UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

FLOGGY:
FRAMEWORK DE PERSISTÊNCIA PARA J2ME/MIDP

PRISCILA T. LUGON
THIAGO ROSSATO

FLORIANÓPOLIS
Outubro de 2005

PRISCILA T. LUGON
THIAGO ROSSATO

FLOGGY:
FRAMEWORK DE PERSISTÊNCIA PARA J2ME/MIDP

Trabalho de conclusão de curso apresentado à Universidade Federal de Santa Catarina, como parte dos requisitos para a obtenção do grau de bacharel em Sistemas de Informação.

Prof. Frank Siqueira

FLORIANÓPOLIS
Outubro de 2005

PRISCILA T. LUGON
THIAGO ROSSATO

FLOGGY:
FRAMEWORK DE PERSISTÊNCIA
PARA J2ME/MIDP

Monografia aprovada em ___/___/2005, como requisito para a obtenção do grau de bacharel em Sistemas de Informação.

Banca Examinadora

_____
Professor Frank Augusto Siqueira
Orientador

_____
Professor Mário Antônio Ribeiro Dantas
Membro

_____
Rodrigo Campiolo
Membro

"Os computadores são incrivelmente rápidos, precisos e burros; os homens são incrivelmente lentos, imprecisos e brilhantes; juntos, seu poder ultrapassa os limites da imaginação."

Albert Einstein

Agradecimentos

A Deus, pois sem Ele nada é possível.

Aos pais pelo incentivo, apoio e compreensão.

Ao orientador Frank Siqueira, que esteve ao nosso lado durante todo este desafio.

Ao co-orientador Thiago Leão Moreira, mais conhecido como Timba, que participou ativamente e acolheu com carinho este projeto. Durante todo o trajeto se demonstrou um grande parceiro.

Ao Rodrigo Campiolo que, mesmo estando distante de Florianópolis, sempre contribuiu para o engrandecimento de nosso trabalho. E a Cly, que o representou em nossa defesa e contribuiu para a correção de nosso texto.

Aos nossos cachorrinhos, Mel e Lord, que sempre nos alegravam nos momentos de dificuldade.

Enfim, agradecemos a todos que de alguma maneira contribuíram para este trabalho.

Sumário

Resumo

Os avanços tecnológicos e o crescente uso de dispositivos móveis demandam aplicações cada vez maiores e complexas. Essas aplicações necessitam armazenar uma quantidade maior de informações. Os sistemas operacionais dos dispositivos fornecem recursos escassos para persistência dos dados e os poucos bancos de dados existentes são proprietários e disponíveis somente para determinados tipos de dispositivos. Estas restrições refletem durante o processo de desenvolvimento de uma aplicação para dispositivos móveis, pois a forma como os dados serão armazenados torna-se um trabalho complexo para o engenheiro de software. Uma solução possível é a introdução de uma camada de persistência, responsável por encapsular toda a lógica necessária para que os dados de um objeto sejam salvos ou recuperados de um repositório. O uso de uma camada de persistência é muito adotado no desenvolvimento de aplicativos para plataformas tradicionais. Entretanto as camadas de persistência para estas plataformas são demasiadas complexas para o ambiente de execução em um dispositivo móvel. Neste contexto, este trabalho propõe a construção de um *framework* de persistência de objetos para aplicações desenvolvidas utilizando J2ME/MIDP, que é uma das plataformas mais utilizadas no desenvolvimento de aplicações para dispositivos móveis. Este *framework*, denominado *Floggy*, visa abstrair os detalhes de persistência do engenheiro de software, ficando este responsável apenas pela modelagem lógica da aplicação.

Abstract

The technological advances and the increasing use of mobile devices demand more complex applications. These mobile applications need to store a large amount of data. The operational systems for mobile devices supply scarce resources for data persistence and the few databases are available only for some types of devices. The resource restrictions affect the development of an application for mobile devices, because programming data storage will become a complex work for the software engineer. The solution is the introduction of a persistence layer, responsible for encapsulating all the logic that supports objects being stored and loaded from a repository. Persistence layers are often adopted in the development of applications for traditional platforms. However, these persistence layers are exaggerated complex for the mobile environment.

This work considers the creation of an object persistence framework, for the J2ME/MIDP technology. This is the most used technology to develop applications for mobile devices. The framework, called Floggy, aims to abstract the data persistence details from the software engineer (only responsible for the application logic).

Lista de Figuras

Lista de Tabelas

Lista de Exemplos

Glossário de Acrônimos

API	*Application Programming Interface*
AWT	*Abstract Windows Toolkit*
BD	Banco de Dados
CDC	*Connected Device Configuration*
CLDC	*Connected, Limited Device Configuration*
CVM	*Compact Virtual Machine*
FP	*Foundation Profile*
GPS	*Global Positioning System*
HQL	*Hibernate Query Language*
IDP	*Information Device Profile*
J2EE	*Java 2, Enterprise Edition*
J2ME	*Java 2, Micro Edition*
J2SE	*Java 2, Standard Edition*
JAD	*Java Application Descriptor*
JAR	*Java Archive*
JCP	*Java Community Process*
JDBC	*Java Database Connectivity*
JDO	*Java Data Objects*
JSR	*Java Specification Request*
JVM	*Java Virtual Machine*
KB	*Kilobyte(s)*
KVM	*Kilobyte Virtual Machine*
MB	*Megabyte(s)*
MIDP	*Mobile Information Device Profile*
OID	*Object Identifier*
OO	Orientação a Objeto / Orientado a Objeto
OS	*Operational System*
PBP	*Personal Basis Profile*
PC	*Personal Computer*
PDA	*Personal Digital Assistant*
PP	*Personal Profile*

RAM	*Random Access Memory*
RMS	*Record Management System*
SGBD	Sistema Gerenciador de Banco de Dados
SMS	*Short Message System*
SQL	*Structured Query Language*
UML	*Unified Modeling Language*
VM	*Virtual Machine*
XML	*Extensible Markup Language*

1 - Introdução

Este trabalho consiste na construção de um *framework* de persistência de objetos para aplicações voltadas a dispositivos móveis utilizando a plataforma J2ME[1] / MIDP[2].

No decorrer deste capítulo são abordados os seguintes tópicos: motivação, objetivos e organização deste trabalho.

1.1 Motivação

Seja para uso pessoal, seja para uso profissional, o uso de dispositivos móveis como celulares e PDAs vem crescendo rapidamente nos últimos anos.

Segundo a Nokia, um dos maiores fabricantes deste mercado, em 2004 o número de dispositivos móveis era de aproximadamente 643 milhões contra 490 milhões em 2003, o que aponta um crescimento global de 32%. A estimativa é que este número chegue a 2 bilhões no final de 2005 e 3 bilhões em 2010. Este grande crescimento deve-se principalmente a mercados de regiões emergentes como América Latina, Rússia, Índia e China [NOKIA].

Grande parte deste sucesso pode ser atribuído também aos avanços tecnológicos destes dispositivos alcançados nesta última década e à introdução de outras tecnologias, como câmeras digitais embutidas.

O mercado de dispositivos móveis está se expandindo para novas áreas além da comunicação de voz, como entretenimento multimídia (músicas, imagens, vídeos, rádio, etc), localização usando GPS e aplicações corporativas.

Com o avanço tecnológico, as aplicações para este tipo de dispositivos estão se tornando cada vez maiores e mais complexas, exigindo

[1] J2ME (Java 2 Micro Edition) é uma.
[2] MIDP (Mobile Information Device Profile) é um perfil de J2ME que define um ambiente de execução para a maioria dos dispositivos móveis.

do dispositivo maior poder de processamento e espaço para armazenamento dos dados.

Um dos pontos críticos de uma aplicação para dispositivos móveis de pequeno porte – celulares e PDAs – é o armazenamento e a manipulação dos dados. As principais limitações vão desde as APIs disponibilizadas pelos sistemas operacionais aos meios de armazenamento destes dispositivos, que apresentam capacidade de armazenamento e velocidade de acesso aos dados restritos.

Além das limitações supracitadas, a grande maioria das plataformas de desenvolvimento para dispositivos móveis não fornece suporte a conceitos como tabelas, campos e índices. Há poucas opções de bancos de dados e, muitas vezes, as soluções disponíveis são dependentes de determinado dispositivo e/ou sistema operacional.

Assim, para armazenar e recuperar os dados de uma aplicação, o engenheiro de software / desenvolvedor passa por um processo oneroso até chegar ao resultado final.

Uma solução para o problema apresentado é a introdução de uma camada de persistência, responsável por encapsular a lógica necessária para que um objeto ou uma coleção de objetos sejam salvos ou recuperados de um meio de armazenamento. Para tanto, é necessário o desenvolvimento de um *framework* de persistência.

Existem vários *frameworks* de persistência para aplicações cliente / servidor ou aplicações corporativas como Hibernate, JDO, OJB, Castor entre outros. Eles permitem ao engenheiro de software / desenvolvedor concentrar-se em detalhes específicos da aplicação e abstrair todo o processo de persistência.

No entanto, as camadas de persistência para estas plataformas são complexas para o ambiente de execução de um dispositivo móvel. O único *framework* de persistência de objetos para dispositivos móveis encontrado é o FramePersist, um trabalho acadêmico, implementado para a plataforma J2ME / PersonalJava[3].

[3] Produto já descontinuado pelo fabricante: Sun Microsystems.

Dentre as plataformas de desenvolvimento para dispositivos móveis, destaca-se a tecnologia J2ME / MIDP, uma das mais utilizadas atualmente [GUPTA]. Esta tecnologia é gratuita e traz consigo a portabilidade da linguagem Java.

Pelos motivos citados, será desenvolvido um *framework* de persistência para a plataforma J2ME / MIDP. Este *framework* deve ser fácil de usar e aprender, apresentar bom desempenho e atender aos requisitos de camadas de persistência, detalhados na seção 3.4 deste trabalho.

1.2 Objetivos

O objetivo principal deste trabalho é o desenvolvimento de um *framework* de persistência de objetos, denominado *Floggy*, que visa facilitar o desenvolvimento de aplicações para dispositivos móveis baseadas na plataforma J2ME / MIDP.

O perfil MIDP fornece um mecanismo de persistência chamado RMS que fornece operações básicas de inclusão, exclusão, alteração, busca e ordenação de registros, compostos por uma seqüência de bytes. Cabe ao desenvolvedor codificar todas as funções necessárias para armazenamento e leitura dos dados, transformando um objeto numa seqüência de bytes, conceito este chamado de serialização. A finalidade do *framework* é reduzir a complexidade e o trabalho de geração do código de persistência, busca, filtros e ordenação.

Os objetivos específicos são:

- Entender a tecnologia J2ME/MIDP;
- Compreender e aplicar o conceito de camadas de persistência;
- Projetar e implementar o *framework* de persistência de objetos;
- Validar os requisitos e o framework através de um estudo de caso.

1.3 Organização do Trabalho

Além do capítulo introdutório, este trabalho é composto de mais seis capítulos que estão estruturados da seguinte forma:

- O capítulo 2 apresenta a Plataforma Java 2 Micro Edition.
- O capítulo 3 descreve a Persistência de Objetos, mostrando suas vantagens, estratégias e alguns exemplos de *frameworks* bem sucedidos.
- O capítulo 4 apresenta a persistência dos dados em J2ME/MIDP usando o RMS, mostrando sua estrutura e suas limitações.
- O capítulo 5 apresenta o Floggy, um *framework* de persistência para J2ME/MIDP;
- O capítulo 6 apresenta uma aplicação-exemplo desenvolvida em J2ME/MIDP utilizando o *Floggy*.
- O capítulo 7 apresenta as conclusões finais e sugestões para trabalhos futuros.

2 - Plataforma Java 2 Micro Edition

Java 2 Micro Edition (J2ME) é uma versão compacta da linguagem Java padronizada pela Sun, direcionada ao mercado de dispositivos móveis. É um conjunto de especificações que disponibiliza uma máquina virtual Java (JVM), API e ferramentas para equipamentos móveis e qualquer outro dispositivo com processamento menor do que os computadores desktop.

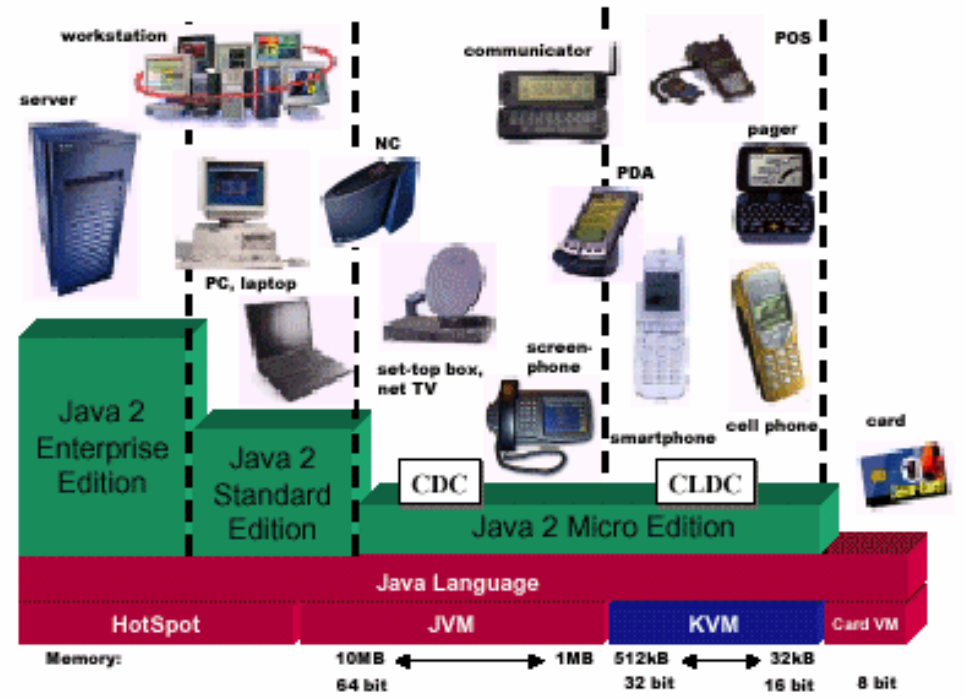

Figura 2.1 - Plataforma Java [Sun (1)]

A Figura 2.1 mostra a relação entre máquinas virtuais, as plataformas Java e as especificações do J2ME. Existe uma máquina virtual (JVM – Java Virtual Machine) para J2EE e J2SE, plataformas destinadas a servidores e estações corporativas e PCs; uma JVM para a especificação CDC

(*Connected Device Configuration*), que é destinada para TVs, PDAs e outros dispositivos embarcados; uma KVM (*Kilobyte Virtual Machine*) para a especificação CLDC (*Connected, Limited Device Configuration*), onde o perfil é voltado para PDAs e celulares; e, enfim, uma Card VM (*Virtual Machine*) para o desenvolvimento de sistemas para *Smart Cards*. Nas próximas seções serão apresentadas de forma mais detalhada as configurações, perfis e máquinas virtuais para J2ME.

A plataforma J2ME foi criada devido à demanda gerada pelo aumento expressivo de equipamentos eletrônicos móveis, como celulares e PDAs. Esse mercado precisava de uma tecnologia compatível, que abrangesse as características dos dispositivos como mobilidade, limitado poder de processamento e pouca memória disponível, alimentação elétrica por baterias e pequenas áreas de display.

As principais vantagens dos ambientes de execução J2ME são o uso eficiente do processador, utilização otimizada dos recursos de memória e baixo consumo de energia. Há ainda os benefícios herdados da tecnologia Java, que oferece um modelo robusto de segurança, suporte a aplicações em rede ou *offline* e portabilidade das aplicações.

2.1 Arquitetura

Há uma diversidade de dispositivos móveis, sendo que estes apresentam muitas características em comum. Em virtude disso, a arquitetura J2ME define uma estrutura multicamada, apresentada na Figura 2.2, composta de configurações, perfis e pacotes opcionais. Essas camadas são elementos fundamentais da plataforma J2ME e que, ao serem combinadas, transformam-se em ambientes de execução para a maioria dos dispositivos móveis.

Por exemplo, uma configuração como CLDC (*Connected Limited Device Configuration*) fornece serviços fundamentais para um grande conjunto de dispositivos, como dispositivos móveis com poder de processamento e largura de banda limitada. Já o perfil MIDP (*Mobile Information Device Profile*) fornece serviços de mais alto nível para um conjunto de dispositivos mais específicos como celulares e PDAs. Um pacote opcional adiciona serviços

especializados que são úteis para alguns dispositivos mas não necessários para todos, como serviços de localização (GPS), envio de mensagem SMS entre outros.

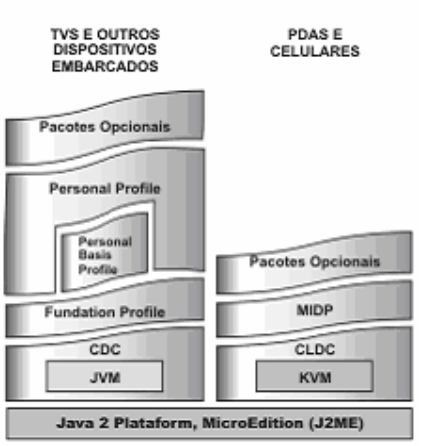

Figura 2.2 – Arquitetura J2ME [Sun (2)]

2.2 Configurações

Uma configuração define as funcionalidades mínimas para dispositivos com características semelhantes. Sendo assim, mais especificamente define as características quanto à máquina virtual e o conjunto de classes derivadas da plataforma J2SE. Atualmente há duas configurações definidas pela Sun, conforme ilustrado na Figura 2.3:

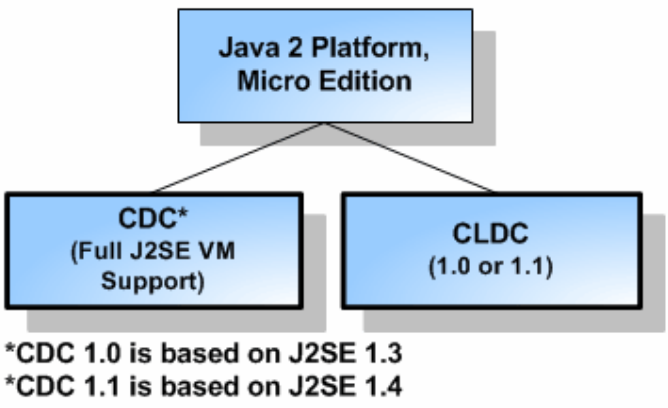

Figura 2.3 - Configurações J2ME [Sun (3)]

- ***Connected Limited Device Configuration*** **(CLDC)**: é a configuração mínima da plataforma J2ME, atendendo principalmente a dispositivos com conexão intermitente, largura de banda limitada, processador de baixa capacidade e pouca memória, como celulares, pagers e PDAs. Esta configuração é formada por um subconjunto de pacotes da plataforma J2SE e suporta dispositivos com processadores de 16 ou 32 bits com no mínimo 160 KB de memória não volátil e 32 KB de memória volátil. Usa uma máquina virtual reduzida em relação à máquina virtual clássica (JVM), que é a *Kilobyte Virtual Machine* (KVM), abordada na seção 2.4.

- ***Connected Device Configuration*** **(CDC)**: é um superconjunto da configuração CLDC, voltada para dispositivos com maior poder de processamento, maior capacidade de memória e conexões rápidas, como PDAs topo de linha, *gateways* residenciais, entre outros. Esta configuração suporta dispositivos com processadores de 32 bits, tipicamente baseados na arquitetura ARM[4], com no mínimo 2 MB de memória principal e 2,5 MB de memória somente leitura, e alguma forma de conectividade.

Por ser um superconjunto da configuração CLDC, inclui um maior número de classes e pacotes de J2SE. Usa a máquina virtual CVM que é uma máquina virtual completa e projetada para os dispositivos que necessitam de toda a funcionalidade presente no J2SE (JVM), entretanto oferecem menos requisitos de memória.

2.3 Perfil

Uma configuração não fornece classes para gerenciamento do ciclo de vida da aplicação, apresentação (interface com o usuário), persistência local os dados ou acesso seguro de informação armazenada em um servidor

[4] Tipo de design da arquitetura de processadores para dispositivos móveis.

remoto. Este tipo de funcionalidade é fornecido pelos perfis ou pacotes opcionais. Um perfil adiciona classes de domínio específico à lista de classes fornecidas pela configuração. Segundo [CORBERA, 2005] a diferença entre configuração e perfil é que a primeira descreve de forma geral uma família de dispositivos, enquanto a segunda é mais específica para um tipo particular de aparelho em uma dada família. O perfil somente acrescenta funcionalidades àquele aparelho.

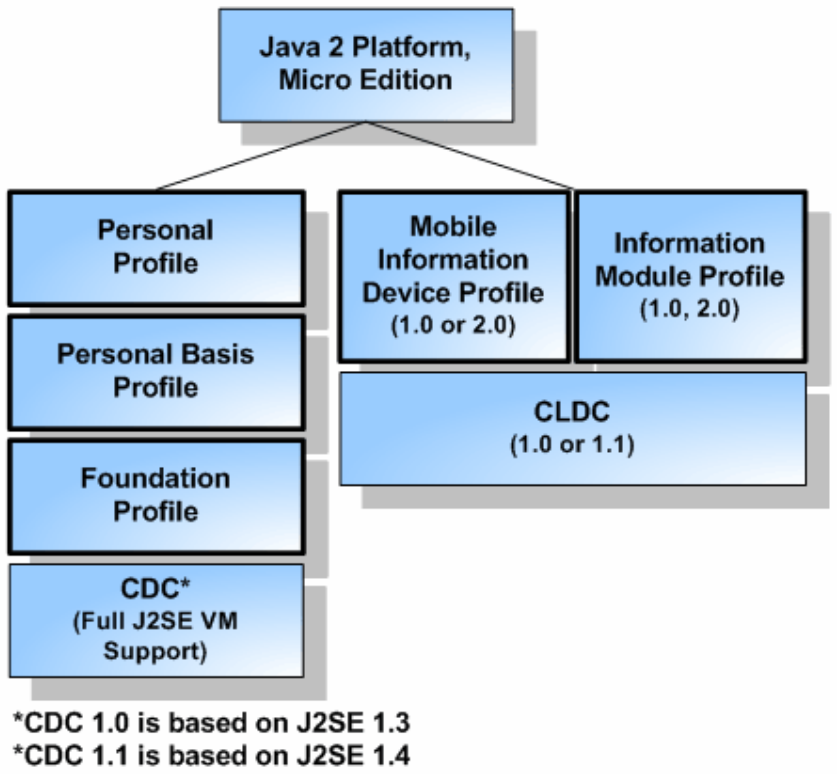

Figura 2.4 – Configurações e perfis da plataforma J2ME [Sun (3)]

Conforme a Figura 2.4, os perfis existentes são:

- ***Foundation Profile* (FP)**: Este é o perfil com menos funcionalidades da configuração CDC. Ele fornece conectividade à rede, mas não fornece classes de interface para construção de aplicações gráficas.

- ***Personal Basis Profile* (PBP)**: Este perfil CDC fornece um ambiente para dispositivos conectados que suportem um nível básico de interface gráfica ou necessitem o uso de *toolkits* específicos para aplicações.

- ***Personal Profile* (PP)**: Este perfil CDC é utilizado para dispositivos que possuem suporte completo de interface ou applet, como PDAs de alto padrão. Ele inclui a biblioteca AWT completa e é fiel ao ambiente web, executando facilmente applets feitos para ambientes desktop.

- ***Mobile Information Device Profile* (MIDP)**: O perfil MIDP é voltado especificamente para dispositivos portáteis, como celulares e PDAs de baixo desempenho. Este perfil proporciona funcionalidades como: interface com o usuário, conectividade à rede, persistência de dados, manipulação de eventos, mensagens e controle de aplicações. Junto com o CLDC, fornece um ambiente de execução Java completo que aumenta a capacidade dos dispositivos e minimiza o consumo de memória e energia, sendo suas aplicações chamadas de *midlets*.

- ***Information Device Profile* (IDP)**: O perfil IDP é direcionado para dispositivos com as características similares àquelas de dispositivos MIDP, mas com quase nenhuma potencialidade de interação com o usuário.

2.4 Maquina Virtual

A máquina virtual Java é a ponte entre a aplicação e a plataforma utilizada, convertendo o *bytecode* da aplicação em código de máquina adequado para o hardware e sistema operacional utilizados. Além disso, ela administra a memória do sistema, provendo segurança contra código malicioso e administrando as *threads* da aplicação. A máquina virtual Java usada na plataforma padrão (J2SE) e na plataforma corporativa (J2EE) foi desenvolvida para ser utilizada em sistemas *desktop* e servidores, respectivamente. Já na plataforma J2ME deve-se utilizar uma máquina virtual apropriada para os

dispositivos com baixo poder de processamento e memória reduzida – como celulares, pagers e PDAs – e outra para dispositivos com maior poder de processamento e disponibilidade de memória – como PDAs topo de linha, TV com internet, *handhelds* e outros.

Em virtude disso, duas máquinas virtuais foram projetadas:

- ***Kilobyte Virtual Machine* (KVM)**: É uma nova implementação da JVM, com suporte à configuração CLDC, para dispositivos portáteis com recursos limitados. A KVM foi desenhada para ser pequena, portátil, customizada e para uso de memória estática entre 40Kb a 80Kb. A quantidade mínima de memória requerida pela KVM é 128Kb. O processador pode ser de 16 ou 32 bits.

- ***Compact Virtual Machine* (CVM)**: Foi desenvolvida para adicionar suporte a mais funcionalidades do que a KVM. Com suporte à configuração CDC, esta máquina virtual engloba quase todas as características de uma máquina virtual Java convencional (JVM), só que de forma mais otimizada para dispositivos móveis.

2.5 Pacotes Opcionais

Pacotes opcionais são componentes muito importantes dentro da arquitetura J2ME. Considere-os como extensão dos perfis. Eles fornecem suporte a funcionalidades como: multimídia (imagens, vídeo e músicas), envio de mensagens (SMS), serviços de localização (GPS), comunicação via Bluetooth e integração com banco de dados.

2.6 Resumo

Este capítulo apresenta uma visão geral da plataforma J2ME e sua arquitetura, composta de configurações, perfis e pacotes opcionais. As configurações são as responsáveis por definir as funcionalidades mínimas (conjunto de classes e máquina virtual) para dispositivos com características

semelhantes. Os perfis, juntamente com os pacotes opcionais, complementam as configurações, definindo o ciclo de vida da aplicação, interface com usuário, persistência dos dados, entre outros. Juntos, configuração, perfil e pacotes opcionais, formam um ambiente de execução completo.

No capítulo 3 é abordado um estudo sobre persistência de objetos, função muito importante para armazenamento de dados em linguagens orientadas a objeto, caso da plataforma J2ME baseada na linguagem de programação Java.

3 - Persistência de objetos

O paradigma de orientação a objetos ocasionou uma mudança radical na estruturação e organização da informação. Entretanto, os bancos de dados mais utilizados continuaram sendo relacionais. Devido a isto, é comum a adaptação dos modelos de objetos na tentativa de adequá-los com o modelo relacional. Além disso, é notório o esforço aplicado no processo de persistência manual dos objetos no banco de dados – onde os desenvolvedores precisam dominar a linguagem SQL e utilizá-la para realizar acessos ao banco de dados. Como conseqüência ocorre uma redução considerável na qualidade do produto final, construção de uma modelagem "orientada a objetos" inconsistente e a um desperdício considerável de tempo na implementação manual da persistência.

Para permitir um processo de mapeamento entre sistemas baseados em objetos e bases de dados relacionais, foram propostas diversas idéias que conduziriam para o conceito de **Camada de Persistência**.

3.1 Estratégias de Persistência

Uma estratégia de persistência pode ser definida como a forma de implementação da persistência dos dados em uma aplicação. De acordo com [AMBLER (1)], existem três tipos de estratégias de persistência:

- Código SQL contido em classes de negócio, onde se inclui o código SQL para acesso ao SGBD em meio ao restante da lógica do sistema. Essa estratégia é muito usada graças à sua rapidez de implementação. Porém, ela é perigosa, pois implica, muitas vezes, no acoplamento do sistema ao SGBD utilizado, o que dificulta a manutenção do código.

Além disso, qualquer mudança na estrutura das tabelas existentes no Banco de Dados implica em mudanças no código da aplicação.

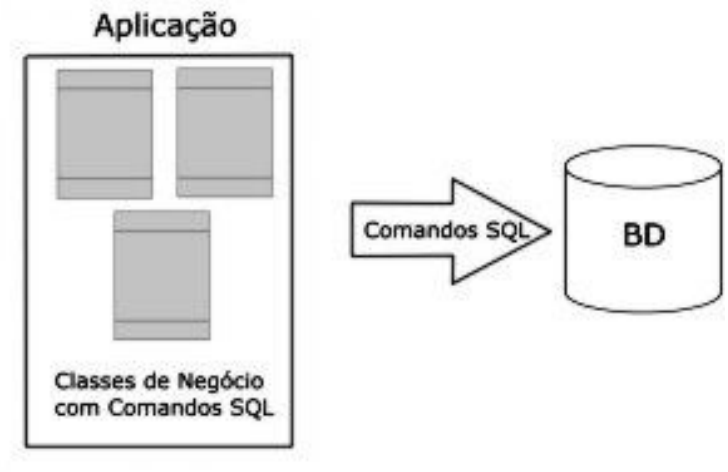

Figura 3.1 –SQL contido nas classes de negócio. Adaptada [AMBLER (1)]

- Código SQL contido em classes de acessos a dados, com intuito de diminuir o acoplamento da estratégia anterior. Consiste em separar o código SQL das classes da aplicação, de forma que as alterações no modelo de dados requeiram modificações apenas nas classes de acesso a dados (Data Access Classes), restringindo o impacto das mudanças no sistema como um todo. No entanto, apesar da clareza do código e melhor divisão de responsabilidades, a solução está longe de ser ideal, por ainda manter as duas formas de representação e armazenamento de dados (objetos e dados relacionais) intimamente ligadas.

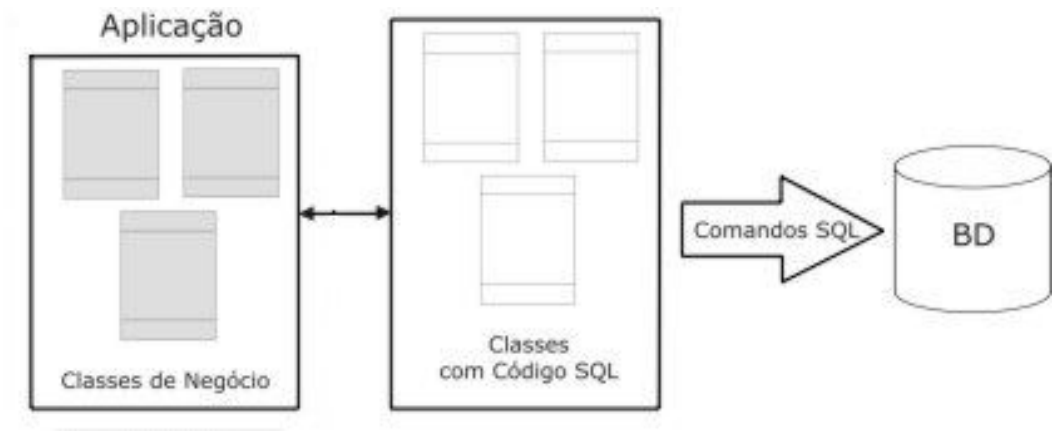

Figura 3.2 –SQL contidos nas classes de acesso a dados. Adaptada [AMBLER (1)]

- Camada de persistência, onde sua função é garantir aos desenvolvedores a total independência entre o modelo de objetos e o esquema de dados do banco, permitindo que a base ou detalhes do esquema de dados sejam substituídos sem impacto nenhum na aplicação. Além da manipulação dos dados ser realizada através de uma linguagem natural, na qual é composta de comandos de alto nível, como salvar e remover.

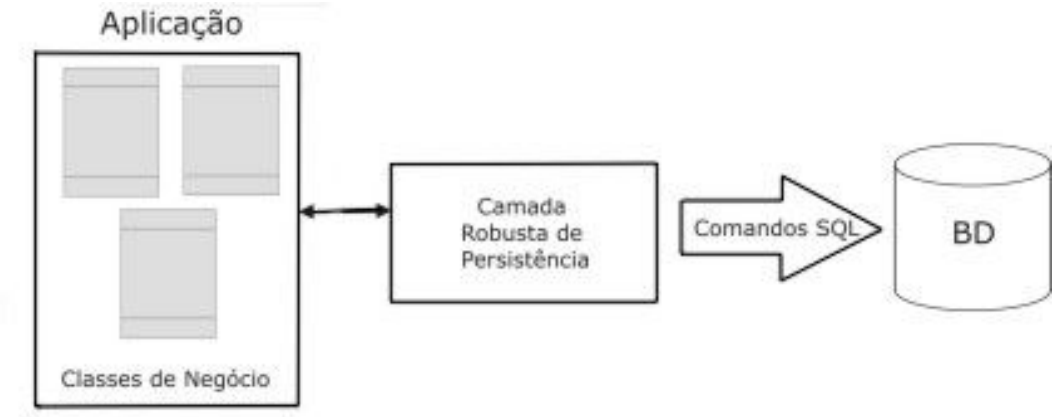

Figura 3.3 – Código de manipulação de dados contido na Camada de Persistência. Adaptada [AMBLER (1)]

3.2 Camada de Persistência

Conceitualmente, uma Camada de Persistência de Objetos é uma biblioteca que permite a realização do processo de persistência[5] de forma transparente [FREIRE]. Graças à independência entre a camada de persistência e o repositório utilizado, também é possível gerenciar a persistência de um modelo de objetos em diversos tipos de repositórios (ex. um banco de dados relacional, um banco de dados orientado a objeto ou até mesmo um arquivo XML). A utilização deste conceito permite ao desenvolvedor trabalhar como se estivesse em um sistema completamente orientado a objetos, utilizando métodos de alto nível para incluir, alterar e remover objetos e uma linguagem para realizar consultas que retornam coleções de objetos instanciados.

As camadas de persistência podem ser classificadas como *implícita* ou *explicita,* devido à forma com que efetuam a persistência dos objetos. Em uma persistência do tipo *implícita,* os objetos não precisam de uma requisição para serem salvos, alterados ou excluídos em um repositório de dados. Estas operações ocorrem simultaneamente com as operações realizadas sobre o objeto. Já no caso de uma persistência do tipo *explicita,* é necessário indicar que tais operações realizadas sobre o objeto devam ser persistidas. Esse último caso é o mais comum dentre as camadas de persistência atuais.

3.3 Vantagens e Desvantagens

As vantagens decorrentes do uso de uma Camada de Persistência no desenvolvimento de aplicações são evidentes: a sua utilização isola os acessos realizados diretamente ao banco de dados na aplicação, bem como centraliza os processos de construção de consultas (queries) e operações de manipulação de dados (insert, update e delete) em uma camada de objetos inacessível ao programador. Este encapsulamento de responsabilidades garante maior confiabilidade às aplicações e permite que, em alguns casos, o

[5] Processo de Persistência: Armazenamento e manutenção do estado de objetos em algum meio não-volátil, como um banco de dados.

próprio SGBD ou a estrutura de suas tabelas possam ser modificados, sem trazer impacto à aplicação nem forçar a revisão e recompilação de códigos.

Uma desvantagem desta estratégia é que ela pode ter impacto sobre o desempenho da aplicação. Este é um ponto crítico levando em consideração que, neste trabalho, os dispositivos-alvo apresentam limitações de capacidade de processamento. Entretanto, esta é a estratégia utilizada neste trabalho, tomando os cuidados necessários para que as operações, especialmente as consultas, sejam simples o suficiente para não tornar o código complexo a ponto de prejudicar o desempenho das aplicações.

3.4 Requisitos de uma Camada de Persistência

Segundo [AMBLER (1)], uma Camada de Persistência deve implementar as seguintes características:

- ***Suporte a diversos tipos de mecanismos de persistência*:** deve suportar a substituição de um mecanismo de persistência (ex. SGBD relacional ou OO, arquivo XML, outros) livremente e permitir a gravação de estado de objetos em qualquer um destes meios.

- ***Encapsulamento completo da camada de dados*:** a utilização apenas de comandos de alto nível para lidar com a persistência dos objetos, como salvar ou excluir, deixando o tratamento desses comandos para a camada de persistência em si.

- ***Ações com multi-objetos*:** Suportar listas de objetos sendo instanciadas e retornadas da base de dados deve ser um item comum para qualquer implementação, tendo em vista a freqüência desta situação.

- ***Transações*:** disponibilizar ao desenvolvedor o controle do fluxo de transação ou oferecer garantias sobre o mesmo, caso a própria Camada de Persistência preste este controle.

- ***Extensibilidade*:** deve permitir a adição de novas classes ao esquema e a modificação fácil do mecanismo de persistência.

- ***Identificadores de Objetos*:** A implementação de algoritmos de geração de chaves de identificação garante que a aplicação trabalhará com objetos com identidade única e sincronizada entre o banco de dados e a aplicação.

- ***Cursores e Proxies*:** Em muitos casos os objetos armazenados são muito grandes – e recuperá-los por completo a cada consulta não é uma boa idéia. Técnicas como o *lazy loading* (carregamento tardio) utilizam-se de *proxies* para garantir que atributos só serão carregados à medida que forem necessários ao usuário, e cursores para manter registro da posição dos objetos no banco de dados (e em suas tabelas específicas).

- ***Registros*:** Apesar da idéia de trabalhar-se apenas com objetos, as camadas de persistência devem dispor de um mecanismo de recuperação de registros. Isto permite a recuperação de atributos de diversos objetos relacionados com uma só consulta, além de também facilitar a integração com mecanismos de relatórios que não trabalham com objetos.

- ***Arquiteturas Múltiplas*:** suportar arquiteturas onde o banco de dados encontra-se em um servidor central ou arquiteturas mais complexas (em várias camadas).

- ***Diversas versões de banco de dados e fabricantes*:** deve reconhecer as diferenças de recursos, sintaxe e outras particularidades existentes no acesso aos bancos de dados suportados.

- ***Múltiplas conexões*:** deve gerenciar múltiplas conexões. Essa técnica, usualmente chamada de *pooling*, garante que vários usuários utilizarão o sistema simultaneamente sem quedas de desempenho.

- ***Queries SQL*:** apesar do poder trazido pela abstração em objetos, este mecanismo não é funcional em todos os casos. Para as exceções, a camada de persistência deve prover um mecanismo de consulta que permita o acesso direto aos dados ou então algum tipo de linguagem de consulta simular à SQL, de forma a permitir consultas com um grau de complexidade maior que o comum.

- ***Controle de Concorrência*:** acesso concorrente a dados pode ocasionar inconsistências. Para prever e evitar problemas decorrentes do acesso simultâneo, a camada de persistência deve prover algum tipo de mecanismo de controle de acesso. Este controle geralmente é feito utilizando-se dois níveis:
  - Pessimista *(pessimistic locking):* no qual as tuplas no banco de dados relativas ao objeto acessado por um usuário são travadas e tornam-se inacessíveis a outros usuários até o mesmo liberar o objeto.
  - Otimista *(optimistic locking)*: toda a edição é feita em memória, permitindo que outros usuários venham a modificar o objeto.

É importante ressaltar que os requisitos listados acima são voltados para o desenvolvimento de aplicações tradicionais que, geralmente, têm seus dados armazenados em arquivos locais, os quais são rápida e facilmente acessados de um disco rígido ou de uma rede local. No entanto, dispositivos móveis, tais como PDAs e celulares, não possuem discos locais, possuem baixo poder de processamento e memória e raramente estão conectados de forma contínua a uma base de dados em um servidor. Conseqüentemente, alguns dos requisitos deverão ser reavaliados, adaptados ou descartados para não tornar a complexidade de uma camada de persistência para aplicações

móveis tão grandes a ponto de prejudicar o desempenho das aplicações, e tornar a utilização da camada inviável.

3.5 Processo de Mapeamento

O processo de mapeamento é a definição de como os objetos são mapeados para o repositório de dados. Este processo, segundo [AMBLER (2)], pode ser dividido em três grandes etapas: mapeamento de atributos, mapeamento classes e hierarquias e mapeamento de relacionamentos.

3.5.1 Atributos

Os atributos das classes são mapeados como colunas de uma tabela do modelo relacional. Este processo de mapeamento deve levar em consideração fatores como a tipagem e o comprimento máximo dos campos. O mapeamento envolve atributos do tipo *simples* – como *strings,* caracteres, números inteiros e reais, datas, etc – e atributos *complexos* – como um objeto ou uma lista de objetos.

Também é importante ressaltar que, nem sempre atributos de um objeto são obrigatoriamente uma coluna em uma tabela. Além disso, existem casos onde um atributo pode ser mapeado para diversas colunas ou vários atributos podem ser mapeados para uma mesma coluna.

No mapeamento de objetos, costuma-se adotar um identificador único, ou *Objects Ids* (OIDs), que são gerados internamente para cada objeto, a fim de simplificar o mapeamento e posterior persistência dos dados. Os OIDs funcionam como um tipo de chave primária nos sistemas Orientados a Objetos, mantendo relacionamentos entre objetos e garantindo a unicidade dos objetos em todo o esquema.

3.5.2 Mapeamento de Classes e Hierarquias

O mapeamento das classes e suas hierarquias para tabelas é o aspecto mais complexo do mapeamento, dado que o modelo relacional não

oferece suporte ao conceito de herança. Existem diversas técnicas que permitem o mapeamento de conjuntos de objetos, cada qual com suas vantagens e desvantagens sobre as demais. Em geral, uma camada de persistência implementa uma destas técnicas e o desenvolvedor que for utilizá-la deve organizar as tabelas em seu banco de dados de acordo com o esquema de objetos.

As três técnicas de mapeamento de objetos mais comumente implementadas são:

- ***Mapeamento de uma tabela por hierarquia*:** Toda a hierarquia de classes deve ser representada por uma mesma tabela: uma coluna que identifique o tipo do objeto (Object Type) serve para identificar a classe do objeto representado por cada linha na tabela, quando nenhum outro modo de identificação é viável. As desvantagens são: ausência de normalização e uma proliferação de campos com valores nulos para hierarquias de classes com muitas especializações.

- ***Mapeamento de uma tabela por classe concreta*:** Nesta estratégia, tem-se uma tabela para cada classe concreta presente no sistema. A tabela identifica a classe de todos os elementos contidos na mesma, tornando desnecessário o mecanismo de Object Type. Essa estratégia leva à redundância de dados: quaisquer atributos definidos em uma classe abstrata na hierarquia devem ser criados em todas as tabelas que representam classes-filhas da mesma. Além disso, mudar o tipo (especializar ou generalizar) de um objeto torna-se um problema, já que é necessário transferir todos os seus dados de uma tabela para outra no ato da atualização.

- ***Mapeamento de uma tabela por classe*:** Cria-se uma tabela para cada classe da hierarquia, relacionadas através do mecanismo de especialização padrão do banco de dados (utilização de chaves estrangeiras). Segundo esta estratégia, tenta-se ao máximo manter a

normalização de dados, de forma que a estrutura final das tabelas fica bastante parecida com a hierarquia das classes representada pela UML. A colocação de um identificador de tipo (Object Type) na classe-pai da hierarquia permite identificar o tipo de um objeto armazenado nas tabelas do sistema sem forçar junções entre as tabelas, garantindo melhorias no desempenho e, é uma estratégia comumente utilizada. A quantidade de junções (*joins*) entre tabelas para obter todos os dados de um objeto é o seu principal ponto negativo.

A Tabela 3.1 faz um comparativo destas três técnicas de mapeamento de classes.

	Uma tabela por hierarquia de classes	**Uma tabela por classe concreta**	**Uma tabela por classe**
Facilidade a consulta interativa dos dados	Simples	Médio	Médio/Difícil
Facilidade de implementação	Simples	Médio	Difícil
Facilidade de acesso	Simples	Simples	Médio/Simples
Acoplamento	Muito alto	Alto	Baixo
Velocidade de acesso	Rápido	Rápido	Médio/Rápido
Suporte a polimorfismos	Médio	Baixo	Alto

Tabela 3.1 – Comparativos das técnicas de mapeamentos de classes

3.5.3 Mapeamento de Relacionamentos

O mapeamento de relacionamentos não apresenta muitas maneiras como no caso do mapeamento das classes e sua hierarquia. Cada um dos tipos de relacionamento (*um para um, um para muitos, muitos para muitos*) possui uma ou no máximo duas formas de serem implementados; a única diferença é a visibilidade do relacionamento: unidirecional ou bidirecional.

- *Relacionamento **um para um***: um atributo na tabela funciona como uma chave estrangeira para a outra tabela, estabelecendo o relacionamento entre as duas. Caso deseje-se um relacionamento bidirecional, ambas as tabelas possuirão colunas funcionando como chaves estrangeiras uma para a outra.

- *Relacionamento* ***um para muitos***: o fato de optar-se por um relacionamento unidirecional ou bidirecional muda a forma como ele é representado. No caso do relacionamento unidirecional, define-se uma coluna na tabela representante do lado não-unitário do relacionamento que aponte para a linha da tabela do lado unitário. Caso se deseje uma visibilidade bidirecional, é necessário criar uma tabela que contenha as chaves primárias de cada uma das tabelas integrantes do relacionamento, onde cada linha representa um relacionamento entre as classes.

- *Relacionamento* ***muitos para muitos***: cria-se uma tabela auxiliar que conterá como colunas as chaves primárias das outras tabelas, onde cada linha representa um relacionamento entre as tabelas.

3.6 Implementações de Camadas de Persistência

Existem atualmente diversas implementações (bibliotecas) de camadas de persistência. As implementações atendem os requisitos de formas variadas, cada uma com suas vantagens e desvantagens.

Algumas implementações fornecem suporte a funcionalidades específicas e não obrigatórias às camadas de persistência, como por exemplo geração dos esquemas de dados automaticamente e criação de classes e hierarquias a partir do esquema de tabelas do banco utilizando engenharia reversa.

Destaque para o Hibernate e a especificação JDO (*Java Data Objects*).

3.6.1 Hibernate

O Hibernate é uma ferramenta gratuita que permite a persistência de objetos de aplicações Java em bases de dados relacionais de forma

transparente. Permite também o desenvolvimento de classes seguindo as características de orientação a objetos da linguagem Java – incluindo associações, herança, polimorfismo e composição [HIBERNATE].

A forma como o Hibernate deve carregar ou salvar um objeto é descrita em arquivos de mapeamento, no formato XML. Neles estão contidas informações que relacionam as propriedades de um objeto aos campos das tabelas. Além destas informações, os arquivos de mapeamento definem os relacionamentos entre classes e a propriedade do objeto que se relaciona com a chave primária. Esses arquivos de mapeamento são compilados em tempo de inicialização da aplicação e provêem ao Hibernate as informações necessárias sobre as classes.

O Hibernate utiliza o mecanismo de reflexão da linguagem Java para instanciar objetos dinamicamente e acessar suas propriedades. Por isto, define como requisito que uma classe persistente deve possuir um construtor sem parâmetros e que, preferencialmente, estas classes devem ser construídas seguindo o padrão *JavaBean*, com métodos *get* e *set* para acesso às propriedades.

Para seleção dos objetos o Hibernate fornece três mecanismos: *HQL, API Criteria e Native SQL*. *HQL* (*Hibernate Query Language*) se parece muito com o SQL, porém é orientado a objetos, compreendendo notações de hierarquias, polimorfismos e associações. *API Criteria* permite a criação de consultas através da utilização de classes e métodos, ou seja, totalmente orientada a objeto e sem a utilização de uma linguagem de texto. Já *Native SQL* permite o uso da linguagem nativa do banco de dados em uso, sendo útil se houver necessidade de utilizar funcionalidades específicas.

A comunicação entre o Hibernate e um banco de dados relacional é estabelecida através de conexões JDBC (*Java Database Conectivity*). As configurações necessárias para que esta comunicação seja estabelecida podem ser definidas em um arquivo de simples propriedades (*hibernate.properties*), um arquivo mais sofisticado (*hibernate.config.xml*) ou ainda via programação.

Em resumo, o Hibernate é uma ferramenta que permite a persistência de objetos em banco de dados relacionais sem a necessidade do uso de instruções SQL. Atualmente, é uma das ferramentas de persistência mais utilizadas. Contudo, não possui suporte a bancos de dados orientados a objetos e outras formas de armazenamento, como por exemplo arquivos XML.

3.6.2 JDO

Java Data Objetcs (JDO) é uma especificação para gerenciamento de dados de aplicações Java, desenvolvida como *Java Specification Request 12 (*JSR-12) pelo *Java Community Process* (JCP). Nesta especificação é descrita a semântica de como uma aplicação Java deve persistir, recuperar e consultar objetos em um repositório de dados.

As interfaces definidas na especificação JDO não são suficientes para realizar a persistência de objetos. É necessário fazer uso de uma implementação JDO, comercializada por empresas ou distribuídas gratuitamente como TJDO[6], OJB[7], JPOX[8], Castor[9] e outros.

As implementações JDO diferem quanto às funcionalidades oferecidas, como por exemplo, a otimização para um banco de dados específico. As variadas implementações trabalham com banco de dados relacionais, bancos de dados orientados a objetos, sistemas de arquivos ou XML.

As implementações devem estar de acordo com a especificação JDO, o que na maioria das vezes não ocorre. Certas implemetações contemplam a especificação parcialmente e adicionam alguns recursos extras.

O uso de JDO, assim como o uso de outras camadas de persistência, permite que o engenheiro de software possa focar seus esforços no modelo de objetos e delegar os detalhes de persistência para a implementação JDO.

---

[6] http://tjdo.sourceforge.net/
[7] http://db.apache.org/ojb/
[8] http://www.jpox.org/
[9] http://www.castor.org/

Para que um objeto possa ser persistido pelo JDO ele deve implementar a interface *PersistenceCapable*. Essa interface define os métodos que permitem o gerenciamento das instâncias pelo JDO e os que permitem conhecer os estados de uma instância. Para saber se uma instância está associada a uma transação, se é transiente ou persistente, utiliza-se esta interface [JDO].

Os métodos da interface *PersistableCapable* são implementados nas classes candidatas a persistência de forma automática pelo enxertador (*enhancer*), através da alteração do *bytecode* das classes. As alterações são baseadas em um arquivo descritor do objeto.

Este mecanismo de alteração de *bytecode* não é utilizado por todas as implementações de JDO. Algumas delas alteram o próprio código-fonte das classes.

Enfim, o JDO é um padrão para persistência transparente de objetos de aplicações Java, que apresenta diversas implementações, cada uma com suas vantagens e desvantagens.

O uso do JDO permite uma modelagem de persistência dentro do paradigma de orientação a objetos, porém não restringe a persistência a nenhum paradigma de armazenagem.

3.7 Resumo

Neste capítulo são abordadas três estratégias de persistência, através de uma comparação entre as vantagens e desvantagens de cada estratégia. A estratégia de camada de persistência foi enfatizada por ser um dos focos deste trabalho.

A vantagem no uso da camada de persistência é o encapsulamento do código de manipulação de dados, proporcionando total abstração ao engenheiro de *software*. Em contrapartida, há a possibilidade de haver impacto no desempenho da aplicação. Este problema pode se tornar crítico considerando as limitações dos dispositivos móveis.

Posteriormente, foram listados os processos de mapeamento de objetos considerando as etapas de mapeamento de atributos, classes,

hierarquias e relacionamentos. Estes processos serão utilizados no decorrer do desenvolvimento do framework proposto.

Por fim, foram analisadas duas implementações de camadas de persistência: Hibernate e JDO. Constatou-se que as camadas de persistência existentes são soluções robustas e voltadas para plataformas tradicionais (J2SE e J2EE). Essas soluções são complexas demais para serem utilizadas em ambientes de execução para dispositivos móveis (J2ME) e confirmam a necessidade de uma camada de persistência destinada especializada, sendo este um dos principais motivadores deste trabalho.

No próximo capitulo será apresentada a forma de armazenamento dos dados em dispositivos móveis que utilizam a tecnologia J2ME/MIDP.

4 - Persistência de dados em J2ME/MIDP

Tipicamente, uma aplicação tradicional armazena seus dados em arquivos locais ou em banco de dados relacionais, os quais são rapidamente e facilmente acessados através de um disco rígido ou de uma rede local. Entretanto, os dispositivos móveis geralmente não têm discos rígidos locais e raramente estão conectados de forma contínua a uma base de dados em um servidor. O meio mais comum de armazenamento nestes dispositivos é a memória flash[10].

As vantagens da memória *flash* são pequeno tamanho físico, peso reduzido, resistência a choques e alto desempenho para leitura de dados. Em contrapartida, as operações de escrita são lentas, assim como a capacidade de armazenamento é reduzida quando comparada aos discos rígidos dos computadores pessoais ou corporativos.

A especificação MIDP exige que todas as implementações forneçam persistência para que as informações sejam preservadas durante a execução de uma MIDlet[11] ou quando o dispositivo for desligado. O mecanismo de armazenamento pode variar de acordo com os dispositivos, porém a interface de programação (API) não, o que torna os MIDlets que utilizam a facilidade de persistência mais portáveis.

O MIDP fornece um mecanismo de armazenamento de dados chamado RMS (*Record Management System*; em português, Sistema de Gerenciamento de Registros). Além deste mecanismo, há também um pacote opcional, definido pela JSR-75, que implementa um sistema de armazenamento através de arquivos. Porém, existem poucos dispositivos que possuem suporte a este pacote opcional.

---

[10] Memória de alta densidade e alto desempenho. Utiliza a tecnologia EEPROM (Electrically Erasable Programmable Read-Only Memory).

[11] Nome dado a uma aplicação J2ME/MIDP.

A API de RMS encontra-se no pacote javax.microedition.rms, e possui diversas classes e interfaces, que são detalhadas a seguir.

4.1 Record Store

O mecanismo RMS é composto por *record stores*[12]. Um *record store* é formado por um conjunto de registros (ver Figura 4.1 – Estrutura de um *record store*). Cada registro consiste em um identificador único – valor inteiro e incremental que desempenha papel de chave principal – e um conjunto de bytes. A responsabilidade de interpretar o conteúdo de um registro – ou seja, sua seqüência de bytes - fica a cargo da aplicação.

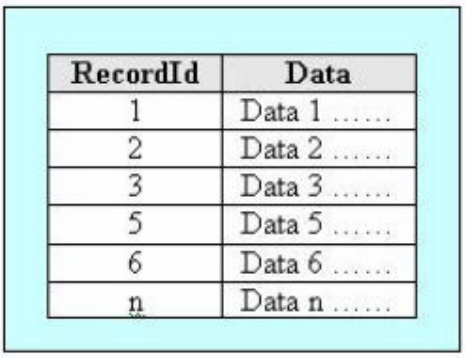

Figura 4.1 – Estrutura de um *record store* [GUPTA]

Um *record store*, além dos registros em si, armazena alguns dados como última modificação e versão. Isto permite que uma aplicação descubra quanto um *record store* foi modificado (inclusão de novos registros e alteração e exclusão de registros). Se necessário, há ainda a possibilidade de se adicionar um *record listener*[13] que será notificado quando tais modificações ocorrerem.

[12] Em português, armazéns de registro.
[13] Objeto notificado em caso de mudanças no record store.

4.2 Forma de armazenamento

O RMS é o responsável por alocar o *record store* no dispositivo. Portanto, onde o *record store* está alocado fisicamente no dispositivo não é importante porque a coleção de registros não é acessada diretamente pela aplicação. O acesso aos registros é realizado somente através da API RMS, conforme ilustrado na Figura 4.2.

Figura 4.2 – Estrutura da API RMS [MAGALHÃES 2005]

Uma MIDlet pode criar um número ilimitado de *record stores*. Cada *record store* é identificado por um nome. O nome de um *record store* deve conter até 32 caracteres UNICODE[14], considerar a diferença entre o uso de letras maiúsculas e minúsculas e ser exclusivo dentro de um *MIDlet suíte*[15].

No MIDP 1.0 só é possível compartilhar um *record store* com *MIDlets* de um mesmo *MIDlet suite* (ver Figura 4.3). Tal limitação acaba na versão 2.0 do MIDP, onde um r*ecord store* pode ser compartilhado com outras *MIDlet suites*. Neste caso, o nome do *record store*, se acessado por outra *MIDlet*, será formado pela concatenação do nome da *MIDlet suite* que o criou, do nome do fornecedor[16] e do nome do *record store*.

[14] Sistema de codificação de caracteres onde cada caractere recebe um número independente de plataforma, programa e língua.
[15] MIDlet Suite: Conjunto de uma ou mais MIDlets, distribuídos em um único arquivo, normalmente um .jar.
[16] Responsável pelo desenvolvimento da aplicação.

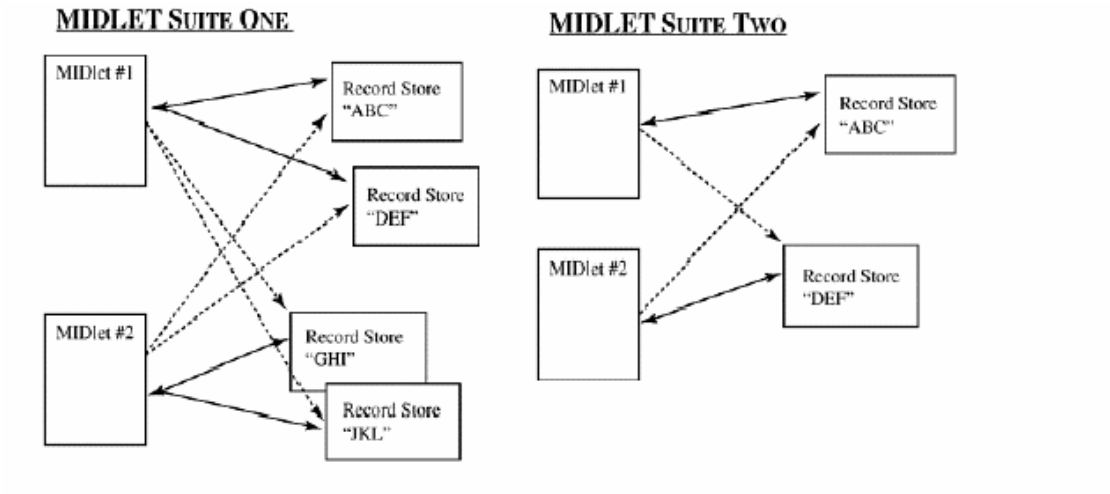

Figura 4.3 – Acesso de MIDlets a record stores no MIDP 1.0. Adaptada [MUCHOW 2001].

Na Figura 4.3 as linhas cheias indicam o acesso aos record stores que o MIDlet criou. As linhas tracejadas indicam o acesso aos record stores criados por outra MIDlet do mesmo conjunto.

Quando um conjunto de MIDlets é removido, os *record stores* criados pelo conjunto são apagados. Quando a versão de um conjunto de MIDlets é atualizada, os *record stores* criados pelo conjunto são mantidos.

4.3 Limites de armazenamento

A especificação de MIDP requer que os dispositivos reservem pelo menos 8k de memória não-volátil para persistência de dados. Como não há nada definido em relação ao tamanho do registro, não há garantia de que todos os dispositivos se comportem da mesma maneira.

No momento da instalação do aplicativo, é possível indicar ao dispositivo a quantidade mínima de bytes requeridos pela aplicação através da propriedade MIDlet-Data-Size, descrita no arquivo de manifesto do arquivo JAR e no arquivo JAD[17].

[17] Descritor da aplicação

4.4 Velocidade de acesso

Operações em memória persistente normalmente são mais lentas do que as operações em memória volátil (não-persistente). Logo, operações de escrita podem ser muito demoradas em algumas plataformas devido ao sistema de armazenamento de cada dispositivo. Uma boa prática seria manter as informações muito acessadas na memória volátil e evitar realizar operações de persistência constantemente durante a execução da aplicação.

4.5 API RMS

O RMS apresenta quatro interfaces, uma classe e cinco exceções. Além das funcionalidades básicas – criar e apagar um *record store* e adicionar, remover e alterar um registro – ele fornece um mecanismo de busca e ordenação de registros. As funcionalidades dessa classe e das interfaces, assim como as exceções possíveis no uso do RMS, são mostradas na Tabela 4.1.

Elemento	**Funcionalidade**
Interfaces	
RecordComparator	Compara dois registros (ordena, classifica)
RecordEnumeration	Enumera um registro (ID)
RecordFilter	Filtra registros, baseado em um critério
RecordListener	Recebe notificação dos eventos relacionados aos registros (excluídos, adicionados, alterados)
Classe	
RecordStore	Responsável pelo armazenamento de registros: inclusão, exclusão, alteração.
Exceções	
InvalidRecordIDException	Indica que uma operação não pode ser completada porque o ID do registro não é válido
RecordStoreException	Indica que ocorreu um erro em uma operação de arquivo
RecordStoreFullException	Indica que a operação não pode ser completada porque o armazenamento do sistema do registro está cheio
RecordStoreNotFoundException	Indica que a operação não pode ser completada porque o registro procurado não foi encontrado
RecordStoreNotOpenException	Indica que a operação foi feita sobre um RecordStore que não está aberto

Tabela 4.1 – Elementos do RMS [adaptação de MAGALHÃES 2005]

4.6 Resumo

O RMS é simples, flexível e pequeno, características importantes para um ambiente MIDP. Entretanto, cabe à aplicação gerenciar o código de persistência dos dados, transformando um objeto em uma seqüência de bytes e vice-versa.

Em aplicações pequenas, com poucos dados a serem persistidos, como jogos, lista de tarefas e agenda de contatos, o RMS se mostra como uma alternativa ideal. Já em aplicações de maior porte, como automação de força de venda, que apresentam uma grande quantidade de dados a serem persistidos, o RMS adiciona a necessidade e complexidade de gerenciamento de todo o código de persistência.

No próximo capitulo é abordado o *framework* Floggy para persistência de objetos em J2ME/MIDP. O *framework* visa abstrair o mapeamento necessário para transformação de objetos em seqüências de bytes (e vice-versa) através de comandos de alto nível (incluir, alterar ou excluir um objeto) que devem ser utilizados para manipular um objeto para que este seja armazenado e recuperado a partir de um meio persistente.

5 - Floggy

Conforme discutido nos capítulos anteriores, o perfil MIDP apresenta grande carência de uma camada de persistência que, através apenas de comandos de alto nível, seja responsável por encapsular toda a lógica necessária para que um objeto possa ser armazenado e recuperado de um meio de armazenamento. Esta camada facilitaria o desenvolvimento de aplicações para dispositivos móveis que se utilizam dos recursos de persistência de dados.

O *framework* de persistência de objetos, denominado *Floggy*, proposto neste trabalho é composto por dois módulos: *Framework* e *Compiler*. O módulo *Framework* é responsável por fornecer às aplicações o suporte a persistência, recuperação e busca dos dados, enquanto o módulo *Compiler* é responsável por gerar e enxertar o código de persistência aos objetos.

Na seção 5.1, é abordada uma visão geral do uso do *Floggy* em uma aplicação. Na seção 5.2, são descritos os requisitos de uma camada de persistência implementados pelo *Floggy*. Na seção 5.3, é descrita a arquitetura geral, incluindo tópicos sobre suas classes, processos de mapeamento e identificação e seleção de objetos. Na seção 5.4, é descrita a forma de funcionamento do enxertador de código de persistência. E por fim, na seção **Erro! Fonte de referência não encontrada.**, através das conclusões, é descrita a validação dos requisitos.

5.1 Visão Geral

O *Floggy* foi desenvolvido para agilizar o desenvolvimento de aplicações MIDP, pois estas aplicações podem fazer uso das facilidades oferecidas por este framework de persistência. Desta forma o engenheiro de software pode focar seus esforços na solução do problema principal a ser tratado pela aplicação, delegando toda a persistência dos dados ao framework.

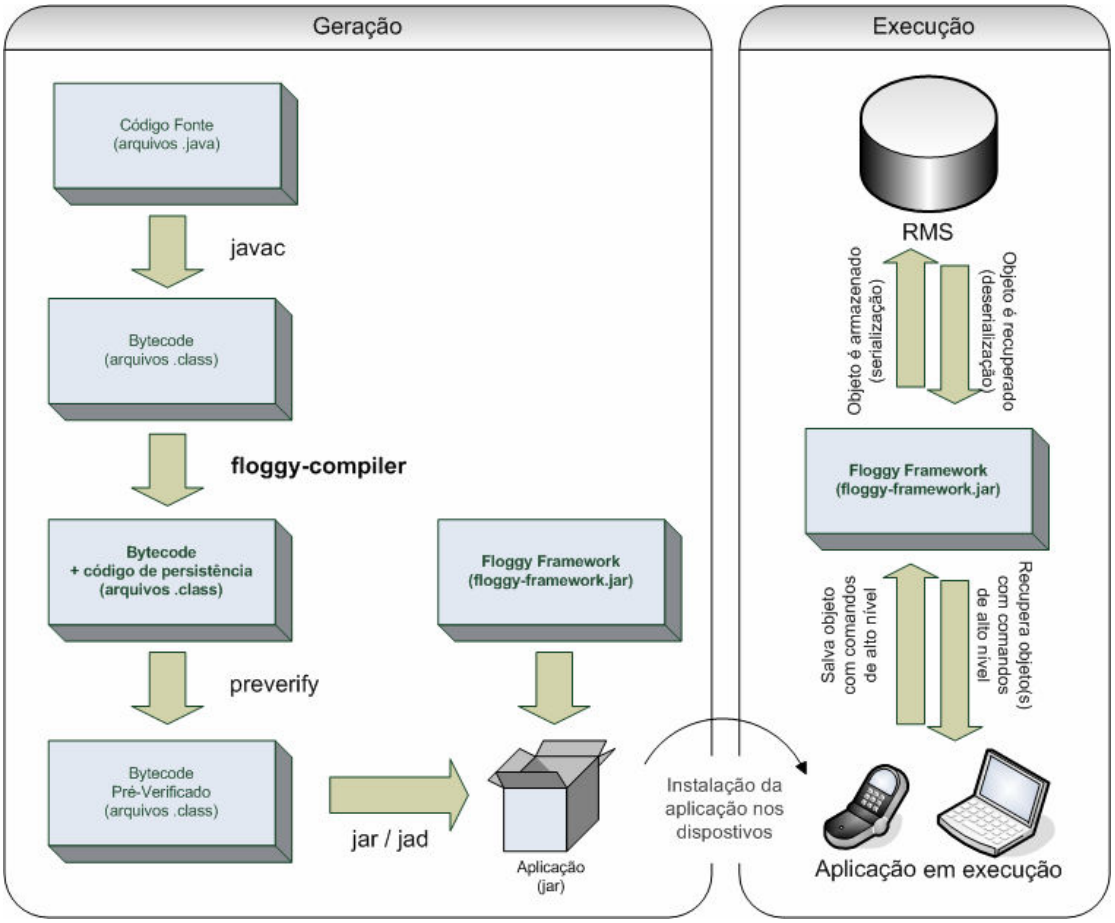

Figura 5.1 - Visão geral da arquitetura do Floggy

A Figura 5.1 apresenta, de forma resumida, a utilização do *Floggy* na geração e execução de uma aplicação. No início do processo os arquivos fontes são compilados (*javac*) e transformados em arquivos de classes (*bytecode*). Estas classes são analisadas pelo enxertador (*floggy-compiler*), que gera e adiciona o código de persistência às classes consideradas persistentes. Posteriormente, as classes passam pelo processo de pré-verificação[18] (*preverify*) e são empacotadas (*jar*) em um arquivo Java (.jar) junto com o módulo *Framework*. Este arquivo é instalado e executado nos dispositivos MIDP. Durante a execução da aplicação, as operações de persistência são gerenciadas pelo módulo *Floggy*, que utiliza os *Record Stores* para armazenar os objetos.

[18] Processo responsável por enxertar elementos nos arquivos de classe (.class) garantindo que o código da aplicação J2ME segue determinada especificação de configuração e perfil.

5.2 Requisitos

Os objetivos principais deste *framework* de persistência são facilidade de uso, facilidade de aprendizado, bom desempenho e, principalmente, cumprimento aos requisitos de uma camada de persistência (ver seção 3.4). Entretanto, os requisitos de uma camada de persistência para aplicações tradicionais não são necessariamente os mesmos para dispositivos móveis, visto que tais dispositivos possuem restrições quanto ao armazenamento, memória, processamento e conectividade.

Considerando as limitações dos dispositivos móveis, os requisitos de uma camada de persistência (ver seção 3.4) foram reavaliados de forma a criar um conjunto de requisitos que atendesse as principais necessidades de uma camada de persistência em um ambiente de execução para dispositivos móveis. Abaixo estão listados os requisitos selecionados:

- ***Encapsulamento completo da camada de dados:*** O desenvolvedor deve utilizar-se apenas de mensagens de alto nível, como “salvar” e “excluir", para tratar a persistência de objetos, deixando a camada de persistência realizar a conversão destas mensagens em ações sobre os dados. Não é necessário que o desenvolvedor entre em detalhes de como efetuar qualquer uma das operações de persistência.

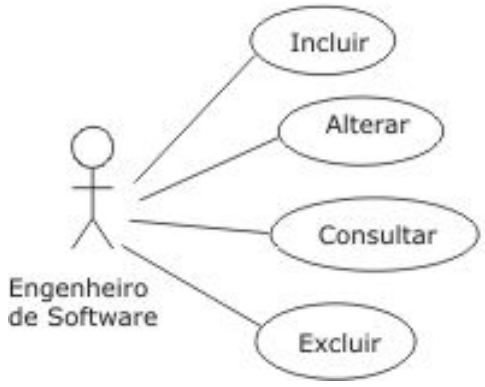

Figura 5.2 – Mensagens de Alto Nível [MAGALHÃES 2005]

- ***Persistência Seletiva*:** deve permitir a persistência apenas dos atributos desejados de um objeto, minimizando o espaço armazenado com atributos transientes.

- ***Extensibilidade*:** deve permitir a adição de novas classes ao esquema e a modificação fácil no mecanismo de persistência.

- ***Identificadores de Objetos*:** Garantia de que a aplicação trabalhará com objetos de identidade única.

- ***Consulta a Objetos:*** Fornecer um mecanismo de consulta eficiente, que permita o acesso direto aos dados. Assim, são permitidas consultas com um grau de complexidade maior que o usual.

- ***Cursores*:** Fornecer mecanismo otimizado para recuperação de objetos a fim de evitar que muitos objetos sejam carregados de uma única vez em memória. Assim, uma lista de objetos obtida através de uma consulta, armazenará somente as referências aos objetos, permitindo que os mesmos sejam recuperados posteriormente.

5.3 Arquitetura

A arquitetura do *Floggy* foi elaborada considerando as limitações computacionais dos dispositivos móveis e as restrições apresentadas pelo perfil MIDP.

Nas plataformas tradicionais, a persistência dos dados é feita geralmente através da serialização dos dados em arquivos, da transformação dos dados em XML ou, mais comumente, da linguagem declarativa SQL aplicada a um SGBD.

Uma das limitações impostas pelo perfil MIDP deve-se ao fato de que a persistência dos dados, na grande maioria das aplicações, deve ser feita através do uso do RMS (ver capítulo 4), o que implica na transformação de objetos em seqüências de bytes (serialização).

Outra limitação imposta pelo perfil MIDP é a inexistência de um mecanismo de reflexão. Este mecanismo é utilizado pela maioria das camadas de persistência tradicionais para acessar informações internas das classes – métodos, atributos e construtores – em tempo de execução. Logo, a falta de suporte a um mecanismo de reflexão impede que o *Floggy* siga as implementações existentes das camadas de persistência tradicionais.

As limitações computacionais impedem, por exemplo, que o mapeamento das classes e atributos persistentes seja feito através de arquivos XML. Isto porque a interpretação de XML em tempo de execução pode ocasionar perda de desempenho e ocupação extra de memória.

Para suprir as limitações da falta de um mecanismo de reflexão e a baixa performance no processamento de arquivos XML, o código de persistência dos objetos e a lógica de mapeamento são gerados pelo *Floggy Compiler* através do uso de reflexão e alteração do *bytecode* após o processo de compilação.

5.3.1 Estrutura de classes

Devido às limitações de memória e armazenamento, as aplicações MIDP são geralmente pequenas. Este limitador foi fator determinante para a escolha da arquitetura do módulo *Framework*, pois este deveria atender, através de um conjunto limitado de classes, aos requisitos de uma camada de persistência.

As classes e interfaces do módulo *Framework* estão apresentadas no diagrama de classes (ver Figura 5.3).

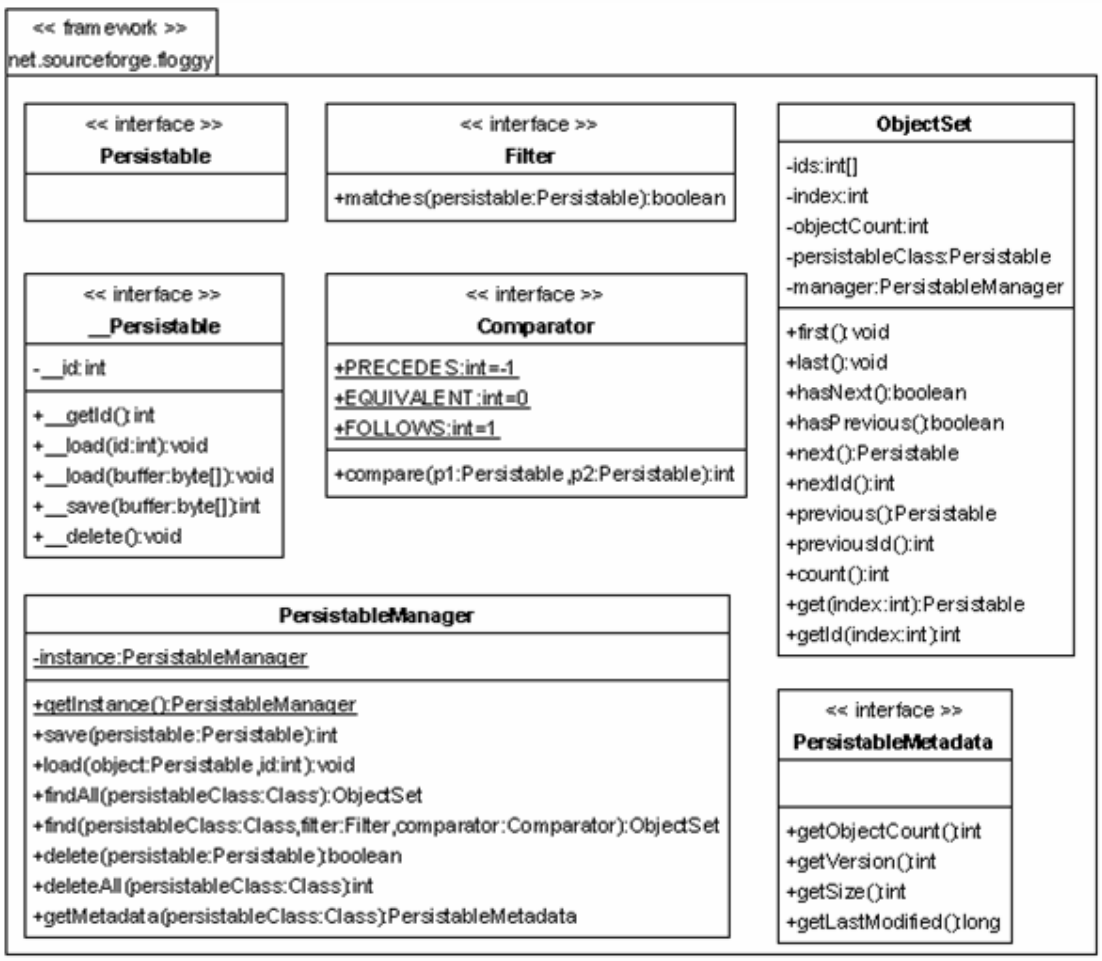

Figura 5.3 - Diagrama de classe do módulo *Framework*

5.3.1.1 Persistable

Esta interface funciona como um marcador e não obriga que nenhum método seja implementado. As classes que implementam direta ou indiretamente a interface *Persistable* são consideradas persistentes pelo *Floggy*.

5.3.1.2 __Persistable

O desenvolvedor não deve fazer uso direto da interface *__Persistable*, pois ela e seus métodos são adicionados ao *bytecode* das

classes consideradas persistentes pelo *Enxertador* (ver seção 5.4). Todas as operações de persistência são acionadas pelo gerenciador de persistência *PersistableManager* através dos métodos desta interface.

5.3.1.3 PersistableManager

Esta classe atua como o gerenciador de persistência e pode ser considerada a principal classe do *framework*. É através dela que todas as operações de persistência – armazenar, carregar, remover e executar buscas – são executadas.

Os métodos que dão suporte a tais operações são:

- **load(Persistable persistable, int id):** permite que um objeto seja carregado do repositório para a memória através do identificador de persistência (OID);
- **save(Persistable persistable):** permite que um objeto seja armazenado no repositório, atribuindo a ele um identificador de persitência (OID);
- **findAll(Class persistableClass):** retorna todos os objetos armazenados no repositório da classe passada como parâmetro;
- **find(Class persistableClass, Filter filter, Comparator comparator):** permite que sejam realizadas buscas em repositórios de objetos de mesma classe. Podem ser definidos critérios de busca (ver seção 5.3.1.4) e a forma de ordenação do resultado (ver seção 5.3.1.5);
- **deleteAll(Class persistableClass):** remove todos os objetos de uma mesma classe;
- **delete(Persistable persistable):** permite a exclusão do objeto armazenado no repositório;
- **getMetadata(Class persistableClass):** retorna os metadados (ver seção 5.3.1.7) de uma classe persistente.

5.3.1.4 Filter

As classes que implementam a interface *Filter* são responsáveis por definir os critérios de uma seleção. Para isto, é necessário implementar o método `matches(Persistable persistable)`.

Estas classes são passadas como parâmetro para o método `find(Class persistableClass, Filter filter, Comparator comparator)` da classe *PersistableManager*, responsável pela pesquisa dos objetos e ordenação do resultado.

5.3.1.5 Comparator

As classes que implementam a interface *Comparator* são responsáveis por definir os critérios de ordenação do resultado de uma seleção. Para isto, faz-se necessário a implementação do método `compare(Persistable p1, Persistable p2)`.

Estas classes são passadas como parâmetro para o método `find(Class persistableClass, Filter filter, Comparator comparator)` da classe *PersistableManager*, responsável pela pesquisa dos objetos e ordenação do resultado.

5.3.1.6 ObjectSet

Os resultados de seleções resultam em objetos da classe *ObjectSet*. Para otimizar o uso de memória, uma das principais restrições do ambiente MIDP, esta classe funciona como um cursor e contém uma lista de identificadores de persistência (OID) de objetos obtidos através de uma seleção de objetos realizada utilizando o método `find(Class persistableClass, Filter filter, Comparator comparator)` da classe *PersistableManager*.

5.3.1.7 PersistableMetadata

Esta interface agrupa informações estatísticas sobre os objetos de determinada classe no repositório. As informações disponíveis são: número de objetos no repositório, data de última modificação do repositório, número de bytes ocupados pelos objetos, e versão do repositório.

5.3.2 Processo de mapeamento

O processo de mapeamento é composto por três grandes etapas (ver seção 3.5): o mapeamento de atributos, o mapeamento de classes e hierarquias e o mapeamento de relacionamentos.

5.3.2.1 Mapeamento de atributos

Um objeto pode conter diversos atributos, dentre os quais alguns devem ser persistidos e outros não. Por exemplo, os atributos de um objeto Pessoa são: nome, data de nascimento e idade. O atributo idade não precisa ser persistido, pois o mesmo poderia ser calculado pela própria aplicação, através da data de nascimento e data do sistema, conforme ilustrado na Figura 5.4.

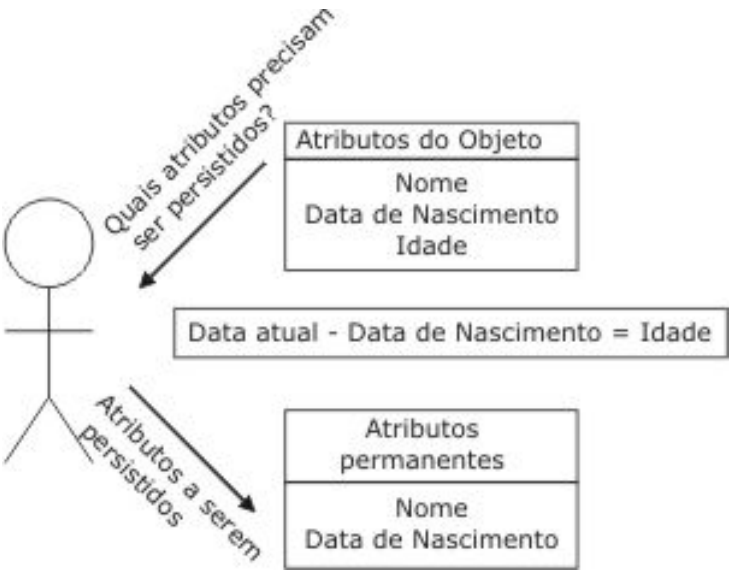

Figura 5.4 - Mapeamento de atributos [MAGALHÃES 2005]

Atributos que não devem ser persistidos são considerados transientes. Para que estes atributos não sejam persistidos pelo *Floggy* devem fazer uso da palavra reservada **transient**.

Os atributos estáticos têm seu valor associado à classe e não a uma instância de objeto e, por este motivo, também não são persistidos pelo *Floggy*.

```
public class Pessoa implements
    net.sourceforge.floggy.Persistable {

    static int numeroPessoas = 0;

    String nome;

    Date dataNascimento;

    transient int idade;

    ...
}
```

Exemplo 5.1 - Mapeamento de atributos

O Exemplo 5.1 mostra o código-fonte para o mapeamento de atributos para a classe Pessoa. Neste caso, somente os atributos **nome** e **dataNascimento** são persistidos pelo *Floggy*.

Os tipos de dados suportados pelo *Floggy* são:

- Tipos Primitivos: boolean, byte, char, double, float, int, long, short;
- Classes Empacotadoras de Tipos (*type wrappers*): java.lang.Boolean, java.lang.Byte, java.lang.Character, java.lang.Double, java.lang.Float, java.lang.Integer, java.lang.Long e java.lang.Short;
- Classes java.lang.String e java.lang.Date;
- Coleções: Arrays e java.util.Vector;
- Persistentes: objetos de classes que implementam direta ou indiretamente a interface *Persistable*.

5.3.2.2 Mapeamento de classes e hierarquias

Assim como os atributos, nem todas as classes de uma aplicação devem ser consideradas como persistentes. O *Floggy* identifica como classe persistente àquela que implementa direta ou indiretamente a interface *Persistable*. No exemplo a seguir, a classe *Pessoa* e a classe *Medico* são consideradas persistentes. A classe *Pessoa* implementa diretamente a interface *Persistable*, enquanto a classe *Medico* implementa a interface de forma indireta, pois herda a interface de sua superclasse.

```
public class Pessoa implements
    net.sourceforge.floggy.Persistable {

    String nome;

    Date dataNascimento;

}

(...)

public class Medico extends Pessoa {

    String crm;

    Vector formacoes;

}
```

Exemplo 5.2 - Implementando a interface *Persistable* direta e indiretamente

O *Floggy* suporta ainda a persistência de objetos de classes que estendem classes não persistentes, isto é, classes que não implementam a interface *Persistable*. Neste caso, os atributos herdados destas classes não são persistidos pelo *Floggy*.

```
public class Animal {

    String som;

}

public class Cachorro extends Animal implements
    net.sourceforge.floggy.Persistable {

    String nome;

    String raça;

}
```

Exemplo 5.3 – Classes persistentes que estendem classes não persistentes

No Exemplo 5.3, a classe *Animal* é considerada não persistente. Logo os atributos persistidos são somente aqueles que pertencem à classe *Cachorro* (nome e raça).

Depois de entender como o *Floggy* identifica uma classe persistente, é preciso compreender como estas classes são mapeadas no repositório de uma aplicação MIDP.

Em aplicações MIDP, a estrutura utilizada para armazenar dados chama-se *Record Store* (ver seção 4.1). Estas estruturas podem ser comparadas a uma tabela de um banco de dados relacional, pois armazenam os dados na forma de registros.

O *Floggy* cria um *Record Store* para cada classe persistente. Todos os objetos de uma mesma classe persistente ficam armazenados no *Record Store* correspondente (ver Figura 5.5).

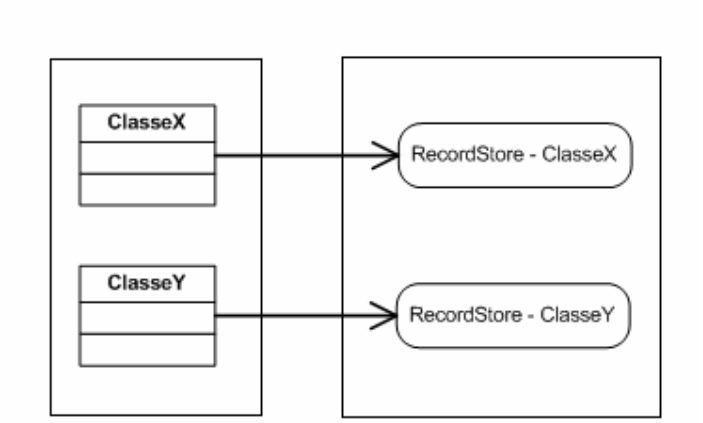

Figura 5.5 - Mapeamento de classes em *Record Stores*

Conforme descrito na seção 3.5.2, existem três formas de mapear as hierarquias de classe: uma tabela por hierarquia, uma tabela por classe concreta e uma tabela por classe.

O mapeamento de uma tabela por hierarquia fere as regras da teoria de modelagem de dados, pois apresenta ausência de normalização. Além disto, hierarquias com muitas especializações podem apresentar muitos campos com valores nulos.

O mapeamento de uma tabela por classe concreta resulta na redundância de dados, pois os atributos definidos em uma classe abstrata são criados em todas as tabelas que representam as classes filhas.

O mapeamento de uma tabela por classe busca, ao máximo, manter a normalização de dados. Desta forma, a estrutura final das tabelas fica semelhante a representação UML das classes.

O *Floggy* faz o mapeamento de hierarquia da última forma, isto é, mapeando cada classe persistente (concreta ou abstrata) em um *Record Store*. Para manter a referência entre as classes, o identificador único da classe pai é armazenado.

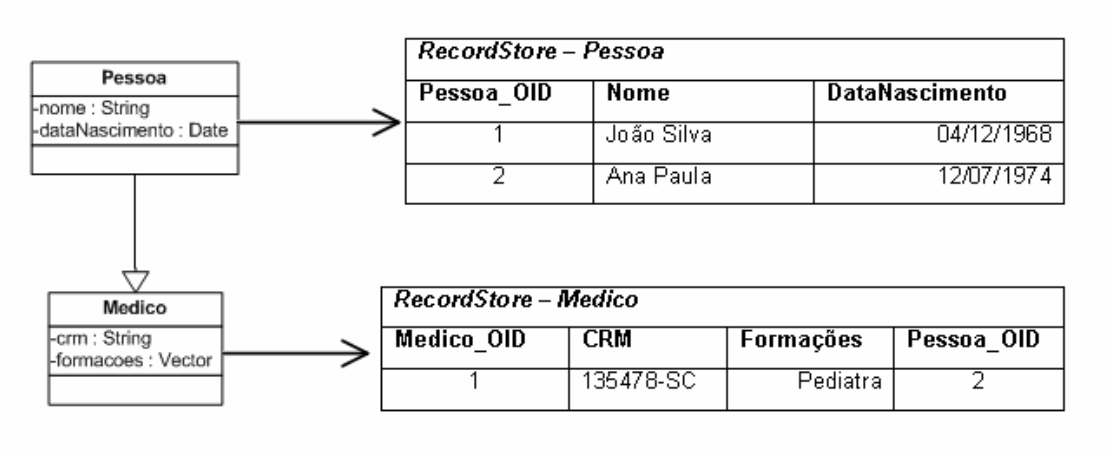

Figura 5.6 - Mapeamento de hierarquia de classes

5.3.2.3 Mapeamento de relacionamentos

Os objetos são estruturas complexas formadas não somente pelos dados que contêm, mas também pelos relacionamentos estabelecidos com outros objetos. Estes relacionamentos envolvem relações de herança, associação, agregação e composição que são mapeadas para o repositório conforme especificado na modelagem.

As associações são relações unidirecionais ou bidirecionais entre os objetos. As relações bidirecionais ocorrem quando há referência mútua entre os objetos.

Cada relacionamento possui uma cardinalidade que pode ser definida como o número de instâncias de uma classe relacionada com uma instância de outra classe.

O *Floggy* oferece suporte a todos os tipos de relacionamentos citados acima, exceto as associações bidirecionais. O desenvolvedor que utilizar o *Floggy* deve atentar para este detalhe, pois caso este tipo de relacionamento seja utilizado, o programa pode entrar em *loop* quando executadas operações de persistência e recuperação de objetos. Pretende-se eliminar esta limitação em futuras versões do *framework*.

Como pode ser visto no exemplo (Figura 5.7) a relação entre um objeto do tipo Paciente e um objeto do tipo Endereço é uma relação *um para um* (um paciente possui somente um endereço e um endereço pertence

somente a um paciente). Já a relação entre um objeto paciente e um objeto telefone é uma relação *uma para muitos* (um telefone está associado somente a um paciente, que por sua vez pode possuir vários telefones). E por último, a relação entre um objeto paciente e um objeto médico é uma relação *muitos para muitos* (um médico pode atender vários pacientes e um paciente pode ser atendido por vários médicos).

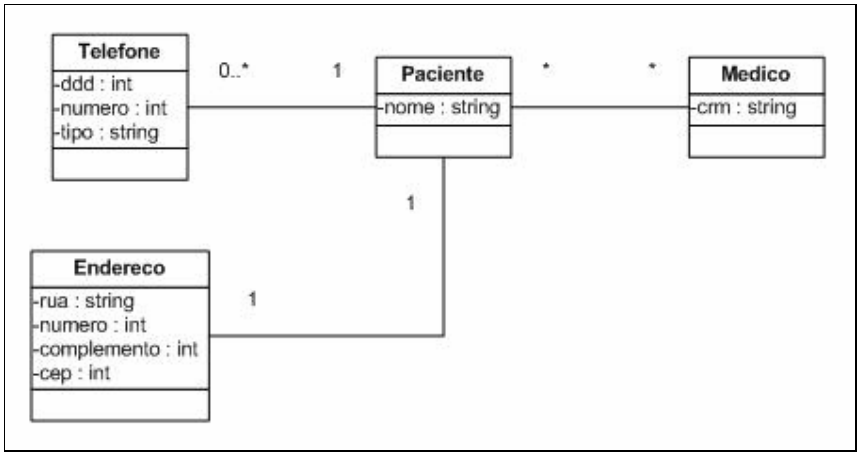

Figura 5.7 - Exemplos de relacionamentos (1-1, 1-N e N-N)

Abaixo são demonstradas as formas de mapeamento utilizando o *Floggy* para cada tipo de cardinalidade.

I. Relacionamento um para um (1-1)

Para criar um relacionamento *um para um* entre objetos deve-se escolher um dos objetos e inserir a referência para o objeto com o qual se deseja estabelecer o relacionamento.

No Exemplo 5.4, um objeto da classe paciente se relaciona com um objeto da classe endereço. Um atributo referente ao endereço é adicionado a classe paciente.

```
public class Paciente implements
    net.sourceforge.floggy.Persistable {

    String nome;
```

```
    Endereco endereco;

}

public class Endereco implements
    net.sourceforge.floggy.Persistable {

    String rua;

    int numero;

    String complemento;

    int cep;

}
```

Exemplo 5.4 - Mapeamento de relacionamento *um para um*

II. Relacionamento um para muitos (1-N)

Existem duas formas para se criar um relacionamento *um para muitos*. Na primeira forma o relacionamento é criado no objeto unitário através da adição de uma coleção do objeto não-unitário. Na segunda forma o relacionamento é criado no objeto não-unitário através da adição de uma referência simples do objeto unitário.

O Exemplo 5.5 é demonstra a primeira forma de mapeamento. O relacionamento foi criado no objeto da classe *Paciente* (objeto unitário da relação), adicionando a ele uma coleção de objetos da classe *Telefone*.

```
public class Paciente implements
    net.sourceforge.floggy.Persistable {

    String nome;

    Vector telefones; // ou Telefone[] telefones;
}

public class Telefone implements
    net.sourceforge.floggy.Persistable {

    int ddd;

    int numero;

    String tipo;

}
```

Exemplo 5.5 - Mapeamento de relacionamento *um para muitos* (primeira forma)

Já o Exemplo 5.6 demonstra a segunda forma de mapeamento. O relacionamento foi criado no objeto da classe *Telefone* (objeto não-unitário da relação), adicionando-se uma referência a um paciente.

```
public class Paciente implements
    net.sourceforge.floggy.Persistable {

    String nome;

}

public class Telefone implements
    net.sourceforge.floggy.Persistable {

    int ddd;

    int numero;

    String tipo;

    Paciente paciente;

}
```

Exemplo 5.6 - Mapeamento de relacionamento *um para muitos* (segunda forma)

III. Relacionamento muitos para muitos (N-N)

Existem duas formas para se criar um relacionamento *muitos para muitos* entre objetos. Na primeira forma o relacionamento é criado através do uso de uma classe de associação. Na segunda forma o relacionamento é criado através de uma referência múltipla entre os objetos, isto é, ambos os objetos possuem uma coleção de objetos da classe que se relacionam.

Contudo, conforme citado na seção 5.3.2.3, o *Floggy* não oferece suporte a associações bidirecionais. Com isto, a segunda forma de mapeamento não pode ser utilizada.

O Exemplo 5.7 demonstra a primeira forma de mapeamento. O relacionamento foi estabelecido através da criação da classe de associação *Atendimento* que contém referência para os dois objetos da relação. Desta forma, vários objetos da classe *Atendimento* resultam no relacionamento *muitos para muitos* desejado.

```
public class Paciente implements
    net.sourceforge.floggy.Persistable {

    String nome;

}

public class Medico implements
    net.sourceforge.floggy.Persistable {

    String crm;

}

public class Atendimento implements
    net.sourceforge.floggy.Persistable {

    Medico medico;

    Paciente paciente;
}
```

Exemplo 5.7 - Mapeamento de relacionamento *muitos para muitos*

5.3.3 Ciclo de vida

O ciclo de vida de um objeto persistente (ver Figura 5.8) é composto de três estados: transiente, persistente e não sincronizado.

Um objeto pode iniciar seu ciclo de vida no estado transiente ou persistente. Encontram-se no estado transiente as instâncias de objetos criadas e não persistidas. Já no estado persistente, encontram-se os objetos salvos ou recuperados do repositório que ainda não foram modificados, isto é, são idênticos na memória do programa e no repositório. Caso um objeto no estado persistente seja modificado, ele passará ao estado não sincronizado e só sairá deste estado caso seja salvo ou excluído.

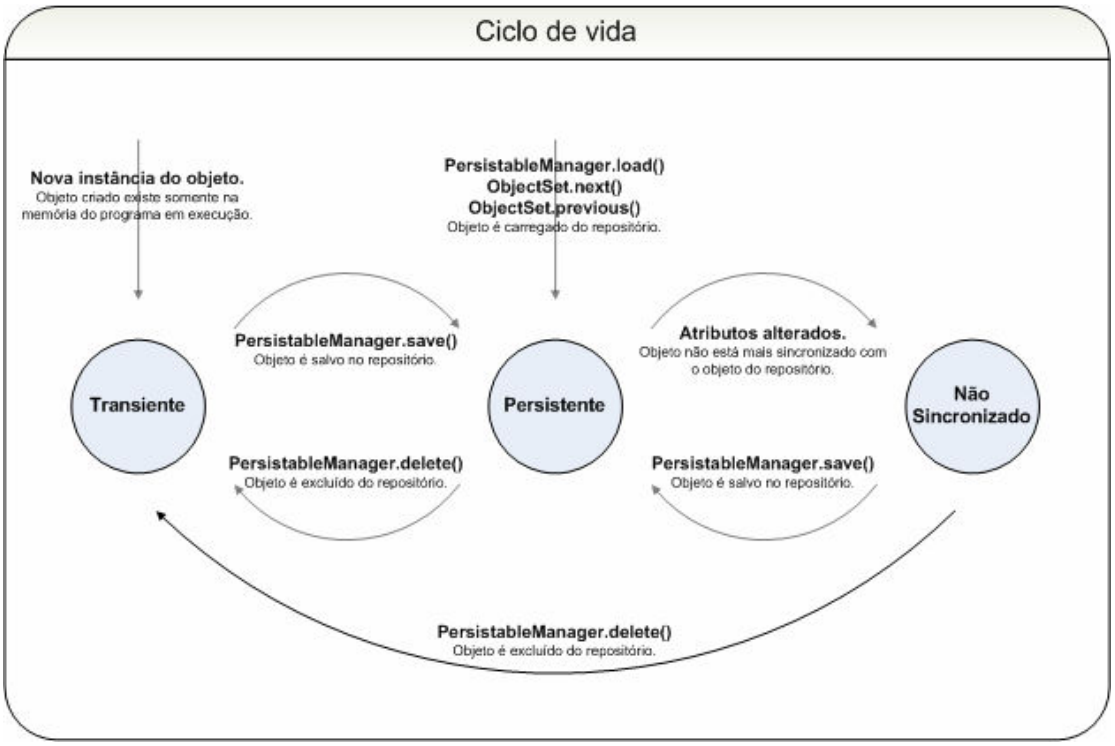

Figura 5.8 - Ciclo de vida de um objeto persistente

Abaixo estão listados os estados do ciclo de vida de um objeto persistente:

- **Transiente:** Um objeto que é instanciado sem nunca ter sido persistido encontra-se no estado transiente. Este objeto não possui identidade de persistência. Objetos no estado transiente têm o seu ciclo de vida encerrado no momento em que suas referências deixam de existir e, conseqüentemente, são coletados pelo *garbage collector*. Para que um objeto passe do estado transiente para o estado persistente e receba uma identidade de persistência, este deve ser salvo no repositório pelo gerenciador de persistência através do método **save()** ou deve estar associado a um objeto salvo da forma citada acima. Um objeto no estado *persistente* ou *não sincronizado* pode passar para o estado *transiente* caso seja excluído pelo gerenciador de persistência através do método **delete()**.

- **Persistente:** Neste estado é garantido que o objeto possui uma identidade de persistência e possui um registro correspondente idêntico no repositório. O objeto pode chegar a este estado sendo salvo pelo gerenciador de persistência através do método **save()** ou carregado através do método **load()**. Ele permanece neste estado até o momento em que seus atributos sejam alterados, passando ao estado *não sincronizado* ou caso ele seja excluído pelo gerenciador de persistência através do método **delete()**, retornando ao estado *transiente*.

- **Não sincronizado:** O objeto transita para este estado somente quando se encontra no estado *persistente* e seus atributos são alterados. Neste momento, o objeto não corresponde, de forma idêntica, ao seu respectivo registro no repositório. Para voltar ao estado *persistente*, o objeto deve ser salvo novamente pelo gerenciador de persistência através do método **save()**. Caso seja excluído do repositório pelo gerenciador de persistência através do método **delete()**, o objeto retorna ao estado *transiente*.

5.3.4 Identidade de objetos

Para melhor compreensão do conceito de identidade de um objeto é preciso esclarecer a diferença entre os conceitos de identidade e igualdade.

Na linguagem Java, o conceito de identidade está associado à posição do objeto na memória, sob o controle da máquina virtual. Ou seja, dois objetos são considerados idênticos se suas referências apontarem para a mesma posição de memória. Já o conceito de igualdade refere-se ao valor dos atributos de cada objeto, que pode ou não estar sob o controle do programador através da redefinição do método *equals()*, ou seja, dois objetos são considerados iguais se seus atributos são iguais.

O *Floggy*, por sua vez, introduz o conceito de identidade de persistência, o qual identifica unicamente um objeto no repositório. Este identificador único é também conhecido como OID (*Object Identifier*).

A geração dos OIDs é delegada ao mecanismo de persistência das aplicações MIDP, chamado *RecordStore* (ver seção 4.1). Isto é possível pela forma de mapeamento de classes e hierarquias utilizada (ver seção 5.3.2.2), que faz o mapeamento de um repositório para objetos de mesma classe.

Um objeto é associado a um OID durante a operação de inserção no repositório.

Quando um objeto é excluído do repositório ele perde sua identidade de persistência, isto é, o objeto é dissociado de seu OID. Desta forma, ele só existe na memória no programa, não possuindo nenhum objeto correspondente no repositório. Caso este objeto seja salvo, receberá novamente um OID que o associe a um objeto no repositório.

5.3.5 Manipulação de objetos

O *Floggy* é um framework de persistência explícito. Para que uma mudança em um objeto seja refletida no repositório, é necessário efetuar uma requisição indicando que tais alterações devam ser persistidas. Portanto, para manipular os objetos faz-se necessário o uso de uma instância do gerenciador de persistência *PersistableManager* (Exemplo 5.8).

```
PersistableManager pm = PersistableManager.getInstance();
```

Exemplo 5.8 - Obtendo uma instância do gerenciador de persistência

O *Floggy* encapsula no método **save()** do gerenciador de persistência as operações de incluir ou salvar objetos no repositório. Este método recebe como parâmetro o objeto a ser persistido e retorna, caso a operação seja completada com sucesso, o identificador único deste objeto no repositório.

O gerenciador de persistência, ao analisar um objeto a ser persistido, verifica se o mesmo possui uma indentidade de persistência, que é

evidenciada pelo fato do objeto possuir ou não um identificador único. Caso este objeto possua uma identidade de persistência, o gerenciador de persistência grava os dados do objeto no respectivo registro no repositório. Caso contrário, o gerenciador de persistência cria um novo registro no repositório e o associa ao objeto através do identificador de persistência.

O Exemplo 5.9 demonstra, no mesmo trecho de código, a inserção e alteração de um mesmo objeto no repositório.

```
int oid = -1; // Identificador único

Medico medico;

// Cria nova instância
medico = new Medico();
medico.setNome("João Silva");
medico.setCrm("12345-SC");

// Insere nova instancia no repositório
oid = pm.save(medico); // ex.: oid → 10

// Altera dados do médico
medico.setNome("João da Silva");
oid = pm.save(medico); // ex.: oid → 10

// Cria outra instância
medico = new Medico();
medico.setNome("Maria da Silva");
medico.setCrm("54321-SC");

// Insere nova instancia no repositório
oid = pm.save(medico); // ex.: oid → 11
```

Exemplo 5.9 – Inserindo e alterando um objeto no repositório

Um objeto persistente pode ser composto também por objetos capazes de serem persistidos. Neste caso, todos os objetos considerados persistentes são salvos automaticamente no repositório pelo *Floggy*.

O Exemplo 5.10 demonstra o relacionamento mostrado no diagrama de classes da Figura 5.7, onde um paciente possui uma coleção de telefones. Neste exemplo, tanto o paciente quanto os telefones a ele associados, serão salvos no repositório.

```
(...)

// Criando Telefones
```

```
Vector telefones = new Vector();
telefones.addElement(new Telefone(48, 32664275, "Casa"));
telefones.addElement(new Telefone(48, 91252354, "Celular"));

// Criando o Paciente
Paciente paciente = new Paciente();
paciente.setNome("Maria do Bairro");
paciente.setTelefones(telefones);

// Salva o Paciente e os Telefones
pm.save(paciente);

(...)
```

Exemplo 5.10 - Salvando um objeto complexo

Para remover objetos do repositório é necessário que estes estejam associados a uma identidade de persistência. A operação de remoção deve ser executada através do método **delete()**, que recebe como parâmetro o objeto a ser removido. Caso a operação não ocorra com sucesso, o método lança uma exceção.

Através do método **deleteAll(),**é possível remover todos os objetos de uma determinada classe,. Este método recebe como parâmetro a classe dos objetos que se deseja remover e, como resultado, retorna o número de objetos removidos do repositório.

É importante ressaltar que a remoção de objetos persistentes não ocorre em *cascata,* isto é, somente o próprio objeto é removido, enquanto os objetos associados a ele permanecem no repositório. No Exemplo 5.10, se o paciente fosse removido, os telefones associados a ele seriam mantidos no repositório.

```
// Referência ao objeto
Paciente paciente = new Paciente();
pm.load (paciente, 10);

// Remover os telefones associados ao Paciente
Vector telefones = paciente.getTelefones();
for (int i = 0; i < telefones.size(); i++) {
     Telefone tel = (Telefone) telefones.elementAt(i);
     pm.delete(tel);
}

// Remover o paciente
pm.delete(paciente);
```

```
// Remover todos os pacientes
int qtdRemovidos = pm.deleteAll(Paciente.class);
```

Exemplo 5.11 - Removendo objetos do repositório

5.3.6 Seleção de objetos

O *Floggy* permite selecionar e recuperar os dados armazenados no repositório de três formas:

- Seleção direta de um objeto utilizando identificador único (ver seção 5.3.4) através do método **load()** do gerenciador de persistência;
- Seleção de todos os objetos de uma classe através do método **findAll()** do gerenciador de persistência;
- Seleção de objetos utilizando um filtro (ver seção 5.3.1.4) através do método **find()** do gerenciador de persistência.

O exemplos ilustrados nas próximas subseções utilizam objetos das classes *Medico* e *Formacao*, assim definidas:

```
class Medico implements Persistable {
    String nome;
    String crm;
    float salario;
    Formacao formacao;
}
```

```
class Formacao implements Persistable {
    String descricao;
}
```

Exemplo 5.12 – Definição das classes Medico e Formacao

5.3.6.1 Seleção de objetos através de identificador único

A forma mais simples e eficiente de selecionar um objeto do repositório é utilizando seu OID. Esta operação pode ser realizada através do método **load()** do gerenciador de persistência. Os parâmetros necessários são uma instância do objeto da classe e o identificador único (ver Exemplo 5.13).

```
(...)

Medico medico =  new Medico();
try {
    persitableManager.load(medico, 10);
}
match (FloggyException e) {
    // Objeto não encontrado
}

(...)
```

Exemplo 5.13 - Utilizando o identificador único para buscar um objeto

Caso não exista um objeto no repositório cujo identificador foi passado como parâmetro, este método indicará uma exceção.

5.3.6.2 Seleção de todos os objetos de uma classe

A forma mais abrangente de selecionar objetos é através do método **findAll()**. Este método retorna todos os objetos de uma determinada classe.

Para não sobrecarregar a memória do sistema, o resultado da seleção é encapsulado dentro de um cursor, representado pela classe *ObjectSet*. Um cursor fornece um mecanismo otimizado para recuperação de objetos a fim de evitar que muitos objetos sejam carregados de uma única vez em memória, armazenando somente as referências aos objetos, permitindo que os mesmos sejam recuperados posteriormente.

No exemplo abaixo são evidenciados os vários métodos e formas diferenciadas para acessar os objetos através do *ObjectSet*, ficando a critério do desenvolvedor escolher aquela que lhe parecer mais adequada.

```
ObjectSet medicos = pm.findAll(Medico.class);

// Iterando os objetos através do método get()
for (int i = 0; i < medicos.count(); i++) {
     Medico medico = (Medico) medicos.get(i);
     System.out.println(medico.getNome());
}

// Iterando os objetos através do método next();
while (medicos.hasNext()) {
   Medico medico = (Medico) medicos.next();
   System.out.println(medico.getNome());
}
```

```
// Iterando os objetos através do método nextId();
// Uso otimizado da memória; somente uma instância é utilizada
Medico medico = new Medico();
while (medicos.hasNext()) {
   pm.load(medico, medicos.nextId());
   System.out.println(medico.getNome());
}
```

Exemplo 5.14 - Selecionando todos os objetos de uma classe

5.3.6.3 Seleção de objetos através de filtros

Em algumas ocasiões deseja-se buscar objetos de forma mais qualificada, seguindo determinados critérios. Com o *Floggy* isto é possível através do método **find()**, que recebe como parâmetro um objeto da classe *Filter* e outro objeto da classe *Comparator* para ordenação dos resultados (ver seção 5.3.6.4).

O Exemplo 5.15 demonstra a seleção de todos os médicos com salário igual ou superior a R$ 2.000,00.

```
class FiltroMedicoSalario implements Filter {

    public boolean matches(Persistable objeto) {
        Medico medico = (Medico) objeto;
        return medico.getSalario() >= 2000.00;
    }

}
```

```
(...)

ObjectSet medicos =
    pm.find(Medico.class, new FiltroMedicoSalario(), null);

// Iterando os objetos selecionados através do método get()
for (int i = 0; i < medicos.count(); i++) {
     Medico medico = (Medico) medicos.get(i);
     System.out.println(medico.getNome());
}
```

Exemplo 5.15 - Selecionando os objetos de uma classe através de filtro (parte 1)

O Exemplo 5.16 demonstra a seleção de todos os médicos de determinada especialidade.

```
class FiltroFormacao implements Filter {
```

```
    String formacao;

    public FiltroFormacao(String formacao) {
        this.formacao = formacao;
    }

    public boolean matches(Persistable objeto) {
        Medico medico = (Medico) objeto;
        return
            this.formacao.equals(
                 medico.getFormacao().getDescricao());
    }

}
```

```
(...)

ObjectSet medicos =
    pm.findAll(
        Medico.class, new FiltroFormacao("Ortopedia"), null);

// Iterando os objetos selecionados através do método get()
for (int i = 0; i < medicos.count(); i++) {
     Medico medico = (Medico) medicos.get(i);
     System.out.println(medico.getNome());
}
```

Exemplo 5.16 - Selecionando os objetos de uma classe através de filtro (parte 2)

5.3.6.4 Seleção de objetos com ordenação

É muito comum que os dados de uma seleção retornem de forma ordenada. Com o *Floggy* isto é possível através do uso de um objeto da classe *Comparator* quando se realiza uma seleção através do método **find()**.

```
class OrdenacaoPorSalario implements Comparator {

    public int compare(Persistable p1, Persistable p2) {
        Medico m1 = (Medico) p1;
        Medico m2 = (Medico) p2;

        if(m1.getSalario() > m2.getSalario()) {
            return Comparator.PRECEDES;
        }
        if(m1.getSalario() < m2.getSalario()) {
            return Comparator.FOLLOWS;
        }
        return Comparator.EQUIVALENT;
    }
}
```

```
(...)

ObjectSet medicos =
```

```
    pm.findAll(Medico.class, null, new OrdenacaoPorSalario());

// Iterando os objetos ordenados por salário
for (int i = 0; i < medicos.count(); i++) {
     Medico medico = (Medico) medicos.get(i);
     System.out.println(medico.getNome());
}
```

Exemplo 5.17 - Ordenando objetos

5.4 Enxertador

O módulo *Floggy Compiler* é o programa responsável por gerar e enxertar o código de persistência nas classes de uma aplicação MIDP. Somente as classes consideradas persistentes, isto é, aquelas que implementam a interface *Persistable*, têm seu código modificado por este módulo.

O código de persistência, contendo as operações de armazenamento, recuperação e exclusão de objetos, é enxertado nos arquivos de classe (.class) na forma de *bytecode* após o processo de compilação[19] dos arquivos fontes e antes do processo de pré-verificação[20] das classes (ver Figura 5.9).

Ao enxertar o código de persistência nos arquivos de classes, a correspondência entre as linhas do código fonte original e do código incrementado é mantida, não prejudicando assim a depuração *(debugging)* da aplicação.

[19] Processo responsável por gerar o bytecode das classes (os arquivos .class) a partir dos arquivos fonte (.java)

[20] Processo responsável por enxertar elementos nos arquivos de classe (.class) garantindo que o código da aplicação J2ME seguirá determinada especificação de configuração e perfil.

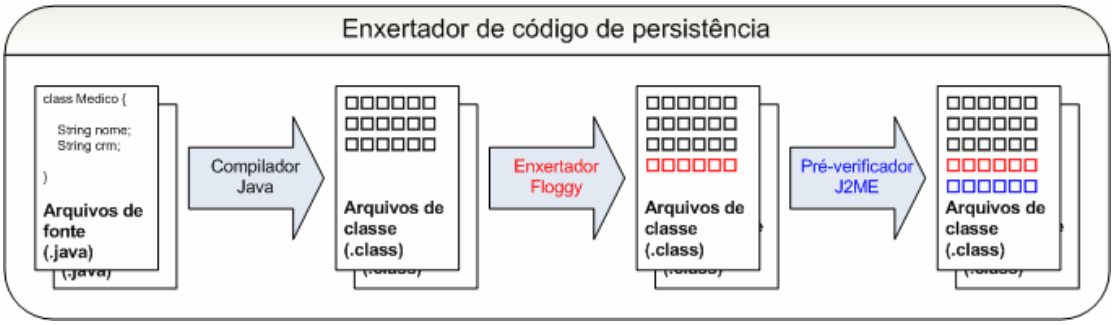

Figura 5.9 - Processo de geração do código de persistência

Para enxertar o *bytecode* nas classes foi utilizada a biblioteca *Javassist*, versão 2.6, fornecida pela JBoss (http://www.jboss.com/products/javassist). Esta biblioteca é formada por um conjunto de classes que fornecem suporte a edição de *bytecodes* em Java, permitindo a criação e modificação de uma ou mais classes em tempo de execução.

Conforme citado, todo código de persistência gerado é ilegível por ser enxertado nos arquivos de classe na forma de *bytecode*. O exemplo abaixo visa demonstrar o conteúdo gerado pelo *Enxertador*, se o mesmo incrementasse os arquivos de código fonte ao invés dos arquivos de classe.

```
public class Medico implements Persistable, __Persistable {

    String nome;

    String crm;

    private int __id = -1;

    public int __getId() {
        return this.__id;
    }

    public void __load(byte[] buffer) throws java.lang.Exception {
        java.io.ByteArrayInputStream bais = new java.io.ByteArrayInputStream(
                buffer);
        java.io.DataInputStream dis = new java.io.DataInputStream(bais);
        if (dis.readByte() == 0) {
            nome = dis.readUTF();
        } else {
            nome = null;
        }
        if (dis.readByte() == 0) {
            crm = dis.readUTF();
        } else {
            crm = null;
        }
        dis.close();
        return;
    }
```

```
    public void __load(int id) throws java.lang.Exception {
        javax.microedition.rms.RecordStore rs =
net.sourceforge.floggy.PersistableManager
                .getRecordStore("Medico");
        byte[] buffer = rs.getRecord(id);
        rs.closeRecordStore();
        this.__load(buffer);
        this.__id = id;
    }

    public int __save() throws java.lang.Exception {
        java.io.ByteArrayOutputStream baos = new
java.io.ByteArrayOutputStream();
        java.io.DataOutputStream dos = new java.io.DataOutputStream(baos);
        javax.microedition.rms.RecordStore rs =
net.sourceforge.floggy.PersistableManager
                .getRecordStore("Medico");
        if (nome == null) {
            dos.writeByte(1);
        } else {
            dos.writeByte(0);
            dos.writeUTF(nome);
        }
        if (crm == null) {
            dos.writeByte(1);
        } else {
            dos.writeByte(0);
            dos.writeUTF(crm);
        }
        if (this.__id == -1) {
            this.__id = rs.addRecord(baos.toByteArray(), 0, baos.size());
        } else {
            rs.setRecord(this.__id, baos.toByteArray(), 0, baos.size());
        }
        rs.closeRecordStore();
        dos.close();
        return this.__id;
    }

    public void __delete() throws java.lang.Exception {
        if (this.__id == -1) {
           throw new FloggyException("Object wasn't found in the repository.");
        }

        javax.microedition.rms.RecordStore rs =
net.sourceforge.floggy.PersistableManager
                .getRecordStore("Medico");
        rs.deleteRecord(this.__id);
        rs.closeRecordStore();

        this.__id = -1;
    }
}
```

Exemplo 5.18 - Código de persistência gerado pelo *Enxertador*

5.5 Resumo

Este capítulo apresenta o *Floggy*, uma camada de persistência composta pelos módulos *Framework* – módulo que funciona embutido com a aplicação e é responsável por gerenciar todas as operações de persistência – e *Compiler* – módulo responsável por gerar e enxertar o código de persistência.

O *Floggy* contempla grande parte dos requisitos de uma camada de persistência considerando as limitações impostas pelos ambientes de execução dos dispositivos móveis.

No próximo capítulo é descrita uma aplicação desenvolvida utilizando o *Floggy* como estudo de caso para validação deste framework.

6 - Estudo de Caso

O objetivo deste estudo de caso é a validação do *Floggy* e a demonstração da utilização do mesmo em uma aplicação real, sendo irrelevante uma discussão sobre a modelagem do problema proposto. Portanto, modelou-se um problema real que abrange o processo de mapeamento de atributos, classes, hierarquias e relacionamentos.

Na seção 6.1 é apresentada a aplicação e suas principais funcionalidades. Na seção 6.2 é apresentada a forma como é feita a persistência dos dados utilizando o *Floggy*.

6.1 Aplicação

O problema proposto consiste em desenvolver um sistema de controle das internações realizadas em um hospital. Neste sistema seria possível cadastrar médicos e suas formações, pacientes, leitos e registrar internações e altas.

A aplicação construída poderá ser executada em um dispositivo móvel, pois foi desenvolvida para plataforma J2ME/MIDP, voltada para o desenvolvimento de aplicações para este tipo de dispositivo.

Por exemplo, uma enfermeira equipada com um computador de mão ou um celular, teria em suas mãos todas as informações dos pacientes internados, leitos disponíveis, qual médico é o responsável por determinado paciente, o motivo da internação e outras informações pertinentes.

O diagrama de classe da Figura 6.1 apresenta a modelagem das classes do domínio do problema e os relacionamentos entre estas classes.

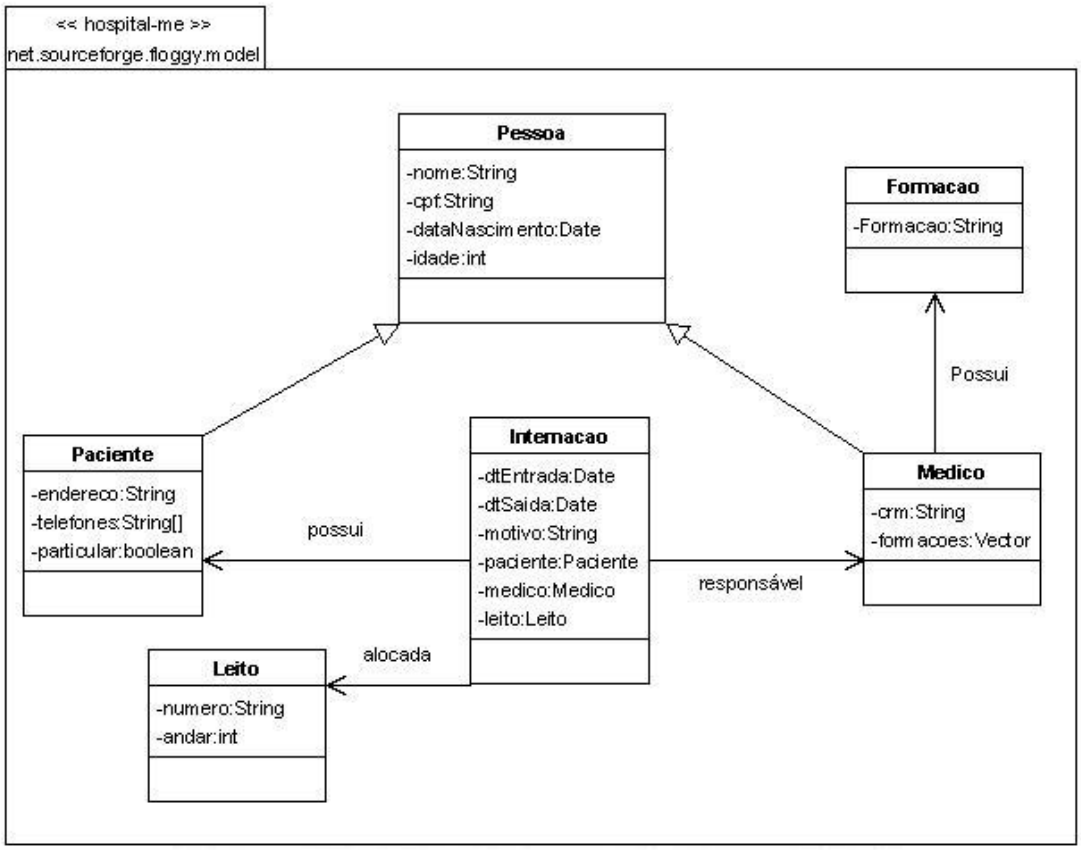

Figura 6.1 - Diagrama das classes do domínio do problema

De acordo com a modelagem, seis classes foram criadas: *Pessoa*, *Médico*, *Paciente*, *Leito*, *Formacao* e *Internacao*.

A classe *Pessoa* possui os atributos nome, CPF, data de nascimento e idade, sendo este último calculado com base no valor informado na data de nascimento. As classes *Medico* e *Paciente* são especializações da classe *Pessoa*, sendo que a classe *Medico* possui o atributo CRM e uma coleção de formações e a classe *Paciente* possui um endereço, uma lista de telefones, e o tipo de convênio (particular ou plano de saúde). A classe *Leito* é uma classe simples que possui apenas um número identificador do leito e o andar no qual este se localiza. E por último, a classe Internacao, que possui os atributos data de internação e alta, o motivo da internação, o paciente internado, o médico responsável e o leito alocado.

A Figura 6.2 demonstra algumas telas da aplicação desenvolvida.

Figura 6.2 - Imagens da aplicação sendo executada em um *emulador* de celular

A imagem da esquerda é a tela principal do sistema. O menu apresentado por esta tela leva aos cadastros de médicos, pacientes, leitos e formações, à inclusão de internações, à liberação do paciente através de alta e aos relatórios. A imagem da direita é a tela de inclusão e alteração de cadastro de um paciente.

Para demonstrar também a portabilidade da aplicação entre diferentes plataformas, a Figura 6.3 apresenta a aplicação sendo executada em um simulador do sistema operacional PalmOS[21].

[21] http://www.palmsource.com/palmos/

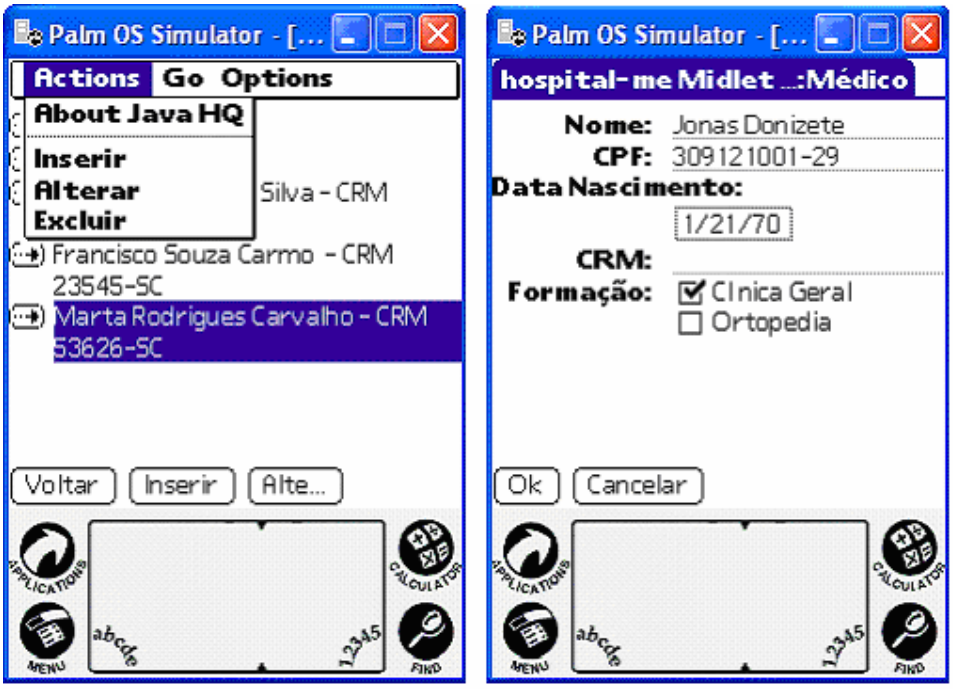

Figura 6.3 - Imagens da aplicação sendo executada em um *emulador* de Palm

Na Figura 6.3, a imagem da esquerda é a tela de cadastros de médicos, onde são listados todos os médicos cadastrados no repositório. O usuário tem a opção de incluir, alterar ou excluir um médico. A imagem da direita é a tela de inclusão ou alteração de um médico.

Figura 6.4 - Imagens da aplicação sendo executada em um *emulador* de Palm

Na Figura 6.4, a imagem da esquerda é a tela de internação de um paciente, onde o usuário informa a data de entrada, o motivo da internação, o nome do paciente, o médico responsável e, por último, o leito alocado. A imagem da direita é a tela de liberação de um paciente internado. Nela são listados todos os pacientes internados no momento, e através dela, o usuário pode escolher um paciente e cadastrar sua saída.

6.2 Implementando a aplicação com o Floggy

A aplicação delega as responsabilidades de persistência, recuperação e pesquisa dos dados ao *Floggy*.

Inicialmente, um projeto de aplicação para J2ME/MIDP foi criado contendo somente a implementação do problema proposto sem utilizar qualquer mecanismo de persistência.

Para utilizar as classes e interfaces do *Floggy* foi necessário adicionar ao projeto o módulo *Framework*, através da inclusão do arquivo *floggy-framework.jar* no *classpath* do projeto. Os arquivos de classe contidos

neste módulo devem ser empacotados junto com os arquivos de classe e imagens da aplicação.

Com o projeto configurado, o próximo passo foi adicionar aos objetos a capacidade de persistência através da implementação da interface *Persistable*. As classes consideradas persistentes e que foram modificadas são: *Pessoa, Leito, Formacao* e *Internacao* (ver Exemplo 6.1). As classes *Medico* e *Paciente* herdaram a capacidade de persistência da classe *Pessoa*. Em seguida, todos os atributos que não deveriam ser persistidos foram marcados como transientes através da palavra reservada *transient*. No caso da aplicação, apenas o atributo *idade* da classe *Pessoa* não deveria ser persistido, tendo em vista que este valor é calculado com base na data de nascimento (ver Exemplo 6.1).

```
public class Pessoa implements
    net.sourceforge.floggy.Persistable {

    String cpf;

    String nome;

    Date dataNascimento;

    transient int idade;

    ...
}
```

Exemplo 6.1 - Adicionando capacidade de persistência à classe *Pessoaa*

Após a marcação das classes como persistentes, o próximo passo foi a implementação das buscas e operações de manipulação dos objetos no repositório.

Como pode ser observado na Figura 6.3, as telas de cadastros da aplicação – Médico, Paciente, Formação e Leito – foram feitas a partir de uma listagem dos objetos cadastrados e fornecem as opções de inclusão, alteração e exclusão. Para realizar as listagens dos dados cadastrados, são utilizados os métodos **find()** e **findAll()** da classe *PersistableManager*. Em algumas ocasiões, é necessário implementar a classe *Comparator* para a ordenação dos dados.

Com os cadastros básicos desenvolvidos, passamos ao cadastro das internações e liberações de pacientes que podem ser observados na Figura 6.4.

Todas as operações de cadastro, sejam elas inclusão, alteração ou exclusão dos objetos, foram realizadas através dos métodos **save()** e **delete()** da classe *PersistableManager*.

Os relatórios foram a última etapa de implementação da aplicação. Para este tópico foram necessárias consultas mais complexas utilizando objetos da classe *Filter*, envolvendo mais de um repositório de objetos simultaneamente. O Exemplo 6.2 demonstra a forma como foi desenvolvida a seleção de todos os leitos livres, ou seja, todos os leitos que não estão sendo utilizados por pacientes internados.

```
public class FiltroInternacoes implements Filter {
   public boolean matches(Persistable p) {
       return ((Internacao) p).getDtSaida() == null;
   }
}
```

```
public class FiltroLeitosLivres implements Filter {

    ObjectSet internacoes;
    Internacao internacao;
    Leito leito;

    public FiltroLeitosLivres() {
        internacoes =
            PersistableManager.getInstance().find(
                Internacao.class,
                new FiltroInternacoes(),
                null);
    }

    public boolean matches(Persistable p) {
        leito = (Leito) p;

        internacoes.first();
        while(internacoes.hasNext()) {
            internacao = (Internacao) internacoes.next();
            if(internacao.getLeito().getNumero().equals(leito.getNumero())) {
                return true;
            }
        }
        return false;
    }
}
```

```
(...)

    PersistableManager pm = PersistableManager.getInstance();
    ObjectSet leitosLivres =
        pm.find(Leito.class, new FiltroLeitosLivres(), null);

    Leito leito = new Leito();
    while (leitosLivres.hasNext()) {
        pm.load(leito, leitosLivres.nextId());
        System.out.println(leito.getNumero() + " - " + leito.getAndar(), null);
    }
```

```
(...)
```

Exemplo 6.2 - Relatórios envolvendo consultas complexas

6.3 Enxertando o código de persistência na aplicação

Concluída a implementação do código da aplicação, é necessário utilizar o módulo *Compiler* para enxertar o código de persistência nas classes definidas como persistentes. Conforme citado na seção 5.4, o código de persistência é enxertado nos arquivos de classe (.class) na forma de *bytecode* após o processo de compilação dos arquivos fontes e antes do processo de pré-verificação das classes.

A aplicação enxertadora de código é disponibilizada através do arquivo *floggy-compiler-j2me-midp-1.0.jar*. Para enxertar código de persistência nos arquivos de classe do projeto é necessário informar à aplicação o *classpath*, incluindo o caminho dos arquivos a serem modificados, e o diretório onde devem ser gerados os arquivos modificados (ver Exemplo 6.3).

```
java -classpath floggy-compiler-j2me-midp2-1.0.jar
net.sourceforge.floggy.Main -cp D:\Projetos\Hospital\classes -o
D:\Projetos\Hospital\classes-modificadas
```

Exemplo 6.3 - Comando para gerar o código de persistência através do enxertador

Após a execução deste comando, todas as classes do *classpath* (opção -cp) são analisadas e, se necessário, o código de persistência é enxertado. O resultado é gerado no diretório de saída (opção -o).

Finalizando com a pré-verificação das classes da aplicação e depois o empacotamento dos arquivos de classe e arquivos adicionais em um único arquivo de distribuição (.jar).

6.4 Resumo

Através do desenvolvimento da aplicação proposta neste estudo de caso foi possível testar e validar o *Floggy*, framework de persistência de objetos proposto por este trabalho de conclusão de curso.

Do ponto de vista do desenvolvedor, o uso do *Floggy* fez com que o tempo de desenvolvimento das operações de persistência diminuísse tendo em vista o atual modelo, no qual o desenvolvedor tem que implementar manualmente a serialização e deserialização dos objetos e os controles de persistência no repositório.

O uso da aplicação de forma mais intensiva revelou que o *Floggy* necessita de uma otimização no mecanismo de seleção, discutido na seção 7.2 sobre trabalhos futuros.

7 - Conclusão

Este trabalho de conclusão de curso chega a seu final e apresenta, como principal resultado, a construção de um framework de persistência de objetos, focado em aplicações para dispositivos móveis desenvolvidas com a tecnologia J2ME / MIDP, denominado *Floggy*.

O framework visa facilitar o trabalho do engenheiro de software e do desenvolvedor, permitindo que estes concentrem seus esforços na solução do problema que a aplicação objetiva tratar, abstraindo a forma como os dados são persistidos no repositório.

O *Floggy* é composto por dois módulos: *Framework* e *Compiler*. O módulo *Framework*, desenvolvido com a tecnologia J2ME / MIDP, é o responsável por fornecer às aplicações o suporte a persistência, recuperação e busca dos dados, enquanto o módulo *Compiler*, desenvolvido com a tecnologia J2SE, é responsável por gerar e enxertar o código de persistência na forma de *bytecode* nos arquivos de classe.

Na seção 7.1, são apresentados os resultados alcançados. Na seção 7.2, os trabalhos futuros. E na seção 7.3, as considerações finais.

7.1 Resultados Alcançados

Para atingir os objetivos deste trabalho, foram estudadas as características da tecnologia J2ME/MIDP, utilizada para desenvolvimento de aplicações que executam em dispositivos móveis, e as formas de persistência de dados disponíveis neste contexto, sendo a principal delas o RMS (*Record Management System*).

A partir deste levantamento constata-se a inexistência de um *framework* de persistência para a tecnologia J2ME/MIDP. A falta deste recurso acarreta ao engenheiro de software sobrecarga de trabalho, pois este ocupa

seu tempo com a forma como os dados serão persistidos, ao invés de focar seus esforços em assuntos inerentes à solução do problema.

Assim, foi feito um estudo sobre as camadas de persistência, com a finalidade de detalhar os requisitos necessários, bem como a forma de mapeamento e armazenamento dos dados. Neste estudo buscou-se também pesquisar as principais camadas de persistências existentes para plataformas tradicionais.

A pesquisa demonstra que nem todos os requisitos de uma camada de persistência para aplicações tradicionais devem ser contemplados, devido às restrições impostas pelas características dos dispositivos móveis. Desta forma, estes requisitos foram reavaliados e redefinidos em um subconjunto (ver seção 5.2) que atendesse às principais necessidades de uma camada de persistência para dispositivos móveis.

A Tabela 7.1 apresenta o conjunto de requisitos de uma camada de persistência para dispositivos móveis. Estes requisitos foram validados a partir da aplicação desenvolvida como estudo de caso.

Requisito	Validação
Encapsulamento completo da camada de dados	O gerenciador de persistência, representado pela classe *PersistableManager*, permite o encapsulamento completo da camada de dados. Esta classe providencia a persistência dos objetos através de métodos de manipulação de alto nível: **save(), load(), delete(), deleteAll(), find() e findAll()**.
Persistência seletiva	A palavra reservada **transient** pode ser utilizada para distinguir os campos que devem ser persistidos daqueles que não devem ser persistidos.
Identificadores de objetos	Cada objeto da aplicação, quando persistido, é associado a um identificador de persistência (OID). Este identificador é imutável e único dentre um conjunto de objetos de mesma classe.
Consulta a Objetos	A realização de pesquisas de objetos pode ser feita utilizando-se um objeto da classe *Filter*.
Extensibilidade	A adição de uma nova classe ao modelo não afetaria o funcionamento da aplicação e os dados já persistidos.
Cursores	Os resultados das seleções resultam em objetos da classe *ObjectSet*. Os objetos desta classe funcionam como cursores, pois permitem a recuperação dos objetos selecionados através da lista que mantém internamente de identificadores de persistência (OID).

Tabela 7.1 - Validação dos requisitos de uma camada de persistência para dispositivos móveis

7.2 Trabalhos Futuros

Uma das principais restrições existentes na atual implementação do *Floggy* é a impossibilidade de persistência de relacionamentos bidirecionais entre objetos. Conforme citado na seção 5.3.2.3, o engenheiro de software deve atentar a este detalhe pois o uso deste recurso pode fazer com que a aplicação entre em *loop*.

Atualmente, a seleção de objetos não está sendo realizada de forma otimizada, isto é, todos os atributos de todos os objetos são carregados para a memória para serem analisados de acordo com os critérios de seleção. Como os dispositivos móveis apresentam, geralmente, restrição de processamento, seria importante a criação de estratégias de seleção de objetos mais eficientes, como por exemplo, através da utilização de índices.

Os dispositivos móveis também apresentam restrições quanto ao armazenamento, o que poderia ser amenizado através do uso de um mecanismo de compactação dos dados.

Além das melhorias citadas, outras ainda poderiam ser implementadas como: chaves primárias, transações, garantia de integridade (verificação de chaves estrangeiras) e aprimoramento da remoção de objetos (cascata e *set null*).

7.3 Considerações Finais

Ao longo deste trabalho, adquirimos conhecimentos sobre desenvolvimento de aplicações para dispositivos móveis e sobre o benefício do uso de camadas de persistência.

Em relação a aplicações para dispositivos móveis, enfatizamos que as restrições impostas por este tipo de dispositivo (processamento, memória, armazenamento e conectividade) precisam ser levadas em consideração durante o desenvolvimento de uma aplicação.

Sobre camadas de persistência destacamos a importância da utilização das mesmas no processo de desenvolvimento de um software. A partir do seu uso, o engenheiro de software pode abstrair os detalhes de como a persistência dos dados será realizada, focando-se em questões inerentes à resolução do problema-alvo da aplicação.

O *Floggy* atingiu alguns dos objetivos propostos, se mostrou fácil de usar e e atendeu aos requisitos fundamentais de uma camada de persistência.

Temos como objetivo dar continuidade ao desenvolvimento do *Floggy*. Para isto, criamos um projeto no *SourceForge.net*[22] (http://floggy.sourceforge.net) a fim de distribuí-lo para a comunidade de desenvolvedores J2ME/MIDP. Posteriormente, as melhorias propostas como trabalhos futuros serão implementadas e liberadas em novas versões do *framework*.

[22] SourceForge.net é o maior web site de desenvolvimento e distribuição de software Open Source do mundo.

8 - Referências

[AMBLER (1)] AMBLER, Scott W. **The Design of a Robust Persistent Layer for Relational Databases.** Disponível em: < http://www.ambysoft.com/downloads/persistenceLayer.pdf http://www.AmbySoft.com/persistenceLayer.pdf>. Acesso em 20 de julho de 2005.

[AMBLER (2)] AMBLER, Scott W. **Mapping Objects to Relational Databases: O/R Mapping In Detail.** Disponível em: < http://www.agiledata.org/essays/mappingObjects.html >. Acesso em 20 de julho de 2005.

[CORBERA] CORBERA, Rodrigo Garcia. **Tutorial de programação J2ME.** Disponível em: < http://geocities.yahoo.com.br/brasilwireless/tutorial_j2me/j2me_01/j2me_01.html >. Acesso em 06 de junho de 2005.

[FREIRE 2003] FREIRE, Herval. **Mapeando Objetos para Banco de Dados Relacionais: técnicas e implementações**. Disponível em: < www.mundooo.com.br >. Acesso em 15 de junho de 2005.

[GUPTA] GUPTA, Rahul Kumar.**Intelligent appliances and J2ME RMS**. Disponível em: < http://java.ittoolbox.com >. Acesso 23 de maio de 2005.

[HIBERNATE] Website **HIBERNATE**. Disponível em: < http://www.hibernate.org >. Acesso em 16 de outubro de 2005.

[JDO] Website **JDO**. Disponível em: < http://java.sun.com/products/jdo/overview.html >. Acesso em 16 de outubro de 2005.

[MAGALHÃES 2005] MAGALHÃES, Katy Cristina Paulino. **Framepersist: Um framework de persistência de objetos para o desenvolvimento de aplicações para dispositivos móveis.** Dissertação da Universidade Federal do Ceará.

[MUCHOW 2001] MUCHOW, John W. **Core J2ME.** Sun Microsystems Press e Prentice Hall. 736p.

[NOKIA] Website **NOKIA**. Disponível em: < http://www.nokia.com/nokia/0,8764,73210,00.html. Acesso em 05 de junho de 2005.

[PALUDO 2003] PALUDO, Lauriana. **Um estudo sobre as tecnologias java de desenvolvimento de aplicações móveis.** Trabalho de Conclusão de Curso da Universidade Federal de Santa Catarina.

[SUN (1)] SUN Brasil. **J2ME APIs: Which APIs come from the J2SE Platform?.** Disponível em < http://br.sun.com/ >. Acesso em 06 de junho 2005.

[SUN (2)] SUN Brasil. **J2ME Configurations.** Disponível em < http://br.sun.com/ >. Acesso em 06 de junho 2005.

[SUN (3)] SUN Brasil. **A Survey of J2ME Today.** Disponível em < http://br.sun.com/ >. Acesso em 06 de junho 2005.

[TOPLEY 2002] TOPLEY, Kim. **J2ME is a Nutshell.** O´Reilly & Associates. 478p.

[WARPECHOWSKI], WARPECHOWSKI José Carlos e CARVALHO, Moacyr Flêvio de Lima. **Análise Comparativa entre Java Data Objects e Entity Bean (CMP)**

9 - Anexos

9.1 Artigo

Floggy:
Framework de persistência para J2ME

Priscila Tavares Lugon
Thiago Rossato

Bacharelado em Sistemas de Informação, 2005
Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC), Brasil, 88040-900
{priscila}{rossato}@inf.ufsc.br

Resumo

Um dos pontos críticos no desenvolvimento de aplicações MIDP - para dispositivos móveis - é a codificação do armazenamento de dados. Surge então, o Floggy, framework de persistência de objetos que visa fornecer um mecanismo de persistência eficiente para aplicações MIDP. Este framework *é composto por classe, interfaces e métodos com operações básicas de inclusão, alteração, exclusão e busca de objetos levando em consideração as limitações que os dispositivos móveis apresentam.*

Palavras-chave: J2ME, MIDP, RMS, Floggy, Persistência

Abstract

One of the critical points in the development of MIDP applications is the codification of the data storage. The Floggy, a framework for object persistence, aims to supply an efficient mechanism of persistence for MIDP applications. This framework is composed by classes, interfaces and methods with basic operations such inclusion, alteration, exclusion and search of objects.

Keywords: J2ME, MIDP, RMS, Floggy, Persistence

INTRODUÇÃO

Celulares, computadores de mão e organizadores pessoais estão se tornando peças fundamentais no dia-a-dia de pessoas e organizações. Estes dispositivos estão em constante evolução. Diariamente surgem

notícias sobre processadores menores, mais velozes e que consomem menos bateria. Os meios de armazenamentos seguem a mesma linha, cada vez menores e capazes de armazenar mais informações.

O maior poder computacional faz com que tais dispositivos suportem aplicações mais complexas e que necessitem armazenar grandes quantidades de informações. Os dispositivos mais modernos podem rodar aplicações até então existente somente em computadores.

Um dos pontos críticos de uma aplicação para dispositivos móveis é a codificação para o armazenamento e a manipulação dos dados. As principais limitações vão desde as APIs disponibilizadas pelos sistemas operacionais aos meios de armazenamento destes dispositivos, que apresentam capacidade de armazenamento e velocidade de acesso aos dados restrita.

A grande maioria das plataformas de desenvolvimento para dispositivos móveis não fornece suporte a conceitos como tabelas, campos e índices. Há poucas opções de bancos de dados e, muitas vezes, as soluções disponíveis são dependentes de determinado dispositivo e/ou sistema operacional.

J2ME

A plataforma J2ME é uma das plataformas de desenvolvimento para dispositivos móveis mais utilizadas atualmente. Esta tecnologia é gratuita e traz consigo a portabilidade da linguagem Java.

As principais vantagens do ambiente de execução J2ME são: uso eficiente do processador, otimização dos recursos de memória e baixo consumo de energia. Há ainda os benefícios herdados da tecnologia Java, que oferece um modelo robusto de segurança, suporte a aplicações em rede e portabilidade das aplicações.

Os dispositivos móveis apresentam muitas características em comum. Em virtude disto, a arquitetura J2ME define uma estrutura multicamada (Ilustração 1), composta de configurações, perfis e pacotes opcionais. Essas

camadas são elementos fundamentais da plataforma J2ME e, quando combinadas, transformam-se em ambientes de execução para a quase totalidade dos dispositivos móveis.

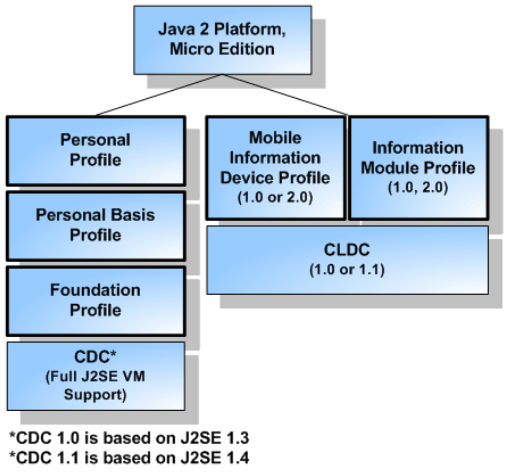

Ilustração 1 - Configurações e perfis J2ME

O perfil MIDP (*Mobile Information Device Profile*) é o mais difundido entre celulares e computadores de mão. Consequentemente, a maioria das aplicações é desenvolvida para este perfil.

RMS

O perfil MIDP fornece um mecanismo de armazenamento de dados chamado RMS (*Record Management System*; em português, Sistema de Gerenciamento de Registros).

O RMS é composto por *record stores*. Um *record store,* por sua vez, é formado por um conjunto de registros. Os registros são estruturas simples, formados por um identificador único (ID) – valor inteiro e incremental que desempenha papel de chave – e um conjunto de bytes (Figura 5).

ID	DATA
1	byte[]
2	byte[]
3	byte[]
...	...

Figura 5 - Record Store

Cabe à aplicação gerenciar o código de persistência dos dados, transformando um objeto em uma seqüência de bytes e vice-versa. Em aplicações pequenas, com poucos dados a serem persistidos, o RMS se mostra como uma alternativa ideal. Já em aplicações de maior porte, que apresentam uma grande quantidade de dados a serem persistidos, a codificação torna-se custosa.

CAMADAS DE PERSISTÊNCIA

Uma solução para o problema apresentado é a introdução de uma camada de persistência, responsável por encapsular a lógica necessária para que um objeto ou uma coleção de objetos sejam salvos ou recuperados de um meio de armazenamento.

Conceitualmente, uma Camada de Persistência de Objetos é uma biblioteca que permite a realização do processo de persistência de forma transparente. A utilização deste conceito permite ao desenvolvedor trabalhar como se estivesse em um sistema completamente orientado a objetos, utilizando métodos de alto nível para incluir, alterar e remover objetos e uma linguagem para realizar consultas que retornam coleções de objetos instanciados.

Esta camada facilitaria muito o desenvolvimento de aplicações para dispositivos móveis que se utilizam dos recursos de persistência de dados.

FLOGGY

O Floggy é um framework de persistência de objetos para dispositivos móveis, que visa abstrair o mapeamento necessário para transformação de objetos em seqüências de bytes (e vice-versa) através de comandos de alto nível (incluir, alterar ou excluir um objeto).

Os comandos devem ser utilizados para manipular um objeto para que este seja armazenado e recuperado a partir de um meio persistente.

O *Floggy* é composto por dois módulos: *Framework* e *Compiler*. O módulo *Framework* é responsável por fornecer às aplicações o suporte à persistência, recuperação e busca dos dados, enquanto o módulo *Compiler* é responsável por gerar e enxertar o código de persistência aos objetos.

Este *framework* contempla os principais requisitos para uma camada de persistência, que são:

- ***Encapsulamento completo da camada de dados:*** O desenvolvedor utiliza-se apenas comandos de alto nível, como “salvar” e “excluir", para tratar a persistência de objetos,

deixando a camada de persistência realizar a conversão destas mensagens em ações sobre os dados.

- ***Persistência Seletiva*:** permitir a persistência apenas dos atributos desejados de um objeto, minimizando o espaço armazenado com atributos transientes.

- ***Identificadores de Objetos*:** Garantia de que a aplicação trabalhará com objetos de identidade única.

- ***Consulta a Objetos:*** Fornecer um mecanismo de consulta eficiente, que permita o acesso direto aos dados. Assim, são permitidas consultas com um grau de complexidade maior que o usual.

- ***Cursores*:** Fornecer mecanismo otimizado para recuperação de objetos a fim de evitar que muitos objetos sejam carregados de uma única vez em memória. Assim, uma lista de objetos obtida através de uma consulta, armazenará somente as referências aos objetos, permitindo que os mesmos sejam recuperados posteriormente.

Conclusão

O uso de *frameworks* de persistência é comum em aplicações desenvolvidas para computadores tradicionais. O desenvolvimento de aplicações MIDP pode ser beneficiado através do uso destas ferramentas.

O *Floggy* surge como opção a um ambiente que até então não contava com *frameworks* de persistência.

Os desenvolvedores de aplicações MIDP podem usufruir das facilidades oferecidas por este *framework* de persistência, e focar seus esforços na solução do problema principal a ser tratado pela aplicação, delegando toda a persistência dos dados ao *framework*.

9.2 Código Fonte Floggy Compiler

ClassVerifier

```
package net.sourceforge.floggy;

import java.util.Date;
import java.util.Vector;

import javassist.ClassPool;
import javassist.CtClass;
import javassist.CtField;
import javassist.Modifier;
import javassist.NotFoundException;

public class ClassVerifier {

    CtClass ctClass;

    public ClassVerifier(CtClass ctClass) {
        this.ctClass = ctClass;
    }

    /**
     * Check if the class implements the persistable interface
     * (net.sourceforge.floggy.Persistable).
     *
     * @return True if the class implements such interface, false
otherwise.
     * @throws NotFoundException
     */
    public boolean isPersistable() throws NotFoundException {
        // Checks if superclass is persistable.
        CtClass superClass = ctClass.getSuperclass();
        if (superClass != null) {
            ClassVerifier verifier = new ClassVerifier(superClass);
            if (verifier.isPersistable()) {
                return true;
            }
        }

        CtClass[] interfaces = null;
        try {
            interfaces = ctClass.getInterfaces();
        } catch (NotFoundException e) {
            e.printStackTrace();
            // Ignore
        }

        if (interfaces != null) {
            for (int i = 0; i < interfaces.length; i++) {
                if (interfaces[i].getName().equals(
                        "net.sourceforge.floggy.Persistable")) {
                    return true;
                }
            }
```

```
        }

        return false;
    }

    public boolean isModified() throws NotFoundException {
        // Checks if the class implements
net.sourcefloggy.__Persistable
        // interface.
        CtClass[] interfaces = null;
        try {
            interfaces = ctClass.getInterfaces();
        } catch (NotFoundException e) {
            e.printStackTrace();
            // Ignore
        }

        if (interfaces != null) {
            for (int i = 0; i < interfaces.length; i++) {
                if (interfaces[i].getName().equals(
                        "net.sourceforge.floggy.__Persistable")) {
                    return true;
                }
            }
        }

        return false;

    }

    /**
     * Check if all attributes are persistable.
     *
     * @throws CompilerException
     */
    public void checkDependencies() throws CompilerException {
        CtField[] fields = this.ctClass.getDeclaredFields();
        ClassPool classPool = this.ctClass.getClassPool();

        try {
            if (fields != null) {
                for (int i = 0; i < fields.length; i++) {
                    CtClass type = fields[i].getType();

                    // Primitive types.
                    if (type.isPrimitive()) {
                        continue;
                    }

                    // String, Boolean, Integer, Long, Double, Float,
Date,
                    // Vector
                    String name = type.getName();
                    if (name.equals(String.class.getName())
                            || name.equals(Boolean.class.getName())
                            || name.equals(Integer.class.getName())
                            || name.equals(Long.class.getName())
                            || name.equals(Double.class.getName())
                            || name.equals(Float.class.getName())
```

```
                            || name.equals(Date.class.getName())
                            || name.equals(Vector.class.getName())) {
                        continue;
                    }

                    // Transient attributes
                    int modifiers = fields[i].getModifiers();
                    if ((Modifier.isTransient(modifiers))) {
                        continue;
                    }

                    // Persistable arrays
                    if (type.isArray()) {
                        ClassVerifier verifier = new
ClassVerifier(type
                                .getComponentType());
                        // TODO Voltar
                        // if (verifier.isPersistable()) {
                        continue;
                        // }
                    }

                    // Persistable attributes
                    CtClass attributeClass =
classPool.get(type.getName());
                    ClassVerifier verifier = new
ClassVerifier(attributeClass);
                    if (verifier.isPersistable()) {
                        continue;
                    }

                    throw new CompilerException("Attribute \""
                            + fields[i].getName() + "\" does not
implements "
                            + Persistable.class.getName());
                }
            }
        } catch (NotFoundException e) {
            throw new CompilerException(e.getMessage());
        }
    }
}
```

Compiler

```
package net.sourceforge.floggy;

import java.io.File;
import java.io.IOException;

import javassist.CannotCompileException;
import javassist.ClassPool;
import javassist.CtClass;
import javassist.NotFoundException;
import net.sourceforge.floggy.codegen.CodeGenerator;
import net.sourceforge.floggy.pool.InputPool;
import net.sourceforge.floggy.pool.OutputPool;
import net.sourceforge.floggy.pool.PoolFactory;
```

```
/**
 * Main compiler class!
*/
public class Compiler {

    private ClassPool classpathPool;

    private InputPool inputPool;

    private OutputPool outputPool;

    /**
     * Creates a new instance
     *
     * @param args
     */
    protected Compiler() {
        this.classpathPool = ClassPool.getDefault();
    }

    /**
     * Sets the classpath.
     *
     */
    protected void setClasspath(String[] classpath) {
        if (classpath != null && classpath.length > 0) {
            for (int i = classpath.length - 1; i >= 0; i--) {
                try {
                    this.classpathPool.insertClassPath(classpath[i]);
                } catch (NotFoundException e) {
                    // Ignore
                }
            }
        }
    }

    /**
     * Sets the input file.
     *
     */
    protected void setInputFile(File inputFile) throws
CompilerException {
        this.inputPool = PoolFactory.createInputPool(inputFile);

        try {

this.classpathPool.insertClassPath(inputFile.getCanonicalPath());
        } catch (NotFoundException e) {
            // Ignore
        } catch (IOException e) {
            // Ignore
        }
    }

    /**
     * Sets the output file.
     *
     * @param outputFile
```

```
     */
    protected void setOutputFile(File outputFile) throws
CompilerException {
        this.outputPool = PoolFactory.createOutputPool(outputFile);
    }

    protected void execute() throws CompilerException {
        long time = System.currentTimeMillis();
        int modifiedCount = 0;
        int classCount = this.inputPool.getFileCount();

        try {
            for (int i = 0; i < classCount; i++) {
                String fileName = this.inputPool.getFileName(i);
                String className = getClassName(fileName);

                // Adds non-class files to output pool
                if (className == null) {
                    this.outputPool.addFile(fileName);
                    continue;
                }

                CtClass ctClass = this.classpathPool.get(className);

                System.out.print(className + " ... ");
                ClassVerifier verifier = new ClassVerifier(ctClass);
                if (verifier.isPersistable() &&
!verifier.isModified()) {
                    verifier.checkDependencies();

                    CodeGenerator codeGenerator = new
CodeGenerator(ctClass);
                    codeGenerator.generateCode();

                    System.out.println("MODIFIED");
                    modifiedCount++;

                    // Adds modified class to output pool
                    this.outputPool.addClass(ctClass);
                } else {
                    System.out.println("PASS");

                    // Adds non-persistable class to output pool
                    this.outputPool.addFile(fileName);
                }

            }
        } catch (NotFoundException e) {
            throw new CompilerException(e.getMessage());
        } catch (CannotCompileException e) {
            throw new CompilerException(e.getMessage());
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Status
        System.out.println();
        System.out.println("Classes modified: " + modifiedCount + " of
"
```

```
                + classCount);
        time = System.currentTimeMillis() - time;
        System.out.println("Time elapsed: " + time + "ms");
    }

    /**
     * Returns the class name given a file name.
     *
     * @param fileName
     * @return
     */
    private static String getClassName(String fileName) {
        if (fileName.endsWith(".class")) {
            String className = fileName.replace(File.separatorChar,
'.');
            return className.substring(0, className.length() - 6);
        }

        // File name does not represents a class file.
        return null;
    }
}
```

CompilerException

```
package net.sourceforge.floggy;

public class CompilerException extends Exception {

    public CompilerException(String message) {
        super(message);
    }

}
```

Compiler

```
package net.sourceforge.floggy;

import java.io.File;

import org.apache.tools.ant.BuildException;
import org.apache.tools.ant.Task;

public class CompilerTask extends Task {

    String classpath;

    String inputFile;

    String outputFile;

    String frameworkClasspath;

    public void execute() throws BuildException {
        Compiler compiler = new Compiler();

        try {
```

```
compiler.setClasspath(classpath.split(File.pathSeparator));
            compiler.setInputFile(new File(inputFile));
            compiler.setOutputFile(new File(outputFile));
            compiler.execute();
        } catch (CompilerException e) {
            e.printStackTrace();
            throw new BuildException(e);
        }
    }

    public void setClasspath(String classpath) {
        this.classpath = classpath;
    }

    public void setInputFile(String inputFile) {
        this.inputFile = inputFile;
    }

    public void setOutputFile(String outputFile) {
        this.outputFile = outputFile;
    }

    public void setFrameworkClasspath(String frameworkClasspath) {
        this.frameworkClasspath = frameworkClasspath;
    }

}
```

Main

```
package net.sourceforge.floggy;

import java.io.File;
import java.util.Arrays;
import java.util.Vector;

public class Main {

    /**
     * @param args
     */
    public static void main(String[] args) {
        Vector params = new Vector(Arrays.asList(args));

        String[] classpath = initClasspath(params);
        File outputFile = initOutputFile(params);
        File inputFile = initInputFile(params);

        // If no output file is defined, sets the output file the same
as the
        // input file
        if (outputFile == null) {
            outputFile = inputFile;
        }

        Compiler compiler = new Compiler();
```

```
        try {
            compiler.setClasspath(classpath);
            compiler.setOutputFile(outputFile);
            compiler.setInputFile(inputFile);
            compiler.execute();
        } catch (CompilerException e) {
            System.out.println(e.getMessage());
        }
    }

    /**
     * Get the classpath option.
     *
     */
    private static String[] initClasspath(Vector params) {
        String[] classpath = null;

        int index = params.indexOf("-cp");
        if (index != -1) {
            try {
                String value = params.get(index + 1).toString();
                classpath = value.split(File.pathSeparator);

                // Remove classpath options
                params.remove("-cp");
                params.remove(value);
            } catch (ArrayIndexOutOfBoundsException e) {
                System.out.println("-cp requires class path
specification");
                usage();
            }
        }

        return classpath;
    }

    /**
     * Get the output file.
     *
     * @param args
     * @return
     */
    private static File initOutputFile(Vector params) {
        File outputFile = null;

        int index = params.indexOf("-o");
        if (index != -1) {
            String value = params.get(index + 1).toString();

            outputFile = new File(value.trim());

            params.remove(index);
            params.remove(value);

        }

        return outputFile;
    }
```

```
    /**
     * Get the input file option.
     *
     */
    private static File initInputFile(Vector params) {
        File inputFile = null;

        if (params.size() == 1) {
            String value = params.get(0).toString();

            inputFile = new File(value.trim());
            if (inputFile.exists()) {
                params.remove(0);
            } else {
                System.out.println("File \"" + value + "\" does not
exists.");
                usage();
            }
        } else {
            System.out.println("Invalid number of parameters.");
            usage();
        }

        return inputFile;
    }

    /**
     * Prints usage message.
     */
    private static final void usage() {
        System.out.println("Usage:");
        System.out.println("java " + Compiler.class.getName()
                + " [-options] jarfile");
        System.out.println("java " + Compiler.class.getName()
                + " [-options] zipfile");
        System.out.println("java " + Compiler.class.getName()
                + " [-options] classfile");
        System.out.println();
        System.out.println("Where options are:");
        System.out
                .println("-cp\t<Class search path of directories and
zip/jar files separeted by "
                        + File.pathSeparatorChar + ">");
        System.out
                .println("-o\t<Output file or directory where files
will be stored>");

        System.exit(1);
    }

}
```

ZipInputPool

```
package net.sourceforge.floggy.pool;

import java.io.File;
import java.io.IOException;
```

```
import java.util.Vector;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;

import net.sourceforge.floggy.CompilerException;

public class ZipInputPool implements InputPool {

    Vector files;

    public ZipInputPool(ZipInputStream in) throws CompilerException {
        this.files = new Vector();
        this.initFiles(in);
    }

    public int getFileCount() {
        return this.files.size();
    }

    public String getFileName(int index) {
        return this.files.get(index).toString();
    }

    private void initFiles(ZipInputStream in) throws CompilerException
{
        try {
            for (ZipEntry entry = in.getNextEntry(); entry != null;
entry = in
                    .getNextEntry()) {
                String name = entry.getName();
                name = name.replace('/',
File.separatorChar).replace('\\',
                        File.separatorChar);
                this.files.add(name);
            }
        } catch (IOException e) {
            throw new CompilerException("Error reading file.");
        }
    }


}
```

DirectoryInputPool

```
package net.sourceforge.floggy.pool;

import java.io.File;
import java.io.IOException;
import java.util.Vector;

public class DirectoryInputPool implements InputPool {

    File rootDirectory;

    Vector files;

    public DirectoryInputPool(File directory) throws IOException {
```

```
        this.rootDirectory = directory;
        this.files = new Vector();
        this.initFiles(directory);
    }

    public int getFileCount() {
        return this.files.size();
    }

    public String getFileName(int index) {
        return this.files.get(index).toString();
    }

    private void initFiles(File directory) throws IOException {
        File[] files = directory.listFiles();

        if (files != null) {
            String rootPath = this.rootDirectory.getCanonicalPath();

            for (int i = 0; i < files.length; i++) {
                if (files[i].isDirectory()) {
                    this.initFiles(files[i]);
                } else if (files[i].isFile()) {
                    String filePath = files[i].getCanonicalPath();

                    String className =
filePath.substring(rootPath.length());
                    if (className.startsWith(File.separator)) {
                        className = className.substring(1);
                    }

                    this.files.add(className);
                }
            }
        }
    }
}
```

DirectoryOutputPool

```
package net.sourceforge.floggy.pool;

import java.io.File;

public class DirectoryOutputPool extends OutputPool {

    public DirectoryOutputPool(File directory) {
        super(directory);
    }

    public void writeOutput() {
    }


}
```

InputPool

```
package net.sourceforge.floggy.pool;

public interface InputPool {

    public int getFileCount();

    public String getFileName(int index);


}
```

ZipInputPool

```
package net.sourceforge.floggy.pool;

import java.util.jar.JarInputStream;

import net.sourceforge.floggy.CompilerException;

public class JarInputPool extends ZipInputPool {

    public JarInputPool(JarInputStream in) throws CompilerException {
        super(in);
    }


}
```

OutputPool

```
package net.sourceforge.floggy.pool;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Vector;

import javassist.CannotCompileException;
import javassist.CtClass;
import javassist.NotFoundException;

public abstract class OutputPool {

    protected Vector files;

    private File tempDir;

    protected File outputFile;

    public OutputPool(File file) {
        this.outputFile = file.getAbsoluteFile();

        if(!outputFile.exists()) {
            outputFile.mkdirs();
        }
        this.initTempDir();
    }
```

```
    public void addClass(CtClass ctClass) throws NotFoundException,
            IOException, CannotCompileException {
        File classFile = this.getOutputFile(ctClass);
        classFile.getParentFile().mkdirs();

        if(classFile.exists()) {
            classFile.delete();
        }

        FileOutputStream out = new FileOutputStream(classFile);
        out.write(ctClass.toBytecode());
        out.close();


        this.addFile(classFile.getPath());
    }

    public void addFile(String fileName) {
    }

    /**
     * Returns the temp directory to be used for
     *
     * @return
     */
    private void initTempDir() {
        // TODO Generate temporary random directory name
        this.tempDir = new File("temp");
        this.tempDir.deleteOnExit();
    }

    private File getOutputFile(CtClass ctClass) {
        File outputFile = new File(this.outputFile.getPath()
                + File.separatorChar
                + ctClass.getName().replace('.', File.separatorChar) +
".class");
        return outputFile;
    }

    public abstract void writeOutput();


}
```

PoolFactoy

```
package net.sourceforge.floggy.pool;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.jar.JarInputStream;
import java.util.zip.ZipInputStream;

import net.sourceforge.floggy.CompilerException;

public class PoolFactory {
```

```
    public static InputPool createInputPool(File file) throws
CompilerException {
        if (!file.exists()) {
            throw new CompilerException(file.getName()
                    + " does not exists. (Full path: " +
file.getPath() + ")");
        }

        if (file.isFile() && file.getName().endsWith(".jar")) {
            try {
                return new JarInputPool(new JarInputStream(new
FileInputStream(
                        file)));
            } catch (IOException e) {
                throw new CompilerException("Invalid input file
type.");
            }
        } else if (file.isFile() && file.getName().endsWith(".zip")) {
            try {
                return new ZipInputPool(new ZipInputStream(new
FileInputStream(
                        file)));
            } catch (IOException e) {
                throw new CompilerException("Invalid input file
type.");
            }
        } else if (file.isDirectory()) {
            try {
                return new DirectoryInputPool(file);
            } catch (IOException e) {
                throw new CompilerException(e.getMessage());
            }
        }

        throw new CompilerException(
                file.getName()
                        + " is a invalid file type. Valid types are:
.jar, .zip and directories.");
    }

    public static OutputPool createOutputPool(File file)
            throws CompilerException {
        if (file.isFile() && file.getName().endsWith(".jar")) {
            // try {
            // return new JarInputPool(new JarInputStream(new
FileInputStream(
            // file)));
            // } catch (IOException e) {
            // throw new CompilerException("Invalid input file
type.");
            // }
            // } else if (file.isFile() &&
file.getName().endsWith(".zip")) {
            // try {
            // return new ZipInputPool(new ZipInputStream(new
FileInputStream(
            // file)));
            // } catch (IOException e) {
```

```
            // throw new CompilerException("Invalid input file
type.");
            // }
        } else {
            return new DirectoryOutputPool(file);
        }

        throw new CompilerException(
                file.getName()
                        + " is a invalid file type for output file.
Valid types are: .jar, .zip and directories.");
    }

}
```

CodeFormatter

```
package net.sourceforge.floggy.formatter;

import java.util.StringTokenizer;

public class CodeFormatter {

    public static String format(String source) {
        StringBuffer formatted = new StringBuffer();
        StringTokenizer tokenizer = new StringTokenizer(source, "\n");

        int tabIdent = 0;
        while (tokenizer.hasMoreTokens()) {
            String line = tokenizer.nextToken();

            int lineTabIdent = getTabIdent(line);
            if (lineTabIdent <= 0) {
                tabIdent += lineTabIdent;
            }

            for (int i = 0; i < tabIdent; i++) {
                line = "    " + line;
            }

            formatted.append(line + "\n");

            if(lineTabIdent > 0) {
                tabIdent += lineTabIdent;
            }
        }

        return formatted.toString();
    }

    private static int getTabIdent(String line) {
        if (line == null) {
            return 0;
        }

        int tabIdent = 0;
        char[] charArray = line.toCharArray();
        for (int i = 0; i < charArray.length; i++) {
```

```
            if (charArray[i] == '{') {
                tabIdent++;
            } else if (charArray[i] == '}') {
                tabIdent--;
            }
        }

        return tabIdent;
    }

}
```

WrapperGenerator

```
package net.sourceforge.floggy.codegen;

import javassist.CtClass;
import javassist.NotFoundException;

public class WrapperGenerator extends SourceCodeGenerator {

    public WrapperGenerator(String fieldName, CtClass classType) {
        super(fieldName, classType);
    }

    public void initLoadCode() throws NotFoundException {
        addLoadCode("if(dis.readByte() == 0) {");
        addLoadCode(fieldName + " = new " + classType.getName() +
"(dis.read"
                + getType() + "());");
        addLoadCode("}");
    }

    public void initSaveCode() throws NotFoundException {
        String type = getType();

        addSaveCode("if(" + fieldName + " == null) {");
        addSaveCode("dos.writeByte(1);");
        addSaveCode("}");
        addSaveCode("else {");
        addSaveCode("dos.writeByte(0);");
        addSaveCode("dos.write" + type + "(" + fieldName + "."
                + type.toLowerCase() + "Value());");
        addSaveCode("}");
    }

    public boolean isInstanceOf() throws NotFoundException {
        String name = classType.getName();
        return name.equals(Boolean.class.getName())
                || name.equals(Byte.class.getName())
                || name.equals(Character.class.getName())
                || name.equals(Double.class.getName())
                || name.equals(Float.class.getName())
                || name.equals(Integer.class.getName())
                || name.equals(Long.class.getName())
                || name.equals(Short.class.getName());
    }
```

```
    private String getType() throws NotFoundException {
        String name = classType.getName();

        if (name.equals(Boolean.class.getName())) {
            return "Boolean";
        }
        if (name.equals(Byte.class.getName())) {
            return "Byte";
        }
        if (name.equals(Character.class.getName())) {
            return "Char";
        }
        if (name.equals(Double.class.getName())) {
            return "Double";
        }
        if (name.equals(Float.class.getName())) {
            return "Float";
        }
        if (name.equals(Integer.class.getName())) {
            return "Int";
        }
        if (name.equals(Long.class.getName())) {
            return "Long";
        }
        if (name.equals(Short.class.getName())) {
            return "Short";
        }

        throw new NotFoundException("Type not found!");
    }

}
```

ArrayGenerator

```
package net.sourceforge.floggy.codegen;

import javassist.CtClass;
import javassist.NotFoundException;

public class ArrayGenerator extends SourceCodeGenerator {

    public ArrayGenerator(String fieldName, CtClass classType) {
        super(fieldName, classType);
    }

    public void initLoadCode() throws NotFoundException {
        SourceCodeGenerator generator;

        addLoadCode("if(dis.readByte() == 0) {");
        addLoadCode("int count = dis.readInt();");
        addLoadCode("for(int i = 0; i < count; i++) {");
        generator =
SourceCodeGeneratorFactory.getSourceCodeGenerator(fieldName
                + "[i]", classType.getComponentType());
        addLoadCode(generator.getLoadCode());
        addLoadCode("}");
```

```
        addLoadCode("}");
        addLoadCode("else {");
        addLoadCode(fieldName + " = null;");
        addLoadCode("}");
    }

    public void initSaveCode() throws NotFoundException {
        SourceCodeGenerator generator;

        addSaveCode("java.lang.Object array =  " + fieldName + ";");
        addSaveCode("if(array == null) {");
        addSaveCode("dos.writeByte(1);");
        addSaveCode("}");
        addSaveCode("else {");
        addSaveCode("dos.writeByte(0);");
        addSaveCode("int count = " + fieldName +".length;");
        addSaveCode("dos.writeInt(count);");
        addSaveCode("for(int i = 0; i < count; i++) {");
        generator =
SourceCodeGeneratorFactory.getSourceCodeGenerator(fieldName
                + "[i]", classType.getComponentType());
        addSaveCode(generator.getSaveCode());
        addSaveCode("}");
        addSaveCode("}");
    }

    public boolean isInstanceOf() {
        return classType.isArray();
    }
}
```

CodeGenerator

```
package net.sourceforge.floggy.codegen;

import javassist.CannotCompileException;
import javassist.CtClass;
import javassist.CtField;
import javassist.CtMethod;
import javassist.CtNewMethod;
import javassist.NotFoundException;
import net.sourceforge.floggy.ClassVerifier;
import net.sourceforge.floggy.formatter.CodeFormatter;

/**
 * Class CodeGenerator
 *
 * @author Thiago Rossato
 */
public class CodeGenerator {

    private static final boolean DEBUG = true;

    /**
     * Class to be modified;
     */
    CtClass ctClass;
```

```
    /**
     * Source code of function <code>public int __getId();</code>.
     */
    StringBuffer funcitonGetId;

    /**
     * Source code of function <code>public void __load(byte[]
buffer);</code>.
     */
    StringBuffer functionLoad_Buffer;

    /**
     * Source code of function <code>public void __load(int
id);</code>.
     */
    StringBuffer functionLoad_Id;

    /**
     * Source code of function <code>public int __save();</code>
     */
    StringBuffer functionSave;

    /**
     * Creates a new code generator for the class.
     *
     * @param ctClass
     *            Class to be modified.
     */
    public CodeGenerator(CtClass ctClass) {
        this.ctClass = ctClass;
        this.funcitonGetId = new StringBuffer();
        this.functionLoad_Buffer = new StringBuffer();
        this.functionLoad_Id = new StringBuffer();
        this.functionSave = new StringBuffer();
    }

    /**
     * Generate all the necessary source code for this class.
     *
     * @throws NotFoundException
     * @throws CannotCompileException
     */
    public void generateCode() throws NotFoundException,
CannotCompileException {
        // Implements interface
        this.generateInterfacePersistable();

        // Attributes
        this.generateFieldId();

        // Methods
        this.generateFunctionGetId();
        this.generateFunctionLoad_Buffer();
        this.generateFunctionLoad_Id();
        this.generateFunctionSave();
    }

    /**
     * @throws NotFoundException
```

```
     */
    private void generateInterfacePersistable() throws
NotFoundException {
        this.ctClass.addInterface(this.ctClass.getClassPool().get(
                "net.sourceforge.floggy.__Persistable"));
    }

    /**
     *
     * @throws CannotCompileException
     */
    private void generateFieldId() throws CannotCompileException {
        CtField id = CtField.make("int __id = -1;", ctClass);
        this.ctClass.addField(id);
    }

    /**
     * @throws CannotCompileException
     */
    private void generateFunctionGetId() throws CannotCompileException
{
        this.funcitonGetId.append("public int __getId() {\n");
        this.funcitonGetId.append("return this.__id;\n");
        this.funcitonGetId.append("}");

        debug("");
        debug(CodeFormatter.format(this.funcitonGetId.toString()));

        CtMethod getIdMethod =
CtNewMethod.make(this.funcitonGetId.toString(),
                this.ctClass);
        this.ctClass.addMethod(getIdMethod);
    }

    /**
     *
     * @throws CannotCompileException
     * @throws NotFoundException
     */
    private void generateFunctionLoad_Buffer() throws
CannotCompileException,
            NotFoundException {
        // Header
        addLoad_Buffer("public void __load(byte[] buffer) throws
java.lang.Exception {\n");

        // Streams
        addLoad_Buffer("java.io.ByteArrayInputStream bais = new
java.io.ByteArrayInputStream(buffer);\n");
        addLoad_Buffer("java.io.DataInputStream dis = new
java.io.DataInputStream(bais);\n");

        // Save the superclass if it is persistable.
        CtClass superClass = this.ctClass.getSuperclass();
        ClassVerifier verifier = new ClassVerifier(superClass);
        if (verifier.isPersistable()) {

addLoad_Buffer(SuperClassGenerator.generateLoadSource(superClass));
        }
```

```
        CtField[] fields = ctClass.getDeclaredFields();
        if (fields != null && fields.length > 0) {
            SourceCodeGenerator generator;
            CtField field;
            for (int i = 0; i < fields.length; i++) {
                field = fields[i];

                // Ignores compiler fields.
                if (field.getName().equals("__id")) {
                    continue;
                }

                generator =
SourceCodeGeneratorFactory.getSourceCodeGenerator(
                        field.getName(), field.getType());
                if (generator != null) {
                    this.addLoad_Buffer(generator.getLoadCode());
                }
            }
        }

        // Close the streams
        addLoad_Buffer("dis.close();\n");

        addLoad_Buffer("return;\n");
        addLoad_Buffer("}\n");

        debug("");
        debug(CodeFormatter.format(functionLoad_Buffer.toString()));


this.ctClass.addMethod(CtNewMethod.make(this.functionLoad_Buffer
                .toString(), this.ctClass));
    }

    /**
     *
     * @throws CannotCompileException
     */
    private void generateFunctionLoad_Id() throws
CannotCompileException {
        this.functionLoad_Id
                .append("public void __load(int id) throws
java.lang.Exception {\n");

        // RecordStore
        this.functionLoad_Id
                .append("javax.microedition.rms.RecordStore rs =
net.sourceforge.floggy.PersistableManager.getRecordStore(\""
                        + this.ctClass.getName() + "\");\n");
        this.functionLoad_Id.append("byte[] buffer =
rs.getRecord(id);\n");
        this.functionLoad_Id.append("rs.closeRecordStore();\n");

        // Load
        this.functionLoad_Id.append("this.__load(buffer);\n");
        this.functionLoad_Id.append("this.__id = id;\n");
        this.functionLoad_Id.append("}");
```

```
        debug("");
        debug(CodeFormatter.format(functionLoad_Id.toString()));

        this.ctClass.addMethod(CtNewMethod.make(
                this.functionLoad_Id.toString(), this.ctClass));
    }


    /**
     *
     * @throws CannotCompileException
     * @throws NotFoundException
     */
    private void generateFunctionSave() throws CannotCompileException,
            NotFoundException {
        // Header
        addSave("public int __save() throws java.lang.Exception {\n");

        // Streams
        addSave("java.io.ByteArrayOutputStream baos = new
java.io.ByteArrayOutputStream();\n");
        addSave("java.io.DataOutputStream dos = new
java.io.DataOutputStream(baos);\n");

        // Save the superclass if it is persistable.
        CtClass superClass = this.ctClass.getSuperclass();
        ClassVerifier verifier = new ClassVerifier(superClass);
        if (verifier.isPersistable()) {

addSave(SuperClassGenerator.generateSaveSource(superClass));
        }

        // RecordStore (open)
        addSave("javax.microedition.rms.RecordStore rs =
net.sourceforge.floggy.PersistableManager.getRecordStore(\""
                + this.ctClass.getName() + "\");\n");

        CtField[] fields = ctClass.getDeclaredFields();
        if (fields != null && fields.length > 0) {
            SourceCodeGenerator generator;
            CtField field;
            for (int i = 0; i < fields.length; i++) {
                field = fields[i];

                // Ignores compiler fields.
                if (field.getName().equals("__id")) {
                    continue;
                }

                generator =
SourceCodeGeneratorFactory.getSourceCodeGenerator(
                        field.getName(), field.getType());
                if (generator != null) {
                    this.addSave(generator.getSaveCode());
                }
            }
        }
```

```
        // RecordStore (save and close)
        addSave("if(this.__id == -1) {\n");
        addSave("this.__id = rs.addRecord(baos.toByteArray(), 0,
baos.size());\n");
        addSave("}\n");
        addSave("else {\n");
        addSave("rs.setRecord(this.__id, baos.toByteArray(), 0,
baos.size());\n");
        addSave("}\n");
        addSave("rs.closeRecordStore();\n");

        // Close the streams
        addSave("dos.close();\n");

        addSave("return this.__id;\n");
        addSave("}");

        debug("");
        debug(CodeFormatter.format(functionSave.toString()));

        CtMethod saveMethod = CtNewMethod
                .make(functionSave.toString(), ctClass);
        ctClass.addMethod(saveMethod);
    }

    private void addLoad_Buffer(String source) {
        this.functionLoad_Buffer.append(source);
    }

    private void addSave(String source) {
        this.functionSave.append(source);
    }

    private static void debug(String message) {
        if (DEBUG) {
            System.out.println(message);
        }

    }
}
```

DateGenerator

```
package net.sourceforge.floggy.codegen;

import javassist.CtClass;
import javassist.NotFoundException;

public class DateGenerator extends SourceCodeGenerator {

    public DateGenerator(String fieldName, CtClass classType) {
        super(fieldName, classType);
    }

    public void initLoadCode() throws NotFoundException {
        addLoadCode("if(dis.readByte() == 0) {");
        addLoadCode(fieldName + " = new
java.util.Date(dis.readLong());");
```

```
        addLoadCode("}");
        addLoadCode("else {");
        addLoadCode(fieldName + " = null;");
        addLoadCode("}");
    }

    public void initSaveCode() throws NotFoundException {
        addSaveCode("if(" + fieldName + " == null) {");
        addSaveCode("dos.writeByte(1);");
        addSaveCode("}");
        addSaveCode("else {");
        addSaveCode("dos.writeByte(0);");
        addSaveCode("dos.writeLong(" + fieldName + ".getTime());");
        addSaveCode("}");
    }

    public boolean isInstanceOf() throws NotFoundException {
        return classType.getName().equals("java.util.Date");
    }
```

PersistableGenerator

```
os direiros reservados aos autores.
 */
package net.sourceforge.floggy.codegen;

import javassist.CtClass;
import javassist.NotFoundException;
import net.sourceforge.floggy.Persistable;

public class PersistableGenerator extends SourceCodeGenerator {

    public PersistableGenerator(String fieldName, CtClass classType) {
        super(fieldName, classType);
    }

    public void initLoadCode() throws NotFoundException {
        addLoadCode("if(dis.readByte() == 0) {");
        addLoadCode("Class someClass = Class.forName(\"" +
classType.getName()
                + "\");");
        addLoadCode(fieldName + " = (" + classType.getName()
                + ") someClass.newInstance();");
        addLoadCode("((net.sourceforge.floggy.__Persistable) " +
fieldName
                + ").__load(dis.readInt());");
        addLoadCode("}");
        addLoadCode("else {");
        addLoadCode(fieldName + " = null;");
        addLoadCode("}");
    }

    public void initSaveCode() throws NotFoundException {
        addSaveCode("if(" + fieldName + " == null) {");
        addSaveCode("dos.writeByte(1);");
        addSaveCode("}");
        addSaveCode("else {");
```

```
        addSaveCode("dos.writeByte(0);");

        addSaveCode("((net.sourceforge.floggy.__Persistable) " +
fieldName
                + ").__save();");

addSaveCode("dos.writeInt(((net.sourceforge.floggy.__Persistable) "
                + fieldName + ").__getId());");
        addSaveCode("}");
    }

    public boolean isInstanceOf() throws NotFoundException {
        CtClass persistableClass = classType.getClassPool().get(
                Persistable.class.getName());
        return classType.subtypeOf(persistableClass);
    }
}
```

PrimitiveTypeGenerator

```
package net.sourceforge.floggy.codegen;

import javassist.CtClass;
import javassist.NotFoundException;

public class PrimitiveTypeGenerator extends SourceCodeGenerator {

    public PrimitiveTypeGenerator(String fieldName, CtClass classType)
{
        super(fieldName, classType);
    }

    public void initLoadCode() throws NotFoundException {
        addLoadCode(fieldName + " = dis.read" + getType() + "();");

    }

    public void initSaveCode() throws NotFoundException {
        addSaveCode("dos.write" + getType() + "(" + fieldName + ");");
    }

    public boolean isInstanceOf() {
        return classType.isPrimitive();
    }

    private String getType() throws NotFoundException {
        if (classType == CtClass.booleanType) {
            return "Boolean";
        }
        if (classType == CtClass.byteType) {
            return "Byte";
        }
        if (classType == CtClass.charType) {
            return "Char";
        }
        if (classType == CtClass.doubleType) {
            return "Double";
        }
```

```
        if (classType == CtClass.floatType) {
            return "Float";
        }
        if (classType == CtClass.intType) {
            return "Int";
        }
        if (classType == CtClass.longType) {
            return "Long";
        }
        if (classType == CtClass.shortType) {
            return "Short";
        }

        throw new NotFoundException("Type not found!");
    }

}
```

SourceCodeGenerator

```
package net.sourceforge.floggy.codegen;

import javassist.CtClass;
import javassist.NotFoundException;

public abstract class SourceCodeGenerator {

    protected String fieldName;

    protected CtClass classType;

    private StringBuffer loadCode;

    private StringBuffer saveCode;

    protected SourceCodeGenerator(String fieldName, CtClass classType)
{
        this.fieldName = fieldName;
        this.classType = classType;
        this.loadCode = new StringBuffer();
        this.saveCode = new StringBuffer();
    }

    protected void addLoadCode(String part) {
        this.loadCode.append(part);
        this.loadCode.append('\n');
    }

    protected void addSaveCode(String part) {
        this.saveCode.append(part);
        this.saveCode.append('\n');
    }

    public String getLoadCode() throws NotFoundException {
        if (this.loadCode.length() == 0) {
            this.initLoadCode();
        }
```

```
        return this.loadCode.toString();
    }

    public String getSaveCode() throws NotFoundException {
        if (this.saveCode.length() == 0) {
            this.initSaveCode();
        }

        return this.saveCode.toString();
    }

    public abstract void initLoadCode() throws NotFoundException;

    public abstract void initSaveCode() throws NotFoundException;

    public abstract boolean isInstanceOf() throws NotFoundException;
}
```

SourceCodeGeneratorFactory

```
package net.sourceforge.floggy.codegen;

import javassist.CtClass;
import javassist.NotFoundException;

public class SourceCodeGeneratorFactory {

    public static SourceCodeGenerator getSourceCodeGenerator(String
fieldName,
            CtClass classType) throws NotFoundException {
        SourceCodeGenerator generator;

        // Primitive types
        generator = new PrimitiveTypeGenerator(fieldName, classType);
        if (generator.isInstanceOf()) {
            return generator;
        }

        // Wrapper classes
        generator = new WrapperGenerator(fieldName, classType);
        if (generator.isInstanceOf()) {
            return generator;
        }

        // String
        generator = new StringGenerator(fieldName, classType);
        if (generator.isInstanceOf()) {
            return generator;
        }

        // Date
        generator = new DateGenerator(fieldName, classType);
        if (generator.isInstanceOf()) {
            return generator;
        }

        // Persistable
        generator = new PersistableGenerator(fieldName, classType);
```

```
        if (generator.isInstanceOf()) {
            return generator;
        }

        // Array
        generator = new ArrayGenerator(fieldName, classType);
        if (generator.isInstanceOf()) {
            return generator;
        }

        // Vector
        generator = new VectorGenerator(fieldName, classType);
        if(generator.isInstanceOf()) {
            return generator;
        }

        return null;
    }

}
```

StringGenerator

```
package net.sourceforge.floggy.codegen;

import javassist.CtClass;
import javassist.NotFoundException;

public class StringGenerator extends SourceCodeGenerator {

    public StringGenerator(String fieldName, CtClass classType) {
        super(fieldName, classType);
    }

    public void initLoadCode() throws NotFoundException {
        addLoadCode("if(dis.readByte() == 0) {");
        addLoadCode(fieldName + " = dis.readUTF();");
        addLoadCode("}");
        addLoadCode("else {");
        addLoadCode(fieldName + " = null;");
        addLoadCode("}");
    }

    public void initSaveCode() throws NotFoundException {
        addSaveCode("if(" + fieldName + " == null) {");
        addSaveCode("dos.writeByte(1);");
        addSaveCode("}");
        addSaveCode("else {");
        addSaveCode("dos.writeByte(0);");
        addSaveCode("dos.writeUTF(" + fieldName + ");");
        addSaveCode("}");
    }

    public boolean isInstanceOf() throws NotFoundException {
        return classType.getName().equals(String.class.getName());
    }
}
```

SuperClassGenerator

```
package net.sourceforge.floggy.codegen;

import javassist.CtClass;

public class SuperClassGenerator {

    public static String generateLoadSource(CtClass superClass) {
        String source = "\n";

        source += "javax.microedition.rms.RecordStore rs =
net.sourceforge.floggy.PersistableManager.getRecordStore(\""
                + superClass.getName() + "\");\n";
        source += "int superClassId = dis.readInt();\n";
        source += "byte[] superClassBuffer =
rs.getRecord(superClassId);\n";
        source += "rs.closeRecordStore();\n";
        source += "super.__load(superClassBuffer);\n";
        source += "\n";

        return source;
    }

    public static String generateSaveSource(CtClass superClass) {
        String source = "\n";

        source += "int superClassId = super.__save();\n";
        source += "dos.writeInt(superClassId);\n";
        source += "\n";

        return source;
    }
}
```

VectorGenerator

```
package net.sourceforge.floggy.codegen;

import javassist.CtClass;
import javassist.NotFoundException;

public class VectorGenerator extends SourceCodeGenerator {

    public VectorGenerator(String fieldName, CtClass classType) {
        super(fieldName, classType);
    }

    public void initLoadCode() throws NotFoundException {
        addLoadCode("if(dis.readByte() == 1) {");
        addLoadCode(fieldName + " = null;");
        addLoadCode("}");
        addLoadCode("else {");
        addLoadCode("int size = dis.readInt();");
        addLoadCode(fieldName + " = new java.util.Vector(size);");
        addLoadCode("for(int i = 0; i < size; i++) {");
        addLoadCode("if(dis.readByte() == 1) {");
        addLoadCode(fieldName + ".addElement((Object) null);");
```

```
        addLoadCode("continue;");
        addLoadCode("}");
        addLoadCode("else {");
        addLoadCode("String className = dis.readUTF();");
        addLoadCode("if(className.equals(\"java.lang.Boolean\")) {");
        addLoadCode(fieldName + ".addElement(new
java.lang.Boolean(dis.readBoolean()));");
        addLoadCode("continue;");
        addLoadCode("}");
       addLoadCode("if(className.equals(\"java.lang.Byte\")) {");
        addLoadCode(fieldName + ".addElement(new
java.lang.Byte(dis.readByte()));");
        addLoadCode("continue;");
        addLoadCode("}");
        addLoadCode("if(className.equals(\"java.lang.Character\"))
{");
        addLoadCode(fieldName + ".addElement(new
java.lang.Character(dis.readChar()));");
        addLoadCode("continue;");
        addLoadCode("}");
        addLoadCode("if(className.equals(\"java.lang.Double\")) {");
        addLoadCode(fieldName + ".addElement(new
java.lang.Double(dis.readDouble()));");
        addLoadCode("continue;");
        addLoadCode("}");
        addLoadCode("if(className.equals(\"java.lang.Float\")) {");
        addLoadCode(fieldName + ".addElement(new
java.lang.Float(dis.readFloat()));");
        addLoadCode("continue;");
        addLoadCode("}");
        addLoadCode("if(className.equals(\"java.lang.Integer\")) {");
        addLoadCode(fieldName + ".addElement(new
java.lang.Integer(dis.readInt()));");
        addLoadCode("continue;");
        addLoadCode("}");
        addLoadCode("if(className.equals(\"java.lang.Long\")) {");
        addLoadCode(fieldName + ".addElement(new
java.lang.Long(dis.readLong()));");
        addLoadCode("continue;");
        addLoadCode("}");
        addLoadCode("if(className.equals(\"java.lang.Short\")) {");
        addLoadCode(fieldName + ".addElement(new
java.lang.Short(dis.readShort()));");
        addLoadCode("continue;");
        addLoadCode("}");
        addLoadCode("if(className.equals(\"java.lang.String\")) {");
        addLoadCode(fieldName + ".addElement(new
java.lang.String(dis.readUTF()));");
        addLoadCode("continue;");
        addLoadCode("}");
        addLoadCode("if(className.equals(\"java.util.Date\")) {");
        addLoadCode(fieldName + ".addElement(new
java.util.Date(dis.readLong()));");
        addLoadCode("continue;");
        addLoadCode("}");
        addLoadCode("Class persistableClass =
java.lang.Class.forName(className);");
        addLoadCode("Object object =
persistableClass.newInstance();");
```

```
        addLoadCode("if(object instanceof
net.sourceforge.floggy.__Persistable) {");
        addLoadCode("((net.sourceforge.floggy.__Persistable)
object).__load(dis.readInt());");
        addLoadCode(fieldName + ".addElement(object);");
        addLoadCode("}");
        addLoadCode("}"); // else
        addLoadCode("}"); // for
        addLoadCode("}"); // else
    }

    public void initSaveCode() throws NotFoundException {
        addSaveCode("if(" + fieldName + " == null) {");
        addSaveCode("dos.writeByte(1);");

        addSaveCode("}");
        addSaveCode("else {");
        addSaveCode("dos.writeByte(0);");
        addSaveCode("int size = " + fieldName + ".size();");
        addSaveCode("dos.writeInt(size);");
        addSaveCode("for(int i = 0; i < size; i++) {");
        addSaveCode("Object object = " + fieldName +
".elementAt(i);");
        addSaveCode("if(object == null) {");
        addSaveCode("dos.writeByte(1);");
        addSaveCode("continue;");
        addSaveCode("}"); // if(object == null) {
        addSaveCode("dos.writeByte(0);");
        addSaveCode("dos.writeUTF(object.getClass().getName());");
        addSaveCode("if(object instanceof java.lang.Boolean) {");
        addSaveCode("dos.writeBoolean(((java.lang.Boolean)
object).booleanValue());");
        addSaveCode("}");
        addSaveCode("else if(object instanceof java.lang.Byte) {");
        addSaveCode("dos.writeByte(((java.lang.Byte)
object).byteValue());");
        addSaveCode("}");
        addSaveCode("else if(object instanceof java.lang.Character)
{");
        addSaveCode("dos.writeChar(((java.lang.Character)
object).charValue());");
        addSaveCode("}");
        addSaveCode("else if(object instanceof java.lang.Double) {");
        addSaveCode("dos.writeDouble(((java.lang.Double)
object).doubleValue());");
        addSaveCode("}");
        addSaveCode("else if(object instanceof java.lang.Float) {");
        addSaveCode("dos.writeFloat(((java.lang.Float)
object).floatValue());");
        addSaveCode("}");
        addSaveCode("else if(object instanceof java.lang.Integer) {");
        addSaveCode("dos.writeInt(((java.lang.Integer)
object).intValue());");
        addSaveCode("}");
        addSaveCode("else if(object instanceof java.lang.Long) {");
        addSaveCode("dos.writeLong(((java.lang.Long)
object).longValue());");
        addSaveCode("}");
        addSaveCode("else if(object instanceof java.lang.Short) {");
```

```
        addSaveCode("dos.writeShort(((java.lang.Short)
object).shortValue());");
        addSaveCode("}");
        addSaveCode("else if(object instanceof java.lang.String) {");
        addSaveCode("dos.writeUTF(((java.lang.String) object));");
        addSaveCode("}");
        addSaveCode("else if(object instanceof java.util.Date) {");
        addSaveCode("dos.writeLong(((java.util.Date)
object).getTime());");
        addSaveCode("}");
        addSaveCode("else if(object instanceof
net.sourceforge.floggy.__Persistable) {");
        addSaveCode("int id = ((net.sourceforge.floggy.__Persistable)
object).__save();");
        addSaveCode("dos.writeInt(id);");
        addSaveCode("}");
        addSaveCode("}");
        addSaveCode("}");
    }

    public boolean isInstanceOf() throws NotFoundException {
        return classType.getName().equals("java.util.Vector");
    }

}
```

9.3 Código Fonte Floggy Framework

__Persistable

```
package net.sourceforge.floggy;

public interface __Persistable {

    public int __getId();

    public void __load(int id) throws Exception;

    public void __load(byte[] buffer) throws Exception;

    public int __save() throws Exception;

    // public void __delete() throws Exception;


}
```

Comparator

```
package net.sourceforge.floggy;

public interface Comparator {

      public static final int PRECEDES = -1;
```

```
      public static final int EQUIVALENT = 0;

      public static final int FOLLOWS = 1;

      public int compare(Persistable p1, Persistable p2);

}
```

Filter

```
package net.sourceforge.floggy;

public interface Filter {

      public abstract boolean matches(Persistable persistable);

}
```

FloggyException

```
package net.sourceforge.floggy;

public class FloggyException extends Exception {

    /**
     * @param message
     */
    public FloggyException(String message) {
        super(message);
    }


}
```

NoMoreObjectsException

```
package net.sourceforge.floggy;

public class NoMoreObjectsException extends RuntimeException {

      public NoMoreObjectsException() {
            super();
      }

      public NoMoreObjectsException(String message) {
            super(message);
      }
}
```

ObjectSet

```
package net.sourceforge.floggy;

/**
```

```
 * @author move0005
 *
 */
public class ObjectSet {

	int[] ids;

	int index;

	int objectCount;

	Class persistableClass;

	PersistableManager manager;

	/**
	 * Creates a new instance of an object set.
	 *
	 * @param ids
	 * @param persistableClass
	 * @param filter
	 * @param comparator
	 */
	protected ObjectSet(int[] ids, Class persistableClass) {
		this.ids = ids;
		this.persistableClass = persistableClass;

		// Init attributes
		this.index = -1;
		this.objectCount = (ids == null ? 0 : ids.length);

		// Retrieve the manager instance
		this.manager = PersistableManager.getInstance();
	}

	public boolean hasNext() {
		return index + 1 < objectCount;
	}

	public boolean hasPrevious() {
		return index > 0;
	}

	public int nextId() {
		if (!hasNext()) {
			throw new NoMoreObjectsException(
					"No more next objects in this object
set.");
		}

		return ids[++index];
	}

	public int previousId() {
		if (!hasPrevious()) {
			throw new NoMoreObjectsException(
					"No more previous objects in this object
set.");
		}
```

```
            return ids[--index];
      }

      public int firstId() {
            if (objectCount == 0) {
                  throw new NoMoreObjectsException("No objects in this
object set.");
            }

            this.index = 0;
            return ids[this.index];
      }

      public int lastId() {
            if (objectCount == 0) {
                  throw new NoMoreObjectsException("No objects in this
object set.");
            }

            this.index = objectCount - 1;
            return ids[this.index];
      }

      public Persistable next() throws FloggyException {
            return load(nextId());
      }

      public Persistable previous() throws FloggyException {
            return load(previousId());
      }

      public Persistable first() throws FloggyException {
            return load(firstId());
      }

      public Persistable last() throws FloggyException {
            return load(lastId());
      }

      private Persistable load(int id) throws FloggyException {
            Persistable persistable = newObject();
            manager.load(persistable, id);
            return persistable;
      }

      public Persistable get(int index) throws FloggyException {
            return load(this.getId(index));
      }

      public int getId(int index) throws FloggyException {
            this.index = index;
            return this.ids[this.index];
      }

      public int count() {
            return objectCount;
      }
```

```
      private Persistable newObject() throws FloggyException {
            try {
                  return (Persistable) persistableClass.newInstance();
            } catch (Exception e) {
                  throw new FloggyException(e.getMessage());
            }
      }


}
```

Persistable

```
package net.sourceforge.floggy;

public interface Persistable {

}
```

```
package net.sourceforge.floggy;

import javax.microedition.rms.RecordEnumeration;
import javax.microedition.rms.RecordStore;
import javax.microedition.rms.RecordStoreException;

import net.sourceforge.floggy.search.ObjectComparator;
import net.sourceforge.floggy.search.ObjectFilter;
```

PersistableManager

```
public class PersistableManager {

    /**
     * The instance.
     */

    private static PersistableManager instance;

    public static PersistableManager getInstance() {
        if (instance == null) {
            instance = new PersistableManager();
        }
        return instance;
    }

    /**
     * Creates a new instance of FloggyManager.
     */
    private PersistableManager() {
    }

    public PersistableMetadata getMetadata(Class persistableClass)
            throws FloggyException {
        RecordStore rs = getRecordStore(persistableClass.getName());

        final int objectCount;
        final long lastModified;
        final int size;
        final int version;
```

```
        try {
            objectCount = rs.getNumRecords();
            lastModified = rs.getLastModified();
            size = rs.getSize();
            version = rs.getVersion();
        } catch (RecordStoreException e) {
            throw new FloggyException(
                    "Error requesting information from repository.");
        }

        PersistableMetadata pm = new PersistableMetadata() {
            public long getLastModified() {
                return lastModified;
            }

            public int getObjectCount() {
                return objectCount;
            }

            public int getSize() {
                return size;
            }

            public int getVersion() {
                return version;
            }
        };
        return pm;
    }

    /**
     * Returns a RecordStore given the class name of the persistable
class.
     *
     * @param className
     *            Class name of a persistable class.
     * @return The RecordStore corresponding to the persistable class.
     * @throws FloggyException
     *             If the class name does not represents a persistable
class.
     */
    public static RecordStore getRecordStore(String className)
            throws FloggyException {
        try {
            if (className.lastIndexOf('.') != -1) {
                className =
className.substring(className.lastIndexOf('.') + 1);
            }
            className += className.hashCode();
            return RecordStore.openRecordStore(className, true);
        } catch (Exception e) {
            throw new FloggyException(e.getMessage());
        }
    }

    /**
     * @param persistable
     * @param id
```

```
     * @throws Exception
     */
    public void load(Persistable persistable, int id) throws
FloggyException {
        if (persistable instanceof __Persistable) {
            __Persistable __persistable = (__Persistable) persistable;

            try {
                __persistable.__load(id);
            } catch (Exception e) {
                throw new FloggyException(e.getMessage());
            }

            return;
        }

        throw new FloggyException(persistable.getClass().getName()
                + " is not a valid persistable class.");
    }

    /**
     * @param persistable
     * @return
     * @throws FloggyException
     */
    public int save(Persistable persistable) throws FloggyException {
        if (persistable instanceof __Persistable) {
            __Persistable __persistable = (__Persistable) persistable;

            try {
                return __persistable.__save();
            } catch (Exception e) {
                throw new FloggyException(e.getMessage());
            }
        }

        throw new FloggyException(persistable.getClass().getName()
                + "\" is not a valid persistable class.");
    }


    /**
     * @param persistable
     * @return
     * @throws FloggyException
     */
    public void delete(Persistable persistable) throws FloggyException
{
        if (persistable instanceof __Persistable) {
            __Persistable __persistable = (__Persistable) persistable;

            try {
                // TODO Gerar o método __delete() dinamicamente
                RecordStore rs = getRecordStore(persistable.getClass()
                        .getName());
                rs.deleteRecord(__persistable.__getId());
                rs.closeRecordStore();
            } catch (Exception e) {
                throw new FloggyException(e.getMessage());
```

```
            }

            return;
        }

        throw new FloggyException(persistable.getClass().getName()
                + "\" is not a valid persistable class.");
    }

    /**
     *
     * @param persistableClass
     * @param filter
     * @param comparator
     * @return
     */
    public ObjectSet find(Class persistableClass, Filter filter,
            Comparator comparator) throws FloggyException {
        ObjectFilter objectFilter = null;
        ObjectComparator objectComparator = null;

        // Creates an auxiliar filter (if necessary)
        if (filter != null) {
            objectFilter = new ObjectFilter(persistableClass, filter);
        }

        // Creates an auxiliar comparator (if necessary)
        if (comparator != null) {
            objectComparator = new ObjectComparator(persistableClass,
                    comparator);
        }

        // Searchs the repository and create an object set as result.
        int[] ids = null;

        RecordStore rs = getRecordStore(persistableClass.getName());
        try {
            RecordEnumeration en = rs.enumerateRecords(objectFilter,
                    objectComparator, false);
            int numRecords = en.numRecords();
            if (numRecords > 0) {
                ids = new int[numRecords];
                for (int i = 0; i < numRecords; i++) {
                    ids[i] = en.nextRecordId();
                }
            }
        } catch (RecordStoreException e) {
            throw new FloggyException(e.getMessage());
        }

        return new ObjectSet(ids, persistableClass);
    }
}
```

PersistableMetadata

```
package net.sourceforge.floggy;
```

```
public interface PersistableMetadata {

      /**
       * Returns the number of objects.
       *
       * @return
       */
      public int getObjectCount();

      /**
       * Return the
       *
       * @return
       */
      public long getLastModified();


      /**
       *
       * @return
       */
      public int getSize();

      /**
       *
       * @return
       */
      public int getVersion();

}
```

ObjectFilter

```
package net.sourceforge.floggy.search;

import javax.microedition.rms.RecordFilter;

import net.sourceforge.floggy.Filter;
import net.sourceforge.floggy.FloggyException;
import net.sourceforge.floggy.Persistable;
import net.sourceforge.floggy.__Persistable;

public class ObjectFilter implements RecordFilter {

    Persistable persistable;

    Filter filter;

    public ObjectFilter(Class persistableClass, Filter filter)
            throws FloggyException {
        try {
            this.persistable = (Persistable)
persistableClass.newInstance();

            if (!(this.persistable instanceof __Persistable)) {
                throw new FloggyException(persistableClass.getName()
                        + " is not a valid persistable class.");
            }
```

```
        } catch (Exception e) {
            throw new FloggyException(e.getMessage());
        }

        this.filter = filter;
    }

    /**
     * @floggy Modified in compilation time.
     */
    public boolean matches(byte[] buffer) {
        try {
            ((__Persistable) this.persistable).__load(buffer);
        } catch (Exception e) {
            // Ignore
        }

        return filter.matches(persistable);
    }
}
```

ObjectComparator

```
package net.sourceforge.floggy.search;

import javax.microedition.rms.RecordComparator;

import net.sourceforge.floggy.Comparator;
import net.sourceforge.floggy.FloggyException;
import net.sourceforge.floggy.Persistable;
import net.sourceforge.floggy.__Persistable;

public class ObjectComparator implements RecordComparator {

    Comparator comparator;

    Persistable obj1;

    Persistable obj2;

    public ObjectComparator(Class persistableClass, Comparator
comparator)
            throws FloggyException {
        this.comparator = comparator;

        try {
            this.obj1 = (Persistable) persistableClass.newInstance();
            if (!(this.obj1 instanceof __Persistable)) {
                throw new FloggyException(persistableClass.getName()
                        + " is not a valid persistable class.");
            }

            this.obj2 = (Persistable) persistableClass.newInstance();
        } catch (Exception e) {
            throw new FloggyException(e.getMessage());
        }
    }
```

```
    /**
     * @floggy modified in compilation time.
     */
    public int compare(byte[] buffer1, byte[] buffer2) {
        try {
            ((__Persistable) obj1).__load(buffer1);
            ((__Persistable) this.obj2).__load(buffer2);
        } catch (Exception e) {
            // Ignore
        }

        int result = comparator.compare(obj1, obj2);
        if (result < 0) {
            return RecordComparator.PRECEDES;
        }
        if (result > 0) {
            return RecordComparator.FOLLOWS;
        }
        return RecordComparator.EQUIVALENT;
    }

}
```

9.4 Código Fonte Caso de Estudo

HospitalMIDlet

```
package net.sourceforge.floggy;

import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;

import net.sourceforge.floggy.gui.MainForm;

public class HospitalMIDlet extends MIDlet {

    static Display display;

    static MIDlet midlet;

    public HospitalMIDlet() {
        super();

        display = Display.getDisplay(this);
        midlet = this;
    }

    protected void startApp() throws MIDletStateChangeException {
        MainForm mainForm = new MainForm();
        HospitalMIDlet.setCurrent(mainForm);
    }

    protected void pauseApp() {
    }
```

```
    protected void destroyApp(boolean arg0) throws
MIDletStateChangeException {
    }

    public static void setCurrent(Displayable displayable) {
        display.setCurrent(displayable);
    }

    public static void sair(){
        midlet.notifyDestroyed();
    }
}
```

Pessoa

```
package net.sourceforge.floggy.model;

import java.util.Calendar;
import java.util.Date;

import net.sourceforge.floggy.Persistable;

public class Pessoa implements Persistable {

    String cpf;

    String nome;

    Date dataNascimento;

    transient int idade;

    public Pessoa() {
        //
    }

    public Pessoa(String cpf, String nome, Date dataNascimento) {
        setCpf(cpf);
        setNome(nome);
        setDataNascimento(dataNascimento);
    }

    public String getCpf() {
        return cpf;
    }

    public void setCpf(String cpf) {
        this.cpf = cpf;
    }

    public Date getDataNascimento() {
        return dataNascimento;
    }

    public void setDataNascimento(Date dataNascimento) {
        this.dataNascimento = dataNascimento;
        if (dataNascimento != null) {
```

```
            Calendar c1 = Calendar.getInstance();
            Calendar c2 = Calendar.getInstance();

            c2.setTime(dataNascimento);
            this.idade = c1.get(Calendar.YEAR) -
c2.get(Calendar.YEAR);
        } else {
            this.idade = 0;
        }

    }

    public int getIdade() {
        return idade;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

}
```

Formacao

```
package net.sourceforge.floggy.model;

import net.sourceforge.floggy.Persistable;

public class Formacao implements Persistable {

    String formacao;

    public Formacao() {

    }

    public Formacao(String formacao) {
        this.formacao = formacao;
    }

    public String getFormacao() {
        return formacao;
    }

    public void setFormacao(String formacao) {
        this.formacao = formacao;
    }

    public boolean equals(Object object) {
        if (object instanceof Formacao) {
            Formacao formacao = (Formacao) object;
            return this.formacao.equals(formacao.getFormacao());
        }
```

```
        return false;
    }
}
/*
```

Internacao

```
package net.sourceforge.floggy.model;

import java.util.Date;

import net.sourceforge.floggy.Persistable;

public class Internacao implements Persistable {

    Date dtEntrada;

    Date dtSaida;

    String motivo;

    Paciente paciente;

    Medico medico;

    Leito leito;

    public Date getDtEntrada() {
        return dtEntrada;
    }

    public void setDtEntrada(Date dtEntrada) {
        this.dtEntrada = dtEntrada;
    }

    public Date getDtSaida() {
        return dtSaida;
    }

    public void setDtSaida(Date dtSaida) {
        this.dtSaida = dtSaida;
    }

    public Leito getLeito() {
        return leito;
    }

    public void setLeito(Leito leito) {
        this.leito = leito;
    }

    public Medico getMedico() {
        return medico;
    }

    public void setMedico(Medico medico) {
        this.medico = medico;
    }
```

```
    public String getMotivo() {
        return motivo;
    }

    public void setMotivo(String motivo) {
        this.motivo = motivo;
    }

    public Paciente getPaciente() {
        return paciente;
    }

    public void setPaciente(Paciente paciente) {
        this.paciente = paciente;
    }

}
```

Leito

```
package net.sourceforge.floggy.model;

import net.sourceforge.floggy.Persistable;

public class Leito implements Persistable {

    int andar;

    String numero;

    public Leito(){

    }

    public Leito (int andar, String numero) {
        this.andar  = andar;
        this.numero = numero;
    }

    public int getAndar() {
        return andar;
    }

    public void setAndar(int andar) {
        this.andar = andar;
    }

    public String getNumero() {
        return numero;
    }

    public void setNumero(String numero) {
        this.numero = numero;
    }



}
```

Medico

```
package net.sourceforge.floggy.model;

import java.util.Vector;

import net.sourceforge.floggy.Persistable;

public class Medico extends Pessoa implements Persistable {

    String crm;

    Vector formacoes;

    public Medico() {
        //
    }

    public String getCrm() {
        return crm;
    }

    public void setCrm(String crm) {
        this.crm = crm;
    }

    public Vector getFormacoes() {
        return formacoes;
    }

    public void setFormacoes(Vector formacoes) {
        this.formacoes = formacoes;
    }
}
```

Paciente

```
package net.sourceforge.floggy.model;

import net.sourceforge.floggy.Persistable;

public class Paciente extends Pessoa implements Persistable {

    private static int TELEFONE_CASA = 0;

    private static int TELEFONE_CELULAR = 1;

    private static int TELEFONE_TRABALHO = 2;

    String endereco;

    String[] telefones;

    boolean particular;

    public Paciente() {
        this.telefones = new String[3];
```

```
    }

    public String getEndereco() {
        return endereco;
    }

    public void setEndereco(String endereco) {
        this.endereco = endereco;
    }

    public boolean isParticular() {
        return particular;
    }


    public void setParticular(boolean particular) {
        this.particular = particular;
    }

    public String getTelefoneCasa() {
        return this.telefones[TELEFONE_CASA];
    }

    public void setTelefoneCasa(String telefone) {
        this.telefones[TELEFONE_CASA] = telefone;
    }

    public String getTelefoneCelular() {
        return this.telefones[TELEFONE_CELULAR];
    }

    public void setTelefoneCelular(String telefone) {
        this.telefones[TELEFONE_CELULAR] = telefone;
    }

    public String getTelefoneTrabalho() {
        return this.telefones[TELEFONE_TRABALHO];
    }

    public void setTelefoneTrabalho(String telefone) {
        this.telefones[TELEFONE_TRABALHO] = telefone;
    }

}
```

FormacaoForm

```
package net.sourceforge.floggy.gui;

import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.TextField;

import net.sourceforge.floggy.FloggyException;
import net.sourceforge.floggy.HospitalMIDlet;
```

```
import net.sourceforge.floggy.PersistableManager;
import net.sourceforge.floggy.model.Formacao;

public class FormacaoForm extends Form implements CommandListener {

    Formacao formacao;

    TextField txtFormacao;

    Command cmdOk;

    Command cmdCancelar;

    public FormacaoForm(Formacao formacao) {
        super("Formação");

        this.formacao = formacao;

        iniciaComponentes();
    }

private void iniciaComponentes() {
        this.txtFormacao = new TextField("Formação",
formacao.getFormacao(), 30, TextField.ANY);
        this.append(this.txtFormacao);

        this.cmdOk = new Command("Ok", Command.OK, 0);
        this.addCommand(this.cmdOk);

        this.cmdCancelar = new Command("Cancelar", Command.CANCEL, 1);
        this.addCommand(this.cmdCancelar);

        this.setCommandListener(this);
}

public void commandAction(Command cmd, Displayable dsp) {
        if(cmd == this.cmdOk) {
            PersistableManager pm = PersistableManager.getInstance();

            try {
                formacao.setFormacao(this.txtFormacao.getString());
                pm.save(formacao);
            } catch (FloggyException e) {
                e.printStackTrace();
            }
        }

        HospitalMIDlet.setCurrent(new FormacaoList());
    }
}
```

FormacaoList

```
package net.sourceforge.floggy.gui;

import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
```

```
import javax.microedition.lcdui.List;

import net.sourceforge.floggy.Comparator;
import net.sourceforge.floggy.FloggyException;
import net.sourceforge.floggy.HospitalMIDlet;
import net.sourceforge.floggy.ObjectSet;
import net.sourceforge.floggy.Persistable;
import net.sourceforge.floggy.PersistableManager;
import net.sourceforge.floggy.model.Formacao;

public class FormacaoList extends List implements CommandListener,
Comparator {

    ObjectSet formacoes;

    Command cmdInserir;

    Command cmdAlterar;

    Command cmdExcluir;

    Command cmdVoltar;

    public FormacaoList() {
        super("Lista de Formações", List.IMPLICIT);

        iniciaDados();
        iniciaComponentes();
    }

    private void iniciaDados() {
        PersistableManager pm = PersistableManager.getInstance();

        try {
            this.deleteAll();

            formacoes = pm.find(Formacao.class, null, this);
            while (formacoes.hasNext()) {
                Formacao element = (Formacao) formacoes.next();
                this.append(element.getFormacao(), null);
            }

        } catch (FloggyException e) {
            e.printStackTrace();
        }
    }

    private void iniciaComponentes() {
        this.cmdVoltar = new Command("Voltar", Command.BACK, 0);
        this.addCommand(this.cmdVoltar);

        this.cmdInserir = new Command("Inserir", Command.ITEM, 1);
        this.addCommand(this.cmdInserir);

        this.cmdAlterar = new Command("Alterar", Command.ITEM, 2);
        this.addCommand(this.cmdAlterar);

        this.cmdExcluir = new Command("Excluir", Command.ITEM, 3);
        this.addCommand(this.cmdExcluir);
```

```
        this.setCommandListener(this);
    }

    public void commandAction(Command cmd, Displayable dsp) {
        if (cmd == this.cmdVoltar) {
            MainForm mainForm = new MainForm();
            HospitalMIDlet.setCurrent(mainForm);
        } else if (cmd == this.cmdInserir) {
            Formacao formacao = new Formacao();
            HospitalMIDlet.setCurrent(new FormacaoForm(formacao));
        } else if (cmd == this.cmdAlterar) {
            if (this.getSelectedIndex() != -1) {
                Formacao formacao = null;

                try {
                    formacao = (Formacao) formacoes
                            .get(this.getSelectedIndex());
                    HospitalMIDlet.setCurrent(new
FormacaoForm(formacao));
                } catch (FloggyException e) {
                    e.printStackTrace();
                }
            }
        } else if (cmd == this.cmdExcluir) {
            if (this.getSelectedIndex() != -1) {

                try {
                    Formacao formacao = (Formacao) formacoes.get(this
                            .getSelectedIndex());
                    PersistableManager.getInstance().delete(formacao);
                    this.iniciaDados();
                } catch (FloggyException e) {
                    e.printStackTrace();
                }
            }
        }
    }

    public int compare(Persistable p1, Persistable p2) {
        Formacao f1 = (Formacao) p1;
        Formacao f2 = (Formacao) p2;

        return f1.getFormacao().compareTo(f2.getFormacao());
    }
}
```

InternacaoForm

```
/*
 * Criado em 20/09/2005.
 *
 * Todos os direiros reservados aos autores.
 */
package net.sourceforge.floggy.gui;
```

```
import java.util.Date;

import javax.microedition.lcdui.ChoiceGroup;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.DateField;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.TextField;

import net.sourceforge.floggy.Comparator;
import net.sourceforge.floggy.FloggyException;
import net.sourceforge.floggy.HospitalMIDlet;
import net.sourceforge.floggy.ObjectSet;
import net.sourceforge.floggy.Persistable;
import net.sourceforge.floggy.PersistableManager;
import net.sourceforge.floggy.model.Internacao;
import net.sourceforge.floggy.model.Leito;
import net.sourceforge.floggy.model.Medico;
import net.sourceforge.floggy.model.Paciente;

public class InternacaoForm extends Form implements CommandListener {

    Internacao internacao;

    ObjectSet pacientes;

    ObjectSet medicos;

    ObjectSet leitos;

    DateField dtEntrada;

    TextField txtMotivo;

    ChoiceGroup cgPacientes;

    ChoiceGroup cgMedicos;

    ChoiceGroup cgLeitos;

    Command cmdOk;

    Command cmdCancelar;

    public InternacaoForm() {
        super("Internação");

        this.internacao = new Internacao();

        iniciaComponentes();

        iniciaPacientes();

        iniciaMedicos();

        iniciaLeitos();
```

```
    }

    private void iniciaComponentes() {
        this.dtEntrada = new DateField("Data Entrada",
DateField.DATE);
        this.dtEntrada.setDate(new Date());
        this.append(this.dtEntrada);

        this.txtMotivo = new TextField("Motivo",
internacao.getMotivo(), 100,
                TextField.ANY);
        this.append(this.txtMotivo);

        this.cgPacientes = new ChoiceGroup("Paciente",
ChoiceGroup.EXCLUSIVE);
        this.append(cgPacientes);

        this.cgMedicos = new ChoiceGroup("Medico",
ChoiceGroup.EXCLUSIVE);
        this.append(cgMedicos);

        this.cgLeitos = new ChoiceGroup("Leitos",
ChoiceGroup.EXCLUSIVE);
        this.append(cgLeitos);

        this.cmdOk = new Command("Ok", Command.OK, 0);
        this.addCommand(this.cmdOk);

        this.cmdCancelar = new Command("Cancelar", Command.CANCEL, 1);
        this.addCommand(this.cmdCancelar);

        this.setCommandListener(this);
    }

    public void iniciaPacientes() {
        PersistableManager pm = PersistableManager.getInstance();
        try {
            pacientes = pm.find(Paciente.class, null, new Comparator()
{

                public int compare(Persistable arg0, Persistable arg1)
{
                    Paciente p1 = (Paciente) arg0;
                    Paciente p2 = (Paciente) arg1;

                    return p1.getNome().compareTo(p2.getNome());
                }

            });

            while (pacientes.hasNext()) {
                Paciente paciente = (Paciente) pacientes.next();
                this.cgPacientes.append(paciente.getNome(), null);
            }

        } catch (FloggyException e) {
            e.printStackTrace();
        }
```

```
    }

    public void iniciaMedicos() {
        PersistableManager pm = PersistableManager.getInstance();
        try {
            medicos = pm.find(Medico.class, null, new Comparator() {

                public int compare(Persistable arg0, Persistable arg1)
{
                    Medico p1 = (Medico) arg0;
                    Medico p2 = (Medico) arg1;

                    return p1.getNome().compareTo(p2.getNome());
                }

            });

            while (medicos.hasNext()) {
                Medico medico = (Medico) medicos.next();
                this.cgMedicos.append(medico.getNome(), null);
            }

        } catch (FloggyException e) {
            e.printStackTrace();
        }

    }

    public void iniciaLeitos() {
        PersistableManager pm = PersistableManager.getInstance();
        try {
            leitos = pm.find(Leito.class, null, new Comparator() {

                public int compare(Persistable arg0, Persistable arg1)
{
                    Leito p1 = (Leito) arg0;
                    Leito p2 = (Leito) arg1;

                    return p1.getNumero().compareTo(p2.getNumero());
                }

            });

            while (leitos.hasNext()) {
                Leito leito = (Leito) leitos.next();
                this.cgLeitos.append(leito.getNumero(), null);
            }

        } catch (FloggyException e) {
            e.printStackTrace();
        }

    }

    public Paciente getPacienteSelecionado() {
        for (int i = 0; i < this.cgPacientes.size(); i++) {
            if (this.cgPacientes.isSelected(i)) {
                try {
                    return (Paciente) this.pacientes.get(i);
```

```
                } catch (Exception e) {
                    //
                }

            }

        }
        return null;
    }

    public Medico getMedicoSelecionado() {
        for (int i = 0; i < this.cgMedicos.size(); i++) {
            if (this.cgMedicos.isSelected(i)) {
                try {
                    return (Medico) this.medicos.get(i);
                } catch (Exception e) {
                    //
                }

            }

        }
        return null;
    }


    public Leito getLeitoSelecionado() {
        for (int i = 0; i < this.cgLeitos.size(); i++) {
            if (this.cgLeitos.isSelected(i)) {
                try {
                    return (Leito) this.leitos.get(i);
                } catch (Exception e) {
                    //
                }

            }

        }
        return null;
    }

    public void commandAction(Command cmd, Displayable dsp) {
        if (cmd == this.cmdOk) {
            PersistableManager pm = PersistableManager.getInstance();

            try {

this.internacao.setDtEntrada(this.dtEntrada.getDate());
                this.internacao.setMotivo(this.txtMotivo.getString());
                this.internacao.setPaciente(getPacienteSelecionado());
                this.internacao.setMedico(getMedicoSelecionado());
                this.internacao.setLeito(getLeitoSelecionado());
                pm.save(this.internacao);
            } catch (FloggyException e) {
                e.printStackTrace();
            }
        }
        HospitalMIDlet.setCurrent(new MainForm());
    }
```

```
}
```

InternacaoList

```
/*
 * Criado em 20/09/2005.
 *
 * Todos os direiros reservados aos autores.
 */
package net.sourceforge.floggy.gui;

import java.util.Date;

import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.List;

import net.sourceforge.floggy.Comparator;
import net.sourceforge.floggy.Filter;
import net.sourceforge.floggy.FloggyException;
import net.sourceforge.floggy.HospitalMIDlet;
import net.sourceforge.floggy.ObjectSet;
import net.sourceforge.floggy.Persistable;
import net.sourceforge.floggy.PersistableManager;
import net.sourceforge.floggy.model.Internacao;

public class InternacaoList extends List implements CommandListener {
    ObjectSet internacoes;

    Command cmdAlta;

    Command cmdVoltar;

    public InternacaoList() {
        super("Internações", List.IMPLICIT);

        iniciaDados();
        iniciaComponentes();
    }

    private void iniciaDados() {
        PersistableManager pm = PersistableManager.getInstance();

        try {
            this.deleteAll();

            internacoes = pm.find(Internacao.class, new Filter() {

                public boolean matches(Persistable arg0) {
                    return ((Internacao) arg0).getDtSaida() == null;
                }

            }, new Comparator() {
                public int compare(Persistable arg0, Persistable arg1)
{
```

```
                    String s1 = ((Internacao)
arg0).getPaciente().getNome();
                    String s2 = ((Internacao)
arg1).getPaciente().getNome();

                    return s1.compareTo(s2);

                }
            });

            while (internacoes.hasNext()) {
                Internacao element = (Internacao) internacoes.next();
                this.append(element.getPaciente().getNome() + " - "
                        + element.getLeito().getNumero(), null);
            }

        } catch (FloggyException e) {
            e.printStackTrace();
        }
    }

    private void iniciaComponentes() {
        this.cmdVoltar = new Command("Voltar", Command.BACK, 0);
        this.addCommand(this.cmdVoltar);

        this.cmdAlta = new Command("Alta", Command.ITEM, 1);
        this.addCommand(this.cmdAlta);

        this.setCommandListener(this);
    }

    public void commandAction(Command cmd, Displayable dsp) {
        if (cmd == this.cmdVoltar) {
            MainForm mainForm = new MainForm();
            HospitalMIDlet.setCurrent(mainForm);
        } else if (cmd == this.cmdAlta) {
            if (this.getSelectedIndex() != -1) {
                try {
                    Internacao internacao = (Internacao)
internacoes.get(this
                            .getSelectedIndex());
                    internacao.setDtSaida(new Date());
                    PersistableManager.getInstance().save(internacao);
                    this.iniciaDados();
                } catch (FloggyException e) {
                    e.printStackTrace();
                }
            }
        }

    }
}
```

LeitoForm

```
package net.sourceforge.floggy.gui;
```

```
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.TextField;

import net.sourceforge.floggy.FloggyException;
import net.sourceforge.floggy.HospitalMIDlet;
import net.sourceforge.floggy.PersistableManager;
import net.sourceforge.floggy.model.Leito;

public class LeitoForm extends Form implements CommandListener {

    Leito leito;

    TextField txtNumeroLeito;

    TextField txtAndarLeito;

    Command cmdOk;

    Command cmdCancelar;

    public LeitoForm(Leito leito) {
        super("Leito");

        this.leito = leito;

        iniciaComponentes();
    }

    private void iniciaComponentes() {
        this.txtNumeroLeito = new TextField("Número do Leito",
leito.getNumero(), 30, TextField.ANY);
        this.append(this.txtNumeroLeito);

        this.txtAndarLeito  = new TextField("Andar do Leito",
String.valueOf(leito.getAndar()), 15, TextField.NUMERIC);
        this.append(this.txtAndarLeito);

        this.cmdOk = new Command("Ok", Command.OK, 0);
        this.addCommand(this.cmdOk);

        this.cmdCancelar = new Command("Cancelar", Command.CANCEL, 1);
        this.addCommand(this.cmdCancelar);

        this.setCommandListener(this);
    }

    public void commandAction(Command cmd, Displayable dsp) {
        if(cmd == this.cmdOk) {
            PersistableManager pm = PersistableManager.getInstance();

            try {
                leito.setNumero(this.txtNumeroLeito.getString());

leito.setAndar(Integer.parseInt(this.txtAndarLeito.getString()));
                pm.save(leito);
```

```
            } catch (FloggyException e) {
                e.printStackTrace();
            }
        }

        HospitalMIDlet.setCurrent(new LeitoList());
    }
}
```

LeitoList

```
package net.sourceforge.floggy.gui;

import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.List;

import net.sourceforge.floggy.Comparator;
import net.sourceforge.floggy.FloggyException;
import net.sourceforge.floggy.HospitalMIDlet;
import net.sourceforge.floggy.ObjectSet;
import net.sourceforge.floggy.Persistable;
import net.sourceforge.floggy.PersistableManager;
import net.sourceforge.floggy.model.Leito;

public class LeitoList extends List implements CommandListener {

    ObjectSet leitos;

    Command cmdInserir;

    Command cmdAlterar;

    Command cmdExcluir;

    Command cmdVoltar;

    public LeitoList() {
        super("Lista de Leitos", List.IMPLICIT);

        iniciaDados();
        iniciaComponentes();
    }

    private void iniciaDados() {
        PersistableManager pm = PersistableManager.getInstance();

        try {
            this.deleteAll();

            leitos = pm.find(Leito.class, null, new Comparator() {

                public int compare(Persistable arg0, Persistable arg1)
{
                    Leito l1 = (Leito) arg0;
                    Leito l2 = (Leito) arg1;
```

```
                    return l1.getNumero().compareTo(l2.getNumero());
                }

            });

            while (leitos.hasNext()) {
                Leito element = (Leito) leitos.next();
                this.append(element.getNumero() + " - Andar " +
element.getAndar(), null);
            }

        } catch (FloggyException e) {
            e.printStackTrace();
        }
    }

    private void iniciaComponentes() {
        this.cmdVoltar = new Command("Voltar", Command.BACK, 0);
        this.addCommand(this.cmdVoltar);

        this.cmdInserir = new Command("Inserir", Command.ITEM, 1);
        this.addCommand(this.cmdInserir);

        this.cmdAlterar = new Command("Alterar", Command.ITEM, 2);
        this.addCommand(this.cmdAlterar);

        this.cmdExcluir = new Command("Excluir", Command.ITEM, 3);
        this.addCommand(this.cmdExcluir);

        this.setCommandListener(this);
    }

    public void commandAction(Command cmd, Displayable dsp) {
        if (cmd == this.cmdVoltar) {
            MainForm mainForm = new MainForm();
            HospitalMIDlet.setCurrent(mainForm);
        } else if (cmd == this.cmdInserir) {
            Leito leito = new Leito();
            HospitalMIDlet.setCurrent(new LeitoForm(leito));
        } else if (cmd == this.cmdAlterar) {
            if (this.getSelectedIndex() != -1) {
                Leito leito = null;

                try {
                    leito = (Leito)
leitos.get(this.getSelectedIndex());
                    HospitalMIDlet.setCurrent(new LeitoForm(leito));
                } catch (FloggyException e) {
                    e.printStackTrace();
                }
            }
        } else if (cmd == this.cmdExcluir) {
            if (this.getSelectedIndex() != -1) {

                try {
                    Leito leito = (Leito)
leitos.get(this.getSelectedIndex());
                    PersistableManager.getInstance().delete(leito);
```

```
                    this.iniciaDados();
                } catch (FloggyException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

MainForm

```
package net.sourceforge.floggy.gui;

import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.List;

import net.sourceforge.floggy.HospitalMIDlet;
import net.sourceforge.floggy.PersistableManager;
import net.sourceforge.floggy.PersistableMetadata;
import net.sourceforge.floggy.model.Formacao;

public class MainForm extends List implements CommandListener {

    Command cmdSair;

    public MainForm() {
        super("Hospital", List.IMPLICIT);

        iniciaComponentes();
    }

    protected void cadastrosIniciais() {
        PersistableManager pm = PersistableManager.getInstance();

        // Formacaoes
        try {
            PersistableMetadata metadata =
pm.getMetadata(Formacao.class);

            if (metadata.getObjectCount() == 0) {
                pm.save(new Formacao("Pediatra"));
                pm.save(new Formacao("Ginecologista"));
                pm.save(new Formacao("Clinico Geral"));
                pm.save(new Formacao("Dermatologista"));
                pm.save(new Formacao("Ortopedista"));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    protected void iniciaComponentes() {
        this.append("Cadastrar Formação", null);
        this.append("Cadastrar Leito", null);
        this.append("Cadastrar Médico", null);
```

```
        this.append("Cadastrar Paciente", null);
        this.append("-----", null);
        this.append("Incluir Internação", null);
        this.append("Autorizar Alta", null);
        this.append("-----", null);
        this.append("Relatório de Leitos Livres", null);
        this.append("Relatório de Internações por Médicos", null);

        this.cmdSair = new Command("Sair", Command.ITEM, 3);
        this.addCommand(this.cmdSair);

        this.setCommandListener(this);
    }

    public void commandAction(Command cmd, Displayable dsp) {
        if (cmd == this.cmdSair) {
            HospitalMIDlet.sair();
        } else if (cmd == List.SELECT_COMMAND) {
            switch (this.getSelectedIndex()) {
            case 0:
                HospitalMIDlet.setCurrent(new FormacaoList());
                break;
            case 1:
                HospitalMIDlet.setCurrent(new LeitoList());
                break;
            case 2:
                HospitalMIDlet.setCurrent(new MedicoList());
                break;
            case 3:
                 HospitalMIDlet.setCurrent(new PacienteList());
                 break;
            case 5:
                HospitalMIDlet.setCurrent(new InternacaoForm());
                break;
            case 6:
                HospitalMIDlet.setCurrent(new InternacaoList());
                break;
            case 8:
                HospitalMIDlet.setCurrent(new
RelatorioLeitosLivres());
                break;
            }
        }
    }
}
```

MedicoForm

```
package net.sourceforge.floggy.gui;

import java.util.Vector;

import javax.microedition.lcdui.ChoiceGroup;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.DateField;
```

```
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.TextField;

import net.sourceforge.floggy.Comparator;
import net.sourceforge.floggy.FloggyException;
import net.sourceforge.floggy.HospitalMIDlet;
import net.sourceforge.floggy.ObjectSet;
import net.sourceforge.floggy.Persistable;
import net.sourceforge.floggy.PersistableManager;
import net.sourceforge.floggy.model.Formacao;
import net.sourceforge.floggy.model.Medico;

public class MedicoForm extends Form implements CommandListener {

    Medico medico;

    ObjectSet formacoes;

    TextField txtNome;

    TextField txtCPF;

    DateField dtNascimento;

    TextField txtCRM;

    ChoiceGroup cgFormacao;

    Command cmdOk;

    Command cmdCancelar;

    public MedicoForm(Medico medico) {
        super("Médico");

        this.medico = medico;

        iniciaComponentes();

        iniciaFormacao();
    }

    private void iniciaComponentes() {
        this.txtNome = new TextField("Nome", medico.getNome(), 30,
                TextField.ANY);
        this.append(this.txtNome);

        this.txtCPF = new TextField("CPF", medico.getCpf(), 30,
TextField.ANY);
        this.append(this.txtCPF);

        this.dtNascimento = new DateField("Data Nascimento",
DateField.DATE);
        this.dtNascimento.setDate(medico.getDataNascimento());
        this.append(this.dtNascimento);

        this.txtCRM = new TextField("CRM", medico.getCrm(), 30,
TextField.ANY);
```

```
        this.append(this.txtCRM);

        this.cgFormacao = new ChoiceGroup("Formação",
ChoiceGroup.MULTIPLE);
        this.append(cgFormacao);

        this.cmdOk = new Command("Ok", Command.OK, 0);
        this.addCommand(this.cmdOk);

        this.cmdCancelar = new Command("Cancelar", Command.CANCEL, 1);
        this.addCommand(this.cmdCancelar);

        this.setCommandListener(this);
    }

    public void commandAction(Command cmd, Displayable dsp) {
        if (cmd == this.cmdOk) {
            PersistableManager pm = PersistableManager.getInstance();

            try {
                this.medico.setNome(this.txtNome.getString());
                this.medico.setCpf(this.txtCPF.getString());
                this.medico.setCrm(this.txtCRM.getString());

this.medico.setDataNascimento(this.dtNascimento.getDate());

                if(this.medico.getFormacoes() != null) {
                    this.medico.getFormacoes().removeAllElements();
                }
                else {
                    this.medico.setFormacoes(new Vector());
                }
                for (int i = 0; i < this.cgFormacao.size(); i++) {
                    if (this.cgFormacao.isSelected(i) ) {

this.medico.getFormacoes().addElement(this.formacoes.get(i));
                    }
                }

                pm.save(this.medico);

            } catch (FloggyException e) {
                e.printStackTrace();
            }
        }
        HospitalMIDlet.setCurrent(new MedicoList());
    }

    public void iniciaFormacao() {

        PersistableManager pm = PersistableManager.getInstance();
        try {
            formacoes = pm.find(Formacao.class, null, new Comparator()
{

                public int compare(Persistable arg0, Persistable arg1)
{
                    Formacao f1 = (Formacao) arg0;
                    Formacao f2 = (Formacao) arg1;
```

```
                    return
f1.getFormacao().compareTo(f2.getFormacao());
                }

            });

            while (formacoes.hasNext()) {
                Formacao formacao = (Formacao) formacoes.next();
                int index =
this.cgFormacao.append(formacao.getFormacao(), null);
                if ((medico.getFormacoes() != null) &&
(this.medico.getFormacoes().contains(formacao))) {
                    this.cgFormacao.setSelectedIndex(index, true);
                }

            }

        } catch (FloggyException e) {
            e.printStackTrace();
        }

    }
}
```

MedicoList

```
/*
 * Criado em 18/09/2005.
 *
 * Todos os direiros reservados aos autores.
 */
package net.sourceforge.floggy.gui;

import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.List;

import net.sourceforge.floggy.Comparator;
import net.sourceforge.floggy.FloggyException;
import net.sourceforge.floggy.HospitalMIDlet;
import net.sourceforge.floggy.ObjectSet;
import net.sourceforge.floggy.Persistable;
import net.sourceforge.floggy.PersistableManager;
import net.sourceforge.floggy.model.Medico;

public class MedicoList extends List implements CommandListener {
    ObjectSet medicos;

    Command cmdInserir;

    Command cmdAlterar;

    Command cmdExcluir;

    Command cmdVoltar;
```

```
    public MedicoList() {
        super("Lista de Médicos", List.IMPLICIT);

        iniciaDados();
        iniciaComponentes();
    }

    private void iniciaDados() {
        PersistableManager pm = PersistableManager.getInstance();

        try {
            this.deleteAll();

            medicos = pm.find(Medico.class, null, new Comparator() {
                public int compare(Persistable arg0, Persistable arg1)
{
                    String s1 = arg0 == null ? "" : ((Medico)
arg0).getNome();
                    String s2 = arg1 == null ? "" : ((Medico)
arg1).getNome();

                    return s1.compareTo(s2);
                }

            });

            while (medicos.hasNext()) {
                Medico element = (Medico) medicos.next();
                this.append(element.getNome() + " - CRM " +
element.getCrm(),
                        null);
            }

        } catch (FloggyException e) {
            e.printStackTrace();
        }
    }

    private void iniciaComponentes() {
        this.cmdVoltar = new Command("Voltar", Command.BACK, 0);
        this.addCommand(this.cmdVoltar);

        this.cmdInserir = new Command("Inserir", Command.ITEM, 1);
        this.addCommand(this.cmdInserir);

        this.cmdAlterar = new Command("Alterar", Command.ITEM, 2);
        this.addCommand(this.cmdAlterar);

        this.cmdExcluir = new Command("Excluir", Command.ITEM, 3);
        this.addCommand(this.cmdExcluir);

        this.setCommandListener(this);
    }

    public void commandAction(Command cmd, Displayable dsp) {
        if (cmd == this.cmdVoltar) {
            MainForm mainForm = new MainForm();
            HospitalMIDlet.setCurrent(mainForm);
        } else if (cmd == this.cmdInserir) {
```

```
            Medico medico = new Medico();
            HospitalMIDlet.setCurrent(new MedicoForm(medico));
        } else if (cmd == this.cmdAlterar) {
            if (this.getSelectedIndex() != -1) {
                Medico medico = null;

                try {
                    medico = (Medico)
medicos.get(this.getSelectedIndex());
                    HospitalMIDlet.setCurrent(new MedicoForm(medico));
                } catch (FloggyException e) {
                    e.printStackTrace();
                }
            }
        } else if (cmd == this.cmdExcluir) {
            if (this.getSelectedIndex() != -1) {

                try {
                    Medico medico = (Medico) medicos.get(this
                            .getSelectedIndex());
                    PersistableManager.getInstance().delete(medico);
                    this.iniciaDados();
                } catch (FloggyException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

RelatiorioLeitosLivres

```
package net.sourceforge.floggy.gui;

import java.util.Date;
import java.util.Vector;

import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.List;

import net.sourceforge.floggy.Comparator;
import net.sourceforge.floggy.Filter;
import net.sourceforge.floggy.FloggyException;
import net.sourceforge.floggy.HospitalMIDlet;
import net.sourceforge.floggy.ObjectSet;
import net.sourceforge.floggy.Persistable;
import net.sourceforge.floggy.PersistableManager;
import net.sourceforge.floggy.model.Internacao;
import net.sourceforge.floggy.model.Leito;

public class RelatorioLeitosLivres extends List implements
CommandListener {

      Command cmdVoltar;
```

```
	public RelatorioLeitosLivres() {
		super("Leitos Livres", List.IMPLICIT);

		iniciaDados();
		iniciaComponentes();
	}

	private void iniciaDados() {
		PersistableManager pm = PersistableManager.getInstance();

		try {
			this.deleteAll();

			final ObjectSet internacoes =
pm.find(Internacao.class,
					new Filter() {
						public boolean matches(Persistable
arg0) {
							return ((Internacao)
arg0).getDtSaida() == null;
						}
					}, null);

			ObjectSet leitosLivres = pm.find(Leito.class, new
Filter() {

				public boolean matches(Persistable arg0) {
					Leito leito = (Leito) arg0;

					for (int i = 0; i < internacoes.count();
i++) {

						try {
							if (((Internacao)
internacoes.get(i)).getLeito()

	.getNumero().equals(leito.getNumero())) {
								return false;
							}
						} catch (FloggyException e) {
							e.printStackTrace();
						}
					}

					return true;
				}

			}, new Comparator() {

				public int compare(Persistable arg0,
Persistable arg1) {
				  Leito l1 = (Leito) arg0;
				  Leito l2 = (Leito) arg1;

				  return l1.getNumero().compareTo(l2.getNumero());
				  }

			});
```

```
                Leito leito = new Leito();
                while (leitosLivres.hasNext()) {
                      pm.load(leito, leitosLivres.nextId());
                      this.append(leito.getNumero() + " - "
                                  + leito.getAndar(), null);
                }

          } catch (FloggyException e) {
                e.printStackTrace();
          }
    }

    private void iniciaComponentes() {
          this.cmdVoltar = new Command("Voltar", Command.BACK, 0);
          this.addCommand(this.cmdVoltar);

          this.setCommandListener(this);
    }

    public void commandAction(Command cmd, Displayable dsp) {
          if (cmd == this.cmdVoltar) {
                MainForm mainForm = new MainForm();
                HospitalMIDlet.setCurrent(mainForm);
          }
    }
}
```

PacienteForm

```
package net.sourceforge.floggy.gui;

import javax.microedition.lcdui.ChoiceGroup;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.DateField;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.TextField;

import net.sourceforge.floggy.FloggyException;
import net.sourceforge.floggy.HospitalMIDlet;
import net.sourceforge.floggy.ObjectSet;
import net.sourceforge.floggy.PersistableManager;
import net.sourceforge.floggy.model.Paciente;

public class PacienteForm extends Form implements CommandListener {

    Paciente paciente;

    ObjectSet formacoes;

    TextField txtNome;

    TextField txtCPF;

    DateField dtNascimento;

    TextField txtEndereco;
```

```
    ChoiceGroup cgConvenio;

    TextField txtTelefoneCasa;

    TextField txtTelefoneCelular;

    TextField txtTelefoneTrabalho;

    Command cmdOk;

    Command cmdCancelar;

    public PacienteForm(Paciente paciente) {
        super("Paciente");

        this.paciente = paciente;

        iniciaComponentes();
    }

    private void iniciaComponentes() {
        this.txtNome = new TextField("Nome", paciente.getNome(), 30,
                TextField.ANY);
        this.append(this.txtNome);

        this.txtCPF = new TextField("CPF", paciente.getCpf(), 30,
TextField.ANY);
        this.append(this.txtCPF);

        this.dtNascimento = new DateField("Data Nascimento",
DateField.DATE);
        this.dtNascimento.setDate(paciente.getDataNascimento());
        this.append(this.dtNascimento);

        this.txtEndereco = new TextField("Endereço",
paciente.getEndereco(),
                100, TextField.ANY);
        this.append(txtEndereco);

        this.append("Telefones");
        this.txtTelefoneCasa = new TextField("Casa",
paciente.getTelefoneCasa(), 20, TextField.PHONENUMBER);
        this.append(this.txtTelefoneCasa);
        this.txtTelefoneCelular = new TextField("Celular",
paciente.getTelefoneCelular(), 20, TextField.PHONENUMBER);
        this.append(this.txtTelefoneCelular);
        this.txtTelefoneTrabalho = new TextField("Trabalho",
paciente.getTelefoneTrabalho(), 20, TextField.PHONENUMBER);
        this.append(this.txtTelefoneTrabalho);

        this.cgConvenio = new ChoiceGroup("Tipo:",
ChoiceGroup.EXCLUSIVE);
        this.cgConvenio.append("Particular", null);
        this.cgConvenio.append("Convênio", null);
        this.cgConvenio.setSelectedIndex(0, paciente.isParticular());
        this.cgConvenio.setSelectedIndex(1, !paciente.isParticular());
        this.append(cgConvenio);
```

```
        this.cmdOk = new Command("Ok", Command.OK, 0);
        this.addCommand(this.cmdOk);

        this.cmdCancelar = new Command("Cancelar", Command.CANCEL, 1);
        this.addCommand(this.cmdCancelar);

        this.setCommandListener(this);
    }

    public void commandAction(Command cmd, Displayable dsp) {
        if (cmd == this.cmdOk) {
            PersistableManager pm = PersistableManager.getInstance();

            try {
                this.paciente.setNome(this.txtNome.getString());
                this.paciente.setCpf(this.txtCPF.getString());

this.paciente.setDataNascimento(this.dtNascimento.getDate());

this.paciente.setParticular(this.cgConvenio.isSelected(0));

this.paciente.setTelefoneCasa(this.txtTelefoneCasa.getString());

this.paciente.setTelefoneCelular(this.txtTelefoneCelular.getString());

this.paciente.setTelefoneTrabalho(this.txtTelefoneTrabalho.getString()
);
                pm.save(this.paciente);

            } catch (FloggyException e) {
                e.printStackTrace();
            }
        }
        HospitalMIDlet.setCurrent(new PacienteList());
    }
}
```

PacienteList

```
/*
 * Criado em 18/09/2005.
 *
 * Todos os direiros reservados aos autores.
 */
package net.sourceforge.floggy.gui;

import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.List;

import net.sourceforge.floggy.Comparator;
import net.sourceforge.floggy.FloggyException;
import net.sourceforge.floggy.HospitalMIDlet;
import net.sourceforge.floggy.ObjectSet;
import net.sourceforge.floggy.Persistable;
import net.sourceforge.floggy.PersistableManager;
```

```
import net.sourceforge.floggy.model.Paciente;

public class PacienteList extends List implements CommandListener {
    ObjectSet pacientes;

    Command cmdInserir;

    Command cmdAlterar;

    Command cmdExcluir;

    Command cmdVoltar;

    public PacienteList() {
        super("Lista de Pacientes", List.IMPLICIT);

        iniciaDados();
        iniciaComponentes();

    }

    private void iniciaDados() {
        PersistableManager pm = PersistableManager.getInstance();

        try {
            this.deleteAll();

            pacientes = pm.find(Paciente.class, null, new Comparator()
{
                public int compare(Persistable arg0, Persistable arg1)
{
                    String s1 = arg0 == null ? "" : ((Paciente)
arg0).getNome();
                    String s2 = arg1 == null ? "" : ((Paciente)
arg1).getNome();

                    return s1.compareTo(s2);
                }
            });

            while (pacientes.hasNext()) {
                Paciente element = (Paciente) pacientes.next();
                String tipo;
                if (element.isParticular()) {
                    tipo = "Particular";
                } else {
                    tipo = "Convênio";
                }
                this.append(element.getNome() + " - " + tipo, null);
            }

        } catch (FloggyException e) {
            e.printStackTrace();
        }
    }

    private void iniciaComponentes() {
        this.cmdVoltar = new Command("Voltar", Command.BACK, 0);
        this.addCommand(this.cmdVoltar);
```

```
        this.cmdInserir = new Command("Inserir", Command.ITEM, 1);
        this.addCommand(this.cmdInserir);

        this.cmdAlterar = new Command("Alterar", Command.ITEM, 2);
        this.addCommand(this.cmdAlterar);

        this.cmdExcluir = new Command("Excluir", Command.ITEM, 3);
        this.addCommand(this.cmdExcluir);

        this.setCommandListener(this);
    }

    public void commandAction(Command cmd, Displayable dsp) {
        if (cmd == this.cmdVoltar) {
            MainForm mainForm = new MainForm();
            HospitalMIDlet.setCurrent(mainForm);
        } else if (cmd == this.cmdInserir) {
            Paciente paciente = new Paciente();
            HospitalMIDlet.setCurrent(new PacienteForm(paciente));
        } else if (cmd == this.cmdAlterar) {
            if (this.getSelectedIndex() != -1) {
                Paciente paciente = null;

                try {
                    paciente = (Paciente) pacientes
                            .get(this.getSelectedIndex());
                    HospitalMIDlet.setCurrent(new
PacienteForm(paciente));
                } catch (FloggyException e) {
                    e.printStackTrace();
                }
            }
        } else if (cmd == this.cmdExcluir) {
            if (this.getSelectedIndex() != -1) {

                try {
                    Paciente paciente = (Paciente) pacientes.get(this
                            .getSelectedIndex());
                    PersistableManager.getInstance().delete(paciente);
                    this.iniciaDados();
                } catch (FloggyException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

build.xml

```
<?xml version="1.0"?>

<project name="floggy-compiler-j2me-midp" default="main" basedir=".">
  <taskdef name="floggy-compiler"
classname="net.sourceforge.floggy.CompilerTask"
```

```
classpath="${user.home}/.maven/repository/floggy/jars/floggy-compiler-
j2me-midp2-1.0.jar"/>

  <target name="main">
    <floggy-compiler
classpath="${user.home}/.maven/repository/j2me/jars/midpapi-
2.0.jar${path.separator}${user.home}/.maven/repository/j2me/jars/cldca
pi-
1.1.jar${path.separator}${user.home}/.maven/repository/floggy/jars/flo
ggy-framework-SNAPSHOT.jar" inputfile="${basedir}/bin"
outputfile="${basedir}/bin"/>
  </target>
</project>
```