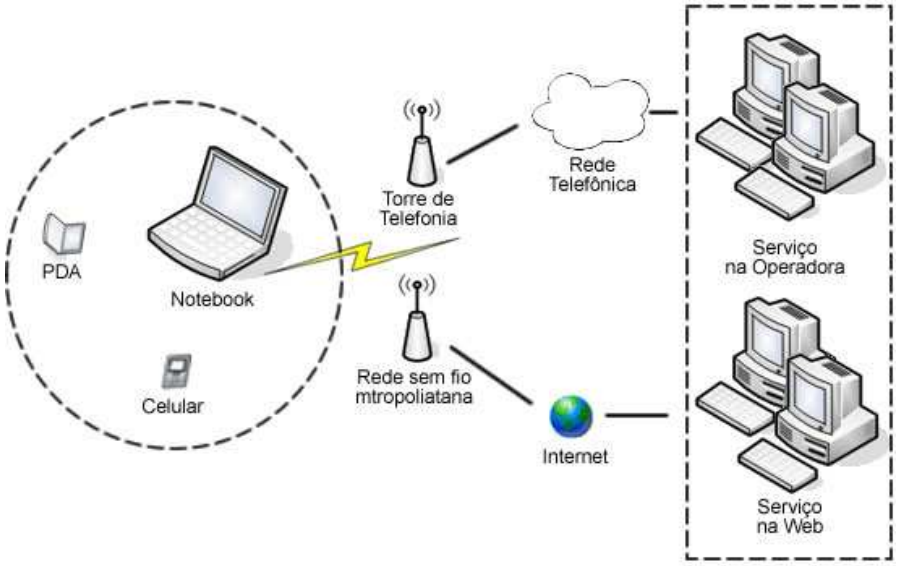

Figura 1 - Cenário da Computação Móvel.[3]

2.2. Plataforma Java

A plataforma Java foi desenvolvida pela *Sun Microsystems*[4] em 1994, apresentando sucesso tanto em meio empresarial quanto acadêmico. A peculiaridade da plataforma é a sua grande portabilidade, visto que a aplicação desenvolvida com Java pode ser usada em diferentes plataformas como *desktop*, servidores, em diferentes tipos de Sistemas Operacionais (SO), aparelhos celulares e televisões.

[3] Fonte: (JOHNSON; 2007)
[4] Página Oficial: http://www.sun.com/

O intuito da *Sun* era criar uma linguagem de programação universal, que poderia ser utilizada desde dispositivos móveis, eletrodomésticos, computadores pessoais, grandes servidores, dentre vários outros aparelhos. Para isso, a linguagem, quando compilada, gera um *bytecode*[5] que deverá ser interpretado por uma *Java Virtual Machine* (JVM). Portanto, para cada sistema operacional, a *Sun* produziu uma JVM, assim, um código que execute em uma máquina poderá executar em outra diferente, desde que ela possua os recursos utilizados. O funcionamento básico de como Java trata essa portabilidade pode ser resumido na Figura 2.

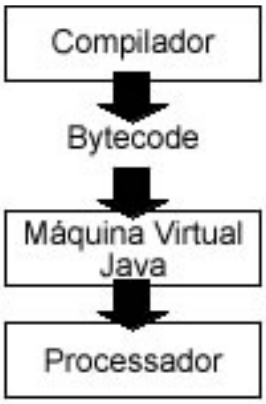

Figura 2 - Máquina Virtual Java.[6]

Porém, o intuito de usar sempre o mesmo código não é bem o que acontece. Devido à grande gama de recursos computacionais existentes, desde um servidor até um dispositivo móvel, nota-se que o mesmo código fonte não poderá ser utilizado. Isto se deve ao fato de que um celular, por exemplo, não terá o mesmo poder de processamento, memória, dentre outros fatores que um

[5] Código gerado que não é imediatamente executável. Necessita de uma máquina virtual para ser interpretado.
[6] Fonte: (JOHNSON; 2007)

computador *desktop*. Para isso, a *Sun* subdividiu a plataforma Java em quatro. São elas: *Java 2 Enterprise Edition* (J2EE), *Java 2 Standard Edition* (J2SE), *Java 2 Micro Edition* (J2ME) e *Java Card*, *vide* Figura 3 (JOHNSON; 2007).

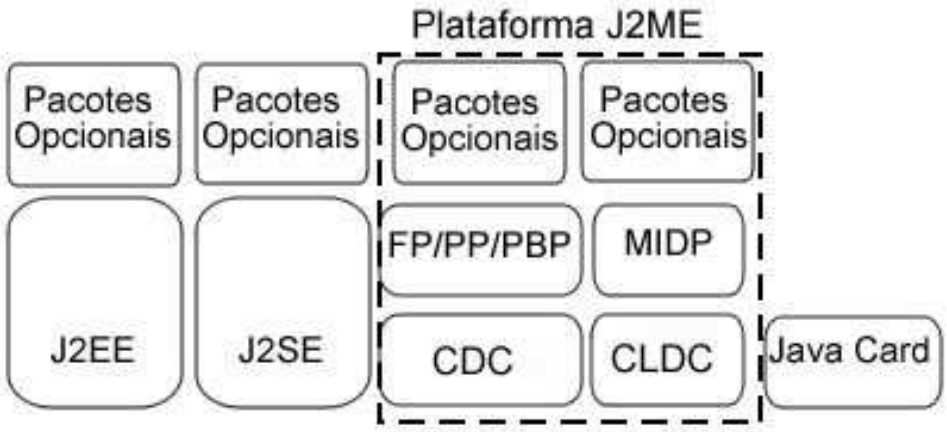

Figura 3 - Plataforma Java e suas subdivisões.[7]

2.3. J2ME

Em 1999, a *Sun* lançou a plataforma J2ME, com intuito de permitir o desenvolvimento de aplicativos para dispositivos com baixa capacidade de processamento e memória.

A plataforma J2ME permite a distribuição de aplicativos em diferentes tipos de dispositivos móveis, porém estes dispositivos devem possuir os mesmos perfis de configuração da aplicação elaborada. Para prover essa funcionalidade a *Sun* dividiu a plataforma J2ME em camadas *vide* Figura 4 (JOHNSON; 2007).

[7] Fonte: (JOHNSON; 2007)

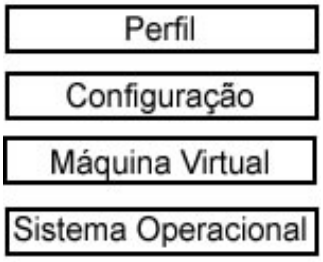

Figura 4 - Camadas da Arquitetura J2ME.[8]

A primeira camada é a Máquina Virtual. Em J2ME, a JVM é chamada de *Kilobyte Virtual Machine* (KVM). Seu tamanho é de aproximadamente 80 *Kbytes* e tem finalidade executar programas em dispositivos com capacidade computacional de até 512 *Kbytes* de memória total e com processadores de 16 ou 32 bits (JOHNSON; 2007).

A segunda camada é representada pela configuração, que é o conjunto de classes que dispõem as funcionalidades essenciais e básicas para o dispositivo móvel. As configurações disponíveis para dispositivos móveis são a *Connected Limited Device Configuration* (CLDC) e *Connected Device Configuration* (CDC). A CLDC é voltada a dispositivos mais pobres em poder computacional, como celulares. Já o CDC é voltado aos dispositivos com mais capacidade computacional que celulares, porém com menor carga que um *laptop*. Um exemplo são os PDAs (JOHNSON; 2007).

Acima das configurações temos os perfis. Os perfis (ou *profiles*) mais populares são o *Mobile Information Device Profile* (MIDP), voltado para aplicativos com configuração CLDC, e o *Personal Profile* (PP), voltado para aplicativos com a configuração CDC. Os perfis possuem bibliotecas mais

[8] Fonte: (JOHNSON; 2007)

específicas do que as bibliotecas providas pelas configurações (JOHNSON; 2007).

As aplicações Java desenvolvidas com o perfil MIDP são chamadas de MIDlets. Ao construir um aplicativo em J2ME, as classes são empacotadas em um arquivo jar. As MIDlets conseguem fazer vários tipos de conexão, seja entre dispositivos móveis através de conexão *Bluetooth* ou com servidores WEB através da internet com o protocolo HTTP 1.1 (JOHNSON; 2007). Além disto, provê funcionalidades com o uso de recursos externos, como GPS (KENTERIS, GAVALAS, ECONOMOU; 2009).

2.3.1. Persistência de dados utilizando Floggy

Floggy é um *framework* brasileiro de persistência de dados sob a licença *Apache License Version* 2.0[9]. O objetivo deste *framework* para J2ME é abstrair os detalhes da persistência de dados em J2ME do desenvolvedor. (FLOGGY, 2006-2009)

De acordo com as informações em sua página, *Floggy* é composto em dois módulos:

- *Framework*: responsável por toda a manutenção da persistência de dados no aparelho móvel. Suas principais funcionalidades são adicionar, alterar, remover e recuperar dados em uma aplicação J2ME. Suas classes e métodos serão utilizados dentro de uma aplicação MIDP e
- *Weaver*: gera e analisa o *bytecode* responsável pela persistência das classes desejadas.

[9] http://www.apache.org/licenses/LICENSE-2.0

Devido à sua facilidade de implementação, *Floggy* foi escolhido para realizar a persistência dos dados da aplicação gerada pelo GACIDIM. O modo como o *Floggy* funciona pode ser observado *vide* Figura 5.

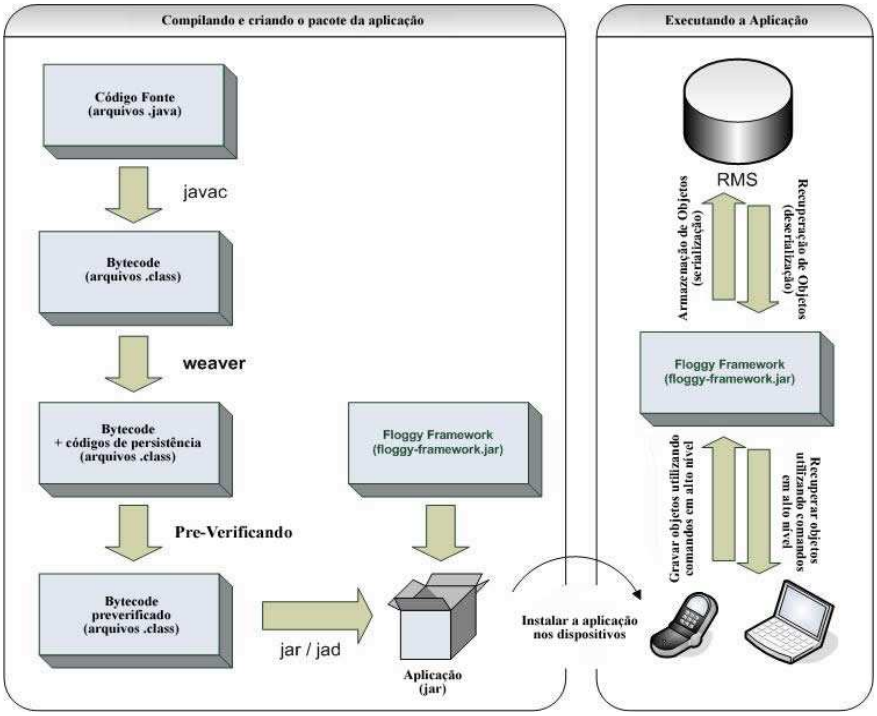

Figura 5 - Funcionamento *Floggy*.[10]

2.4. J2EE

A linguagem Java não poderia deixar de abranger também a rede de comunicação *internet*. Para isso, foi desenvolvido o módulo J2EE[11]. Suas

[10] Fonte: (FLOGGY, 2006-2009)
[11] Java 2 Enterprise Edition

características são muito parecidas com as de outras linguagens utilizadas na WEB, como exemplo PHP[12]. O J2EE é a plataforma mais rica e poderosa do Java. Ela consta de vários *frameworks* e bibliotecas de alto nível disponibilizadas na rede. Todas as ferramentas disponibilizadas são baseadas no básico de J2EE, que são os *Servlets* e as páginas *Java Server Page* (JSP).

2.4.1. Servlets e JSP

O nome *servlet* tem como origem serviços. Isso se deve ao fato de que os *servlet* são comumente utilizados para prestar serviços via servidores WEB. As páginas JSP são sempre convertidas em *servlets* equivalentes, porém, a estrutura de uma JSP foi projetada para que o desenvolvedor tenha a camada de negócio separada da camada de apresentação (DEITEL, DEITEL; 2006), *vide* o paradigma de desenvolvimento de software *Model-View-Controlle*r (MVC) (VEIT, HERRMANN; 2003).

Os *servlets* e JSP ganharam um lugar de destaque no mundo da programação. Os servidores de WEB mais importantes, como o *Apache*[13], *Websphere*[14], IIS[15], dentre outros, reservaram um lugar para a tecnologia desenvolvida pela *Sun* ou possuem algum *plug-in* desenvolvido por terceiros. (DEITEL, DEITEL; 2006)

Dentre todos os *frameworks* criados para facilitar o trabalho no desenvolvimento de *servlets* e JSPs, um deles, utilizado no desenvolvimento do

[12] Página Oficial: http://php.net/
[13] Página Oficial: httpd.apache.org
[14] Página Oficial: http://www-01.ibm.com/software/websphere/
[15] Página Oficial: Internet Information Services: http://www.iis.net/

GACIDIM, foi o JSF. Sua principal funcionalidade é facilitar o desenvolvimento da camada de visualização de aplicações WEB.

2.5. Serviços Baseados em Localização

Serviços Baseados em Localização, *Location-Based Service* (LBS), é o conceito que consiste em fornecer às pessoas serviços relacionados à sua posição geográfica atual. De acordo com Mahmoud (2004), os serviços baseados em localização consistem em três perguntas básicas. São elas: "onde estou?", "quem está ao meu redor?" e "como faço para chegar em um lugar específico?".

Para a determinação da posição geográfica, basta saber a latitude, longitude e altitude da posição em questão. A Latitude é medida em uma escala de 0° a 90°, seja a norte da linha do Equador, normalmente estipulado com valores positivos, ou a sul da linha do Equador, normalmente estipulado com valores negativos. A linha do Equador é representada com o valor de 0°. A medida de longitude tem um leque de 180°, sendo considerado o marco 0° o Meridiano de *Greenwich*[16]. A medida para o lado leste normalmente é caracterizada pelo sinal positivo, e a medida para o lado oeste utiliza o sinal negativo. Já a altitude é representada na unidade metros, seu valor é calculado pela diferença de altura em relação ao nível do mar, considerado altura 0. (MAHMOUD, 2004)

O método de reconhecimento da localização pode ser efetuado por diversas tecnologias. Há três maneiras adotadas por aplicativos móveis utilizando J2ME. São elas: uso de satélites através do Sistema de

[16] Linha imaginária que passa pela cidade de Greenwich na Inglaterra.

Posicionamento Global (GPS), uso da rede de telefonia móvel ou localização por posicionamento de curta distância. (MAHMOUD, 2004)

2.6. GPS

Global Positioning System (GPS) (BULUSU, HEIDEMANN, ESTRIN; 2000 *apud* HOFMANN-WELLENHOF, LICHTENEGGER, COLLINS; 1997) foi desenvolvido pelo Departamento de Defesa dos Estados Unidos da América (*Departament of Defense* – DoD) . O GPS é um sistema que abrange o mundo inteiro. Permite que o usuário na superfície terrestre tenha quatro satélites para serem rastreados. Com esses quatro satélites é possível calcular a posição do usuário em relação à posição dos satélites em um sistema de coordenadas (latitude, longitude e altitude). Um esquema de disposição dos satélites ao longo do globo terrestre é demonstrado pela Figura 6.

O GPS fornece dois tipos de serviços, são eles o *Standard Positioning Service* (SPS) e o *Precise Positioning Service* (PPS). O primeiro é gratuito para qualquer usuário no mundo inteiro, porém proporciona uma taxa de erro em metros enquanto o PPS fornece uma taxa de erro em centímetros em ambientes abertos. Já o serviço PPS é utilizado somente para uso militar e para usuários autorizados (MONICO; 2000).

Aplicações que utilizam GPS possuem diversas funcionalidades e podem variar na precisão de acordo com o aparelho GPS utilizado. Algumas aplicações fazem transformações entre os sistemas de coordenadas e ou produzem uma saída compatível com o Sistema de Informações Geográficas (SIG) (KUHNEN; 2003 apud CÂMARA; 1996).

As vantagens de utilizar receptores GPS são: o seu tamanho e a grande precisão que estes dispositivos provêem comparado a outros métodos de captura de localização, por exemplo via operadora de telefonia celular. Porém, quando utilizados em ambiente fechados, podem não ter uma boa precisão devido à obstáculos entre o receptor GPS e os satélites. (KUHNEN; 2003 *apud* ROCHA; 2001).

Figura 6 - Constelação de Satélites GPS.[17]

2.7. Modelo Entidade Relacionamento

O Modelo Entidade Relacionamento (MER) tem como objetivo representar as estruturas de dados de uma forma mais próxima do mundo real. O modelo foi criado a partir de análises de outros três modelos: *network model*,

[17] Fonte: (http://media.obsessable.com/media/2009/01/21/24satellite-gps.jpg)

relational model e *entity set model*. Sua peculiaridade é a capacidade de separar o modelo de entidades do modelo de relacionamento.

O MER possui três conceitos fundamentais para visualização. São eles: entidade, atributo e relacionamento. O conceito de entidade é atribuído aos objetos reais, que possuem alguma função ou realmente existem no mundo real. Os atributos são as características que se deseja saber sobre as entidades envolvidas. Já o relacionamento é a descrição de como essas entidades relacionam entre si.

Os atributos possuem várias classificações distintas, são: atributo simples, atributo composto, atributo chave e atributo chave estrangeira. Os atributos simples não possuem nenhum fator em especial; Já atributos compostos, são atributos que possuem vários outros atributos, por exemplo, um atributo endereço, que poderá compor-se pelo atributo rua, número, complemento. Os atributos chaves são os responsáveis por tornar uma entidade única, exclusiva.E , por fim, os atributos chaves estrangeira, são utilizados para indicar com qual outra entidade a entidade que pertence está relacionada.

Os relacionamentos são classificados de acordo com sua cardinalidade. Uma relação pode ser: de um para um, uma relação única entre duas entidades; ou uma relação de um para muitos, que simboliza a relação de uma entidade com várias outras entidades; ou uma relação de muito para muitos, que simboliza uma relação em que várias entidades podem se relacionar com várias outras entidades (CHEN; 1976a).

Com o uso do modelo proposto por Peter Pin Chen (1976a), teve-se a necessidade de uma padronização de como representar o MER em diagrama. O Diagrama Entidade Relacionamento (DER) propõe onze regras utilizadas para

interpretar o MER e esquematizar em forma de diagrama, Figura 7, são: (CHEN; 1976b).

- Regra 1: Um substantivo próprio, em inglês, corresponde a uma entidade;
- Regra 2: Verbos transitivos, em inglês, representam um relacionamento em um DER;
- Regra 3: Um adjetivo, em inglês, corresponde a atributos de uma entidade;
- Regra 4: Advérbios, em inglês, correspondem a atributos de um relacionamento;
- Regra 5: Sentenças na forma “Há... A em B” poderá ser convertida a forma mais adequada ao DER como “B possui ... A”;
- Regra 6: Sentenças na forma “O A de B é C” e também se o C for um substantivo próprio, em inglês, então A poderá ser considerado como um relacionamento entre B e C e além disso B e C serão considerados entidades;
- Regra 7: Sentenças na forma “O A de B é C” e também se o C não for um substantivo próprio, em inglês, então A deve ser considerado como um atributo de B. Assim Y representa uma entidade e Z é considerado um valor;
- Regra 8: Operações numéricas ou objetos algébricos são considerados atributos;
- Regra 9: O gerúndio, em inglês, corresponde a um relacionamento convertido no tipo entidade no DER;

- Regra 10: Uma cláusula, em inglês, é representada por uma entidade de mais alto nível. Abstrai-se um conjunto de entidades e relacionamentos de um DER e
- Regra 11: Uma sentença, em inglês, representa o topo da abstração. É composto por entidades e um relacionamento, que recursivamente podem ser compostos por outras entidades e relacionamentos.

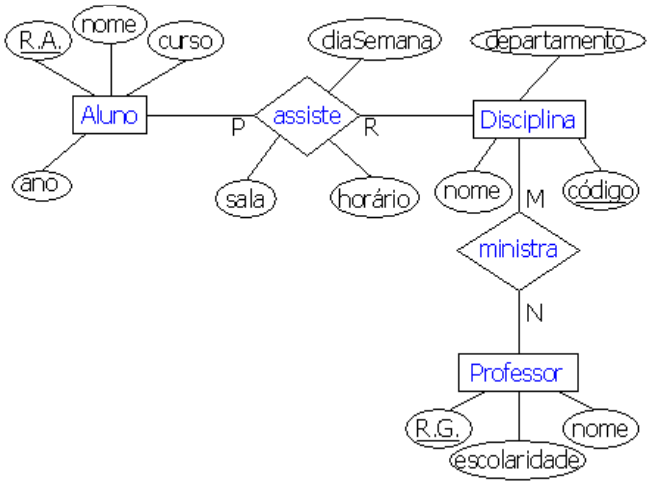

Figura 7 - Exemplo de um Diagrama Entidade Relacionamento.[18]

2.8. Rapid Application Development

O termo *Rapid Application Development* ou simplesmente RAD é uma metodologia de desenvolvimento de sistemas que auxilia o programador a desenvolver em menor tempo. Normalmente, as ferramentas RAD utilizam protótipo para auxiliar o desenvolvedor seja em testes ou planejamento, além de

[18] Fonte: (http://www2.dc.ufscar.br/~carvalho/GUBD/aulas/35K.gif)

reutilizar componentes, promovendo assim maior produtividade e qualidade (COLEMAN, VERBRUGGEN; 1998 apud MARTIN; 1991).

Ferramentas RAD aumentam a produtividade, diminuem o tempo de entrega e, além disso, fornecem ao desenvolvedor um melhor conforto no desenvolvimento (COLEMAN, VERBRUGGEN; 1998 apud REILLY; 1995). Isto se deve ao fato de prover uma alta interação com usuário, fornecer respostas rápidas as ações e mudanças escolhidas pelo usuário e por reutilizar componentes. Porém, projetos que utilizam RAD podem sofrer sérias consequências se não forem adotados alguns critérios como: uma equipe qualificada para o trabalho, uma gestão do projeto bem capacitada e uma definição de quais metodologias serão utilizadas desde o início do projeto (COLEMAN, VERBRUGGEN; 1998). Vantagens e desvantagens foram analisadas por Coleman e Verbruggen através de outros documentos sobre RAD, veja Tabela 1.

Tabela 1- Vantagens e desvantagens de uma ferramenta RAD.[19]

Vantagens	Desvantagens
Fácil implementação	Desenvolvimento rápido pode resultar em um produto mal concebido
Melhoria na satisfação do usuário	Necessita de pessoal mais experiente na gerência
Menor prazo para implementação	Necessita de pessoal mais experiente no desenvolvimento

[19] Fonte: (COLEMAN, VERBRUGGEN; 1998).

Entretanto, um ponto a ser considerado e não abordado nos artigos utilizados nesse trabalho são que as ferramentas RAD, utilizadas no desenvolvimento de parte do sistema, tornam o trabalho mais fácil. Com isso, muitas linguagens que utilizam essa metodologia acabam por chamar atenção de novos desenvolvedores por prover a seria de vantagens já descritas. Um bom exemplo é o *iPhone* SDK desenvolvido pela *Apple*, *vide* Figura 8.

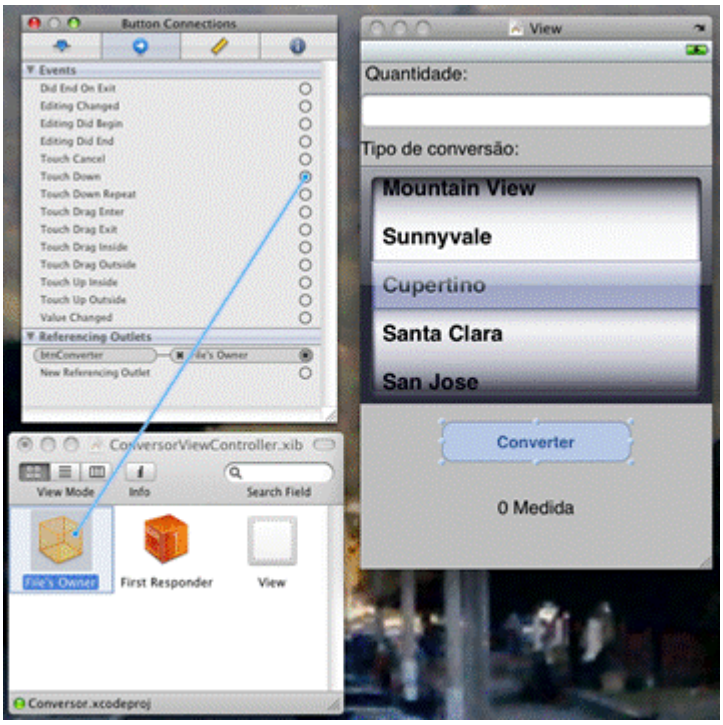

Figura 8 - SDK para desenvolvimento para *iPhone*.[20]

[20] Fonte: (ARAÚJO; 2009)

2.8.1. Netbeans - Visual Mobile Designer 2

Netbeans é uma *Integrated Development Environm*ent (IDE) fornecida pela *Sun Microsystems* para auxiliar o desenvolvedor Java. O *Visual Mobile Designer* 2 (VMD) é uma ferramenta RAD que faz parte da aplicação *Netbeans*. O VMD fornece uma interface gráfica para a elaboração de aplicações em celulares utilizando a plataforma J2ME. Com um simples arrastar e soltar, a pessoa é capaz de elaborar telas e transições complexas em poucos minutos (HAREZLAK; 2009).

O VMD contém vários componentes de interface padrão para o usuário, como caixa de textos, alertas, telas de espera, tela de apresentação, e vários outros componentes visuais. Além disso, o VMD fornece várias ferramentas que representam um conjunto de instrução, como o "*IF*", "*SWITCH*" e chamadas de métodos alternativos (HAREZLAK; 2009). A Figura 9 exibe como os *Screens* de uma aplicação móvel si interagem utilizando o *Visual Mobile Designer 2*.

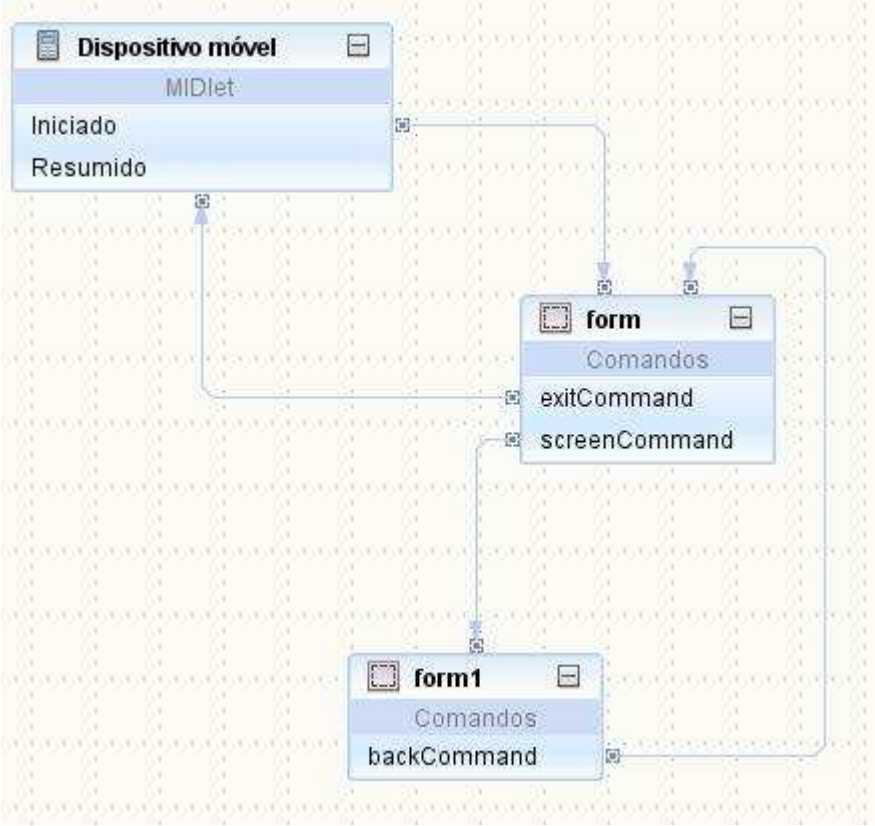

Figura 9 - Fluxo de Telas do *Netbeans Visual Mobile Designer* 2.

2.8.2. e-Gen Developer

O *e-Gen Developer* é uma ferramenta RAD para desenvolvimento de aplicações WEB. Foi desenvolvido totalmente com a linguagem Java e tem como objetivo aumentar a produtividade, reduzir tempo de treinamento e reduzir os custos de manutenção (BECKER).

O *e-Gen* utiliza o navegador padrão para seu funcionamento. Além disso, ele necessita a instalação do container *Apache Tomcat* para seus *Applets* e

classes. Todos os seus códigos são padronizados e qualquer alteração necessária pode ser realizada dentro de seu ambiente (BECKER; 200-).

Segundo Becker, as principais características do *e-Gen* para a construção de suas aplicações são:

- HTML;
- Java Script;
- XML;
- DHTML;
- APIs Java;
- JSP e
- SQL.

De acordo com a página principal do *e-Gen*, a ferramenta fornece recursos para a internacionalização da aplicação gerada, aplicação totalmente de acordo com os padrões do mercado, códigos focados para melhor desempenho, além de outros. A Figura 10 exibe a interface *e-Gen*.

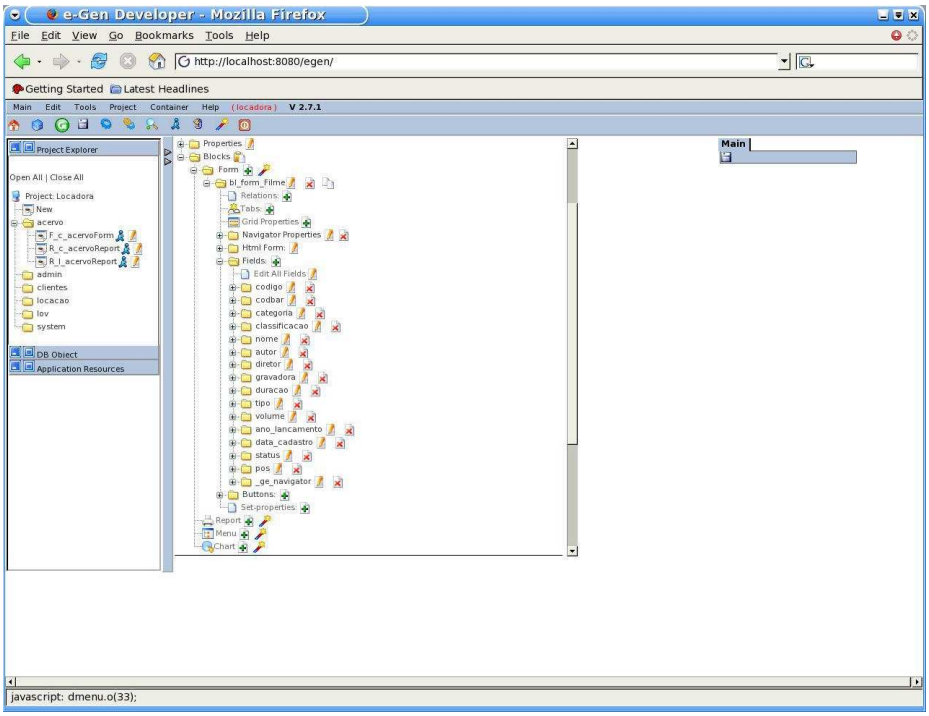

Figura 10 - Interface da aplicação *e-Gen*.[21]

[21] Fonte: (BECKER; 200-)

3. OBJETIVO

O objetivo deste trabalho foi desenvolver um modelo de mapeamento que simbolize quais são os componentes visuais de uma aplicação móvel mais adequados para tabelas e atributos de um banco de dados relacional. Para a validação do mapeamento, foi elaborada uma ferramenta que gera automaticamente o código fonte de uma aplicação para celulares.

3.1. Estrutura

O trabalho foi planejado em três etapas. Primeiramente, foi desenvolvida uma pesquisa sobre os conceitos de banco de dados relacional e sobre o desenvolvimento de ferramentas RAD. A segunda etapa consistiu na elaboração do mapeamento. O desenvolvimento da ferramenta RAD, denominada GACIDIM, que implementa o mapeamento desenvolvido de acordo com a segunda etapa é caracterizado como a terceira e última etapa do trabalho. O intuito do desenvolvimento do GACIDIM foi corrigir e validar o mapeamento elaborado.

4. METODOLOGIA

4.1. Elaboração do Mapeamento

Para elaborar o mapeamento, estudou-se como os dados se relacionam (CHEN; 1976a) e exemplos de diagramas entidade-relacionamento (CHEN; 1976b). Além disso, analisou-se o comportamento de uma aplicação J2ME (JOHNSON; 2007) para aparelhos celulares e *palmtops*, ou seja, como os campos visuais são criados, dispostos na tela, quais informações devem ser consideradas na sua criação e que tipo de dados cada componente visual pode suportar. Analisou-se também o fluxo da aplicação J2ME, e como este fluxo poderia ser representado através do mapeamento. A combinação de todos os estudos possibilitou a forma de elaborar o mapeamento.

Para o entendimento do relacionamento devem ser consideradas algumas definições. Cada tela do aplicativo, ou *Screen* como normalmente chamada, denomina-se de tela-tabela. Isto porque cada tabela selecionada em um banco de dados é representada por uma tela na aplicação. Os componentes visuais possíveis pelo mapeamento foram divididos em cinco categorias: Campo Texto, Campo Data, Campo Seleção Booleana, Caixa de Seleção em Lista e Botão. Todos esses componentes visuais são representados de acordo com a Figura 11.

Figura 11 - Tela-Tabela Pai. Emulador WTK 2.5.

O mapeamento realiza-se através da análise dos metadados[22] de cada atributo das tabelas do banco de dados escolhido. O relacionamento entre as tabelas é necessário na elaboração do fluxo entre as tela-tabelas da aplicação final. Já informações como o tipo do atributo, quantidade de caracteres possíveis, auto-incrementado ou não, são utilizadas para a automação da escolha do componente visual mais adequado para compor a tela-tabela. Para a melhor visualização, o mapeamento elaborado utiliza o diagrama de fluxo de dados.

[22] São dados que representam informações de outros dados.

4.2. Desenvolvimento do GACIDIM

O GACIDIM tem como objetivo a correção e validação do mapeamento proposto e apresentado neste trabalho. A aplicação deve ser executada via WEB, utilizando a linguagem Java. No desenvolvimento de sua interface utilizou-se o *framework* JSF em conjunto com HTML. A aplicação em J2ME gerada pelo GACIDIM poderá ser executa em aparelhos que suportam a configuração CLDC 1.1 ou superior e o perfil MIDP 2.0 e ou superior.

A ferramenta desenvolvida utiliza treze classes para realizar suas funcionalidades. As figuras a seguir representam o diagrama de classes. Não foi possível expressar todo o diagrama em apenas uma imagem, por isso realizou-se uma divisão. Em sequência a Figura 12, Figura 13, Figura 14, Figura 15 e Figura 16 demonstram o diagrama de classes.

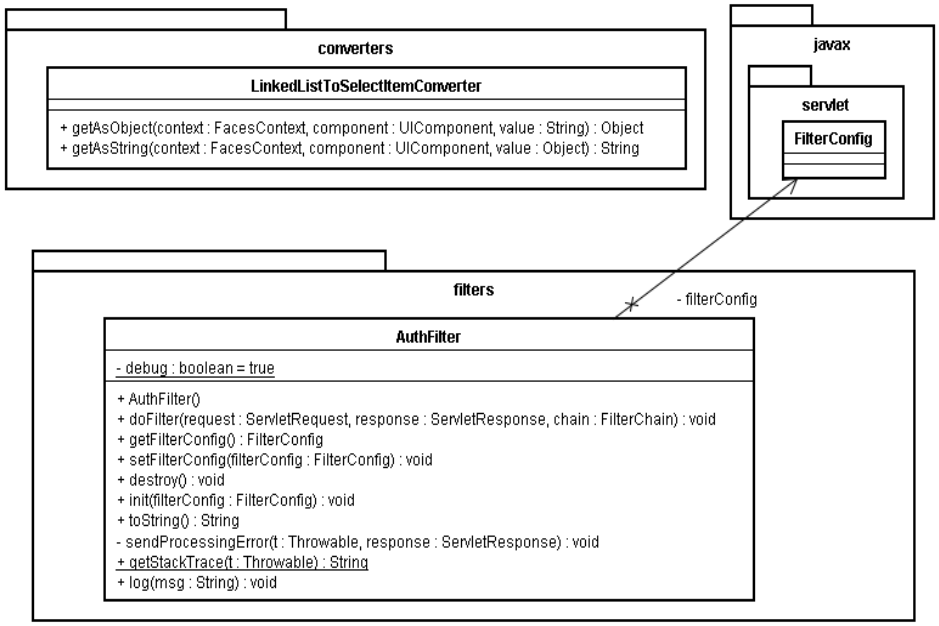

Figura 12 - Diagrama de Classes. Parte A

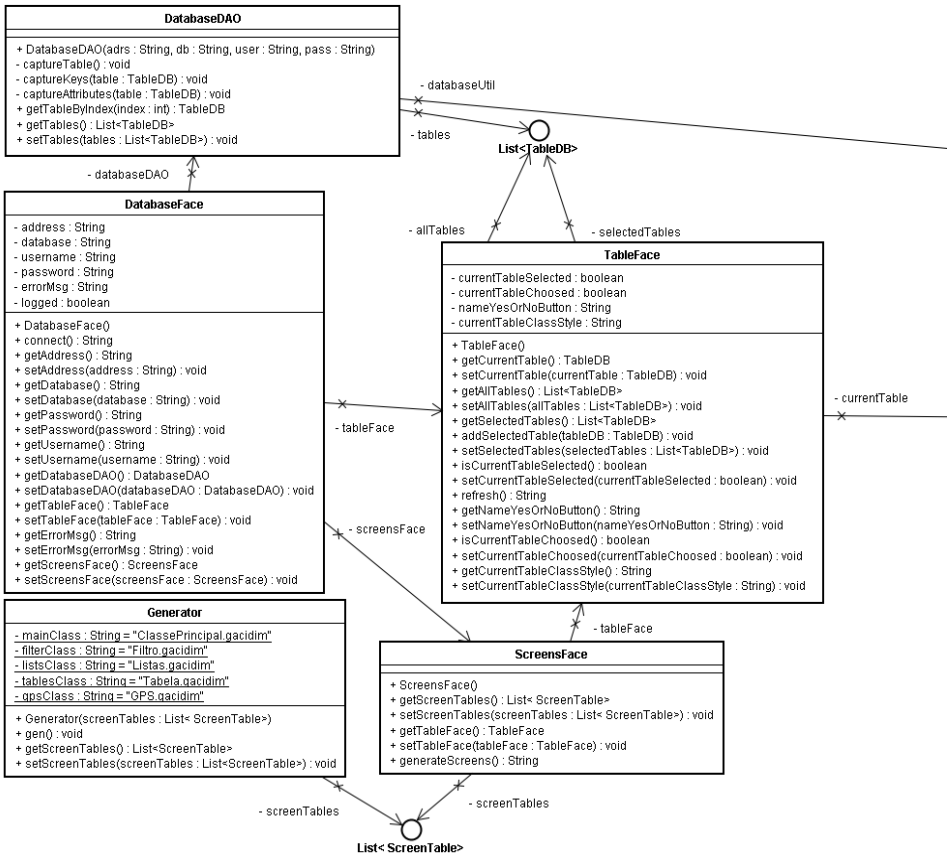

Figura 13 - Diagrama de Classes. Parte B

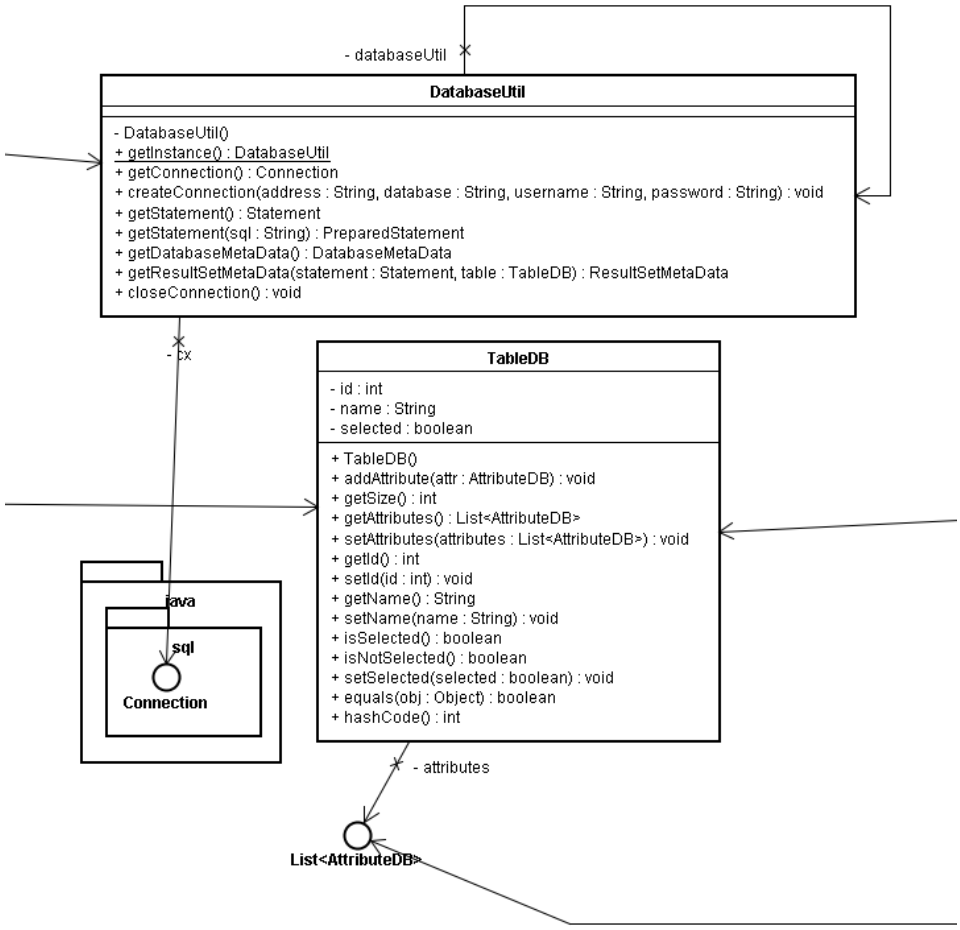

Figura 14 - Diagrama de Classes. Parte C

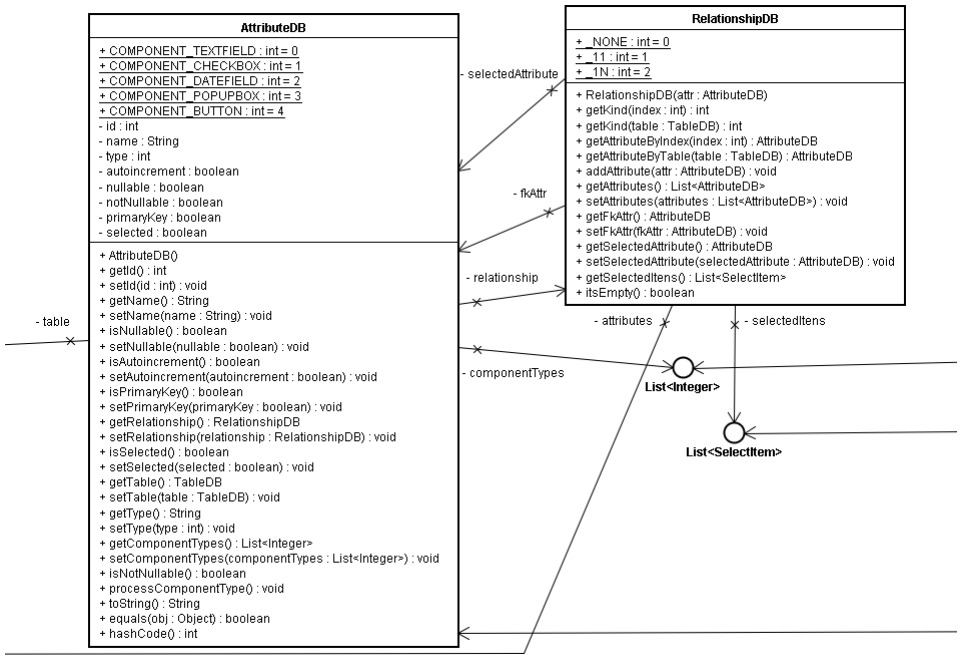

Figura 15 - Diagrama de Classes. Parte D

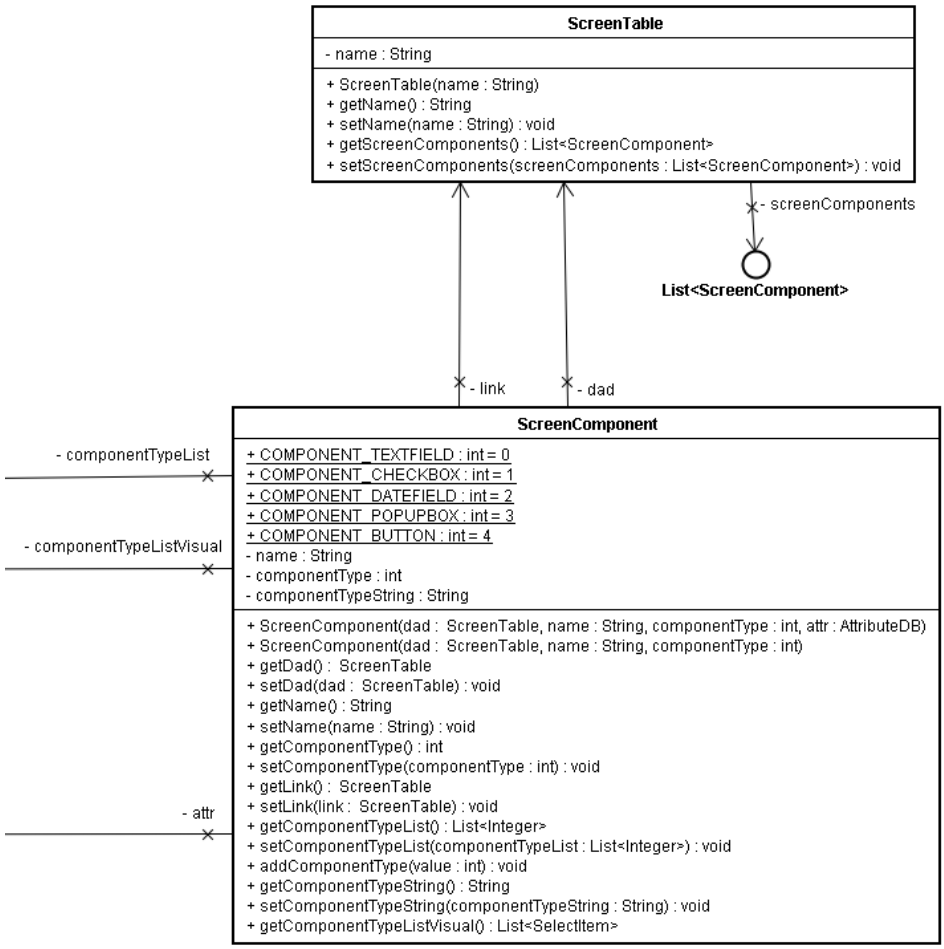

Figura 16 - Diagrama de Classes. Parte E

O princípio básico de funcionamento do GACIDIM é acessar um esquema de um banco de dados MySQL e aplicar o mapeamento elaborado neste trabalho. Com o mapeamento aplicado, cria-se automaticamente uma aplicação em J2ME com todas as informações inseridas pelo usuário. O diagrama de atividades apresentado na Figura 17 representa o processo realizado pelo GACIDIM desde o inicio da aplicação.

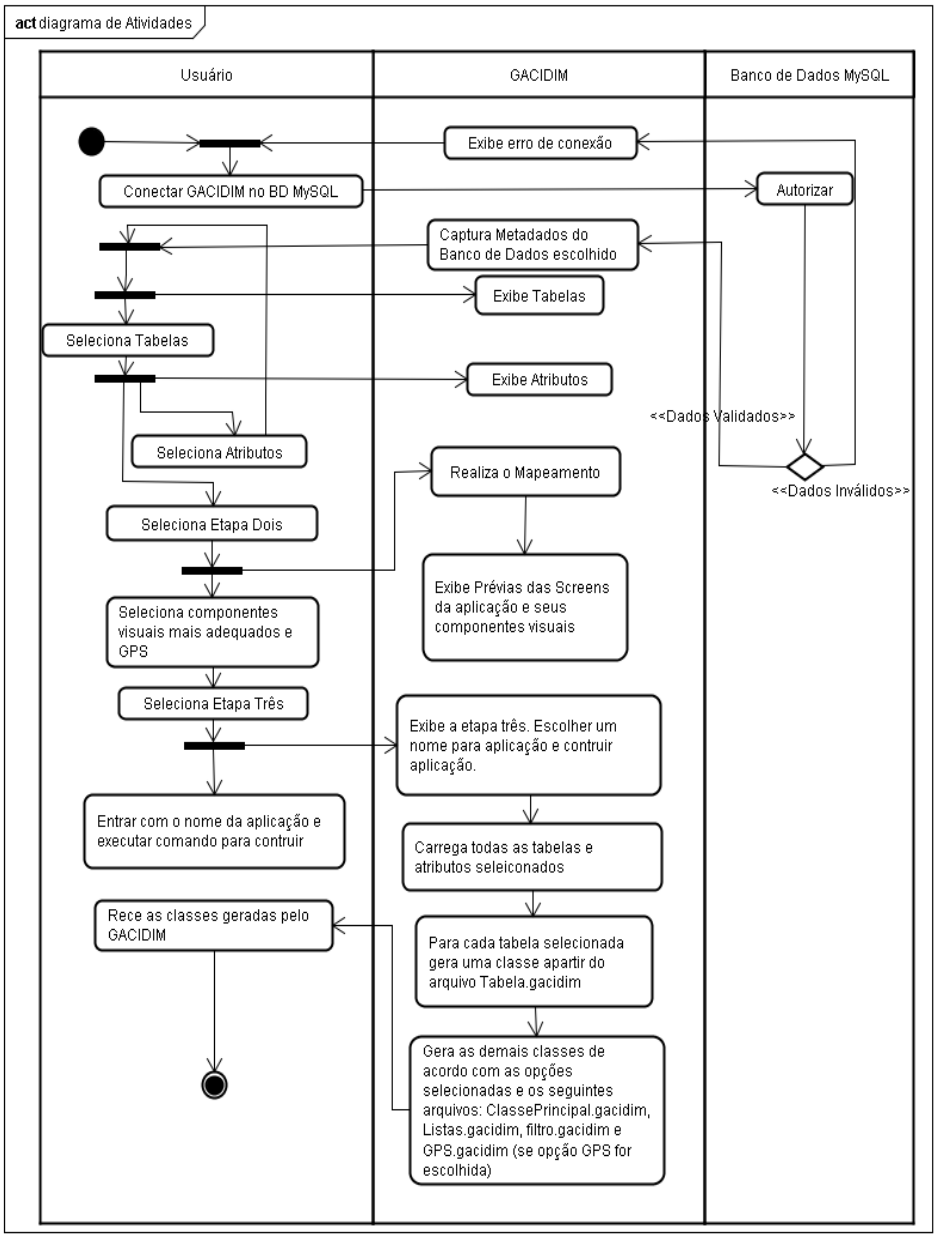

Figura 17 - Diagrama de Atividades do GACIDIM

O GACIDIM fornece ao usuário uma gama de configurações como:

- A escolha de qual tabela e atributo irá fazer parte da aplicação;
- Qual o componente visual mais adequado para um atributo caso este atributo possua mais de uma maneira de ser representado;
- Definir o uso de GPS, e se for optado, deverá ser escolhido quais campos irão obter dados de localização como latitude, longitude, altitude e velocidade e
- Possibilidade de definir o nome da aplicação.

O diagrama de casos de uso, demonstrado pela Figura 18, define quais funcionalidades o GACIDIM fornece pela perspectiva do usuário.

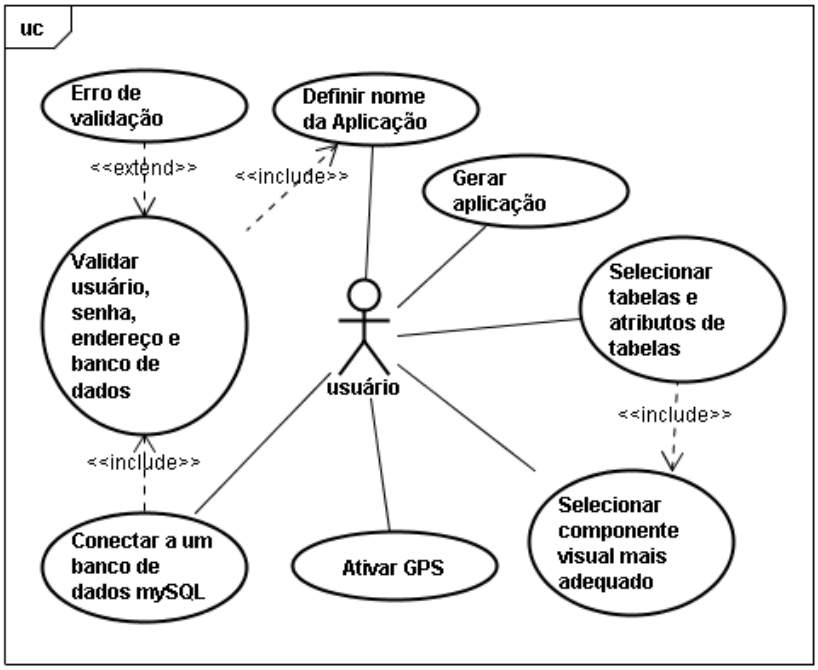

Figura 18 - Diagrama de Casos de Uso

Demais configurações desejadas pelo usuário poderão ser desenvolvidas em cima das classes Java geradas pelo GACIDIM.

Para geração das classes, a ferramenta cria uma estrutura de arquivos composta por dois grupos. O grupo de classes básicas – que sempre fazem parte da aplicação – é composto pelas seguintes classes:

- Classe principal, denominada pelo nome escolhido pelo usuário. É a classe que herda a classe MIDlet, responsável pela execução;
- Classe *Lists* é responsável por diminuir o custo de memória e processamento gasto na aplicação móvel. Evita que sejam criadas várias instâncias de objetos dispensáveis e
- A Classe *Filter* é responsável por filtrar classes quando é utilizado o *framework Floggy* para consulta dos dados armazenados.

O outro grupo é composto pelas classes dinâmicas, que dependem das tabelas selecionadas pelo usuário e pela ativação do GPS:

- Classes tabelas: cada tela-tabela do mapeamento tem uma representação por uma classe que implementa a classe *Persistable* de acordo com a documentação do *Floggy* para ser persistida. Os campos de cada classe são exatamente os mesmos atributos escolhidos da tabela do banco de dados e
- Classe GPS: é responsável por capturar os dados GPS para a aplicação gerada. Essa classe somente é criada se o usuário optar por utilizar GPS na aplicação.

No ato de geração do código, o seguinte padrão foi adotado: cinco arquivos previamente elaborados servem de base para a criação de todas as

classes geradas. São eles: ClassePrincipal.gacidim, Tabela.gacidim, Filtro.gacidim, Listas.gacidim e GPS.gacidim. Esses arquivos possuem uma estrutura bastante parecida com a estrutura de uma classe Java padrão. Porém, foram elaborados dois tipos de *tags*, utilizadas nestes arquivos, para o GACIDIM prover os códigos de acordo com as configurações e opções escolhidas pelos usuários.

A primeira *tag* é demarcada entre < (menor) e > (maior). Sua funcionalidade é demonstrar ao GACIDIM que naquele trecho da classe deverá ser colocado apenas um trecho de código correspondente. Exemplo, a tag: <*nome_aplicação*> simboliza que no trecho em que deverá ser substituída pelo nome apropriado da aplicação.

A segunda *tag* utilizada pelo GACIDIM é demarcada por <* (menor asterisco) e *> (asterisco maior). O objetivo é indicar ao GACIDIM que no trecho correspondente deverão ser acrescentadas várias repetições de código de acordo com as opções selecionadas pelo usuário. Por exemplo, a *tag* <**atributos_tabela**> significa que a *tag* deverá ser substituida pelo código correspondente várias vezes até que todos os atributos da tabela correspondente terminem. O APÊNDICE A apresenta os cinco arquivos utilizados na geração do código.

5. RESULTADOS

5.1. Mapeamento

O mapeamento elaborado sofreu várias alterações no decorrer do desenvolvimento do trabalho devido a erros encontrados em sua elaboração. Primeiramente, era representado apenas por um diagrama que abordava somente os atributos das tabelas de um banco de dados. O diagrama não considerava como uma tabela poderia ser representada. Após a elaboração da ferramenta RAD, verificou-se a necessidade de criar outro diagrama para preceder o primeiro desenvolvido.

O diagrama de mapeamento de Tabela para Tela-Tabela, Figura 19, é bastante simplório. O mapeamento foi elaborado apenas para verificar se as tabelas do banco de dados escolhido foram selecionadas ou não. Se forem selecionadas, cada tabela irá fazer parte da aplicação como uma tela-tabela.

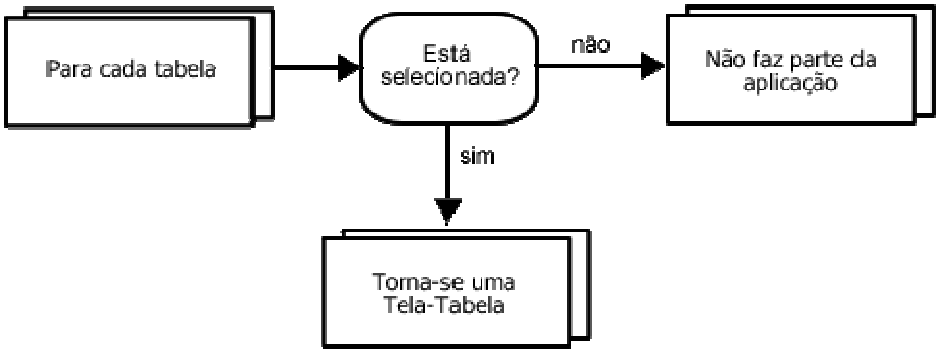

Figura 19 - Mapeamento de Tabela para Tela-Tabela.

Já o mapeamento dos componentes visuais de cada tela-tabela é um pouco mais elaborado. A definição de cada componente visual depende dos seus atributos e relações, *vide* Figura 20.

Alguns atributos possuem mais de um tipo de componente visual disponível. Atributos chave estrangeira podem ser representados por até dois componentes visuais, porém são dispostos em duas tela-tabelas. Já o componente visual Caixa de Seleção em Lista possui como conteúdo inicial uma lista de dados previamente cadastrados no banco de dados utilizado.

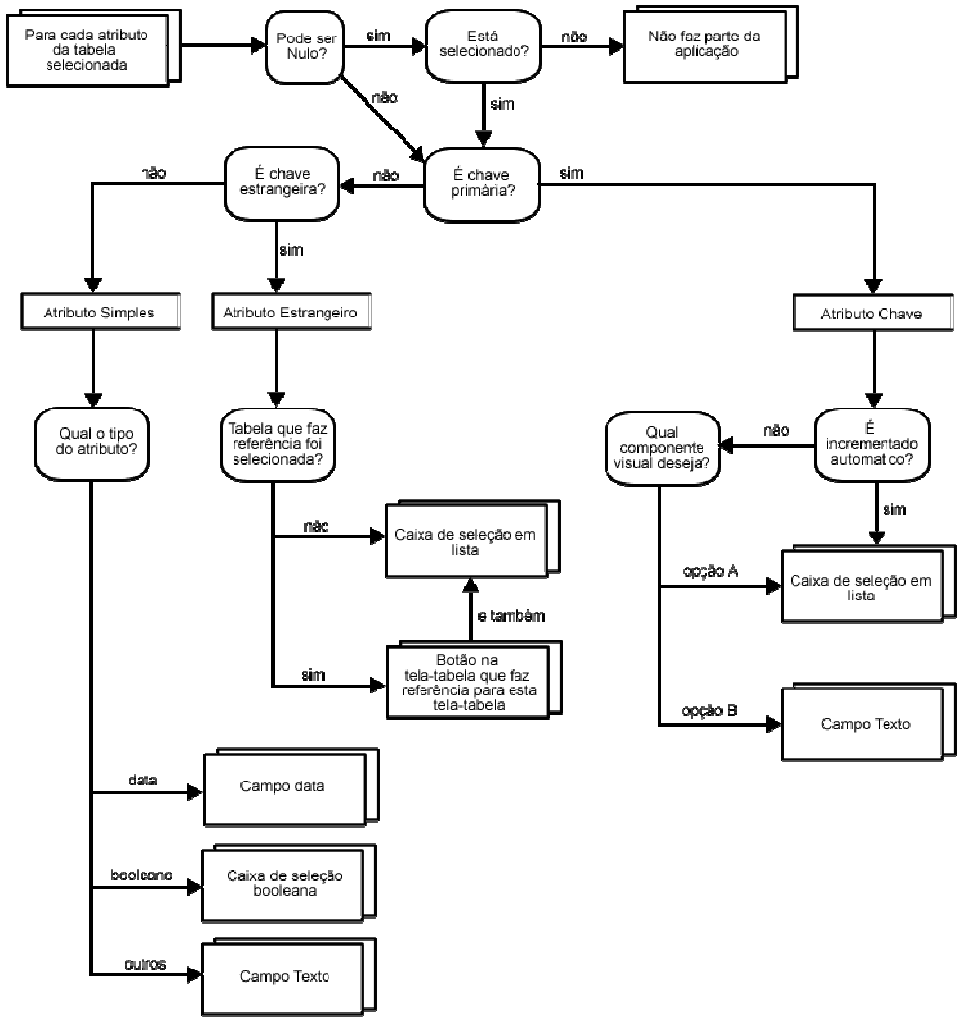

Figura 20 - Mapeamento dos Atributos para Componentes Visuais.

O diagrama representado na Figura 19 deve ser aplicado inicialmente. Após processar todas as tabelas, o diagrama representado na Figura 20 poderá ser aplicado para cada saída de tela-tabela fornecida pelo diagrama da Figura 19.

Para exemplo de utilização do mapeamento, foram elaboradas quatro tabelas para representação dos metadados: Tabela 2, Tabela 4, Tabela 6, Tabela 8 e quatro tabelas que contêm o conteúdo, ou seja, os dados: Tabela 3, Tabela 5, Tabela 7 e Tabela 9. Considera-se que apenas a Tabela Profissão não foi selecionada para aplicação. As demais tabelas geraram suas respectivas tela-tabelas de acordo com o mapeamento fornecido pela Figura 19.

Tabela 2 - Metadados da Tabela Profissão

Nome	Tipo	Chave	Chave Estrangeira	Auto Incrementado	Nulo	Selecionado
Profissão	String	*true*	*false*	*false*	*false*	*false*
Salário	double	*false*	*false*	*false*	*true*	*false*

Tabela 3 - Dados da Tabela Profissão

Profissão	Salário
Advogado	2000,00
Programador	15000,00
Médico	10000,00

Tabela 4 - Metadados da Tabela Pai

Nome	Tipo	Chave	Chave Estrangeira	Auto Incrementado	Nulo	Selecionado
Identidade	String	*true*		*false*	*false*	*true*
Nome	String	*true*		*false*	*false*	*true*
Nasc	Data	*false*		*false*	*true*	*true*
Motorista	Booleano	*false*		*false*	*true*	*true*
Profissão	String	*false*	Tabela Profissão	*false*	*true*	*true*
Idade	Int	*false*		*false*	*true*	*false*

Tabela 5 - Dados da Tabela Pai

Identidade	Nome	Nasc	Motorista	Profissão	Idade
MG12345678	João	12/11/1972	*True*	Programador	Nulo
MG124355766	Mário	25/03/1959	*false*	Advogado	50
MG14443241	José	21/05/1969	*false*	Médico	40

Tabela 6 - Metadados Tabela Filho

Nome	Tipo	Chave	Chave Estrangeira	Auto Incrementado	Nulo	Selecionado
Nome	String	*true*		*false*	*false*	*true*
Idade	Int	*false*		*false*	*true*	*true*
Pai	String	*false*	Tabela Pai	*false*	*false*	*true*

Tabela 7 - Dados Tabela Filho

Nome	Idade	Pai
Pedro	10	João
Augusto	8	Mário

Tabela 8 - Metadados Tabela Brinquedo

Nome	Tipo	Chave	Chave Estrangeira	Auto Incrementado	Nulo	Selecionado
Nome	String	*true*		*false*	*false*	*true*
Valor Pago	Double	*false*		*false*	*true*	*true*
Filho	String	*false*	Tabela Filho	*false*	*false*	*true*

Tabela 9 - Dados Tabela Brinquedo

Nome	Valor Pago	Filho
Comandos em Ação	10,00	Pedro
Lego	35,00	Augusto

Para cada tela-tabela criada foi aplicado o mapeamento descrito na Figura 20. Para exemplificar o processo, um atributo de cada tabela selecionada foi escolhido para demonstrar passo a passo a escolha do seu componente visual mais adequado. Os atributos escolhidos foram o campo Idade da Tabela 4 - Metadados da Tabela Pai, campo Pai da Tabela 6 - Metadados Tabela Filho e o campo Nome da Tabela 8 - Metadados Tabela Brinquedo. As figuras seguintes, Figura 21, Figura 22 e Figura 23, representam passo a passo o processo de mapeamento dos campos de cada tabela.

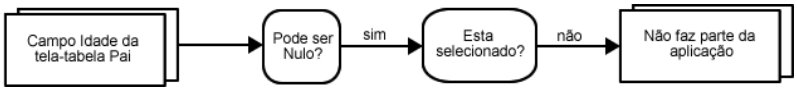

Figura 21 - Mapeamento do campo Idade da tela-tabela Pai

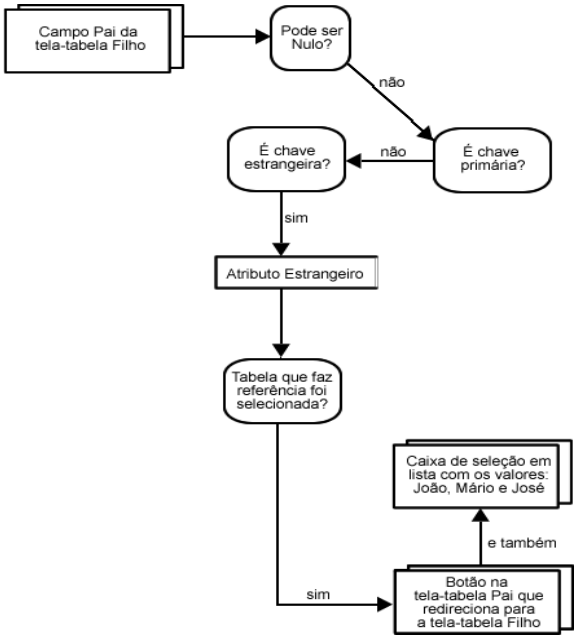

Figura 22 - Mapeamento do campo Pai da tela-tabela Filho

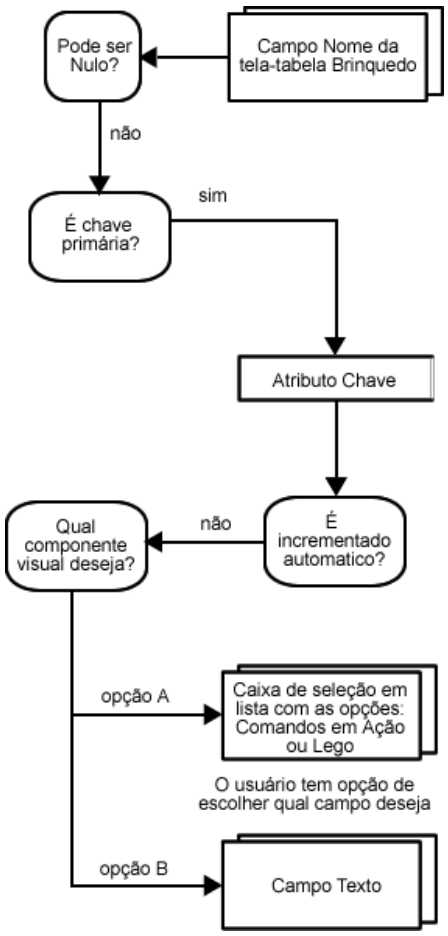

Figura 23 - Mapeamento do campo Nome da tela-tabela Brinquedo

Após o mapeamento de todos os atributos selecionados, as tela-tabelas podem ser visualizadas de acordo com a Figura 24.

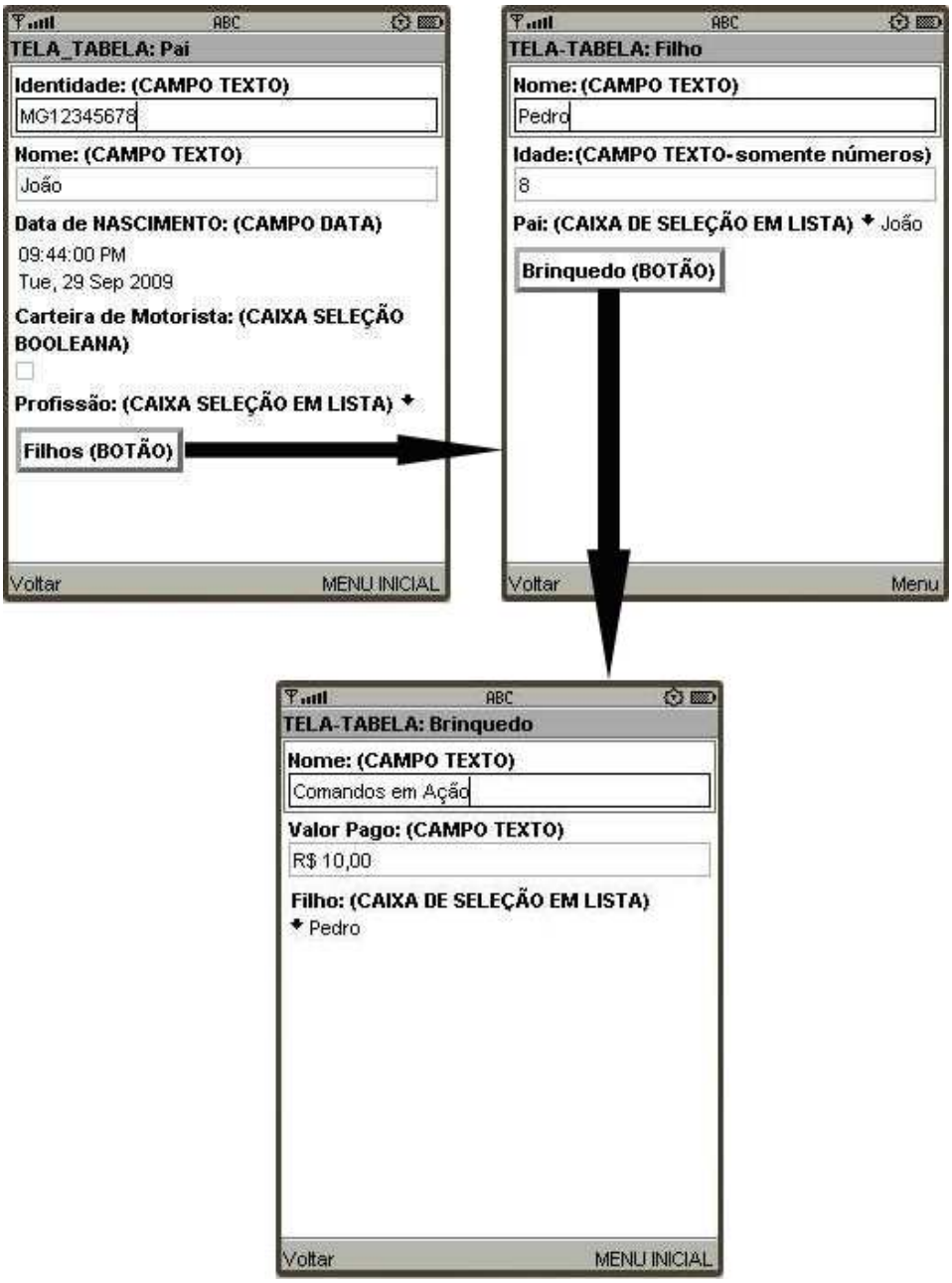

Figura 24 - Tela-tabelas geradas com os respectivos atributos selecionados

5.2. GACIDIM

A Ferramenta GACIDIM funciona através de um fluxo contínuo de telas. Primeiramente, é feita a conexão com a base de dados. Após a conexão, o usuário irá criar a aplicação que deseja em apenas três passos.

O passo um é definido pelo momento em que o usuário tem a exibição de todo o esquema de tabelas à que ele se conectou. A exibição de todos os atributos que compõem cada tabela inclui em suas descrições quais os relacionamentos que cada um proporciona com outra tabela. É neste passo que o usuário irá definir quais tabelas ira fazer parte da aplicação. Ao definir uma tabela, será possível para o usuário definir quais atributos da tabela irão fazer parte da aplicação. Esclarecendo que atributos chaves primárias e atributos não nulos não fornecem o poder de escolha de fazer parte da aplicação ao usuário. Esses atributos automaticamente irão compor a aplicação caso a tabela a que pertencem for selecionada.

O passo dois é caracterizado pela configuração dos componentes visuais da aplicação. A divisão de tabelas exibida pela ferramenta RAD no passo dois simboliza cada tela-tabela do aplicativo com os seus respectivos componentes visuais. Há componentes que proporcionam mais de uma opção de componente visual, e é neste contexto que o usuário do GACIDIM tem a possibilidade de escolher qual o componente mais adequado para sua aplicação. O segundo passo também é caracterizado pela a opção de utilizar o sistema GPS na aplicação. Sua escolha é feita por tela-tabela. Se a tela-tabela der suporte a GPS, então todos atributos passíveis de serem campos de coordenadas GPS (latitude, longitude, altitude e velocidade) podem ser configurados para receberem o tipo de dados coletado pelo receptor GPS mais adequado.

Já a passo três simbolizada pela definição do nome da aplicação e pelo botão de ação para a construção das classes. Informações importantes para compilação do código a ser gerado são representadas no terceiro passo. O código final gerado deverá ser compilado em conjunto com o *weaver* produzido pela equipe do *framework Floggy*. Isto porque o *weaver* é responsável por inserir os códigos que abstraem a persistência de dados do aplicativo gerado no aparelho móvel.

O GACIDIM é composto apenas por quatro páginas básicas, com exceção da página utilizada para demonstrar os erros de execução. Primeiramente é apresentada uma página para o usuário realizar a conexão no banco de dados sobre o qual deseja criar a aplicação móvel. Deve-se levar em consideração que o usuário tenha uma conta e senha com privilégios suficientes para ler todas as tabelas do banco escolhido. *Vide* Figura 25.

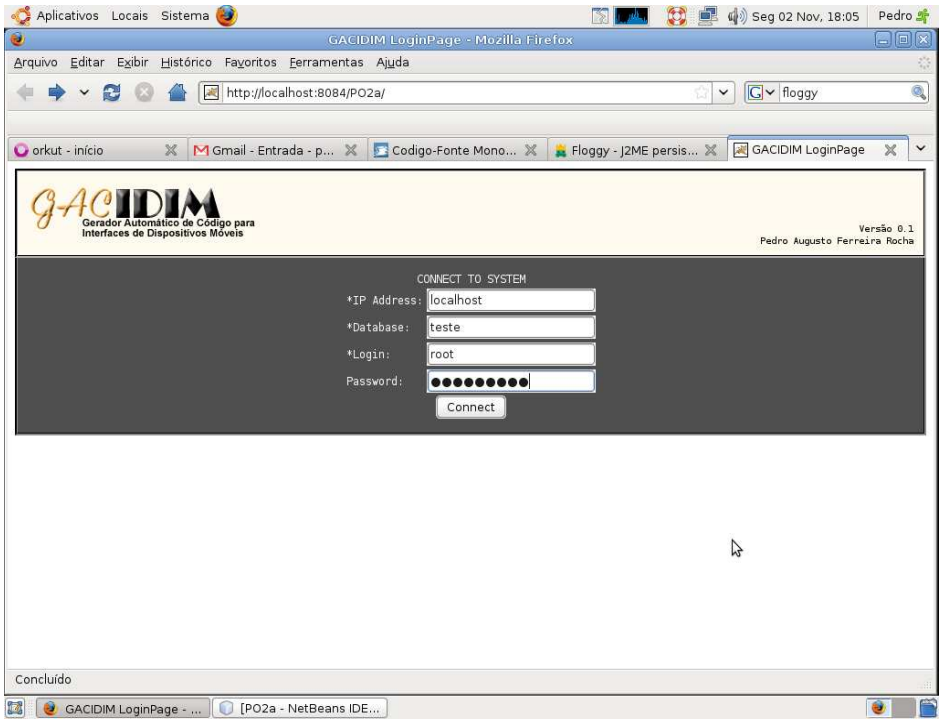

Figura 25 - Página inicial da aplicação.

Na segunda página, representada pelo primeiro passo, o usuário tem à disposição no lado esquerdo a lista de tabelas do banco conectado. Cada tabela com o campo de seleção marcado ao lado do seu nome simboliza que este foi escolhido para fazer parte da aplicação. Para exibir os atributos de cada tabela, deve-se clicar em cima de seu nome. Os atributos só poderão ser selecionados se a tabela for selecionada pelo botão exibido logo abaixo da tabela. Para seguir para o próximo passo basta clicar no item correspondente no *menu* superior. *Vide* Figura 26.

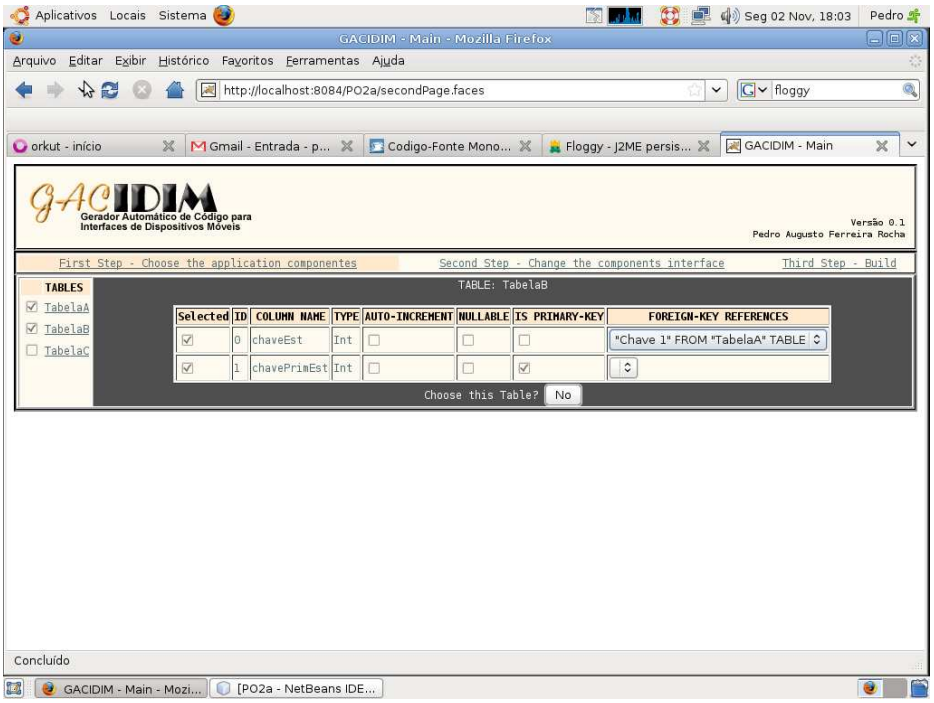

Figura 26 - Passo um: escolha das tabelas e atributos.

A terceira página é representada o passo dois e consiste na exibição simbólica de cada tela-tabela que irá compor a aplicação final gerada pelo GACIDIM. O usuário que deseja utilizar GPS deverá marcar o campo denominado por *Active* GPS. *Vide* Figura 27.

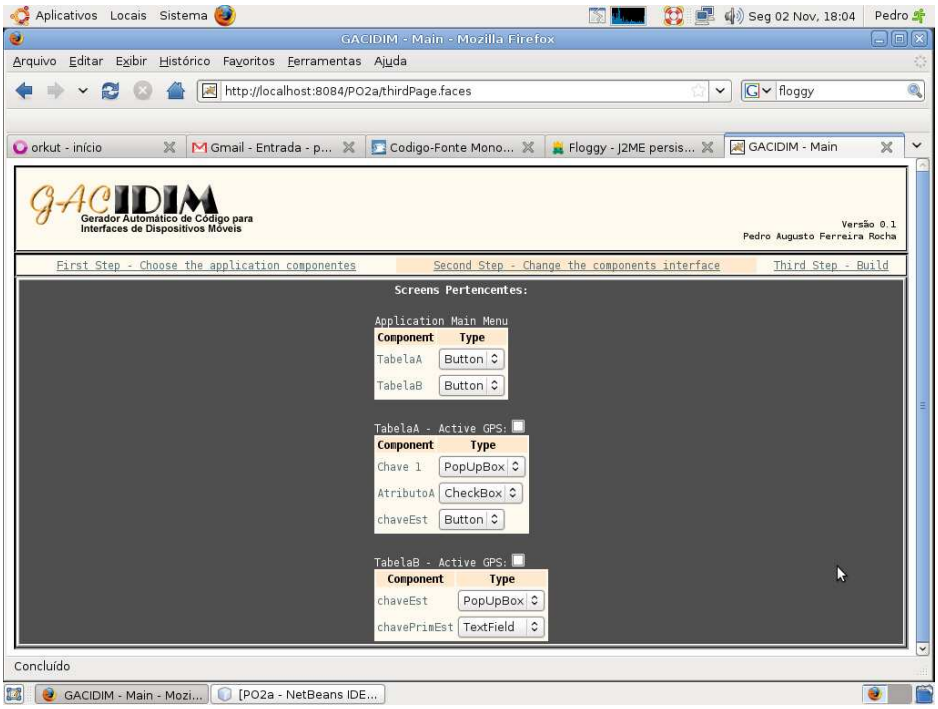

Figura 27 - Passo dois: escolha dos componentes visuais e GPS.

A Figura 28 exibe a quarta página simbolizada pelo passo três, em que o usuário deve definir o nome da aplicação e realizar a construção da aplicação final.

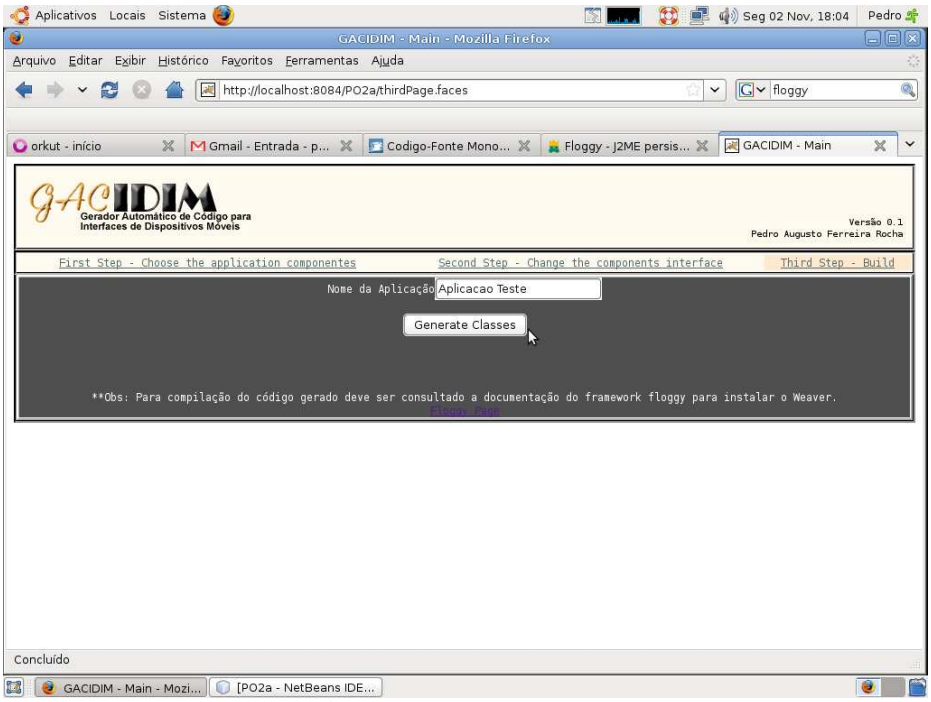

Figura 28 - Passo três: construção.

A implementação do mapeamento é realizada por vários trechos de códigos não contínuos no GACIDIM. Porém, uma grande parte do mapeamento de atributos para componentes visuais pode ser observada de acordo com o trecho de código a seguir.

```
1.  if (selected) {
2.    if (primaryKey) {
3.      if (autoincrement) {
4.        componentTypes.add(COMPONENT_POPUPBOX);
5.      } else {
6.        componentTypes.add(COMPONENT_TEXTFIELD);
7.        componentTypes.add(COMPONENT_POPUPBOX);
```

```
8.       }
9.     } else {
10.      if (relationship.itsEmpty()) {
11.        if (type == 91 || type == 92 || type == 93) {
12.          componentTypes.add(COMPONENT_DATEFIELD);
13.        } else if (type == -7) {
14.          componentTypes.add(COMPONENT_CHECKBOX);
15.        } else {
16.          componentTypes.add(COMPONENT_TEXTFIELD);
17.        }
18.      } else {
19.          boolean referencedTableSelected = false;
20.          for (AttributeDB attr : relationship.getAttributes()) {
21.            if (attr.getTable().isSelected()) {
22.              referencedTableSelected = true;
23.            }
24.          }
25.          if (referencedTableSelected) {
26.            componentTypes.add(COMPONENT_POPUPBOX);
27.            componentTypes.add(COMPONENT_BUTTON);
28.          } else {
29.            componentTypes.add(COMPONENT_POPUPBOX);
30.        }
31.      }
32.    }
33.  }
```

6. CONCLUSÃO

O modelo de mapeamento proposto sofreu muitas mudanças desde o começo do desenvolvimento deste trabalho devido a erros encontrados em sua primeira versão. Porém, seu resultado final foi bastante satisfatório. A metodologia desenvolvida em sua elaboração mostrou-se eficiente ao propor o desenvolvimento da ferramenta RAD para a validação e correção. O GACIDIM foi impactante na diagramação final do mapeamento.

A grande dificuldade encontrada no desenvolvimento deste projeto foi justamente a elaboração da ferramenta RAD. Gastou-se muito tempo no estudo de como elaborá-la e a implementação foi bastante custosa. Porém, os resultados foram positivos.

6.1. Trabalhos Futuros

Alguns trabalhos futuros que poderão ser desenvolvidos em cima do mapeamento são:

- Trabalhos que utilizam o mapeamento para alguma finalidade;
- Aperfeiçoamento do mapeamento e
- Elaboração de mapeamento de interface para aplicações em diferentes arquiteturas como, *desktop* e televisão digital.

Trabalhos relacionados com o GACIDIM também poderão ser elaborados, são:

- Melhoria no processo de geração automática do código;
- Aplicar o mapeamento de uma maneira mais eficiente;

- Suportar outros banco de dados relacional além do MySQL;
- Verificar a consistência do banco de dados com as onze regras elaboradas por Chen (1976b) e
- Abranger um número maior de aparelhos que suportem a aplicação gerada.

7. REFERÊNCIAS BIBLIOGRÁFICAS

ARAÚJO, R. P., Introdução ao desenvolvimento de aplicações para iPhone com iPhone SDK. Webmobile 23:DevMedia, p. 45-54, 2009.

BECKER, V. H., Um Estudo sobre a Ferramenta EGEN Developer, Porto Alegre:UFRGS, 9 p.; [200-].

BULUSU, N.; HEIDEMANN, J.; ESTRIN, D., GPS-less low Cost Outdoor Localization for Very Small Devices, IEEE Personal Communications Magazine, p. 1-7, out. 2000.

CÂMARA, G.; Anatomia de Sistemas de Informação Geográfica; Rio de Janeiro: SBC, 193 p., 1996. Disponível em: <http://www.dpi.inpe.br/gilberto/livro/anatomia.pdf>. Acesso em: 06 nov. 2009.

CHEN, P.P., The Entity-Relationship Model: Toward a Unified View of Data, ACM Transactions on Database Systems, p. 1-36, mar. 1976a.

CHEN, P.P, English Sentence Structure and Entity-Relationship Diagrams, ACM Transactions on Database Systems, p. 127-149, mar. 1976b.

COLEMAN, G.; VERBRUGGEN, R., A Quality Software Process for Rapid Application Development, Software Quality Jornal, p. 107-122, 7 abr. 1998.

DEITEL, H. M.; DEITEL, P. J., Java How To Program, 6. ed., Pearson, 1110 p., 2006.

E-GEN, Disponível em: <http://www.egen.com.br>. 2006, Apresenta informações sobre a ferramenta e-Gen Developer, Acesso em: 22 jul. 2009.

FLOGGY, Disponível em: <http://floggy.sourceforge.net>. 2006-2009, Apresenta informações sobre o framework Floggy. Acesso em: 01 nov. 2009.

HAARTSEN, J. C., Ericsson Radio System B.V., The Bluetooth Radio System, IEEE Personal Communications, p. 28-36, fev. 2000.

HAREZLAK, K., Componentes Personalizados do Visual Mobile Designer: Navegador de Arquivo, Disponível em: <http://www.netbeans.org/kb/60/mobility/filebrowser_pt_BR.htm>. Acesso em: 21 jul. 2009.

JOHNSON, T. M., Java para Dispositivos Móveis, 1. ed., Novatec, p. 25-36, 2007.

KENTERIS, M., GAVALAS, D., ECONOMOU, D., An Innovative Mobile Eletronic Tourist Guide Application, Pers Ubiquit Comput, p. 103-118., 2009.

KUHNEN, A., Protótipo de uma Aplicação LBS utilizando GPS Conectado em Celulares para Consultar dados Georeferenciados, Universidade Regional de Blumenal, 61 p., 2003.

MAHMOUD, Q. H., J2ME and Location-Based Services, Disponível em: <http://developers.sun.com/mobility/apis/articles/location/>. Acesso em: 01 nov. 2009.

MARTIN, J.; Rapid Application Development, Macmillan, 1991.

MONICO, J. F. G., Posicionamento pelo NAVSTAR-GPS: Descrição, Fundamentos e Aplicações, 1. ed., UNESP: Presidente Prudente Editora, 21 p., 2000.

REILLY, J. P.; Does RAD live up to the hype?, IEEE Software, p. 24-26, set. 1995.

SCHILLER, J. H.; VOISARD, A., Location-Based Services, Elsevier, 2004.

SUN MICROSYSTEMS, The Java EE 5 Tutorial, Santa Clara, USA, 1126 p., out. 2008.

VEIT, M.; HERMANN, S., Model-View-Controller and Object Teams: A Perfect Match of Paradigms, ACM Transactions on Database Systems: New York, USA, p. 140-149, 2003.

WELLENOF-HOFMANN, B., LICHTENEGGER, H., COLLINS, J.; Global Positioning System: Theory and Pratice, 4. ed, Spring-Verlag, 1997

APÊNDICE A

ClassePrincipal.gacidim

```
1.  import java.util.Vector;
2.  import javax.microedition.lcdui.*;
3.  import javax.microedition.midlet.*;
4.  import net.sourceforge.floggy.persistence.*;
5.
6.  public class <nome_aplicacao> extends MIDlet implements CommandListener,
    ItemCommandListener {
7.    private Display display;
8.    private Vector sequencia_de_telas = new Vector();
9.    private PersistableManager pm =
      PersistableManager.getInstance();
10.   private Filter filter = new Filter(this);
11.   private ObjectSet os;
12.   private Lists lists = Lists.getInstance();
13.   private Alert alert = new Alert("Atenção", "Todos os
      campos com asterisco(*) devem ser preenchidos!", null,
      AlertType.WARNING);
14.   private Command menuInicial = new Command("Menu Inicial",
      Command.BACK, 1);
15.   private Command voltar = new Command("Voltar",
      Command.BACK, 0);
16.   private Command adicionar = new Command("Adicionar e
      Salvar", Command.OK, 0);
17.   private Command finalizar = new Command("Finalizar",
```

```
        Command.OK, 0);
18.     private String[] _00001_list = lists.getArray00001_list();
19.     private List _00001;
20.     <GPS>
21.     <*tela-tabelas*>
22.     <*componentes_visuais*>
23.
24.     public Aplicacao() {
25.       <*instanciação_componentes_visuais*>
26.       _00001 = new List(<nome_aplicacao>, List.IMPLICIT,
          _00001_list, null);
27.       _00001.setCommandListener(this);
28.       <*instanciacao_tela_tabela*>
29.     }
30.
31.     public void startApp() {
32.       display = Display.getDisplay(this);
33.       display.setCurrent(_00001);
34.     }
35.
36.     public void pauseApp() {
37.     }
38.
39.     public void destroyApp(boolean unconditional) {
40.       this.notifyDestroyed();
41.     }
42.
43.     public void cleanScreens() {
```

```
44.       <*limpar_componentes*>
45.    }
46.
47.    public boolean save(Displayable d) {
48.       <*salvar_telas_tabela*>
49.       System.gc();
50.       return false;
51.    }
52.
53.    public void commandAction(Command c, Displayable d) {
54.       if (c == finalizar) {
55.          if (!save(display.getCurrent())) {
56.             gotoAlert();
57.          } else {
58.             display.setCurrent((Displayable)
59.             sequencia_de_telas.lastElement());
60.             sequencia_de_telas.removeElement(
                sequencia_de_telas.lastElement());
61.          }
62.       }
63.       if (c == menuInicial) {
64.          gotoooooo1();
65.       }
66.       if (c == voltar) {
67.          display.setCurrent(
68.          (Displayable)sequencia_de_telas.lastElement());
69.          sequencia_de_telas.removeElement(
             sequencia_de_telas.lastElement());
```

```
70.       }
71.       if (c == List.SELECT_COMMAND && d == _00001) {
72.          cleanScreens();
73.          sequencia_de_telas.addElement(_00001);
74.          switch (_00001.getSelectedIndex()) {
75.             <*goto_tela-tabela_pelo_menu_inicial*>
76.          }
77.       }
78.    }
79.
80.    public void commandAction(Command c, Item item) {
81.       if (c == adicionar) {
82.          if (save(display.getCurrent())) {
83.             <*goto_tela-tabela_referenciada*>
84.          } else {
85.             gotoAlert();
86.          }
87.       }
88.    }
89.
90.    public void gotoAlert() {
91.       display.setCurrent(alert, display.getCurrent());
92.    }
93.
94.    public void gotoooooo1() {
95.       cleanScreens();
96.       sequencia_de_telas.removeAllElements();
97.       display.setCurrent(_00001);
```

```
98.    }
99.    <*goto_tela-tabela_metodos*>
100.   <*get_tela-tabela_metodos*>
101. }
```

Filtro.gacidim

```
1.  import javax.microedition.lcdui.Displayable;
2.  import net.sourceforge.floggy.persistence.Persistable;
3.
4.  public class Filter implements
    net.sourceforge.floggy.persistence.Filter {
5.      private Aplicacao main;
6.      private Displayable current;
7.      private Object obj;
8.
9.      public Filter(Aplicacao main) {
10.         this.main = main;
11.     }
12.
13.     public void set(Displayable current, Object obj) {
14.         this.current = current;
15.         this.obj = obj;
16.     }
17.
18.     public boolean matches(Persistable persistable) {
19.         <*filter_tela-tabela*>
20.         return false;
21.     }
22. }
```

Listas.gacidim

```
1.  import java.util.Vector;
2.  import net.sourceforge.floggy.persistence.*;
3.
4.  public class Lists implements Persistable {
5.      <*listas_da_aplicacao*>
6.
7.      public Lists() {
8.      }
9.
10.     private Lists(boolean alwaysTrue) {
11.         alwaysTrue = true;
12.         PersistableManager PM;
13.         PM = PersistableManager.getInstance();
14.         ObjectSet os;
15.         try {
16.             os = pm.find(Lists.class, null, null);
17.             if (os.size() > 0) {
18.                 Lists lists = (Lists) os.get(0);
19.                 copy(lists);
20.             } else {
21.                 <*instanciacao_das_listas*>
22.                 <*preencher_listas_com_valores_iniciais*>
23.                 pm.save(this);
24.             }
25.         } catch (FloggyException ex) {
26.             ex.printStackTrace();
```

```
27.         }
28.     }
29.
30.     public static Lists getInstance() {
31.         Lists toReturn = new Lists(true);
32.         return toReturn;
33.     }
34.
35.     public void copy(Lists lists) {
36.         set00001_list(lists.get00001_list());
37.         set00003_00001_list(lists.get00003_00001_list());
38.     }
39.
40.     <*add_listas_metodo*>
41.     <*set_listas_metodo*>
42.     <*set_listas_array_metodo*>
43.     <*get_listas_metodo*>
44.     <*get_listas_array_metodo*>
45. }
```

Tabela.gacidim

```
1.  import net.sourceforge.floggy.persistence.Persistable;
2.
3.  public class <nome_tabela> implements Persistable {
4.      <*atributos_tabela*>
5.      <*set_atributos_metodo*>
6.      <*set_atributos_double_metodo*>
7.      <*get_atributos_metodo*>
8.
9.      public boolean equals(Object obj) {
10.         <nome_tabela> aux = (<nome_tabela>) obj;
11.         if(<atributo_chave>.equals(
            <nome_tabela>.<get_atributo_chave_metodo>)) {
          return true;
12.         }
13.         return false;
14.     }
15. }
```

GPS.gacidim

```
1.  import javax.microedition.location.*;
2.  import javax.microedition.lcdui.TextField;
3.
4.  public class GPS extends Thread {
5.      private LocationProvider locProvider;
6.      private TextField itemLat;
7.      private TextField itemLong;
8.      private TextField itemAlt;
9.      private TextField itemSpeed;
10.
11.     public void capturarCoordenadas(TextField itemLat,
        TextField itemLong, TextField itemAlt, TextField itemSpeed) {
12.         this.itemLat = itemLat;
13.         this.itemLong = itemLong;
14.         this.itemAlt = itemAlt;
15.         this.itemSpeed = itemSpeed;
16.         setOpcoes();
17.         start();
18.     }
19.
20.     public void setOpcoes() {
21.         try {
22.             locProvider = LocationProvider.getInstance(null);
23.         } catch (LocationException ex) {
24.             ex.printStackTrace();
25.         }
```

```
26.    }
27.
28.    public void run() {
29.      try {
30.        Location loc = locProvider.getLocation(5000);
31.        if (loc != null && loc.isValid()) {
32.          if(itemLat != null) {
33.            itemLat.setString(
               loc.getQualifiedCoordinates().getLatitude()+"");
34.          }
35.          if(itemLong != null) {
36.            itemLong.setString(
               loc.getQualifiedCoordinates().getLongitude()+"");
37.          }
38.          if(itemAlt != null) {
39.            itemAlt.setString(
               loc.getQualifiedCoordinates().getAltitude()+"");
40.          }
41.          if(itemSpeed != null) {
42.            itemSpeed.setString(loc.getSpeed()+"");
43.          }
44.        }
45.      } catch (LocationException ex) {
46.        ex.printStackTrace();
47.      } catch (InterruptedException ex) {
48.        ex.printStackTrace();
49.      }
50.    }
```

```
51.  }
```